



UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

TECNOLOGÍA ESPECÍFICA DE
INGENIERÍA DE SOFTWARE

TRABAJO FIN DE GRADO

Diseño y desarrollo de videojuego Android para
visitas interactivas en edificios culturales
utilizando realidad aumentada

Alejandro Velasco Rodríguez

Julio, 2019



UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA
Tecnología específica de Ingeniería de Software

Intensificación: Ingeniería del software

TRABAJO FIN DE GRADO

**Diseño y desarrollo de videojuego Android para
visitas interactivas en edificios culturales
utilizando realidad aumentada**

Autor: Alejandro Velasco Rodríguez

Director: David Vallejo Fernández

Julio, 2019

Alejandro Velasco Rodríguez

Villafranca de los Caballeros – España

E-mail: alejandro.velasco@alu.uclm.es

Teléfono: +34 638 54 83 12

Web site: <http://webpub.esi.uclm.es>

© 2019 Alejandro Velasco Rodríguez

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Se permite la copia, distribución y/o modificación de este documento bajo los términos de la Licencia de Documentación Libre GNU, versión 1.3 o cualquier versión posterior publicada por la *Free Software Foundation*; sin secciones invariantes. Una copia de esta licencia esta incluida en el apéndice titulado «GNU Free Documentation License».

Muchos de los nombres usados por las compañías para diferenciar sus productos y servicios son reclamados como marcas registradas. Allí donde estos nombres aparezcan en este documento, y cuando el autor haya sido informado de esas marcas registradas, los nombres estarán escritos en mayúsculas o como nombres propios.

TRIBUNAL:

Presidente:

Vocal:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE

VOCAL

SECRETARIO

Fdo.:

Fdo.:

Fdo.:

Resumen

En los últimos años el uso de la realidad aumentada no ha parado de crecer, las posibilidades que ofrece hoy en día en un mercado en el que cualquier ámbito puede resultar ser objeto de aplicación de la realidad aumentada son inmensas. En concreto los espacios culturales e históricos se aprovechan cada vez más de estas tecnologías para ofrecer métodos alternativos a la hora de mostrar e interactuar con el contenido educativo que se requiera.

Además de ello, la posibilidad de incluir la realidad aumentada en un videojuego ofrece nuevas vías para potenciar la *gamificación* y proyectar la industria del videojuego en el sector educativo y cultural. Ofrece también un incentivo para ser parte de la historia que se pretende contar.

Lo que plantea este trabajo es el diseño y desarrollo de un videojuego *Android* para visitas interactivas en edificios culturales utilizando realidad aumentada, de esta manera se pretende que el jugador pueda seguir una ruta a través de un edificio cultural en la que se hallen ciertos eventos puntuales o desafíos en forma de minijuegos que el jugador tendrá que superar. Estos minijuegos tendrían un contexto y una temática con los elementos culturales del edificio para completar el apartado educativo.

El desarrollo de este videojuego se llevará a cabo utilizando herramientas de desarrollo de videojuegos actuales como *Unity3D* además de complementarlo con las tecnologías de realidad aumentada *Vuforia Engine*. Estas herramientas ofrecen todos los componentes necesarios para llevar a cabo lo que se propone, además de hacer uso de herramientas complementarias de edición y modelado 3D. El resultado de este trabajo se espera que pueda tener una gestión de minijuegos y que se lleve a cabo la implementación de un minijuego que utilice realidad aumentada.

Abstract

In recent years the use of augmented reality has not stopped growing, the possibilities offered today and the market in which any field can be subject to application of augmented reality are immense. Specifically, cultural and historical spaces are increasingly taking advantage of these technologies to offer alternative methods when displaying and interacting with the educational content required.

In addition to this, the possibility of including reality will increase in a videogame, as well as in the ways to enhance the *gamification* and project the videogame industry in the educational and cultural sector. It also offers an incentive to be part of the story.

What this work does is the design and development of a video game for *Android*. Find certain specific events or activities in the form of mini-games that the player will have to overcome. These mini-games will have a context and a theme with the cultural elements of the building to complete the educational section.

The development of this videogame will be handled using game development tools as *Unity3D*. In addition, they are complemented by augmented reality technologies with *Vuforia Engine*. These tools offer all the necessary components to carry out the proposed purpose, in addition other tools will be necessary as 3D editing and modeling tools. The result of this work is expected to have a mini-game management and the implementation of a mini-game that must use augmented reality.

Agradecimientos

Aún escribiendo estas palabras no me encuentro concienciado de qué es lo que está sucediendo realmente, está ocurriendo el cierre de otra etapa más, pero una que ha estado marcada por ser la más intensa, increíble y divertida de mi vida hasta ahora, sin duda no podría haber llegado hasta este momento si no hubiera pasado por todos los momentos que me han obligado a llegar hasta aquí, a veces pensamos que es cosa del destino, pero realmente todas las opciones son fruto del destino, sólo hay que coger un hilo y tirar de él. Me siento realmente feliz de estar aquí ahora y tengo la necesidad de agradecer a estas personas que me han acompañado todo este tiempo...

...A aquel que me gusta llamarle de forma diferente cada día, me abrió las puertas a una nueva forma de ver las cosas, reímos, nos contamos las paranoias acerca del universo que consideramos potencialmente viables. Por nuestra realidad paralela llena de bromas y fantasía.

...A aquel que me enseñó a cómo jugar a las *Magic* mientras compartíamos todas nuestras aficiones en videojuegos, películas, series, animes. No puedo estar más cómodo cuando estoy junto a ti.

...A aquella que durante tantos años y después de todo lo que ha pasado sigue enviando fotos de perritos para alegrarme el día, porque la vida sin duda no sería la misma sin ti.

...A aquel que me acompaña en todos los viajes en los que podemos descubrir cómo es la vida fuera de la burbuja que llamamos rutina.

...A aquel profesor que me enseñó cómo es la realidad y la importancia de conocer a las personas que nos rodean.

...A todos los compañeros y compañeras con los que me reunía en las noches calurosas de junio en la biblioteca para compartir fuerzas y ánimos y conseguir sacar todo esto adelante.

...A mí mismo por haberme soportado en los momentos más complicados y aún así ser capaz de seguir cantando delante del espejo mientras me río de mí mismo. Espero que nunca se vaya el niño que me observa desde dentro y me anima.

Alejandro Velasco, 27 Julio de 2019

A mi padre Jose Antonio por animarme e impulsarme en esta vida llena de retos, a mi madre Felisa por apoyarme cuando más lo necesitaba, a mi hermano Jose Angel por ayudarme siempre y a todos mis amigos y amigas por acompañarme.

Índice general

Índice general	13
1. Introducción	1
1.1. Contexto del proyecto	1
1.2. Motivación	2
1.3. Descripción del proyecto	2
1.4. Estructura del documento	4
2. Objetivos	5
2.1. Objetivo general	5
2.2. Objetivos específicos	5
3. Estado del Arte	7
3.1. Desarrollo en dispositivos móviles	7
3.1.1. Introducción y evolución	7
3.1.2. Desarrollo en Android	9
3.1.3. Herramientas Transversales	11
3.2. La Industria del videojuego	13
3.2.1. Introducción y contexto socio-económico	13
3.2.2. Videojuegos en el ámbito cultural	14
3.2.3. Estrategias de monetización y fidelización de usuarios	15
3.3. Realidad aumentada	17
3.3.1. Conceptos Fundamentales	17
3.3.2. Aplicaciones de la realidad aumentada	18
3.3.3. Realidad aumentada en entornos culturales	20
3.3.4. Herramientas de desarrollo	20
4. Método de Trabajo	23
4.1. Elección de la metodología de desarrollo	23

4.1.1.	Motivación de una metodología ágil	23
4.1.2.	Aplicación de eXtreme Programming	24
4.2.	Herramientas de trabajo	27
4.2.1.	Características Hardware	27
4.2.2.	Características Software	27
5.	Arquitectura	29
5.1.	Descripción del juego	29
5.2.	Visión general de la arquitectura	31
5.3.	Menú Principal	33
5.3.1.	Componentes de la interfaz	35
5.3.2.	Selección de Minijuegos	38
5.3.3.	Sistema de Almacenamiento	40
5.3.4.	Controlador de Audio	42
5.3.5.	Gestor de Escenas	43
5.4.	Minijuego: Defiende la Torre del Homenaje	44
5.4.1.	Game Manager	45
5.4.2.	<i>ImageTarget</i> de Minijuego	48
5.4.3.	Sistema de Puntuación	50
5.4.4.	Sistema de Pool de Objetos	51
5.4.5.	Sistema de salud	52
5.4.6.	Enemigos	54
5.4.7.	Jugador	56
5.5.	Patrones de diseño utilizados	60
5.5.1.	Modelo-Vista-Controlador	60
5.5.2.	Observador	60
5.5.3.	Componente	61
6.	Resultados	63
6.1.	Proceso de desarrollo	63
6.1.1.	Iteración 1: Prototipo	63
6.1.2.	Iteración 2 : Enemigos	65
6.1.3.	Iteración 3: Refactorización y mejoras	67
6.1.4.	Iteración 4: Apartado visual	68
6.1.5.	Iteración 5: Menú principal y optimización	69
6.1.6.	Iteración 6: Torre con cañón, selección de minijuegos y mantenimiento	71

6.2. Implementación del videojuego	73
6.3. Coste del proyecto	74
7. Conclusiones	75
7.1. Objetivos alcanzados	75
7.2. Dificultades encontradas	78
7.3. Trabajo futuro	78
7.4. Conclusiones finales	78
A. Game Design Document	83
B. Prototipo de Santiago de la Torre RA	91
Referencias	95

Capítulo 1

Introducción

LA cultura y la historia son una de las bases de nuestra sociedad, el patrimonio de la misma se debería conservar mediante la educación para evitar que caiga en el olvido, puesto que sin cultura ni educación no existiría la sociedad. Traer de vuelta una historia en ruinas mediante la educación es lo que motiva este proyecto.

1.1 Contexto del proyecto

Este proyecto se ve enmarcado en el propósito de incentivar el proceso de musealización del castillo de *Santiago de la Torre*, este se trata de una fortaleza medieval que se encuentra anexionada al municipio de San Clemente¹. El castillo fue construido en el siglo XIII en tiempos de la reconquista y desde entonces ha servido a diferentes propietarios incluyéndose entre ellos la *Orden de Santiago* en el siglo XIV además de dar origen a la aldea de *Santiago de la Torre* y la ermita que lo circunda.

Desde el año 1955 se realizan celebraciones con temáticas religiosas como romerías desde *El Provencio* y aunque el castillo ha permanecido en buen estado durante todo ese tiempo, fue restaurada por primera vez en el año 1973, año a partir del cual no se celebraría ningún acto religioso y por lo que, aceleraría su peor periodo histórico.

A partir del siglo XX el castillo sufrió un abandono y la desaparición de su aldea, por lo que se reconvierte en una casa de labor, aunque no quiere decir que mejorara sus condiciones puesto que el castillo en su interior se vería saqueado y expuesto al vandalismo acelerando así su proceso de ruina.

En febrero de 2018 se realiza un acuerdo entre el ayuntamiento de *El Provencio* y *San Clemente* para realizar un análisis histórico y arqueológico del complejo para valorar la posibilidad de una restauración, después de hacer los trámites necesarios para que los propietarios del castillo pudieran donar parte de este al ayuntamiento, a finales de agosto del mismo año comienzan los trámites de la restauración².

El ayuntamiento de *El Provencio* tiene como objetivo que una vez completados los procesos de restauración, puedan acondicionar el interior del castillo para convertirlo en un museo

¹[https://es.wikipedia.org/wiki/San_Clemente_\(Cuenca\)](https://es.wikipedia.org/wiki/San_Clemente_(Cuenca))

²<http://www.elprovencio.com/objetivo-santiago-de-la-torre/>

en el que se pueda ver los elementos del castillo, su historia, anécdotas así como exponer el por qué se trata de un ejemplo de arquitectura defensiva y la sociedad feudal y del Antiguo Régimen en el área de La Mancha.

Aquí entra en el contexto la empresa con la que se colabora en este proyecto *Akura Games*³, una empresa dedicada al desarrollo de videojuegos entre otros proyectos de ingeniería que utilizan toda clase de tecnologías incluyéndose entre ellas la realidad aumentada. Así es como desde el ayuntamiento de *El Provencio* contrató a esta empresa para realizar un proyecto de videojuego del castillo haciendo uso de realidad aumentada.

Debido a que el proceso de restauración y musealización puede tomar bastante tiempo, la empresa *Akura Games* ofertó estas prácticas en las que se establece este proyecto TFG como una oferta del programa *FORTE*⁴ de la *Escuela Superior de Informática* en Ciudad Real para realizar un prototipo de lo que podría ser el videojuego que se desea realizar.

1.2 Motivación

Hoy en día la realidad aumentada se hace cada vez más presente en una gran variedad de ámbitos, como es el caso de los espacios culturales y educativos, la constante necesidad de reinventar las maneras de enseñar e interactuar con la historia o el arte permite que casi cualquier entorno pueda ser objeto de aplicación de realidad aumentada.

Por lo que el uso de la realidad aumentada en este proyecto permitiría visualizar mejor las partes y elementos de la cultura e historia que ofrece el castillo de *Santiago de la Torre*, además de motivar la interacción de los turistas que deseen conocer este patrimonio cultural de *La Mancha* feudal.

No sería la primera vez que se pretende realizar un proyecto de utilizando tecnologías similares y que mostrara parte de la historia de *Castilla La-Mancha*, como el ayuntamiento de *Poblete* que ha inaugurado una ruta que recrea la batalla de *Alarcos* en realidad aumentada [1].

En el contexto de la empresa surge la motivación de hacerse un hueco en el área de la realidad aumentada y además en el ámbito de la educación, de esta manera se pretende diseñar un proyecto que permita su reutilización en otros espacios culturales con características similares.

1.3 Descripción del proyecto

El objetivo principal de este proyecto es el diseño y desarrollo de un videojuego *Android* para visitas interactivas en edificios culturales, el videojuego en cuestión debe actuar a modo de guía o mapa en el que el jugador avanzará a través de un entorno real en el que se

³<http://akura.es>

⁴<http://webpub.esi.uclm.es/spa/paginas/professionalizate-forte>

encontrarán eventos puntuales como acertijos, puzzles, o pequeños minijuegos utilizando la realidad aumentada para que la interactividad con estos atraiga la atención y permita hacer uso de gamificación para que también tenga un marco educativo en el que se enseñen los aspectos culturales e históricos del lugar en el que se encuentra el jugador.

Se pretende que el videojuego tenga los componentes necesarios para que sea escalable y mantenible en el tiempo debido al proceso de musealización, por lo que se espera que sea un prototipo que permita visualizar cómo el jugador va a interaccionar en el entorno real con los elementos, y cómo se verán incluidos los eventos y minijuegos dentro del entorno.

Para ello el videojuego debe contar con todas las mecánicas propias de un videojuego *Android* como un sistema de interacción, sonido, modelos tridimensionales, una interfaz que cumpla con ciertos estándares de usabilidad, etc.

El desarrollo del proyecto se realizará con el motor de juegos multiplataforma *Unity3D*, el cual ofrece todas las herramientas necesarias para desarrollar lo que se espera de este proyecto, incluyendo tecnologías de realidad aumentada, en este caso *Vuforia Engine*⁵, un sistema de exportación para dispositivos móviles y además siendo una herramienta gratuita⁶ hace que su accesibilidad sea aún mayor. Además *Unity* cuenta con un *marketplace*, *AssetStore* en el que hay todo tipo de recursos para videojuegos, desde modelos, sistemas completos, audio, efectos... Esto reduce considerablemente los tiempos de desarrollo al facilitar ciertos componentes que de otra manera tendrían un coste en el desarrollo más elevado.

El lenguaje de programación que se utilizará es *C#*, el cual está integrado en un *IDE* llamado *Visual Studio*, este se complementa al desarrollo en *Unity* ofreciendo herramientas de depuración de código y visualización de la arquitectura.

Para la integración de los diferentes modelos tridimensionales y su edición será necesario usar un programa externo de edición 3D como *Blender 3D*, ya que además se trata de un programa gratuito. También serán necesarios otros recursos como de audio, imágenes, u otros. Para ello se hará uso de páginas web que ofrezcan estos recursos de forma gratuita como *opengameart.org*, *mixamo.com*, *sketchfab.com*, etc.

Uno de los aspectos más fundamentales de este proyecto es llevar a cabo un desarrollo basado en patrones de diseño para permitir que el resultado final sea una solución modular con una arquitectura que sea escalable y mantenible, además de aplicar ciertas normas de calidad de desarrollo móvil.

El resultado de este proyecto se considera que debe ser un ejecutable para el sistema operativo *Android* que sea un videojuego que contenga una arquitectura que permita la inclusión de minijuegos basados en un entorno real y que utilice realidad aumentada como principal vía de jugabilidad en los minijuegos, además estos deben tener una temática y contexto

⁵<https://developer.vuforia.com>

⁶La versión *Unity Personal* resulta gratuita para desarrolladores

relacionada con el edificio cultural que entra dentro del contexto del proyecto.

1.4 Estructura del documento

En esta sección se listarán los contenidos que pueden encontrarse a lo largo de este documento, añadiendo una breve descripción de cada capítulo.

Capítulo 2: Objetivos

En este capítulo se detalla a modo de listado cuáles son las metas definidas en el proyecto exponiendo qué se pretende hacer de forma sistemática.

Capítulo 3: Estado del Arte

Aquí se realizará una revisión de los temas relacionados en este proyecto. En concreto se hace una revisión sobre el actual desarrollo en los dispositivos móviles, pasando por su evolución hasta las herramientas de desarrollo actuales, también se comenta el estado actual de la industria de videojuegos para exponer después la realidad aumentada describiendo sus conceptos fundamentales y sus aplicaciones haciendo especial hincapié en aquellas relacionadas con entornos culturales

Capítulo 4: Método de Trabajo

Con el fin de contextualizar la forma en la que se ha afrontado el desarrollo se describe aquí la metodología que se ha seguido, además de incluir las condiciones hardware y software que caracterizan este proyecto.

Capítulo 5: Arquitectura

Este capítulo está dedicado en detalle al diseño y particularidades en cuanto a la implementación de la solución que se ha llevado a cabo. Ofrece una panorámica general de cómo se estructura el proyecto explicando los componentes que se han desarrollado para dar solución a la propuesta del proyecto.

Capítulo 6: Resultados

Aquí se muestra y describe el resultado final obtenido, indicando las diferentes etapas de desarrollo que ha tenido el proyecto y detallando las tareas llevadas a cabo para alcanzar el objetivo del proyecto.

Capítulo 7: Conclusiones

El último capítulo sirve a modo de reflexión sobre la utilidad del resultado obtenido, planteando posibles escenarios futuros en los que se podría realizar la implementación completa del videojuego, además de un pequeño apartado para describir conclusiones personales.

Capítulo 2

Objetivos

UNA vez expuestos el enfoque y las ideas del proyecto en la introducción, este capítulo concreta qué es lo que se va a llevar a cabo, determinando cuál es el objetivo general para posteriormente detallar el conjunto de objetivos específicos que se derivan. Así es como se establece el camino a seguir para poder obtener un resultado final.

2.1 Objetivo general

Con la finalidad de incentivar el proceso de musealización del castillo de *Santiago de la Torre* el objetivo de este proyecto es el de *diseñar y desarrollar un videojuego Android basado en realidad aumentada para dinamizar visitas a edificios culturales*.

Se espera obtener un videojuego en el que el jugador pueda recorrer un edificio cultural explorando ciertos puntos interesantes del mismo y que en estos, se encuentren pequeños desafíos en forma de minijuegos, puzzles o acertijos con los que se pretende mostrar de forma interactiva parte de la cultura o historia del lugar. De esta manera se quiere potenciar la *gamificación* con el uso de realidad aumentada para que la información aumentada sea mucho más visible y divertida a la vez que educativa.

El desarrollo se hace en colaboración con la empresa *Akura Games* la cual está centrada en el desarrollo de videojuegos por lo que se adapta perfectamente a la realización de este tipo de proyecto, sin embargo sería la primera vez que se realiza un videojuego haciendo uso de realidad aumentada, por lo que será necesario realizar una investigación acerca de qué tipo de herramientas existen y cuál encaja mejor en el contexto de desarrollo.

2.2 Objetivos específicos

A continuación se expondrá un listado de los objetivos específicos que se esperan alcanzar en este proyecto, a grandes rasgos se puede resumir en el desarrollo de una solución modular de los sistemas que se necesiten en el videojuego para dar cabida a los requisitos de usuario que se indican, componer una arquitectura escalable haciendo uso de patrones de diseño que permitan a su vez una gestión de los minijuegos que se pueden encontrar en el videojuego y aplicación de normas de calidad para desarrollo en dispositivos móviles *Android*.

- **Diseño y desarrollo de una solución modular adaptada a los requisitos del usuario**

Con este objetivo se espera identificar los requisitos de usuario para poder realizar un correcto seguimiento de cómo debe ser la solución que se desarrolle, además se espera que esta solución deba ser modular, evitar que haya un sólo sistema que se encargue de controlar todas las características relacionadas con este proyecto y diseñar varios sistemas que formen a su vez un conjunto, de esta manera si en un futuro los requisitos pudieran cambiar, la modificación de la solución no sería compleja.

- **Desarrollo de una arquitectura escalable, basada en patrones de diseño, para la inclusión de nuevos minijuegos**

Este objetivo trata de realizar una arquitectura en el videojuego que permita incluir nuevos minijuegos de forma sencilla y adaptable, esto será así cumpliendo con un desarrollo que esté basado en los principales patrones de diseño que se proponen en desarrollo de videojuegos como el patrón observador o *object pool*, entre otros.

- **Desarrollo de un sistema para la gestión de minijuegos basada en marcadores**

Para cumplir este objetivo se espera que la gestión de los tipos de minijuegos y el contenido de los mismos esté basado en los marcadores que se detecten haciendo uso de la realidad aumentada, esto es, diseñar marcadores de tal manera que se permita generar un número determinado de los mismos y se puedan asignar a diferentes minijuegos evitando pasar por la identificación de cada uno de ellos para mostrar contenido.

- **Aplicación de normas de calidad para desarrollo en dispositivos móviles Android**

Este proyecto se enmarca en la intensificación de *Ingeniería del Software* por lo que se espera con este objetivo es que a la hora de desarrollar la aplicación, se tengan en cuenta normas de calidad para desarrollo móvil como pueden ser las de usabilidad, eficiencia, flexibilidad, etc.

Capítulo 3

Estado del Arte

EN este capítulo se introducirá el contexto actual del desarrollo en dispositivos móviles, su relación en la industria de videojuegos y los aspectos más relevantes e interesantes de la realidad aumentada. Se abordarán los siguientes temas:

- **Desarrollo en dispositivos móviles** Aquí se introducirá los primeros dispositivos móviles que ofrecían videojuegos entre sus características, cómo estos evolucionaron hasta convertirse en lo que es hoy en día un teléfono móvil inteligente con acceso a internet y altas prestaciones técnicas. También se hablará de cómo se estructura el sistema operativo Android, qué ventajas ofrece desarrollar en él y algunas herramientas que permiten desplegar aplicaciones para ese sistema.
- **Industria del videojuego** En esta sección se espera introducir cómo se encuentra la industria del videojuego tanto a nivel global como nacional, explicar cuál es su contexto socio-económico, indagar acerca de la controvertida relación entre la cultura y los videojuegos exponiendo algunos de los ejemplos más claros de videojuegos culturales y por último señalar cuáles son las principales estrategias de monetización y fidelización de usuarios.
- **Realidad aumentada** Por último, se introducirán los conceptos fundamentales de la realidad aumentada para entender qué es y cómo funciona incluyendo algunos de los ámbitos en los que se aplica haciendo especial hincapié en los entornos culturales y exponer las herramientas de desarrollo actuales con sus características principales.

3.1 Desarrollo en dispositivos móviles

3.1.1 Introducción y evolución

El dispositivo móvil conocido actualmente se trata de un ordenador de bolsillo que permite una conexión a internet constante, esto favorece su evolución tanto en características como en funcionalidades. Puede ser de muchos tipos, así como smartphones, relojes inteligentes, videoconsolas portátiles, etc.

Hoy en día el desarrollo en dispositivos móviles se encuentra mayormente alojado en los *marketplaces* de aplicaciones como *Google Play*, *App Store*, etc. Es un mercado que no para

de crecer, ya que tienen la posibilidad de ofrecer multitud de servicios centralizados en un sólo dispositivo. Para hacerse una idea del volumen de aplicaciones, en mayo de 2019 Google Play cuenta con más de 3 millones de aplicaciones.¹

En 2007 llegó el 3G a los dispositivos móviles, fue con el primer iPhone de Apple con el que vio la luz la App Store, en 2008 contaba con 500 aplicaciones de terceros para iPhone y iPod Touch, y no fue hasta 2009 cuando se lanzó Android Market con 2300 aplicaciones. Esto supuso una revolución en el desarrollo de aplicaciones ya que se eliminaban los gastos de sistemas de distribución y pagos a operadoras telefónicas, facilitando la accesibilidad y abriendo las puertas a todas las desarrolladoras [3].

Además con el continuo desarrollo en el apartado de hardware, permitían la evolución de las aplicaciones para que utilizaran el gps, la cámara, giroscopio, acelerómetro,... Gracias a estas características surgen nuevas ideas donde emprender y donde nacen nuevas empresas como resultado de una aplicación exitosa, por ejemplo Uber, Instagram, Snapchat.

Se puede concluir en que la dirección en la que evoluciona el desarrollo en dispositivos móviles es al de las aplicaciones móviles, gracias a la versatilidad que ofrecen en desarrollo de servicios, y las últimas tendencias indican que se profundizará el desarrollo en los siguientes aspectos:

■ Internet de las cosas

Cada día se dispone de más dispositivos que recopilan información y la envían al teléfono móvil, el uso de este tipo de dispositivos puede variar desde el uso doméstico como ajustar la temperatura de una habitación hasta obtener el número de pulsaciones con una pulsera inteligente.

En la figura (3.1) se muestra que en 2018 se registraron más de 7 billones de dispositivos IoT conectados, y además indica que existirá un crecimiento del 17 % para 2025

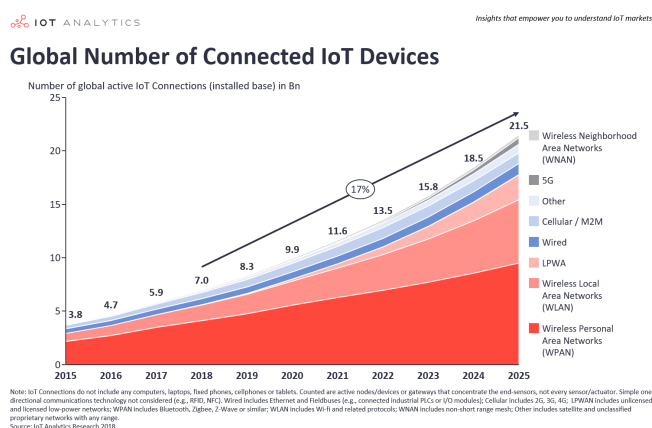


Figura 3.1: Numero de aparatos IoT conectados

¹Numero de Aplicaciones en Google Play

■ Realidad aumentada y virtual en aplicaciones móviles

Gracias a los últimos avances en cuanto a potencia en los terminales móviles, el desarrollo de realidad aumentada y virtual ha revolucionado la industria de las aplicaciones móviles permitiendo que se puedan incorporar algunos mercados como el de turismo, la sanidad, inmobiliarias, etc.

■ Apps On-Demand

Estas aplicaciones ofrecen productos y servicios bajo demanda, como pedir comida a domicilio, o un taxi. Se espera que el uso de este tipo de aplicaciones crezca, además teniendo en cuenta que cada vez los dispositivos móviles estarán más conectados entre sí.

El aspecto más característico de este tipo de aplicaciones es el apartado económico ya que en algunos casos, se trata de personas las que ofrecen un tipo de servicio y pueden obtener un beneficio económico gracias a la automatización de otros procesos como manufacturación o de distribución

Algunos ejemplos de este tipo de aplicaciones pueden ser el caso de *Uber*, *AirBnB*, *Glovo*, etc.

■ Inteligencia Artificial

Hoy en día son muy conocidos los nombres de *Alexa*, *Google Home*, *Siri* o *Cortana*, estos agentes virtuales son precisamente muy útiles en cuanto a sus usos con aplicaciones móviles como por ejemplo los comandos de voz para leer mensajes o buscar información en internet.

Se estima que el gasto mundial de sistemas de inteligencia artificial y cognitiva alcancen los 77.6 mil millones de dólares en 2022 ²

En conclusión el desarrollo de dispositivos móviles ofrece muchos retos y posibilidades para emprender en proyectos, y cada día crece más.

3.1.2 Desarrollo en Android

Android se trata de uno de los sistemas operativos más populares actualmente, nace de la mano de *Android Inc*, una pequeña empresa de Palo Alto con la idea de integrar sistemas operativos Linux en dispositivos táctiles. Fue adquirida por Google en 2005 para impulsar este desarrollo que finalmente se acabó presentando en 2007.

El desarrollo en este sistema se encuentra en pleno auge, existen más de 2 billones de dispositivos *Android* activos en el mundo, además según Forbes³ la cuota de mercado móvil de Android se encontraba entre el 80 y el 90 % comparado con iOS dejando muy poco

²Worldwide Spending on Cognitive and Artificial Intelligence Systems Forecast to Reach \$77.6 Billion in 2022

³Google Reveals iOS Market Share Is 65% to 230% Bigger Than We Thought

margen a otros sistemas operativos. En la figura 3.2 se aprecia la evolución de las unidades smartphones vendidas, donde se aprecia cómo Android consigue dominar el mercado.

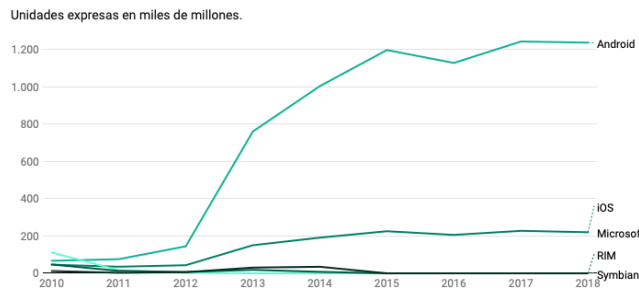


Figura 3.2: Evolución de las ventas de smartphones según el sistema operativo (2010-2018) [4]

El hecho de que existan tantos dispositivos utilizando Android, no quiere decir que todos tengan características similares, existe un gran número de dispositivos Android que no se encuentran en las últimas actualizaciones y otro gran porcentaje que utilizan *Open Source Android* de los que Google no conoce nada.

Pero esto abre muchas puertas a la hora de desarrollar, ya que hay un mercado detrás y para explicar cómo se realiza el desarrollo en Android se analizarán los siguientes apartados:

- **Estructura del sistema operativo** La estructura del sistema operativo está organizado en varias capas, en la más baja se encuentra el núcleo basado en *Linux* que proporciona los servicios genéricos para el manejo de seguridad, procesos, memoria, E/S, etc.

La capa intermedia está formada por las librerías escritas en C y C++ que contienen todo lo necesario para la visualización, reproducción de audio y vídeo, modelado de gráficos 3D, bases de datos, etc. En el mismo nivel se encuentra el Runtime de Android que es un conjunto de librerías escritas en JAVA.

Y en las capas superiores se encuentra el framework de Android y el entorno de aplicaciones con los elementos como botones, iconos, cajas de textos, gestores de notificaciones, navegador web, lector de correo electrónico, etc.

- **Lenguaje de programación**

Las aplicaciones de Android están programadas en Java, el cual se encuentra entre los lenguajes de programación más utilizados globalmente. Hay otras alternativas como C# que se utiliza en caso de videojuegos.

- **Entorno de Desarrollo**

Por defecto el entorno de desarrollo integrado oficial es *Android Studio*, el cual ya tiene integrado todas las librerías necesarias, funcionalidades para desarrollar en cualquier versión de *Android*, plantillas de aplicaciones, etc.

Una de las ventajas con las que cuenta Android es que para desarrollar aplicaciones podemos tener múltiples entornos de programación con la condición de que incluyan Android SDK ⁴, el cual es un conjunto de herramientas que nos ofrecen entre otras cosas un simulador de Android, un depurador de código, documentación, etc.

■ **Publicación de aplicaciones**

Google Play se trata del *marketplace* en el que se alojan las aplicaciones de Android, y es el sitio ideal donde se prefiere publicar las aplicaciones desarrolladas, no contiene medidas tan estrictas a la hora de verificar la calidad del contenido como App Store, esa es una de las razones por las que existen aplicaciones con una reputación muy negativa, aunque estas se suelen retirar a tiempo gracias a comentarios o reseñas.

Además de superar estas verificaciones, el precio por publicar una aplicación en Google Play actualmente es una tarifa de pago único de 25 \$.

En definitiva, tenemos un sistema operativo completo y personalizable, aunque no siempre se puede obtener la última versión para Android, ya que también esto depende de los fabricantes de móviles, los cuales desarrollan su propia versión de Android para aprovechar de la mejor forma posible las capacidades de sus terminales.

3.1.3 Herramientas Transversales

Las librerías Android SDK⁵ son libres y se pueden utilizar casi en cualquier otro framework de desarrollo, esto es vital para agilizar los tiempos entre proyectos, ya que no siempre se trabaja con las mismas tecnologías o en caso de existir cambios en los proyectos, sería complejo migrar por completo todo un proyecto.

En esta sección se van a repasar algunos de los frameworks más utilizados actualmente para desarrollar aplicaciones:

■ **Flutter**

Este se trata de un reciente *framework*⁶ de Google en el que prima la velocidad de desarrollo, uno de los inconvenientes que existe a la hora de desarrollar aplicaciones es el tiempo necesario para incorporar cambios en el proyecto o visualizar nuevas opciones, *Flutter* incluye funciones que permiten hacer cambios en el código y ver su resultado en tiempo real en un emulador.

Flutter no utiliza archivos XML en las plantillas de aplicaciones por lo que toda la interfaz se realiza mediante código, esto al principio puede resultar en un proceso de aprendizaje inicial superior, pero permite agilizar el desarrollo a la larga.

⁴Android Studio Development Kit

⁵Software Development Kit

⁶Un *framework* es un entorno de trabajo con una estructura conceptual y tecnológica definida, con artefactos o módulos concretos de Software. Fuente: *Wikipedia*

■ Unity

Unity es un caso particular ya que no es exclusivamente un framework de desarrollo de aplicaciones móviles pero ofrece esa opción, además será el framework en el que se desarrolla este proyecto por lo que es interesante incluirlo.

Se trata de un framework para desarrollo de videojuegos que se encuentra disponible para las plataformas Windows, OS X y Linux, además ofrece soporte de compilación para diferentes plataformas como PS4, PC, Android, iOS, etc. Dispone de tres versiones:

- Personal Es gratuito hasta superar 100.000 \$ al año en ingresos y cuenta con las funcionalidades básicas del motor.
- Pro Tiene un precio de 125 \$ por mes e incluye acceso prioritario a especialistas de Unity y servicio al cliente, además de soportar funciones de sincronización de proyectos y servicio en cloud este sería ideal para el caso de una desarrolladora de videojuegos con un equipo de desarrollo grande
- Plus Esta opción es el punto intermedio, cuesta 25 \$ al mes y el público objetivo sería para los aficionados que quieran acelerar su aprendizaje y desarrollo al contar con servicio técnico y algunos descuentos en el *Asset Store*⁷.

Además Unity es uno de los motores de videojuegos más usados hoy en día, según afirma el CEO de Unity *John Riccitiello* más del 60 o 70 % de los videojuegos que utilizan realidad aumentada, mixta o virtual, dependiendo de la plataforma, se desarrollan en Unity [5].

Y es que una de las características más interesantes de Unity es la accesibilidad que tiene a la hora de lanzar una aplicación multiplataforma, ya que incluye una característica⁸ que permite realizar compilaciones dependiendo de la plataforma que se elija, lo que hace es recompilar las librerías internas del motor para que las funciones básicas como la entrada (teclado, toques en la pantalla, mando, etc) o las librerías gráficas se adapten a la plataforma objetivo. Esto es una gran ventaja ya que impulsa y acelera el desarrollo multiplataforma.

Actualmente no es el único ejemplo de framework de videojuegos que permiten generar una aplicación para Android, existen otros casos como *Game Maker*, *Unreal Engine*, etc.

⁷Unity cuenta con una plataforma online donde los usuarios pueden subir contenido así como modelos, música, piezas de código, etc

⁸Platform Dependent Compilation

3.2 La Industria del videojuego

3.2.1 Introducción y contexto socio-económico

El videojuego es un concepto mundialmente conocido que combina tecnología y cultura, encontrándose en un sector estratégico, que asienta las bases de una nueva cultura digital que mezcla todas las artes como fotografía, música, cine, etc. Debido a esto, resulta ser el principal motor de entretenimiento global.

España es uno de los países donde más ha crecido el sector en 2018 facturó 1530 millones de euros [6], está situada por encima del resto de industrias audiovisuales, como el cine o la música. Además se encuentra en el Top 10 de los mercados de videojuegos situándose EEUU en el puesto número uno como se puede ver en la figura 3.3











LOGO	RANK	COUNTRY	REGION	POPULATION	INTERNET POPULATION	TOTAL REVENUES IN US DOLLARS
	1	United States of America	North America	329M	274M	\$36,869M
	2	China	Asia	1,420M	901M	\$36,540M
	3	Japan	Asia	127M	121M	\$18,952M
	4	Republic of Korea	Asia	51M	49M	\$6,194M
	5	Germany	Western Europe	82M	77M	\$6,012M
	6	United Kingdom	Western Europe	67M	65M	\$5,616M
	7	France	Western Europe	65M	59M	\$4,091M
	8	Canada	North America	37M	35M	\$2,772M
	9	Spain	Western Europe	46M	40M	\$2,735M
	10	Italy	Western Europe	59M	42M	\$2,689M

Figura 3.3: Top 10 países listados por ingresos de videojuegos Fuente: Newzoo

Según señala el informe económico de AEVI [7], se deriva que la industria de videojuegos equivale al 0,11 % del PIB español, por cada euro invertido se obtiene un impacto de 3 euros en el conjunto de la economía y por cada empleo, se crean 2,6 en otros sectores. La producción efectiva del sector de los videojuegos en España fue de 1.177 millones de euros. Su valor añadido fue de 503 millones de euros y su empleo directo de 8790 personas

En cuanto al público que tienen los videojuegos cabe decir que es abundante y muy diverso, en España se ha registrado que el 44 % de los españoles juegan a videojuegos, donde el 56 % son hombres y el 44 % mujeres, con edades comprendidas entre los 5 y los 64 años de edad, en total suman unos 16.8 millones de jugadores en 2018. Entre todos los jugadores, componen un 26 % aquellos con edades entre 25 y 34 años, un 19 % entre 35 y 44 y otro 19 % entre 15 y 24 años, por lo que mayoritariamente cabe decir que el público regular del videojuego suele ser joven ya que tienen mayor accesibilidad y familiaridad con las tecnologías [8].

Y entre los dispositivos más utilizados se encuentran las consolas, con un 26 %, PC y Smartphone un 21 %, aunque puede pensarse que en las consolas es donde los jugadores

pasan mayor tiempo a la semana no es así, se dice de 3,9 horas a la semana en consolas y en smartphone+Tablet 5,1 horas a la semana [9].

3.2.2 Videojuegos en el ámbito cultural

Hoy en día existen opiniones enfrentadas respecto si los videojuegos se pueden considerar como un producto cultural o si sólo se trata de un trabajo puramente tecnológico, aunque se puede afirmar que existen videojuegos que son reconocidos por su apartado gráfico, su música o su propia narración. Hay mucha controversia al respecto y de hecho esto ha llevado a debate a diversas figuras políticas, en concreto en España la comisión de cultura del congreso aprobó ya en 2009, hace 10 años, considerar el videojuego como cultura para así facilitar la generación de empleo en este sector y fomentar el desarrollo.⁹

Se puede considerar los videojuegos como una vía de comunicación audiovisual cuyo mensaje es interpretable dependiendo de la experiencia de cada usuario, el hecho de que los videojuegos sean interaccionables los diferencia del resto de medios artísticos, por esto, muchos críticos consideran que el videojuego es un arte y por tanto puede ser un medio para transmitir aspectos culturales.

CD Projekt Red es un estudio de videojuegos polaco, fundado en 2002 y no fue hasta 2007 cuando finalmente lanzaron su primer producto *The Witcher*, este videojuego está inspirado en unas populares novelas polacas de la mano de *Andrzej Sapkowski*, estas novelas relatan las aventuras de un brujo por un mundo medieval inspirado en el *folclore*¹⁰ europeo marcado por los tradicionales cuentos de brujas y criaturas mitológicas.

Tras años de dedicación e inversiones lograron realizar una recreación en forma de Videojuego RPG (Role Playing Game) donde jugador podía encarnar el protagonista de la famosa obra literaria, la jugabilidad, el carisma de los personajes y la historia dieron lugar a que esta desarrolladora hiciera una secuela con la que en conjunto obtuvieron más de 6 millones de copias y además comenzaban a ganarse un nombre en la industria, ya que *The Witcher 2* ganó muchos premios en la GamesCom 2012, además de conseguir que en el segundo juego que lanzaban, una capacidad gráfica importante, tanto que en los años próximos utilizarían el videojuego para realizar *Benchmarking*¹¹.

Con esto no sólo consiguieron alzarse como desarrolladora sino también retroalimentar a la propia obra, la saga de *The Witcher* hoy en día es conocida internacionalmente, tanto por la saga de videojuegos como las obras literarias. Aquí se puede observar la influencia que pueden llegar a generar los videojuegos sobre la cultura si el desarrollo del mismo es adecuado, la figura 3.4 muestra la evolución de ingresos que obtuvieron los diferentes videojuegos de la saga.

⁹ 20minutos.es "Los videojuegos ya están considerados por el Gobierno como un área más de la cultura"

¹⁰ El *folclore* es el cuerpo expresivo de la cultura compartida por un grupo particular de personas

¹¹ Benchmarking es una práctica muy común en el desarrollo de videojuegos, se trata de una serie de pruebas que determinan el nivel de rendimiento Fuente: XatakaMovil

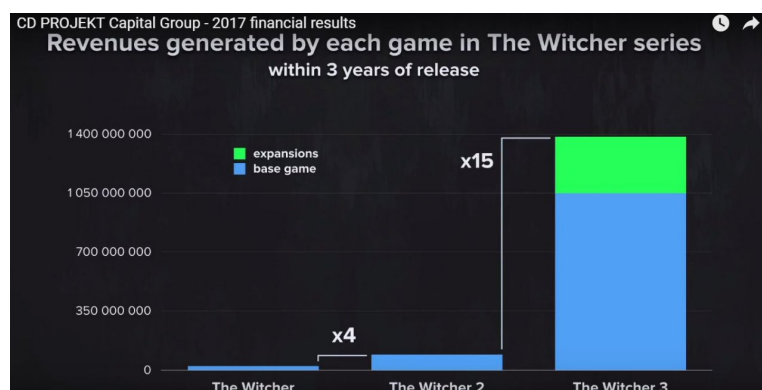


Figura 3.4: Ingresos generados de la saga *The Witcher*. Fuente: *CDProjektRed*

3.2.3 Estrategias de monetización y fidelización de usuarios

El videojuego dentro del contexto empresarial se debe considerar como un producto del que se espera un beneficio, pero como se ha indicado previamente, un videojuego no es un negocio precisamente seguro, depende de muchos factores objetivos y subjetivos para que llegue a ser rentable.

Para ello surge la necesidad de explorar diferentes estrategias de monetización para conseguir que un videojuego pueda aportar beneficios antes y después de su lanzamiento. En esta sección se explicarán desde las más tradicionales como puede ser videojuegos con un precio fijado hasta los últimos modelos basados en publicidad.

Videojuegos con precio fijado.

Estos videojuegos son aquellos que tienen un precio fijado por adelantado, es decir, para acceder a su contenido es necesario pagar, esta es una estrategia muy utilizada pero a la vez arriesgada dependiendo de cómo se quiera distribuir el videojuego. Si es en una plataforma online como puede ser Google Play, Steam, App Store, etc, las condiciones de distribución se basan en la retención de un porcentaje de los beneficios, por ejemplo, en el caso de Google Play tratándose de un 15 % de los beneficios.¹²

Esta es una estrategia ideal en el caso de que el videojuego ya tenga cierto reconocimiento o se tenga asegurado que puede llegar a tener las ventas esperadas para sacar beneficio, normalmente es una estrategia que se sigue mucho en las empresas AAA¹³.

El precio puede cambiar dependiendo del tipo de videojuego, por ejemplo un juego para una plataforma móvil puede costar 0.99 \$. y 2.99\$ mientras que un videojuego para consola o PC puede rondar los 60\$.

Videojuegos con cuotas mensuales

Este método se parece a un método de pago por adelantado, pero en este caso no se trata

¹²El androide Libre: Google Play ganará menos con cada aplicación: el 85% del beneficio irá al desarrollador

¹³Las empresas AAA son aquellas que obtienen la máxima calificación en cumplimiento de obligaciones
Fuente : El economista

de un pago único sino que requiere de una suscripción para poder jugarlo. Normalmente se establece un precio por mes, o año, y una vez que se termina, el progreso del videojuego queda registrado.

El ejemplo más clásico que podemos encontrar aquí se trata de *World of Warcraft*, el cual aún mantiene este sistema de pago aunque desgraciadamente ya no resulta tan rentable como lo fue en su pleno auge.¹⁴

Videojuego Gratuito

El concepto de *Free to Play* es relativamente joven, se utiliza para denominar aquellos videojuegos que son lanzados, normalmente en markets y plataformas online, y se pueden jugar sin requerir ningún tipo de coste, aunque a *priori* pueda parecer algo inútil cada vez se opta más por este método ya que es una manera sencilla de atraer jugadores casuales, que no saben si el videojuego les puede gustar o no, así que se ofrece la posibilidad de acceder a él sin ningún coste, y el propio jugador decide si continuar jugándolo.

Dentro de esta estrategia se pueden incluir diversas formas de monetización:

- **Publicidad In-App** Consiste en mostrar publicidad dentro del videojuego, para ello, es necesario tener una gran base de usuarios, obtener información acerca de ellos para posteriormente vender esta información a las empresas interesadas en publicitarse.

La publicidad puede aparecer de diversas maneras, con un *banner*, una pausa en el propio videojuego a pantalla completa, como un *Pop-up* cuando se realiza alguna acción sobre el videojuego o integrados con el resto de la interfaz.

De esta manera se pueden generar ingresos cada vez que un jugador hace click sobre un anuncio para descargar una aplicación, o por visualizaciones.

- **Freemium** Estos videojuegos ofrecen algunas funcionalidades *Premium* cuyo acceso es sólo mediante pago previo, de este modo, se fideliza a los jugadores que ya se encuentren cómodos en el videojuego y quieran tener ciertas características especiales para facilitar o mejorar su jugabilidad.

Aunque es ideal mantener un equilibrio entre las funcionalidades básicas y las premium ya que es posible que si los jugadores que pagan por estas características tengan una ventaja claramente superior respecto a los otros, es posible que los nuevos jugadores no se sientan cómodos si no tienen la posibilidad de vencer a los premium, de aquí surgió el termino *Pay to Win*

La figura 3.5 muestra el videojuego *Clash Royale* el cual, es un videojuego de cartas que tiene su propia moneda virtual llamadas Gemas, con las que se puede comprar una serie de características como por ejemplo evitar tiempos de espera para obtener nuevas cartas.

¹⁴Los ingresos de World of Warcraft se desploman



Figura 3.5: Ejemplo app *Freemium Clash Royale*. Fuente: Flickr

Complementos y DLCs

Los contenidos descargables de pago son una manera efectiva de añadir un valor extra al videojuego y permitir que evolucione en la dirección que se desee, normalmente se presentan estos contenidos como cambios importantes sobre algunos elementos del videojuego o añadir nuevos, ya sean mapas en un multijugador, armas nuevas, una aventura alternativa en el caso de juegos de aventuras, etc.

3.3 Realidad aumentada

3.3.1 Conceptos Fundamentales

Para introducir el concepto de realidad aumentada, es necesario enmarcarlo en un conjunto de conceptos tecnológicos que sirvan de apoyo para comprender cómo esta se diferencia de otros.

La realidad aumentada es una de las variaciones de la realidad virtual, que compone un conjunto de tecnologías que trata de sumergir al usuario dentro de un entorno sintético en el que no puede ver el entorno real a su alrededor. Por lo que trata la realidad aumentada, trata de integrar sobre un entorno real contenido digital en tiempo real.

Tanto realidad aumentada como virtual tienen un entorno en el que interactúan de formas diferentes, para entender mejor cómo se diferencia este entorno se puede enmarcar dentro de la *continuidad de la realidad virtual*, la cual es una escala que oscila entre lo que es completamente real y lo que es virtual (realidad virtual), en la figura 3.6 se muestra esta escala.

Para entender cómo la realidad aumentada se sitúa en esta escala, Azuma [10] define que un sistema de realidad aumentada debe tener las siguientes características:

- **Combina mundo real y virtual** Para poder combinar ambos, es necesario captar el entorno real en el que se encuentra el sistema y así crear imágenes sintéticas en las que

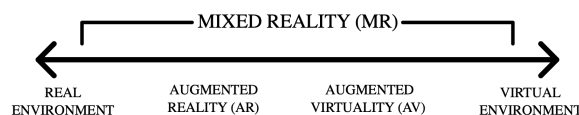


Figura 3.6: Continuidad de la realidad virtual de acuerdo a *Milgram, Takemura, Utsumi y Kishino (1994)* Fuente: Wikipedia

se superpondrá el contenido aumentado.

- **Interactivo en tiempo real** Esto quiere decir que las imágenes captadas por el sistema son en tiempo real y por lo tanto, para poder implementar contenido aumentado sobre las imágenes es necesario que el cálculo se haga de forma interactiva.
- **Alineación 3D** La información del mundo virtual debe ser tridimensional y debe estar correctamente alineada con la imagen del entorno real, esto es, a la hora de calcular el posicionamiento de los objetos que se quieran superponer, estos deben alinearse con el entorno real, hay excepciones en las que en los sistemas de realidad aumentada utilizan capas 2D

Para que un sistema de realidad aumentada funcione necesita realizar ciertos cálculos que permitan posicionar de manera correcta el contenido aumentado, para ello es necesario el cálculo del registro, que consiste en posicionar la cámara relativamente a los objetos de la escena, para ello existen diferentes tecnologías para realizar el registro, como sensores mecánicos, magnéticos, ultrasónicos, inerciales y basados en visión.

El método con el que se realiza este cálculo y posicionamiento se le llama *método de tracking* y existen diferentes tipos aunque el más extendido y accesible es el basado en visión, el cual, a través de lo que percibe en la cámara, recrea la escena virtual y la compone sobre la escena real.

3.3.2 Aplicaciones de la realidad aumentada

Para analizar la motivación que mueve la realidad aumentada a su aplicación es conveniente observar por qué combinar un entorno virtual y uno real puede ser útil. La realidad aumentada potencia la percepción e interacción del usuario con el mundo real lo cual permite que reciba información que no puede captar con sus propios sentidos.

Esta información puede ser utilizada para ayudar al usuario a realizar tareas en su entorno real, por lo que las capacidades del mismo se verían aumentadas, la realidad aumentada se ve como un caso específico de lo que denomina Fred Brooks [11] *Amplificación de Inteligencia*, como el uso de una computadora para facilitar al humano realizar una tarea afirmando que la combinación de máquina y mente puede ser superior a una inteligencia artificial por sí misma.

La posibilidad de tener una interfaz que interaccione con cualquier elemento del entorno

real abre muchas posibilidades en cualquier ámbito, algunos de los ámbitos en los que se aplica la realidad aumentada son los siguientes:

- **Medicina**

En el campo de la medicina ya se encuentra utilizándose en hospitales realidad aumentada para ayudar a facilitar y desarrollar el uso de técnicas como rayos X, tomografías computarizadas y resonancias magnéticas.

Las aplicaciones de realidad aumentada en la medicina consisten en combinar imágenes y modelos 3D que se pueden extraer con esas técnicas para obtener una ayuda, suelen ser utilizadas en el área quirúrgica y la anatomía. También puede ser utilizada para el entrenamiento de doctores. En la actualidad es el médico el que, de una manera continua, hace uso de esta tecnología para poder tomar decisiones y emitir diagnósticos, que junto a sus conocimientos y experiencias, permite salvar vidas.

- **Fabricación**

Otro campo en el que se puede aplicar la realidad aumentada se trata del montaje y mantenimiento de maquinaria compleja, a menudo para la manipulación de ciertas herramientas se hace uso de manuales de texto con figuras, las instrucciones de uso serían más sencillas si las figuras fueran objetos tridimensionales superpuestos al equipamiento real y que mostraran paso por paso la manipulación de ciertas herramientas.

Es interesante analizar cómo beneficia la realidad aumentada a la fabricación ya que los tiempos de producción y costes disminuirían, el usuario cometería menos errores mecánicos, incrementaría su eficiencia y aumentaría la trazabilidad de las operaciones que realiza.

- **Entretenimiento**

La interacción que ofrece la realidad aumentada con el usuario además de facilitar y gestionar mejor las tareas, las convierte en una especie de reto, un objetivo que superar. La industria del entretenimiento se ha encargado de aprovechar esto para ofrecer experiencias interactivas como videojuegos, visitas guiadas, salud, etc.

Y es que la realidad aumentada puede aprovechar cualquier situación o entorno para entretener, ya puede ser recorriendo las calles como en *Pokemon Go*, el cual es una aplicación de realidad aumentada que utiliza la cámara y el sistema gps del teléfono móvil para posicionar al jugador en las calles y simular que encuentra *pokemons* en su camino.

- **Publicidad** Las campañas publicitarias en medios digitales últimamente se han visto impulsadas a reinventarse en cuanto a maneras de llamar la atención sobre sus productos y aquí la realidad aumentada puede resultar ser de bastante utilidad.

Con el uso de realidad aumentada se exploran posibilidades de contar nueva información como utilizando contenido tridimensional que permita que el usuario pueda verse inmerso junto al producto que se pretende vender, así como introducirse en las redes sociales con los filtros de realidad aumentada que permiten poner pendientes o gafas de sol superpuestas a la imagen del usuario.

Se espera que más compañías comiencen a integrar realidad aumentada en la publicidad de sus productos¹⁵

3.3.3 Realidad aumentada en entornos culturales

Desde hace años ya se comenzó a popularizar la realidad aumentada para utilizarla en entornos culturales como pueden ser los museos, exposiciones de artes, bibliotecas, etc. Ya que es una tecnología que permite visualizar contenido superpuesto a un entorno real, se puede llegar a pensar en cómo aplicarlo sobre espacios antiguos, ruinas, castillos... Son muchas posibilidades y aquí se mostrarán algunos ejemplos de cómo se utiliza.

- **Histora** Se trata de un proyecto fin de carrera que permite visualizar algunos edificios emblemáticos de Buenos Aires durante la invasión inglesa y conocer la historia de la universidad en la que se realizó el proyecto¹⁶.
- **Museo de la autonomía de Andalucía** En España ya se han producido varios casos de aplicaciones de realidad aumentada en espacios museísticos como el museo de la autonomía de Andalucía ubicado en Sevilla, el cual ofrece en su sala de exposición una experiencia con realidad aumentada haciendo uso del sistema *Daram SDK*, diseñado por *Arpa Solutions*. En este caso la experiencia se trata de mostrar las diferentes instituciones andaluzas, con modelos virtuales tridimensionales de los edificios donde se ubican.¹⁷

Así la realidad aumentada ofrece la posibilidad de enseñar contenido a través de la red, ofreciendo la interacción con los diferentes objetos que se pueden representar manipulando el marcador correspondiente. El carácter didáctico de este tipo de aplicaciones queda manifiesto y supone una llamada sobre la cultura que se pretende mostrar. Además facilita el acceso al contenido que puede encontrarse en museos a miles de kilómetros o se encuentran desaparecidos en el caso de exposiciones temporales.

3.3.4 Herramientas de desarrollo

Hoy en día existen múltiples herramientas para desarrollar contenido con realidad aumentada, además las empresas suelen optar por desarrollar su propio contenido para así poder tener más autoridad sobre qué y cómo mostrar su producto.

¹⁵La realidad aumentada llegó para quedarse: 6 campañas sorprendentes

¹⁶Vídeo presentación de *Histora* <https://youtu.be/-Bvg7LyKK1w>

¹⁷Realidad aumentada en el museo de la Autonomía de Andalucía



Figura 3.7: Usuario interactuando en el museo de la Autonomía de Andalucía Fuente: *Arpa-Solutions S.L Plataforma Software DARAM* de Realidad Aumentada

Aquí se nombrarán algunas de las herramientas de desarrollo que se utilizan actualmente:

- **Google ARCore**

Se encuentra entre los entornos de desarrollo más populares y es que este aparte de pertenecer a Google, es gratuita y provee kits de desarrollo para casi cualquier plataforma en la que se genera contenido en realidad aumentada, *Android*, *iOS*, *Unity*, *Unreal*. Provee las características esenciales de realidad aumentada como rastreo de movimiento, reconocimiento de entorno y estimación de luz.

Es una herramienta que aún se encuentra en desarrollo y los dispositivos compatibles aún son limitados debido a los requisitos para poder ejecutarlo, sin embargo Google afirma que se encuentran desarrollando para 100 de los 2.000 millones de dispositivos Android en el mundo.¹⁸

Además actualmente se encuentran desarrollando un sistema llamado *Visual Positioning Service (VPS)* que permite reconocer con patrones visuales y haciendo uso de técnicas de aprendizaje automático, ubicar al usuario con una precisión de centímetros en interiores, esto resulta muy útil en el caso de querer usar la geolocalización del usuario en un interior ya que no es posible localizar al usuario via satélite.

- **AR ToolKit**

Es una biblioteca de funciones de realidad aumentada que permite desarrollar aplicaciones de RA usando técnicas ópticas para la detección de marcadores en tiempo real.

¹⁸XatakaMovil: La realidad aumentada llega a Android de la mano de Google ARCore

Algunas de las características más interesantes son:

- *Tracking* de una sola cámara La versión básica soporta de forma nativa el *tracking* de una sola cámara.
- Marcadores negros cuadrados Emplea métodos de *tracking* de superficies planas en las que los marcadores pueden ser personalizables con patrones.
- Rápido y multiplataforma Se encuentra disponible en varios sistemas operativos (*Linux, Mac, Windows,...* y es compatible con los dispositivos portátiles y smartphones actuales.
- Licencia libre Se permite utilizar, modificar y distribuir programas realizados con AR ToolKit bajo la licencia GPL v2.

■ Vuforia

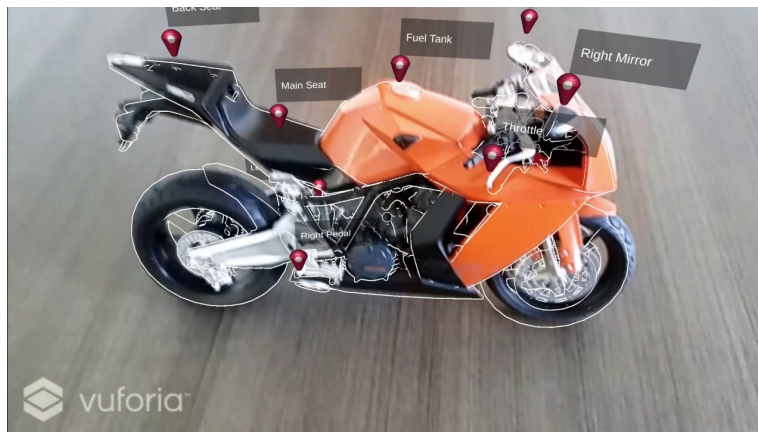


Figura 3.8: Ejemplo del uso de Marcadores de Objetos reales Fuente: *Vuforia*

Es ampliamente conocido como el líder de la industria en cuanto a tecnología de visión por computador debido a sus robustas capacidades de *tracking* en base a características y *deep learning* y la amplitud de plataformas en la que es soportado. Es necesaria una cuenta de desarrollador en su página web y ahí es donde se suben los marcadores que se deseen utilizar y se registran con una licencia de desarrollador.

Una de las características más interesantes de *Vuforia* es que permite el uso de marcadores no planos, es decir, se pueden utilizar marcadores con forma de cubo, cilindro, etc. También cuenta con una aplicación que permite hacer reconocimiento de objetos reales para utilizarlos como marcadores en tiempo real. Como se ve en la figura 3.8

Método de Trabajo

EN este capítulo se va a tratar la manera en la que se ha llevado a cabo el trabajo, en qué contexto y con qué medios se ha realizado. Exponiendo la metodología de trabajo que se ha seguido, justificando su elección desde la naturaleza del proyecto y cómo se ha adaptado. Por otro lado, se indicarán las herramientas de software y hardware que han sido necesarias para llevar a cabo el desarrollo.

4.1 Elección de la metodología de desarrollo

Es importante escoger una metodología adecuada dependiendo del tipo de desarrollo, en este caso, se ha optado por seguir las pautas básicas que se establecen en *eXtreme Programming* [12], esta es una metodología de desarrollo ágil orientada a la mejora de la calidad del software y capacidad de respuesta ante cambios en los requisitos. Este es un proyecto que se encuentra enmarcado en el contexto del programa *FORTE*¹ que ofrece la Escuela Superior de Informática en el que una empresa puede elegir un alumno para realizar prácticas con una especialidad determinada y el alumno tiene la posibilidad de realizar también el Trabajo Fin de Grado (TFG) en el contexto de las prácticas. Debido a esto, es interesante abarcar el proyecto con esta metodología ya que todo el desarrollo será supervisado tanto por la empresa de prácticas como por el tutor del proyecto, por lo que la comunicación entre estas entidades será fundamental.

Aunque para su aplicación será necesaria una adaptación ya que el proyecto aunque se encuentre en un marco empresarial, se abordará por una persona, sin embargo, existirá una fuerte interacción entre el principal responsable del proyecto en la empresa que se colabora y el tutor del proyecto.

4.1.1 Motivación de una metodología ágil

Tomando como referencia la comparativa entre metodologías ágiles y tradicionales que se recoge en [13], se va a analizar por qué escoger una metodología ágil, para ello es necesario conocer el tipo de proyecto que se está abordando, el cual se trata un videojuego que pretende ser un prototipo de algo mayor, por lo que la escalabilidad es un factor importan-

¹<http://webpub.esi.uclm.es/paginas/professionalizate-forte>

te, los requisitos no están muy bien definidos ni acotados por lo que debe estar preparado para cambios durante el proyecto, además no se desea una planificación global de la arquitectura, en cambio, es preferible avanzar con pequeñas entregas de software que funcionen para así proyectar un desarrollo creciente que permita evolucionar el producto de una manera más orgánica y mediante una comunicación continua que permita descubrir errores y planteamientos diferentes.

Una vez conocido el tipo de proyecto se concluye que este proyecto sigue algunos de los principios del manifiesto ágil² y encaja mejor con los procesos que siguen las metodologías ágiles.

4.1.2 Aplicación de eXtreme Programming

Antes de comentar cómo se procede a aplicar *eXtreme Programming* es interesante reflejar por qué se ha elegido esta metodología ágil, para ello se tomará como referencia la definición de Iacovelli [14] de un framework para la clasificación de las principales metodologías ágiles, el cual utiliza de base un diagrama que permite obtener diferentes vistas sobre los métodos ágiles que se muestra en la figura 4.1.

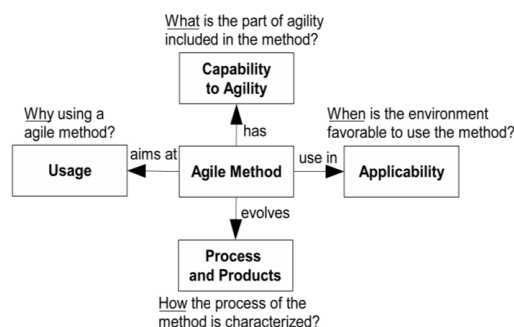


Figura 4.1: Cuatro vistas de los métodos ágiles según Iacovelli

En la tabla que se refleja en [15] se muestran las cuatro vistas de 4.1 seguidas de una serie de atributos que reciben un valor binario, esta tabla permite realizar una evaluación de las diferentes metodologías ágiles, de este modo es mucho más sencillo saber qué metodología ágil se adapta mejor a un proyecto y contexto de trabajo. De esta manera, se ha escogido *eXtreme Programming* en base a los atributos mostrados en la tabla.

Tomando como referencia los atributos, se espera lo siguiente:

- **Capacidad de uso**
 - Orientación a la productividad
 - Fecha de entrega no determinante

² Doce principios del manifiesto ágil <https://agilemanifesto.org/iso/es/principles.html>

■ Capacidad de agilidad

- Políticas de refactorización y testing con una intergración fácil de cambios
- Riesgo y complejidad no elevados

■ Vista de procesos y productos

- Definición de requisitos y Modelado
- Codificación con test y ejecutables como artefacto de salida

A continuación se comentarán los principios que caracterizan la metodología particularizando y justificando cómo se han aplicado o se han procurado aplicar. En la figura 4.2 muestra el flujo de trabajo destacando los procesos y conceptos que son involucrados en XP³.

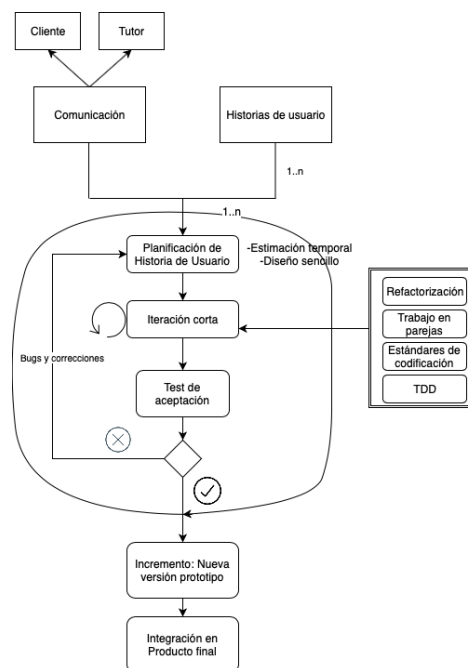


Figura 4.2: Flujo de trabajo adaptado de *eXtreme Programming*

- **Comunicación** Este desarrollo se realiza de manera individual en colaboración con la empresa *Akura Games S.L.*, la cual participará en rol de cliente y el tutor de este proyecto quien también participa como cliente y como equipo de soporte proporcionando requisitos nuevos y recomendaciones en el desarrollo. Por lo que la comunicación entre estas entidades se considera un punto importante, ya que se pretende que el resultado se pueda llegar a implementar en un futuro por la empresa cliente con el soporte técnico del tutor.
- **Roles Simplificados** Debido a la naturaleza del proyecto y como se ha reflejado anteriormente, existen varios roles que recaen sobre la misma persona. Por lo tanto, el rol

³ Acrónimo para eXtreme Programming

de programador, encargado de pruebas y del seguimiento lo cumple una misma persona. Así pues, el rol de cliente y de *Big Boss* lo asumirá la empresa que ofrece este proyecto (*Akura Games*). Y por último, el rol de consultor lo realiza tanto el equipo de la empresa como el tutor de este proyecto, ya que en ambos se pueden reforzar los conocimientos y dudas técnicas en caso de ser requeridos.

- **Iteraciones cortas** Se consideran unas iteraciones o *sprints*⁴ cortas de 1 o 2 semanas en las que al final, se fijará una reunión para enseñar los resultados que se han obtenido, de esta manera se favorece el *seguimiento* y así permite detectar errores e incidencias y debatir acerca de las correcciones o soluciones que sean necesarias.
- **Entregas** Cuando finaliza una iteración no es necesario entregar un ejecutable que sea operativo ni cerrado, sino que es preferible revisar cómo se han realizado las historias de usuario, de esta manera, al cabo de una serie de iteraciones pueden dar lugar al resultado del desarrollo de un módulo o una característica específica que permite crecer el sistema de manera incremental.
- **Refactorización** Este es un principio que promueve el incremento de calidad a lo largo del ciclo de vida del proyecto, mientras se pretende mantener un diseño simple y evitar la complejidad y el desorden innecesario. Resulta fundamental para mejorar los errores que se cometen a lo largo del periodo de aprendizaje
- **Proceso de integración continua** Según *eXtreme Programming* el desarrollo que se realice debe ser integrado en el repositorio del código siempre que sea posible. En este caso, este principio se ha relajado, ya que se trata de un desarrollo individual y no es crítica la paralelización del trabajo, sino que la idea de integración continua se considerará en una fase más avanzada del desarrollo en la que a partir del resultado de este proyecto se pueda facilitar la agregación de funcionalidades.
- **Orientación a testing** Este es uno de los puntos más fuertes del *eXtreme Programming*, ya que la metodología de desarrollo está orientada hacia el *Test Driven Development (TDD)* [16], se trata de una metodología de desarrollo en la que primero se escriben las pruebas y después se refactoriza. En este caso, las pruebas se tratan de *Playtest* los cuales son pruebas que se realizan sobre el videojuego cuando se encuentra en ejecución para conocer en qué falla.
- **Seguimiento de estándares de programación** Desde el principio se ha tratado de seguir los estándares que resultan adecuados para el diseño de sistemas interactivos según se indican en [17] como son el de *usabilidad* (ISO/IEC-9241, 1998), *accesibilidad* y el estándar clásico de evaluación de calidad de un sistema software (ISO 9126) para permitir que el sistema interactivo que se pretende desarrollar permita cumplir de

⁴Nombre que recibe las iteraciones o unidades de tiempo para entregar un incremento o módulo de software en las metodologías ágiles Fuente: <http://www.cyta.com.ar/ta0502/v5n2a1.htm>

manera correcta con la definición que según Nielsen Norman Group da sobre *Experiencia de usuario*⁵

4.2 Herramientas de trabajo

A continuación se expone qué características software y hardware se han requerido durante todas las fases del proyecto.

4.2.1 Características Hardware

Modelo de ordenador	Lenovo V130-15IKB
Procesador	Intel Core i5-7200U
Tarjeta Gráfica	Intel HD Graphics 620
Memoria Ram	4GB DDR4-2400 MHz
Almacenamiento	256GB HDD
Sistema Operativo	Windows 10 Home 64 bits
Webcam	Creative Live! Cam Sync HD
Teléfono Android	Huawei Ascend G7

Cuadro 4.1: Lista de características hardware utilizadas

Al tratarse de un videojuego, el proyecto ha requerido el uso de un ordenador portátil que ofreciera unas capacidades gráficas que al menos se ajustaran a los requisitos de las tecnologías de realidad aumentada, acompañado de una webcam aparte para facilitar su manejo y un teléfono móvil con sistema operativo *Android* para realizar pruebas de rendimiento. En el cuadro 4.1 se muestra una lista con lo detallado anteriormente.

4.2.2 Características Software

Descripción	Nombre	Versión
Desarrollo de videojuego	<i>Unity</i>	2018.3.6f
IDE de programación	<i>Visual Studio 2017</i>	15.9.6
Realidad Aumentada	<i>Vuforia Engine</i>	8.0.10
Modelado y edición 3D	<i>Blender</i>	2.79b
Gestión de Configuración	<i>GitHub</i>	-
Gestión del proyecto	<i>Hack n' Plan</i>	-
Creación de artefactos y diagramas	<i>draw.io</i>	-
Redacción de la memoria del documento	<i>TexShop(Latex)</i>	4.27

Cuadro 4.2: Lista de características Software utilizadas

Las características software que se han utilizado se pueden ver en la tabla 4.2, se han escogido estas herramientas al resultar las más accesibles, ya sea por tratarse de herramientas libres y gratuitas o facilidad a la hora de utilizarse.

⁵"La experiencia del usuario abarca todos los aspectos de la interacción del usuario final con la empresa, servicio o producto". Fuente: <https://www.nngroup.com/articles/definition-user-experience/>

Capítulo 5

Arquitectura

ESTE capítulo describe la arquitectura software de *Santiago de la Torre RA*. Se pretende explicar con una descripción del juego en qué consiste y qué elementos pueden contenerse en él una vez que el jugador se encuentre jugando. A esta descripción le sigue un análisis técnico de la estructura interna del juego desde un punto de vista general que permita visualizar las escenas que se encuentran en él y posteriormente detallar los componentes y sistemas que intervienen en cada una de ellos.

5.1 Descripción del juego

Santiago de la Torre RA (Realidad Aumentada) es un videojuego en el que se pretende que el jugador recorra el castillo de Santiago de la Torre¹, el cual es una fortaleza medieval que se encuentra anexionada al municipio de San Clemente². Actualmente el castillo se encuentra en estado de ruina por lo que se espera que este videojuego sea un prototipo de lo que el jugador podrá hacer dentro del castillo. Hay diversos puntos de interés turístico y cultural dentro del castillo que representan parte de su propia historia, el jugador tendrá que visitar estos puntos en persona donde se encontrarán marcadores que representarán minijuegos o pequeños desafíos que el jugador debe superar para poder avanzar y visitar otros puntos.

El videojuego comienza con una pantalla de título en la que aparece el modelo del castillo y varios botones, uno que lleva a la selección de minijuegos, otro que permite entrar en las opciones del videojuego como ajustes de audio y otro que permite salir del videojuego.

Debido a la ambición del proyecto, se ha considerado el uso del plano del castillo como base para establecer los puntos de minijuegos, donde el jugador podrá seleccionar un punto que le llevará a la zona del castillo en un modelo 3D, que junto al plano del castillo servirán de referencia al jugador para saber a qué localización ir. En la figura 5.1 se muestra cómo se representa el mapa con el plano del castillo y los minijuegos.

Cuando el jugador seleccione sobre un punto en el mapa, si este se encuentra bloqueado, puede desbloquearlo usando una llave que adquiere al completar minijuegos, por defecto el primer minijuego no estará bloqueado. Debido al tiempo del que se disponía, se ha realizado

¹Wikipedia: Castillo de Santiago de la Torre

²<https://goo.gl/maps/X8cqbL2pEDptQqseA>



Figura 5.1: Captura del menú de selección de minijuegos

un solo minijuego que se encuentra ubicado e inspirado en la zona de la *Torre del Homenaje* del castillo.

El minijuego que se ha desarrollado se llama *¡Defiende la Torre del Homenaje!* y su objetivo es defender de las diferentes hordas de enemigos que tienen como objetivo atacar a la Torre hasta agotar su vida. Para defenderla, el jugador dispone de dos herramientas:

- *Espingarda*³, el cual es un fusil del siglo XV que se utilizaba por los españoles y se conoce que era una de las principales armas de las que disponía las defensas del castillo en caso de invasiones.
- *Torre con Cañón* es un elemento que se ha utilizado para incrementar la jugabilidad con la realidad aumentada. Se trata de una torre con un cañón que apunta a los enemigos más cercanos y tiene una gran potencia de fuego. El jugador deberá usar un marcador para colocar la torre de manera estratégica y posteriormente utilizarla para disparar a los enemigos. Tiene una vida limitada por lo que si los enemigos consiguen destruirla, el jugador tendrá que recolocarla.

Los enemigos a los que el jugador debe enfrentarse aparecen por rondas en las que a cada una se suma la dificultad apareciendo en zonas estratégicas para llegar a la torre lo antes posible, al eliminar a un enemigo el jugador obtendrá una puntuación dependiendo del tipo de enemigo que se sumará al récord en este minijuego, hay tres tipos de enemigos:

- *Soldado* se mueven rápido y atacan con una *espingarda*, por lo que tienen el mismo daño que el jugador. Tienen una característica especial y es que son los únicos enemigos que detectan cuándo se coloca una Torre con cañón, lo que provoca que cambien de objetivo y ataquen a esta torre. Tienen una vida reducida comparada con el resto de enemigos.
- *Cañonero* tienen una velocidad media, disparan bolas de cañón con una gran potencia

³<https://es.wikipedia.org/wiki/Espingarda>

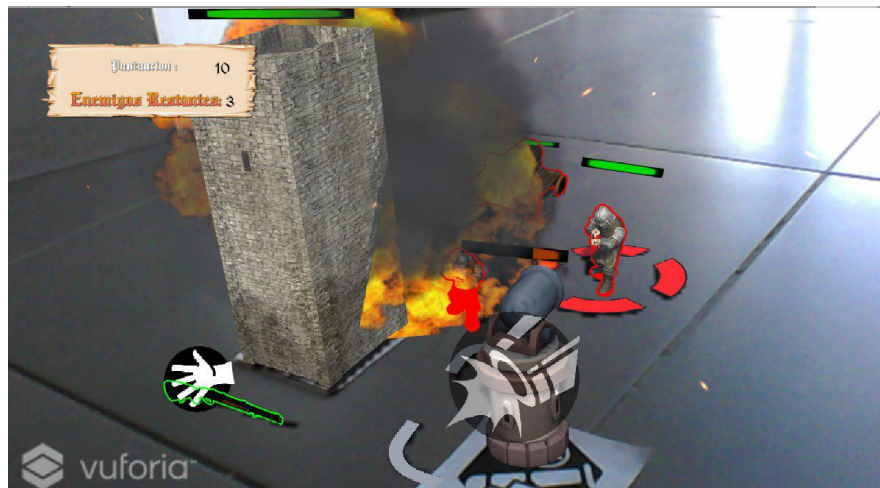


Figura 5.2: Captura de la Torre con Cañón apuntando a los soldados

de fuego y además a una gran distancia por lo que es un enemigo muy amenazador si el jugador se distrae mucho.

- *Ariete* son los más lentos, pero si consiguen llegar a la Torre del Homenaje tienen un daño considerablemente superior al resto, también son los más resistentes por lo que el jugador deberá eliminarlos antes de que se acerquen demasiado.

Después de superar un número de rondas determinado, las oleadas de enemigos terminarán y aparecerá una llave para recoger en lo alto de la torre que se podrá utilizar para desbloquear el siguiente minijuego y así el jugador explorará otras zonas del castillo. Si el jugador no consigue superar el número de rondas, aparecerá de nuevo el menú para reintentar el minijuego.

El progreso del jugador se guarda entre minijuegos, de forma que si el juego se cierra, se almacenará la mejor puntuación que haya obtenido y si se ha completado o no el minijuego, así como el número de llaves.

Santiago de la Torre RA constará de un archivo ejecutable con extensión *.apk* para Android, para poder iniciar el juego, sólo será necesario instalarlo en el terminal e iniciarlo.

5.2 Visión general de la arquitectura

Para comprender mejor la arquitectura y su desarrollo es conveniente seguir un enfoque *top-down* según el cual se presentan en un nivel de abstracción alto los componentes del sistema para así exponer en una panorámica las partes del sistema que en conjunto componen el proyecto diseñado y desarrollado.

Al tratarse de un videojuego desarrollado en *Unity*, es necesario indicar la arquitectura de componentes que ofrece *Unity*, en la que los objetos tienen comportamientos y propiedades determinados por los componentes que estos tengan asociados. Por lo general, un *script* que

se asocia a un objeto en *Unity* es una clase que hereda de la clase *MonoBehaviour*⁴. De esta clase se derivan diferentes comportamientos como funciones y propiedades que facilitan la tarea del programador a la hora de utilizar funcionalidades relacionadas por ejemplo con la interfaz de usuario, eventos que se ejecutan cuando el juego detecta entradas, cuándo el objeto colisiona con otro, etc.

Además el proyecto a lo largo de todos sus componentes trata de seguir el *principio de responsabilidad única* que según se define en [18] una clase sólo debe tener una razón para cambiar. Esto es que si una clase tiene dos responsabilidades, es conveniente separarlas para que si los requisitos del proyecto cambian, afecten de manera aislada a los componentes. Esto encaja totalmente con la filosofía de los componentes de *Unity* y además con la metodología con la que se trabaja.

Para visualizar de manera clara cómo es la arquitectura del sistema desde un punto de vista general y abstracto, se han agrupado los componentes más fundamentales de cada una de las escenas de *Unity*⁵ que componen el videojuego como se muestra en la figura 5.3.

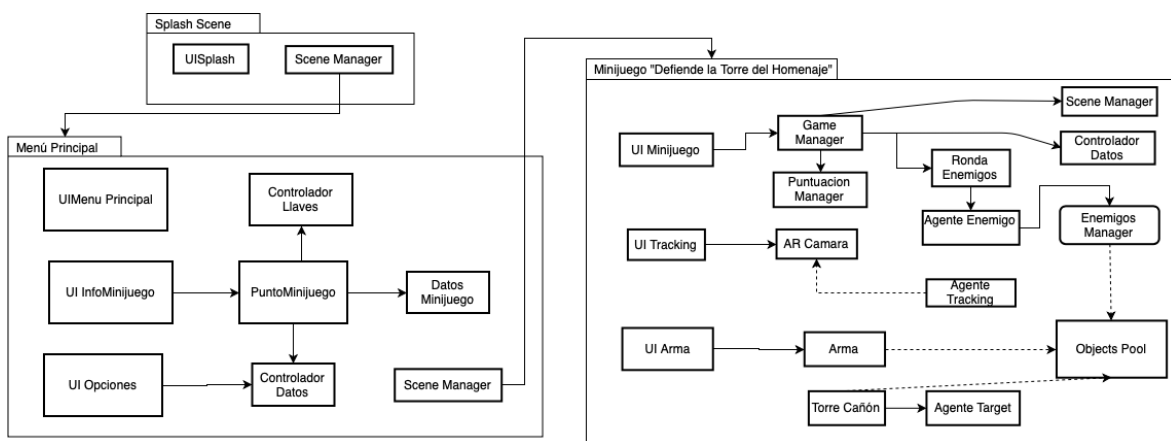


Figura 5.3: Diagrama de clases del videojuego agrupado en escenas

A continuación se explican detallando ligeramente los componentes según las escenas:

- **Splash Scene** En esta escena sólo hay dos clases, *Scene Manager* el cual será el gestor de las escenas de *Unity* y se encarga de cargar las escenas y mostrar una pantalla con el porcentaje de carga, esta clase también se encuentra en el resto de escenas debido a que es fundamental para la gestión de minijuegos y escenas. También tiene una pequeña clase que muestra el logo de la empresa con la que se colabora *Akura Games*.
- **Menú Principal** En el menú principal se pueden encontrar diferentes clases relacionadas con la gestión de los menús del juego. La función de estas clases es leer la entrada del jugador. Por otro lado están las clases relacionadas con los puntos de minijuegos

⁴API de la clase *MonoBehaviour* <https://docs.unity3d.com/ScriptReference/MonoBehaviour.html>

⁵Una escena contiene el entorno y menús del videojuego, esencialmente es donde se compone el videojuego en piezas. Fuente: <https://docs.unity3d.com/Manual/CreatingScenes.html>

que se encargan de gestionar la información de los minijuegos, consultar las llaves para poder desbloquearlos y la clase *Controlador Datos* que sirve para almacenar toda la información que sea necesaria de forma persistente.

- **Minijuego Defiende la Torre del Homenaje** En la escena del minijuego se encuentran todas las clases relativas a la lógica del minijuego con un núcleo en la clase *Game Manager* que se encarga de actualizar la interfaz del minijuego, consultar la puntuación del jugador, generar rondas de enemigos, cargar escenas y consultar la información del minijuego. Por otro lado están las clases relacionadas con el jugador, como es la clase de *AR Camara* que se encarga de conocer cuándo la cámara apunta al marcador que activa el minijuego, y los objetos que puede utilizar el jugador como es el *Arma* y la *Torre Cañón*.

Es interesante reflejar la complejidad del sistema a la hora de conocer cómo se comunican estas clases, ya que al combinar la interfaz del editor de *Unity* e intentar realizar sistemas que pretendan controlar todo desde una única clase suele acabar adquiriendo una gran complejidad para comprender bien qué hace, por lo que se ha procurado seguir un patrón de *Modelo-Vista-Controlador* [19] que separe los datos y la lógica de su representación.

5.3 Menú Principal

Santiago de la Torre RA cuenta con una escena que actúa como nexo de unión entre la visualización del castillo y los minijuegos en realidad aumentada, de esta manera el jugador puede conocer qué zonas son explorables en el castillo y reduce considerablemente la complejidad de la localización del jugador dentro del castillo.

A lo largo de esta sección se detallará en profundidad cada uno de estos componentes.

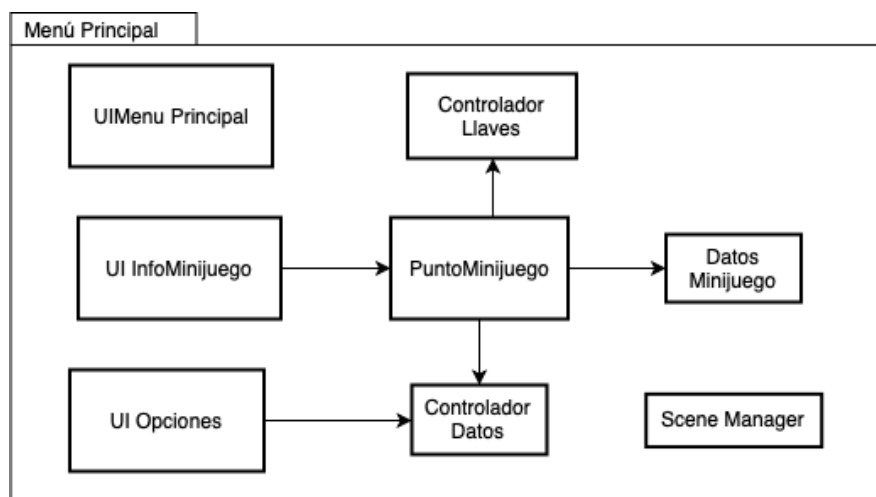


Figura 5.4: Componentes del Menu Principal

El comportamiento que sigue esta escena es sencillo se ve reflejado en la figura 5.5, las transiciones que se ven indican cómo es el flujo de ventanas que sigue el menú principal

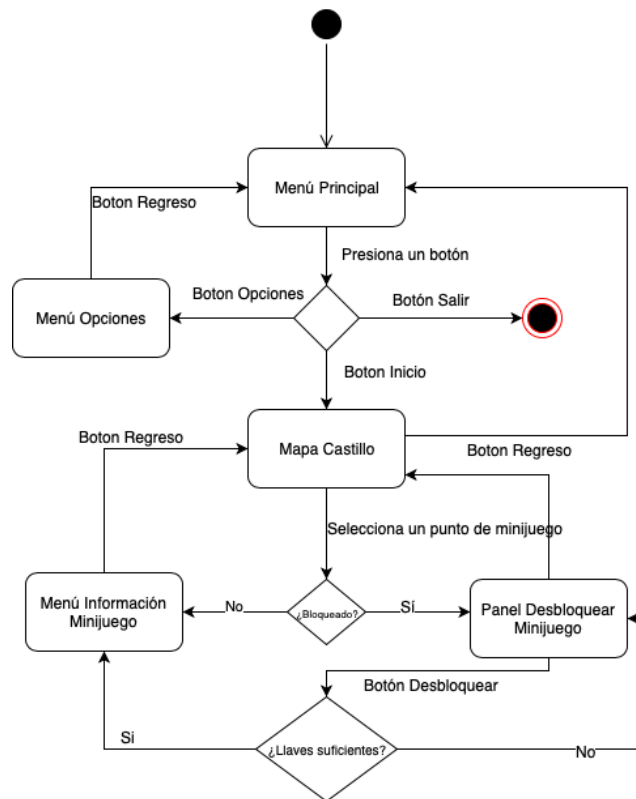


Figura 5.5: Diagrama de estados que siguen los componentes de la interfaz del menú principal

dependiendo de la interacción del jugador. Principalmente muestra un menú con el título del videojuego, en el fondo el modelo del castillo de *Santiago de la Torre* y tres botones con diferentes comportamientos entre ellos:

- **Inicio** Esta opción abre el mapa del castillo donde el jugador tiene la opción de seleccionar algún punto en el minijuego.
- **Opciones** Abre un menú que permite ajustar el volumen de la música y los efectos que se reproduce en el juego.
- **Salir** Permite cerrar el videojuego.

Una vez el jugador abra el mapa del castillo y seleccione un punto de minijuego, puede ocurrir lo siguiente, si está bloqueado se abrirá un panel con un cuadro de diálogo con la opción de desbloquearlo con una llave. Si no está bloqueado, la cámara en la escena mostrará la zona en el modelo del castillo donde se situaría la habitación acondicionada para el minijuego, es decir, donde haya diferentes marcadores para interaccionar, el menú con la información del minijuego y un botón que permita cargar la escena del propio.

Esta escena contiene un conjunto de objetos agrupados de forma jerárquica que forman el comportamiento y la representación del videojuego llamados *GameObjects*⁶. Para detallar

⁶<https://docs.unity3d.com/560/Documentation/Manual/class-GameObject.html>

los componentes y sus funcionalidades se procederá primero indicando la estructura de estos objetos, y a partir de ahí explicar cómo funcionan.

5.3.1 Componentes de la interfaz

Los componentes que se pueden clasificar en cuanto a la interfaz se diferencian en dos tipos, uno que se ajusta al tamaño de la pantalla dependiendo de la resolución y otro que se ajustan dependiendo de la posición relativa respecto al mundo en la escena ⁷.

En este caso se han utilizado ambos, el que se ajusta a la resolución de la pantalla se utiliza para la interfaz relativa al menú principal y las opciones, y el que se ajusta a la posición en el mundo para mostrar la interfaz del mapa del castillo, se ha hecho así ya que la cámara utiliza un componente llamado *CineMachine*⁸ el cual permite trasladar la cámara a diferentes posiciones y tratarla como si se trabajara con cinemáticas, lo que ha facilitado el trabajo a la hora de enseñar las diferentes zonas del castillo.

Los *GameObjects* que se pueden encontrar en la interfaz escalada por pantalla son:

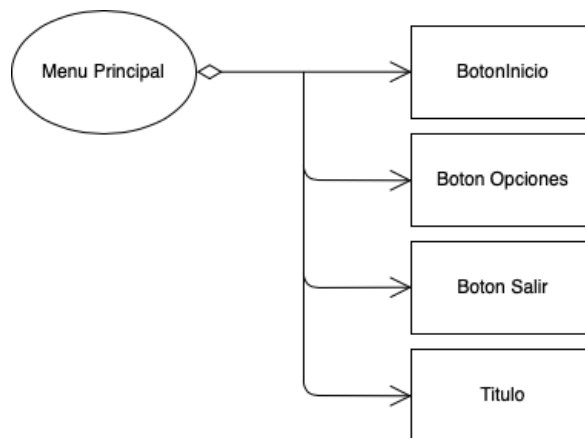


Figura 5.6: *MenuPrincipal* y sus componentes

- **MenuPrincipal** Este objeto (figura 5.6) está compuesto por tres botones, los cuales tienen comportamientos asociados a los estados que se reflejan en la figura 5.5, los botones⁹ en *Unity* tienen un apartado de *listeners* a los cuales se les puede asignar un *GameObject* y llamar a cualquier método público que contenga sus componentes, esto facilita mucho el trabajo a la hora de asignar funciones a botones sin tener que pasar por el código. Además también tiene un *Título* que simplemente se trata de un componente *TextMeshPro*¹⁰, el cual se trata de un fragmento de Texto que tiene algunas propiedades de los modelos 3D como la posibilidad de asignarle materiales o *Shaders*¹¹. Se ha elegido este tipo de componente ya que permite una mejor visualización

⁷Canvas Scaler Component, Fuente: <https://docs.unity3d.com/Manual/script-CanvasScaler.html>

⁸<https://unity3d.com/es/learn/tutorials/topics/animation/using-cinemachine-getting-started>

⁹<https://docs.unity3d.com/ScriptReference/UI.Button.html>

¹⁰<https://docs.unity3d.com/Packages/com.unity.textmeshpro@2.0/manual/index.html>

¹¹<https://docs.unity3d.com/es/current/Manual/Shader.html>

al ser escalado.

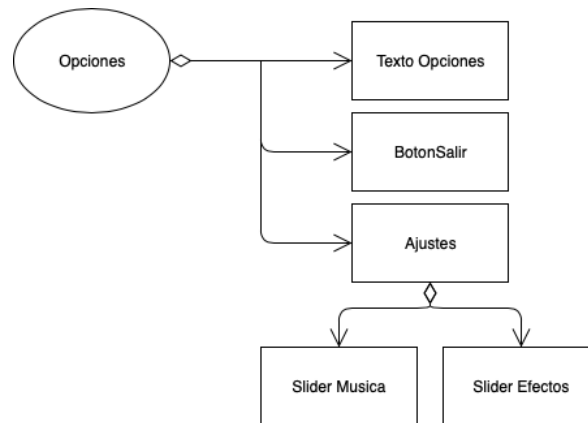


Figura 5.7: *Opciones* y sus componentes

- **Opciones** Este objeto (figura 5.7) tiene un texto que actúa a modo de título del menú, un botón para regresar al Menu Principal, y contiene otro *GameObject* llamado Ajustes, el cual tiene dos *Sliders*¹², son barras que representan un porcentaje, en este caso son interaccionables por el jugador por lo que puede asignar cuál es el porcentaje de cada una. Se utilizan para asignar el volumen de la música y de los efectos.
- **InfoMinijuego** Este objeto (figura 5.8) es un poco más complejo ya que tiene dos *paneles* que sirven para ajustar y ordenar mejor la interfaz en la pantalla, además de un texto que sirve para mostrar el título del minijuego. En el *PanelInfo* se muestra un desplegable que contiene dos textos, uno para mostrar la descripción del minijuego y otro la mejor puntuación obtenida, también cuenta con un *Toggle*¹³ que sirve para mostrar si el minijuego está completado o no. Por otro lado está el Panel de Acción que servirá para captar la interacción del jugador, cuenta con tres botones, *Jugar* para cargar la escena del minijuego, *Regreso* para desactivar el objeto y mostrar el menú principal y por último el botón para desplegar el panel de *Info*.

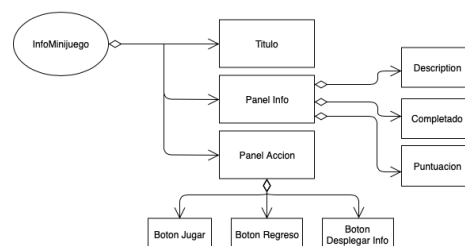


Figura 5.8: *InfoMinijuego* y sus componentes

En cuanto a la interfaz que se ajusta según la posición cabe decir que esta interfaz está sujeta a las propiedades de la *Cámara principal*¹⁴ de la escena y contiene otros componentes

¹²<https://docs.unity3d.com/es/current/Manual/script-Slider.html>

¹³<https://docs.unity3d.com/es/current/Manual/script-Toggle.html>

¹⁴<https://docs.unity3d.com/es/current/Manual/class-Camera.html>

que ayudan a una mejor visualización, en la figura 5.9 se pueden observar los *GameObjects* que ésta tiene asociado, a continuación se pasará a explicar los componentes más relevantes.

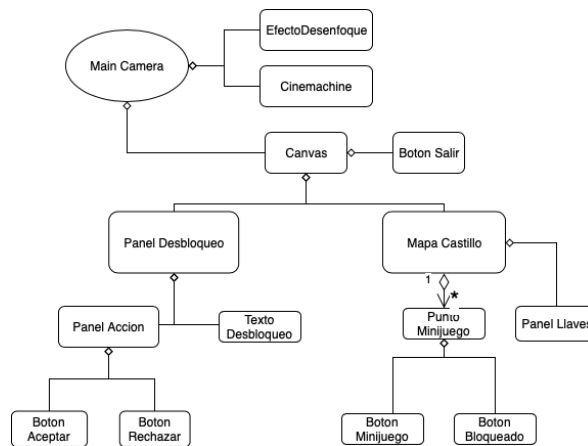


Figura 5.9: *Main Camera* y sus componentes

- **Efecto Desenfoque** Este es un plano que se coloca delante de la cámara, el cual contiene un material con un *shader* asociado que permite ver de forma desenfocada a través de él. Se ha utilizado el *shader* "*SimpleGrabPassBlur*"¹⁵
- **CineMachine** Este componente es bastante complejo ya que a su vez contiene otros componentes para su funcionamiento pero se puede resumir en que utiliza un conjunto de cámaras, las cuales se sitúan en el mundo, y mediante un *Animator*¹⁶ el cual es una máquina de estados que permite establecer comportamientos asociando componentes, en este caso, las cámaras o cualquier efecto que se desee, este componente se utiliza para situar cámaras en la zona del modelo del castillo en la que se encuentre el minijuego, de esta manera cuando se selecciona un minijuego, este componente cambia de estado y la cámara desde la que se encontraba el menú principal se traslada a la zona requerida.
- **Mapa Castillo** Este objeto contiene varios componentes, uno de ellos es un *Image*¹⁷ que se encarga de renderizar una imagen del plano del castillo, dentro de este, existen varios *Puntos de Minijuego* que representan el minijuego en la localización del castillo. estos contienen dos botones:
 - Botón de minijuego que muestra el icono del minijuego cuando está desbloqueado.
 - Botón bloqueado que muestra un icono de un candado cuando éste se encuentra bloqueado.

¹⁵<https://forum.unity.com/threads/simple-optimized-blur-shader.185327/>

¹⁶<https://docs.unity3d.com/es/current/Manual/Animator.html>

¹⁷<https://docs.unity3d.com/ScriptReference/UI.Image.html>

Se ha realizado de esta manera ya que resultaba más sencillo activar o desactivar un botón antes que cambiar el icono y las funciones a las que llama.

- **Panel Llaves** En este objeto contiene una imagen para representar las llaves y un texto que indica el número de llaves que el jugador dispone.

5.3.2 Selección de Minijuegos

Para la realización de los minijuegos, cabe decir que estos necesitan almacenar cierta información de forma persistente a la vez que tienen una representación en la pantalla por lo que existen diferentes *GameObjects* que en conjunto permiten la funcionalidad de selección de minijuegos.

Los minijuegos se definen como *Punto Minijuego* dentro del mapa del castillo, estos minijuegos se guardan como un *Prefab*¹⁸ en *Unity*, que permite almacenar los *GameObject* con sus componentes y valores para reutilizarlos, se ha realizado de esta manera para facilitar la tarea de crear nuevos minijuegos y asignarlos en un punto del castillo.

En la figura 5.10 se muestran las clases que componen todo el sistema de minijuegos, como se puede ver se ha procurado separar la parte de representación (*UIMinijuego* y *UIPanelDesbloqueo*), los controladores que se encargan de transportar la información, modificarla y guardarla (*PuntoMinijuego* y *Controlador Llaves*) y el modelo (*DatosMinijuego* y *Minijuego*). Así se permite un flujo de la información muy directo, ya que el menú de *UIMinijuego* y *UIPanelDesbloqueo* no siempre se encuentran activos, sólo hasta que el jugador interactúe con un *PuntoMinijuego*, y éste se encarga de manipular la interfaz del minijuego o activar el panel de desbloqueo. Es interesante resaltar que el *PuntoMinijuego* al ser un *script* que se utiliza en varios *GameObjects*, cada vez que se desee activar por ejemplo el *UIPanelDesbloqueo*, este debe apuntar al *PuntoMinijuego* que lo ha activado para desbloquearlo. A continuación se explicarán cómo estas funcionan explicando cada una de las clases.

- **DatosMinijuego** Esta clase tiene el estereotipo de *ScriptableObject*¹⁹, se trata de una clase que permite generar *Assets*²⁰ que son capaces de almacenar información de forma persistente y modificarla, esto es ideal para poder crear información de los minijuegos desde el editor de *Unity* ya que si se desea crear un nuevo Minijuego, para su información más básica como un nombre, título, descripción, sus iconos y si se encuentra bloqueado o no y la escena asociada al minijuego.
- **Minijuego** Esta es una estructura de datos que sirve de soporte cada vez que se desee guardar la información del jugador que obtiene del minijuego, como si lo ha completado y cuál ha sido su mejor puntuación, esta clase se encuentra en el contexto de los datos del jugador que se mencionan en el punto 5.3.3.

¹⁸<https://docs.unity3d.com/es/current/Manual/Prefabs.html>

¹⁹<https://docs.unity3d.com/es/530/Manual/class-ScriptableObject.html>

²⁰<https://docs.unity3d.com/Manual/AssetWorkflow.html>

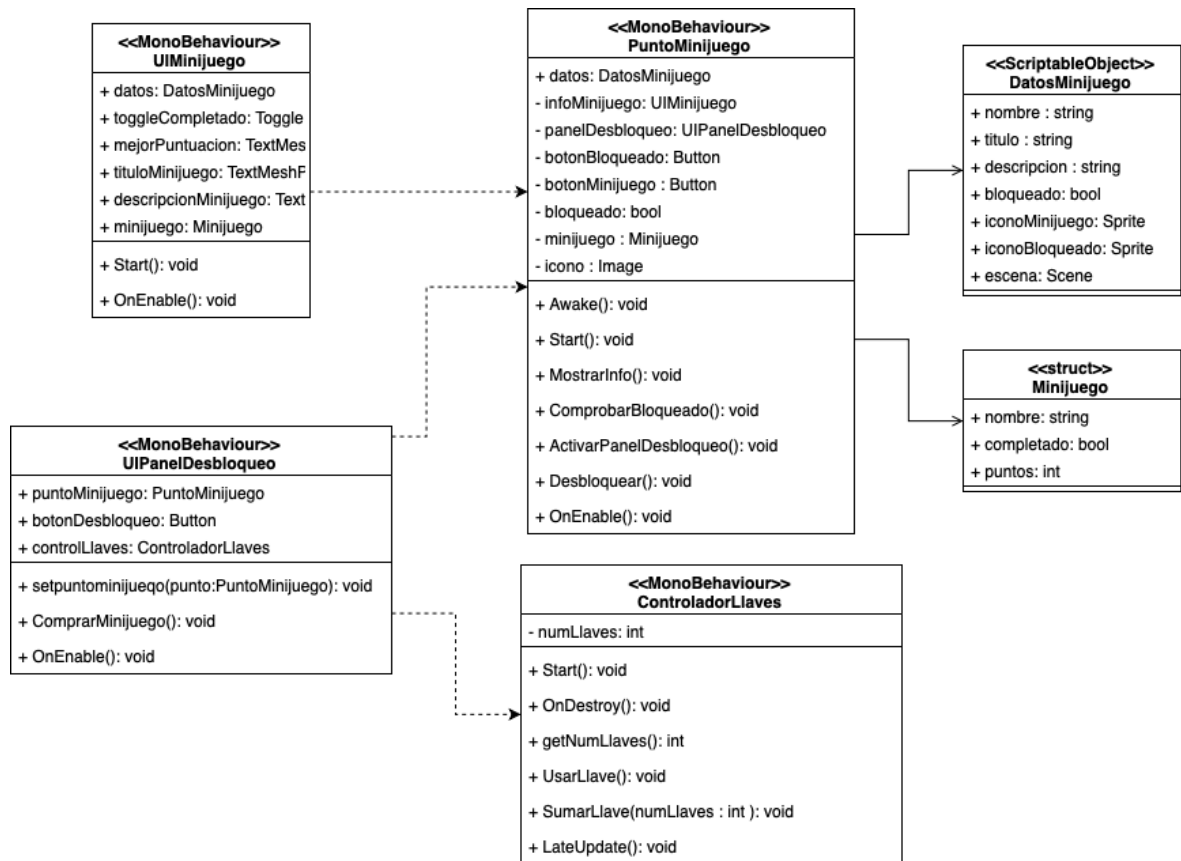


Figura 5.10: Diagrama de clases del sistema de Minijuegos

- **PuntoMinijuego** Esta clase se puede entender como el núcleo de los minijuegos, ya que también es la que se utiliza para los *prefabs* de puntos de minijuego, hace uso de los *DatosMinijuego* y la estructura de *Minijuego* para cargar la información que se carga desde el editor y actualizar los datos que se utilizarán en la parte de la interfaz. Además tiene algunos métodos que sirven para activar y desactivar algunos componentes de la interfaz y un método que sirve para desbloquear el minijuego que tiene asignado. Esto es, cambia el estado de *DatosMinijuego.bloqueado* a false, y al tratarse de un *ScriptableObject* se guardará si se cierra el juego y se vuelve a abrir.
- **ControladorLlaves** Esta es una clase que en este contexto sirve para mostrar al jugador las llaves que tiene y cargarlas cada vez que el jugador inicie el juego, tiene los métodos necesarios para sumar o restar al contador del número de llaves además de guardar de forma persistente esta información.
- **UIMinijuego** Esta clase sirve exclusivamente para sostener la información y mostrarla cada vez que se habilite su *GameObject* con el método *OnEnable()*, de esta manera no es necesario recuperar la información del minijuego en cada *frame* del juego, sino que cuando se habilita recupera la información del minijuego como si está completado, la mejor puntuación, descripción, etc.
- **UIPanelDesbloqueo** Esta es una clase que se encarga de realizar algunas funcionali-

dades para facilitar el control del *desbloqueo de minijuegos*, ya que tiene asignado un botón con un *listener* que se actualiza en tiempo de ejecución dependiendo del *PuntoMinijuego*. De esta manera se puede utilizar el mismo *script* de *PuntoMinijuego* a diferentes *GameObject* y cada vez que el jugador hace click sobre un minijuego bloqueado, se actualiza la información del *PuntoMinijuego*. Esto permite que incluso se puedan realizar modificaciones de manera más sencilla como por ejemplo establecer un coste de llaves dependiendo del minijuego.

Así pues a la hora de seleccionar un punto en el mapa del castillo se activarán los paneles convenientes si está o no bloqueado, y además si se desean realizar más minijuegos sólo hay que rellenar la información y colocarlo en otro punto del castillo. De esta manera se obtiene una arquitectura que permite ser escalable a la hora de incluir nuevos minijuegos puesto que la gestión de persistencia y representación se realiza automáticamente.

5.3.3 Sistema de Almacenamiento

Hoy en día todos los videojuegos tienen un módulo encargado para guardar el progreso del jugador y permitir que este tenga una sensación de progreso sobre el juego, además de permitir la competición mediante la comparación de estadísticas con otros jugadores.

En este caso al tratarse de un prototipo, se ha realizado un sistema de guardado sencillo pero escalable, anteriormente se comentaba cómo con los *ScriptableObject* se puede almacenar información persistente y es que resultan una herramienta muy potente y sencilla, aunque no es muy manejable en un entorno ajeno a *Unity* por lo que es conveniente tener un módulo aparte que sea capaz de almacenar la información en caso de que se quiera cambiar la base de datos o incluso las tecnologías con la que se trabaja.

Para ello, *Unity* proporciona una directiva llamada *PlayerPrefs*²¹ la cual sirve para almacenar información en el dispositivo en el que se ejecute el juego, esta herramienta es muy útil por su sencillez y además por resultar ser multiplataforma. Es muy parecido a un diccionario, se establece una clave con un nombre y un valor, este valor puede ser *string*, *int* o *float*.

Sin embargo, utilizar esa directiva para almacenar información por separado no es muy escalable, por lo que es preferible utilizar un *script* asociado a un *GameObject* que permita controlar esta información y así establecer un único flujo a través del que se gestiona la información, se ha creado un *script* llamado *ControladorDatos* el cual utiliza el *PlayerPrefs* para cargar y guardar los datos. Esto lo hace cuando se inicia el juego y se deshabilita su *GameObject* respectivamente, de esta manera se protege la información en caso de que se cierre el juego de forma inesperada.

En la figura 5.11 se muestra un diagrama con las clases involucradas en el sistema de guardado. Este sistema se encuentra aislado de cualquier representación en interfaz y sólo se

²¹<https://docs.unity3d.com/ScriptReference/PlayerPrefs.html>

encarga de mantener actualizados los *DatosJugador*, el cual es un campo estático para poder modificarlo sólo desde el *ControladorDatos*. Además los métodos encargados de guardar y cargar los diferentes datos también son estáticos para permitir ser invocados desde cualquier *script*. A continuación se explicarán las partes que componen este sistema.

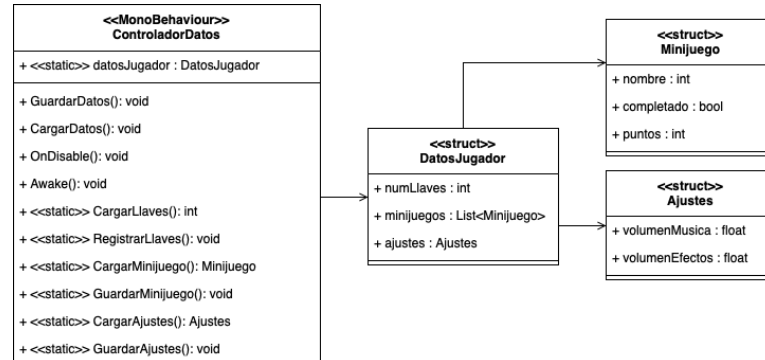


Figura 5.11: Diagrama de clases del sistema de guardado

- **DatosJugador** Se trata de la estructura que da el formato a los datos que se desean almacenar de forma persistente, por un lado están las llaves que servirán para desbloquear los minijuegos, una lista con la información de los *minijuegos* y la estructura de *Ajustes* que permite guardar la configuración del volumen del juego.
- **ControladorDatos** Es una clase que se adhiere a cualquier *GameObject* y permite mantener de forma estática los *datos del jugador*. Cuando se llama a *Awake()*, es decir, cuando la instancia del *script* se encuentra cargando, hace una comprobación si en *PlayerPrefs* existe la clave "*DatosJugador*", si existe, carga los datos, si no, crea la nueva clave y guarda los datos (se juega por primera vez). El valor que se almacena en *PlayerPrefs* se trata de un *String*, para darle el formato adecuado y poder recuperar los valores de manera correcta, se ha utilizado la función de *JSONUtility*²², se trata de una función que permite convertir strings a formato *Json* y viceversa. El resto de funciones se encargan de devolver y guardar los datos estructurados dependiendo de su funcionalidad, si se trata de cargar un minijuego, los ajustes o las llaves.

Se ha decidido optar por el formato *JSON* debido a su facilidad para serializar la información que se requiera, no se trata de una estructura de datos definida sino que se puede ampliar lo que se requiera, tiene sentido ya que primero se comenzó almacenando la información de ajustes, y a partir de ahí se amplió la estructura de *DatosJugador*. Además también desde el punto de vista de la seguridad, es interesante utilizar este método, ya que es más sencillo encriptar una cadena de texto y almacenarla en una base de datos con *JSON*.

Así pues con este pequeño sistema se da por cumplida la norma de calidad de desarrollo móvil definida como *Persistencia de Datos* en la que la aplicación gestionará información

²²<https://docs.unity3d.com/ScriptReference/JsonUtility.html>

alocada de forma persistente, en este caso se delega toda la gestión a *Unity* para agilizar el proceso.

5.3.4 Controlador de Audio

En todos los juegos casi es necesario incluir sonidos o música para llamar la atención del jugador o potenciar su experiencia, aunque esto no es una tarea fácil ya que en este caso no se dispone ni de las herramientas para generar piezas musicales ni el conocimiento, por lo que mayormente toda la música o efectos de sonido se han extraído de páginas que ofrecen recursos para videojuegos de forma gratuita, principalmente se ha tratado de la página *opengameart.org*²³

Los *GameObject* que se requieran que emitan sonidos necesitan un componente llamado *AudioSource*²⁴, que proporciona las opciones necesarias para asignar un clip de audio, ajustar el volumen, efectos, prioridad, etc. Y como en la realidad, es necesario tener un receptor para poder escuchar el audio, en este caso se trata de un componente llamado *AudioListener* que está asociado a la cámara, pues esta representa al jugador.

Sin embargo, es interesante poder controlar los componentes de audio de manera más directa, sin tener que recurrir a comprobar y cambiar el volumen de todos los componentes de audio que se encuentren en la escena y además poder agruparlos dependiendo del tipo de sonido que sea. Para ello, *Unity* ofrece un *Asset* llamado *Audio Mixer*²⁵ que permite crear pistas de audio que servirán de salida a los diferentes componentes *AudioSource*. De esta manera a cada componente se le asocia una pista dependiendo de qué tipo de sonido se desee. Se han creado tres pistas de audio, una para los efectos de sonido, otra para la música y una que agrupa las dos siendo la máster.

También para facilitar la tarea de gestionar la configuración de audio que el jugador seleccione y poder controlar desde el código cómo se comporta o cuándo activar algunos clips de audio generales como es la música de fondo, en la figura 5.12 se muestran algunos de los *script* que se han utilizado para controlar esto. El *script* de *OpcionesAudio* sirve para el

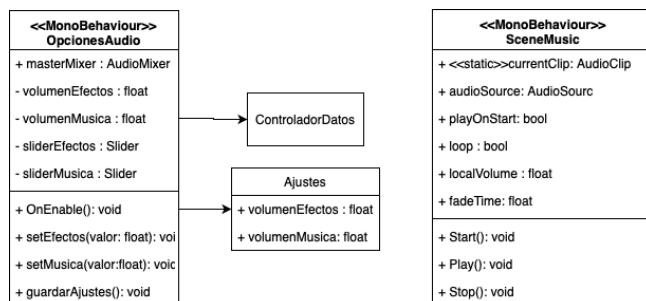


Figura 5.12: Diagrama de clases del sistema de los sistemas de Audio

²³<https://opengameart.org>

²⁴<https://docs.unity3d.com/ScriptReference/AudioSource.html>

²⁵<https://docs.unity3d.com/Manual/AudioMixer.html>

menú de opciones del menú principal tanto en la parte de la interfaz como la de almacenar la información, al tratarse de sólo dos ajustes se ha preferido controlar directamente la entrada del jugador con *Sliders* y modificar el valor en el *AudioMixer* haciendo la distinción de cada *Slider* por cada pista de audio en el *AudioMixer*

Para facilitar la labor de gestionar la música de fondo en cualquier escena se ha recurrido a un *script* aparte llamado *SceneMusic* que facilita la integración de cualquier clip y controlar desde cualquier otro componente la música que suena en una escena, esto es interesante para no acoplar todo el comportamiento de la música de fondo en un sólo *script*. Éste hace uso de un paquete que se ha obtenido desde el *AssetStore*²⁶ de *Unity* para facilitar y agilizar el proceso de control sobre el audio, que además de música de fondo ofrece otras opciones.

5.3.5 Gestor de Escenas

El menú principal es la escena que actuará como nexo de unión entre el conjunto de mini-juegos y sus escenas, por lo que es necesario que aquí se defina un gestor que permita realizar la carga asíncrona de escenas, para ello se utiliza un *prefab* que a la vez que se encargue de gestionar la carga de escenas permita una visualización del progreso de carga, para que el jugador no tenga la sensación que el juego se ha bloqueado sino que necesita tiempo para cargar el contenido. Esto es fundamental, ya que al tratarse de un juego que utiliza realidad aumentada es necesario realizar una petición para el uso de cámara y esperar a su inicialización, además de todos los modelos y componentes que se encuentren en la escena que se pretende cargar.

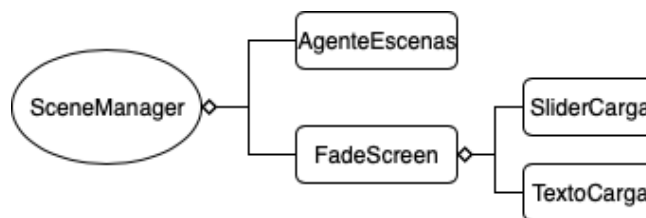


Figura 5.13: Diagrama de componentes del gestor de escenas

En la figura 5.13 se muestran los componentes que forman el *prefab* de *SceneManager*, el cual tiene varios componentes que son interesantes comentar:

- **AgenteEscenas** Este es un *script* que se asocia al *GameObject* para facilitar la gestión tanto de la representación de la pantalla de carga como de la carga de escenas. Para ello en la figura 5.14 se muestra el diagrama de clases que esta sigue, lo primero que hace uso del paquete de *Unity*, *SceneManagement*²⁷ que ofrece una serie de funciones propias de *Unity* para facilitar la carga de escenas, en este caso se han usado las funciones de *LoadSceneAsync()* que permite cargar una escena de forma asíncrona, esto

²⁶<https://assetstore.unity.com/packages/tools/audio/sound-manager-audio-sound-and-music-manager-for-unity-56087>

²⁷<https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.html>

permite que mientras se esté cargando la escena se puedan seguir ejecutando bloques de código, en este caso se utilizan para mostrar la pantalla de carga y actualizar sus valores.

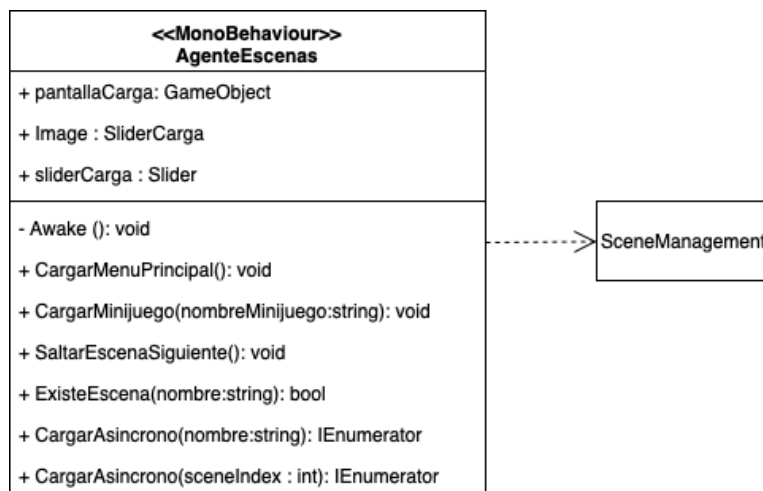


Figura 5.14: Diagrama de clases del Agente de Escenas.

- **FadeScreen** Este es un componente de interfaz sencillo que consiste en otros dos componentes, con una imagen de fondo negra, tiene un *Slider* que se actualiza con el porcentaje de carga que ofrece el *AgenteEscenas* mientras se carga de forma asíncrona la escena, y un *TextMeshPro* que sólo muestra Cargando...

Es interesante que este pequeño gestor sea un *prefab* ya que si se desea cargar una nueva escena es muy cómodo hacer uso del *AgenteEscenas* para cargar cualquier escena, ya que además ofrece métodos de comprobación de la existencia del nombre dentro de las escenas que se encuentran en la lista de compilación de escenas y además de ofrecer una pantalla de carga totalmente modificable desde el propio *prefab*.

5.4 Minijuego: Defiende la Torre del Homenaje

El minijuego que se ha llevado a cabo se llama *Defiende la Torre del Homenaje*, es un juego sencillo en el que el objetivo es evitar que la Torre del Homenaje sea destruida por diferentes enemigos que se acercarán a ella en oleadas, para esto, el jugador podrá hacer uso de dos tipos de objetos, un fusil que se adhiere a la cámara y un marcador en el que se sitúa una torre con un cañón para disparar a los enemigos.

Si bien este minijuego es sencillo está compuesto por muchos componentes que interactúan entre sí y plantear una visión general de este puede ser complicado pero se va a abordar desglosando los diferentes sistemas que actúan entre sí, como se ve en la figura 5.15 hay algunos componentes que se reutilizan como es el *SceneManager* y el *ControladorDatos* que son necesarios para regresar al *Menú Principal* y almacenar la información del minijuego respectivamente. Cabe decir que se ha procurado separar los componentes dependiendo de

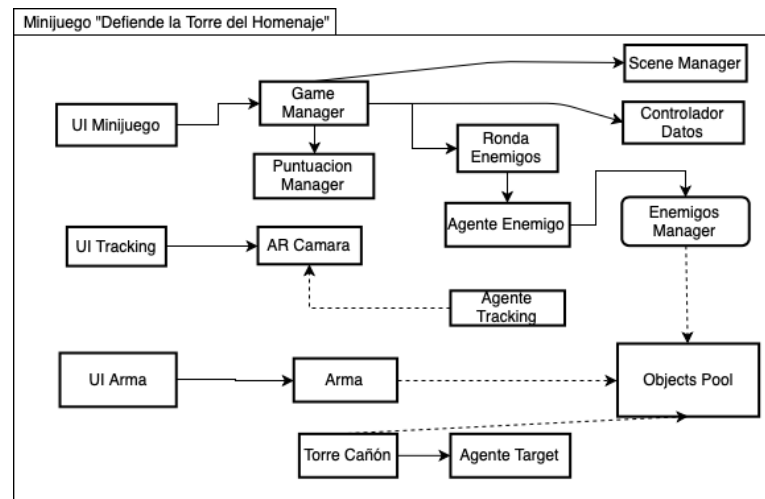


Figura 5.15: Diagrama de la Arquitectura del Minijuego *Defiende la Torre del Homenaje*

los elementos con los que interactúan aunque es necesario indicar que en numerosas veces se hace uso de un sistema de eventos²⁸ que proporciona *C#* para comunicar información sin necesitar acoplar los componentes, son los denominados *delegates* que actúan como punteros orientados a objetos. Esto permite diseñar y desarrollar los objetos que se componen en la escena de una manera más individual y centrarse en las funcionalidades sin tener en cuenta otro tipo de información de componentes externos.

5.4.1 Game Manager

Este minijuego centra toda la lógica de cuándo se gana y se pierde en un único *script* llamado *GameManager*, actúa a modo de máquina de estados y de encargado para preparar los diversos elementos que interaccionan en el minijuego como los enemigos, la interfaz con la información de la partida, etc.

En la figura 5.16 se refleja los diferentes estados por los que pasa el minijuego, a continuación se pasará a explicar el funcionamiento de éstos y sus transiciones:

- **Iniciar Juego** Este estado se activa una vez que el jugador encuentra el marcador correspondiente y prepara todo lo relativo a comenzar una nueva partida, como es establecer el número de ronda a 0, los valores booleanos de *ganaEnemigos* y *ganaJugador* a false, activar los elementos de la interfaz y reiniciar la vida de la torre del homenaje.
- **Comienza Ronda** Este estado se encarga de recoger la información de la ronda actual, indicar mediante la interfaz que se encuentra en una ronda en concreto y establecer a los enemigos en su posición.
- **Jugando** Aquí se activan a los enemigos, los cuales pasan a ser visibles al jugador y se reinician sus estados. Éste contiene un bucle cuya condición de salida es que el jugador elimine a todos los enemigos que se encuentran en la ronda o que la torre sea

²⁸<https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/delegates/using-delegates>

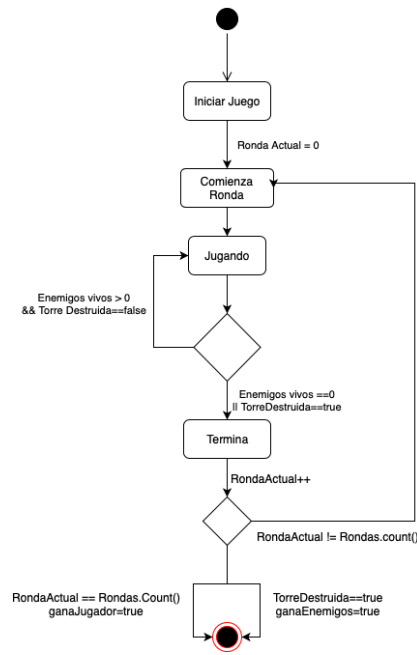


Figura 5.16: Máquina de estados del *GameManager*

destruida.

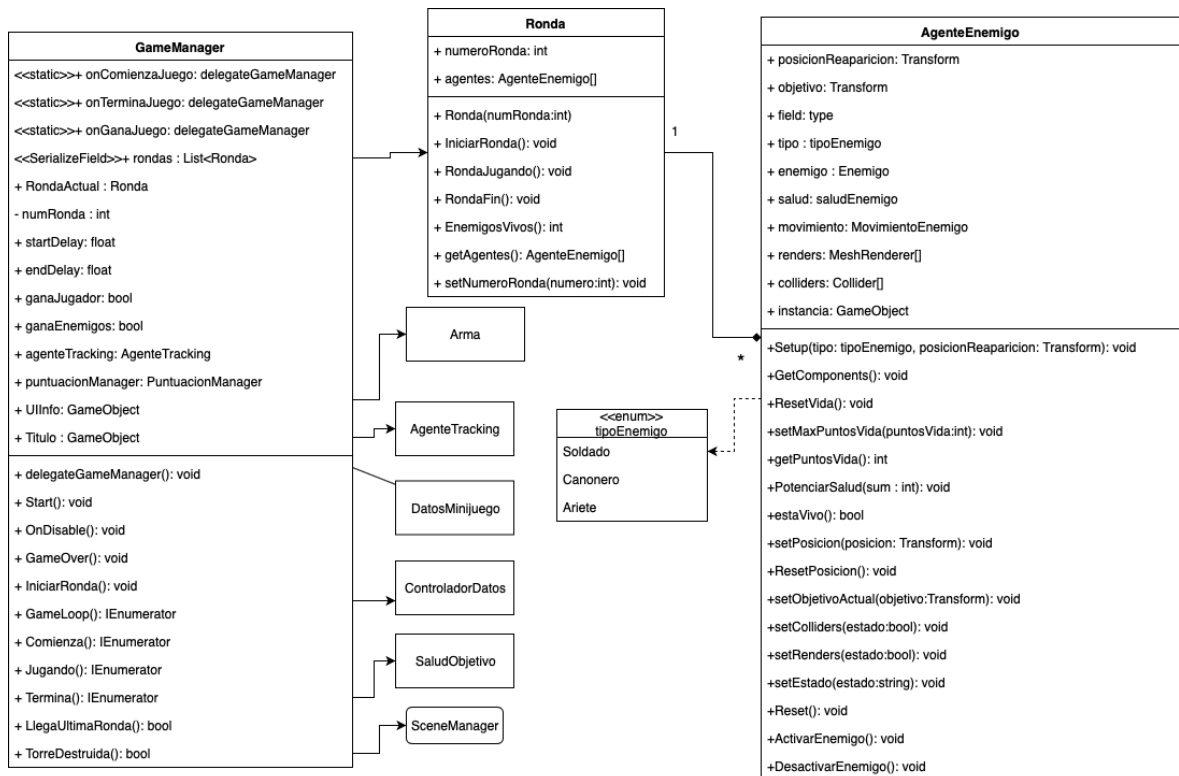
- **Termina** Se encarga de comprobar qué ha ocurrido durante la partida, si la torre ha sido destruida activa el valor booleano de *ganaEnemigos*, si el jugador ha conseguido llegar hasta la última ronda activa el valor booleano de *ganaJugador* y si no se ha llegado a la última ronda y no se ha destruido la torre vuelve al estado de *ComienzaRonda*.

Para comprender cómo el *GameManager* permite controlar tanto los enemigos como cuándo se cumplen ciertos objetivos es necesario enmarcarlo en un diagrama de clases como se refleja en la figura 5.17, sin embargo, se ha simplificado ya que hay algunos componentes que se pretende explicar con mejor detalle en secciones posteriores.

El diseño se ha llevado a cabo de esta manera debido a que es más sencillo en cuanto se trata de definir una lógica establecida, hacerlo todo desde un sólo *script* como en el *GameManager* que contiene todos los campos y métodos que permiten que el minijuego funcione. Además también hace referencias a otros *script* que son necesarios para su funcionamiento. Por otro lado están los *script* de *Ronda* y *AgenteEnemigo* que actúan más bien a modo de interfaz para llevar a cabo el control sobre qué ocurre en cada ronda del minijuego y qué ocurre con cada enemigo en una ronda en concreto.

Primero como se ha indicado en las transiciones del *GameManager* lo que desencadena el inicio del minijuego es encontrar el marcador, para ello el *GameManager* hace uso del *AgenteTracking*, el cual se inicia con un método síncrono que determina cuándo se detecta el marcador en el que se contiene el minijuego.

Una vez que lo encuentra pasa al método de *IniciarRonda()* el cual hace uso de la clase

Figura 5.17: Diagrama de clases que engloban al *GameManager*

Ronda, este campo se marca como *SerializeField*²⁹ el cual es un estereotipo que contiene Unity, el cual permite rellenar los datos que contienen la clase desde el editor de Unity, esto es, en el componente de *GameManager* existe un apartado en el que se puede editar el número de rondas, qué enemigos habrá en cada una y dónde van a reaparecer, de esta manera es muy sencillo modificar las rondas en caso de que se quiera ampliar el minijuego, reducir o subir la dificultad.

La clase *Ronda* contiene el número de la ronda y una lista de *AgenteEnemigo*, también contiene todos los métodos necesarios para iniciar las rondas, finalizarlas y comprobar el estado de los enemigos, además de toda la lógica de qué ocurre con los enemigos en cada momento, se ha contenido en esta clase debido a que acoplar todo el comportamiento de los enemigos en el *GameManager* suponía aumentar considerablemente la complejidad a lo que es una máquina de estados para controlar el minijuego. De esta manera se tiene por un lado los estados y por otro el control de los enemigos.

La clase *AgenteEnemigo* es el controlador que permite definir los enemigos según su tipo y su posición, además provee los métodos necesarios para controlar todas sus características como puede ser su salud, el movimiento, los *colliders* o si el enemigo está activo o no. Es interesante mencionar que se ha utilizado un enumerado para definir el tipo del enemigo que puede ser *Soldado*, *Canonero* o *Ariete*, de esta manera es más sencillo seleccionar en el

²⁹<https://docs.unity3d.com/ScriptReference/SerializeField.html>

editor de *Unity* el tipo de enemigo del que se trata.

También el *GameManager* hace uso de la estructura *DatosMinijuego* para que al llegar al estado de *Termina()* se pueda hacer una comprobación si el jugador ha ganado o han sido los enemigos, dependiendo de esto creará un nuevo *DatosMinijuego* con la puntuación actual obteniéndola de la clase *puntuacionManager* y si ha ganado el jugador, marcará el minijuego como completado, posteriormente se almacenará usando el *ControladorDatos*.

Es interesante mencionar que hay tres campos de tipo *delegateGameManager* que actuarán como eventos para conocer cuándo se comienza el juego, cuándo termina y si se ha ganado, estos eventos actuarán en otros componentes haciendo unos comportamientos relativos a estos estados.

5.4.2 *ImageTarget* de Minijuego

El minijuego se activa una vez que el jugador encuentra el marcador con la cámara, el marcador se trata de un *ImageTarget* que son imágenes que el motor de *Vuforia* es capaz de reconocer, para su uso simplemente se agrega un *GameObject* en la escena de *Unity* de tipo *ImageTarget* que contiene los componentes necesarios para su definir primero el tipo de *Target* del que se trata, la base de datos de la que *Vuforia* extrae la imagen y la imagen en sí. El modelo que genera es un plano con un material cuya textura es la imagen con la que se carga, esto ayuda para visualizarlo dentro del editor. También se incluye un *script* que implementa una interfaz llamada *ITrackableEventHandler*³⁰, que provee las funciones para conocer el estado del marcador, si la cámara ha comenzado a buscarlo, lo ha encontrado o lo ha perdido. Para personalizar el uso y la interacción con el reconocimiento de las marcas,

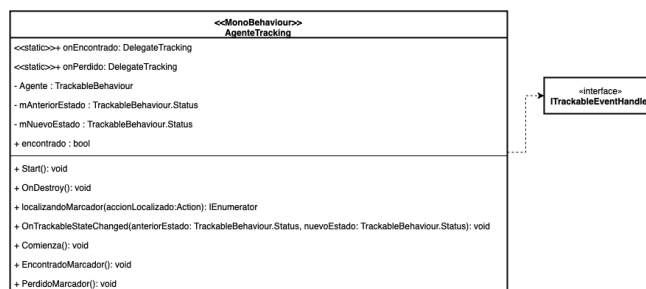


Figura 5.18: Diagrama de clase del *AgenteTracking*

se ha creado un *script* llamado *AgenteTracking*, en la figura 5.18 se pueden ver los métodos y campos que utiliza además de los implementados por la interfaz *ITrackableEventHandler*. Es curioso señalar el método de *localizandoMarcador()* que permite realizar una llamada síncrona que se desbloquea cuando la cámara encuentra el marcador. Gracias a este método se puede iniciar el minijuego y su contenido cuando se encuentra por primera vez. Los

³⁰https://library.vuforia.com/content/vuforia-library/en/reference/unity/interfaceVuforia_1_ITrackableEventHandler.html

métodos auxiliares de *EncontradoMarcador()* y *PerdidoMarcador()* se utilizan para activar o desactivar los componentes de colisión y modelos 3D respectivamente.

De esta manera se utiliza un *ImageTarget* en el que se contienen algunos de los elementos principales del minijuego, se debe hacer de esta manera ya que al utilizar los *ImageTarget*, todo el contenido aumentado que se quiera visualizar es necesario que sea hijo de este *GameObject*, así el contenido copiará la posición relativa del marcador en el mundo real.

En la figura 5.19 se observan los componentes contenidos en el *ImageTarget*. Este *GameObject* sigue esta estructura debido a que son los componentes que deben aparecer contenidos en el marcador una vez que este se encuentre. Se procederá a explicar cada uno:

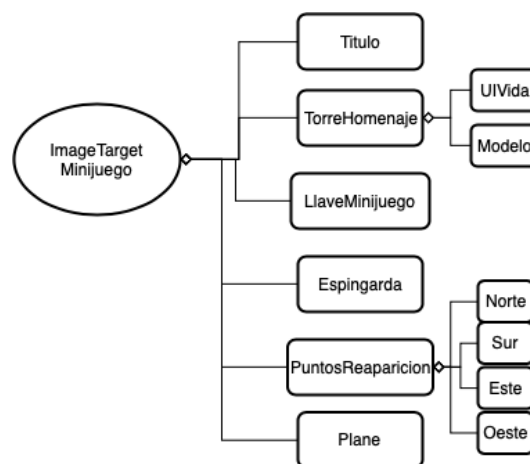


Figura 5.19: Diagrama de componentes que forman el *Image Target* del minijuego

- **Titulo** Se trata de una interfaz que se posiciona en el mundo real y contiene tres elementos, el titulo del minijuego, un botón que desencadena el comienzo de las rondas en el *GameManager* y un botón para regresar al menú principal.
- **TorreHomenaje** Es una torre que se sitúa en el centro del marcador, tiene dos componentes dentro de él, uno es la interfaz de su salud *SaludVida*, se trata de un componente que se encarga de registrar la salud de la torre, este componente se detallará en el punto 5.4.5 y el modelo, el cual es un modelo libre extraído de la página *TurbosQuid*³¹ y editado en *Blender* para su uso en el proyecto.
- **LlaveMinijuego** Se trata de la llave que se obtiene al ganar el minijuego pues está registrado al evento *OnGanaJuego* y se activa en lo alto de la torre, contiene un *script* sencillo que cuando el jugador pulsa en la pantalla sobre la llave (*OnMouseDown()*) se suma al número de llaves del jugador una llave.
- **Espingarda** Es el arma principal del jugador, aparece junto a la torre para que sea recogida una vez comienza el minijuego, este componente se detallará en el punto 5.4.7.

³¹<https://www.turbosquid.com/3d-models/3d-blender-tower-medieval-1336111>

- **PuntosReaparicion** Es una serie de *Transform*³² que se colocan en los puntos cardinales de la torre, sirven de apoyo para poder colocar a los enemigos de una manera más visual.
- **Plane** Este es un plano que se coloca a la misma altura que el marcador, contiene un material cuyo *Shader* que permite reflejar sombras de los objetos que están por encima de él. Este *Shader* se incluye en las últimas versiones de *Unity* junto a *Vuforia* y se llama *ShadowAR*.

5.4.3 Sistema de Puntuación

Para permitir que el videojuego tenga un componente de competición, es interesante que de alguna manera se pueda maximizar el juego y para ello se ha llevado a cabo un pequeño sistema de puntuación que permita ser sencillo de utilizar y cómodo para consultar.

Se ha realizado una clase que se muestra en la figura 5.20 que contiene además del contador de puntos una funcionalidad que muestra el número de puntos en forma de un texto animado. En este caso, para sumar puntos es necesario eliminar a un enemigo, por lo que el sistema de puntuación está suscrito al evento *OnEnemigoMuerto* que se detalla en el punto 5.4.6, básicamente cada vez que se elimina un enemigo se llama a *EnemigoEliminado* indicando el tipo de enemigo y su posición. Con esto, el sistema suma puntos dependiendo del tipo de enemigo, si es Soldado 5, Cañonero 10, y Ariete 20, posteriormente muestra un texto animado con los puntos que se han sumado que se coloca en la posición donde el enemigo ha sido eliminado. Hay que indicar que esta clase se trata de un *Singleton* para evitar que puedan existir errores en cuanto a la consulta de puntos.

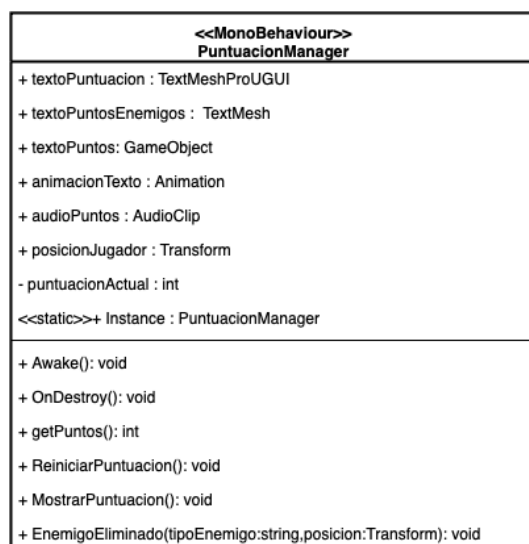


Figura 5.20: Diagrama de la clase de *PuntuacionManager*

El diseño de esta clase se justifica en que hay algunos componentes que tienen que ver con

³²<https://docs.unity3d.com/ScriptReference/Transform.html>

la presentación al jugador de la obtención de puntos. Esta realizándose con la animación y el texto que son campos que podrían haberse incluido en un *script* aparte encargado de esta gestión, pero se ha preferido simplificar este sistema al tratarse de un sólo minijuego.

5.4.4 Sistema de Pool de Objetos

Este videojuego hace uso de objetos que continuamente aparecen y desaparecen como son los proyectiles que utilizan los enemigos, las balas de la *espingarda* o los propios enemigos cuando se activan. Normalmente para generar un objeto lo que se hace es ejecutar el método *Instantiate()* al que se le pasa un *GameObject* y una serie de valores como la posición, rotación, su padre, etc. Pero esto obliga a que se reserve un nuevo espacio de memoria para este objeto y libera otro espacio de memoria, el hecho de realizar esta acción desencadena que se genere lo que se conoce como basura en el *GarbageCollector*, el cual es un sistema que se encarga de gestionar la memoria automáticamente pero si no se usa adecuadamente puede resultar en un rendimiento pésimo. Además teniendo en cuenta que este es un videojuego cuyo objetivo es lanzarse en una plataforma móvil, y se conoce que los teléfonos móviles hoy en día no disponen de mucha memoria por lo que es necesario diseñar un sistema que evite este problema.

Para ello, se ha diseñado un sistema de piscinas de objetos, cuyo objetivo es instanciar nada más comenzar la ejecución los *GameObject* que se deseen en cadena, aunque teniendo en cuenta que los objetos pueden variar desde una bala hasta un enemigo ha sido necesario realizar un sistema generalizado que resulte sencillo de manejar.

En el diagrama de la figura 5.21 se ve parte del sistema que se utiliza para los elementos que cumplen más o menos la misma funcionalidad, servir como proyectiles, esta clase resulta muy útil en el editor de *Unity* ya que se trata de una clase abstracta que contiene un campo que se rellena desde el editor con un *GameObject*, con este objeto crea una cola de instancias de ese objeto y también proporciona los métodos necesarios para recuperar ese *GameObject* o devolverlo a la piscina. En este caso se utilizan tres tipos de clases *GameObjectPool*, todas están contenidas en un objeto que actúa a modo de proyectil que se mueve en una dirección un determinado tiempo, al llegar a su tiempo límite regresan a la piscina, tiene un comportamiento sencillo por lo que era muy adecuado usar este sistema.

El diseño de esta clase se puede justificar en la necesidad de tener un *script* que pudiera reutilizarse simplemente cambiando el componente *T* que equivale a un *GameObject*. Así para hacer otro *pool de objetos* sencillo, sólo es necesario crear un *Script* con el estereotipo de *GameObjectPool*. Así se permite hacer un sistema de *Pool* muy manejable.

Sin embargo para los enemigos este sistema se quedaba algo corto ya que no ofrecía la posibilidad de generar un número determinado de un mismo *GameObject* y para ello ha sido necesario crear otro sistema aparte llamado *ManagerEnemigos* cuyo diseño se refleja en la figura 5.22

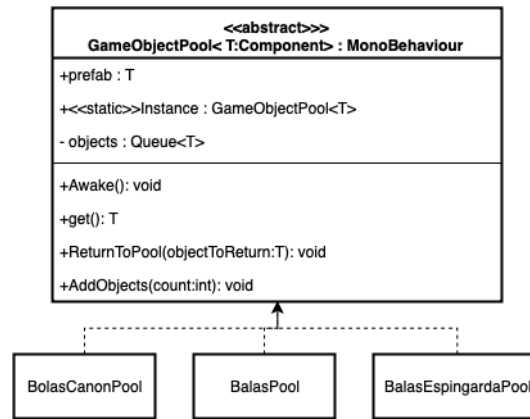


Figura 5.21: Diagrama de clases de la piscina de GameObjects

Esta clase cuenta con un Diccionario en el que se incluye un *string* a modo de clave y una cola de *GameObject*s para su valor y una lista de *PoolEnemigos* que sirve para asignar un *tag* a un *GameObject* y establecer un tamaño. Este sistema es así para facilitar la generalización de una piscina que contenga un número de un cierto tipo de *GameObject*s y permitir activarlos dependiendo de su *tag*, además es así más sencillo rellenar la información desde el editor de *Unity*.

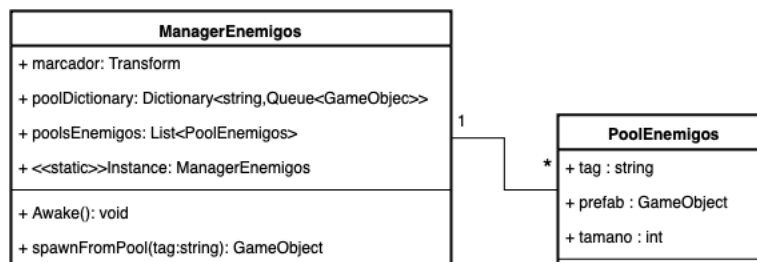


Figura 5.22: Diagrama de clases del ManagerEnemigos

Es interesante el uso de este tipo de sistemas y patrones de diseño para permitir alcanzar una de las normas de calidad en desarrollo móvil como es la *Eficiencia*, que es el correcto funcionamiento de una aplicación tanto en respuesta de la interacción del usuario como en el funcionamiento. Al alojarse la memoria permite que se sature menos y así lograr que no haya bajadas repentinas de *frames*.

5.4.5 Sistema de salud

Para permitir que se pueda realizar daño sobre algún objeto y que se pueda reflejar de una manera que el jugador entienda cuánto daño ha realizado y qué proporción se ha desarrollado un sistema cuyos componentes se muestran en la figura 5.23 que consta de dos partes, por un lado el ReceptorDano, que consta de un *script* que se encarga de la lógica de la recepción de daño que se activa cada vez que un objeto colisiona con su *Collider* y la gestión de los puntos de vida, y por otro lado la representación de los puntos de vida, que es un *Slider* (*BarraVida*) que se actualiza cada vez que el objeto recibe daño siguiendo un código de colores. Verde

para el 100 % de la vida, Amarillo para el 50 % y Rojo para menos de un 25 %.

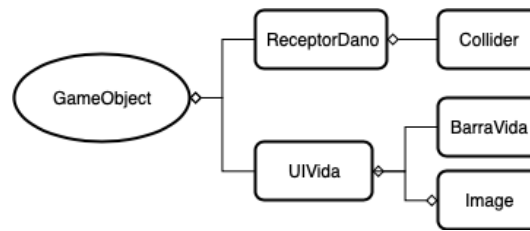


Figura 5.23: Diagrama de componentes del sistema de salud

Es interesante que se pueda extraer este sistema y utilizarlo para diferentes objetos, por lo que el diseño de las clases que se muestra en la figura 5.24 permite realizar esto, teniendo el estereotipo de *abstract* en *ReceptorDano* y así poder extraer clases con los campos y métodos que estas compartirían.

De base se tiene una clase abstracta *ReceptorDano* con tres eventos *onDanoRecibido*, *onMorir* y *onResetVida*, estos eventos sirven para activarse dependiendo de qué interactúa con el objeto o colisionador.

En el caso del método *RecibirDano()* lo que sirve es para restar los puntos de vida que tiene el objeto, a este método se le llama desde fuera, ya sea un proyectil o el componente que sea, esto permite desacoplar el comportamiento de los proyectiles de los receptores de daño, y así se puede recibir el daño independientemente de qué interactúe con el objeto.

Al recibir daño se lanzan los eventos de *onDanoRecibido* y *onMorir* en el caso de que los puntos de vida lleguen a cero, estos eventos sirven para que en la parte de la interfaz se actualice con el porcentaje de vida que tiene el objeto justo en el momento en que recibe daño.

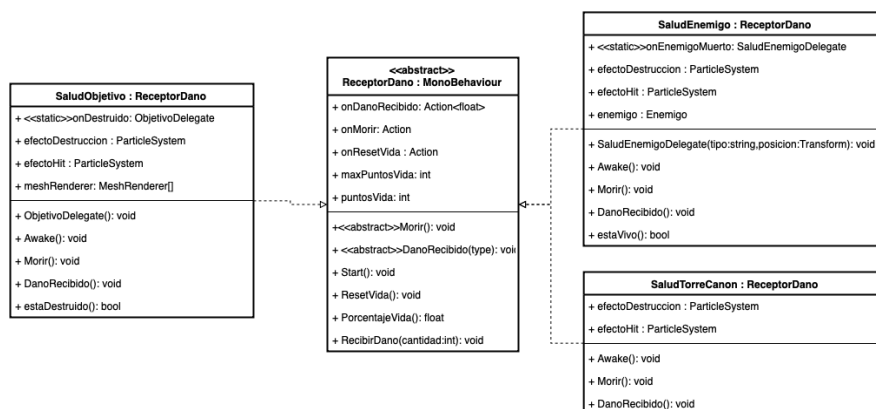


Figura 5.24: Diagrama de clases del sistema de salud

Por otro lado se han creado tres clases con el estereotipo de *ReceptorDano* para facilitar el uso de este *script* y permitir usarlo independientemente del tipo de objeto que sea, particularizando en cada caso lo que se desee que ocurra cuando el objeto reciba daño, por

ejemplo con un efecto de partículas (*ParticleSystem*) diferente para cada objeto y en el caso de *SaludObjetivo* y *SaludEnemigo* con eventos que permiten conocer cuándo la torre ha sido destruida o cuándo un enemigo ha sido eliminado respectivamente.

5.4.6 Enemigos

Los enemigos es una parte fundamental de este minijuego y desde el principio del desarrollo se han refinado y optimizado para que estos pudieran encajar con el resto de sistemas y actualizarse utilizando modelos nuevos, animaciones, sonidos, etc.

Para abordar todo lo que componen a los enemigos primero se va a analizar los componentes que tiene un *GameObject* de tipo *Enemigo* como se muestra en la figura 5.25, se divide en cuatro componentes, cada uno está dedicado para aislar diferentes comportamientos o separar la representación de la lógica, como es el caso del *ModeloEnemigo* y el *Arma*, aunque en la práctica el arma debe estar colocada entre las manos del enemigo. A continuación se pasarán a comentar los diferentes componentes:

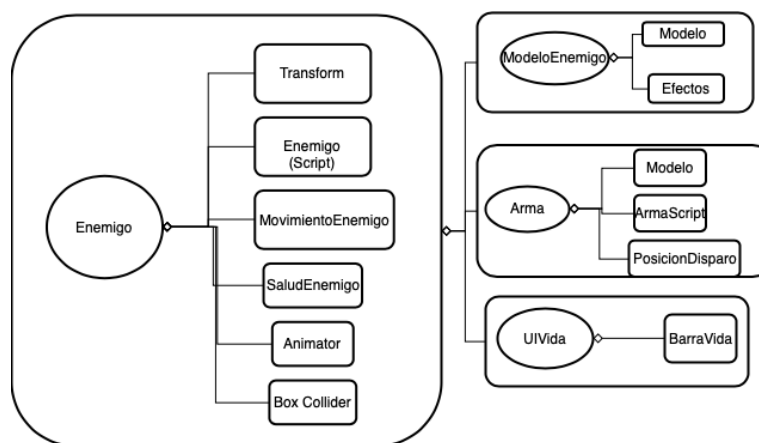


Figura 5.25: Diagrama de componentes del *GameObject de Enemigo*

- **Enemigo** Este es el componente padre que reúne toda la lógica del *Enemigo*, contiene el *Transform* que actúa como la posición del enemigo, el script del enemigo, el cual se encarga de cargar todos los componentes y actuar como controlador principal de ellos, aparte de proveer las funciones necesarias para los estados del enemigo.

Por otro lado contiene el movimiento del enemigo que en cada tipo de enemigo se especifican unos valores de velocidad máxima y mínima, la salud del enemigo que dependiendo del tipo de enemigo se establece un máximo de puntos de vida. El *Animator* que es el componente encargado de la máquina de estados del enemigo y por último el *BoxCollider* que será el responsable de recibir los proyectiles.

- **ModeloEnemigo** Ese componente resulta más específico para cada enemigo, los efectos son sistemas de partículas que se activan cuando el enemigo recibe daño o muere, y el modelo excluyendo el Soldado, el cual se ha extraído de la página web *mixa-*

*mo.com*³³ que provee modelos con esqueletos humanos animados, se ha elegido así para evitar realizar animaciones complejas y tener que recrear un modelo desde cero. Por otro lado el modelo del cañonero, el cual se ha extraído del *AssetStore*³⁴ y el modelo del *Ariete* se ha extraído de la página *SketchFab*³⁵. Se ha requerido hacer una búsqueda por diferentes páginas que ofrecieran modelos gratuitos para que la estética se adecuara correctamente con la temática del videojuego.

- **Arma** El arma depende de cada enemigo, pero todos comparten un *script* llamado *ArmaScript* que se muestra en la figura 5.26. Es una clase abstracta que contiene los campos necesarios para colocar el arma en una posición y proveer las funciones que permitan utilizarla, además de agregar efectos de partículas y de sonido. El Rifle y el Cañón ambos utilizan además una clase *Proyectil* que es contenida en un objeto y actúa a modo de proyectil dependiendo del tipo, si es una bola de cañón toma una trayectoria diferente a la de una bala. Además estas armas hacen uso del sistema de piscinas para instanciar los proyectiles. Los modelos de las armas del cañón y la cabeza del ariete vienen junto a los modelos del propio enemigo mientras que el del rifle proviene de un paquete del *AssetStore*³⁶. El objetivo de este diseño es proveer los campos necesarios

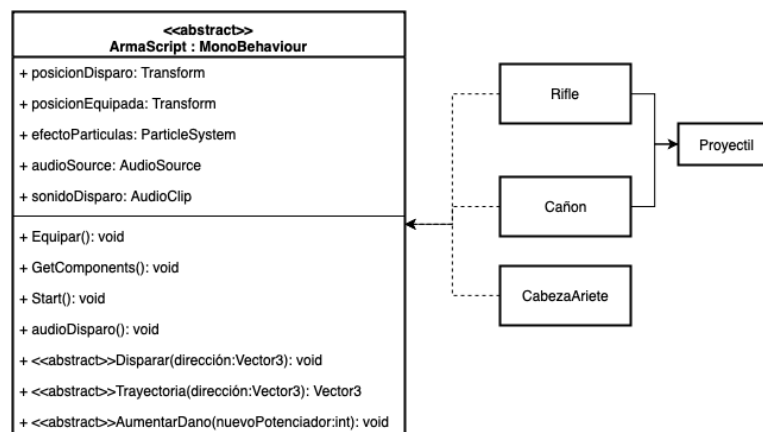


Figura 5.26: Diagrama de clase de *ArmaScript*

que serán utilizados en todas las armas como pueden ser los efectos de partículas o un sonido cuando se vayan a utilizar, por lo que resulta fundamental hacer que la clase contenga el estereotipo *abstract*.

- **UIVida** Este es el *prefab* que se utiliza tanto para los enemigos como para la torre y se sitúa justo encima del enemigo para mostrar su porcentaje de vida actual.

En cuanto a los estados del enemigo, todos tienen los mismos estados y actúan de la misma manera, la transición entre estos estados se puede observar en la figura 5.27, comienzan

³³<https://www.mixamo.com/>

³⁴<https://assetstore.unity.com/packages/3d/olde-props-pack-1-pbr-legacy-32221>

³⁵<https://skfb.ly/6HZzn>

³⁶<https://assetstore.unity.com/packages/3d/olde-props-pack-1-pbr-legacy-32221>

en el estado *Idle* en el que se encuentran estáticos durante unos segundos, esto sirve para que el jugador vea que los enemigos han reaparecido y pueda detectar su posición, después pasan al estado de *Moviéndose* en el que activan el componente de *MovimientoEnemigo* y se mueven en la dirección del objetivo que tienen, este objetivo se trata de un *Transform* que permite recoger su posición y en este estado se calcula la distancia a la que se encuentra el enemigo del objetivo, cuando llega a una distancia dada (*DistanciaAgro*) pasa al estado *Atacando*, cada enemigo tiene una distancia asociada, los cañoneros son los que tienen la mayor distancia, los soldados una distancia intermedia y los arietes son los que más se deben acercar al objetivo. Cuando se encuentran en el estado *Atacando* lo que hacen es utilizar el *Arma* a su disposición, y disparar cada cierto tiempo, cada uno también tiene un tiempo de ataque, los soldados son los más rápidos disparando, los cañoneros tardan el doble y los arietes son los que más tardan debido a su daño.

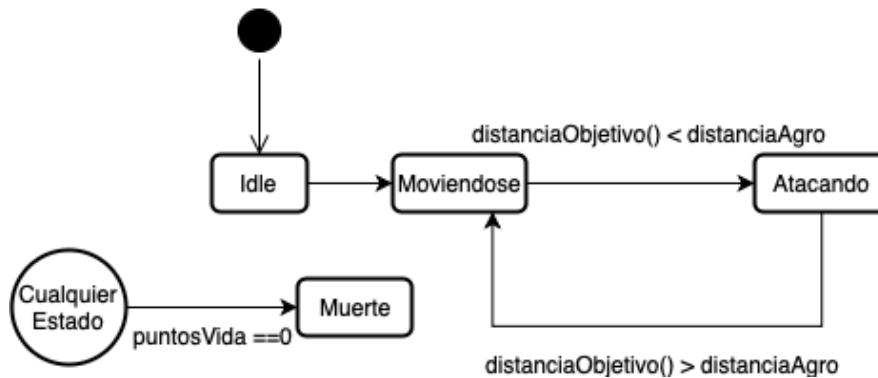


Figura 5.27: Máquina de estados del enemigo

5.4.7 Jugador

El jugador en el minijuego se representa con la cámara encargada de localizar los marcadores, por lo que entre los componentes de ésta se debe incluir alguno que sea capaz de reconocer estos. En la figura 5.28 se observan los componentes relacionados con el jugador entre los que se ve un componente llamado *VuforiaBehaviour*, este es un script que se incluye en *Vuforia* y permite configurar la cámara con la que se detectan los marcadores. Actualmente está configurada para que ésta permita detectar al menos dos marcadores simultáneamente y el centro del mundo estableciéndolo en el propio terminal.

El diseño de este sistema para el jugador se realiza así para simplificar la cámara en un sólo *GameObject*, de tal manera que tenga la cámara propia de *Unity*, adherida a las funcionalidades de realidad aumentada de *Vuforia* y con un *script* para modificar las opciones de la cámara.

Por otro lado están los componentes de *OpcionesCamara* el cual es un pequeño *script* que permite configurar cómo se enfoca la cámara, esto es útil en el caso que se desee tener diferentes tipos de marcadores o situados a diferentes distancias. También se encuentra el

controlador de llaves, que permite que al recoger la llave en el minijuego, ésta quede registrada junto a los datos del jugador. Por último un *AudioListener* para que todos los sonidos que provengan del juego sean escuchados por el jugador.

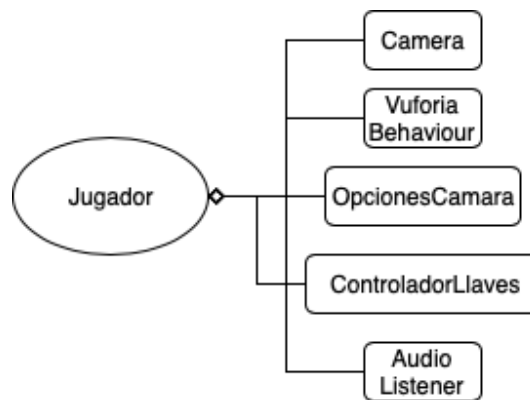


Figura 5.28: Diagrama de los componentes del jugador

En este punto también se va a comentar los objetos que puede utilizar el jugador y como se comentaba anteriormente se tratan de dos:

- **Espingarda** Este se trata de un fusil que se utilizaba en el siglo XV, para su recreación se ha seguido la estructura de componentes que se muestra en la figura 5.29, está jerarquizado de tal manera que en el componente padre tiene toda la lógica de cómo se comporta el objeto, contiene un colisionador y un componente que le aplica físicas para que sea un objeto sólido, además de un *Animation* que contiene dos animaciones, una de disparo y otra de recarga, también un *AudioSource* que contiene dos clips de audio, de disparo y de recarga, ambos clips de audio se han extraído de la página opengameart.com.

Contiene un *script* llamado *Arma* que es muy parecido al que utilizan los enemigos, solo que éste al tratarse de un objeto que utiliza el jugador necesita unos cambios como por ejemplo encargarse de la gestión de la interfaz o la interacción del jugador, por último en esta parte contiene un componente llamado *Outline* que sirve para dibujar una silueta alrededor del arma para llamar la atención del jugador.

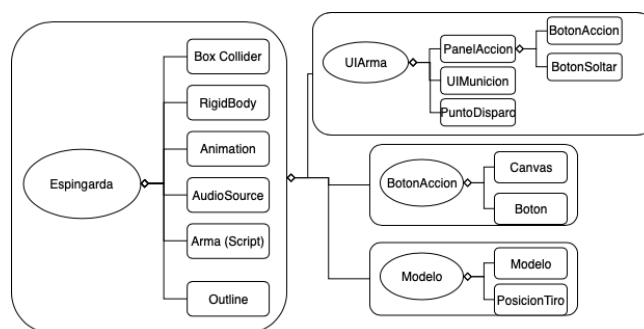


Figura 5.29: Diagrama de los componentes de la *Espingarda*

Por otro lado se encuentra el componente de *UIArma*, el cual a su vez está compuesto de varios paneles que sirven a modo de representación de la información del arma y la interacción, como el panel acción que contiene dos botones, uno que sirve para disparar y recargar (*BotonInteracción*) y otro que sirve para soltar el arma, también se encuentra *UIMunición* que sirve para mostrar la munición restante del arma. El *BotonAcción* sirve para recoger el arma, aunque se trata de un *Prefab* que se encarga de posicionarse en el mundo y mirar constantemente en la dirección donde se encuentra el jugador, de esta manera es más sencillo interactuar con él, por último se encuentra el modelo que se ha extraído del *AssetStore*, el cual venía en el mismo paquete que el cañón.

- **Torre con Cañón** La torre con cañón es un objeto un poco más complejo ya que hace uso también de los *ImageTarget* pero se puede estructurar como se muestra en la figura 5.30, para explicar cómo funciona primero se detallarán los componentes y después se indicará la máquina de estados que tiene la *Torre con cañón*:

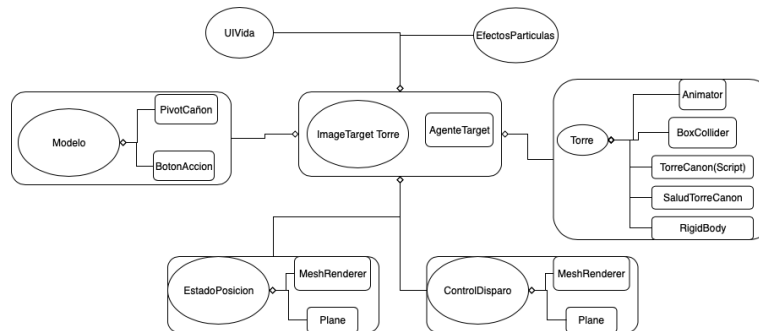


Figura 5.30: Diagrama de los componentes de la *Torre con Cañón*

- *ImageTarget Torre* El componente padre se trata de un *ImageTarget* que hace uso de un *script* personalizado llamado *AgenteTarget*, este *script* implementa la interfaz de *ITrackableEventHandler* al igual que la clase *AgenteTracking*, sólo que en los eventos en los que se encuentra el marcador se hace uso de un campo que ofrece Unity llamado *UnityEvent*³⁷ el cual permite asignar *listeners* cuando ocurren ciertos eventos. De esta manera es muy sencillo asignar comportamientos como los botones pero en este caso cuando comienza la búsqueda, se encuentra o se pierde el marcador.
- *Torre* En este componente se contiene toda la lógica del objeto, asignándose un *Animator* que servirá de máquina de estados, un *Box Collider* para registrar cuándo los enemigos disparan a la torre. El *script* de *TorreCanon* que se encarga de cargar los componentes y proveer las funcionalidades necesarias para utilizar la Torre, *SaludTorreCanon* que sirve para registrar los puntos de vida de la misma y un *RigidBody* para que el objeto sea sólido.

³⁷<https://docs.unity3d.com/ScriptReference/Events.UnityEvent.html>

- *ControlDisparo* Este es un plano que tiene un material con una textura con forma de diana, este sirve de referencia para conocer dónde está apuntando el cañón.
- *EstadoPosicion* Este plano sirve para tomar referencia del estado de la torre, si está construida, destruida o recargando.
- *Modelo* Este está formado por dos componentes, aunque contiene también el modelo de la propia Torre, el cual está extraído del *AssetStore*³⁸ de *Unity*. Por un lado está el pivot del cañón que sirve para poder rotar el cañón en la dirección que se desee y por otro el *BotonAccion* que sirve para utilizar la torre, ya sea para construirla o para disparar.

La torre con cañón sigue una serie de estados para dar respuesta a la interacción del jugador, las transiciones de estos estados se pueden ver en la figura 5.31, comienza en el estado *Sin Construir*, este estado sirve para mostrar la Torre con cañón e indicar el estado si se puede colocar o no, a su alrededor está comprobando si hay *Colliders* con los que colisione en un radio de acción. Esto se realiza así puesto que el jugador es libre de colocar el marcador donde quiera y para evitar que se comenten errores se hace esta comprobación. Si el jugador pulsa el botón de acción pasará al estado de *Construcción*, el cual es un estado de espera para evitar que la torre pase a disparar automáticamente, tras unos segundos se encuentra en el estado de *Preparada* el cual busca a su alrededor enemigos, y si los encuentra, coloca el *ControlDisparo* en su posición y activa el botón de acción para que se pueda disparar. Una vez que el jugador dispare se activa un *Trigger* en el *Animator* que obliga a pasar al estado *Recargando*, este es otro estado de espera en el que el botón de acción se deshabilita durante unos segundos.

En cualquier momento los enemigos pueden derribar la torre por lo que desde cualquier estado puede pasar al estado *Destruído* una vez que sus puntos de vida sean cero, y aquí la torre estará esperando unos segundos para poder pasar de nuevo al estado *Sin Construir*.

Este objeto resulta muy útil para despistar a los Soldados, ya que son los únicos que en su código de *Enemigo* están suscritos a un evento *onTorreConstruida* que se lanza cuando la torre termina de construirse. De esta manera los Soldados cambian su objetivo y lo actualizan hasta que la Torre pase al estado *Destruída*, además de proveer una potencia de fuego superior respecto a la *Espingarda*.

Con esto el jugador tiene la opción de poder defender la *Torre del Homenaje* utilizando ambos objetos, se pueden utilizar a la vez o utilizar uno en cada momento, sin embargo es necesario ambos para conseguir superar todas las oleadas.

³⁸<https://assetstore.unity.com/packages/3d/environments/human-cannon-145437>

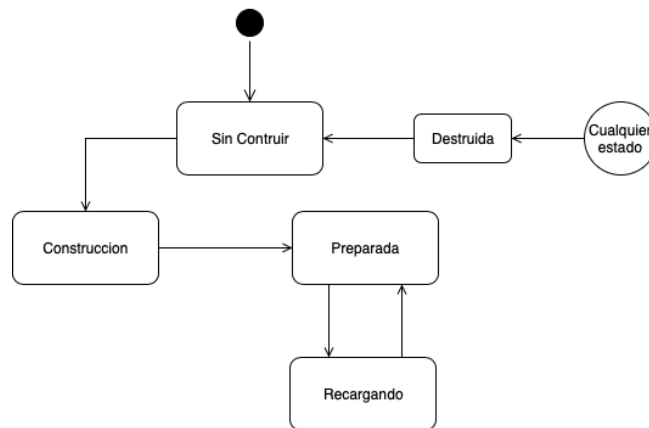


Figura 5.31: Diagrama de los estados de la *Torre con Cañón*

5.5 Patrones de diseño utilizados

En esta sección se exploran los diferentes patrones de diseños utilizados en la arquitectura de este trabajo. Para el aprendizaje y el uso de estos se ha seguido los ejemplos expuestos en [20], que permiten ver estos patrones aplicados en el contexto de un videojuego, por lo que resultaba más fácil su comprensión.

5.5.1 Modelo-Vista-Controlador

Es un patrón cuyo objetivo es separar la lógica de negocio de la interfaz de usuario, lo que permite realizar un desarrollo más desacoplado y coherente. Este patrón se ha procurado seguir desde el principio junto a otros patrones que juntos permiten crear una solución modular a lo que se pretende con este trabajo, a la vez que hacerlo escalable y mantenible.

Un ejemplo del uso de este patrón se ve en el punto 5.3 en los componentes que se utilizan en el menú principal, los cuales están separados en la distribución que indica este patrón.

5.5.2 Observador

Este patrón es la clave del patrón de arquitectura *Modelo-Vista-Controlador*, cuyo objetivo es el de establecer una dependencia de uno a muchos entre objetos, de tal forma que existe un objeto *Observado* y otros *Observadores* que son notificados automáticamente cuando el *Observado* cambia de estado.

Este patrón es muy recurrente en el desarrollo de videojuegos ya que permite desacoplar la clase de los objetos clientes del objeto, aumentando la modularidad de los sistemas que se desarrollan.

Se ha necesitado utilizar este patrón para simplificar algunos sistemas como el de salud en el punto 5.4.5, se especifica cómo se utiliza un sistema de eventos para notificar cuándo el objeto recibe daño, muere, o sus puntos de vida se reinician.

También se ha utilizado en el caso de las armas, o el propio *GameManager* del minijuego

en el punto 5.4.1 .

5.5.3 Componente

Este patrón consiste en permitir que una sola entidad abarque múltiples dominios sin acoplar los dominios entre sí. *Unity* está basado completamente la filosofía que sigue este patrón, por lo que no utilizar este patrón sería desarrollar en dirección contraria a como se pretende hacerlo en *Unity*.

A lo largo de todo este capítulo se han hecho referencias a diagramas de componentes que permiten ver de manera sencilla y general qué engloba un *GameObject* en específico simplemente viendo los componentes que tiene. El uso de este patrón se refleja bien en el punto 5.4.6 donde se aprecia cómo se separa los diferentes elementos que componen a un Enemigo, por un lado el modelo, su arma, la representación de su vida y la lógica que lo dirige.

- **Singleton** Este patrón se asegura de que sólo exista una instancia de una clase específica y provee un punto global de acceso a ella. Esto se cumple en varios casos pero el mejor ejemplo de ellos es en el 5.4.3.

Aunque en *Unity* al permitir realizar un videojuego en varias escenas, es posible que en todas haya que declarar una instancia de un mismo objeto, es decir, que el *GameObject* de un objeto que use *Singleton* sea necesario repetirlo en cada escena para que sí que exista una instancia en cada escena o por otro lado no permitir que el objeto se destruya al cambiar de escena. Pero esto obliga a que a la hora de recrear nuevas escenas sea necesario un orden de ejecución en las mismas.

- **Estado** Este patrón permite a un objeto alterar su comportamiento cuando su estado interno cambio. Este patrón se ha aplicado en muchas ocasiones a lo largo de este capítulo al haber utilizado máquinas de estado finitas en algunos de los comportamientos como por ejemplo el de enemigos 5.27.

El comportamiento de los enemigos se basa en la distancia a la que se encuentran del objetivo, este cálculo se realiza cada cierto tiempo y cuando se encuentran a una distancia específica del objetivo su estado cambiará en consecuencia.

- **Object Pool** El objetivo de este patrón es el de mejorar el rendimiento y la memoria reutilizando objetos de un *pool* en vez de alojándolos y liberándolos de la memoria individualmente.

Este patrón es clave para permitir que la jugabilidad en un dispositivo móvil sea agradable cuando se tratan muchos objetos que además utilizan sistemas de físicas, por lo que ha sido una obligación su uso. Para este trabajo se han realizado varios sistemas de *Pool* que permitan utilizarlos de manera sencilla en *Unity* y se explican en el punto 5.4.4

Capítulo 6

Resultados

EN este capítulo se exponen los resultados del proceso de desarrollo de *Santiago de la Torre RA*, analizando las diferentes iteraciones en las que se ha enmarcado todo el proceso, analizando las tareas que se realizan, la duración de las etapas y mostrando los resultados que se obtuvieron en el tiempo. Posteriormente se comentan los puntos necesarios para que el videojuego en una versión final más completa se consiga implantar en el entorno real del castillo.

6.1 Proceso de desarrollo

El resultado de todo el desarrollo se puede obtener en forma de ejecutable (.apk) o se puede consultar en el repositorio de *GitHub* en el que se encuentran todos los *Assets* del proyecto en *Unity* incluidos los *scripts* utilizados <https://github.com/alejandroVeRod/TFGSantiagoRA>.

El desarrollo de *Santiago de la Torre RA* se dividió en varias iteraciones para regular así el desarrollo permitiendo incluir cambios y propuestas para una nueva iteración, en cada iteración se obtenían nuevas historias de usuario y tareas relacionadas con las propuestas y se procuraban completar todas. El objetivo de cada iteración es obtener un producto funcional más sofisticado y mejorado que en la iteración anterior.

Para obtener una visión general de cómo evolucionó temporalmente el proyecto en la figura 6.1 se muestran los componentes o sistemas que se desarrollaron a lo largo de cada iteración. Se ven incluidos en nuevas iteraciones aquellos que fueron refactorizados o a los que se añadieron nuevas funcionalidades en la siguiente iteración.

La duración del desarrollo del proyecto se ha visto muy acotada por lo que las iteraciones debían tener una duración estimada de 2 semanas, de esta manera se permitía avanzar sobre el desarrollo ofreciendo resultados lo suficientemente significativos como para señalar errores y mejoras, además de no tener un enfoque claro sobre cómo plantear el objetivo del minijuego y qué era lo correcto a la hora de conocer cómo el jugador exploraría el castillo.

6.1.1 Iteración 1: Prototipo

Ante la necesidad de encontrar algún minijuego que tuviera sentido dentro del contexto histórico del castillo pero a la vez tenga un componente de diversión, se optó por llevar a cabo

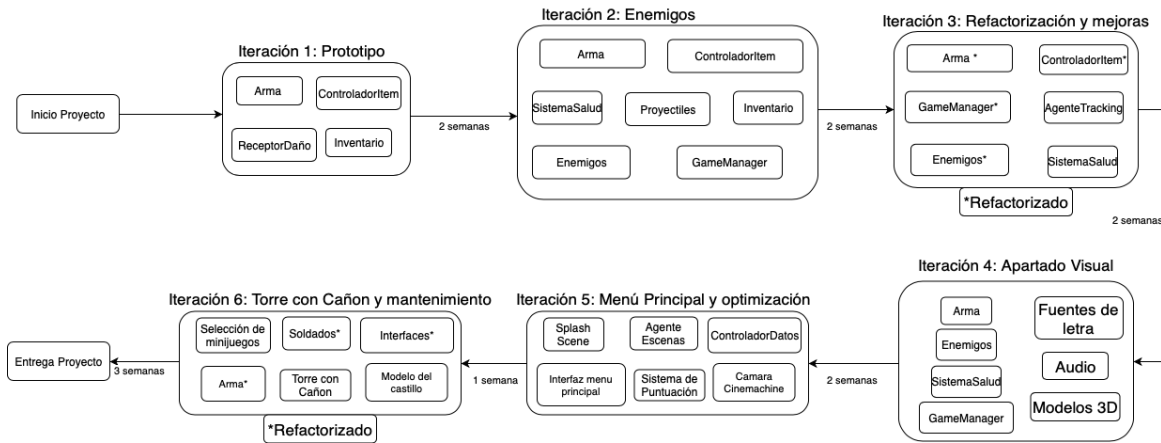


Figura 6.1: Evolución temporal del proyecto dividido en iteraciones

un minijuego sencillo que tuviera las mecánicas típicas de un *Shooter*, además encajaría con la manera en la que se mueve el jugador en un videojuego de realidad aumentada.

Por lo que primero se desarrolló un *prototipo* que tuviera una implementación sencilla que permitiera encontrar fallos de diseño o descubrir si realmente este tipo de minijuego se podría llevar a cabo pudiendo acercarse más a una estimación de lo que costaría el desarrollo, para esto sirven los prototipos.

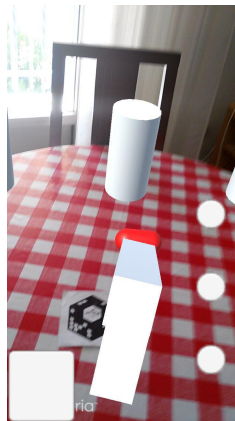


Figura 6.2: Captura del prototipo de minijuego *shooter*

En la figura 6.2 se ve una captura del prototipo, como se puede apreciar existen diferencias significativas entre este prototipo y la versión final, entre ellas se puede ver:

- Al igual que en la versión final, existe un arma que se acopla en la cámara aunque la manera que tiene de acoplarse es totalmente distinta, ya que se hacía mediante un *script* llamado *ControladorItem* el cual servía para manejar desde ese controlador todo lo que el jugador seleccionaba y colocaba en la cámara.
- Existe una interfaz que controla lo que el jugador puede colocarse aunque no se aprecia bien para qué sirve cada botón puesto que esta versión carecía de iconos y cualquier representación.

- Los enemigos carecen de un modelo y son completamente estáticos, solo tienen un contador para las balas que reciben y cuando llega al máximo, se destruyen, sin ningún tipo de control sobre estos.
- Las balas son instanciadas cada vez que el jugador pulsa sobre la pantalla y suelta el dedo, esto hacía que para apuntar se tapara con el dedo el objetivo al que se desea disparar y no quedaba bien a la hora de jugar, además del coste de rendimiento que suponía calcular constantemente la posición del dedo y las instancias de las balas.

Con este prototipo se podía entender en qué consistiría el minijuego pero aún faltaba mucho por delante para que sobre todo se viera y se pudiera jugar de forma agradable, los primeros fallos que se encontraron fueron en cuanto al rendimiento del juego ya que era pésimo, esto era debido a la falta de gestión de los recursos de memoria y además tener muchos componentes que realmente no tenían una función útil como detectar constantemente a qué seleccionaba el jugador desde un único componente que era la cámara.

El desarrollo de este prototipo duró dos semanas desde que se concibió la idea hasta que se desarrolló con un total de horas en la realización de tareas y completitud de historias de usuario de 45 horas.

6.1.2 Iteración 2 : Enemigos

El prototipo resultante de la anterior iteración resultaba útil para conocer cómo sería la principal mecánica del videojuego, que es disparar a los enemigos y ese es el primer objetivo que se consideraría para el minijuego, pero aún faltaban bastantes mecánicas que tener en cuenta, así como una condición de victoria y derrota

Una vez se adquiere las principales mecánicas de disparo y recepción de daño, se procede a animar a los enemigos para que suponga un desafío para el jugador dispararles, además de comenzar a incluir comportamientos sobre ellos, ampliar la interfaz y comenzar a establecer la lógica del minijuego, de esta manera se llevaron a cabo varias tareas que permitieran esto:

- **Realizar tipos de enemigos** Esta tarea consistía en la idea de establecer diferentes tipos de enemigos que tuvieran características que permitieran un equilibrio entre dificultad y visibilidad, también teniendo en cuenta que en un futuro debería animarse a estos enemigos y que sus modelos tuvieran sentido con el contexto del minijuego
- **Estados de enemigos** Los enemigos necesitarían un propósito por el que moverse y hacer algo, por lo que se les asignó tres estados básicos que al final representarían sus comportamientos, un estado en el que se mostraran quietos para acto seguido comenzar a moverse y luego un estado de ataque
- **Objetivos de enemigos** Necesitaban una dirección en la que moverse, principalmente se optó porque los enemigos persiguieran la cámara del jugador pero esto derivó en una

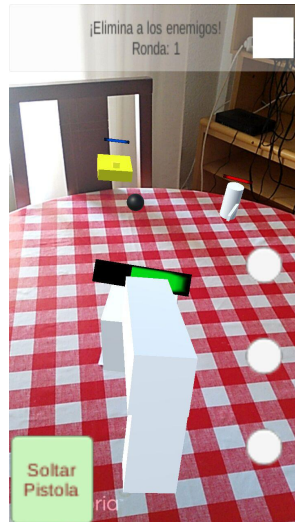


Figura 6.3: Captura del minijuego con diferentes tipos de enemigos

sensación de agobio a la hora de jugar ya que los enemigos perseguían infinitamente al jugador y este la primera reacción sería huir de ellos, por lo que se colocó un objetivo en el centro que permitiera centrar su atención y liberar al jugador para que su objetivo fuera únicamente el de eliminar los enemigos.

- **Sistema de Salud** Se amplió el sistema de recepción de balas colocando una pequeña barra de vida que permitiera conocer la vida de los enemigos y del objetivo céntrico, de esta manera también el jugador podría decidir mejor a qué atacar primero ya que los enemigos tienen diferentes puntos de vida máximo.
- **Proyectiles** Antes los enemigos y la pistola del jugador instanciaban una cápsula cuyo propósito era moverse en línea recta en la dirección que se le asignara, esto se cambió para utilizar un *prefab* que permitiera establecer diferentes tipos de trayectorias, así se pudo realizar las bolas de cañón para el cañonero.
- **GameManager** Se comenzó a construir un *GameManager* para establecer un sistema de rondas sencillo que siempre activara los enemigos en la misma localización y con un contador sumara las rondas.
- **Interfaz con información** También se comenzó a cuidar un poco más la interfaz indicando información acerca del botón para recoger y para soltar el objeto además de incluir un panel en la parte superior con el número de ronda en el que se encontraba.

Esta iteración resultó un salto considerable ya que se le comenzaba a dar forma al minijuego y también un sentido, el desarrollo duró dos semanas lo que parece poco tiempo para la cantidad de sistemas que se incluyeron aunque es necesario comentar que antes de comenzar a desarrollar este videojuego se realizó pruebas de otros videojuegos para familiarizarse con las diferentes herramientas por lo que se permitió que el proyecto escalara de una manera muy rápida y significativa.

6.1.3 Iteración 3: Refactorización y mejoras

Gracias a la iteración anterior se consiguió avanzar mucho en cuanto a elementos y sistemas con los que interaccionar pero también era necesario comprender bien qué sistemas se tenían y cómo estos podrían escalar en un futuro pues habría que procurar tener también algo jugable y que fuera agradable para ver.

Por lo tanto esta iteración consiste en la refactorización de algunos de los componentes utilizados anteriormente e inclusión de mejoras en algunos apartados del juego. Para ello se llevaron a cabo las siguientes tareas:

- **Control sobre el estado del marcador** En las etapas anteriores cada vez que se cargaba el minijuego, los enemigos ya comenzaban a disparar y a moverse, lo que provocaba una desventaja sobre el jugador, el problema venía dado por la falta de control sobre el marcador, por lo que para ello se creó el *script* de *AgenteTracking* con el que se puede controlar tanto de forma externa como interna el estado del marcador.
- **Integración de una interfaz para buscar el marcador** Uno de los principales problemas que se encontraba en el minijuego es que este no daba nada de información respecto a qué hacer antes de encontrar el marcador, esto podría generar confusión para las personas que no comprendieran bien cómo funciona la realidad aumentada por lo que se llevó a cabo la realización de una interfaz que se activara cada vez que el marcador se perdiera y se desactivara cuando este se encontrara.
- **Título del Minijuego** El minijuego comenzaría a cobrar sentido tanto temático como a nivel jugable, se le proporciona un nombre, *Defiende La Torre del Homenaje* el cual se integra dentro de lo que podría ser una localización del castillo, por lo que tiene sentido que sea un minijuego de defender la torre ya que también se trataba de un lugar de defensa estratégico.
- **Control sobre el inicio del minijuego** Antes el minijuego comenzaría justo cuando el jugador encontrara el marcador pero ahora se despliega una interfaz con el título del minijuego y un botón que activaría el minijuego como se ve en la figura 6.4.
- **Retirado el componente de inventario** Este componente servía para las pruebas que se realizaban antes de comenzar a plantear una idea de minijuego, pero se arrastraba este componente desde entonces y es uno que se utilizaba para poder guardar objetos y sacarlos, debido a la falta de tiempo y de recursos no se ha planteado la inclusión de muchos más objetos en el minijuego salvo la *espingarda* y por lo tanto no tenía mucho sentido seguir utilizándolo.
- **Refactorización del controlador de *item*** El controlador de *items* estaba totalmente acoplado al controlador del inventario y para permitir retirar el inventario fue necesaria una refactorización del mismo, permitiendo que se encargara este de la gestión de los objetos y la colocación de los mismos.



Figura 6.4: Captura de la pantalla de *Localiza el marcador*

- **Modelo de pistola** para comenzar a darle un toque más realista se decidió incluir un modelo de una pistola extraído del *AssetStore*, aunque este modelo no duró mucho ya que se trataba de un arma que no encajaba mucho con lo que se esperaría de este minijuego.

Esta iteración tuvo una duración de dos semanas y sirvió sobre todo para comenzar a pulir los sistemas que estaban en funcionamiento, darle un sentido al minijuego añadiéndole un título y una interfaz un poco más limpia.

6.1.4 Iteración 4: Apartado visual

Uno de los aspectos más importantes dentro de un videojuego es el apartado visual, esto incluye los modelos tridimensionales, las interfaces, fuentes de letra, efectos de partículas, sonidos, etc. Lo que más se echaba en falta en la iteración anterior era cuidar más este aspecto así que las tareas que se propusieron para esta iteración fueron:

- **Retirar controlador de *item*** Debido a la falta de objetos con los que interaccionar se decidió retirar el controlador de *item* y delegar la lógica de cuándo se selecciona un objeto y se coloca en los propios objetos, de esta manera se puede centrar más en el comportamiento individual de cada uno.
- **Añadir modelo de la *Espingarda*** Se cambió la pistola por un fusil y además uno que estuviera dentro del contexto histórico del castillo para incluir también algo de *gamificación* en el propio videojuego.
- **Modelos de enemigos** Para los enemigos se hizo una búsqueda de modelos que encajaran con la temática del castillo, esta era una tarea complicada ya que en el caso del soldado debía tratarse de un modelo que también tuviera un esqueleto animado, para ello se recurrió a la página *mixamo.com* y el resto de páginas web con modelos

gratuitos como *turbosquid.com* y *sketchfab.com*, el resultado es el de la figura 6.5



Figura 6.5: Captura de *Unity* de la carpeta de *prefabs Enemigos*

- **Modelo de la Torre** Antes se utilizaba un cubo que tenía una textura de piedra, pero para darle un toque más realista se utilizó un modelo de torre que además se pudiera parecer lo máximo posible a la *Torre del Homenaje*.
- **Efectos de Partículas** Cuando un enemigo recibía daño, el jugador solo podía conocer que este lo recibía viendo cómo disminuía su barra de vida, por lo que se consideró incluir un pack de efectos de partículas estándar¹ y aplicarlos a casi todos los objetos que interaccionan en el minijuego.
- **Sonidos** El minijuego carecía completamente de sonidos y aunque durante las pruebas en *Unity* no parecían necesarios, a la hora de probar el juego en el móvil se echaba en falta sonidos al disparar, explosiones, etc. Por lo que se descargaron varios sonidos de la página *opengameart.org*.
- **Fuente de letra** El tipo de letra estándar que ofrece *Unity* no parecía encajar mucho, así que se agregó dos nuevas fuentes de letras con tonos medievales descargadas de *dafont.com*

El resultado de esta iteración se puede ver en la figura 6.6, la duración de esta iteración fue de dos semanas que se involucraron la mayor parte del tiempo en encontrar modelos que encajaran y se ajustaran correctamente.

6.1.5 Iteración 5: Menú principal y optimización

La etapa anterior sirvió para comprender mejor cómo se comportarían los enemigos, qué dimensiones tendrían, cómo se verían, además de añadir otros elementos en el apartado visual, sin embargo esto generaba un problema y sería el peso del ejecutable *.apk* ya que al utilizar las texturas que venían por defecto en los modelos, esto aumentaba considerablemente el tamaño, para solucionarlo se redujeron la resolución de estas y se ajustó el tipo de compresión, de esta manera se pasaba de 126MB a 50MB, aunque aún sigue siendo un peso alto comparado con la media en el resto de aplicaciones, es un avance.

En cuanto a las tareas que se llevaron a cabo en esta iteración fueron las siguientes:

¹<https://assetstore.unity.com/packages/essentials/asset-packs/unity-particle-pack-5-x-73777>

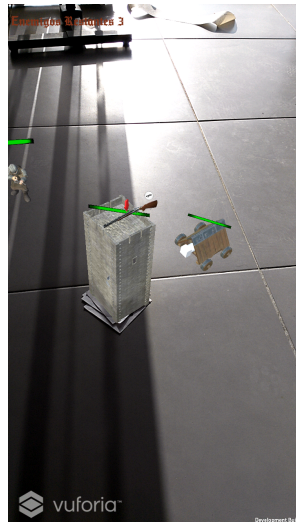


Figura 6.6: Captura del minijuego *Defiende la Torre del Homenaje* con modelos y nuevos elementos

- **Orientación apaisado** Se consideró que la jugabilidad en orientación retrato no era muy cómoda, por lo que se cambió todo el videojuego a esta orientación, para ello se tuvo que adaptar algunas de las interfaces que ya estaban hechas.
- **Splash Scene** Ya que el videojuego comenzaba a tomar forma, sería ideal comenzar a colocar la escena que muestra el logo de la empresa al iniciar el videojuego.
- **Menu Principal** El minijuego ya contenía condición de victoria y derrota, además de gráficos y sonido, por lo que sería ideal comenzar a diseñar un menú principal que sirviera de nexo entre los diferentes puntos del castillo y los minijuegos, primero se desarrolló una interfaz sencilla con dos menús, uno de opciones para ajustar el volumen y otro para mostrar la información del minijuego, en la figura 6.7



Figura 6.7: Captura de la primera versión del menú principal

- **AgenteEscenas** Ya que se esperaba que desde el menú principal se pudiera acceder a los diferentes minijuegos se realizó el objeto de *AgenteEscenas* que permitiera cargar cualquier escena y mostrar una pantalla con el porcentaje de carga.
- **ControladorDatos** Para poder almacenar la información del minijuego y los ajustes

se vio necesario usar un controlador que permitiera acceder y almacenar estos datos de una manera sencilla.

- **Sistema de puntuación** El minijuego simplemente consistía en eliminar a los enemigos sin ningún tipo de recompensa por ello, por eso se implementó un pequeño sistema que llevara un recuento de puntuación dependiendo del tipo de enemigo
- **Outline** Este es un componente que se obtuvo de la *AssetStore* que permite dibujar siluetas alrededor de un objeto, con esto se pretende llamar la atención al jugador sobre diferentes elementos en el juego, como la *Espingarda* o los enemigos cuando están atacando. En la figura 6.8 se ve cómo funciona este componente.

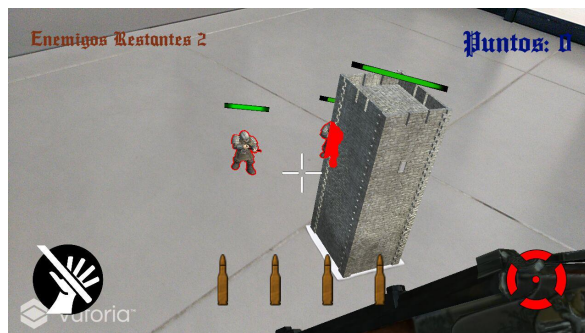


Figura 6.8: Captura del minijuego *Defiende la Torre del Homenaje* con *Outlines*

Esta iteración tuvo una duración de 1 semana ya que no se llevaron a cabo modificaciones sino que se añadieron diferentes sistemas y aún no se tenía claro cómo estos se comportarían en un futuro, aunque el videojuego ya comenzaba a cerrarse.

6.1.6 Iteración 6: Torre con cañón, selección de minijuegos y mantenimiento

La etapa anterior sirvió para establecer las bases de lo que sería el videojuego completo con un menú principal, un gestor de escenas, un sistema de almacenamiento, el sistema de puntuación, etc. Lo que permite que a esta iteración se le puedan añadir nuevos componentes y mejorar los que ya estaban, en la última reunión con el tutor del proyecto, se consideró apropiado agregar un nuevo objeto pero que utilizara marcadores para integrar un poco más la realidad aumentada, para ello se pensó en un marcador que tuviera una torre con un cañón para disparar a los enemigos.

Por lo tanto para esta iteración se consideraron las siguientes tareas:

- **Torre con Cañón** Esta fue la tarea más compleja y la que más tiempo requirió ya que primero era necesario utilizar un marcador diferente a los que ya había diseñados, segundo, diseñar una máquina de estados que se ajustara de forma correcta a la jugabilidad de la torre, y tercero encontrar los recursos gráficos visuales que se ajustaran tanto a la temática del minijuego como a lo que se pretendía construir. Por lo que para

llevar a cabo esta tarea se utilizó casi todo el tiempo de la propia iteración, ya que era necesario volver a ajustar el diseño, equilibrar la jugabilidad y crear un objeto nuevo desde cero.

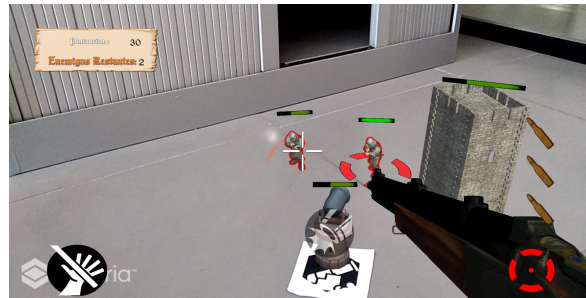


Figura 6.9: Captura de la Torre con Cañón en el minijuego

- **Selección de minijuegos** El objetivo de esta tarea era permitir que el videojuego pudiera contener diferentes minijuegos y seleccionarlos, en un principio se pensaba realizar mediante un sistema de marcadores, y que fueran estos los encargados de contener los diferentes minijuegos, pero como no se conoce qué zonas son consideradas que se pueden explorar se prefirió el uso del mapa del plano del castillo para poder colocar minijuegos de forma generalizada en cualquier parte del castillo. En la figura 6.10 se puede ver el plano del castillo con puntos de minijuegos.



Figura 6.10: Captura de la selección de minijuegos

- **Ajustar comportamiento Soldados** Con la nueva torre con cañón se pretendía mejorar la jugabilidad aunque con la potencia de fuego que posee hacía relativamente fácil de superar el minijuego, por lo que se rediseñó la lógica de los Soldados para que solo estos pudieran atacar a la torre una vez que el jugador la posicionara.
- **Modelo del castillo de *Santiago de la Torre*** En el menú principal se mostraba un modelo de un castillo gratuito, pero en las últimas semanas, la empresa con la que se colabora en este proyecto facilitó un modelo tridimensional del castillo realizado con un dron.
- **Pulir aspecto gráfico** Esta tarea consistía en cuidar el cómo y dónde se colocan los elementos de la interfaz, además de añadir algunos gráficos y texturas que permitieran

encajar mejor la información que se quiere transmitir del videojuego, por ejemplo en la figura 6.9 se ve cómo la munición de la *espingarda* pasa a un lateral para no ocupar tanto en la pantalla, la información de la ronda como enemigos restantes y puntuación se agrupa arriba a la izquierda para que no se queden tan separados, añadir sombras y ajustar *shaders* para que el aspecto visual del videojuego sea agradable. Esta tarea es muy subjetiva a la hora de definir cuánto tiempo puede durar ya que depende de muchos factores.

Esta última iteración duró 3 semanas, se encontraron numerosas dificultades a la hora de ajustar este nuevo objeto, integrar los cambios en la interfaz y ajustar el modelo del castillo para que pudiera verse bien en *Unity*, como resultado final se obtiene un *ejecutable .apk*

6.2 Implementación del videojuego

Como se ha mencionado anteriormente este proyecto se ha realizado en colaboración con la empresa *Akura Games* y bajo la tutela del tutor de este proyecto. La dedicación de este trabajo tenía como objetivo desarrollar un videojuego que sirviera como prototipo a modo de visualización de la temática y la gestión de minijuegos en el entorno real. Sin embargo debido al actual estado de ruina del castillo, se planteará a continuación cómo se realizaría la implementación.

- Primero que todo se debería definir exactamente qué puntos dentro del castillo sería interesante ver, además de qué elementos pueden intervenir en ellos, para eso habría que realizar una encuesta sobre las personas familiarizadas con el castillo y aquellos que están relacionados con el proceso de musealización del castillo.
- Una vez definidos los puntos interesantes en el castillo se debería realizar una lluvia de ideas sobre qué minijuegos podrían integrarse en estos puntos y cómo se pueden reutilizar los elementos que ya se han desarrollado en el minijuego que se ha llevado a cabo este proyecto y así realizar un análisis del impacto que pueden tener estas nuevas ideas en tiempo y en coste de desarrollo para que pueda encajar en el proceso de musealización.
- Sería necesario contactar con artistas, músicos, diseñadores de gráficos tridimensionales que pudieran aportar recursos gráficos y modelos con un listado de los objetos que se considerarían adecuados dentro de los minijuegos.
- Habría que realizar un *storyboard* de cómo sería el flujo de minijuegos que se podría seguir dentro del castillo para que tuviera sentido conocer los elementos culturales del mismo de una forma cronológica, y en qué orden deberían poderse desbloquear estos.
- Colocar los marcadores utilizados en los minijuegos en los puntos que se consideren dentro del castillo, reservando aquellos que sean interaccionables y los que deberían quedarse estáticos.

- Una vez que el videojuego concluyera con una segunda versión de lo que es este proyecto, integrarlo en el *marketplace de Google Play* para que cualquier usuario que lo desee, pueda jugarlo.
- Analizar la posibilidad de una gestión de usuarios mediante *Google Play* para que se pueda obtener información sobre los posibles turistas que quieran acudir al castillo y obtener así conocimiento sobre qué minijuegos gustan más y si realmente el videojuego tiene una visión adecuada.

6.3 Coste del proyecto

En esta sección se hará una revisión del coste del proyecto teniendo en cuenta la estimación en horas de lo que supondría cumplir con las iteraciones y el salario medio de un desarrollador en aplicaciones móviles, ya que este proyecto trata un desarrollo para *Android*. Es difícil encontrar información acerca del salario medio en desarrollo de videojuegos ya que el medio aún no está tan asentado como para poder diferenciar entre el sueldo de un desarrollador de aplicaciones al de un desarrollador especializado en videojuegos.

Según se refleja en [21] el rango salarial de los desarrolladores *Android* se encuentra entre los 25.000 y los 45.000 €, teniendo en cuenta que este proyecto se enmarcaría con un perfil desarrollador sin experiencia, el sueldo sería aproximadamente de 25.000 € anuales. Haciendo un cálculo de horas efectivas y con el sueldo anual bruto, se estima que el coste por hora de un desarrollador sería de 13.89€/ hora.

En la tabla 6.1 se hace una estimación del coste de cada iteración teniendo en cuenta el precio de desarrollador por hora efectiva y la estimación en horas que supone el cumplimiento de todas las tareas.

Iteración	Estimación horas	Coste
1ª Prototipo	45 horas	625 €
2ª Enemigos	75 horas	1.041,75 €
3ª Refactorización y mejoras	60 horas	833,4 €
4ª Apartado Visual	70 horas	972,3 €
5ª Menú principal y Optimización	30 horas	416,7€
6ª Torre con Cañón y Mantenimiento	120 horas	1.666,8 €

Cuadro 6.1: Estimación del coste del proyecto

Por lo que el coste estimado total del proyecto sería de 5.555,95 €. Teniendo en cuenta que gran parte del tiempo se utilizaría en aprender y reforzar conocimientos sobre las herramientas y el precio por hora se ha calculado teniendo en cuenta las horas efectivas de trabajo.

Conclusiones

EN este capítulo se expondrán los objetivos planteados para este proyecto y como se han alcanzado, además de incluir aspectos relacionados con las dificultades que se han podido encontrar, cómo este proyecto se vería en un futuro y algunas conclusiones y afirmaciones finales sobre el mismo.

7.1 Objetivos alcanzados

En esta sección se revisan los objetivos inicialmente planteados para ver en qué medida estos se han cumplido. Aquí están listados los objetivos definidos:

- **Diseño y desarrollo de una solución modular adaptada a los requisitos del usuario**

El cumplimiento de este objetivo está orientado a los requisitos que se especificaban tanto en la descripción del proyecto TFG como en los requisitos que se han refinado durante el desarrollo debido a la falta de concreción de ellos.

Este objetivo se da por cumplido debido a la realización de un sistema de *selección de minijuegos* con una arquitectura modular que permite desarrollar nuevas escenas en *Unity* y poder enlazarlas de manera que además se almacene los datos sobre estos que se considere necesario como estadísticas, completitud, etc. También se han proporcionado soluciones a la hora de almacenar información persistente por lo que se vería posible una adaptación para añadir un módulo de gestión de usuarios en un futuro.

Además de diseñarse un minijuego en concreto en el que se puede encontrar un desafío que resolver, con una temática y contexto relacionadas con el periodo cultural del que se trata y un sistema de reconocimiento de marcadores y eventos que permite incluir la tecnología de realidad aumentada y desacoplar comportamientos en los elementos del videojuego respectivamente .

- **Diseño y desarrollo de una arquitectura escalable, basada en patrones de diseño, para la inclusión de nuevos minijuegos**

Este objetivo se puede calificar como alcanzado, ya que a lo largo de todo el proyecto se ha procurado seguir los patrones de diseño que mejor se aplican al desarrollo de videojuegos, ya que además se encontraba con la limitación de que la plataforma de

este videojuego se trataba de móvil *Android* el cual por lo general no cuenta con unas características hardware potentes, por lo que desde el principio era necesario aplicar patrones como el de *Object Pool* que permite hacer una gestión más eficiente de la memoria, *Singleton* para controlar la persistencia de datos, *Modelo-Vista-Controlador* en el caso de los minijuegos, los cuales tienen una representación e interfaz donde se muestra la información del mismo, un controlador que gestiona el modelo y la información almacenada de forma persistente, el *Principio de responsabilidad único* que además encaja con la arquitectura que ofrece *Unity* pues cada componente que se puede añadir a un *GameObject* tiene una funcionalidad única, sólo en algunas excepciones en las que es necesario hacer un control más general sobre diferentes elementos para que de forma compuesta tengan sentido como el *GameManager* del minijuego que se ha realizado, que se encarga de la lógica, la condición de victoria y derrota y la interfaz con la información de la partida, y por último el *patrón Observador* siguiendo un sistema de eventos que incluye el lenguaje *C#*, de esta manera se podían comunicar los componentes sin necesidad de hacer un acoplamiento directo y así cumplir con el resto de patrones, este se ha utilizado por ejemplo para el estado de los marcadores, en el que se conoce cuándo la cámara los encuentra o los pierde.

Además para la inclusión de nuevos minijuegos se ha realizado lo que en *Unity* se conoce como un *prefab*, lo que permite básicamente es crear nuevos objetos con una serie de componentes y características establecidas arrastrando simplemente en la escena, de esta manera colocar nuevos minijuegos consiste en rellenar la información básica del minijuego con un título, una descripción y la escena correspondiente, y con eso la arquitectura establecida permite cargar la información de forma automática y la escena.

■ Desarrollo de un sistema para la gestión de minijuegos basada en marcadores

Este objetivo se puede dar por cumplido, aunque la gestión de los minijuegos en este momento no se hace desde los propios marcadores, sino desde un menú en el que se seleccionan los minijuegos, sin embargo, el videojuego está preparado para diferenciar los minijuegos según marcadores debido a la creación de unos marcadores de tipo *VuMark* que son unos marcadores especiales del motor de *Vuforia*, que permiten, a partir de un diseño determinado de marcador, generar n combinaciones de marcadores, lo cual permitiría que en cada escena de minijuego se seleccione con un identificador único un *VuMark* específico para designar el contenido del minijuego.

Actualmente en el videojuego se hace uso de dos *ImageTarget* que son marcadores normales basados en una imagen que es reconocida por *Vuforia* en base a la cantidad de características que ésta ofrezca, se ha realizado así por simplicidad a la hora de desarrollar este proyecto.

■ Aplicación de normas de calidad para desarrollo en dispositivos móviles Android

Para la aplicación de normas de calidad se ha seguido el modelo de evaluación de calidad que se establece en [22], en el que establece que se debe seguir cuatro aspectos fundamentales del modelo de calidad de la *ISO 9126* que son:

- **Usabilidad** Esta se define como la capacidad de un producto software para ser entendido, aprendido, usado y atractivo para el usuario, se ha procurado que todas las interfaces que estuvieran en el videojuego fueran lo más sencillas y usables posibles, delegando sólo el comportamiento de los objetos por ejemplo a uno o dos botones máximo, con iconos fáciles de entender e información constante del estado de la partida en el minijuego
- **Eficiencia** Como se ha explicado anteriormente se han seguido desde un principio el uso de patrones de diseño que permitieran un eficaz uso de las características hardware de las que dispone normalmente un teléfono móvil *Android*. Teniendo en cuenta las restricciones que ofrece las tecnologías de realidad aumentada como el uso de una cámara y una potencia de cpu superior respecto a terminales antiguos o carentes de potencia.
- **Flexibilidad** Este se define como la facilidad con la que un sistema o componente puede ser modificado, en el intento de realizar una solución modular este requisito se podría ver cumplido, ya que tanto la intención como *Unity* orientan el desarrollo hacia el diseño de componentes individuales que tengan un único sentido y comportamiento, también este proyecto gracias a las herramientas que soporta *Unity* ofrece flexibilidad a la hora de distribuir el juego de manera que se pudiera exportar a diferentes versiones de *Android* o incluso convertirse en un juego multiplataforma.
- **Persistencia de Datos** Este requisito es necesario para que se pudiera guardar tanto los ajustes del jugador como sus estadísticas en los minijuegos por lo que se cumple, utilizando un sistema lo más desacoplado posible que permite adherir nuevas estructuras de datos sin comprometer la consistencia del sistema.

Por lo que el objetivo general de **Diseñar y desarrollar un videojuego Android basado en realidad aumentada para dinamizar visitas a edificios culturales** se ve alcanzado, ya que contiene todo lo que se espera de este objetivo, el resultado del proyecto es un ejecutable para el sistema operativo *Android* el cual contiene minijuegos basados en realidad aumentada, y la jugabilidad del videojuego se basa en recorrer el castillo de manera que el jugador seleccione un minijuego basándose en la localización del castillo, lo que permite elegir. jugador a los puntos interesantes que él decida visitar, pues se espera que en la implantación del videojuego haya marcadores colocados estratégicamente en las zonas del castillo.

7.2 Dificultades encontradas

De los objetivos que se han planteado, como dificultad notable se podría destacar el *Desarrollo de una arquitectura modular y escalable*, ya que a la hora de desarrollar nuevas funcionalidades en el videojuego, siempre surgía la duda de cómo plantearla desde un punto de vista generalizado, lo cual dificultaba considerablemente el desarrollo de la misma. Además en el desarrollo de videojuegos se pueden encontrar muchos ejemplos de cómo realizar ciertas funcionalidades, pero no quiere decir que una manera sea la mejor para el caso concreto que se trata por lo que sumaba a la dificultad ya existente.

También se requirió un periodo de aprendizaje muy extenso para familiarizarse con la herramienta *Unity* ya que se trataba de una tecnología nueva además de añadir el extra de la realidad aumentada lo que complicaba más el aprendizaje de nuevos conceptos y formas a la hora de programar.

7.3 Trabajo futuro

A continuación se expondrán una serie de ideas por las que el trabajo que se ha realizado podría mejorar o evolucionar:

- El desarrollo ha servido sobre todo para establecer una idea de cómo se espera que este videojuego podría establecerse en un futuro cuando el castillo se encuentre listo para convertirse en un museo, por lo que se puede aprovechar incluso el minijuego realizado con algunos cambios para darle más contexto y sentido a lo que ya se ha realizado, añadiendo diálogos y nuevos personajes.
- Incorporar con el sistema de selección de minijuegos toda una batería de minijuegos que permitan realizar rutas diferentes por el entorno del castillo que contengan elementos de naturaleza cultural en ellos, pudiendo ser tan sencillos como un puzzle o un minijuego más complejo con historia, está la posibilidad y las herramientas para realizarlos.
- Añadir modelos tridimensionales propios al minijuego que se encuentra realizado, para que el videojuego tomara más reconocimiento y la empresa resultara beneficiada, ya que al realizar los modelos, se puede utilizar un mismo estilo y definir su propia identidad.
- Desarrollo de un módulo de usuarios que permitiera establecer *rankings* sobre las puntuaciones en los minijuegos y establecer así una competencia por la rejugabilidad de ellos.

7.4 Conclusiones finales

Si bien este proyecto trata de un prototipo de lo que podría ser el videojuego completo, la utilidad del proyecto prácticamente sirve para establecer las bases del videojuego y agregar

funcionalidades que puedan ser reutilizables, por lo que si se utiliza de manera adecuada y se le da un enfoque parecido y completo, el videojuego podría llegar a atraer la atención de turistas que desconozcan la zona y así familiarizarlos con la historia del castillo.

En un sentido más allá del práctico, este proyecto ha resultado enriquecedor en muchas facetas, la primera sería en la experiencia laboral, al trabajar en un entorno empresarial se han realizado muchos contactos, también conocido a personas dentro de la oficina que expusieran un punto de vista muy alejado del académico invitando a entrar en este mundo laboral y adulto. En segundo lugar por el conocimiento sobre las tecnologías utilizadas, ya que hoy en día el sector de los videojuegos es tan potente, conocer estas herramientas de primera mano es una oportunidad única para mejorar las capacidades propias y es que en este momento se siente como si se pudiera realizar cualquier proyecto de videojuegos que se presenta.

Por último indicar la afirmación de seguridad ganada sobre uno mismo al haber afrontado un proyecto del que no se conocían las herramientas y se ha partido desde cero utilizando los recursos que había disponibles, el hecho de haber conseguido algo funcional y que además sea entretenido es la mayor prueba de gratitud que se puede tener a nivel personal de este proyecto.

ANEXOS

Anexo A

Game Design Document

Game Design Document Outline

A game design document is the blueprint from which a game is to be built. As such, every single detail necessary to build the game should be addressed. The larger the team and the longer the design and development cycle, the more critical is the need. For your purpose, the intent is to capture as much as possible of your design. I want you to think big...bigger than what you are able to develop. I also want you to be clear about what the software delivers and what the design entails. My recommendation is that you define the ultimate game and then clarify what it is that you have developed. If you are finding it too difficult to do that, you may produce too documents.

1. DOCUMENTO DE DISEÑO DE JUEGO

1.1. Santiago RA *

2. **Game Overview** Videojuego en android con realidad aumentada

2.1. **Game Concept** Guía rutas en realidad aumentada

2.2. **Genre** Aventura Puzles

2.3. **Target Audience**

Turistas

2.4. **Game Flow Summary – How does the player move through the game. Both through framing interface and the game itself.**

El jugador buscará marcas para identificarlas con la cámara con las cuales se podrán introducir elementos virtuales con los que el usuario podrá interactuar, así como recoger objetos, usarlos, resolver puzzles, acertijos, utilizar elementos del entorno real para interacciones con los virtuales, etc

2.5. **Look and Feel** – What is the basic look and feel of the game? What is the visual style?

El juego se tiene que ver con un estilo medieval y se tiene que sentir libertad a la hora de interactuar con el mundo virtual, ya sea con diálogos y manipulación de objetos.

El estilo visual aún no está definido.

3. **Gameplay and Mechanics**

3.1. **Gameplay**

El usuario utilizará su teléfono para dirigirse a los elementos virtuales, podrá coger objetos, rotarlos, lanzarlos, mantener diálogos y en base a lo que haga recibirá puntos.

3.1.1. **Game Progression**

El progreso se verá definido en la ubicación del jugador, a medida que avance por el castillo verá marcadores los cuales serán los diferentes niveles o escenarios que el jugador tendrá

3.1.2. **Mission/challenge Structure**

Las misiones serán activadas una vez que el jugador encuentre un marcador, lo normal es que reciba un diálogo o contexto de qué tiene que hacer, normalmente se plantea como un problema que yace en el castillo y que el jugador debe resolver.

3.1.3. **Puzzle Structure**

El jugador tendrá una serie de objetos virtuales con los que puede interactuar, para completar el desafío hallará una manera para colocar los objetos o utilizarlos y así resolver el problema.

3.1.4.Objectives – What are the objectives of the game?

Dar a conocer la historia del castillo a través de minijuegos o retos

3.1.5.Play Flow – How does the game flow for the game player

Existen dos tipos de flow, uno que trata de buscar los marcadores a la vez que el jugador visite el castillo, el videojuego te mostrará pistas de dónde se puede encontrar el siguiente marcador, si no, el jugador puede volver a la anterior zona y echar un vistazo por si encuentre algo necesario.

Después está el flow de gameplay que trata cuando el usuario ya ha encontrado el marcador, se le explica el contexto en el que se encuentra, se le explica un problema y mediante la interacción con los objetos debe resolverlo.

3.2. Mechanics – What are the rules to the game, both implicit and explicit. This is the model of the universe that the game works under. Think of it as a simulation of a world, how do all the pieces interact? This actually can be a very large section.

Principalmente todas las mecánicas se destilan en la dirección y posición del teléfono, dependiendo de dónde mires así podrás interactuar.

Si la cámara del jugador detecta un marcador se activará la escena correspondiente y con ella las misiones o desafíos que conlleva

Si el jugador selecciona un objeto pueden pasar varias cosas:

- Es un objeto clave por lo que si el jugador hace click sobre él se guardará en el inventario y podrá sacarlo pero no podrá moverlo rotarlo ni moverlo en el sitio, el objeto permanecerá estático siempre delante del jugador.
- Es un objeto manejable, el jugador puede mover, lanzar o rotar ese objeto libremente

Si el jugador selecciona un NPC

El jugador puede ver pistas acerca de dónde se encuentra el próximo marcador

3.2.1.Physics – How does the physical universe work?

Los elementos que el jugador pueda mover actuarán con físicas, los demás serán estáticos.

Si un jugador suelta un objeto manejable, este caerá al suelo

Si un jugador saca un objeto clave este permanecerá estático delante del jugador

Las interacciones se realizarán en base a colisiones, todos los objetos llevarán asociados un tag y un Collider que permitirá realizar la lógica del videojuego una vez que estos colisionen.

Habrán elementos del videojuego que actúen con físicas pero el jugador no interacciones con ellas directamente.

3.2.2.Movement in the game

El movimiento del jugador está asociado con el movimiento físico del jugador.

Los movimientos de los objetos estarán asociados a las físicas que tengan.

Habrán elementos que se muevan libremente.

El jugador puede intervenir en el movimiento de algunos objetos.

3.2.3.Objects – how to pick them up and move them

Hay diferentes tipos de objetos:

Objetos Clave: Serán objetos útiles para la lógica o la historia, sin estos objetos no se podría avanzar en el videojuego y sólo se pueden guardar en el inventario o sostenerlos estáticamente, para que si el jugador se acerca con un objeto clave pueda desencadenar un evento.

Objetos Manejables: Son objetos para el libre uso, se pueden mover rotar, también se pueden perder, son objetos no relevantes para la historia pero sí para los desafíos.

Objetos Clickables: Puedes hacer click sobre objetos para que te desvele algo de información, estos objetos no se podrán recoger en el inventario ni se podrán mover.

Los objetos que se guardan en el inventario tendrán un icono que se mostrará en el botón del inventario.

Todos los objetos forman parte de los marcadores, si el jugador coge un objeto pasará a formar parte de la cámara o del inventario, en el caso que los objetos colisiones con un marcador, volverán a formar parte de él.

3.2.4.Actions, including whatever switches and buttons are used, interacting with objects, and what means of communication are used

El juego dispone de una interfaz que le permitirá ajustar opciones, salir del juego, pausarlo, comprobar sus logros, ver qué items tiene en el inventario y cuál es la misión actual.

Las acciones que se pueden llevar a cabo en el juego vienen delimitadas por la escena en la que se encuentre.

Marcador:

Cuando se encuentre un marcador, se cambiará la misión actual por la que define el marcador, se cargarán todos los elementos de la escena, el inventario y los puntos o logros se mantienen

Inventario:

El inventario está compuesto por tres botones, los cuales son una lista de objetos que están esperando a ser instanciados, si haces click sobre un botón se instanciará el objeto delante del jugador y se eliminará de la lista de objetos.

Si haces click sobre un botón vacío no pasará nada.

Si haces click sobre el objeto, volverá al inventario en este caso reordenándose la lista.

Objetos:

Si haces click sobre Objetos llave estos se guardarán en el inventario.

Si haces click sobre Objetos Manejables estos objetos pasarán a formar parte de la cámara y se moverán con ella.

Si utilizas dos dedos puedes rotar los objetos manejables.

Si haces click sobre Objetos Clickables obtienes información.

Personajes:

Habrán diferentes tipos de personajes:

Personajes con diálogo: si haces click sobre ellos, estos te ofrecerán un diálogo y dos respuestas que aparecerán como botones en la pantalla, en este caso el jugador elegirá si pulsar un botón u otro, dependiendo de ello, la misión puede cambiar.

Personajes enemigos: Pueden haber enemigos que tengas que derrotar haciendo click sobre ellos o utilizando objetos.

Menú:

En el menú tendrás diferentes botones, con los que poder ver los logros actuales, el mapa del castillo, la historia, etc.

3.2.5. Combat – If there is combat or even conflict, how is this specifically modeled?

Si hay un sistema de combate, estará basado en barras de vida o porcentajes, el jugador tendrá su propia barra de vida, o le afecta de alguna manera negativa que reciba daño, por ejemplo, pierde dinero o puntos.

Para recibir daño el enemigo disparará contra el jugador, si la cámara detecta colisiones en el sentido, el jugador recibirá daño, también el enemigo puede utilizar otro tipo de armas a corto, medio alcance.

Para realizar daño el jugador podrá disparar con algo al jugador enemigo, o utilizar los objetos en su entorno.

3.2.6. Economy – What is the economy of the game? How does it work?

La economía del juego, ya que se trata de una época medieval, moderna se hará con monedas de oro las cuales podrás utilizar para conocer la historia u obtener ciertos privilegios en los desafíos o diálogos

El jugador dispondrá de una cierta cantidad limitada de dinero o puntos

Aquí hay una idea curiosa, se podrían hacer marcadores de dinero, por ejemplo de 1, 10 y 100 monedas que se podrán encontrar distribuidos por el entorno real.

El dinero virtual se conseguirá en cofres que se abrirán con llaves

Los personajes virtuales pueden intercambiar dinero a cambio de un objeto, o puede ser un requisito para avanzar en el videojuego.

3.2.7. Screen Flow -- A graphical description of how each screen is related to every other and a description of the purpose of each screen.

Splash Screen: La pantalla inicial donde aparecerá el logo de la empresa

Inicio Screen: Es la pantalla que contiene el menú principal del juego donde el usuario tiene tres botones para seleccionar:

- Iniciar Juego: Este botón activará el cambio de escena y cargará la primera escena de gameplay



- Historia: Donde aparece una ventana con un fragmento de texto introduciendo la historia del castillo.

- Tutorial: Aquí aparece una ventana o escena donde te explica cuáles son los controles principales del juego, aparecerá un marcador y una vez que el jugador lo encuentre simplemente se carga la escena.

Gameplay Screen: Esta pantalla está compuesta de varias escenas pero se pueden clasificar en dos:

- Busqueda de Marcadores: En estas escenas el jugador tendrá una pista de dónde se puede encontrar el siguiente marcador.

Dispondrá del inventario por si necesita sacar un objeto.

También aquí se pueden encontrar marcadores que no sean para cargar la historia, sino cofres por ejemplo.

- Historia: Estas escenas estarán relacionadas con los desafíos y la historia que el jugador debe seguir, se activan una vez se encuentra el marcador correspondiente.

La conexión de estas pantallas se realiza con un controlador de escenas que permita enviar información como la misión actual, el inventario del jugador, su dinero.

Además los elementos de la interfaz en la pantalla de Gameplay serán:

1: Misión actual: aquí se reflejará qué tiene que hacer el jugador para avanzar en la historia, las misiones tienen un título, descripción, un icono y una recompensa.

2. Botón de Opciones: Un botón que permita desplegar el menú para ver los logros obtenidos, el mapa o salir del juego

3. Inventario: tres botones que indican los objetos que están en el inventario, si el jugador hace click sobre ellos, el objeto se colocará delante del jugador.

3.3. Game Options – What are the options and how do they affect game play and mechanics?

Aún no se conoce si el juego dispondrá de diferentes maneras de resolver los desafíos o cómo completarlos pero las opciones que ofrece dependen de cómo el jugador quiera interaccionar con los objetos que se le ofrecen.

3.4. Replaying and Saving

El jugador podrá rejugar el videojuego comenzando desde cero

3.5. Cheats and Easter Eggs

Algún marcador personalizado que contenga algún modelo gracioso.

4. Story, Setting and Character

4.1. Story and Narrative – Includes back story, plot elements, game progression, and cut scenes. Cut scenes descriptions include the actors, the setting, and the storyboard or script.

Aquí incluyo los aspectos que considero relevantes del documental

El Castillo en el que se inspira el videojuego se trata de una vieja fortaleza medieval que comenzó su construcción en el Siglo XV de la mano de los caballeros de la Orden de Santiago.

En 1434 se conoce que el nombre del castillo es Santiago el quebrado, y pasaría a llamarse Santiago de la Torre por iniciativa de Pedro Gonzalez del Castillo y Portocarrero que fue su primer comprador, el antiguo propietario era Rodrigo Rodríguez de Aviles, marido de Beatriz Fernandez Pacheco (1428)

El castillo tiene en su interior una torre exenta que fue el primer bloque del castillo que se construyó, formada por tres plantas se llama la torre del homenaje, su diseño interior ha sido modificado durante toda su historia hasta convertirse en un palomar (en el sótano se dice que había un calabozo)

Esa sería la primera fase del castillo, las próximas vendrían de mano de los futuros propietarios, quienes querían expandir el castillo y hacerlo más habitable, la segunda fase se compone de la muralla que rodea la torre del homenaje y la torre circular encontrada en el Noroeste. Después en la tercera fase se construyó el resto.

La arquitectura del castillo no sólo estaba pensada para su habitabilidad sino también para ser capaz de ofrecer estrategias defensivas, dispone de pasillos en las murallas por los que los cañones pueden pasar y ventanas con las que los soldados podrían disparar con su espingarda.

Fueron 4 los señores del linaje del castillo y portocarrero que consiguieron edificar todo el castillo. En 1579 Antonio del Castillo y PortoCarrero vendió el castillo a Alonso Pachecho Guzmán (Regidor de San Clemente) con lo que el castillo pasaría a tomar parte del término municipal de San Clemente

En el Siglo XVI los castillos medievales comienzan a estar en desuso, algunos como el de Santiago de la Torre se utilizan para crear nuevas habitaciones y salones donde se pudiera vivir y trabajar.

En el siglo XIX pasa a manos de varios propietarios, los cuales deciden construir una pequeña aldea a su alrededor para atraer a trabajadores de campo y que pudieran hospedarse en sus cultivos.

Ya en el Siglo XX se abandona el castillo y comienza su deterioro, por lo que sabemos la zona del castillo seguía teniendo actividad gracias a la ermita, la cual era muy visitada por los residentes de El Provencio y San Clemente.

A la espera de extraer ideas para el desarrollo

4.2. Game World

- 4.2.1. General look and feel of world
 - 4.2.2. Areas, including the general description and physical characteristics as well as how it relates to the rest of the world (what levels use it, how it connects to other areas)
- 4.3. Characters. Each character should include the back story, personality, appearance, animations, abilities, relevance to the story and relationship to other characters
- 5. Levels
 - 5.1. Levels. Each level should include a synopsis, the required introductory material (and how it is provided), the objectives, and the details of what happens in the level. Depending on the game, this may include the physical description of the map, the critical path that the player needs to take, and what encounters are important or incidental.
 - 5.2. Training Level
- 6. Interface
 - 6.1. Visual System. If you have a HUD, what is on it? What menus are you displaying? What is the camera model?
 - 6.2. Control System – How does the game player control the game? What are the specific commands?
 - 6.3. Audio, music, sound effects
 - 6.4. Help System
- 7. Artificial Intelligence
 - 7.1. Opponent and Enemy AI – The active opponent that plays against the game player and therefore requires strategic decision making
 - 7.2. Non-combat and Friendly Characters
 - 7.3. Support AI – Player and Collision Detection, Pathfinding
- 8. Technical
 - 8.1. Target Hardware
 - 8.2. Development hardware and software, including Game Engine
 - 8.3. Network requirements
- 9. Game Art – Key assets, how they are being developed. Intended style.

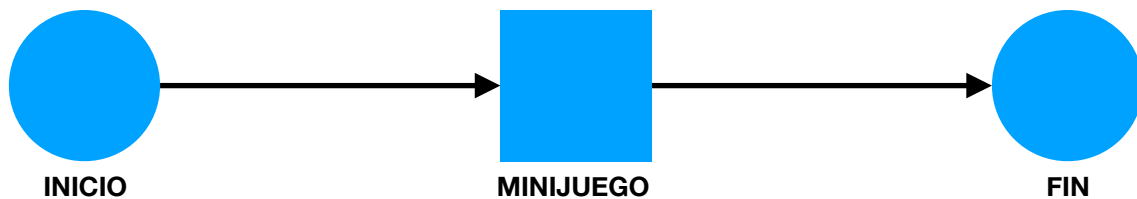
Anexo B

Prototipo de Santiago de la Torre RA

Prototipo Santiago de la Torre (Realidad Aumentada)

Introducción

En este documento se recoge qué va a contener el prototipo y cómo se va a desarrollar, se tiene pensado que consistirá en un principio de 3 escenas compuestas por un Inicio, un minijuego y un fin, lo que se espera es que el diseño permita la inclusión de nuevos minijuegos.



Descripción de Escenas

A continuación se detallará por escrito qué ocurrirá en cada escena:

Escena de Inicio:

El jugador, una vez dentro del videojuego, tendrá un **objetivo inicial** que será encontrar el primer marcador. Una vez que lo encuentre le aparecerá un cartel con el nombre del castillo y un personaje con el que puede interactuar.

Si el jugador toca al personaje este hablará con el jugador, con un sistema de **diálogo** parecido al de los juegos RPG, se ha elegido este sistema debido a su sencillez de edición y fácil desarrollo, Está basado en cajas de dialogo en las que el jugador tendrá que pulsar sobre ellas para continuar la conversación.

El personaje será un modelo sencillo de un caballero medieval, o para el prototipo quizás un cubo, la secuencia de diálogo que tendrá será la siguiente:

“Bienvenido al Castillo de Santiago de la Torre, tengo entendido que vienes a visitar nuestra gloriosa orden.. Bien, pues ve a la Torre del Homenaje, allí te dirán qué tienes que hacer si quieres entrar en la orden de Santiago.”

Al finalizar este diálogo el jugador tendrá un **nuevo objetivo** que será el de alcanzar la torre del homenaje y así es como se acaba esta escena, una vez el jugador le diga que sí al caballero.

Escena de Fin:

Una vez el jugador encuentre el marcador de fin, aparecerá el personaje del principio con un nuevo diálogo, si el jugador interactúa con él, el personaje tendrá el siguiente diálogo:

“Gracias a tus hazañas hoy podemos brindar por nuestra Orden, ¡muchas gracias!”

Cuando el jugador termine la conversación le **aparecerán los objetivos cumplidos**.

Escena de Minijuego:

Cuando el jugador encuentre el marcador que activa el minijuego le aparecerá un personaje con un diálogo:

“En tu camino aquí han venido a atacarnos, rápido, agarra una espingarda y ayúdanos”

Ahora el jugador tendrá en un **marcador** que está situado en una pared, un agujero que permite ver el exterior del castillo, donde **aparecerán enemigos** que han venido a asaltar el castillo. Los enemigos hacen spawn en un punto alejado dentro del agujero, y tienen un tiempo hasta llegar a **posición de tiro**. En ese momento el jugador deberá encontrar la **espingarda** que estará cerca y comenzar a disparar. La espingarda tiene **munición** por lo que una vez se agote el jugador deberá conseguir munición en un marcador cercano mientras los enemigos pueden dispararle.

Los enemigos tienen una cierta **cantidad de vida**, y los disparos funcionan en base a **probabilidad** para hacer más sencilla la implementación, cuanto más tiempo pase, esta probabilidad de disparo aumenta.

Si el jugador recibe daño la pantalla se volverá roja y la barra de vida decrecerá, si el jugador no consigue eliminar los enemigos a tiempo habrá perdido y tendrá que empezar desde el principio, volviendo a hablar con el personaje.

Si el jugador destruye a todos los enemigos y habla de nuevo con el personaje, habrá completado el objetivo y puede continuar hacia el siguiente minijuego o el final.

El personaje le dará una llave que permite abrir un cofre donde habrá una recompensa(?)

Elementos en Común

Las escenas encierran muchos componentes y elementos que si no se llevan adecuadamente pueden sumar la complejidad del prototipo, para ello es interesante analizar qué elementos tienen en común y extraer una primera idea de cómo sería su implementación.

Sistema de Diálogo:

Este sistema será muy sencillo, estará compuesto de una caja de diálogo en el Canvas que nos ofrece Unity el cual recibirá un texto y simplemente lo imprime.

Se puede interacciones con él tocando la cajita de diálogo, que hará que continúe la conversación o activar un objetivo o estado del juego (por ejemplo: escena de inicio, fin).

Sistema de Objetivos:

Este puede ser uno de los más complejos, no por el contenido sino por lo dependiente que será de otros sistemas, puedes actualizar objetivos, ya sea matando enemigos o continuando una conversación. Para ello se tiene pensado utilizar eventos de Unity que permitan desde diferentes partes del videojuego hacer llamadas a objetos y sus scripts.

Por ejemplo: Hemos acabado con todos los enemigos, se activa el evento `OnObjetivoCumplido(String TagObjetivo)` y el sistema recogerá qué objetivo se ha cumplido y actualizará la interfaz con el siguiente.

Es interesante añadir que los objetivos aparte de un título tendrán una imagen asociada, ya sea un objetivo de buscar marcadores o de matar enemigos, la idea es ayudar al jugador de manera visual.

Para ello se pretende usar Scriptable Objects en Unity que es una manera de hacer objetos con información que puedes asociar a instancias.

Sistema de Cámara:

La cámara en este juego es un aspecto fundamental ya que a través de ella el usuario podrá interactuar con todos los elementos del juego, para ello se pretende que la cámara se comporte de maneras diferentes. Por ejemplo en el caso del minijuego Shooter, cuando el jugador toque un punto de la pantalla que no sea de interfaz se proyectarán rayos que permitirán saber hacia dónde dispara el jugador.

Hay diferentes modos:

Item: en este modo el jugador puede interactuar con los elementos que tengan asociados el tag "item", estos objetos son manipulables, es decir, el jugador los puede mover y rotar, una vez haga click sobre ellos se agarrarán a la cámara, con lo cual los puede transportar y llevar a diferentes sitios.

Shooter: cuando el jugador haga click sobre un objeto que tenga asociado el tag "gun" este se colocará delante del jugador y simulará que el jugador lleva ese arma, sólo se puede hacer daño o disparar a aquellos objetos que se encuentren en la capa "Enemy"

(?) Es posible que sólo en este modo tengas una barra de vida

Sistema de Enemigos:

Los enemigos tendrán una cierta cantidad de vida, un comportamiento y un ataque, para este caso en concreto, supondremos que tienen un 100 % de vida, su comportamiento estaría basado en dos etapas, una en la que se encuentran en movimiento en el cual no atacan, y otra en la que están en posición de disparo. El ataque vendrá dado por una probabilidad de disparo, si esa probabilidad se da, el enemigo hará daño al jugador

Referencias

- [1] A. de Juan, “Poblete inaugura la ruta virtual de la batalla de alarcos, una recreación del hecho histórico a través de realidad aumentada,” *Lanza, Diario de La Mancha*, 2019.
- [2] S. Fernández, “Nokia N-Gage, el primer móvil gaming,” *Xataka Movil*, 2018.
- [3] P. Lara, “App store cumple 10 años,” *Apple*, 2018.
- [4] J. G. Nieto, “Así es como Android se ha comido el mercado en diez años,” *xataka móvil*, 2019.
- [5] R. Dillet, “Unity CEO says half of all games are built on Unity,” *techcrunch*, 2018.
- [6] AEVI, *Anuario 2017, Anuario de la industria del videojuego*, ch. 4, p. 33. AEVI, 2017.
- [7] A. E. de Videojuegos (AEVI), “El sector de los videojuegos en España: impacto económico y escenarios fiscales,” *AEVI*, 2018.
- [8] AEVI, *Anuario 2017, Anuario de la industria del videojuego*, ch. 5, p. 39. AEVI, 2017.
- [9] AEVI, *Anuario 2017, Anuario de la industria del videojuego*, ch. 5, p. 40. AEVI, 2017.
- [10] R. T. Azuma, *A Survey of Augmented Reality*, ch. 1.2. Presence: Teleoperators and Virtual Environments, 1997.
- [11] F. Brooks, “The computer scientist as toolsmith,” *Communications of the ACM*, vol. 39, no. 3, p. 64, 1996.
- [12] K. Beck, *Extreme Programming Explained, Embrace Change*. Addison-Wesley Professional, 1999.
- [13] P. Letelier, *Metodologías Ágiles en el Desarrollo de Software*, p. 4. Grupo ISSI, 2003.
- [14] A. Iacovelli, *Framework for Agile Methods Classification*, p. 92. Centre de Recherche en Informatique CRI, 2008.
- [15] A. Iacovelli, *Framework for Agile Methods Classification*, p. 97. Centre de Recherche en Informatique CRI, 2008.

- [16] K. Beck, *Test-Driven Development, by example*. Addison-Wesley Professional, 2002.
- [17] J. L. G. Sánchez, *Jugabilidad Caracterización de la Experiencia del Jugador en Videojuegos*. Editorial de la Universidad de Granada, 2010.
- [18] R. C. Martin, *Agile Principles Patterns and Practices in C*, ch. 8. Pearson Education, 2007.
- [19] E. Gamma, *Patrones de Diseño*, ch. 1, p. 4. Anaya, 2002.
- [20] R. Nystrom, *Game Programming Patterns*. Lightning Source Inc, 2014.
- [21] J. G. Lapresta, “Empleo en it,” *infoempleo research*, 2017.
- [22] D. Franke, “A mobile software quality model,” *Embedded Software Laboratory Ahornstraße*, 2012.

Este documento fue editado y tipografiado con \LaTeX empleando la clase **esi-tfg** (versión 0.20181017) que se puede encontrar en:
<https://bitbucket.org/arco.group/esi-tfg>

[respeta esta atribución al autor]

