Escuela Politécnica Superior

Desarrollo de un Videojuego de Realidad Aumentada en Unity

Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:
Javier Martínez Arias

Tutor/es:
Carlos José Villagra Arnedo

Mireia Luisa Sempere Tortosa

Mayo 2019

Universitat d'Alacant
Universidad de Alicante

RESUMEN

El proyecto presentado en la siguiente memoria consiste en el desarrollo de un videojuego con el motor Unity, de r*ealidad aumentada* y orientado a plataformas móvil. Por tanto, se trata de un juego en el que se combinan elementos virtuales con la realidad (técnica de realidad aumentada). La cámara del dispositivo es uno de los elementos principales para generar este efecto, así como los sistemas de geolocalización. De esta forma se ofrece al usuario nuevas experiencias interactivas con las que jugar con las entidades que aparecen en pantalla. Inicialmente, el videojuego se realiza orientado al sistema operativo Android, por las facilidades que ofrece Unity para esta plataforma respecto a las demás.

En definitiva, el objetivo principal en el juego consiste en avanzar con el móvil mientras visualizamos distintos elementos, pudiendo ser beneficiosos o perjudiciales para la partida. Entre ellos, existen enemigos con los que debemos combatir en distintas situaciones. El combate está basado, en términos generales, en un juego FPS (*First Person Shooter*) en realidad aumentada. Completando este tipo de enfrentamientos y superando otro tipo de objetivos, podremos avanzar gradualmente en la partida, que será guardada automáticamente para conservar el progreso alcanzado.

Con este proyecto se pretende mostrar el potencial de la realidad aumentada, los distintos tipos que existen y la multitud de aplicaciones que tienen en la actualidad. Se ha decidido orientarlo al mundo de los videojuegos por estar relacionado con el itinerario del grado cursado, por trabajar la integración con el motor Unity y por ajustarse a mis preferencias personales y profesionales.

AGRADECIMIENTO

A mi familia, por estar siempre a mi lado y apoyarme en las decisiones y situaciones más difíciles. A mis amigos, tanto de fuera como de dentro de la carrera, que me han ayudado mucho, especialmente en estos últimos años de universidad. A Carlos mi tutor durante este trabajo, que se ha preocupado durante estos meses que el proyecto se realizara lo mejor posible. Por su puesto, agradecer al resto de mis profesores, con los que he podido aprender e introducirme en los contenidos multimedia.

CITAS

“La ciencia de hoy es la tecnología del mañana” - Edward Teller.

“No he fracasado 999 veces. Simplemente he encontrado 999 formas de no crear una bombilla” - Thomas Edison.

“Trabajar por un mejor mañana no es cosa de un día. Las mañanas seguirán viniendo.” - Persona 4 Arena.

Índice de contenidos

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

El auge de la tecnología es un hecho, y qué mejor reflejo en la sociedad que los dispositivos móviles y la evolución que han sufrido en los últimos años. Los avances técnicos de los Smartphones están a la orden del día y, actualmente, se hace casi inconcebible la rutina sin depender de ellos. Por ello, se ha consolidado como la plataforma con mayor cantidad de usuarios. La apuesta de las empresas y la tendencia del mercado apunta a que el sector siga evolucionando notablemente en esta dirección, por lo que se requiere de una gran inversión en todo este tipo de proyectos y profesionales capaces de adaptarse a ellos.

Los videojuegos no son una excepción dentro de este bloque, ya que su auge se ha visto incrementado en igual medida en los últimos años. Aunque ya se pueden encontrar multitud de videojuegos con un gran éxito en las plataformas móviles, estos deben seguir evolucionando, adaptándose a las nuevas tecnologías que integran estos dispositivos. De esta forma, se ofrecen caminos y propuestas innovadoras, capaces de ofrecer experiencias interactivas a los usuarios.

Dos de las tecnologías que poco a poco tienen más presencia en la sociedad son la realidad virtual y la realidad aumentada. Estas propuestas de software hacen uso de los avances alcanzados por el hardware en las unidades de procesamiento. Además de los ordenadores, los teléfonos ya incluyen multitud de herramientas capaces de generar estos efectos.

En concreto, vamos a centrarnos en la realidad aumentada, que permite crear nuevas experiencias basadas en la integración de elementos digitales en la realidad, es decir, aportar más información a la realidad de la existente. Los dispositivos móviles, principalmente, hacen uso de la cámara en combinación con otras tecnologías.

En esta memoria, se pretende explicar los tres conceptos descritos anteriormente, es decir: realizar un videojuego de realidad aumentada para plataforma móvil. El objetivo principal es poner de manifiesto el potencial de la realidad aumentada, aplicada al mundo de los videojuegos para crear nuevas experiencias. Además, está orientado para teléfonos móviles, por ser la plataforma con mayor auge y difusión a nivel de usuarios.

Sin embargo, debemos empezar contextualizando e introduciendo los distintos conceptos necesarios para abordar este proyecto.

2. MARCO TEÓRICO O ESTADO DEL ARTE

En este bloque abordaremos una gran cantidad de temas en relación con el mundo de los videojuegos, enfocados al proyecto a desarrollar. En primer lugar, haremos referencia a la definición y a aspectos teóricos de los videojuegos, así como las herramientas de desarrollo empleadas actualmente para ellos. Nos centraremos también en las tecnologías de Realidad Aumentada que se emplean para llevar a cabo videojuegos de este tipo. Finalmente, dedicaremos un espacio para comentar los videojuegos más conocidos en relación con la *Realidad Aumentada*.

En particular, este juego se va a realizar mediante el motor de videojuegos Unity (Unity, 2019), del cual hablaremos en mayor profundidad a la hora de analizar qué es un motor y las diferentes herramientas de desarrollo. Unity nos permitirá combinar un entorno de programación junto con la integración de medios o elementos gráficos al juego (desde modelado, interfaz de usuario, sonidos, etc). Para añadir estos medios, vamos a hacer uso de otras herramientas multimedia, que nos permitan preparar y trabajar los diferentes tipos de contenidos. Una vez tengan el resultado deseado se integrarán en Unity como proyecto global.

2.1. Concepto de Videojuego

Para llegar a la definición de videojuego, debemos de tener una idea global de los diferentes tipos de contenidos que se integran en el producto final. También debemos generalizar, para que todos los géneros tengan cabida en este concepto.

Podría definirse como: *conjunto de contenidos digitales enriquecidos, unidos en una composición, que pretende crear una experiencia interactiva con el fin de entretener*.

Si buscamos la definición en Wikipedia (Videojuego, Wikipedia, 2019) o en la RAE (Videojuego, ASALE, RAE, 2019), se hace referencia a la necesidad de una pantalla de video para visualizar los contenidos. Es importante comentar que un videojuego requiere de al menos un jugador, que interactúe con el dispositivo mediante algún sistema de control. Este sistema envía respuestas acordes a las acciones del jugador, quedando reflejadas en pantalla. Por ello, tanto la pantalla como el sistema de control son fundamentales a la hora de jugar, así como el ordenador o consola que lo haga funcionar.

Centrándonos más en el juego en sí, se puede comentar que está regido por una serie de normas preestablecidas y que el jugador tiene que entender para poder adaptarse e interactuar con el videojuego. Los objetivos del juego pueden ser muy distintos y determinan, junto con la ambientación, el género del mismo.

2.2. Motores de videojuegos

2.2.1. Definición

En primer lugar, ¿qué es un motor de videojuego? Se puede definir como el software especializado para trabajar en la elaboración de un juego digital. Tiene una gran cantidad de subrutinas y funcionalidades ya preparadas para ahorrar tiempo y esfuerzos a el/los desarrolladores/es. Suele combinar la parte gráfica o escena, junto con la parte de programación. También da muchas facilidades para compilar y preparar juegos a distintas plataformas objetivo.

En general distinguimos tres tipos de herramientas: para desarrollo de juegos 2D, 3D o ambos. Los componentes clave que suelen incluir los motores (Motor de videojuegos, Wikipedia, 2019) son:

- Programa principal: Basado en el editor del conjunto ficheros que componen el juego, que incluyen las instrucciones o reglas del juego.
- Motor de renderizado: Permite visualizar el juego y probarlo en tiempo real. Se encarga de mostrar tanto elementos de interfaz estáticos como aquellos en movimiento. Muy útil para depurar de forma rápida. Dependerá también de las dimensiones espaciales del juego en cuestión.
- Gestor de sonidos: Automatiza el sistema de carga, modificación y reproducción de archivos de audio, evitando así el uso de librerías externas.
- Motor de físicas: Integrado para automatizar y simular las colisiones entre elementos en el juego.
- Herramientas para Inteligencia Artificial (IA): Funciones que agilizan la implementación de algunos comportamientos comunes en la IA de los juegos.

Por tanto, automatiza y simplifica algunas tareas, permitiendo un desarrollo más rápido y fácil. Sin embargo, al tener tanta configuración preestablecida, se limita la posibilidad de personalizar algunas funciones avanzadas, así como el desconocimiento de procesos internos del juego, gestionados por el motor.

2.2.2. Tipos y clasificaciones

Una vez explicado lo que es un motor de juego a grandes rasgos, pasamos a hablar de los diferentes tipos que encontramos en el mercado. Actualmente hay una gran cantidad de motores que agilizan el desarrollo de juegos. Si los clasificamos teniendo en cuenta las dimensiones espaciales en las que trabaja el motor y el tipo de licencia, se puede distinguir la tabla representada en la *figura 1*:

V · T · E	Game engines (list)	[hide]
Source port · First-person shooter engine (list) · Game engine recreation (list) · Game creation system · List of visual novel engines		
Free and open-source	2D	Adventure Game Studio · Beats of Rage · Cocos2d · Moai · OHRRPGCE · OpenFL · ORX · Pygame · Ren'Py · Stratagus · Thousand Parsec · Vassal · Xconq
	2.5D	*Doom* engine
	3D	Away3D · Bork3D · Blender Game · Cafu · Crystal Space · Delta3D · Dim3 · GamePlay · GLScene · Horde3D · Irrlicht · id Tech (*Quake* · *Quake II* · 3 · 4) · JMonkey · OGRE · Open Wonderland · Panda3D · Papervision3D · Platinum Arts Sandbox · Plasma · PlayCanvas · PLIB · Torque · Xenko
	Mix	Allegro · Godot · libGDX · Lightweight Java Game Library · Spring · Wintermute Engine
Proprietary	2D	Construct · Corona · GameMaker Studio · GameSalad · M.U.G.E.N · RPG Maker · Southpaw · Stencyl · UbiArt Framework · Vicious · Virtual Theatre · V-Play · Zillions of Games
	3D	4A · Amazon Lumberyard · AnvilNext · C4 · Chrome · Creation · CryEngine · Crystal Tools · Decima · Diesel · EGO · Elflight · Essence · Fox · Frostbite · HeroEngine · id Tech (5 · 6 · 7) · Ignite · IW · LithTech · Luminous Studio · LyN · MT Framework · Panta Rhei · PhyreEngine · REDengine · RAGE · Shark 3D · ShiVa · Snowdrop · Source · Unigine · Unreal
	Mix	Gamebryo · Unity
Historical		Bitsquid · Build · Dark · Enigma · Flare3D · Game-Maker · GameMaker · Garry Kitchen's GameMaker · Genesis3D · Genie · GoldSrc · Filmation · Freescape · HydroEngine · INSANE · Jade · Jedi · Marmalade · MADE · Pie in the Sky · Q · Reality Lab · RenderWare · Refractor · Riot · SAGE · SCUMM · Silent Storm · Sim RPG Maker · Sith · Titan · Truevision3D · Vision · Visual3D · Voxel Space · XnGine · Zero
Proprietary middleware		Euphoria · Gameware · GameWorks · Havok · iMUSE · Kynapse · SpeedTree · FaceGen

Figura 1: Captura en la que se pueden apreciar los motores juegos más relevantes en la actualidad.
Fuente: https://bit.ly/2R5utiU

Además de esta clasificación, existen otros criterios orientados a determinados géneros de juego, ya que ofrecen facilidades para funciones especializadas (Acción y aventuras, estrategia, etc.). En el caso de Unity, se puede hablar de un motor bastante genérico y relevante, que se explica a continuación.

2.2.3. Unity, el motor a emplear

Unity (Unity, Wikipedia, 2019) nos ofrece la posibilidad de desarrollar todo tipo de videojuegos multiplataforma sin especialización alguna: permite tanto juegos 2D como 3D y de base no tiene funciones enfocadas a ningún de género en concreto. Sin embargo, en

caso de necesitar librerías especializadas, siempre se pueden añadir paquetes o extensiones que ofrezcan esa posibilidad. Esto lo constituye una herramienta muy escalable y flexible, motivo por el que es tan usado en la actualidad.

Es un motor, que presenta las características generales mencionadas anteriormente (punto 2.2.1), con una gran personalización y con libre elección de editor de texto para el desarrollo (basado en el lenguaje de programación C#). Además, soporta una gran cantidad de extensiones de archivos desde imágenes, videos, sonidos, animaciones y modelos 3D. Por ejemplo, para estos últimos, tenemos los siguientes formatos (3D Formats, Unity, 2019): *FBX, dae, 3DS, dxf* y *obj*. Estos elementos pueden elaborarse desde programas específicos de modelado (por ejemplo, *Blender 3D*) para ser importados en los formatos indicados. Sin embargo, Unity presenta un editor de elementos 3D de modelado que, aun siendo muy básico, puede ser de gran utilidad.

La compatibilidad de Unity (Requisitos técnicos, Unity, 2019) es otro elemento que explica su enorme crecimiento como motor, ya que, en la actualidad ofrece muchos sistemas operativos compatibles con los que desarrollar: Windows 7 SP1 o superior, macOS 10.11 o superior, Ubuntu 12.04 o superior y SteamOS. Además, permite su exportación a las siguientes plataformas objetivo: Windows, Mac, Linux, Android, IOS, tvOS, Xbox One, PS Vita, PS4, WebGL, etc.

Por lo que respecta a las licencias, se trata de un Motor con dos tipos principales: la personal y la profesional. La primera de ellas es gratuita y permite desarrollar todo tipo de juegos sin superar unos umbrales de ganancias. Si, queremos obtener beneficios orientando el juego para vender, debemos pagar una licencia profesional de 125$ al mes. Actualmente existe una versión intermedia (Plus) entre las que hemos comentado, por 25$ al mes aproximadamente. En este caso, usaremos la versión personal, pues se trata de un proyecto universitario que no vamos a orientar para su venta.

El motivo por el que se ha escogido este motor se basa en 4 principios:

- La *popularidad* y *utilidad* que tiene hoy en día en el mundo laboral, ya que muchas empresas de videojuegos lo usan como plataforma principal de desarrollo. Por tanto, tener conocimientos de ella es de gran provecho.

- La gran comunidad que da soporte a la plataforma y las ventajas que supone frente a otras plataformas. Además, dispone de una tienda la "*Asset Store*", que ofrece facilidades con elementos prefabricados por otros usuarios.
- El buen rendimiento y fácil exportación a otras plataformas (hasta 25). En este caso el videojuego no debe tener unos requisitos muy altos, ya que está orientado a plataformas móviles. Por esto y por la gran cantidad de plataformas objetivo que soporta, es la mejor opción posible.
- Los paquetes o extensiones que soporta en relación con la realidad aumentada. Este apartado se comentará más en profundidad en el punto 2.3.

Para terminar, además de lo ya comentado, Unity dispone de una tecnología muy avanzada: algoritmos optimizados de *pathfinding* (para IA), sistema de físicas de *Nvidia* (*Physx*) para el 3D y por otro lado Box2D, uso de WebGL para juegos de navegador, etc. Por tanto, en la actualidad está muy consolidado, siendo distinguido como uno de los más completos.

Como última referencia, su competidor principal en la actualidad es ***Unreal Engine*** (Unreal Engine, Wikipedia, 2019). Aunque inicialmente era un motor destinado a la creación de juegos *shooter*, poco a poco ha ido mejorando y adaptándose a más géneros, consolidándose como uno de los mejores en la actualidad a la par con Unity. A diferencia de este, su código se basa en el lenguaje de programación C++. Se encuentra ya en su versión 4 y está diseñado y orientado para una gran cantidad de plataformas, al igual que Unity.

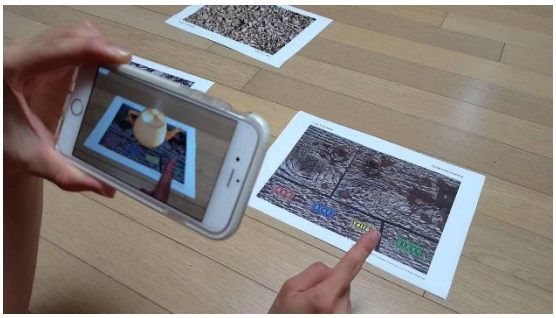
2.3. Herramientas de desarrollo en Realidad Aumentada con Unity

Una vez analizado el motor de juego a usar, se va a explicar la integración de características avanzadas que nos interesa. En este caso, vamos a centrarnos en la realidad aumentada, un componente fundamental del proyecto a desarrollar. Se trata de una tecnología bastante reciente e innovadora, que combina elementos de la realidad con otros digitales. Entre otras técnicas, hacen uso de cámaras con algoritmos especiales, por lo que, mientras la cámara del dispositivo captura la realidad, se generan entidades situadas entre las imágenes visualizadas y el usuario. Todo esto requiere de un *giroscopio*, que ayude a la hora de detectar la orientación el dispositivo.

Actualmente, la realidad aumentada tiene multitud de aplicaciones: desde aportar información adicional sobre elementos reales que vemos, hasta la integración de elementos 3D que simulan la existencia real de estos. Como son aplicaciones con mucho potencial, el mundo de los videojuegos ha sabido aprovechar esta tecnología para integrarla en los entornos de desarrollo. Unity en este caso, ofrece múltiples opciones para añadir las herramientas de realidad aumentada en su plataforma. Para ello dispone de varios paquetes o extensiones de terceros, que ha conseguido hacer 100% compatibles, dando facilidades. Vamos a analizar a continuación los más importantes, que pueden ser de gran utilidad en el desarrollo del proyecto. Cabe decir que todos ellos, son Kits de desarrollo de software (*SDKs*) que integran características especializadas. Se puede destacar:

- **Vuforia** (Vuforia, Wikipedia, 2018): herramienta multiplataforma que principalmente permite reconocer imágenes y objetos 3D simples y realizar un seguimiento espacial de ellos en tiempo real. Por tanto, se trata de una herramienta de realidad aumentada basada en un objeto de referencia o marcador sobre el cual se pueden generar elementos virtuales en 3D. Estos objetos son visualizados a través de la cámara del dispositivo, simulando su existencia en el mundo real. El SDK soporta también, desde sus últimas versiones, objetivos sin marcadores con y sin detección de superficies, configuraciones multiobjetivo, etc.

  Soporta interfaces de programación en los lenguajes: C++, Java, Objective-C++ y .NET mediante una extensión en el entorno de Unity. Vuforia es compatible con una gran cantidad de dispositivos, con sistemas operativos Android, IOS y Windows (UWP). Además, es compatible con gafas especializadas de realidad aumentada, para ofrecer experiencias mejoradas. Por tanto, es una herramienta muy avanzada, pero que para usos comerciales las licencias ascienden a 500€. Algunos ejemplos de aplicación pueden verse en las siguientes figuras 2 y 3:




Figuras 2 y 3: Imágenes en las que se puede ver un ejemplo de realidad aumentada usando Vuforia.

Fuentes: https://bit.ly/2QH4Wx7
https://bit.ly/2SYBGP6

- **ARKit** (Aumented Reality, Xinreality, 2019): Se trata de la plataforma de realidad aumentada de *Apple*, exclusiva para sistemas operativos *IOS 11*. Es un *SDK* muy completo que permite el reconocimiento de superficies a partir de mallas de puntos, cálculo de iluminación y reconocimiento de información de una escena. Todo ello se realiza gracias a una avanzada tecnología que recoge los puntos de interés y tiene sistema de seguimiento de marcadores. Se conoce como *SLAM*. Además de eso, soporta un modo multijugador, que permite que dos dispositivos *IOS* vean los mismos objetos virtuales sincronizados en tiempo real. Es una de las herramientas más potentes actualmente, pero tiene la desventaja de estar limitado a una sola plataforma. Está preparado para múltiples facetas, y tiene una fácil integración en Unity. El funcionamiento de *SLAM* puede verse en la siguiente *figura 4*:

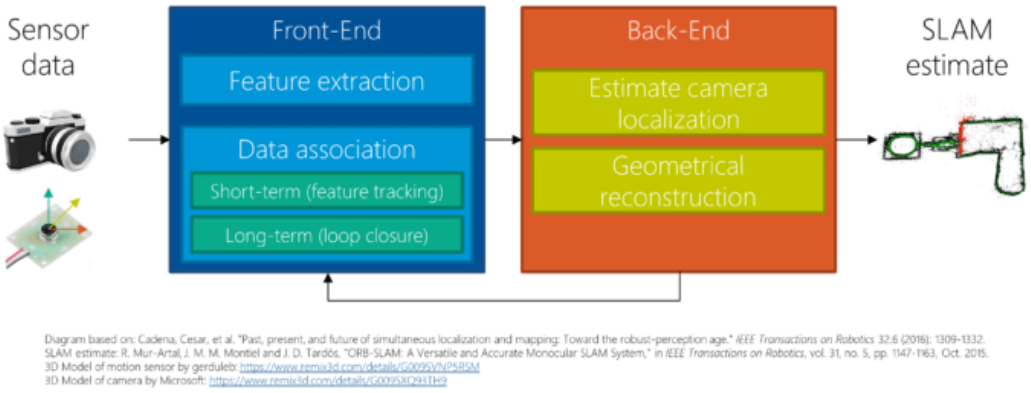


Figura 4: Esquema que resume el funcionamiento de SLAM.
Fuente: https://bit.ly/2BZB0lm

- **ARCore** (ARCore, Wikipedia, 2019): Es el software de desarrollo de Google que permite añadir las funcionalidades de realidad aumentada. Al igual que con *ARKit* de IOS, se caracteriza por tener: seguimiento de marcadores, reconocimiento de escenas y superficies (*figura 5*), así como el cálculo de iluminación. Es la competencia directa del *SDK* anterior ya que soporta las plataformas *Android* 7.0 y *IOS* 11.0 para dispositivos compatibles con *ARKit*. Es una herramienta muy completa, pero todavía le faltan algunas mejoras tecnológicas respecto a su competidor. Por lo que respecta

a la compatibilidad, aun siendo superior a *ARKit*, es muy limitada, ya que únicamente funciona en 50 modelos específicos de *Android* que hay en el mercado en la actualidad (ARCore supported devices, Google, 2019).

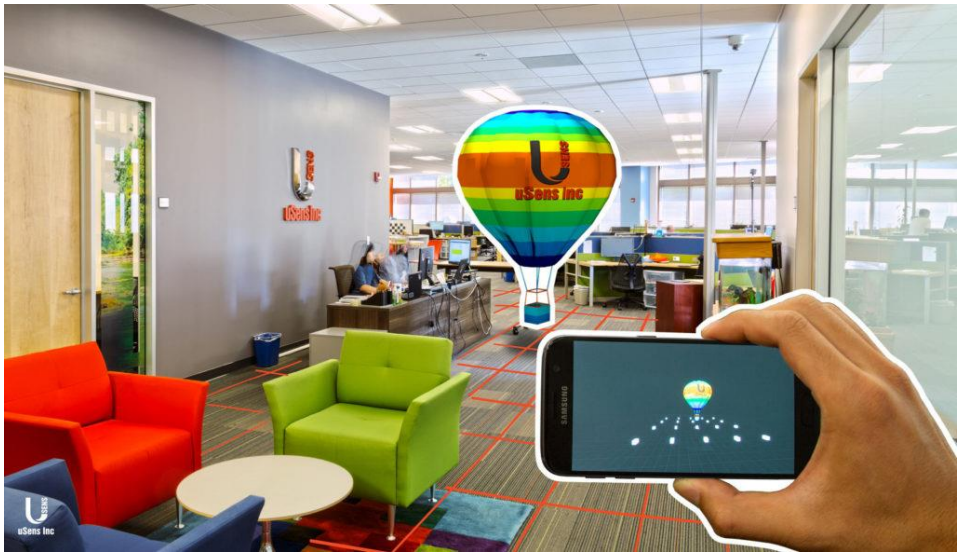


Figura 5: Imagen en las que se puede ver un ejemplo de detección de superficies con SLAM.
Fuente: https://bit.ly/2TtuwT2

Los *SDKs* vistos hasta ahora, son los más avanzados en cuanto a tecnología y soporte por parte de las empresas desarrolladoras. Tienen una mejor adaptación a plataformas como Unity y están en constante evolución.

Sin embargo, si buscamos más herramientas con una inversión menor, pero con sorprendentes capacidades tecnológicas se pueden encontrar múltiples opciones. En general tienen las mismas prestaciones en cuanto a tecnología se refiere, con *SLAM* para reconocimiento de escenas con mapas de puntos, siendo un poco más básico y menos preciso que los comentados anteriormente. Las más destacadas de este tipo son (Herramientas realidad aumentada, Estudio Alfa, 2017):

- **Kudan:** Es uno de los más ambiciosos e interesantes de este sector, ya que ofrece posiblemente la mejor relación calidad/compatibilidad. Reconoce escenas, tiene sistemas de seguimiento, con y sin marcadores, así como una tecnología *SLAM* bastante prometedora.
- **Wikitude:** Competidor directo de *Kudan*, con prestaciones muy similares. Dispone de una gran cantidad de funciones avanzadas incluyendo *SLAM*, de gran precisión.

Se trata de una herramienta cuya versión gratuita incluye marcas de agua en pantalla, por lo que es necesario adquirir una licencia de pago para conseguir resultados profesionales.

- **MAXST:** Nueva herramienta *AR* (realidad aumentada) que busca imponerse frente al resto. Utiliza videos de demostración frente a sus rivales directos, para demostrar que es más preciso en algunas técnicas de seguimiento. Sin embargo, le queda mucho por avanzar, principalmente en la integración con plataformas de desarrollo y exportación.
- **etc.**

Como conclusión, se puede ver que hay una gran cantidad de tecnologías y algoritmos implicados en el funcionamiento de las herramientas de Realidad Aumentada. Simplemente debemos fijarnos en las plataformas y sus respectivas tecnologías, en base a la aplicación o videojuego a desarrollar.

2.4. Juegos de Realidad Aumentada en el mercado

Los juegos de realidad aumentada siempre han llamado la atención. Es posible que en algunos casos el éxito cosechado no haya sido el mejor, pero siempre dan pie a repercusión mediática, al ser una experiencia enriquecedora y novedosa. Actualmente existen bastantes juegos con realidad aumentada, sobre todo enfocados a plataformas móviles. Pero los ejemplos más destacados hasta la actualidad son: *Invizimals* y *Pokemon Go*.

- **Invizimals** (Invizimals, Wikipedia, 2018): Se trata de un videojuego de realidad aumentada desarrollado por la compañía española *Novarama* para la plataforma PSP. Se lanzó el 13 de noviembre de 2009 e incluía una cámara y una tarjeta para permitir este efecto. Por ello, la realidad aumentada de este juego está basada en el uso de la tarjeta como marcador, en la que se generan modelos en 3D, que aparecen simulando su existencia en el mundo real.
    - El objetivo del juego es capturar todos los tipos de *Invizimals* que vamos encontrando a lo largo del juego, con la posibilidad de luchar contra ellos. La *figura 6* refleja mejor este concepto:

Figura 6: Imagen del videojuego Invizimals, aprovechando la AR.
Fuente: https://bit.ly/2EwXSf5

En cuanto la cámara enfoca y detecta el marcador, el juego genera la escena correspondiente, adecuada en contexto: con las criaturas, efectos especiales, animaciones, indicadores, etc.

Cuando el juego salió a la venta, supuso una revolución en la forma de interactuar con los elementos virtuales, ya que no se había explotado prácticamente nada este tipo de tecnología hasta la fecha. Además, el hecho de tratarse de un juego para plataformas portátiles daba mucha versatilidad a la hora de jugar.

Por tanto, se puede destacar que fue un juego con una gran repercusión. Sin embargo, tener colocada la cámara en la PSP y enfocar en todo momento la tarjeta, se volvía en algunos casos un inconveniente, por tener muchos elementos clave separados y de forma modular, de los que dependía el juego. Puede verse en la siguiente *figura 7:*

Figura 7: Invizimals visto desde la consola, con la cámara insertada.
Fuente: https://bit.ly/2CnVKF3

La cámara hacía más difícil la portabilidad de la consola, además de tener siempre presente el marcador.

El juego siguió evolucionando, sacando una nueva entrega para la misma consola con algunas mejoras y más criaturas.

Más adelante apostaron por juegos de *Invizimals* que no Integraban realidad Aumentada en plataformas como *Android*, *Iphone* y *PS3*. Pero, finalmente en 2013 y 2014, lanzaron hasta dos juegos nuevos para *PSVita* (*la Alianza* y *la Resistencia*, respectivamente), con muchas funciones nuevas de interacción para los jugadores, aprovechando innovadoras tecnologías que integraba la consola. En este caso, la consola ya tenía integrada la cámara, por lo que la experiencia era mucho mejor que en la *PSP* de las primeras entregas. Además, tenía una tecnología de reconocimiento de superficies, aunque seguía dependiendo mayoritariamente de los marcadores para mayor precisión, tal y como vemos en la siguiente *figura 8*:

Figura 8: Captura en la que puede verse la versión de PSVita del juego Invizimals: La Alianza.
Fuente: https://bit.ly/2ReIf2R

- **Pokemon Go** (Pokémon GO, Wikipedia, 2019): Se trata de un juego de realidad aumentada para plataformas móviles, desarrollado por la compañía estadounidense *Niantic* con Unity. Se lanzó en España con fecha 15 de julio de 2016. Este videojuego, a diferencia del *Invizimals*, tiene dos tipos de realidad aumentada integrados:
    - Por un lado, el mapa de navegación correspondiente al del mundo real, pero con información adicional referente al juego. La geolocalización es un elemento clave dentro del juego, por lo que mantener informado al usuario en todo momento es clave.
    - Por otro lado, tenemos la visualización de los modelos 3D en el mundo real gracias a la cámara. Este efecto es muy similar al generado en el videojuego anterior. Sin embargo, existe la posibilidad de desactivar la cámara.

  El juego tiene como objetivo, encontrar y capturar a los Pokémon repartidos por el mundo en diferentes zonas, con un factor de aleatoriedad. Para ello, obliga al jugador a desplazarse por la calle para encontrar zonas en las que se esconden estas criaturas. Una vez encontramos un Pokémon, existe la posibilidad de entrar en combate para capturarlo, por lo que entra en juego el segundo factor de realidad aumentada. Aunque existe la opción de desactivarlo, se puede visualizar (en esta fase del juego)

el modelo 3D de este personaje integrado en el mundo real gracias a la cámara, y que dependiendo de la orientación del dispositivo podremos visualizarlo o no.

Este último sistema de realidad aumentada es mucho más sencillo que el utilizado en *Invizimals*, ya que no existe *SLAM* u otro sistema de detección de superficies basado en puntos de interés. Simplemente funciona con una estimación de altura y posición predeterminadas respecto al giroscopio del teléfono, a la hora de colocar a la criatura (puede verse en la *figura 9*). Por ello, en muchos casos no da el efecto deseado, como sería su colocación en superficies planas. Sin embargo, tiene la ventaja de no depender de un marcador en todo momento para jugar, como ocurre en *Invizimals*.

Figura 9: Imagen de Pokémon Go, en la fase de captura con realidad aumentada.
Fuente: https://bit.ly/2A9dFh9

Cuando el juego salió causó una repercusión a nivel mundial nunca vista. Los responsables de este impacto, además de la fama de *Pokémon*, fueron las características de realidad aumentada, el hecho de ser un juego gratuito y la plataforma objetivo para la que funcionaba, como es Android e IOS. Este último factor es de gran importancia ya que, actualmente, casi todo el mundo dispone de un móvil que soporta una de las dos plataformas, por lo que se dispone de inicio de una gran base de jugadores disponibles. Además, se trata de dispositivos muy potentes que integran ya toda la tecnología necesaria para poder jugar (GPS, cámara, giroscopio, etc.), por lo que es muy práctico y sencillo.

Por estos motivos, es una plataforma muy interesante de utilizar a la hora de desarrollar a aplicaciones y juegos. Algunas capturas desde el dispositivo móvil se representan en la *figura 10*:

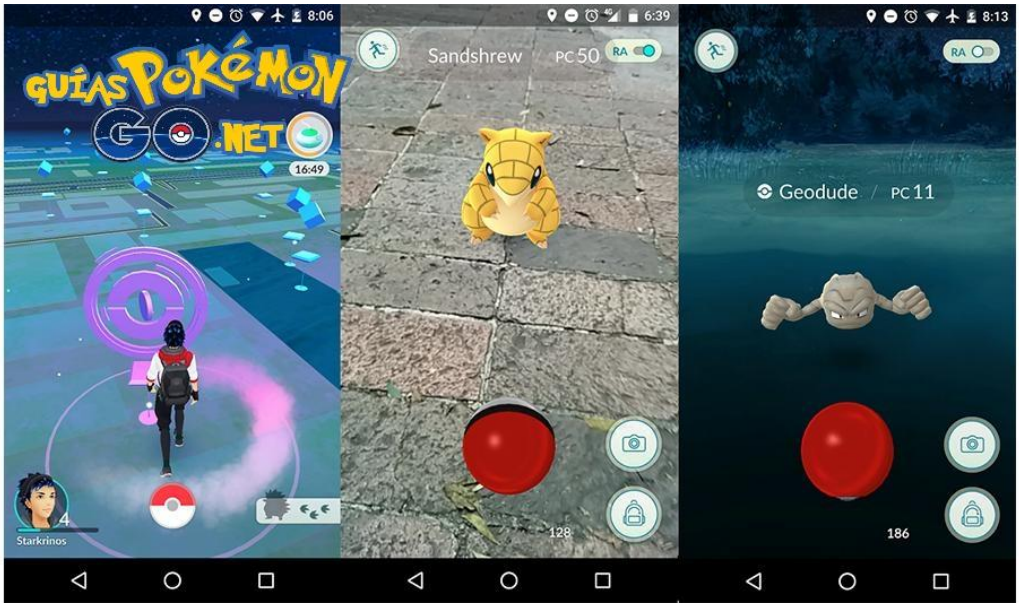


Figura 10: Captura de pantalla con los distintos componentes y fases de AR en Pokémon Go.
Fuente: https://bit.ly/2rO6eYd

Pokémon Go ha seguido evolucionando mucho desde sus comienzos hasta la actualidad, pero las funciones de realidad aumentada en esencia siguen siendo las mismas.

Cabe destacar que el motor de videojuegos usado en este caso es Unity, como el juego que se va a desarrollar en este proyecto.

- **Ingress** (Ingress, Wikipedia, 2019): Se trata de otro juego de realidad aumentada para plataformas móviles y desarrollado por la compañía *Niantic*, siendo considerado como el predecesor de *Pokemon GO*. Se trata de un juego de rol bélico, táctico y de estrategia online que utiliza realidad aumentada en base a los mapas y la ubicación GPS de los dispositivos (*figura 11*). Tiene una historia de fondo de ciencia ficción, en base a un mundo futurista. La principal diferencia de este respecto a *Pokémon GO*, es el tipo de combate, que consiste en hackear y atacar zonas de interés repartidas por el mundo. Este objetivo se realiza contribuyendo a uno de los dos

equipos del juego, que debemos escoger al comienzo. Los equipos y los jugadores que los integran debeb competir entre ellos, para así dominar más zonas que el rival.

Figura 11: Imágen donde se muestra el juego Ingress, predecesor de Pokemon Go
Fuente: https://bit.ly/2GVSncA

Sin embargo, los combates se realizan directamente desde la escena del mapa, a diferencia de *Pokémon Go*, en el que encontramos una escena diferente para ello. En este aspecto de realidad aumentada, vemos que es inferior por usar solo la geolocalización para ello, mientras que Pokémon usa 2: la escena del mapa y la escena de la cámara con el *Pokémon* a capturar.

Como conclusión a los juegos analizados, se puede decir, que los tres son muy interesantes en cuanto a las tecnologías utilizadas. Los dos últimos juegos analizados comparten características basadas en el mapa, sin embargo, con *Invizimals* difieren bastante en este punto. Por otro lado, *Invizimals* y *Pokémon* no difieren tanto en el aspecto temático ya que el objetivo siempre es capturar criaturas de ficción, simulando su integración en el mundo real.

De todos ellos, se pueden extraer muchas buenas ideas y tomar como referencia algunos aspectos para el videojuego a desarrollar. Pero como punto en común se pueden mencionar las plataformas portátiles (principalmente smartphones) y las tecnologías de realidad aumentada a implementar.

3. OBJETIVOS Y JUSTIFICACIÓN

En este apartado hablaremos de las metas que se pretende alcanzar con este proyecto. Una vez analizadas, pasaremos a describir el motivo de estas metas, que justifican la elección de este proyecto respecto a otros.

El objetivo de este proyecto es realizar un juego utilizando Unity, plataforma de desarrollo en la que me gustaría trabajar más de lo ya visto durante la carrera. De esta forma podré aprender mejor cómo funciona el entorno y contrastar el proceso de realización de los juegos, respecto a otros métodos. Además, Unity es un motor muy usado actualmente por las empresas de videojuegos, por lo que aprender una base será de gran utilidad.

Por otra parte, este documento nos sirve para reflejar algunos de los objetivos cumplidos en el desarrollo, explicando los pasos llevados a cabo, así como las ideas y su planificación previa.

Analizando metas del proyecto, los objetivos concretos pueden identificarse como:

- Aprendizaje de uso del entorno de Unity y sus distintas funcionalidades.
    - Conocimiento genérico de las funciones que ofrece y su distribución.
    - Uso del modelo basado en componentes.
    - Programación en C#.
    - Scripts para control de giroscopio y otras funciones avanzadas.
    - Integración de plugins (funciones adicionales) y/o librerías útiles.
    - Modularización correcta de código para una posible adaptación multiplataforma dentro del entorno de móvil.
    - Aprender a usar las herramientas integrables de Realidad Aumentada.
- Concreción de idea/concepto de juego.
- Realizar especificación de requerimientos.
- Establecer y fragmentar requerimientos en tareas.
- Estimación de tiempos a partir de las tareas y subtareas (presupuesto).
- Programar mecánicas base del juego, adaptándolas principalmente a Android y al entorno de realidad virtual.
- Establecer físicas y sistema de colisiones en el juego.
- Añadir Inteligencia artificial y otros aspectos avanzados.
- Mejorar modelos y entornos del juego (parte visual).

- Corrección de errores y mejoras de rendimiento.
- Mejoras en mecánicas y funcionalidades.

Además de los objetivos comentados, se pueden comentar aquellos objetivos orientados al final del proyecto, en relación con su orientación al público y las plataformas disponibles para subirlo. Muy probablemente, el juego será publicado en la tienda de Google Play Store, para dispositivos Android. Con esto hay que tener en cuenta el coste de la publicación del proyecto, así como las librerías y licencias empleadas, que supone unos 25€ de coste para la subida. En relación con la App Store de Apple, en principio queda descartado por suponer un coste mayor de unos 100€ anuales.

Por lo que respecta a la justificación, se trata de un proyecto que siempre me había interesado. Desde hace tiempo, quería hacer un juego de forma individual que me diera la posibilidad de aprender muchos conocimientos útiles, desarrollando prácticamente, todas las partes que lo componen por igual. Sin embargo, no tenía clara la idea de juego hasta el último momento. Quería que se tratara de un juego para plataformas móviles y con algún componente especial. Hacer un juego que combine elementos de la realidad con el mundo digital, es un factor diferenciador, con tecnologías innovadoras y en desarrollo constante. El uso de librerías especializadas ayuda mucho y siempre es interesante ver su estructura y funcionamiento, cuando se integra en un proyecto. Por ello, me decanté por este tipo de juego basado en la realidad aumentada.

Cabe decir que también he elegido este tipo de proyecto por ser del itinerario de entretenimiento digital y al estar más orientado a lo que me gustaría hacer en el futuro.

4. METODOLOGÍA

Para la gestión del proyecto he optado por utilizar una metodología ágil. En concreto, la metodología escogida es *Kanban* (Kanban, Wikipedia, 2019), por adaptarse mejor a las características y necesidades del proyecto. A continuación, describiremos las características de esta metodología.

Kanban es una estrategia de desarrollo muy visual, basada en la fragmentación de las tareas en tarjetas. Estas tarjetas pueden tener subobjetivos, que concreten la funcionalidad a desarrollar en cuestión. Además, están divididas en columnas en función del estado de proceso en el que se encuentran, así como su ordenación por prioridades y personas involucradas. Esta herramienta permite ver fácilmente los bloques en los que tiene carencias el proyecto, así como los cuellos de botella. Todo esto ofrece la posibilidad de detectar errores y mejorar apartados que más lo requieran.

Además, a la hora de implementar Kanban, se le dará enfoque basado en las funcionalidades a desarrollar o FDD (Desarrollo basado en funcionalidades, Wikipedia, 2019). Se trata de dirigir el proyecto en la realización de varias tareas/mecánicas concretas, que vayan poco a poco ampliando y mejorando el juego en todos sus aspectos. Este enfoque, tiene que ver con el tipo de juego a desarrollar, muy diferente a los basados en prototipos. En este caso, las funcionalidades del juego no son tan concretas como para partir de esta base, por lo que las tareas se comportan como extensiones al juego en funcionalidades.

4.1. Gestión de proyecto

Una vez comentado *Kanban*, hablaremos de las herramientas que se van a emplear para llevarlo a cabo y gestionar el proyecto. En este caso la herramienta a destacar es *Trello*, por disponer de las funciones requeridas por la metodología.

Trello (Trello, 2019) es una plataforma web de gestión de proyectos, basado en tareas/tarjetas. Tiene la ventaja de ser una herramienta extensible, ya que permite la integración de más funcionalidades de las ya disponibles. Nos permite clasificar, las tareas según el estado del desarrollo y priorizar unas respecto a otras. Se pueden encontrar múltiples extensiones de flujo de trabajo, en relación con las distintas metodologías, por lo que se trata de una herramienta muy versátil. Además, ya había sido utilizado con

anterioridad en otros proyectos durante la universidad, por lo que no requiere una nueva curva de aprendizaje.

Otro software de gestión a emplear, que nos será de gran utilidad es *GanttProject* (GanttProject, 2019). Se trata de una herramienta de planificación de proyectos basados en diagramas de Gantt. Con ellos, se puede hacer una mejor estimación gráfica, de las tareas/tiempos y será un añadido a la metodología *Kanban*. A su vez, podremos comparar el diagrama de Gantt inicial con la evolución de las tareas en Trello. En este último, se pueden añadir extensiones que pueden ser de gran utilidad para comprobar los desajustes.

Figura 12: Logo del software de planificación Gantt Project
Fuente: https://bit.ly/2WxoWnH

Debido a la importancia de registro de tiempos, también se empleará *Toggl* (Toggl, 2019) como herramienta compatible con *Trello*, para contabilizar las horas dedicadas a las tareas, mientras se está trabajando en ellas. *Toggl* es una aplicación de seguimiento de tiempo muy útil, que dispone de versión de escritorio, móvil y web, aunque en este caso se utilizará esta última. La forma de combinarlo con *Trello* es mediante un plugin/extensión en el navegador Google Chrome, que permite empezar a contabilizar las horas de la tarea/tarjeta correspondiente en *Trello*. En las siguientes *figuras 13 y 14* podemos ver los logos de estas herramientas:

Figuras 13 y 14: Logo de las herramientas Trello y Toggl.
Fuente: http://bit.ly/2WxoWnH y http://bit.ly/2KfqIUf

4.2. Gestión de versiones

Para el control de versiones, se va a hacer uso de un repositorio *Github* (Github, 2019), sincronizado en todo momento con la carpeta del proyecto en cuestión. Aunque se trate de un desarrollo individual, es una buena costumbre hacer uso de este tipo de herramientas, donde el código está siempre controlado por una serie copias asociadas a versiones del proyecto. Siempre que se detecte que se ha logrado implementar alguna funcionalidad o avance importante, es conveniente subir una nueva versión, para dar constancia de ello y por la seguridad del propio proyecto. En concreto, la aplicación de escritorio a utilizar para el control de versiones es *GitKraken* (GitKraken, 2019), que se sincroniza con el repositorio correspondiente y tiene una interfaz de usuario más intuitiva. Además, ya se ha trabajado con ella en proyectos anteriores. Se puede ver su interfaz en la *figura 15*:

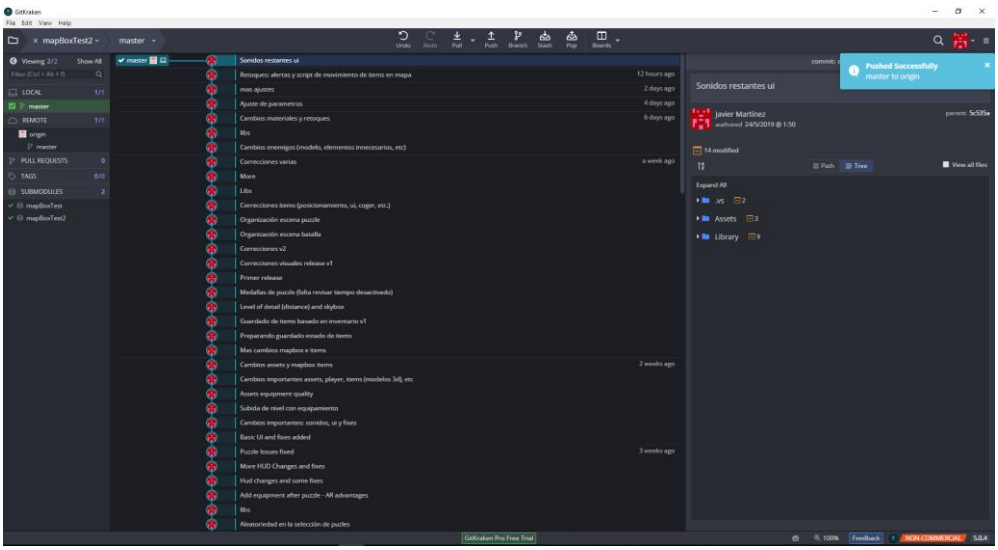


Figura 15: Captura en la que se muestra el uso de Git Kraken para control de versiones
Fuente: Elaboración propia

Otro factor a tener en cuenta es el tamaño del proyecto, que debe ser revisado en todo momento por si se superara un máximo estimado. En ese caso, se aislará la carpeta de *assets* del proyecto, que suele ser la que más espacio consume por contener los recursos externos como: imágenes, modelos 3D, etc.

5. ANÁLISIS DE PROYECTO

En esta sección se van a detallar aspectos clave del proyecto, en lo que a gestión se refiere. En primer lugar, se van a explicar en mayor profundidad, las herramientas de planificación del proyecto, así como el uso que se ha hecho de ellas. Terminaremos hablando de la gestión de riesgos y la importancia que tiene en el proyecto.

5.1. Planificación

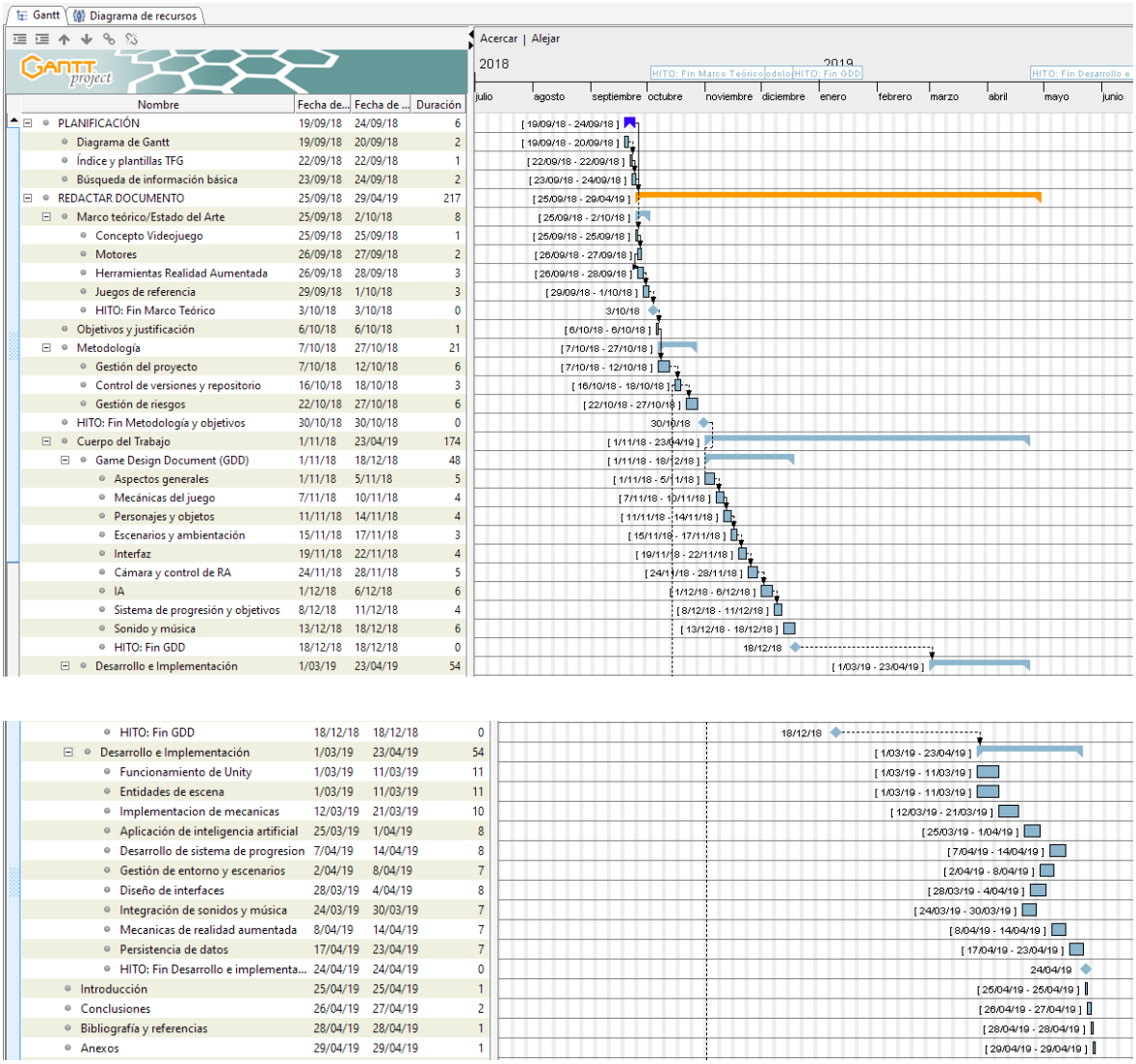
En primer lugar, se van a introducir de las herramientas comentadas en la metodología y el uso que se va a llevar a cabo de ellas para la planificación completa del proyecto. Por ello, esta sección se va a fragmentar en dos partes, en base a las herramientas comentadas anteriormente: *GanttProject* y *Trello*.

5.1.1. GanttProject

Con este programa, se ha realizado la planificación inicial del proyecto, que contempla gran parte de las tareas a desarrollar en base a los requisitos. Este software de código libre permite, como su nombre indica, administrar de proyectos mediante el diagrama de Gantt. La planificación del proyecto se ha dividido en cuatro bloques principales: **Planificación, Redacción de documento, Curva de aprendizaje** y **Desarrollo de proyecto en Unity** (consultar *figuras 16 y 17*). La estimación de esta herramienta se realiza en días.

Para el primer bloque, correspondiente a la Planificación, se ha hecho un análisis de los requisitos, se han tomado referencias de proyectos anteriores y se ha empezado a trabajar con estas herramientas de planificación, una vez obtenida la información necesaria. Con esto, se han estimado unos cinco días para el desarrollo del bloque.

Para el segundo bloque de Redacción del documento, se ha tomado como referencia los distintos puntos que contiene el documento actual, concretamente del índice. Entre ellos, se puede destacar: Marco teórico, Objetivos y justificación, Metodología, GDD, Desarrollo/implementación, Introducción, Conclusiones, Bibliografía y Anexos. Todos estos apartados y sus respectivos subapartados han sido planificados con una cantidad de días. Concretamente, el documento tiene unos 217 días asociados en total. Puede verse la estimación realizada en Gantt Project en las *figuras 16 y 17* a continuación:



Figuras 16 y 17: Capturas en la que se muestra la planificación realizada en Gantt Project (Documento y planificación)
Fuente: Elaboración propia

Para la curva de aprendizaje en Unity, se han estimado unos 23 días, una vez seleccionados los tutoriales y la documentación a emplear.

Finalmente, por lo que respecta al desarrollo del videojuego, se han estimado unos 186 días, aunque es posible que se vea sometido a cambios con el tiempo, pues siempre se encuentran imprevistos o desajustes que afectan principalmente a la implementación del juego o aplicación. Puede consultarse en la siguiente *figura 18*:

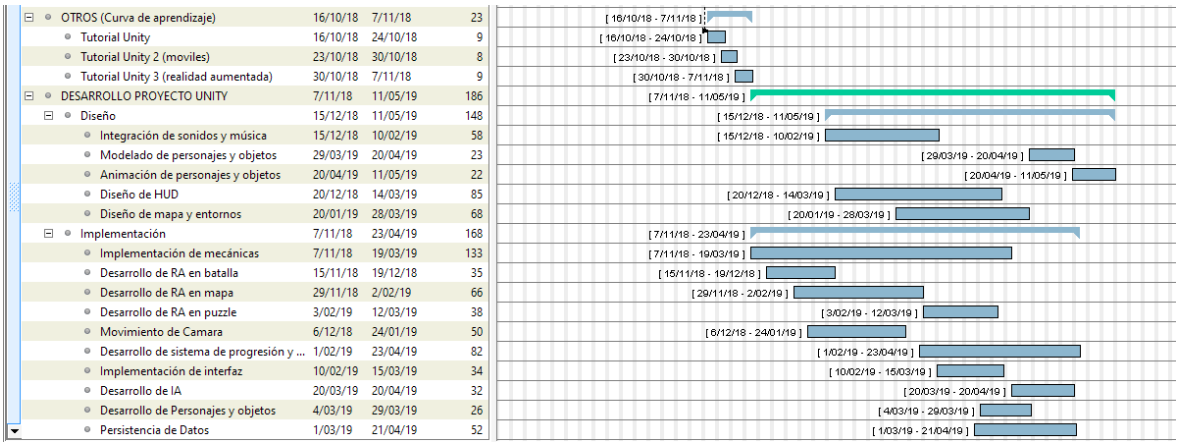


Figura 18: Captura en la que se muestra la planificación realizada en Gantt Project (Curva de aprendizaje y desarrollo en Unity)
Fuente: Elaboración propia

5.1.2. Trello

La otra herramienta empleada para la planificación es *Trello*, que permite llevar un seguimiento activo de las tareas en sus diferentes estados del proceso. Para ello, se han recogido las tareas establecidas en *GanttProject* y se convertido en las tarjetas personalizadas que nos permite crear *Trello* ajustándose metodología *Kanban*.

Para asociar las tareas a los bloques comentados en la herramienta anterior, se han empleado etiquetas y nomenclaturas especificas en las tarjetas de *Trello* (consultar en *figura 19* más adelante).

Por otro lado, para poder controlar en todo momento las diferentes fases por las que pasa una tarea, estas tarjetas pueden colocarse en columnas que representan los diferentes estados. Las columnas creadas son: Pendiente, En proceso, A falta de revisión/retocar, Finalizado y Desechado. En este caso, algunas de las tareas tomadas como referencia en *GanttProject*, se han tenido que fragmentar en tareas más específicas para dividir la carga de trabajo y especificar en cada momento lo que se estaba realizando.

Podemos ver una captura de *Trello* en la *figura 19*, realizada durante el desarrollo del proyecto.

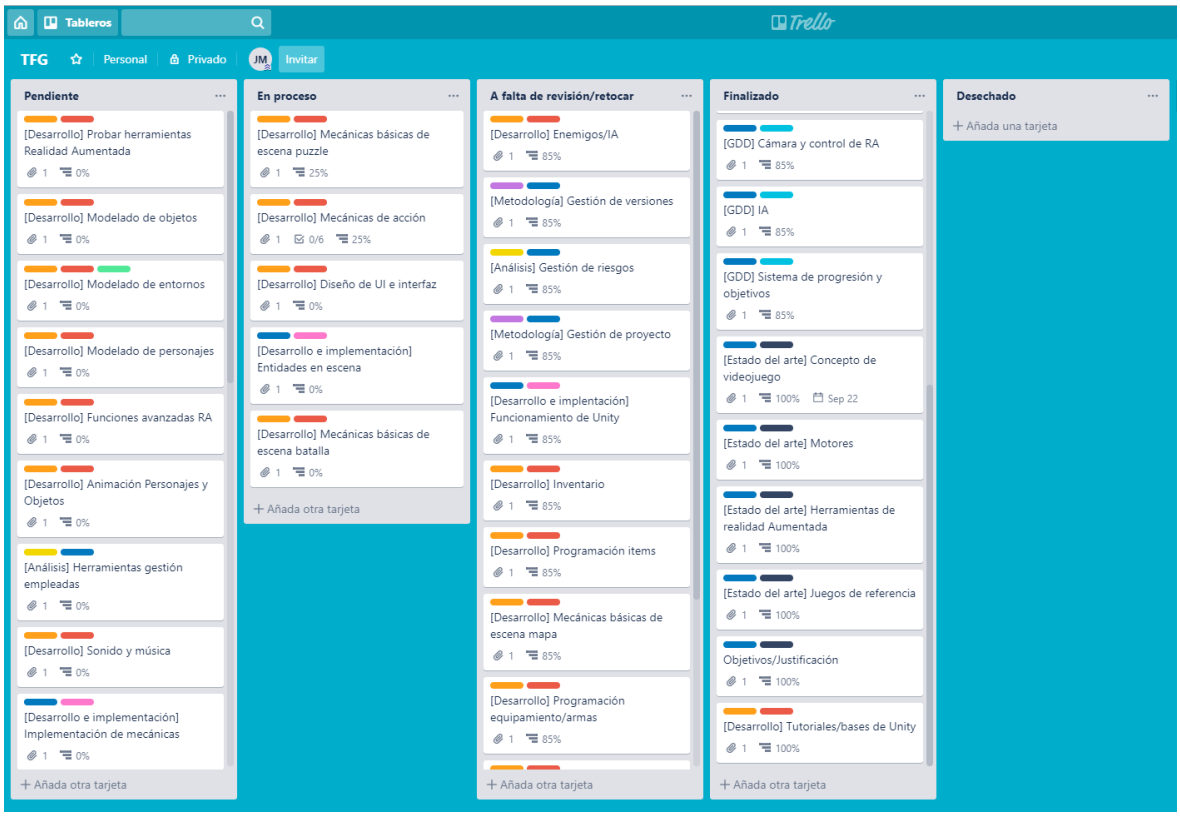


Figura 19: Captura en la que se muestra la planificación/gestión de tareas realizada en Trello, basada en la metodología Kanban
Fuente: Elaboración propia

Además, en esta herramienta se ha añadido el plugin de *Toggl*, que permite contabilizar las horas de cada tarjeta/tarea en base a lo que se esté trabajando en cada momento. De esta forma se complementa el flujo de las tareas con sus respectivos tiempos. Una vez finalizado el proyecto, se establecerá un reporte de las horas totales dedicadas, gracias a la herramienta *Toggl*, aunque se pueden estimar unas 400 horas (aproximadamente).

5.2. Gestión de riesgos

La gestión de riesgos, en cualquier proyecto, es un factor importante y, por ello, hay que dedicar un apartado en el que queden reflejados los posibles problemas que pueden surgir durante el desarrollo (y el impacto que suponen). Este apartado se divide en 4 partes: Identificación, Análisis, Planificación y Monitorización/control.

5.2.1. Identificación

En este primer apartado se van a comentar los posibles problemas con riesgo a aparecer, durante el desarrollo del proyecto.

- **Tecnología:**
  - **Pérdida/borrado de información:** Fallo al guardar archivos clave en alguna de las versiones del proyecto.
  - **Limitación de hardware:** Determinados dispositivos no cumplen los requisitos mínimos, sobre todo, con la potencia que requieren este tipo de aplicaciones, con un procesamiento mayor.
  - **Limitación de software:** Existen librerías de realidad aumentada que solo funcionan a partir de ciertas versiones del sistema operativo. Por ejemplo, AR Core de Google, solo funciona a partir de versiones 7.0 de Android.
  - **Errores de compatibilidad:** Fallo a la hora de sincronizar versiones de los distintos programas de desarrollo (Unity, Blender, librerías, etc.).
  - **Los entornos de desarrollo ocultan ajustes e información avanzada:** Principalmente, afecta a los ajustes de compilación del motor, así como funciones preestablecidas. La documentación de Unity será un aspecto clave.
  - **Errores desconocidos o fallos en las herramientas de desarrollo:** Se trata de fallos internos en las aplicaciones de desarrollo, que obstaculicen el avance del proyecto.

- **Organización:**
  - **Grado de detalle en la planificación de proyecto:** Aspecto a tener en cuenta para tareas muy grandes dentro del proyecto. Debemos dividirlas en subtareas más fácilmente gestionables para la planificación.
  - **Retrasos y desviaciones respecto a la planificación:** Durante el desarrollo existen casos en los que se supera o incluso se acortan, las horas previstas para realizar las funcionalidades.
  - **Tamaño del proyecto:** Contemplar hasta donde se puede llegar, teniendo en cuenta el tiempo de realización y un único desarrollador en el proyecto.

- **Personal**:
    - **Falta de comunicación:** Es un factor clave aunque se trate de una sola persona. La revisión del proyecto cada cierto tiempo es fundamental, para detectar errores y aspectos de mejora (en muchos casos no detectados por el propio desarrollador).
    - **Enfermedad:** Aspecto que puede llegar a afectar al proyecto, por la indisposición del único componente del proyecto.

- **Requerimientos**:
    - **Curva de aprendizaje:** Existen determinadas librerías y herramientas con las que no se está familiarizado. Esto puede suponer un tiempo añadido a las horas presupuestadas, contemplado como aprendizaje o formación básica.
    - **Documentación y tutoriales del software para el desarrollo:** Está relacionado con las facilidades de las herramientas de desarrollo, para explicar cómo se utilizan y las funciones preestablecidas que incluyen.
    - **Rango de edades a las que está orientado el producto final:** El público al que va destinado, en un principio, debe abarcar el mayor rango posible. Poco a poco, el rango de edades puede verse reducido respecto a lo planificado.
    - **El proyecto final cumpla con las modelos y estándares preestablecidos:** Se refiere al cumplimiento de las funcionalidades indicadas y su correcto funcionamiento.
    - **Entrega en el plazo establecido:** Es uno de los principales requerimientos, basado el tiempo establecido. No cumplirlo supone el fracaso del proyecto.

- **Herramientas:**
    - **Errores desconocidos o fallos en las herramientas de planificación:** Se trata de fallos internos en las aplicaciones de planificación, que supongan una pérdida de información o errores a la hora de realizar algunas funciones del programa. La pérdida de la información planificada puede suponer una pérdida de tiempo importante, al reestructurar de nuevo la planificación general.

- **Errores en sincronización de archivos (repositorio):** Errores relacionados con el uso del repositorio Github, que pueden llegar a suponer pérdidas de tiempo e incluso pérdidas de información.

- **Estimación:**
  - **Realización de tareas fuera del plazo planificado:** Supone realizar reajustes en la planificación del proyecto (adelantos y atrasos). Afecta al desarrollo normal del proyecto y de no corregirse, puede ser grave.
  - **Hacer más horas de las estimadas (estimación errónea):** Reajustes en la planificación, que pueden desencadenar en el punto anterior.
  - **Hacer menos horas de las estimadas (estimación errónea):** Reajustes en la planificación que, aun siendo positivos, desajustan las tareas planificadas.

5.2.2. Análisis

En la siguiente tabla se pueden ver los riesgos, con sus respectivas probabilidades y efectos en el proyecto. Ordenadas por efectos/seriedad, tenemos:

Tipo de riesgo	Riesgo	Probabilidad	Efectos
Requerimientos	Entrega en el plazo establecido	Baja	Catastrófico
Personal	Falta de comunicación	Muy baja	Catastrófico
Organización	Tamaño del proyecto	Baja	Catastrófico
Organización	Retrasos y desviaciones respecto a la planificación	Alta	Serio
Requerimientos	El proyecto final cumpla con los modelos y estándares preestablecidos	Moderada	Serio

Tipo de riesgo	Riesgo	Probabilidad	Efectos
Estimación	Realización de tareas fuera del plazo planificado.	Moderada	Serio
Tecnología	Pérdida/borrado de información	Baja	Serio/tolerable (teniendo en cuenta la cantidad perdida)
Estimación	Hacer más horas de las estimadas	Muy alta/alta	Serio/tolerable (teniendo en cuenta la cantidad de horas)
Herramientas	Errores en sincronización de archivos (repositorio)	Alta	Tolerable
Tecnología	Errores desconocidos o fallos en las herramientas de desarrollo	Baja	Tolerable
Herramientas	Errores desconocidos o fallos en las herramientas de planificación	Baja	Tolerable
Tecnología	Errores de compatibilidad	Moderada	Tolerable
Tecnología	Los entornos de desarrollo ocultan ajustes o información avanzada	Moderada	Tolerable
Organización	Grado de detalle en la planificación del proyecto	Alta	Tolerable
Requerimientos	Curva de aprendizaje	Alta	Tolerable

Tipo de riesgo	Riesgo	Probabilidad	Efectos
Requerimientos	Documentación y tutoriales del software para desarrollo	Alta	Tolerable
Personal	Enfermedad	Moderada	Serio/Tolerable (en función de la gravedad de la misma)
Estimación	Hacer menos horas de las estimadas	Moderada	Tolerable
Requerimientos	Rango de edades a las que está orientado el producto final	Moderada	Tolerable
Tecnología	Limitación de hardware	Moderada	Tolerable
Tecnología	Limitación de software	Moderada	Tolerable

Tabla 1: Clasificación de los riesgos, ordenados por seriedad y teniendo en cuenta la probabilidad.

5.2.3. Planificación

En base a la tabla anterior, ordenada por la importancia de los riesgos, se puede establecer una estrategia de planificación a la hora de abordar estos posibles casos.

Las estrategias se dividen en:

- Prevención: Con el fin de evitar que se produzca el riesgo.
- Minimización: Disminuir el impacto del riesgo en caso de producirse.
- Plan de contingencia: Prepararse antes de que sea irreversible/muy grave.

Se han planteado las siguientes estrategias para abordar los riesgos de mayor gravedad:

- **Tamaño del proyecto (Organización):**
    - Prevención: Con tal de evitar este problema, hay que llevar un control de versiones en todo momento, valorando si la estimación de tiempos es

correcta. Con el control de la planificación, se puede ver si se requieren de más o menos funcionalidades para que el proyecto se ajuste a las horas y requisitos necesarios.

- Minimización: En caso de necesitar más tareas para el tamaño del proyecto, se debe agilizar el desarrollo, con funciones sencillas pero efectivas. En caso contrario, se debe considerar la posibilidad de descartar las funcionalidades más prescindibles.
- Plan de contingencia: Se debe recortar en funcionalidades en caso de superar el tamaño y hacer horas extra, aunque supongan un coste adicional para el proyecto.

- **Falta de comunicación (Personal):**
  - Prevención: Reuniones con el tutor cada dos semanas, con fin de revisar el trabajo realizado hasta el momento.
  - Minimización: Las reuniones se harán con más frecuencia de la habitual, hasta que el desarrollo vuelva a estar encaminado de nuevo.
  - Plan de contingencia: Llamada de atención. Revisión intensiva y charla con el profesor para revisar los problemas externos/internos que hayan podido afectar al desarrollo.
- **Entrega en el plazo establecido (Requerimientos):**
  - Prevención: Tratar de ceñirse a la planificación dentro de lo posible.
  - Minimización: Para evitar más problemas con la entrega, hay que definir una planificación concreta, con fechas clave. De esta forma, los retrasos en la planificación deben tratar de rectificarse con “pseudoentregables”, en los que el mínimo debe estar en funcionamiento. Además, los desajustes, deben ser hablados con el tutor, para recuperar dentro de lo posible.
  - Plan de contingencia: Renunciar a funcionalidades, con tal de llegar con los requisitos principales a la entrega.
- **Retrasos y desviaciones respecto a la planificación (Organización):**
  - Prevención: Estas desviaciones, con bastante probabilidad se van a producir. Ya sea por la mala planificación de tareas o por otros motivos, ajenos al proyecto que interfieran en su desarrollo normal. Las revisiones periódicas y la planificación son claves para evitar que vaya a más.

  - Minimización: Para evitar que derive en retrasos en la entrega, la comunicación es esencial, para detectar los problemas que afecten a estos casos.
  - Plan de contingencia: Descartar tareas que requieran de muchas horas sin la importancia suficiente. Por el contrario, si es importante, se debe priorizar su desarrollo lo antes posible.
- **El proyecto final cumpla con los modelos y estándares preestablecidos (Requerimientos):**
  - Prevención: Tratar de ajustarse a la especificación inicial del proyecto, respetando las ideas base y manteniendo el uso de las herramientas de desarrollo comentadas. Hacer uso de la documentación y preguntar en caso de dudas.
  - Minimización: Buscar librerías o funciones alternativas, reemplazando las anteriores, con tal de seguir respetando las bases del proyecto.
  - Plan de contingencia: Justificar los cambios realizados en el documento, explicando el porqué de las desviaciones respecto al modelo.
- **Realización de tareas fuera del plazo planificado (Estimación):**
  - Prevención: Para evitar este problema, la planificación nuevamente es clave.
  - Minimización: Hacer más horas para recuperar tiempo, aunque suponga un reajuste en la planificación.
  - Plan de contingencia: En este caso, la única opción sería hablar con el tutor de las posibilidades para la siguiente convocatoria.
- **Pérdida/borrado de información (Tecnología):**
  - Prevención: Uso responsable de las herramientas de sincronización en repositorios. Realizar copias locales cada semana de trabajo como mínimo.
  - Minimización: Aumentar la frecuencia de copias y en varios dispositivos.
  - Plan de contingencia: Volver a una versión anterior del repositorio, para tratar de minimizar las pérdidas.
- **Hacer más horas de las estimadas (Estimación):**
  - Prevención: Tratar de ajustarse a la planificación y revisar posibles desviaciones (cambios).
  - Minimización: Reajustar tareas y priorizar el desarrollo de las tareas más costosas.

- Plan de contingencia: Descartar funcionalidades para que no afecten a las entregas.

5.2.4. Monitorización/control

- **Tamaño del proyecto (Organización):**
  - En la actualidad, la probabilidad de que ocurra es media, ya que, se tiene una idea global del proyecto, pero no se conoce hasta qué punto puede extenderse. Los indicadores del riesgo pueden ser las grandes desviaciones de las horas en todas las tareas respecto a la planificación. Para evitarlo, hay que revisar con frecuencia las horas asignadas por tarea en la planificación final y así valorar respecto a las horas disponibles.
- **Falta de comunicación (Personal):**
  - La probabilidad de este riesgo es muy baja, ya que se lleva un ritmo constante de trabajo y de reuniones con el tutor. De esta forma, la comunicación es frecuente y el control del flujo del proyecto también. Los indicadores pueden ser la falta de revisiones del proyecto, junto con la falta de reuniones.
- **Entrega en el plazo establecido (Requerimientos):**
  - El riesgo es bajo, pues es obligatorio entregar en proyecto en la fecha límite y se está siguiendo una planificación para que no ocurra. Los indicadores pueden detectarse a partir de las desviaciones en la planificación (mala planificación o poco trabajo).
- **Retrasos y desviaciones respecto a la planificación (Organización):**
  - Se trata de un riesgo probable, que con seguridad se dará en más de una ocasión. Viene dado principalmente por la curva de aprendizaje y el desconocimiento del uso de algunas herramientas. Para evitar estos retrasos, hay que contemplar en la planificación un tiempo de aprendizaje razonable (no excesivo) y que no suponga retrasos en las entregas.
- **El proyecto final cumpla con los modelos y estándares preestablecidos (Requerimientos):**
  - Este riesgo tiene una probabilidad baja, debido a que los objetivos base del juego, se van a trabajar mucho durante el desarrollo. La planificación contempla un mayor tiempo de desarrollo de algunas funcionalidades, al

orientarlo a la realidad aumentada. Indicadores: si se revisan a lo largo del desarrollo los estándares, se podrá comprobar las desviaciones respecto a lo planificado.

- **Realización de tareas fuera del plazo planificado (Estimación):**
    - Riesgo directamente relacionado con los retrasos y desviaciones. La probabilidad es alta, pero se tratará de minimizar, pues los plazos son importantes de cumplir, antes de desencadenar en retrasos de entrega. Indicadores: Comparar respecto a la planificación e identificar con que frecuencia se producen estos retrasos.
- **Pérdida/borrado de información (Tecnología):**
    - Actualmente, poco probable, por el uso de repositorios para el control de versiones, y copias de seguridad de archivos locales (cada semana). Indicadores: Detectar un mal funcionamiento del ordenador, nos puede llevar a tomar medidas más estrictas.
- **Hacer más horas de las estimadas (Estimación):**
    - Muy probable, en relación con los retrasos y desviaciones. Se tomará el mismo tipo de medidas ya comentadas en estos, así como los indicadores (bastante similares).

6. DOCUMENTO DE DISEÑO DEL VIDEOJUEGO (GDD)

6.1. Concepto

El GDD es uno de los bloques principales del cuerpo del trabajo y del documento en sí, por tratar todos los aspectos que definen el proyecto a desarrollar. Se puede definir como el "*documento de diseño cuyo contenido es altamente descriptivo acerca del videojuego a desarrollar*" (GDD, Wikipedia, 2019). Por tanto, pretende ofrecer una visión global, analizando cada uno de los apartados que lo conforman, incluyendo conceptos teóricos, diseño y aspectos de programación. Todo ello, con tal de definir la idea final del juego.

6.2. Aspectos generales

El nombre del videojuego es *AR Hunters*, por dos motivos: AR son las siglas de realidad aumentada y Hunters, porque en el juego hay que cazar/derrotar enemigos aprovechando esta componente para avanzar.

Se trata de un juego para un jugador, del género **acción/aventuras** orientado a **plataformas móviles**, en el que controlaremos a un personaje que tiene como objetivo, encontrar y acabar con las naves robots enemigas, así como encontrar el origen de estas desconocidas amenazas. Para ello hará uso de armas durante el combate y aprovechará su destreza para resolver puzles que poco a poco lo acercarán a su objetivo final.

Este juego, además, estará basado en el factor de la **realidad aumentada**, por lo que se empleará un sistema de geolocalización, con el que podrá ir encontrando diferentes eventos o puntos clave a su alrededor. Estos puntos pueden ser: puzles, zonas de combate u objetos. Además, el sistema de combate incluirá otra componente de realidad aumentada, mediante el uso de la cámara. Con ella, el jugador podrá apuntar a los distintos objetivos que aparezcan, y así enfrentarlos con sus armas a distancia. El hecho de disparar a los enemigos cercanos, le añade una componente de ***shooter* en primera persona**. Para finalizar existirá una escena puzle que emplee la cámara y otra componente de realidad aumentada, para resolver los puzles planteados. Por tanto, se pueden distinguir tres tipos escenas principales: la de movimiento del jugador por el mapa, la de combate y la de puzle.

Finalmente, el juego incluirá un sistema de progresión basado en el nivel del jugador, con el que se irá incrementando la dificultad y las herramientas disponibles a desbloquear. Este sistema de progresión se explicará más adelante en el apartado 6.12.

6.3. Propósito y público objetivo

El propósito final del proyecto, en relación con su orientación, es que el juego será publicado en la tienda de Google Play Store, para dispositivos Android. De esta forma, tendrá una mayor visibilidad que por otros medios.

Pretende ser un juego competitivo a la hora de subir de nivel y que combine elementos de aventura y acción con la realidad aumentada como factor diferenciador. También es un juego que incentiva a moverse en el mundo real para alcanzar los objetivos del juego con mayor rapidez, por su sistema de ubicación.

Por otro lado, a la orientación al público, estará destinado a un rango de edades bastante amplio. Mientras las funcionalidades de realidad aumentada estén lo suficientemente explicadas y justificadas, el rango podría ser entre 7 y 70 años. Por tanto, hay que tener en cuenta este factor explicativo en el juego, para que el jugador no tenga ningún tipo de duda a la hora de realizar las funciones básicas y no derivar en alguna posible confusión.

6.4. Requisitos técnicos

En este apartado, vamos a comentar algunos requisitos técnicos, que afectan al juego a desarrollar.

En primer lugar, se trata de un juego para plataformas móviles, con sistemas operativos Android, cuya versión debe ser superior a 7.0. Esto se debe principalmente a los requisitos propios de las librerías empleadas, como *MapBox* (Mapbox, 2019) que se explica con más detalle en el punto 6.9. La compatibilidad con otras plataformas móviles como IOS está todavía por comprobar, ya que es posible que algunas librerías tengan ciertas limitaciones.

Por tanto, en un principio se establecen los requisitos como: Android 7.0 o superior como sistema operativo y un dispositivo móvil con las tecnologías principales para visualizar correctamente los modelos 3D integrados con la realidad: la cámara y los sistemas de ubicación GPS. Sin estos dos últimos, el juego no funcionará correctamente ya que son los elementos que hacen posible la realidad aumentada, en la cual está basada el juego.

6.5. Mecánicas

El jugador podrá realizar una gran cantidad de funciones, a la hora de interactuar con los elementos y escenarios del videojuego. Hay que tener en cuenta que, aunque las mecánicas estén basadas en un juego de acción/aventuras, se añade una componente adicional, en relación con la realidad aumentada. Las mecánicas de acción básicas del juego se pueden clasificar según, las tres escenas indicadas con anterioridad.

6.5.1. Mecánicas de Escena Mapa

- **Moverse**: a partir del reconocimiento del dispositivo, en cuanto el usuario se desplace lo detectará gracias a los sistemas de geolocalización.
- **Mover vista (cámara)**: Al interactuar con la pantalla, podremos aplicar zoom, centrando la cámara en el personaje. Por lo que respecta al giro de la cámara alrededor del personaje, podremos hacerlo de dos formas:
    - **Giroscopio**: con la cámara centrada en el personaje, las rotaciones realizadas por el modelo se aplicarán también al objeto cámara.
    - **Gestos de pantalla**: mediante gestos podremos girar la vista de la cámara y aplicar las transformaciones necesarias.
- **Recoger objetos:** Habrá objetos como llaves, vidas, entre otros, que nos serán útiles para avanzar. Una vez recogidos aparecerán en nuestro inventario.
- **Interactuar con puntos de combate o puzle:** La interacción con los enemigos o puzles en el mapa, nos permitirá acceder a las escenas de combate y puzle, respectivamente.
- **Usar objetos**: Mejoras en las habilidades o en la energía del jugador, son elementos que le permitirán progresar más rápidamente en nivel y en el desarrollo de la historia.
- **Interactuar con las distintas interfaces**: Elementos visuales que permiten navegar entre los diferentes menús que proporciona el juego: inventario, opciones, etc. Se explica con más detalle en el punto 6.8.

6.5.2. Mecánicas de Escena Batalla

- **Apuntar**: Se podrá apuntar en función de la cámara y del giroscopio del teléfono, centrándolo hacia la dirección que deseemos. La cámara se activará cuando salgamos de la escena del mapa y pasemos a fases de acción, ya sea combates o puzles.
- **Protegerse/esquivar:** Funcionalidad que le permitirá defenderse ante a los ataques de los distintos enemigos.
- **Disparar**: Al pulsar sobre una zona específica o botón en pantalla se podrá disparar a los enemigos, para derrotarlos.
- **Disparo especial**: Al pulsar sobre una zona específica o botón en pantalla se podrá disparar con un ataque especial, que infligirá más daño del habitual.
- **Recoger color en la imagen (mejora opcional)**: Para que la experiencia del juego interactúe con la realidad, me gustaría implementar algún sistema para reconocer colores en la cámara y usarlos en diferentes situaciones durante la partida (interactuando con elementos 3D de ese color, por ejemplo).
- **Recoger con armas/armaduras:** Habrá objetos de equipamiento que se podrán recoger durante el combate. Una vez recogidos aparecerán en nuestro inventario.

6.5.3. Mecánicas de Escena Puzle

- **Movimiento de personaje:** Movimiento del protagonista basado en físicas, necesario para alcanzar el objetivo de la escena o posición final del puzle.

- **Otras mecánicas avanzadas de interacción con objetos 3D:** se trata de mecánicas en relación con la realidad aumentada. Estas estarán basadas en la realización de los puzles mediante marcadores de realidad aumentada, así como el uso del giroscopio del dispositivo (detectando el movimiento/rotación del mismo).

6.6. Personajes y objetos

En lo que se refiere a los elementos 3D que darán vida al juego durante la partida, tendremos, por un lado, aquellos que mejorarán nuestras condiciones en la partida, permitiéndonos avanzar con mayor facilidad en muchas situaciones. Por otro lado, encontramos a los personajes, que por lo general serán enemigos a los que derrotar.

En primer lugar, analizaremos todos los tipos presentes de personajes. En el caso de los enemigos, su comportamiento estará basado en la inteligencia artificial que se va a desarrollar. Pueden ser clasificadas en:

- **Jugador** (protagonista): Humano con el valor y las herramientas necesarias como para enfrentar a los robots invasores, ocultos a los ojos del resto de la sociedad. Gracias a la realidad aumentada, la geolocalización y la cámara, será capaz de reconocer dónde se encuentran estos enemigos, para así poder enfrentarlos. Su objetivo es progresar durante en el juego, subiendo de nivel, mejorando sus habilidades y encontrando el origen de estos robots.
- **Enemigos**: Todos ellos están programados con el objetivo de conquistar la Tierra, repartiéndose en distintos puntos del mundo, hasta ser lo suficientemente numerosos como para realizar un ataque a gran escala. Entre ellos, se pueden distinguir tres tipos:
    - Básicos: Se trata de los enemigos más comunes y fáciles de derrotar. Suelen aparecer en pequeños grupos (tres o cuatro enemigos) para atacar juntos y suponer una amenaza mayor. Vencerlos nos dará recompensas.
    - Avanzados: Con menor probabilidad, aparecerán zonas en las que podremos combatir a un jefe. Suelen venir acompañados de dos en dos y se ayudarán durante el combate. Tienen más salud e infligen más daño. Dan más recompensas de lo habitual.
    - Jefe final: Aparecerá en cuanto alcancemos el nivel suficiente, en una zona cercana a nosotros. Será el objetivo principal del juego, por tratarse del causante de la fabricación de este ejército.

  A todos los enemigos, se les podría incluir la mejora del color opcional, en relación con la mecánica comentada en el punto anterior. Se trataría de atacar a los enemigos con disparos de ese color para vencerlos con mayor facilidad.

Por lo que respecta a los objetos, existirá la posibilidad de obtenerlos al resolver puzles y al vencer enemigos. También se obtendrá siempre que se suba de nivel, factor que dependerá directamente de los dos puntos anteriores. Los objetos pueden clasificarse en:

- Temporales: Elementos de utilidad con uso limitado, ya sean vida, llaves, ventajas de combate, etc.

- Permanentes: Objetos generalmente relacionados con equipo de combate y que una vez desbloqueados, podrán emplearse a lo largo de la partida (armas y armaduras).

6.7. Escenarios y ambientación

En este apartado analizaremos los escenarios y la ambientación que se va a aplicar al juego a nivel genérico. Cabe decir que, por la componente de realidad aumentada, los escenarios y la ambientación tendrán un enfoque diferente al habitual.

Por un lado, los escenarios estarán basados en los diferentes lugares que visite el jugador con su dispositivo, a través de la cámara. En general, los escenarios no estarán basados en modelado 3D, sino que se tratará de lugares reales, que se combinarán con elementos de virtuales. Se puede llegar a establecer una clasificación de los elementos del escenario que pueden aparecer, en base al tipo de escena:

1. **Mapa:** La escena en la que se muestra el mapa podría incluir algunos modelos de escenario, al aplicar los edificios con profundidad 3D, aunque serán elementos muy básicos (formas poligonales simples) con una textura aplicada. Otro tipo de objetos que podremos encontrar son los indicadores de eventos, es decir, marcas para acceder al modo puzle/combate, distribuidas en diferentes puntos del mapa. Esta escena, junto con todas las opciones que contiene, se visualizarán en con la pantalla del dispositivo en vertical.
2. **Puzle:** Donde es posible que encontremos escenarios será en las escenas de puzles donde habrá laberintos y otros elementos que rellenarán el entorno en base a las pruebas del juego. Esta escena se visualizará en con la pantalla del dispositivo en horizontal.
3. **Combate:** Este modo, muy probablemente no incluya elementos de escenario (ni decorativos ni interactuables), ya que el uso de la cámara reemplazará a estos elementos en su lugar. Al igual que la anterior, la escena se mostrará con vista horizontal.

Respecto a la ambientación, tendremos una estética “bélica”, por los combates que se desarrollan a lo largo del juego, y “futurista”, por la integración de robots como punto principal de la historia asociada al protagonista. Sin embargo, el juego no tiene la intención de llegar a ser de carácter realista, sino, que se pretende crear un contraste claramente

identificable entre la realidad actual y los elementos 3D que aparecerán en las distintas situaciones del juego.

6.8. Interfaz

La interfaz del juego a desarrollar se caracteriza por ser variable, en función de la escena en la que nos encontremos. Por tanto, el análisis de la interfaz se va a realizar en base a las diferentes escenas que se pueden encontrar.

6.8.1. Interfaz de Escena Mapa

En primer lugar, la escena principal del juego que vamos a encontrar es la del mapa, basada en la geolocalización y movimiento del jugador en la realidad. La presencia de elementos de menú es bastante reducida, para evitar que interrumpa la experiencia de juego u oculte información relevante. Entre ellos tendremos:

- **Información de perfil y nivel**: Se trata de un elemento situado en la parte superior izquierda de la pantalla que dispondrá de información actualizada sobre el nivel. Si pulsamos en él, pasaremos a ver más información del perfil.
- **Energía**: Centrado en la parte superior, tenemos el elemento que muestra en todo momento la energía disponible del jugador. Si acaba en 0, el jugador no podrá ganar experiencia ni acceder a los combates. Para recuperar energía, debe encontrar objetos repartidos por el mapa que la reestablecerán.
- **Menú desplegable**: Es el elemento principal de la interfaz en esta escena, ya que nos dará acceso a 3 secciones importantes del juego. Por defecto se encontrará sin desplegar, pero cuando pulsemos en él, se nos mostrarán las siguientes opciones:
    - Inventario: Nos permite acceder a la vista del inventario, con tal de gestionar nuestros objetos y/o armas.
    - Opciones: Nos da acceso a la vista de ajustes del juego, para modificar algunos parámetros.

Se puede ver la distribución de estos elementos en un esbozo, como el de la siguiente *figura 20:*

Figura 20: Captura donde se muestra de forma esquemática, la vista del mapa

Fuente: Elaboración propia en https://moqups.com/

Los enemigos y objetos que se ven en la *figura 20*, están colocados de forma arbitraria en el mapa y no forman parte de la interfaz de la escena en sí. Se trata de una simulación de como estarían distribuidos los elementos en la escena en cuestión.

Pasamos a ver la escena del inventario (*figura 21*), accesible desde el menú desplegable comentado anteriormente. En ella destacamos 2 partes principales, que son: la sección de elementos del inventario y la barra de navegación. En la primera de ellas, tenemos todos los objetos y armas que hayamos encontrado durante nuestra partida. Se puede pulsar en ellos para ver en detalle la información. La navegación nos permite clasificar entre las dos categorías de objetos disponibles: Los objetos son de usos limitados, mientras que las armas/armaduras son objetos equipables de usos ilimitados. Tenemos opción de volver a la escena del mapa.

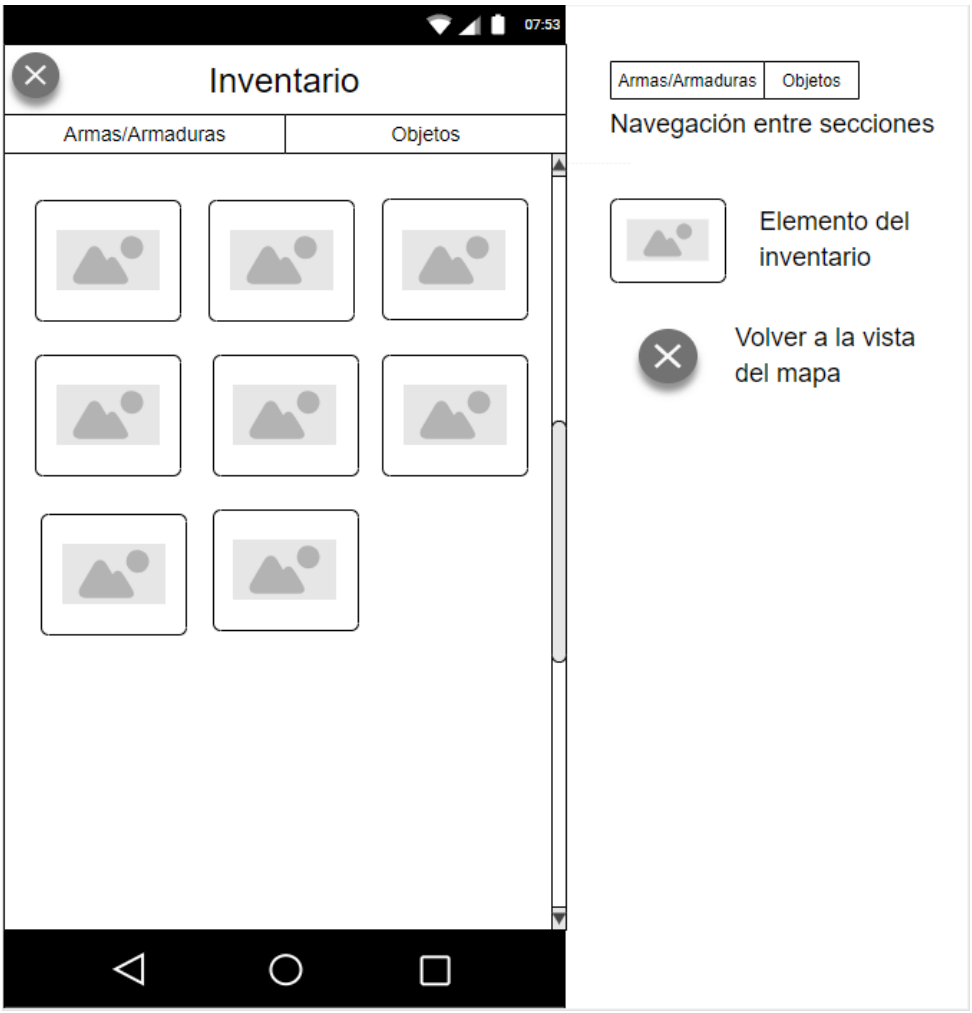


Figura 21: Captura donde se muestra de forma esquemática, la vista del inventario

Fuente: Elaboración propia en https://moqups.com/

Pasamos a analizar la vista del detalle de un elemento del inventario (*figura 22*), al que se puede acceder pulsando desde la escena de anterior. Aquí encontramos más información sobre el elemento en cuestión, como es: el tipo de objeto, información de obtención, el objetivo y las ventajas que ofrece al usarlo. Nuevamente, tenemos la opción de volver a la escena anterior, es decir, la del inventario.

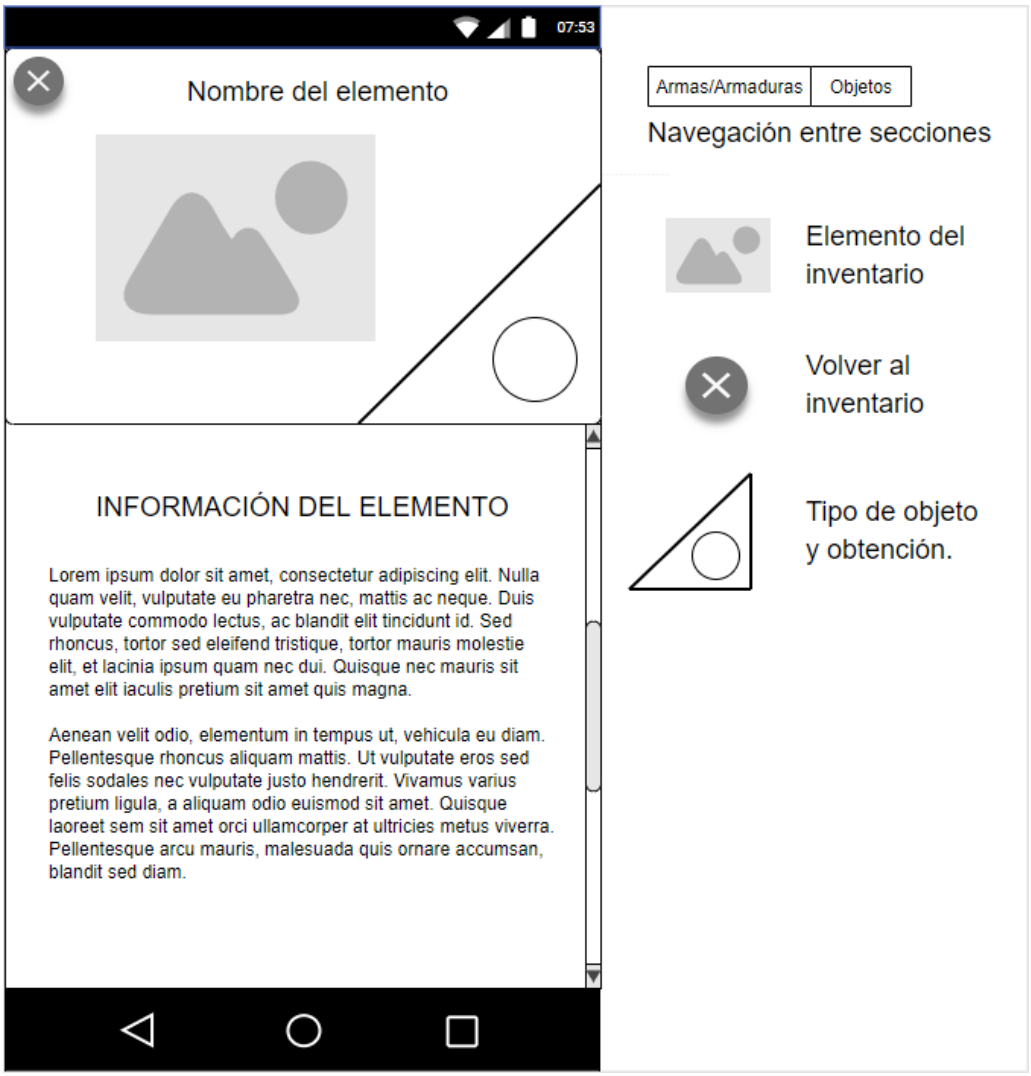


Figura 22: Captura donde se muestra de forma esquemática, la vista del detalle de objeto
Fuente: Elaboración propia en https://moqups.com/

La siguiente escena a analizar se trata de las opciones del juego (*figura 23*). Se puede acceder desde el menú desplegable en la escena del mapa. En ella se pueden revisar y configurar algunas opciones, pero principalmente tendremos:

- Sonido: Ajustar el nivel de volumen del juego en las diferentes escenas.
- Notificaciones: Ajustar la vibración y otras alertas al encontrar enemigos.
- Realidad aumentada: Configurar algunas opciones para las escenas de puzle y combate que incluyan esta funcionalidad.
- Cómo jugar: Instrucciones de juego.

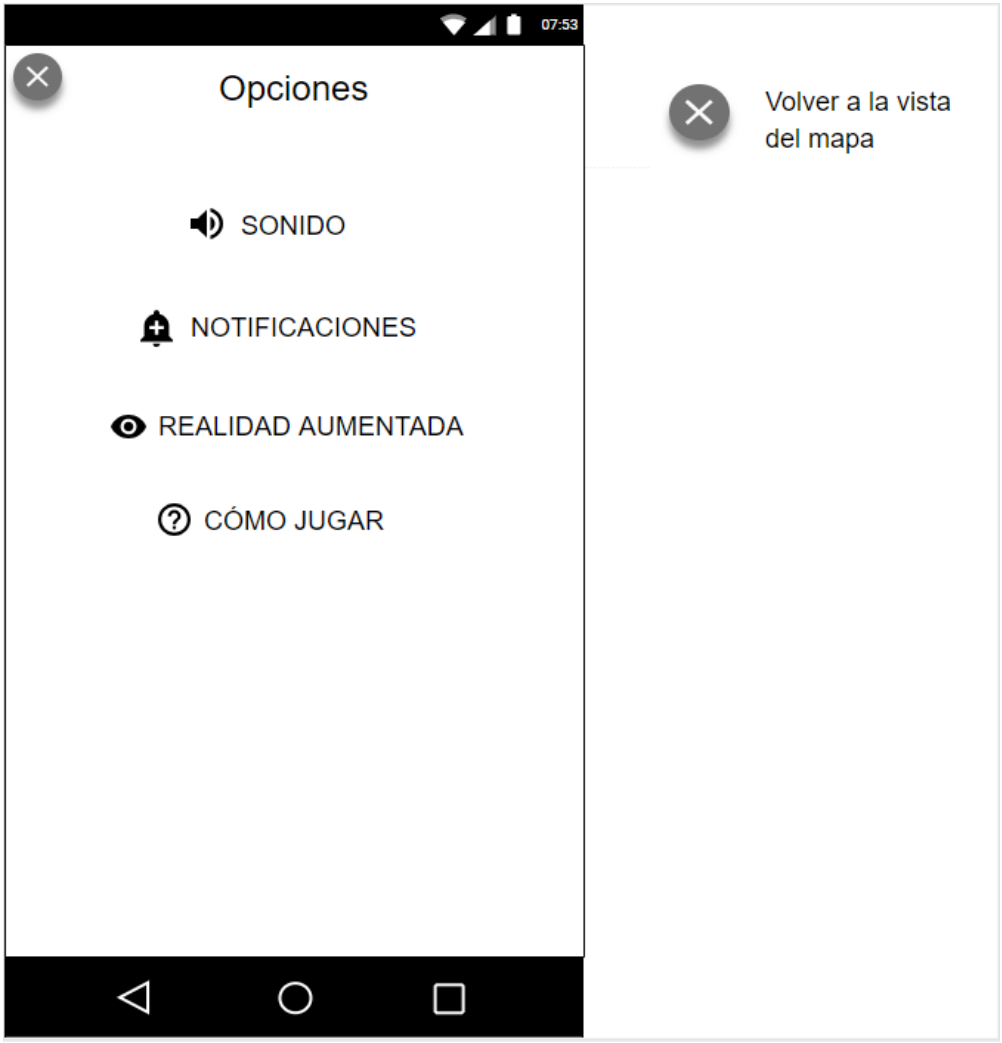


Figura 23: Captura donde se muestra de forma esquemática, la vista de opciones.
Fuente: Elaboración propia en https://moqups.com/

La siguiente escena a analizar es el perfil de usuario (*figura 24*), donde tenemos acceso a los detalles de perfil, junto con el progreso de nivel. Se puede acceder a esta vista desde la escena del mapa, pulsando sobre el nivel. Algunos ejemplos de información adicional que podremos obtener son:

- Número de enemigos eliminados.
- Número de puzzles solventados.
- Km recorridos.
- Etc.

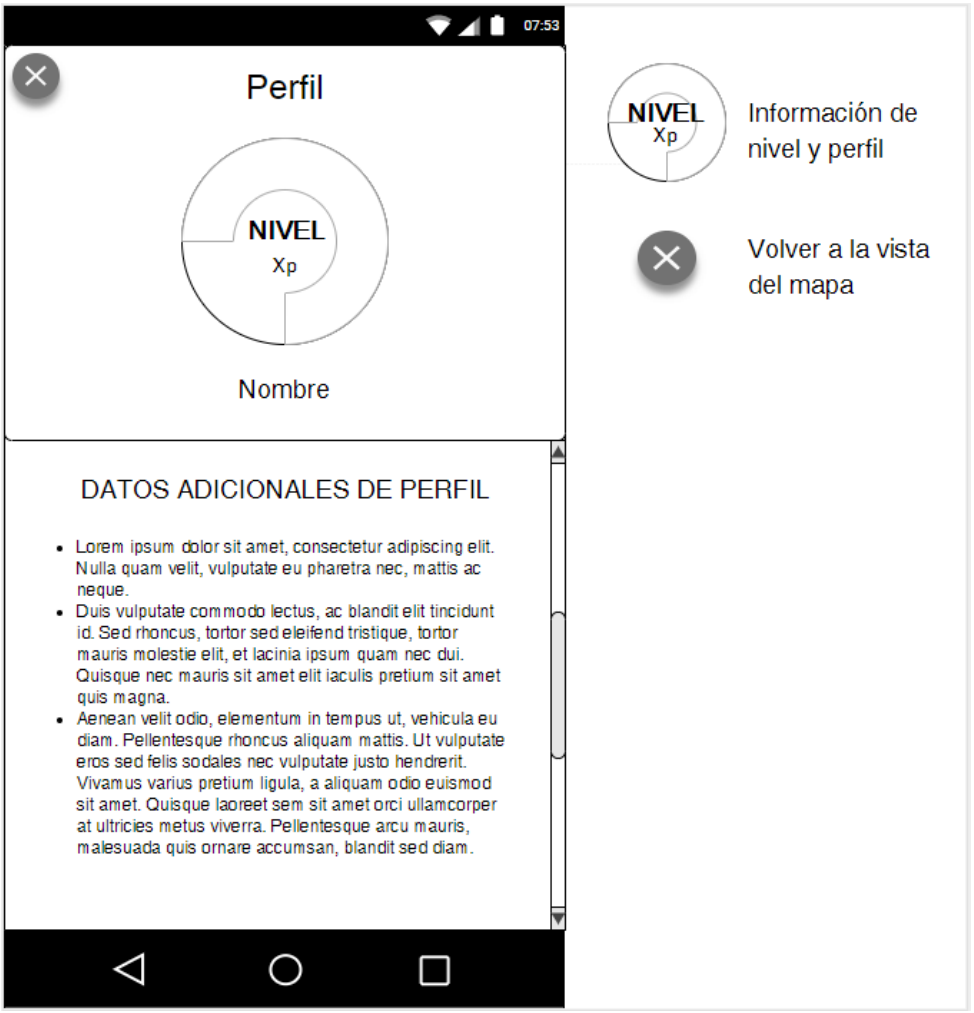


Figura 24: Captura donde se muestra de forma esquemática, la vista del detalle de perfil

Fuente: Elaboración propia en https://moqups.com/

6.8.2. Interfaz de Escena Batalla

Otra de las principales escenas es la escena de batalla. Se puede acceder a ella pulsando en uno de los enemigos dentro del rango. Esta escena también incluirá algunos elementos de interfaz necesarios para el combate. Entre ellos tenemos:

- **Barra de energía**: Coincidirá con la ya vista en la escena del mapa. Es en los combates donde puede verse afectada negativamente. Si nos quedamos sin energía, nos tendremos que retirar del combate sin obtener los puntos de experiencia ni las recompensas.
- **Mirilla:** Necesaria para apuntar a los enemigos y orientar nuestros ataques/interacciones.

- **Botones de interacción**: Zona de la interfaz, situada en la parte inferior que nos permitirá atacar y defendernos en todo momento según los botones que pulsemos.

Puede verse reflejado en el siguiente esbozo o *figura 25*:

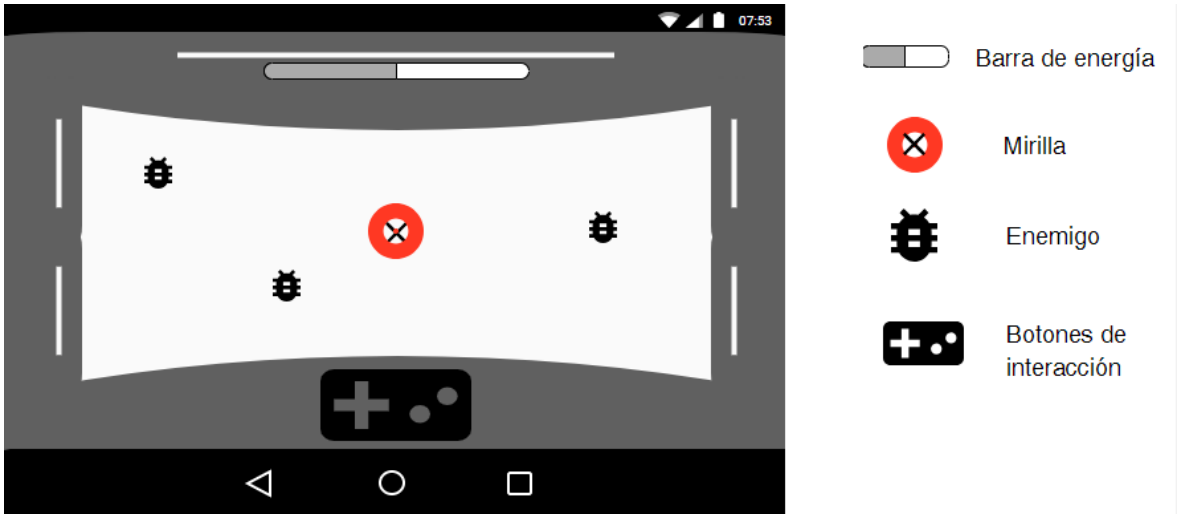


Figura 25: Captura donde se muestra de forma esquemática, la vista de combate
Fuente: Elaboración propia en https://moqups.com/

Los enemigos serán un elemento 3D en movimiento, por lo que no puede considerarse como parte de la interfaz.

6.8.3. Interfaz de Escena Puzle

Por último, la escena que vamos a ver es la de los puzles (*figura 26*). Se trata de la escena con la menor cantidad de información por interfaz, ya que los elementos principales serán: el escenario, el personaje y el objetivo del puzle.

Un ejemplo de esta escena puede verse en la *figura 26* a continuación.

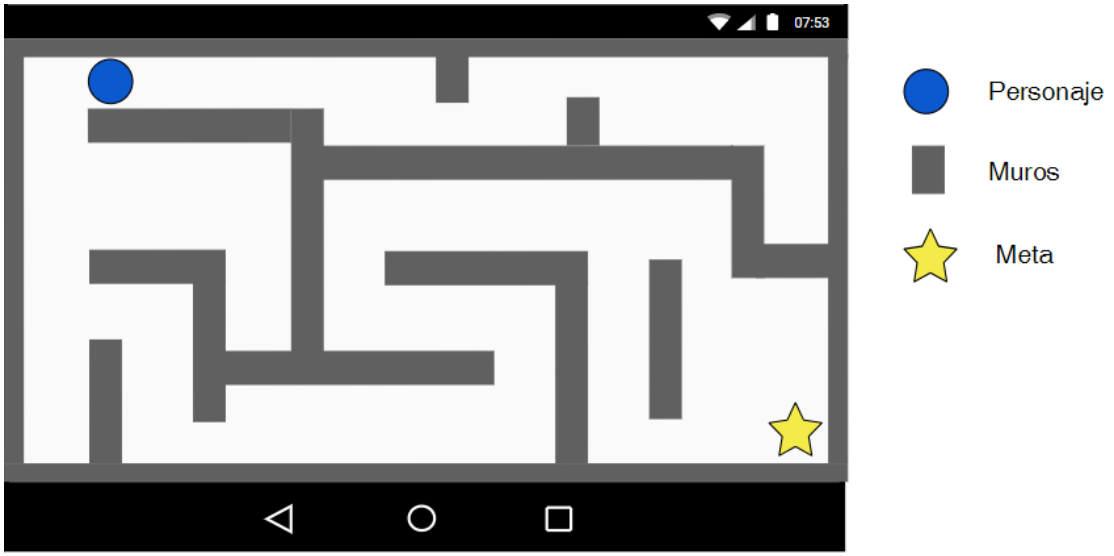


Figura 26: Captura donde se muestra de forma esquemática, la vista de puzle
Fuente: Elaboración propia en https://moqups.com/

Esta hará uso de las funciones del giroscopio del dispositivo, a la hora de mover el personaje y la cámara con la realidad aumentada (marcadores).

6.9. Control de Realidad Aumentada

Durante este apartado se van a tratar dos puntos importantes para el desarrollo del proyecto. Estos están directamente relacionados con la interacción del jugador con el juego y emplean funciones avanzadas de los dispositivos móviles. Vamos a tratar de describir en qué consisten y explicaremos su funcionamiento.

Primeramente, vamos a comentar el funcionamiento de la Realidad Aumentada a emplear durante el desarrollo del proyecto. Se pueden destacar dos tipos de realidad Aumentada, en base a dos tipos de elementos:

- **Mapa**: Se basa en la geolocalización y consiste en representar la realidad en base a un plano con las direcciones y lugares exactos, incluyendo a su vez elementos virtuales. Como el usuario debe moverse en la realidad para alcanzar los objetivos virtuales, se constituye como un elemento clave de las funcionalidades avanzadas.
    - Para su desarrollo se va a hacer uso de una librería llamada *MapBox* (Mapbox, Wikipedia, 2018), que es un proveedor de mapas online fundado en 2010. Como factor diferenciador, permite generar mapas personalizados,

seleccionando elementos que nos resulten de mayor interés, así como seleccionar la combinación de colores. En las siguientes *figuras 27 y 28* se puede ver el logo de *Mapbox* y algunos diseños personalizados del mapa con la librería:

Figura 27: Logo de la librería Mapbox
Fuente: https://bit.ly/2EDmPot

Figura 28: Imagen de mapas personalizados en Mapbox
Fuente: https://bit.ly/2H6xiwr

Además, *Mapbox* tiene como gran ventaja la integración en Unity, dando facilidades para su implementación y configuración. Esta integración se realiza mediante un paquete de librería para Unity, que podemos descargar desde la página oficial de *Mpabox*.

- **Cámara del dispositivo**. En segundo lugar, hablaremos de la cámara del dispositivo y el uso que tendrá durante las partidas. La cámara es un elemento principal que se encuentra desactivado hasta el momento en el que entramos a las fases de batalla y puzle. Estas fases pueden ser consideradas como las más importantes del juego, ya que permiten progresar más rápidamente (obteniendo más experiencia de nivel) y en las que obtendremos ventajas para futuras batallas (principalmente, armas y armaduras). La cámara ofrece la posibilidad de desarrollar una realidad aumentada, combinando imágenes reales con los elementos virtuales superpuestos. Además, junto con el uso del giroscopio del teléfono, nos permitirá cambiar la orientación en la escena y apuntar a los objetivos deseados. Si analizamos por escenas, se tiene:
    - **Batalla**: En esta escena, el objetivo de la cámara es claro, ya que se constituye como la principal mecánica del usuario (apuntar). En este caso, el objetivo es apuntar para disparar a los enemigos que estarán distribuidos en diferentes puntos de la escena. Una vez centrados en la cámara, podremos disparar para acabar con ellos. Esta funcionalidad hace uso del giroscopio del teléfono, necesario para moverse por la escena, en base a la orientación.
    - **Puzle**: Existe la posibilidad de desarrollar los puzles basados en realidad aumentada o no. La opción para habilitarlo estará disponible desde los ajustes del juego. Nuevamente, la cámara será la responsable de esta función que, apuntado a una imagen plana u objetivo, podremos generar el puzle en cuestión. Se trata de la realidad aumentada basada en marcadores. Por tanto, al mover la superficie del marcador, se generará un movimiento en las entidades correspondientes (no fijas).

      En caso de tener esta funcionalidad deshabilitada, los puzles se basarán únicamente en el acelerómetro del teléfono, que comprueba la inclinación del este respecto a la horizontal. Los puzles consistirán en pequeños laberintos o recorridos con obstáculos. Si conseguimos superar obtendremos experiencia y objetos de recompensa (en base a probabilidades).

En concreto, una vez experimentados los diferentes *SDKs* de realidad aumentada, se ha decidido emplear la librería *Vuforia* (Vuforia, 2019). Se empleará principalmente para las funciones basadas en marcadores. El hecho de seleccionar esta librería respecto a otras es por su compatibilidad con Unity y licencia gratuita.

6.10. Cámara

A continuación, vamos a tratar el elemento "cámara", que no depende del hardware del dispositivo, sino del **componente en la escena de Unity** que se posiciona tratando de mostrar la información deseada y limitando el campo de visión. Este elemento como tal, lo encontraremos en la escena del mapa. La posición por defecto de la cámara se sitúa en la parte superior de la escena y siempre asociada al personaje protagonista, al que seguirá y apuntará en todo momento. La cámara se podrá ajustar en posición respecto al eje Y, modificando su altura que dará la sensación de zoom. Las modificaciones en el resto de ejes estarán bloqueadas, con el fin de no perder de vista al jugador en ningún momento. Las transformaciones de rotación, nuevamente, se realizarán en base al eje Y, de forma que podamos mirar en varias direcciones mientras avanzamos por el mundo, pero siempre apuntando al protagonista. En la *figura 29* puede verse como quedaría la cámara respecto al jugador y las sombras proyectadas en base a la posición de la iluminación:

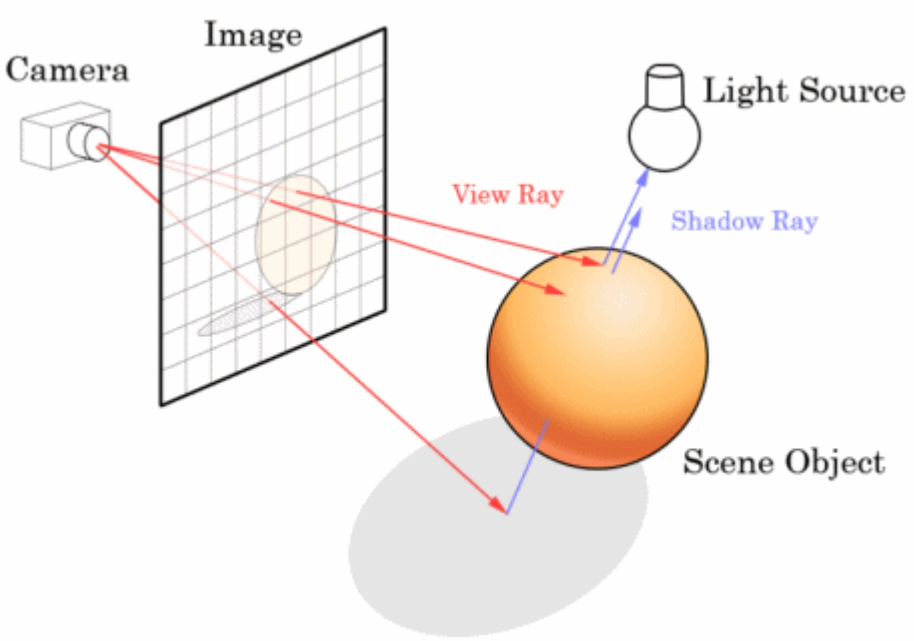


Figura 29: Representación de la cámara en la escena 3D
Fuente: https://bit.ly/2Vtx1Xv

El resto de las escenas o bien dependen de la interfaz, o bien de la cámara del dispositivo, por lo que no harán uso de la cámara de la escena.

6.11. Inteligencia Artificial

Durante este apartado vamos a comentar los aspectos de inteligencia artificial que se van a desarrollar durante el juego. Esta componente únicamente afectará a los enemigos del juego, tomando decisiones en base a las distintas situaciones. Con esto, se puede llegar a diferenciar hasta 3 tipos de inteligencia artificial, por tener 3 tipos de enemigos. Sin embargo, todos comparten algunos mecanismos que describiremos a continuación.

La inteligencia artificial (IA) de los enemigos estará basada principalmente en *behaviour trees* (Behavior tree, Wikipedia, 2019), es decir, nodos de comportamiento que se van iterando en base a distintas situaciones y rangos de tiempo. Los *behaviour trees* describen cambios entre un conjunto finito de tareas, separadas de forma modular, que se recorren de izquierda a derecha y de arriba a abajo en función de la prioridad.

A diferencia de los *decision trees* (Decision Tree, Wikipedia, 2019), se recorren únicamente desde el nodo central (*root*) la primera vez que se ejecutan (o bien si se reinician), recorriéndose a partir de entonces desde los más prioritarios y teniendo en cuenta los estados obtenidos con anterioridad. Cuando se alcanza un nodo que se puede realizar, este pasa a estar "en proceso" y se devuelve el estado actual. En caso de que alguno falle, se volvería hacia atrás, pasando por el nodo padre para reevaluar la condición, que en caso negativo saltaría a otra rama del árbol, hasta que algún nodo finalmente se encuentre "en proceso". El potencial de estos árboles de comportamiento es la habilidad de crear tareas complejas compuestas de tareas más simples. Puede verse un esquema en la siguiente *figura 30*:

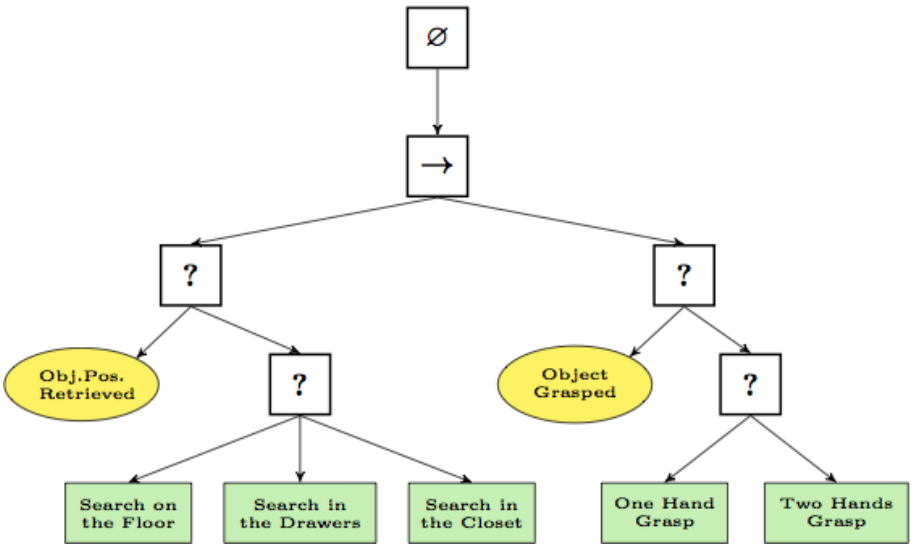


Figura 30: Representación de un modelo de behaviour tree
Fuente: https://bit.ly/2R3SFD3

En base a este punto en común de la IA, vamos a analizar los tres tipos de IA que se pueden encontrar:

- **Enemigo básico**: la inteligencia artificial de estos enemigos estará basada en un *behaviour tree* con tres nodos principales. Ordenando según su prioridad (de izquierda a derecha en el árbol), se pueden identificar las siguientes tareas:
    - Detección de disparos/daño: Se basa en realizar una comprobación de colisión, que influye negativamente en su salud. Esto se cumple cuando es golpeado por alguno de nuestros ataques. Si tras realizar esta comprobación, la respuesta es verdadera, este enemigo pasará a realizar a una reacción correspondiente, que puede verse reflejada en 2 tareas (en orden de prioridad).
        - Contraataque: Se realizará en base a una probabilidad, que irá aumentando a medida que baje la vida del mismo. En ese momento, el enemigo atacará al jugador ya sea con ataque a distancia o con ataque cuerpo a cuerpo.
        - Escape/turbo: En caso contrario, el enemigo pasa a un estado más defensivo que consiste en aumentar su velocidad temporalmente, con tal de evitar ser golpeado de nuevo.
    - Movimiento/patrulla: Se trata en realizar un movimiento alrededor del personaje protagonista durante un tiempo. Puede oscilar en 3 y 7 segundos aproximadamente.
    - Atacar: Como último nodo en orden de prioridad, una vez haya finalizado el tiempo de patrulla, los enemigos pasarán a atacar, con tal de acabar con nosotros. Los ataques podrán ser cuerpo a cuerpo (placaje) o a distancia, para ello existirá un factor de probabilidad que irá alternando estos tipos de ataque.
        - A distancia: Consiste, por lo general, en disparos sencillos que serán previamente notificados al jugador con un indicador.
        - Cuerpo a cuerpo: El enemigo se acercará progresivamente al jugador para chocar con él y reducir su vida. Este ataque inflige más daño que el anterior. De nuevo, tendremos otro indicador que nos avisará.

  El árbol quedaría similar al siguiente diagrama (*figura 31*):

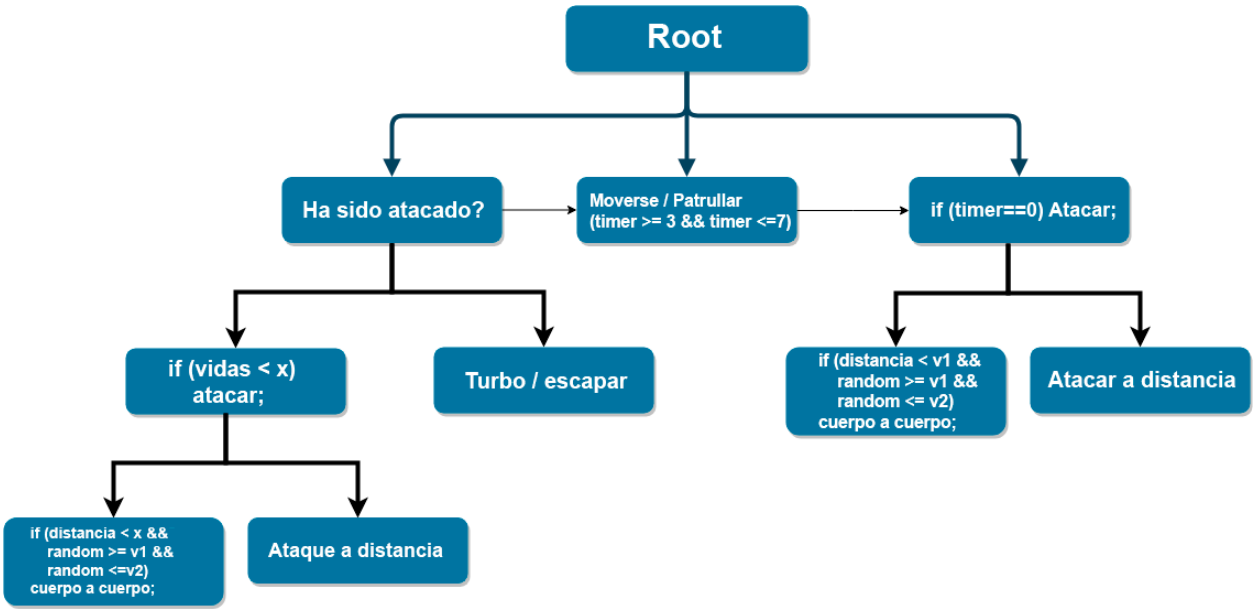


Figura 31: Representación del behaviour tree del primer enemigo
Fuente: Elaboración propia mediante https://www.draw.io/

- **Enemigo avanzado**: Se trata de un tipo de enemigos que además de seguir las rutinas básicas de los anteriores, añadirán habilidades especiales y más factores de variación en su comportamiento. Así mismo, estos enemigos tendrán una cantidad de salud y infligirán un daño superior a la de los enemigos normales. Sobre las habilidades especiales, se puede decir que cada enemigo puede tener como máximo 2 de ellas:
    - Curación (defensivo): El enemigo con esta habilidad podrá curarse durante el combate, cuando detecte que tiene poca vida.
    - Escudo (defensivo): El enemigo con esta habilidad, se podrá equipar un escudo durante el combate.
    - Disparo poderoso (ofensivo): Puede usar un disparo cargado para infligir mucho daño.
    - Ráfaga(ofensivo): Tiene la habilidad de lanzar varios disparos consecutivos.
    - Velocidad aumentada (defensivo): Se desplaza más rápido de lo normal y le permite esquivar más ataques.
    - Ataque cuerpo a cuerpo en carrera (ofensivo): Sus ataques cuerpo a cuerpo serán más rápidos de lo normal.

En base a esto, tenemos el siguiente árbol de comportamiento, *figura 32* (con posibles variantes de comportamiento añadidas):

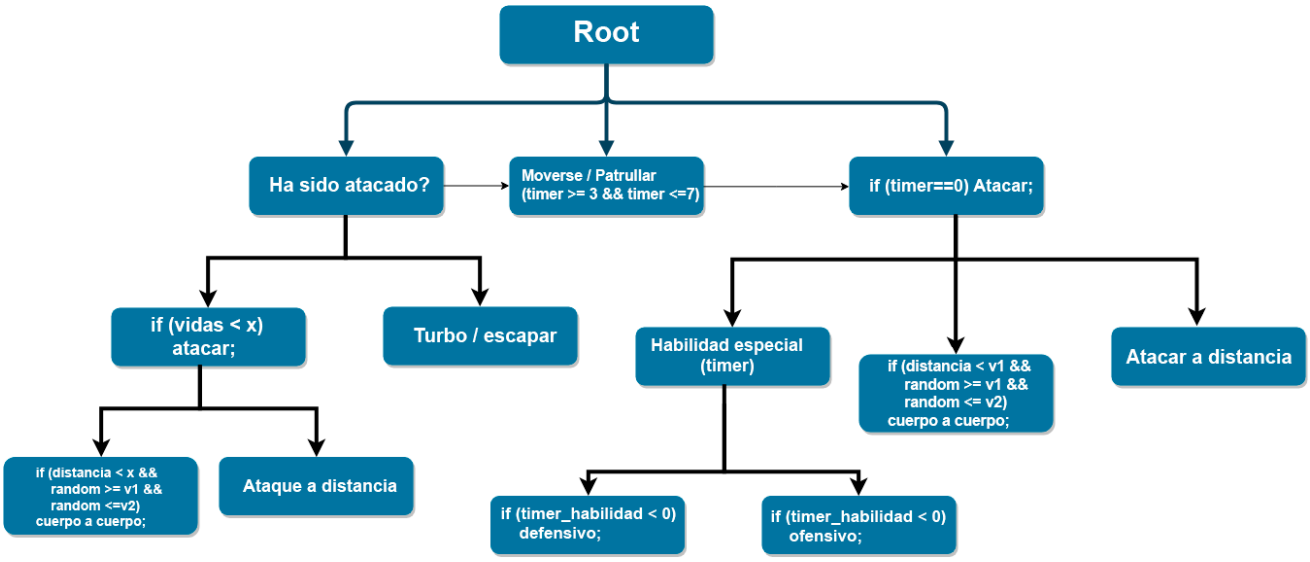


Figura 32: Representación del behaviour tree del segundo enemigo
Fuente: Elaboración propia mediante https://www.draw.io/

- **Enemigo jefe final**: Se trata del enemigo final del juego que puede usar cualquiera de las habilidades expuestas en el apartado anterior. En este sentido, el *behaviour tree* diferirá muy poco respecto al anterior, salvo en el tipo de movimientos y que todas las habilidades podrán ser utilizadas con libertad. Sus ataques infligen más daño del habitual y dispone de una cantidad de salud bastante superior, lo suficiente como para suponer un gran reto para el jugador.

Para el desarrollo de estos enemigos, se hará uso de una herencia, con un enemigo padre que establezca las bases del comportamiento, así como las variables y funciones necesarias.

6.12. Sistema de progresión y objetivos

En este apartado vamos a introducir algunos de los objetivos principales del jugador durante las partidas y cómo avanzar en base al sistema establecido.

El juego está basado en un sistema de niveles ascendente del jugador, mediante el cual, se pretende avanzar en la historia. Para poder subir de nivel, tendremos como primera necesidad o requisito que movernos por el mundo para encontrar puntos de combate, puzles y objetos. Con estos, iremos obteniendo experiencia, es decir un valor comprendido en un rango de 0 a un máximo (variable). Cuando se alcance el máximo, supondrá el ascenso de nivel, y este valor de experiencia volverá a ser 0. La experiencia es un recurso que se conseguirá en mayor cantidad con los combates, seguido de los puzles y finalmente los objetos. Los combates

contra jefes siempre nos darán más experiencia de lo normal, aunque de la misma forma, no serán igual de fáciles de encontrar (menos frecuentes).

A lo largo del juego, además de nivel, iremos obteniendo objetos para nuestro inventario. Por un lado, los objetos temporales serán de un solo uso y tendrán un efecto limitado. Sin embargo, los objetos permanentes tales como armas y armaduras, serán para siempre una vez obtenidos. La forma de obtener objetos temporales es: en puzles y en combates contra jefes, donde existirá una probabilidad de obtenerlos. También podremos encontrar algunos de estos objetos repartidos por el mundo (en la escena de mapa). Por otro lado, los objetos permanentes se conseguirán al subir de nivel.

Como hemos comentado anteriormente, el objetivo es eliminar al jefe final, causante de la creación masiva de robots enemigos. Para poder combatir contra él, debemos alcanzar un nivel alto (por ejemplo, nivel 30). En ese momento aparecerá una misión especial para entrar en el combate final.

Antes de este nivel, solo podremos enfrentarnos a enemigos básicos y, con menor frecuencia, a jefes. Durante el combate, deberemos acabar con los enemigos. Cuando no quede ninguno, nos aparecerá un aviso de victoria (con los datos correspondientes) y seguidamente pasaremos a la escena del mapa de nuevo, con los puntos de experiencia actualizados, así como la barra de energía. Sin embargo, si somos derrotados durante el combate, volveremos instantáneamente a la escena mapa, donde nuestra barra de energía estará vacía. A partir de ese momento no podremos acceder a ninguno de los puntos de combate o puzle, ya que nuestro personaje se encontrará incapacitado y en una situación difícil. Para recuperarse, deberemos obtener objetos de curación repartidos por el mundo. Para ello, al igual que el resto de los objetos, deberemos acercarnos a ellos (dentro de un rango), para poder obtenerlos. Cuando lo hayamos obtenido, deberemos acceder al menú de inventario, para usar el objeto en cuestión. La escena de puzle también nos supondrá un consumo de la barra de energía, tanto si salimos victoriosos como si no, sin embargo, en el primer caso obtendremos experiencia y en el segundo no.

Una vez superado el jefe final, se puede seguir subiendo de nivel, obteniendo objetos, experiencia y armas. Por tanto, el juego no termina una vez superado el jefe final, sino que se permite seguir progresando y explorando el mundo, en el que todavía quedan robots que eliminar.

6.13. Sonido y música

Los sonidos y la música son uno de los elementos principales para dar vida al juego, ya que mejoran mucho la experiencia de usuario y la inmersión a la hora de jugar. Para lograrlo, todos estos sonidos deben ser acordes al entorno y la ambientación del juego en cualquier momento.

Pese a tener una ambientación de tipo bélica/futurista, el juego no tiene intención de ser serio, sino que opta por un enfoque de acción desenfadado. Sin embargo, el cambio de escena debe de estar claramente identificado en todos los aspectos audiovisuales. Por tanto, teniendo en cuenta que el juego tiene varias escenas clave, debemos adaptarnos a la situación que cada una nos presenta. Según las distintas escenas tenemos:

- En primer lugar, la **música de la escena mapa**, debe estar enfocada a la exploración, con un tono más relajado que cualquier otro. Será uno de los temas principales del juego, al encontrarse más presente en las partidas que el resto de escenas. Durante esta escena, además de la música, podremos escuchar otros sonidos. Los principales son:
    - Sonidos de los enemigos al aparecer cerca.
    - Sonidos de interacción con enemigos y puntos de misión del mapa (batalla, puzle, etc.)
    - Sonidos de interacción con los botones de la interfaz.
- Por otro lado, en la transición a la **escena de combate**, la música cambiará hacia un tono más serio que represente acción y batalla. En esta escena tendremos varios sonidos, que ayudarán a entender las diferentes situaciones de la batalla y son:
    - Sonidos de los disparos: Se reproduce al pulsar en el botón de disparo de la interfaz y cuando disparan los enemigos. Puede haber más de un tipo de disparo con su correspondiente sonido.
    - Sonidos de alertas: Cuando un enemigo vaya a atacar, aparecerá tanto una alerta visual como sonora, para advertir al jugador que debe defenderse.
    - Sonidos de defensa: Cuando se active un escudo, se reproducirá un sonido acorde al mismo.
    - Sonidos de curación: Se reproduce cuando algún personaje en el combate se cura.

  - Sonidos de daño a enemigo: Cuando algún disparo del jugador colisiona con el cuerpo de un enemigo, se reproduce este sonido.
  - Sonido de daño al jugador: Sonido que se activa cuando el jugador principal es golpeado por alguno de los ataques enemigos.
  - Sonidos de destrucción de enemigo: Se reproduce cuando la vida del enemigo se reduce a 0 y el enemigo es destruido.
  - Canción de victoria: Cuando todos los enemigos de la batalla han sido destruidos, aparece una ventana indicando nuestra victoria, con datos adicionales. En ese momento se puede escuchar la canción de victoria.
  - Canción de derrota: Si la vida/energía del jugador se acaba, sonará una canción de derrota y una ventana para volver al mapa.
- La **escena puzle**, tendrá un tono intermedio, más enérgico que el de la escena mapa, pero sin llegar a ser de acción como ocurría en la escena de batalla. Los sonidos que se reproducirán en esta escena son:
  - Colisiones físicas entre objetos (obstáculos, paredes, etc.)
  - Colisiones con objetos de interacción (trampa, meta, etc.).
  - Canción de victoria, como la comentada en la escena anterior.
  - Canción de derrota.
- Por lo que respecta a las **escenas de opciones**, por lo general van a reproducir la canción de exploración por el mapa. En estas escenas, únicamente, tendremos sonidos de interacción con los botones de la interfaz.

7. DESARROLLO E IMPLEMENTACIÓN

Durante este apartado se van a tratar todos los aspectos a nivel de desarrollo real del proyecto, es decir, como se han abordado los requerimientos y objetivos planteados en el apartado anterior del GDD.

Para empezar, se explicarán brevemente algunos conceptos del funcionamiento de Unity, así como los principales componentes que se emplean. Este apartado irá seguido de una explicación de los distintos puntos de desarrollados, tanto a nivel de estructura como de implementación.

7.1. Funcionamiento de Unity

Unity es un motor de videojuegos (Documentación, Unity, 2019) muy avanzado que integra una gran cantidad de elementos y herramientas para dar facilidades durante del desarrollo. Está constantemente bajo revisiones y actualizaciones para ser mejorado. En este caso, por las fechas en las que se inició el proyecto se va a realizar en la versión 2018.2.14.

Unity abarca una cantidad de factores y tecnologías tan grande que su funcionamiento es difícil de describir con detalle. Sin embargo, se pueden identificar hasta cinco elementos principales con los que trabaja Unity y que forman parte de cualquier proyecto de juego. La descripción de este tipo de elementos será de gran utilidad a hora de explicar más adelante, el desarrollo de algunos bloques de proyecto. Por tanto. los elementos que se van a describir son: los *Scripts*, los *Assets*, los *GameObjects*, los *Componentes*, los *Prefabs* y las *Escenas*. Esquema de ellos en f*igura 33*:

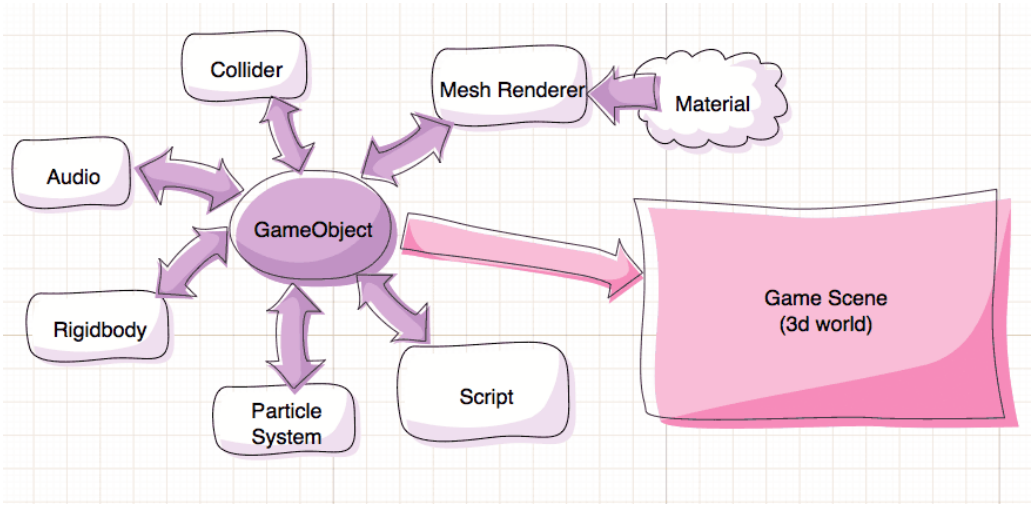


Figura 33: Esquema-resumen de la relación entre los principales elementos de Unity
Fuente: https://bit.ly/2N1FI7p

7.1.1. Escenas

Las escenas son los elementos más generales dentro de un proyecto, ya que engloban a todos los demás dentro de ellas. Un proyecto, puede estar formado por una o varias escenas en base a su tamaño y estructuración.

La función de una escena es guardar toda la organización y configuración de los elementos que contiene, estableciendo un bloque independiente y consolidado del juego a desarrollar. Este bloque debe ser capaz de comunicarse con el resto de escenas, permitiendo una transición/cambio lógico entre ellas.

Todos los elementos que se van a describir a continuación dependen en su totalidad de la escena en la que se incluyen, a excepción de los *assets*.

7.1.2. Assets y Packages

Son uno de los principales recursos dentro del proyecto de Unity. Por un lado, los *assets*, constituyen todos los elementos multimedia del proyecto previos a entrar en escena, es decir, las imágenes/*sprites*, los modelos en 3D, los ficheros de código/scripts, los sonidos y la música, etc. Son las piezas necesarias, utilizadas para crear el proyecto final y generalmente, se estructuran en una jerarquía de carpetas en función del tipo de objeto al que esté orientada su aplicación.

Por otra parte, en relación con lo definido hasta ahora, encontramos los packages. Son un conjunto de *assets* o librerías, preparadas para su perfecta importación en un proyecto. Estos, amplían las posibilidades de creación del proyecto, llegando a integrar nuevas funcionalidades y combinando en muchos casos elementos visuales con ficheros de comportamiento. En este caso, se han importado algunos *packages* ya preparados como: *Vuforia*, *MapBox*, etc.

7.1.3. Scripts

Son los ficheros de código que contienen por lo general, la lógica de la aplicación o el proyecto. Están escritos en lenguaje de programación C# y se estructuran de forma modular en funciones. Estos scripts, en su mayoría, incluyen a su vez ficheros o cabeceras preparadas por Unity, que nos ofrecen la posibilidad de hacer referencia a todos los elementos y clases que integra el motor. Todos los scripts de Unity, por defecto, heredan de una clase llamada *"MonoBehaviour"*, que contiene las rutinas básicas de ejecución. Sin embargo, estos pueden ser modificados para empezar a depender de otras clases o scripts. Durante la ejecución de la aplicación, Unity ejecuta estos scripts en base al esquema de la *figura 34* que a continuación se muestra:

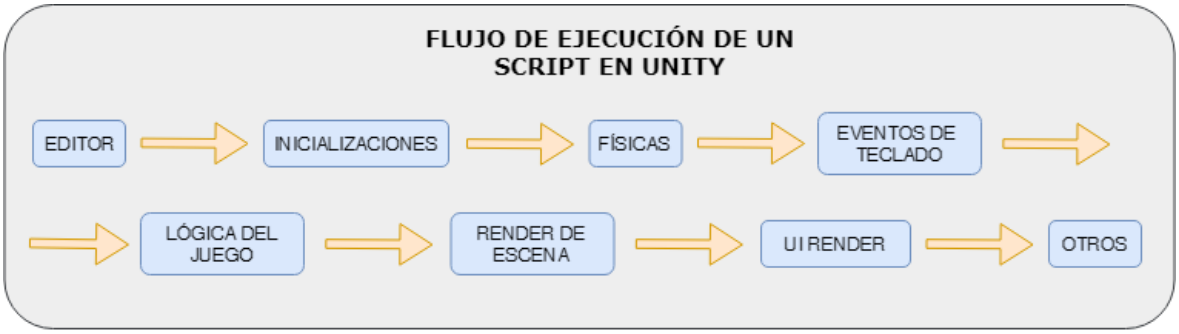


Figura 34: Diagrama que resume el flujo de ejecución en Unity de un script "Monobehaviour"
Fuente: Elaboración propia mediante https://www.draw.io/ en base al diagrama https://bit.ly/2N0cmGD

7.1.4. GameObjects

Los *GameObjects* son los principales elementos que se añaden a las diferentes escenas del proyecto y que permiten visualizar/materializar los *assets* dentro del juego, así como su funcionamiento cuando se integran los Scripts asociados. Constituyen todos los tipos de entidades que tienen una función dentro del juego, tanto a nivel visual como de

comportamiento/lógica. A diferencia de los *assets*, sí que dependen de la escena en la que se encuentren. Para diferenciarlos aún más, se sitúan en un contenedor distinto del entorno de Unity, específico para trabajar con *GameObjects*.

Sin embargo, a pesar de la importancia de estos elementos, debemos indicar que carecerían de relevancia si no fuese su capacidad de incluir todo tipo de componentes. Estos se describen en el siguiente punto.

Todos los *GameObjects* tienen en común un componente de transformaciones geométricas: traslación, rotación y escala.

7.1.5. Componentes

Los componentes son los elementos que configuran y hacen único un *GameObject* en la escena. A los *GameObjects* se les puede aplicar una gran multitud de componentes, pero por lo general, solo uno de cada tipo. En base al tipo de componente podemos encontrar principalmente:

- Componente Cámara: Una vez se establece este tipo de componente, el *GameObject* pasa a tener el comportamiento propio de una cámara en escena, ofreciendo una vista al jugador desde su posición.
- Componente Luz: Este componente, permite emitir luz desde la posición en la que se encuentre el objeto asociado.
- Componentes de físicas: Permiten dotar a la entidad de comportamiento basado en reglas físicas. Entre ellos encontramos: *Collider*, *RigidBody*, etc.
- Componentes de audio: Permiten integrar sonidos al objeto.
- *Scripts*: Los scripts que tenemos en los *assets*, pasan a estar activados cuando se asocian a un *GameObject* activo. Estos permiten establecer todo tipo de comportamientos en relación con el objeto actual u otros alcanzables en el proyecto.
- Componente *Mesh* (filter, renderer, etc): Permiten establecer atributos visuales del *GameObject* desde su forma: con primitivas geométricas o incluso modelos 2D/3D más complejos. En esta sección también podemos integrar los materiales que deben tener las figuras en cuestión
- *Animators*: Se trata de un componente que permite establecer patrones de animación a las entidades o modelos del *Game Object*.

- Sistema de partículas: Componentes avanzados que permiten generar comportamientos/efectos visuales predefinidos en una entidad, simulando un movimiento de partículas.
- Componentes de UI: Se pueden añadir también elementos de interfaz para menús, como, por ejemplo: botones, textos, listas, casillas de marcado, etc.
- Etc.

7.1.6. Prefabs

Se trata de un tipo de *asset* especial, creado a partir de uno o varios *GameObects* y que integra varios componentes combinados entre sí. Por lo general constituye un *asset* preparado para su directa integración en el juego como *GameObject* y que es muy útil para su almacenamiento y reutilización a lo largo del proyecto.

7.2. Entidades en escena

En este apartado se van a comentar, en términos generales, las entidades que se han incluido y podemos encontrar en el proyecto. Algunas de ellas se retomarán más adelante, para ser explicadas con mayor detalle. Estas entidades o *GameObjects*, pueden dividirse en base a la escena en la que se encuentre el juego.

7.2.1. Entidades Escena Mapa

Se trata de la escena principal, desde la que inicia siempre la partida. En ella podemos encontrar los elementos reflejados en la *tabla 2* y en las *figuras 35 y 36*:

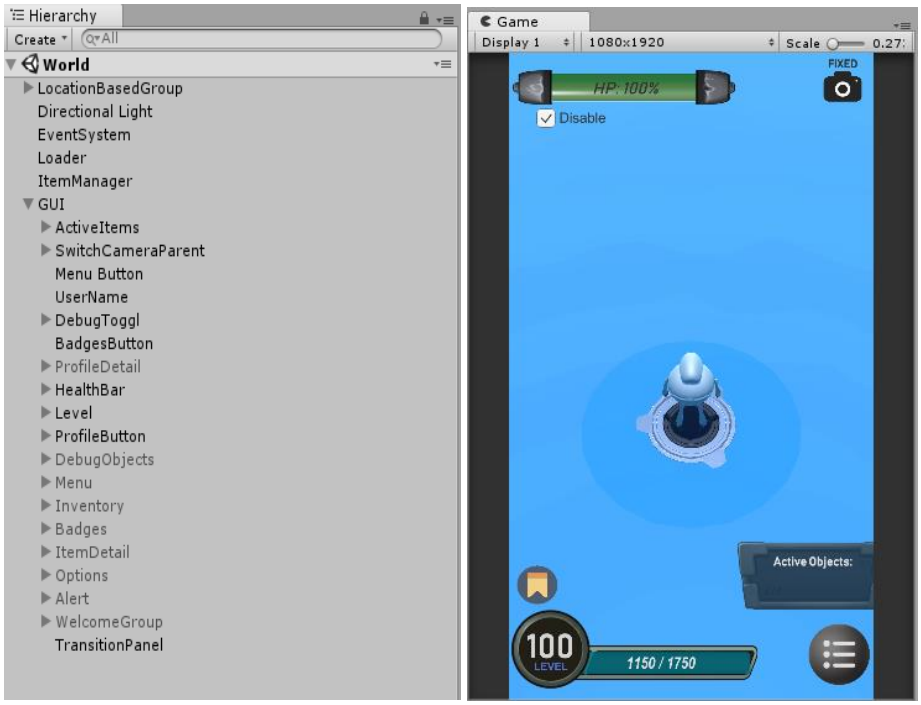
GAMEOBJECT	DESCRIPCIÓN	SUB-OBJETOS	DESCRIPCIÓN SUB-OBJETOS
LocationBased-Group	Grupo de *GameObjects* de realidad aumentada en el mapa. Se trata del grupo principal de objetos, obtenidos inicialmente a partir de un *prefab* de la librería Mapbox. Sirve como punto de partida, para realizar un proyecto basado en la ubicación, integrando todas las modificaciones de personalización.	Proveedor de ubicación	Se trata de un *Prefab* que incluye varios scripts de Mapbox combinados entre sí, para obtener la información de ubicación. Contiene a su vez varios *GameObjects*, para obtener los datos en distintas plataformas.
		Mapa	*GameObject* que se encarga de generar el mapa, junto con todas las

			reglas de personalización asociadas. Una de las configuraciones es la URL del mapa personalizado, que se puede diseñar en la página oficial de la librería.
		Auxiliar de cámara	Elemento empleado para cambiar las reglas de seguimiento de la cámara, en base al G*ameObject* padre. En este caso no se trata de un *GameObject* integrado en el *prefab* de localización, sino que se ha añadido posteriormente, por estar relacionado con el Jugador.
		Jugador	*Prefab* principal del grupo, que además de contener componentes de la librería de *MapBox*, dispone de otros objetos principales en la escena, entre ellos: modelo y animación del personaje, rango de captura asociado y cámara principal de la escena.
Directional Light	Se trata del *GameObject* de iluminación principal en la escena. Dispone de una intensidad intermedia, con la orientación perpendicular al plano del mapa.	*No tiene*	
EventSystem	*GameObject* sin malla de renderizado (no visible), que integra scripts para el control de eventos durante la ejecución (eventos interacción del jugador).	*No tiene*	
Loader	Como en el caso anterior, es un objeto no visible, que contiene la lógica entre escenas y el manejador principal del juego (persistencia entre partidas y escenas).	*No tiene*	

Item Manager	Nuevamente, se trata de un *GameObject* para controlar el comportamiento de otras entidades en la escena. En este caso, se encarga de gestionar los ítems, para su borrado, tiempo de activación, gestión de ids, etc.	*No tiene*	
GUI	*GameObject* que contiene un amplio grupo de elementos basados en entidades visibles. Entre ellas, podemos destacar varios sub-objetos.	Barra de vida	Representa la salud del jugador en tiempo real.
		Barra de experiencia y perfil.	Barra que refleja el progreso de nivel del jugador, así como dar acceso a su perfil pulsando sobre esta sección.
		Menú.	Da acceso al resto de submenús: - Inventario: Contiene toda la información sobre equipamientos y objetos. - Opciones: Permite ajustar y conocer más detalles de las distintas fases del juego. - Salir: Función para cerrar el juego.
		Medallas	Almacena la información de los lugares en los que se han superado puzles recientemente.
		Indicador de objetos activos	Muestra objetos con un tiempo de uso desde su activación.
		Botón para alternar cámaras.	Permite alternar entre cámara fija o libre (rotarla alrededor del jugador). Esta funcionalidad hace uso del "Auxiliar de cámara", entidad que comentábamos en secciones anteriores.

		Alerta (ventana emergente).	Permite mostrar mensajes con información relevante, para poner en contexto al jugador en la escena.

Tabla 2: Principales entidades de la escena Mapa



Figuras 35 y 36: Capturas que representan los elementos en la escena de Mapa
Fuente: Elaboración propia

7.2.2. Entidades Escena Batalla

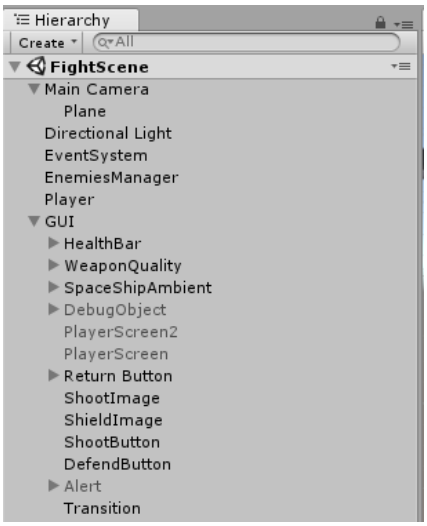
Se trata de la escena en la que nos enfrentamos a las naves enemigas en realidad aumentada. En ella, varios elementos que pueden verse en las *figuras 37 y 38* y en la siguiente *tabla 3*:

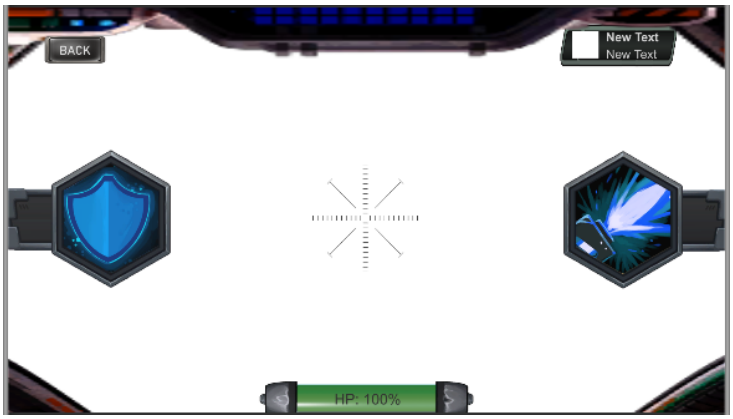
GAMEOBJECT	DESCRIPCIÓN	SUB-OBJETOS	DESCRIPCIÓN SUB-OBJETOS
Main Camera	Cámara principal de la escena, con un comportamiento modificado, por incluir varios	Plano de cámara.	Se trata de un plano asociado a la cámara de la escena, con un script que altera su comportamiento normal,

GAMEOBJECT	DESCRIPCIÓN	SUB-OBJETOS	DESCRIPCIÓN SUB-OBJETOS
	componentes de realidad aumentada para la batalla.		generando el efecto de realidad aumentada.
Directional Light	Se trata del GameObject de iluminación principal en la escena. Dispone de una intensidad intermedia, orientada hacia el plano de la cámara	*No tiene*	
Event System	GameObject sin malla de renderizado (no visible), que integra scripts para el control de eventos durante la ejecución (eventos interacción del jugador).	*No tiene*	
Enemies-Manager	Es uno de los Objetos principales de la escena, ya que se encarga de controlar todo lo referente a los enemigos: instanciarlos al comienzo de la escena, gestionar los turnos para atacar y comprobar si se ha ganado o perdido al final de la batalla (principalmente).	*No tiene*	
Player	Objeto de jugador con un comportamiento adicional al de la escena Mapa. En este caso, además de recuperar sus datos de la escena anterior, dispone de un script que controla las mecánicas de acción durante la batalla. Al tratarse de una batalla en 1ª persona, está centrado en la misma posición que la cámara, con una caja de	*No tiene*	

GAMEOBJECT	DESCRIPCIÓN	SUB-OBJETOS	DESCRIPCIÓN SUB-OBJETOS
	colisión para detectar los ataques recibidos.		
GUI	*GameObject* que contiene un amplio grupo de elementos basados en entidades visibles. Entre ellas, podemos destacar varios sub-objetos.	Barra de vida	Al igual que en la escena anterior, representa la vida del jugador. En esta escena toma un papel más relevante pues está en constante cambio debido a la batalla.
		Información de arma equipada	Proporciona información sobre el tipo de arma, su calidad y poder total. Esta arma debe ser equipada previamente desde el inventario de la escena Mapa. Por defecto: 15 de poder total.
		Elementos de la nave (*sprites*)	Elementos de ambientación y decoración.
		Botón de ataque	Permite disparar a los enemigos.
		Botón de defensa	Permite crear un escudo para protegerse de los enemigos.
		Botón para volver al Mapa	Permite volver a la escena Mapa en cualquier momento.
		Alerta (ventana emergente).	Permite mostrar mensajes con información relevante, para poner en contexto al jugador en la escena.

Tabla 3: Principales entidades en la escena batalla





Figuras 37 y 38: Capturas que representan los elementos en la escena de Batalla
Fuente: Elaboración propia

7.2.3. Entidades Escena Puzle

Se trata de la escena en la que pasamos a resolver los puzles/laberintos, utilizando una esfera y las propiedades físicas del entorno. En ella destacamos los elementos que pueden verse la *tabla 3 y las figuras 39 y 40*:

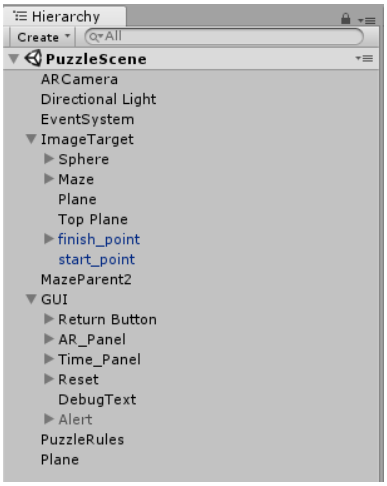
GAMEOBJECT	DESCRIPCIÓN	SUB-OBJETOS	DESCRIPCIÓN SUB-OBJETOS
AR Camera	Se trata de un *Prefab* obtenido de la librería *Vuforia*, que consiste en una cámara de la escena, con un comportamiento modificado, por incluir varios scripts de realidad aumentada que incluye esta librería.	*No tiene*	
Directional Light	Se trata del *GameObject* de iluminación principal en la escena. Dispone de una intensidad intermedia, orientada hacia el objeto puzle de la escena.	*No tiene*	
Event System	*GameObject* sin malla de renderizado (no visible), que integra scripts para el control de eventos durante la ejecución (eventos interacción del jugador).	*No tiene*	

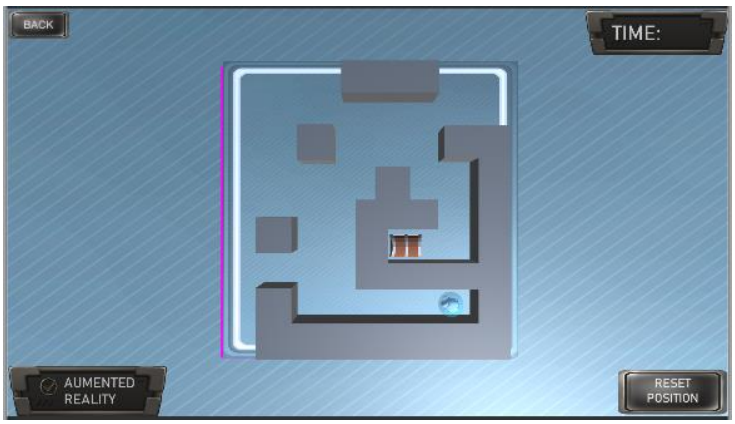
GAMEOBJECT	DESCRIPCIÓN	SUB-OBJETOS	DESCRIPCIÓN SUB-OBJETOS
Image Target	*Prefab* obtenido de la librería *Vuforia* que, junto a la cámara, permite realizar la realidad aumentada basada en marcadores. Es decir, en este objeto podemos establecer la imagen a reconocer, para generar en su superficie los objetos que necesitemos.	*Sphere*	Se trata de una esfera controlada por el jugador durante la escena. Aprovechando las propiedades físicas del entorno, el jugador debe conseguir que este objeto llegue a la meta. Tiene un script de comportamiento para controlar su reposicionamiento automático.
		Maze	Contiene los distintos tipos de escenarios a generar en la escena
		Plane	Plano inferior del laberinto, con caja de colisión, que permite que la esfera deslice correctamente para superar la prueba.
		Top Plane	Plano superior del laberinto, con caja de colisión, que impide que la esfera se salga del recorrido por error.
		Finish Point	Meta del recorrido del laberinto. Cuando su caja de colisión detecta a la esfera, se lanza el aviso de victoria.
		Start Point	Punto de inicio del recorrido, usado al empezar y a la hora de reposicionar la esfera.

GAMEOBJECT	DESCRIPCIÓN	SUB-OBJETOS	DESCRIPCIÓN SUB-OBJETOS
Maze Parent2	*GameObject* empleado para alternar entre la realización del puzle en realidad aumentada o sin realidad aumentada. Es el objeto padre opuesto a *Image Target*. Cuando los objetos se colocan dentro de este objeto, pasan a ser visibles de forma permanente.	*No tiene*	
GUI	*GameObject* que contiene un amplio grupo de elementos basados en entidades visibles. Entre ellas, podemos destacar varios sub-objetos.	*Return Button*	Botón que permite volver a la escena mapa en cualquier momento.
		AR Panel	Botón que permite activar o desactivar la realidad aumentada para realizar el puzle.
		Time Panel	Panel en el que se nos indica el tiempo restante para resolver el puzle.
		Reset	Botón que nos permite reiniciar la posición de la esfera en el laberinto.
		Alert (ventana emergente)	Permite mostrar mensajes con información relevante, para poner en contexto al jugador en la escena.
Puzzle Rules	Se trata de uno de los objetos principales de la escena puzle, ya que se encarga de controlar los objetivos/metas del jugador, el laberinto a generar, las posiciones por defecto, entre otros.		
Plane	Plano que tiene únicamente una función visual, dentro de la escena y que por defecto se encuentra oculto.		

GAMEOBJECT	DESCRIPCIÓN	SUB-OBJETOS	DESCRIPCIÓN SUB-OBJETOS
	Pasa a ser visible cuando la realidad aumentada está deshabilitada.		

Tabla 4: Principales entidades de la escena Puzle





Figuras 39 y 40: Capturas que representan los elementos en la escena de Puzle
Fuente: Elaboración propia

Una vez conocidas todas las entidades en escena, podemos pasar a describir cómo se ha implementado el juego, haciendo referencia a estos objetos explicados anteriormente.

7.3. Implementación de mecánicas

Al igual que en el apartado anterior, podemos explicar las distintas mecánicas implementadas clasificadas según la escena a la que pertenecen. Por tanto, en base a estas escenas tenemos:

7.3.1. Mecánicas Escena Mapa

- **Moverse:** El movimiento del personaje se ha desarrollado conforme estaba previsto en el GDD, es decir, en base a los sistemas de ubicación obtenidos por el dispositivo. Para ello, se ha hecho uso de la librería *Mapbox* que, una vez instalada dentro del proyecto de Unity, nos permite importar el *prefab* con los sistemas de ubicación preparados y adaptados a las distintas plataformas. Se trata del *prefab*

"*LocationBasedGroup*" explicado con anterioridad, que dispone del mapa, el proveedor de ubicación y el objeto jugador. Tras realizar algunos ajustes en la configuración y pruebas en dispositivos Android, comprobamos que su funcionamiento es correcto.

Para que el movimiento del jugador fuera más fluido, se han realizado algunos ajustes.

- En primer lugar, se ha modificado el script que proporcionaba la posición del jugador en el mapa, para interpolar este cambio de posición en todo momento.
- Por otro lado, se han realizado las animaciones del modelo de personaje para que, en caso de detectar cambios en su posición o rotación, realizara los gestos de movimiento correspondientes. Si el jugador pasaba a estar quieto se realizaba la transición de una animación a la otra. Todas estas animaciones se han realizado con la herramienta online *Mixamo* (Mixamo, Adobe, 2019).

- **Mover la vista (cámara):** Esta mecánica relacionada con la cámara y el jugador, se puede entender de dos formas posibles, planteadas en el GDD: Cámara fija y cámara libre. Para ambos tipos, se emplea un script de comportamiento en la cámara: *CameraMovement.cs*, que habilita o deshabilita algunas funciones, así como su posicionamiento por defecto respecto al jugador (ligeramente alejada a su espalda).
  - Cámara fija asociada: Por un lado, tenemos la rotación de la cámara con el jugador, es decir, esta se encuentra orientada en todo momento hacia donde mira el jugador. Para realizar esta funcionalidad, se ha establecido el jugador como objeto padre de la cámara principal de la escena. De esta forma, todas las transformaciones de posición, rotación y escala del objeto jugador se aplicarán también a la cámara en cuestión. La rotación la experimenta el jugador a través del giroscopio, que a su vez hace que la cámara gire con este. La única transformación que el usuario puede aplicar a la cámara de forma independiente es el zoom, controlado en el script mencionado anteriormente. Para realizar esta acción se comprueba que el jugador haya pulsado dos posiciones de la pantalla a la vez, además de comprobar el desplazamiento que experimentan estos puntos (para alejar o acercar).
  - Cámara libre asociada: En este caso, el jugador deja de ser el objeto padre de la cámara, para evitar que sus transformaciones impidan su libre movimiento, pasando a ser hija del *GameObject* Auxiliar de cámara. A partir de este momento, el jugador puede realizar todo tipo de transformaciones en la

cámara dentro de unos límites: aplicar zoom (como en el caso anterior), rotar y mover la cámara alrededor del personaje en para varias alturas. Para desplazar la cámara se comprueba que el jugador haya pulsado solo 1 punto de la pantalla, así como el desplazamiento que experimenta y la dirección en la que lo hace. La cámara se mantiene orientada hacia el objeto de jugador en todo momento.

- **Recoger objetos**: Los objetos que aparecen en los distintos puntos del mapa, pueden ser recogidos cuando el jugador pulse sobre ellos. Esa pulsación es controlada mediante la función de *onclick* de Unity, situada en el script que controla el comportamiento de las entidades del mapa. Además, todos ellas precisan de una caja de colisión. Sin embargo, deben cumplirse otros requisitos para que pasen a añadirse al inventario y se eliminen del mapa. Los otros requisitos son:
    - Rango de captura: Se trata de un objeto, con una malla de colisión, asociado al jugador y que permite detectar si los objetos se encuentran dentro del rango de captura necesario para alcanzar ciertos objetos u objetivos. Este objeto dispone de un script para realizar estas comprobaciones. Si el rango de captura colisiona con algún ítem del mapa, podrá ser recogido por el jugador cuando pulse sobre él. Se identifican por un atributo conocido como "*tag*". Si no está dentro del rango, este objeto no se añadirá.
    - Capacidad de Inventario: Otro factor a tener en cuenta es el espacio disponible del inventario del jugador. En este caso el inventario se ha establecido a una capacidad de 27 objetos por un lado y 27 por otro para las armas. Si se alcanza este límite y se intenta insertar un nuevo objeto, este no se insertará. Esta comprobación se realiza directamente en el script de inventario, asociado al jugador.
- **Interactuar con puntos de combate o puzle:** Se trata de una mecánica muy similar a la de recogida de objetos por la forma en la que el jugador la emplea. De nuevo, el jugador debe pulsar sobre el punto de combate o puzle para acceder a él. Sin embargo, en cuanto a limitaciones se refiere, encontramos una diferencia respecto a la mecánica anterior:
    - Rango de captura: El funcionamiento es el mismo ya explicado en el apartado anterior, con la particularidad de detectar la colisión con este tipo de elementos y no ítems.

    - Barra de vida del jugador: En este caso, para los combates, es necesario tener una cantidad de salud superior al 0%. Si no se cumple este requisito, aparecerá una ventana emergente (*GameObject Alert*) indicando que la cantidad de salud del jugador no puede ser cero.

- **Usar objetos:** Los objetos pueden ser usados desde la ventana de inventario. Cuando accedemos al apartado correspondiente, únicamente debemos pulsar sobre el objeto que queremos usar. En ese momento aparecerá una ventana para confirmar la acción. De ser así, el objeto desaparecerá del listado del inventario y hará la acción que le corresponde.
- **Eliminar objetos/armas:** Al igual que en el caso anterior, el jugador debe acceder a la sección del inventario. Una vez allí debe pulsar sobre el icono de la papelera, situado en la parte superior derecha. Con esto aparecerán unos iconos de cruces encima de los objetos del inventario, para que podamos seleccionar aquellos que queramos borrar. Al pulsar sobre alguno, nos aparecerá otra ventana de confirmación de borrado. En ese caso, el objeto desaparecerá de la lista de objetos sin realizar ninguna opción. Se trata de una mecánica útil para aquellos momentos en los que tengamos el espacio limitado y no nos interese gastar ningún objeto.
- **Equipar armas:** De nuevo, en el inventario, debemos dirigirnos a la sección de equipamiento. Una vez allí, para seleccionar un arma, solo debemos pulsar sobre ella, al igual que haríamos para usar un objeto. Hecho esto, esta pasará a estar la primera de la lista, resaltada con un fondo azul.
- **Consultar detalle de un objeto:** En la parte superior derecha del inventario, al lado del icono de la papelera, se encuentra el icono de información. Si pulsamos sobre este, aparecerá un icono de información sobre cada uno de los objetos del inventario. Si pulsamos sobre alguno de estos iconos, accederemos al detalle del objeto en cuestión.
- **Consultar perfil:** Si pulsamos sobre el icono de nivel, podemos acceder a la información del perfil.
- **Consultar opciones (y respectivos subapartados):** Desde el menú principal (desplegable), podemos acceder al menú de opciones a través del icono correspondiente. Una vez allí, nos encontramos con varias sub-opciones:
    - Ajustar volumen de juego (sonidos).
    - Consultar opciones de realidad aumentada

  - Consultar cómo jugar
  - Consultar créditos.

  Todas estas opciones se encuentran controladas por el gestor de UI, que una vez abierta la ventana de opciones pasa a tener el control el script de opciones, gestionando hacia donde navega el usuario según su elección.

- **Salir del juego:** Desde el menú principal (desplegable), la última de las 3 opciones es salir del juego. Al pulsar sobre el icono correspondiente, solo debemos confirmar la acción desde la ventana emergente que aparece en pantalla. Esta acción es controlada por el gestor de UI (GUI), que llama a *GameManager*, para cerrar la aplicación.

7.3.2. Mecánicas Escena Batalla

- ***Apuntar (naves enemigas):*** Mecánica gestionada por el script de comportamiento de la cámara en la escena que, a su vez, hace uso el plano principal al que se encuentra orientada. Se trata de una combinación de tecnologías, basadas en la rotación del dispositivo. Por un lado, cuando la cámara del dispositivo se mueve, la textura del plano se modifica para mostrar la misma imagen que captura el dispositivo. Por otro lado, la rotación experimentada por el dispositivo durante este movimiento es capturada por el giroscopio, sincronizando así la rotación para que los elementos de cámara y plano giren en la escena de Unity. De esta forma se genera el efecto de apuntar, creando un efecto de realidad aumentada con todos los elementos interpuestos entre la cámara de la escena y el plano (que representa las imágenes de la cámara del teléfono).
- ***Protegerse/esquivar:*** Se trata de una mecánica que permite evitar que el jugador reciba daño durante un tiempo. Para activarlo solo debe pulsar en el icono del escudo situado en el lado izquierdo de la pantalla. Este botón lo que hace es llamar al script del jugador en combate, para establecer la cantidad de tiempo de defensa, en base al arma que tenga equipada. También se activa una pantalla azul y semitransparente, intentando representar que el escudo está activo.
- ***Disparar:*** Mecánica clave dentro de la escena de batalla que permite al jugador atacar a las naves enemigas, disparando balas con diferentes características en base al arma equipada. Puede usarlo pulsando el botón/icono de disparo, situado en la parte derecha de la pantalla. Con él, se llamará al script del jugador que, a su vez, llamará al método de disparo que tiene el arma equipada. Se instanciará entonces un objeto

bala que tendrá más daño y/o más velocidad, en base a las características del arma. Esta bala, está ajustada para ser destruida en cuanto detecta una colisión con un enemigo, aplicando el daño correspondiente. Sin embargo, como puede que no colisione con un enemigo, se ha establecido un temporizador de destrucción del objeto en 3 segundos, cuando ya no es visible por el jugador.

- ***Disparo especial:*** no implementado para el jugador, únicamente para el enemigo (más detalle en punto 7.5).
- ***Recoger color en la imagen (mejora opcional):*** no implementado.
- ***Recoger armas/armaduras:*** se ha optado por integrar esta funcionalidad únicamente en la escena mapa.
- ***Volver al mapa:*** Se encuentra en la parte superior izquierda. Al pulsar en este botón, el script llama al Gestor de escenas para volver a la escena anterior.

7.3.3. Mecánicas Escena Puzle

- ***Activar o desactivar la realidad Aumentada:*** Mecánica que afecta a la forma de resolver el puzle de la escena. En caso de estar habilitada (por defecto sí), la cámara se mantendrá activa con tal de reconocer la imagen objetivo. Si lo desactivamos el tiempo para conseguir el objetivo se reduce a la mitad y la cámara se apagará, pasando a mostrar directamente el puzle de la escena.
- ***Apuntar con la cámara (reconocimiento de imagen):*** Mientras la realidad aumentada esté activa, el jugador puede hacer uso de la cámara del dispositivo para apuntar hacia la superficie objetivo. Esta cámara está generada a partir de un *Prefab* de la librería *Vuforia*, cuyos scripts se encargan de controlar el reconocimiento de imagen y la realidad aumentada.
- ***Reposicionar esfera:*** Mecánica que simplemente se encarga de establecer la posición de la esfera al inicio. Para ello, solo hay que pulsar el botón situado en la parte inferior derecha de la pantalla, que a su vez llamará al script de la esfera para establecerle la posición del objeto de inicio (invisible y estático).
- ***Volver al mapa***. Al igual en que el caso anterior, se encuentra en la parte superior izquierda. Al pulsar en este botón, el script llama al Gestor de escenas para volver a la escena anterior.
- ***Mecánicas avanzadas de interacción con objetos 3D:*** Para completar el puzle planteado esta escena, debemos aprovechar las propiedades físicas del ambiente, en

este caso: las propiedades deslizantes y la gravedad. Con esto en cuenta, la mecánica de acción principal de la escena es movimiento de la esfera. Esta mecánica permite que la esfera llegue al punto final del laberinto para ganar. Se puede dividir, en función de si la realidad aumentada está activa o no:

- ***Movimiento de la esfera:*** Por un lado, si la realidad aumentada no está activa, el movimiento de la esfera pasa a depender únicamente del acelerómetro del dispositivo. De inicio, posición del dispositivo debe ser perpendicular al suelo, para evitar que se deslice la esfera y caída de la plataforma. Cuando empecemos a inclinarlo, se deslizará hacia el lado correspondiente, simulando así la pendiente creada en el plano y la gravedad ejerciendo.
- ***Movimiento de la esfera (AR):*** Por otro lado, si la realidad aumentada está activa, el movimiento de la esfera pasa a depender de la superficie con la imagen objetivo. Para resolver el puzle de este modo, necesitaremos de forma obligada usar las 2 manos: por un lado para sujetar el dispositivo, apuntando a la superficie y por otro, para crear la inclinación necesaria en cada caso para que la esfera avance. La mecánica de deslizamiento de la esfera es la misma que en el caso anterior, pero en superficies diferentes.

7.4. Desarrollo de personajes y objetos

En este apartado, se va a explicar cómo se han desarrollado los personajes y objetos del juego, así como su contraste respecto a la idea inicial. Cabe decir que estos elementos se encuentran concentrados entre las escenas de mapa y batalla. Sin embargo, analizaremos aquí también otro tipo de entidades, que puede considerarse un ítem especial, pero que guarda muchas similitudes con los enemigos que encontramos en el mapa

7.4.1. Objetos

Para empezar, los objetos desarrollados en el juego podemos dividirlos en 2 tipos: armas/equipamientos y objetos o ítems de único uso. En ambos casos estos objetos se almacenan en el inventario, pero la forma de obtenerlos puede ser muy distinta. Por ello, pasamos a analizarlos por separado a continuación:

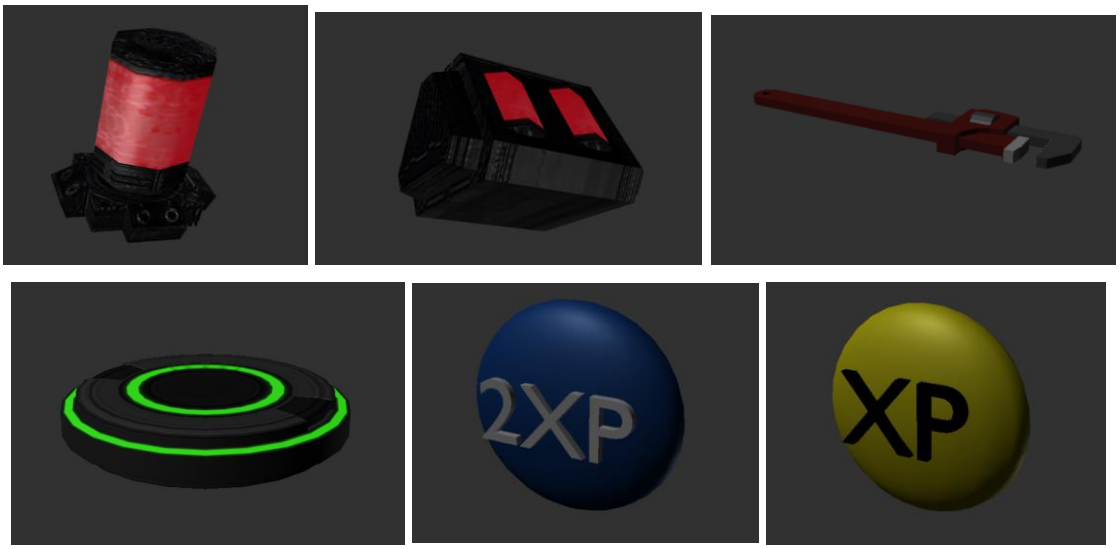
7.4.1.1. Objetos de uso único

Los ítems de uso único tienen como objetivo dar una ventaja temporal al jugador, ya sea de forma instantánea o por un tiempo. Una vez el objeto es usado, se borra del inventario, dejando libre ese espacio. La forma de obtener estos objetos siempre se realiza en la escena mapa, ya que se encuentran distribuidos en distintos puntos clave del mismo. Simplemente, el jugador debe acercarse lo suficiente (dentro del rango de captura) y pulsar sobre ellos, para que desaparezcan del mapa y se introduzcan en el inventario. La estructura componentes de un *GameObject* de tipo ítem es la siguiente (*tabla 5 y figuras* de *41* a *46*):

COMPONENTE	**DESCRIPCIÓN**
Malla de renderizado asociada (*Mesh Renderer*).	Se trata del modelo visible del objeto en el mapa, con la textura correspondiente aplicada. Permite identificar un objeto antes de ser recogido.
Item pickup	Script de comportamiento que controla la recogida de objetos en el mapa. Hereda de una clase llamada *MapEntity*, que contiene todas las características básicas de los objetos del mapa, centradas en el rango de captura. Cuando se recoge un objeto, este script pasa a estar desactivado.
Item	Script que se encarga de definir la función de cada objeto, así como de establecer los valores de aleatoriedad, en cada uno de ellos. Este script es la clase padre de los distintos tipos de ítems, entre los que podemos identificar:
	Health Item: Objeto de curación básico, que puede restablecer la salud del jugador entre 5 y 30 puntos de vida. Tiene un uso inmediato, por lo que una vez realiza su efecto, es eliminado. Para aplicar esta salud, se llama al método de la clase jugador para aplicar la salud adicional. El modelo asociado a este ítem se ha obtenido de un *asset* externo de Unity. Puede verse en la figura.
	Big Health Item: Objeto de curación superior, que restablece la salud del jugador entre 40 y 60 puntos. Como en el caso anterior, tiene un uso inmediato. El modelo asociado a este ítem se ha obtenido de un *asset* externo de Unity. Puede verse en la figura.
	Durability Up: Objeto de reparación de armas que permite restablecer la durabilidad del arma equipada entre 5 y 30 puntos. Tiene uso inmediato, siempre y cuando el jugador tenga

COMPONENTE	DESCRIPCIÓN
	un arma equipada. El modelo asociado a este ítem se ha obtenido de un asset externo de Unity. Puede verse en la figura.
	Extended Capture Range: Objeto que aumenta el rango de captura del jugador. A diferencia de los casos anteriores, no es de uso instantáneo, sino que permanece activo durante unos segundos tras ser usado, por defecto 50 segundos. A nivel de implementación, se llama al rango de captura asociado a la clase jugador, para modificar así su índice de crecimiento. Cuando se acaba el tiempo, vuelve a su valor por defecto. El modelo asociado a este ítem se ha obtenido de un *asset* externo de Unity. Puede verse en la figura.
	XP Multiplier: Objeto que multiplica por 2 la experiencia ganada por el jugador durante un tiempo. Como en el caso anterior, tras su uso, se mantiene activa durante unos segundos, en este caso 30. Esto se aplica para todas las escenas, es decir, si el jugador gana en la escena puzle, escena batalla o consigue algo de experiencia en la escena mapa, la experiencia ganada será el doble. El modelo asociado a este ítem se ha realizado en *Blender* 3D. Puede verse en la figura.
	XP bonus: Se trata de un objeto especial que no se añade al inventario y que únicamente se encarga de añadir experiencia al jugador, llamando al método para ello en la clase de jugador. Por tanto, a diferencia de los demás ítems, no dispone de un script de comportamiento de ítem, ya que la acción se ejecuta al ser recogido. El modelo asociado a este ítem se ha realizado en el software de modelado *Blender 3D*.
Box Collider	Caja de colisión del objeto que permite ser recogido por el jugador cuando está lo suficientemente cerca. Al igual que "*Item Pickup*", pasa a ser desactivado cuando se recoge del mapa, evitando que pueda ser recogido varias veces.
Otros scripts de posicionamiento (opcional)	Se trata de scripts que modifican la posición y/o rotación de los objetos en el mapa. Solo se emplean para algunos objetos, con propósitos visuales.

Tabla 5: Componentes del Prefab de Item



Figuras 41, 42, 43. 44, 45 y 46: Capturas que representan los objetos en la escena Mapa
Fuentes: http://bit.ly/2VyvkXh0 (38 y 39), http://bit.ly/2Qe8Qdo (40), http://bit.ly/2EgrKvi (41) y Elaboración propia (42 y 43)

En la siguiente *figura 47*, se pueden identificar los componentes descritos en el *GameObject* correspondiente de Unity.

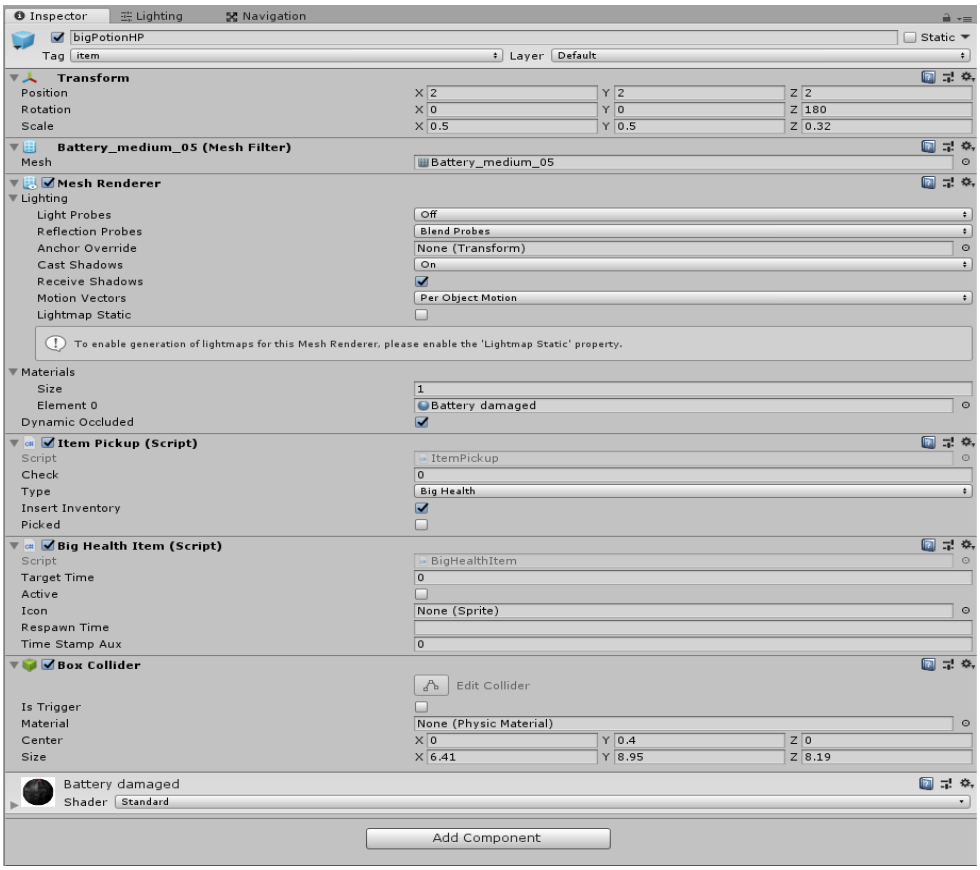


Figura 47: Captura que representa los componentes de los objetos en el Mapa
Fuente: Elaboración propia

7.4.1.2. Armas/Equipamiento

Al igual que en el caso anterior, las armas son objetos. Sin embargo, en este caso cumplen una función muy distinta. En primer lugar, no son de un solo uso, ya que pueden ser equipadas varias veces por el jugador. La única limitación que tienen es su durabilidad, es decir, se pueden utilizar mientras esta no llegue a cero. Además, solo puede ser equipada una a la vez y no suponen ningún efecto u acción visible hasta que no se accede a la escena de combate, en la que podemos comprobar su mejora en ataque y defensa. En cada combate, la durabilidad del arma equipada se verá reducida. Las armas pueden obtenerse tanto en el mapa como al subir de nivel desde cualquier escena.

El *GameObject/prefab* que lo define es único, pues todas las armas tienen el mismo comportamiento. Lo que diferencia unas armas de otras son sus características, que vienen dadas por los distintos parámetros del *GameObject*. Su modelo 3D que aparece en el mapa puede verse en la siguiente *figura 48*:

Figura 48: Captura del objeto arma en el Mapa
Fuente: http://bit.ly/2VNAwLU

En cuanto a su estructura, a nivel de componentes podemos identificar aquellos reflejados en la tabla 6 y la *figura 49* (muy similar a los ítems):

COMPONENTE	DESCRIPCIÓN
Mesh Renderer	Malla de renderizado asociada.
Item Pickup	Su comportamiento es el mismo que para el caso de los objetos, pero se inserta en una sección diferente del inventario (otra lista diferente).
Box Collider	Caja de colisión para el rango de captura.
Equipment	Script de comportamiento del arma, que establece todos los valores aleatorios al ser recogido. Los valores alterables son:
	Tipo de arma: Se establece el tipo de arma en base a la cualidad de ella que más destaca (probabilidad a partes iguales) y puede ser: equilibrio, defensa, ataque y velocidad.
	Calidad: Se establece la calidad del arma, en base a los siguientes porcentajes: • Legendaria con un 5% y con los mejores atributos posibles. • Épica con un 15% de probabilidad y con unos atributos muy buenos. • Rara con un 30% de probabilidad y con atributos intermedios. • Básica con un 50% de probabilidad y con atributos bajos.
	Atributos/parámetros de combate: Son los valores que realmente afectan a la hora de combatir y son: velocidad, defensa y ataque. Se establecen en base a la calidad y el tipo de arma, incrementando los valores a mayor calidad, pero adaptándose al tipo de arma en cada

COMPONENTE	DESCRIPCIÓN
	caso. Por ejemplo, si el arma es defensiva el mayor atributo de los tres será el de la defensa del arma. Los efectos en combate son: • Defensa: Mayor tiempo de activación de escudo (inmunidad). • Ataque: Mayor daño infligido por las balas a los enemigos. • Velocidad: Mayor cadencia de disparo (tiempo de refresco).

Tabla 6: Componentes del Prefab de Equipamiento

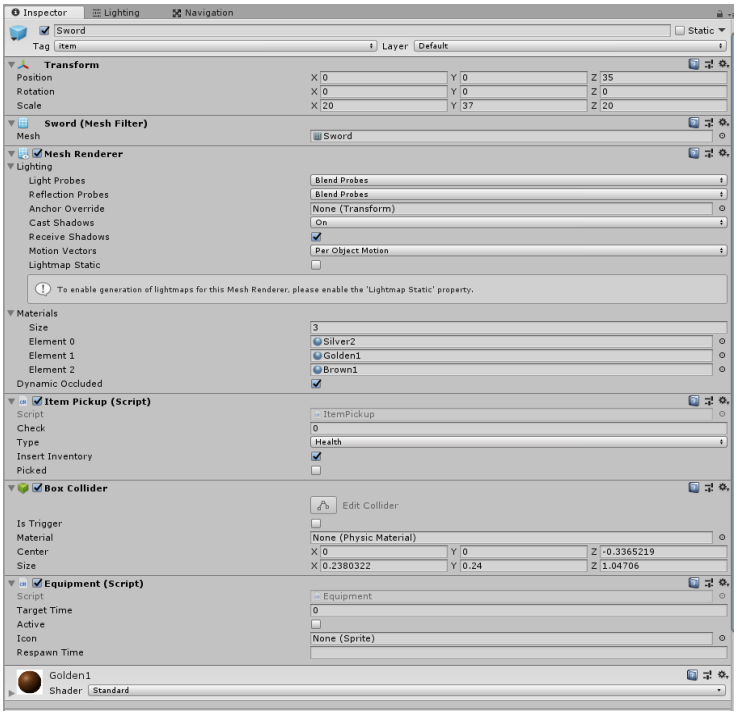


Figura 49: Captura que representa los componentes del arma en el Mapa
Fuente: Elaboración propia

7.4.2. Personajes

Una vez analizados los distintos tipos de ítems, pasamos a revisar los personajes que podemos encontrar en escena.

Durante el juego podemos distinguir 2 tipos de personajes: jugador protagonista y enemigos, con sus respectivos subtipos analizados en el GDD. A continuación, vamos a explicarlos, enunciando y describiendo cada uno de sus componentes. Por tanto, introduciremos a continuación los personajes desarrollados.

7.4.2.1. Jugador

El protagonista que controlamos durante el juego tendrá una estética futurista, por tratarse de un humano con armadura robótica, capaz de hacer frente a la invasión de droides enemigos.

El modelo 3D se ha realizado en Blender y el resultado puede verse en las *figuras 50 y 51*:

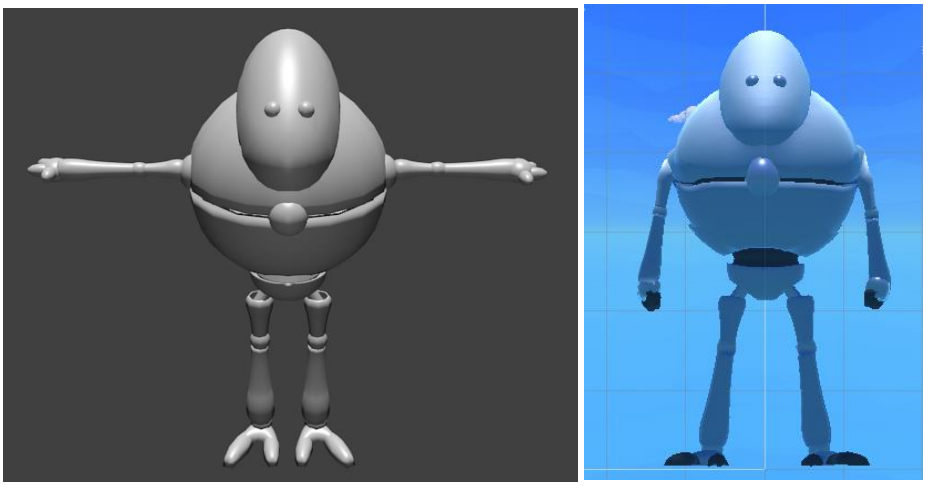


Figura 50 y 51: Capturas del personaje en la escena Mapa
Fuente: Elaboración propia mediante Blender

Aunque no lo controlemos igual en todas las escenas, es una entidad del juego que está presente en cada una de ellas, por contener información relevante en todo momento (salud, experiencia, bonificaciones, etc.). Con las distintas armas que se puede equipar, es capaz de defenderse y atacar a distancia a los enemigos en combate.

Su Prefab en la escena Mapa de Unity, contiene varios componentes que establecen su comportamiento. Podemos destacar aquellos reflejados en la tabla 7:

COMPONENTE	DESCRIPCIÓN
Animator	Componente que permite establecer el control de animaciones del modelo 3D. Para ello, simplemente se ha establecido el avatar al que se aplican las animaciones, la configuración del flujo entre animaciones y los parámetros para realizar la transición entre ellas. Puede verse en la siguiente figura. En este caso, el flujo entre animaciones es sencillo, pues únicamente se realiza la transición entre 2 de ellas: “idle” cuando se encuentra estático y “move” cuando detecta movimiento, alternándose en base al parámetro “walk”. La transición se controla dentro el script de comportamiento del jugador que se comentará en los puntos siguientes.

COMPONENTE	DESCRIPCIÓN
Rotate With Location Provider	Se trata de uno de los Script de comportamiento integrados en el Prefab de jugador en Mapbox. Permite establecer la rotación del jugador en base al giroscopio del dispositivo.
Immediate Position With Location Provider	Es otro script de comportamiento de Mapbox, que establece la posición del jugador por Geolocalización, obtenida del dispositivo. En caso de no estar disponible, pueden ocurrir dos cosas: Si se encuentra en el editor de Unity recoge por fichero una ruta predeterminada (coordenadas), sino se queda estático con el mapa por cargar.
Player	Script de comportamiento principal del jugador, que se encarga de gestionar mayoritariamente: la vida, el nivel, la experiencia, el bonus de objetos activos, el rango de captura asociado y el control de animaciones del modelo 3D. Guardar esta información entre escenas es de gran importancia por ello, se reutiliza el script en todas las escenas. El comportamiento del rango de captura se gestiona desde otro *GameObject* distinto al del jugador (objeto hijo del jugador). Sin embargo, este script tiene la referencia hacia el objeto de rango de captura, por si se aplicará alguna mejora, comunicarla directamente para ser actualizada. Por lo que respecta al control de animaciones, se ha realizado en base a la función de Unity *"transform.hasChanged"* que comprueba entre iteraciones si la componente de transformación del objeto *"Player"*, ha sido modificada.
Inventory	Script que se encarga de gestionar la información y la lógica del inventario asociado al objeto jugador. Desde esta clase se trabaja con las listas de objetos y armas, de forma que puedan ser añadidas, borradas o modificadas en cualquier momento. Por ello, esta clase aplica el patrón de diseño *Singleton*, para que pueda ser llamada desde cualquier script sin problemas y para que el juego limite la creación de 1 único inventario.
Badges	Se trata de un script que maneja la lógica e información de las medallas obtenidas por el jugador al superar los distintos puzles. Su comportamiento es similar al caso del inventario pues se trata de una clase establecida como Singleton, que maneja una lista de objetos (en este caso medallas) y solo nos interesa que pueda haber 1.

Tabla 7: Componentes del Prefab de jugador en la escena Mapa

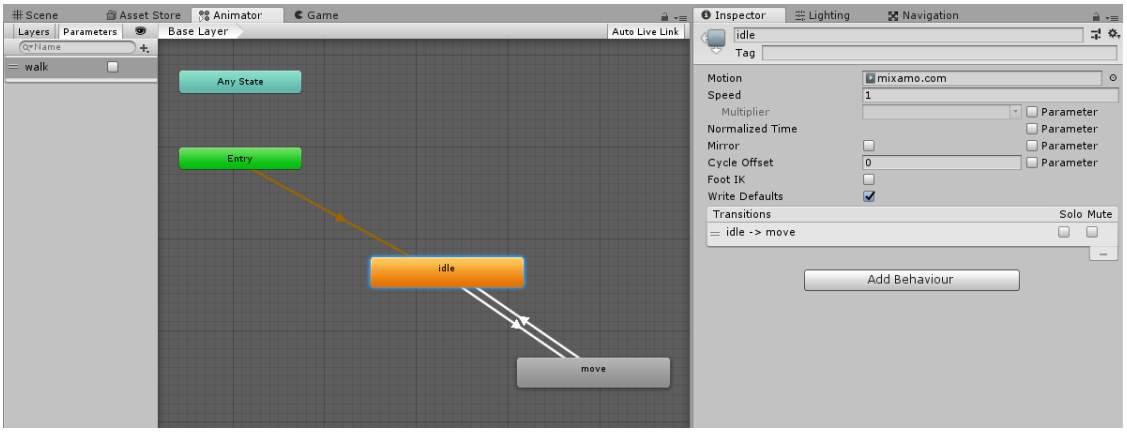


Figura 52: Captura que representa el flujo de las animaciones del jugador
Fuente: Elaboración propia

Por otro lado, mientras que en la escena puzle su *Prefab* solo usa el componente script "*Player*" visto anteriormente, en la escena batalla encontramos (componentes *tabla 8*):

COMPONENTE	DESCRIPCIÓN
Player Fight	Script de jugador adaptado a la escena de batalla, que contiene métodos de ataque, defensa, daño, sanación y otros efectos especiales. Se trata de un script que hereda de la clase padre "entidad de batalla", que comparte junto a los enemigos por reutilizar mecánicas comunes. Se combina con el gestor de UI para dar información extra al jugador durante el combate.
Player	Se trata del script principal comentado anteriormente. En este caso no recoge la referencia del rango de captura, pues no existe en la escena en cuestión.
Box Collider	Caja de colisión necesaria para detectar si ha sido atacado por algún enemigo y debe recibir daño (siempre y cuando el escudo no esté activo). En ese caso, los métodos de "*Player Fight*" recogerán la colisión para actualizar la vida en "*Player*".
Weapon	Arma equipada por el jugador durante el combate generada a partir de los datos de equipamiento de la escena Mapa. Gestiona los ataques, instanciando los objetos tipo "Bala" y la defensa de las entidades en batalla. Por tanto, influye en el daño de las balas, el tiempo de duración de los escudos y la velocidad de disparo.

COMPONENTE	DESCRIPCIÓN
Audio Source	Gestor de sonidos en los que está involucrado el jugador durante la escena.

Tabla 8: Componentes del Prefab de jugador en la escena Batalla

7.4.2.2. Enemigos

Durante el juego, podemos encontrar hasta 3 tipos de enemigos, explicados anteriormente en el GDD. En este caso, vamos a detallar como están desarrollados estos enemigos y los componentes que incluyen, sin entrar en detalles de sus rutinas de comportamiento en batalla, que se definirán más adelante en el apartado de Inteligencia Artificial.

Los modelos 3D de los enemigos en ambas escenas, se han obtenido de fuentes externas, pertenecientes a www.free3d.com y a www.opengameart.org y su resultado se ve reflejado en las *figuras 53, 54 y 55*:

Figuras 53, 54 y 55: Capturas de los enemigos: básico, avanzado y jefe (respectivamente)
Fuentes: http://bit.ly/2Vysvp9 (53)
http://bit.ly/2VOJCbm (54 y 55)

Comenzando por la escena Mapa cabe decir que, a nivel de componentes, todos los *GameObjects* enemigos son iguales. Únicamente se diferencian entre sí por un parámetro que representa el tipo de enemigo y la malla de renderizado de cada uno de ellos (modelo 3D y textura). Con esto, la estructura que encontramos en estos *Prefabs* (*tabla 9*):

COMPONENTE	DESCRIPCIÓN
Mesh renderer	Malla de renderizado de la entidad.
Droid	Script principal de comportamiento en el mapa que establece el tipo de enemigo y la acción a realizar cuando se hace click en su caja de

COMPONENTE	DESCRIPCIÓN
	colisión. Hereda de la clase *MapEntity*, la cual se aplica a todas las entidades de mapa, para generalizar el mecanismo de rango de captura cuando el jugador está cerca. De ser así, cuando el jugador toque al enemigo, se iniciará una transición hacia la escena batalla.
Box Collider	Caja de colisión asociada al enemigo, necesaria para detectar el rango de captura.
Audio Source	Gestor de sonidos en los que está involucrado el enemigo durante la escena.

Tabla 9: Componentes del Prefab de enemigo en la escena Mapa

Por otro lado, si analizamos el *Prefab* del enemigo en la escena batalla podemos identificar los siguientes componentes (*tabla 10*):

COMPONENTE	DESCRIPCIÓN
Mesh Renderer	Malla de renderizado.
Enemy Fight	Script de comportamiento principal del enemigo en la escena. En primera instancia, se encarga de establecer el tipo de enemigo con sus parámetros de combate: arma, vida (en la que tiene un importante factor el nivel del jugador) y la experiencia asociada al ser derrotado. Más adelante encontramos las distintas mecánicas que puede realizar el enemigo en combate, como son: atacar, defenderse, curarse, recibir daño, control de colisiones, etc. Al igual que ocurría con *Player Fight*, este script hereda de la clase *FightEntity*, por lo que comparten muchas funcionalidades en combate.
Panda Behaviour	Script generado a partir de la librería Panda, para la creación de *behaviour trees.*
Enemy Behaviour	Script de comportamiento de la inteligencia artificial, asociado al script de Panda. Ambos se analizarán con mayor detalle en el apartado de Inteligencia Artificial. En este caso se encarga de asociar el *behaviour tree* con las mecánicas de "*Enemy Fight*". Podemos identificar 3 tipos de scripts de este tipo, en base a los 3 tipos de enemigos: *Enemy Behaviour, Enemy Advanced Behaviour y Enemy Boss Behaviour*.

COMPONENTE	DESCRIPCIÓN
Weapon	Se trata del mismo script de comportamiento de arma que incluía el jugador en batalla, pues todas las entidades de batalla dependen de su arma para la poder realizar correctamente las principales mecánicas de combate (atacar y defenderse). En función del tipo de enemigo las características del arma serán más o menos poderosas.
Box Collider	Caja de colisión asociada al enemigo para detectar los ataques del jugador.
Rigid Body	*Rigibody* asociado a la caja de colisiones del enemigo. Todas las entidades en movimiento con físicas deben incluirlo para conseguir una correcta identificación de las colisiones.
Audio source	Gestiona los sonidos asociados a los enemigos.

Tabla 10: Componentes del Prefab de enemigo en la escena Batalla

Además de los componentes analizados con anterioridad, el enemigo en batalla dispone de dos sub-objetos a destacar:

- Cápsula de escudo: Se trata de un elemento visual que indica cuando el enemigo tiene activado el escudo. Envuelve completamente a su malla de renderizado, con una opacidad de textura reducida. Se refleja en la *figura 56*:

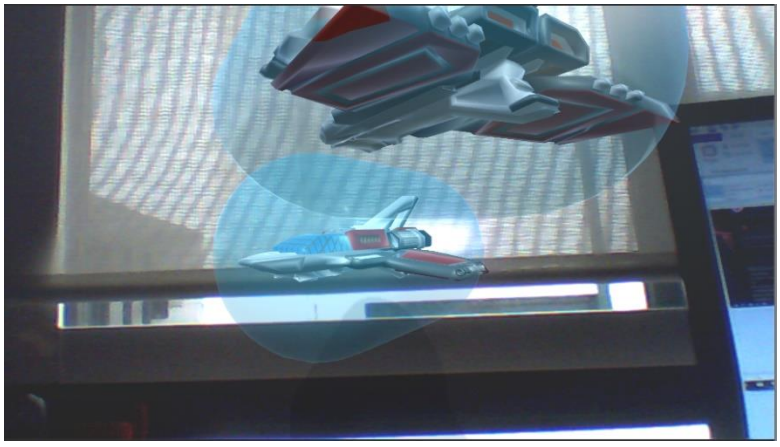


Figura 56: Cápsula/escudo que envuelve al enemigo
Fuente: Elaboración propia

- Estela de la nave: Se trata del rastro que deja la nave en la parte trasera, simulando el trabajo que realiza el motor para desplazarse. Para su realización se ha empleado un sistema de partículas que se incluyen en los "*Standard Assets*" de Unity.

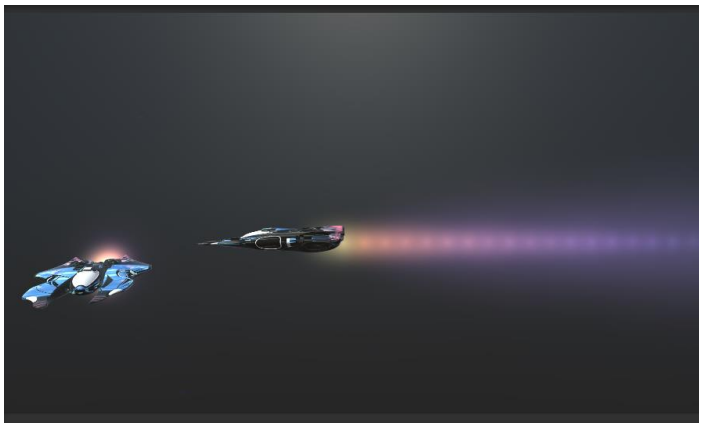


Figura 57: Enemigos con el rastro de estela (partículas)
Fuente: Elaboración propia

Por lo que respecta a la escena puzle, los enemigos no tienen ningún tipo de presencia.

7.4.3. Otras entidades

Existen otros tipos especiales de entidades en la escena del Mapa que no son ni un ítem, ni un enemigo, que a continuación se introducen:

En uno de estos casos, encontramos más correspondencia con los enemigos que con los ítems por su estructura y acciones. Se trata de los puntos de puzle, es decir, aquellos sitios desde los que podemos acceder a la escena puzle. Para ello, se debe estar dentro del rango de captura, acercándose lo suficiente para entrar. Al igual que con los enemigos, si pulsamos sobre ellos, se iniciará una transición para realizar el cambio de escena.

El modelo 3D se ha obtenido a partir de un asset de terceros en Unity (pero con licencias de uso gratuitas). El resultado puede verse en la siguiente *figura 58*:

Figura 58: Captura de un punto de Puzle
Fuente: http://bit.ly/2Ec8I9p

El *Prefab* empleado para estos puntos de puzle contiene los componentes (*tabla 11*):

COMPONENTE	DESCRIPCIÓN
Mesh Renderer	Malla de renderizado del punto de puzle.
Start Puzzle	Script principal del objeto que controla la transición hacia la escena puzle cuando se hace click dentro del rango de captura. De nuevo, este hereda de la clase Map Entity.
Box Collider	Caja de colisión del objeto, necesaria para detectar el rango de captura.
Audio Source	Gestor de sonidos de la entidad en el mapa.

Tabla 11: Componentes del Prefab de punto de Puzle

Por otro lado, existe un prefab especial utilizado en el mapa que tiene una función de indicar con textos los nombres de las zonas importantes. Se trata de Prefabs modificados a partir de los que proporcionaba la librería con este fin. Son conocidos como "POIMarkers" (Marcadores de puntos de interés). La estructura de ellos se puede ver en la tabla 12:

COMPONENTE	DESCRIPCIÓN
Canvas	*Text*: Contiene un *GameObject* hijo para representar texto flotante en la escena.
	Sprite Renderer: Representa un icono flotante en la escena.
POI Label Text Setter	Script de la librería *Mapbox* (incluido inicialmente en el Prefab), que permite recoger y establecer el nombre del punto de interés, asociando las coordenadas a la información

COMPONENTE	DESCRIPCIÓN
	contenida en base de datos. Obtiene la referencia del *Canvas* para orientar en todo momento esta información hacia la cámara.
Location Info	Script empleado para asociar la información de la ubicación con el punto de puzle explicado anteriormente.

Tabla 12: Componentes del Prefab de POIMarkers

7.4.4. Posicionamiento de entidades en mapa

En este apartado se van a comentar los distintos tipos de entidades descritas anteriormente y los diferentes métodos que encontramos para gestionar sus posiciones alrededor de la escena Mapa.

7.4.4.1. Posicionamiento de objetos y otras entidades

Por un lado, la distribución alrededor del mapa de los objetos que se insertan en el inventario, no es arbitraria, sino que se rige por unas reglas establecidas y configuradas a través de la librería *Mapbox*.

Mapbox nos permite recoger puntos clave del mapa, almacenados en la base de datos de la librería en cuestión. La selección de estos puntos o zonas se realiza mediante el uso de filtros. Podemos encontrar varios tipos de filtros en el componente de configuración del mapa, como puede verse en la figura. Sin embargo, el que nos puede ser de mayor utilidad es el "poi_label", es decir, la etiqueta que almacena los nombres de los lugares, así como una gran cantidad de metadatos asociados al mismo.

Todos estos filtros pueden probarse previamente en la página de Mapbox, más concretamente en el entorno de diseño de mapas de la librería (Studio, Mapbox, 2019). Esta web permite realizar y probar mapas personalizados de la librería, modificando ajustes de filtros, así como parámetros de visualización: tema, colores, textos, etc. Más adelante tenemos la opción de importar el mapa personalizado a Unity y otros medios.

Teniendo en cuenta el funcionamiento y potencial de esta herramienta, Unity nos permite instanciar entidades a partir de un *Prefab*, en las posiciones indicadas por los filtros. Para ello, es necesario crear un "*Prefab Modifier*" de la librería *Mapbox*, encargado de traducir la información de Unity a *Mapbox*. Simplemente a este modificador, debemos indicarle la

referencia al *Prefab* que necesite, para que se realicen todas las instancias de este objeto en los puntos indicados. En la siguiente *figura 59*, se puede ver un ejemplo de distribución de objetos:

Figura 59: Captura del juego en la que se ve un ejemplo de distribución de entidades (Prefabs) en el mapa
Fuente: Elaboración propia

Por tanto, los filtros empleados con los objetos del mapa son:

- ***Buildings***: El primer filtro que se ha empleado es el de los edificios, elementos de entorno que se comentarán más adelante.
- ***Checkpoints***: En segundo lugar, se ha empleado un filtro en base a los puntos clave en el mapa, es decir, clasificaciones establecidas por la librería de lugares de mayor interés en el mundo. Estos puntos están clasificados gracias a un filtro de tipo numérico llamado "*localrank*", que puede tomar valores del 1 en adelante. En este caso se han filtrado todos aquellos con valor 1, para limitar el número de puntos en el mapa y evitar la sobrecarga. Los objetos escogidos para aparecer bajo este filtro son:

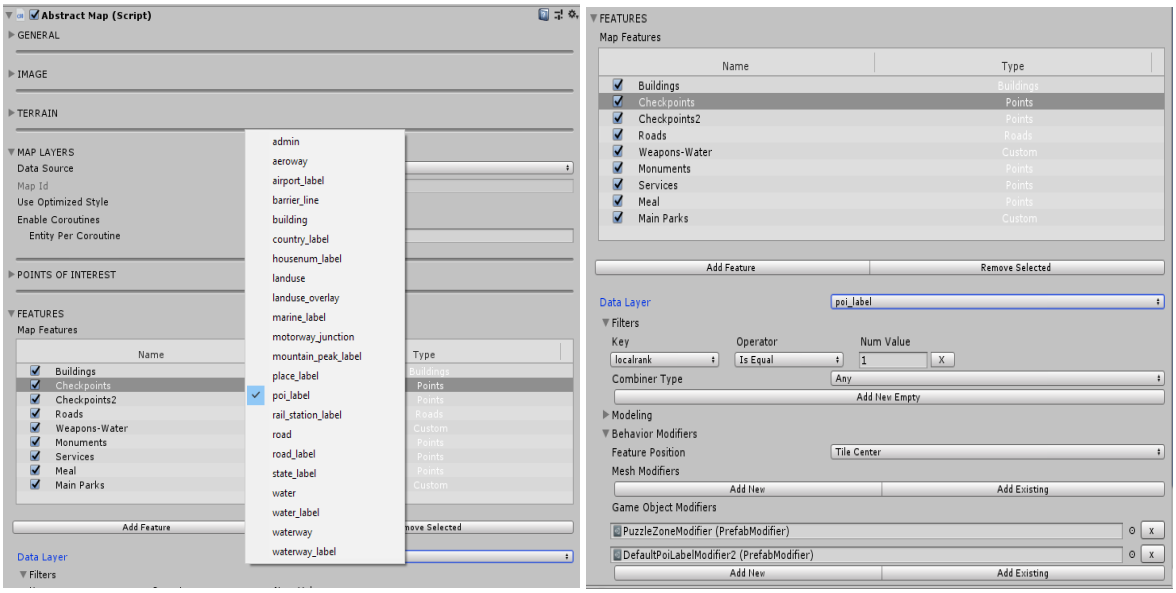
- *PuzzleZoneModifier*: Modificador que contiene la referencia al *prefab* con el punto de acceso a la escena puzle.
- *DefaultPOILabelModifier*: Modificador que contiene la referencia al *prefab* de *POIMarker*, que permite indicar en texto flotante, el nombre del punto de interés del mapa en cuestión. Esta información se asocia con el punto de puzle para su mejor gestión.

- ***Checkpoints2***: El siguiente filtro empleado es muy similar al caso anterior, pues se basa en el parámetro "*localrank*", que filtra los valores en función de su importancia. En este caso se toman valores entre 2 y 3, representando el nivel de importancia, algo menores en este caso.
  - *ExtendedCaptureModifier*: Modificador de *Mapbox* cuyo *prefab* asociado es el objeto para aumentar el rango de captura de objetos en el mapa.
  - *DefaultPOILabelModifier2*: Modificador que contiene la referencia al *prefab* de *POIMarker*, que permite indicar en texto flotante, el nombre del punto de interés del mapa en cuestión. Como en este caso no existe punto de puzle asociado, se ha añadido un icono flotante para especificar más concretamente la zona de interés en el mapa.
- ***Roads***: En este caso se ha hecho uso de un filtro predeterminado de *Mapbox* basado en carreteras. Este filtro padre, contiene a su vez otros más específicos, utilizados para limitar los resultados en el mapa. Estos son: Si se trata de una carretera principal, un túnel, un puente o un vado. De esta forma evitamos que se generen instancias de objetos en todas las carreteras del mapa, limitando el resultado.

  El objeto/modificador a instanciar en estos puntos es:
  - *DurabilityUPModifier*: Modificador cuyo *Prefab* de referencia es el objeto de reparación de armas, permitiendo así aumentar la durabilidad de las mismas.
- ***Weapon-Water***: Se trata de un filtro personalizado, que recoge todos los datos de la capa "*water*", es decir, todos aquellos sitios en los que haya agua en el mapa. Para limitar su aparición, cabe decir que los puntos de interés en estas zonas se localizan únicamente en uno de los vértices de su extensión total.

  Como se podía prever, este filtro en el mapa sirve para establecer los puntos de aparición de armas:
  - *WeaponModifier*: Es el modificador que tiene como referencia el Prefab del objeto de tipo arma, para ser recogido desde el mapa.

- ***Monuments***: Se trata de un conjunto de filtros basados en puntos de interés dentro de la capa "*poi_label*", mencionada con anterioridad. Los filtros adicionales que se aplican en este caso están basados en el parámetro "*maki*", que no es más que un parámetro de clasificación de tipo *string*, de los distintos puntos de interés. Entre ellos, encontramos: lugares religiosos, iglesias, atracciones culturales, monumentos, campings, galerías de arte, museos, ayuntamientos y castillos. Todos los puntos que estén dentro de esta categoría generarán su *prefab* asociado, que en este caso es el siguiente:
    - *Park XP Bonus*: Se trata de un modificador que contiene el *prefab* de objeto que otorga directamente al jugador experiencia adicional al pulsar sobre él. Por tanto, no se introduce dentro del inventario y ayuda a subir de nivel.
- ***Services:*** Conjunto de filtros, que como en el caso anterior, hacen uso de la capa "*poi_label*" y el parámetro "*maki*", para buscar puntos que coincidan con las distintas categorías. Estas categorías son: colegios, consultas del médico, policía, estación de bomberos, hospitales, farmacias, veterinarios, dentistas, puntos de información, gasolineras, bancos, oficinas de correos, institutos y bibliotecas. El modificador empleado en este caso es:
    - *XPMultiplier*: Modificador con referencia al *Prefab* de objeto que multiplica la experiencia ganada durante un tiempo
- ***Meal***: De nuevo es un conjunto de filtros, que como en el caso anterior, hacen uso de la capa "*poi_label*" y el parámetro "*maki*", para buscar puntos que coincidan con las distintas categorías. Las categorías filtradas son: comida rápida, restaurante, cerveza y tienda de comestibles. El modificador en este caso es:
    - *HealthItem*: Modificador con la referencia al objeto de curación, capaz de restaurar ligeramente la vida del jugador.
- ***Main Parks***: Se trata de una combinación de filtros que emplean la capa "*place_label*" y los parámetros "*localrank*" y "*maki*". En este caso, se filtra de forma que se deben cumplir todos los filtros a la vez para que se seleccione un punto. El valor de "*maki*" es "*park*" y el de "*localrank*" es entre 1 y 4. Por lo que respecta al modificador:
    - *BigHealthItem*: Modificador con la referencia del objeto de curación grande, capaz de restaurar una considerable cantidad de salud al jugador.



Figuras 60 y 61: Capturas que representan los filtros aplicados en Mapbox para instanciar los objetos
Fuentes: Elaboración propia.

7.4.4.2. Posicionamiento de personajes

Una vez comentados los puntos de posicionamiento de los objetos en el mapa, pasamos a identificar los métodos empleados para las entidades de tipo personaje.

En primer lugar, el personaje principal, siempre aparecerá centrado en la cámara, representando el centro de la escena. Por tanto, aunque el jugador se desplace utilizando los sistemas de ubicación, siempre aparecerá ubicado delante de la cámara principal, pues es necesario hacer saber al usuario en que posición se encuentra en todo momento.

Por otro lado, vamos a analizar los métodos empleados para distribuir los enemigos en el mapa. En este caso, a diferencia de los dos tipos de entidades anteriores, sí que se emplea un método para generar posiciones de forma aleatoria. Este sistema se realiza gracias a un script de comportamiento llamado "*Droid Factory*", que se integra en un *GameObject* siempre activo de la escena mapa, asegurando así que se realice una redistribución de posiciones en todo momento. Analizamos a continuación su flujo de ejecución de este script:

Cuando la partida se inicia por primera vez, la clase *Droid Factory* se encarga de instanciar hasta 5 enemigos a la vez, basándose en los distintos tipos que existen (tres en total). Para elegir los enemigos a crear en cada caso, existen una serie de probabilidades: el 50% de los

enemigos siempre es de tipo 1 (es decir, el básico). Por otro lado, entre los enemigos de tipo 2 (enemigos veloces mejorados) y 3 (jefes enemigos), se reparten las probabilidades en base al nivel del jugador. Si el nivel del jugador es múltiplo de 5, las probabilidades de que se cree un enemigo de tipo 3 serán superiores, entre un 45% y un 25%, mientras que los enemigos de tipo 2 tendrán entre un 25 y un 5 % (en base al número de enemigos que queden). Para el resto de niveles de jugador, las probabilidades se alternan con el tipo de enemigos opuesto. Este sistema de probabilidades es una forma de incentivar al jugador a subir de nivel, que se explicará más adelante en el sistema de progresión.

Una vez generados los 5 enemigos de inicio, alternando los diferentes tipos, se establecen sus posiciones de forma aleatoria dentro de unos límites. Consiste en establecer la posición de los enemigos en el mapa en los ejes X y Z, a una distancia del jugador entre 5 y 50 unidades, aleatorios en ambos casos. Para el eje Y, la posición siempre es la misma (algo superior a la del jugador), para que se quede siempre un poco por encima del terreno.

Cuando sus posiciones ya están establecidas, se mantendrán así durante al menos 10 minutos. Este tiempo es controlado por este script desde que se inicia la partida y cuando se supera, se eliminan los enemigos y se vuelven a crean por el método anterior, generando tipos y posiciones distintas. De esta forma, se favorece la variabilidad en el mapa.

Otro sistema encargado de reiniciar a los enemigos es aquel basado en las distancias, es decir, cuando el jugador se encuentra a una distancia superior a 80 unidades, los enemigos se volverán a crear para adecuarse a la nueva posición del jugador. De esta forma, se favorece que siempre se puedan realizar combates, independientemente de la posición del jugador en el mapa.

Si el usuario cierra la aplicación y la vuelve abrir, la cantidad y posiciones de enemigos se mantendrá durante al menos 10 minutos, a no ser que el jugador se aleje mucho del punto de inicio, en ese caso se reiniciará antes.

7.5. Aplicación de inteligencia artificial

Durante este apartado, se va a explicar la inteligencia artificial desarrollada en el juego mediante el uso de *Behaviour Trees*. Para ello, se ha empleado un asset gratuito de terceros, disponible en la tienda de Unity y que es conocido como *Panda BT Free* (Panda BT, Eric Begue, 2019).

Panda BT es un *framework* o *plugin* para el desarrollo de *Behaviour Trees*, que permite definir de forma sencilla, la lógica compleja, escalable y reutilizable de estos comportamientos para cualquier juego. El proceso por el que se integra en la aplicación, una vez instalado, es mediante un componente adicional de Unity que se asocia a cualquier *GameObject* y que, para modelar este comportamiento, combina scripts de C# y BT. Este último es un lenguaje de programación minimalista que utiliza la librería para determinar el flujo de ejecución dentro del árbol en cuestión. Es decir, en lugar de utilizar elementos gráficos que representen los nodos del árbol a desarrollar, se emplea este lenguaje, que modulariza de la misma forma las distintas partes del árbol.

Este *framework* dispone además de una página web con toda la documentación necesaria para su uso, que facilita mucho el trabajo para los principiantes (Panda BT Documentation, Eric Begue, 2019).

Revisando la documentación, podemos concluir a grandes rasgos, que el lenguaje de programación BT es un lenguaje secuencial, basado en saltos de línea y tabulados que hay que seguir estrictamente si queremos evitar problemas de compilación. Por otro lado, es importante conocer los 2 tipos de nodos que existen y como se utilizan en la librería:

- ***Task Nodes***: Se trata de aquellos nodos hoja o finales del árbol, que se corresponden con las acciones de bajo nivel programadas en C# (scripts de comportamiento genéricos en Unity). Dentro de estos nodos, se puede retornar un valor al script de BT, indicando si se ha realizado correctamente (*Succeed*) o no (*Fail*), devolviendo el control
- ***Structural Nodes***: Son los elementos que nos proporciona la librería, utilizados dentro del script de BT y que nos permite controlar el flujo de ejecución del árbol con condiciones. En total encontramos hasta 10 nodos de este tipo: *Tree, Sequence, Fallback, parallel, race, random, repeat, while, not* y *mute*. De estos 10, se explican a continuación los 5 empleados en el proyecto:

- *Tree*: Nodo identificador que da nombre a una jerarquía de nodos.
- *Sequence*: Ejecuta los nodos de uno en uno hasta que alguno de ellos falle. En ese caso, se reinicia la ejecución del árbol, con el nodo padre.
- *Fallback*: Ejecuta los nodos de uno en uno hasta que alguno de ellos se realice correctamente (por tanto, todos los anteriores deben haber fallado). En ese caso, se reinicia la ejecución del árbol.
- *Random*: Ejecuta de forma aleatoria uno de todos los nodos que le suceden.
- *While*: Realiza una acción en bucle (ejecución de uno o varios nodos), mientras se cumpla una condición (indicada justo debajo).

Además, en esta web se explica que, con el componente de Unity, se puede consultar el estado de los nodos del árbol en tiempo de ejecución, comprobando/depurando de una forma visual el flujo de ejecución de todos los nodos.

Retomando el trabajo realizado en el proyecto, cabe decir que la inteligencia artificial se ha desarrollado únicamente para los enemigos de la escena batalla, por lo que tenemos un total de 3 árboles de decisiones, que se corresponden con estos 3 tipos de enemigos en cuestión. A continuación, analizaremos uno por uno los árboles desarrollados:

- **Enemigo básico**: Es aquel que establece el comportamiento base del resto de enemigos, ya que en muchos casos reutilizan funciones de este. Cuando se entre en combate con ellos, siempre aparecerán 3 a la vez. Los 2 ficheros que determinan el comportamiento de este enemigo son: *Enemy Tree* escrito en lenguaje de *Panda BT* y *Enemy Behaviour* en C#. Para empezar a explicar su funcionamiento, vamos a tomar como referencia la figura 62 que representa el fichero de *Enemy Tree*.

Figura 62: Captura del script de Behaviour Tree del enemigo básico
Fuente: Elaboración propia.

Como se puede ver, el *behaviour tree* comienza desde el nodo "*Root*", haciendo uso de un nodo estructural: "*fallback*". De esta forma, el nodo se encarga de determinar si los nodos que le suceden se ejecutan correctamente o no. Cuando "*checkAttack*", "*movement*" o "*attack*" devuelven un resultado correcto, el árbol vuelve a empezar su ejecución de nuevo, deteniendo así la sucesión de nodos.

Vamos a analizar los 3 nodos principales de "*Root*" para este enemigo.

- El primero de ellos es "*checkAttack*". Una vez en él, encontramos un nodo "*sequence*", que permite seguir ejecutando todos los nodos siempre y cuando no fallen. Lo primero que se hace es llamar a la función *CheckAttacked*, que se encuentra en el script de C# (*Enemy Behaviour*). Esta función comprueba si el enemigo ha sido atacado recientemente, devolviendo "*success*" en tal caso. De ser así, la ejecución continúa llamado al nodo "*attackedDecision*". Dentro de éste, lo único que se hace es realizar una selección aleatoria

(*random*) entre la función de ataque y defensa, con las mismas probabilidades. Si se realiza correctamente devuelve *success*, deteniendo la ejecución en "*Root*" (se reinicia).

- Si el caso anterior falla en algún punto, se devuelve "*fail*", pasando a ejecutar el siguiente nodo, en este caso "*movement*". Si accedemos a este nodo, lo primero que se hace es ejecutar un bucle con la condición *CheckMovementActive*. En esta función/condición se comprueba si el tiempo de patrulla sigue activo. De ser así, el enemigo sigue las rutinas de moverse y comprobar los límites del mapa.
- Sin embargo, si el tiempo de patrulla finaliza y la función devuelve *fail*, pasamos a ejecutar el siguiente nodo en "*Root*" que es "*attack*". Una vez dentro, pasamos a ejecutar directamente una secuencia, llamando a la función de ataque. Dentro de esta función se decide si el ataque es a distancia o cuerpo a cuerpo. Una vez realizado el ataque, se accede al nodo "*idle*" en el que se llama a *InitMove*. En esta función se reinician los valores de tiempo para el movimiento y se realiza un giro, modificando la dirección del enemigo.

- **Enemigo avanzado**: Enemigo que además de tener el comportamiento del enemigo base, incluye algunas funciones con mejoras en su comportamiento. Además, sus parámetros de velocidad de movimiento, ataque y vida se encuentran potenciados respecto al enemigo anterior. El enfrentamiento con ellos se realiza con 2 de este tipo. Analizamos su comportamiento en base a la figura 63 a continuación:

Figura 63: Captura del script de Behaviour Tree del enemigo avanzado
Fuente: Elaboración propia.

Como en el caso anterior, el *behaviour tree* comienza desde el nodo "*Root*", haciendo uso de un nodo estructural: "*fallback*". En este caso se mantienen lo nodos "*checkAttack*", "*movement*" y "*attack*", sin embargo, aparece un nuevo nodo entre los dos primeros, llamado "*checkBullet*". Este nodo se ejecuta si la comprobación de enemigo atacado devuelve "*fail*". En ese caso, lo primero que se hace es entrar en una secuencia en la que se llama a *CheckProtect*. Esta función se encarga de comprobar si hay una bala de jugador cerca de él. De ser así, pasa a ejecutar el método *Protect*, en el que si el temporizador de tiempo lo permite, el enemigo activará el escudo, defendiéndose antes de ser atacado.

Otro cambio respecto al enemigo anterior lo encontramos en el último nodo de "*Root*", es decir, el de ataque. En este caso, en lugar de llamar a la función de ataque

directamente, lo que se hace es llamar a un nodo en el que se selecciona de forma aleatoria, si se va a atacar o usar una habilidad. Esta habilidad, ha sido establecida al inicializar el enemigo en batalla y solo puede ser una de las siguientes:

- Mala visibilidad: Se aplica durante un tiempo un filtro en la cámara del jugador (modificando los colores y la opacidad en el UI) y dificulta la visibilidad.
- Invisibilidad: Se aplica una transformación en la textura del enemigo en cuestión, reduciendo su opacidad para ser poco visible.
- Ataque fuerte: El enemigo ataca creando una bala con el doble de daño de lo normal.
- Curación: El enemigo es capaz de curarse ligeramente, restableciendo su salud en combate.

Estas habilidades tienen un tiempo de recarga, por lo que en muchos casos no pueden ser usados. Para el resto de los nodos, no se ha alterado el comportamiento del enemigo, por lo que actuará de forma muy similar al enemigo básico.

- **Enemigo jefe**: Se trata del enemigo definitivo, que tiene rutinas muy similares a los enemigos anteriores, pero con un comportamiento diferente. Dispone además de las características de ataque, vida y velocidad más altas de todos. Pasamos a analizar el comportamiento en base a la figura 64.

  Como se puede ver, la estructura de los nodos es muy similar al caso anterior de enemigo avanzado. Sin embargo, a pesar de que la mayoría de nodos se mantienen, podemos identificar los siguientes cambios clave:
  - Rutinas de movimiento: Las funciones a las que se llama para mover al enemigo jefe, son distintas, llamando a move2 y checkLimit2. Esto se debe a que el movimiento del enemigo no es de patrulla, orientándose en varias direcciones, sino que se mueve con una trayectoria circular, alrededor del jugador y siempre orientado hacia la cámara. Por tanto, las comprobaciones de límite son distintas en este caso. En el movimiento, también se varia cada cierto tiempo modificando la altura a la que va la nave y el sentido de giro.
  - Ataque: A la hora de atacar, siempre se realizan ataques a distancia, por lo que se elimina la posibilidad de cuerpo a cuerpo.

- Habilidad: Al igual que ocurría en el caso anterior, se realiza un *random*, para comprobar si el enemigo debía atacar o usar una habilidad. En este caso, el enemigo puede usar cualquier habilidad de las que existen, sin limitar esta opción a una. Se realiza un *random* antes de ser usada para determinar la habilidad en cada caso. Puede verse este cambio en el nodo "*ability*" de la figura 64.

Figura 64: Captura del script de Behaviour Tree del enemigo jefe
Fuente: Elaboración propia.

Finalmente, una vez analizados los distintos enemigos y su inteligencia artificial, se puede concluir que los enemigos han sido desarrollados en base a los *behaviour trees* planificados en el GDD. A pesar de algunas mejoras con nodos adicionales, se corresponde con los nodos

principales de movimiento, ataque y defensa que se habían previsto para los 3 tipos de enemigos.

7.6. Desarrollo de sistema de progresión y objetivos

Durante este apartado se va a comentar como se han implementado los distintos aspectos que afectan al sistema de progresión del juego. También se va a contrastar la información respecto a lo previsto en el GDD, para explicar así algunas diferencias.

En primer lugar, el objetivo más importante desarrollado en la progresión del juego es el sistema de niveles. El jugador protagonista puede subir de nivel a lo largo de la partida para avanzar, enfrentándose cada vez a enemigos más difíciles y teniendo la posibilidad de obtener armas más fuertes. Para subir de nivel, es necesario obtener experiencia, la cual puede conseguirse en las distintas escenas con los siguientes métodos:

- **Escena mapa**: Objetos con bonus de experiencia. Cuando el jugador recoge estos objetos en el mapa, obtiene una cantidad de experiencia adicional para subir de nivel. Por defecto, es un método poco eficaz de obtener experiencia, pues en general no existen muchos puntos para ello en el mapa y no suelen dar una gran cantidad de experiencia; alrededor de unos 10 px.
- **Escena puzle**: Cuando accedemos a la escena puzle y completamos el objetivo del laberinto con la esfera, nos aparece una ventana emergente informándonos sobre los puntos de experiencia obtenidos al ganar. En este caso, estos puntos son muy superiores a los que se consiguen con los objetos bonus. En función de cómo completemos el objetivo, podemos distinguir:
    - Sin realidad aumentada: entre 20 y 50 px.
    - Con realidad aumentada: entre 40 y 100 px.
- **Escena batalla**: En esta escena, obtendremos la mayor cantidad puntos de experiencia, siempre y cuando derrotemos a todos los enemigos que aparezcan durante el combate. De ser así, nos aparecerá nuevamente una ventana con los puntos de experiencia obtenidos. En este caso, las cantidades varían en función del tipo de enemigo:
    - Enemigo básico: Cada enemigo básico derrotado supone una experiencia entre 20 y 60 puntos. Sin embargo, en los combates de este tipo, los enemigos

aparecen de 3 en 3, por lo que la cantidad de experiencia total oscila entre 60 y 180 px.

- Enemigo avanzado: Por cada enemigo avanzado derrotado se obtiene una experiencia entre 50 y 100 puntos, pero como siempre aparecen de 2 en 2, la experiencia total se encuentra entre 100 y 200 px.
- Enemigo jefe: Este tipo de enemigo, nos puede dar una experiencia total entre 150 y 300 px. Aparece 1 solo enemigo de este tipo por combate.

Toda esta cantidad de experiencia puede llegar a multiplicarse por dos, en caso de tener el objeto multiplicador de experiencia activo.

Las funciones encargadas de añadir experiencia y gestionar la subida de nivel se encuentran en la propia clase del jugador. La experiencia máxima requerida por nivel se ha desarrollado de una forma sencilla: multiplicando el nivel del jugador por 10. Por tanto, cuando la barra de experiencia alcanza este máximo mediante lo métodos comentados, el jugador subirá al siguiente nivel. Si los puntos adquiridos, superan el máximo de la barra de nivel, no se pierden, sino que se tienen en cuenta para el siguiente nivel (es decir, no se pone a 0 puntos por defecto con el nuevo nivel).

Por otro lado, vamos a comentar el segundo objetivo más importante en el juego, para el sistema de progresión creado. En este caso se trata de las armas del inventario, que se encuentran muy relacionadas con la subida de nivel. Las armas son generadas de forma aleatoria, con varios parámetros, que las hacen difícilmente replicables entre sí. Esto aporta una gran variabilidad al juego e incita al jugador a comparar y probar varias armas en el combate. En la *figura 65*, se puede ver la progresión del perfil del jugador, con la experiencia, la vida, el arma equipada y la probabilidad de enemigo jefe que se comenta más adelante:

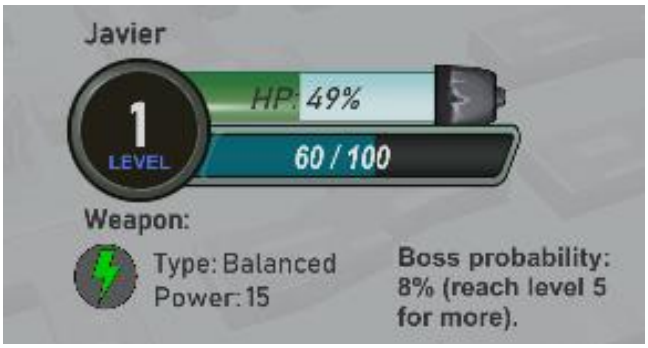


Figura 65: Captura de perfil de jugador, con la información de progresión
Fuente: Elaboración propia.

Las armas, por lo general, tendrán mejores atributos cuanto mayor sea el nivel del jugador, pues en su algoritmo de creación es un valor que se tiene en cuenta. Este incremento de poder basado en el nivel es necesario, pues los enemigos también suben sus estadísticas de salud y ataque con este factor. Por tanto, se requiere obtener armas nuevas potenciadas, que actualicen las que ya tenemos. Las formas de obtener estos equipamientos son:

- Subida de nivel: Desde cualquier escena, cuando el jugador sube de nivel, aparece una ventana emergente, notificando el nuevo nivel y el arma aleatoria obtenida. Es el método más común para obtener estas armas y un incentivo más para seguir subiendo de nivel.
- Recoger armas en el mapa: Como hemos comentado en apartados anteriores, en algunas zonas de agua en el mapa, aparecerán objetos de tipo arma para ser recogidos por el jugador. La aparición de estos elementos por lo general es escasa, pues se encuentra limitado a 1 vértice de toda la zona de agua en cuestión.
- Puzle con realidad aumentada: La última forma de obtener armas es mediante la realización de puzles con la realidad aumentada activa. En este caso, si cumplimos el objetivo antes de que se acabe el tiempo, existe una probabilidad de un 20% de recibir un arma para el inventario. Esta es otra forma de incentivar el uso de la realidad aumentada basada en marcadores que, además, se explica en el apartado de cómo jugar (dentro del juego).

Para terminar, el último objetivo creado para mejorar el sistema de progresión es la obtención de medallas cuando se resuelve un puzle. Se trata de un objetivo independiente del sistema de niveles que permite obtener un reconocimiento cuando se supera el puzle de algún lugar en el mundo. Como las zonas de puzle están asociadas a lugares con una mayor relevancia o interés social, cuando el jugador supera el puzle, obtiene una nueva medalla con el nombre del punto del mapa en el que se resolvió el puzle. De esta forma, el jugador tiene una forma de demostrar que ha estado en algún sitio del mundo y ha superado la prueba que le proponía el juego en esta zona. Se trata por tanto de un incentivo más para que el jugador se mueva alrededor del mundo.

Por lo que respecta al sistema de progresión de enemigos propuesto en el GDD, se han modificado algunas ideas durante el desarrollo. Aunque el objetivo sigue siendo eliminar al jefe final, este no se limita a una sola batalla, sino que se puede combatir con él varias veces a lo largo del juego. De esta forma, se reutiliza este tipo de enemigo para más de una ocasión,

generando una mayor variabilidad en los combates. Sin embargo, la aparición de este rival se encuentra limitada por tratarse de un enemigo especial. Su aparición se gestiona mediante probabilidades, basadas en el nivel del jugador. Cuando el jugador alcanza un nivel múltiplo de 5, la probabilidad de que aparezca este enemigo aumenta, como ya comentamos en el punto de 5.4.4. De esta forma no se limita este combate para el nivel 30, como se proponía en el GDD.

La obtención de objetos de un solo uso se limita al mapa, a diferencia de como se planteaba en el GDD (puzles, combates, etc,). De esta forma, se le da una mayor relevancia a la escena mapa, incitando al jugador a desplazarse para alcanzar ciertos objetos.

Por lo que respecta a la duración del juego, sigue siendo infinita, pero cada vez la subida de nivel es un mayor reto por requerir una mayor cantidad de puntos de experiencia.

Para los demás aspectos del sistema de progresión comentados, sí que se ha implementado de la forma en la que se planteaba originalmente en el GDD.

7.7. Persistencia y guardado de partida

Un apartado importante al que no se le ha dado la suficiente importancia durante la planificación del proyecto es la persistencia de archivos del juego. Se trata de un factor muy relevante para la esencia del juego, pues requiere guardar mucha información entre partidas y/o escenas. De no ser así, el jugador perdería todo su progreso y muchas funcionalidades no tendrían sentido: niveles, vida, inventario, etc.

Para guardar toda esta información se ha optado por el almacenamiento de archivos en local, basado en la creación de ficheros binarios que se encargan de guardar las estructuras de datos y la información que contienen. Este proceso se realiza desde el script "*Game Manager*", objeto común entre escenas y que controla la inicialización de objetos clave en el juego, así como el proceso de guardado.

En "*Game Manager*" encontramos 2 funciones principales: "*save*" (guardar) y "*load*" (cargar). Para que estos métodos realicen su función correctamente, precisan del manejador de archivos binarios y unas estructuras de datos llamadas "clases *serializables*". Estas clases contienen las variables preparadas para guardar la información en archivos binarios. Los tipos de estas variables deben ser primitivos pues son los únicos preparados para ser almacenados en este tipo de ficheros. Para este proceso en el juego, se han creado 4 clases

serializables, clasificadas en base los bloques de información que contienen. Entre ellas se identifican:

- ***PlayerData***: Almacena la información relativa al jugador. Sus variables son:

TIPO DE DATO	VARIABLE	DESCRIPCIÓN
Integer	xp	Almacena la experiencia del jugador.
Integer	requiredXp	Experiencia requerida en el nivel actual del jugador.
Integer	levelBase	Nivel de inicio.
Integer	lvl	Nivel actual.
Integer	total_xp	Experiencia total adquirida por el jugador a lo largo del juego (partida guardada).
Integer	hp	Cantidad de vida del jugador.
Integer	maxHP	Máxima cantidad de vida del jugador.
Float	captureRange	Factor de escala del rango de captura.
Integer	xp_multiplier	Factor multiplicador de experiencia para el jugador.
String	user_name	Nombre de usuario del jugador.
Float	volume	Volumen del juego.
Float[]	pos	Array o listado numérico que guarda la última posición del jugador en el mapa. Se emplea para reiniciar la distribución de los enemigos cuando el jugador se aleja mucho de esta posición.
Float[]	droidPosX	Array que contiene las posiciones de todos los enemigos en el eje X.
Float[]	droidPosY	Array que contiene las posiciones de todos los enemigos en el eje Y.
Float[]	droidPosZ	Array que contiene las posiciones de todos los enemigos en el eje Z.

TIPO DE DATO	VARIABLE	DESCRIPCIÓN
Int[]	droidType	Array que contiene los tipos de enemigos en el mapa (asociados a las posiciones).
Int	droidSelectedType	Guarda el tipo de enemigo seleccionado por el jugador, para instanciarlo en la batalla.
Int	droidSize	Cantidad de enemigos restantes.
Int	droidCombatID	Id de enemigo seleccionado para el combate (a eliminar en caso de ganar).
Bool	combatWin	Guarda la información de victoria en la última batalla.
String	lastGameDate	Fecha y hora de la última partida guardada.

Tabla 13: Variables que guardan la información del jugador

- ***InventoryData***: Guarda la información de inventario asociada al jugador:

TIPO DE DATO	VARIABLE	DESCRIPCIÓN
Integer[]	itemIDs	Array numérico que contiene las ids asociadas a todos los ítems del inventario.
Integer[]	itemRand	Array numérico que contiene el valor generado aleatoriamente por el ítem en la inicialización.
Bool[]	itemActive	Array que contiene toda la información de si los ítems están o no activos.
Integer[]	itemType	Array que contiene los tipos asociados a los ítems.
Float[]	itemPosX	Array de posiciones de los ítems en el eje X, donde fueron recogidos en el mapa.
Float[]	itemPosY	Array de posiciones de los ítems en el eje Y, donde fueron recogidos en el mapa.
Float[]	itemPosZ	Array de posiciones de los ítems en el eje Z, donde fueron recogidos en el mapa.

TIPO DE DATO	VARIABLE	DESCRIPCIÓN
Integer	itemSize	Tamaño de ítems almacenados en el inventario.
Integer[]	equipIDs	Array numérico que contiene las ids asociadas a todas las armas del inventario.
Integer[]	equipQuality	Array que contiene la calidad de cada una de las armas del jugador.
Integer[]	equipType	Array con los tipos asociados a las armas.
Integer[]	equipAttack	Array con el valor de ataque de cada una de las armas.
Integer[]	equipDefense	Array con el valor de defensa de cada una de las armas.
Float[]	equipSpeed	Array con el valor de velocidad de ataque de cada una de las armas.
Float[]	equipDurability	Array con el valor de durabilidad (vida) de cada una de las armas.
Bool[]	equipActive	Array que contiene los tipos de enemigos en el mapa (asociados a las posiciones).
Float[]	equipPosX	Array que contiene las posiciones del eje X en las que fueron recogidas las armas del inventario.
Float[]	equipPosY	Array que contiene las posiciones del eje Y en las que fueron recogidas las armas del inventario.
Float[]	equipPosZ	Array que contiene las posiciones del eje Z en las que fueron recogidas las armas del inventario.
Integer	equipmentSize	Tamaño de armas almacenadas en el inventario.
Integer	id_e_selected	Id de arma equipada
Integer	space	Tamaño total de slots en el inventario.

Tabla 14: Variables que guardan la información del inventario

- ***ItemsManagerData***: Se encarga de gestionar las Ids de los objetos y controlar los ítems activos tras su uso.

TIPO DE DATO	VARIABLE	DESCRIPCIÓN
Integer	lastIDItem	Valor que representa el último Id usado en la creación de ítems.
Integer	lastIDEquip	Valor que representa el último id usado en la creación de armas.
Integer	maxSizeActive	Valor máximo de ítems activos a la vez.
Integer[]	itemIDs	Array que contiene los ids de los ítems.
Integer[]	itemRand	Array que guarda los valores aleatorios generados por los ítems en su inicialización.
Bool[]	itemActive	Array que contiene la activación asociada a cada ítem.
Integer[]	itemType	Array que contiene los tipos asociados a los ítems.
Float[]	itemTargetTime	Array que contiene el tiempo restante (en segundos) de uso de un ítem.
Float[]	itemPosX	Array de posiciones de los ítems en el eje X, donde fueron recogidos en el mapa.
Float[]	itemPosY	Array de posiciones de los ítems en el eje Y, donde fueron recogidos en el mapa.
Float[]	itemPosZ	Array de posiciones de los ítems en el eje Z, donde fueron recogidos en el mapa.
Integer	itemSize	Tamaño total del array de ítems en uso.

Tabla 15: Variables que guardan la información del gestor de ítems

- *PuzleData*: Guarda la información relacionada con los puntos de puzle y las medallas asociadas obtenidas.

TIPO DE DATO	VARIABLE	DESCRIPCIÓN
String	select_location	Guarda el nombre de la ubicación asociada al puzle.
String[]	disabled_locations	Array que almacena los nombres de los puzles que se encuentran temporalmente inaccesibles.
String[]	time	Array de strings que almacena la fecha y la hora en la que se accedió por última vez al puzle (deshabilitados).
Bool	win	Comprobador de victoria para la escena puzle.
String[]	badges	Array que guarda los nombres de las medallas obtenidas, correspondientes al lugar del puzle.
Integer	badges_size	Número de medallas obtenidas.

Tabla 16: Variables que guardan la información relativa los puzles

El método empleado para guardar la partida se realiza siempre en las transiciones entre escenas y cuando se cierra la aplicación. De esta forma, nos aseguramos de mantener en todo momento los ficheros actualizados, con los datos clave de la partida. Si no existían previamente, se crean los cuatro archivos comentados. Pero, si ya existían se sobrescribirán.

Por otra parte, el proceso de carga se realiza, siempre y cuando se encuentren los ficheros binarios en cuestión. En tal caso, cada vez que se inicializa una escena, se llama a la función de carga que recoge los datos anteriormente comentados y los vuelca sobre las clases correspondientes activas.

Cabe decir que, en algunos casos, se omite el guardado y la carga de cierta información, por depender de entidades en otras escenas.

7.8. Gestión de escenarios y entorno

En este punto, se van a comentar los distintos recursos y elementos empleados para gestionar los escenarios y aplicar mejoras a nivel visual. La librería Mapbox es una de las principales responsables en esta sección ya que su gran rango de personalización ha sido esencial para que el resultado final fuera más acabado. También identificamos otros elementos comentados anteriormente, basados en *assets* que se distribuyen alrededor del mapa. Pasamos a identificarlos a continuación:

La personalización de mapas en *Mapbox*, se ha realizado como adelantábamos en el punto 6.9, es decir, desde el editor online que nos proporciona la librería en su página oficial. Para ello, además de consultar la documentación (Documentation, Mapbox, 2019), es necesario registrarse previamente para así guardar los mapas personalizados con anterioridad. Una vez identificado, nos debemos dirigir a la sección de "*Studio*", donde podemos crear nuestros mapas a partir de unas plantillas. En este caso se ha utilizado la plantilla "*Streets*" que nos proporciona la librería. Podemos ver una captura de esta plantilla en la *figura 66*.

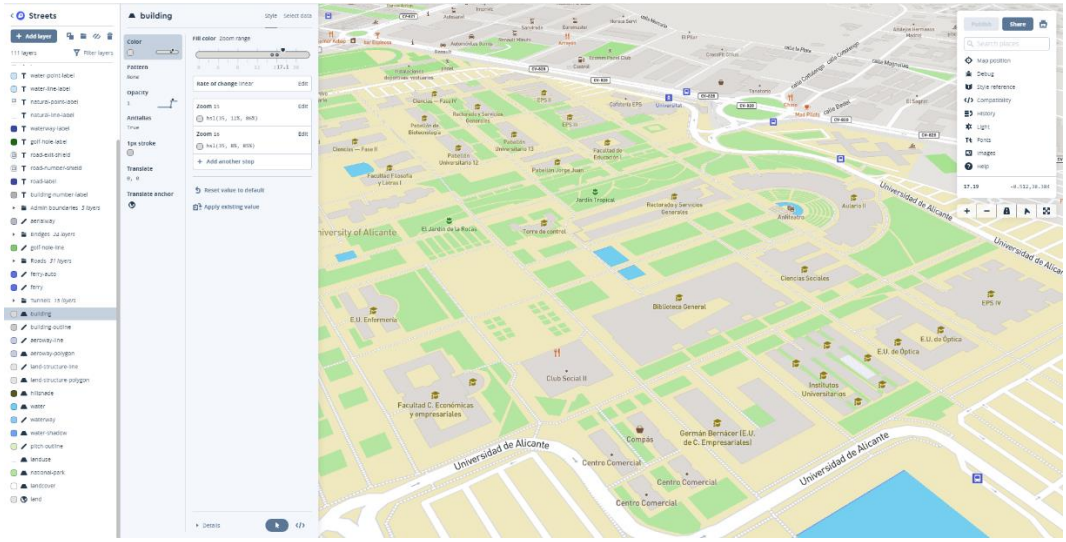


Figura 66: Captura de la plantilla "Streets" en la UA, con Studio de Mapbox
Fuente: Elaboración propia

Esta plantilla contiene por defecto una gran cantidad de *layers* (capas), que podemos consultar en la barra de navegación a la izquierda, proporcionada por el editor web. Sin embargo, para el tipo de juego a desarrollar, no interesa tener todas estas capas en activas en el mapa, ya que pueden afectar al rendimiento de este, así como proporcionar más información de la necesaria. Por ello, se han realizado algunas modificaciones de la plantilla,

eliminando gran parte de los elementos/capas que podíamos encontrar originalmente. El resultado puede verse en la captura 67 a continuación:

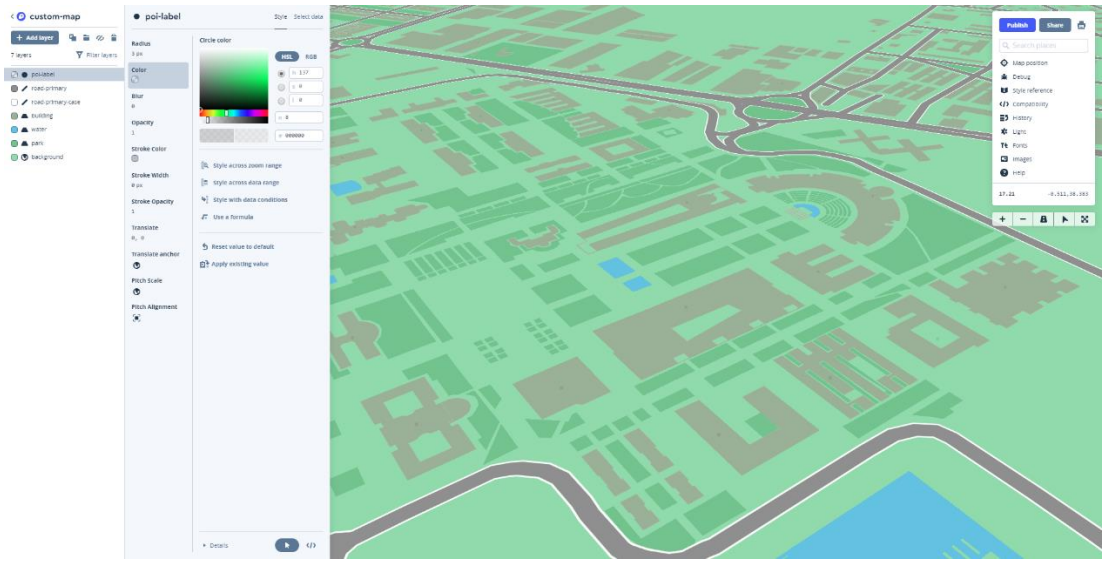


Figura 67: Captura de la plantilla "Streets" modificada en la UA, con Studio de Mapbox – Usada en el juego
Fuente: Elaboración propia

A partir de la figura anterior, podemos ver como se han modificado los colores del entorno, ofreciendo una estética más alegre para el videojuego. Por otro lado, los paneles de información que veíamos anteriormente han sido eliminados junto a sus capas, por contener información excesiva sobre el mapa y de poca utilidad en el juego. Además, estos carteles, una vez se integran en Unity, son estáticos, es decir, letras que forman parte de la imagen que cubre el mapa.

Por tanto, viendo el resultado del mapa personalizado, podemos analizar las distintas capas que lo forman. Todas ellas pueden ajustar su anchura o rango visible en el mapa y adecuarlo a los distintos niveles de zoom aplicables. Además, muchas de estas capas permiten aplicar filtros para limitar los elementos de ese tipo. Entre estas capas tenemos:

- ***Poi-label***: Como ya se ha comentado con anterioridad, se trata de las etiquetas de los puntos de interés. En el mapa tienen una relevancia visual escasa por sí mismos, ya que se identifican por unos puntos grises con poca opacidad. Sin embargo, disponen de una serie de metadatos asociados entre los que se puede distinguir: valor de importancia, nombre, tipo, etc. Cuando se integran en Unity destacamos 2 mejoras a partir de estos puntos:

    - Los edificios de puzle, que mejoran la escena añadiendo construcciones adicionales.
    - Los textos flotantes asociados a los puntos de interés. En algunos de estos puntos, se asocia la información con el puzle en cuestión.
- ***Road-primary:*** Capa que se encarga de representar todos los caminos y las carreteras, almacenadas en la base de datos de la librería. Aplica filtros para representar únicamente las carreteras de mayor relevancia (primarias, secundarias y terciarias). Se representan con un color gris oscuro.
- ***Road-primary-case***: Capa auxiliar de la carretera que sirve para representar los bordes de esta. Para este caso se han representado de color blanco.
- ***Building***: Zonas de edificios representadas de color gris claro. Se recogen la mayoría de ellos, aunque en algunos casos, la librería toma algunas zonas de edificios como explanadas sin construcción, por lo que no aparecen representadas. Cuando se importa el mapa en Unity podemos destacar la siguiente mejora:
    - Se generan unos polígonos con la forma de la zona del edificio, representando con profundidad estas construcciones. Además, para aportarle un mayor realismo, el componente de la librería ofrece la opción de aplicar una textura a estas construcciones basada en las imágenes de satélite, por lo que se ajustan bastante a la realidad.
- ***Water***: Zonas de agua representadas de color azul.
- ***Park***: Capa que representa las zonas de parque/vegetación abundante, mediante un color verde oscuro.
- ***Background***: Capa que representa la textura de fondo, aplicada al resto de superficies que no se ajusten a los filtros de las capas anteriores. Tiene un color verde más claro que en las zonas de parque.

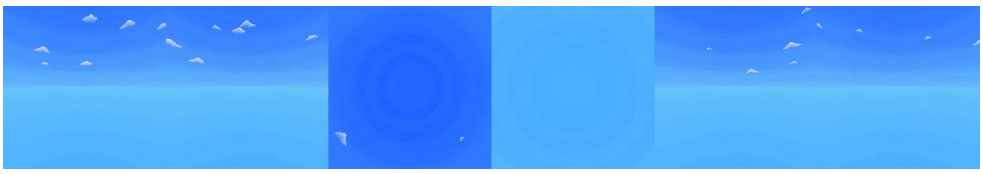
Una vez finalizada la personalización pasamos a integrar el mapa en Unity. Para ello es necesario establecer el token o licencia de uso asociada al perfil de *Mapbox*, así como la URL del mapa creado, que se establece en el componente del mapa en escena. Tras realizar los ajustes de filtros y objetos, si tomamos una captura de forma panorámica, el resultado es el que se puede ver en la siguiente *figura 68*:

Figura 68: Captura panorámica de la escena de Unity en la UA
Fuente: Elaboración propia

Como podemos ver, además de los puntos de puzle y los indicadores de puntos de interés, existen varios objetos distribuidos alrededor del mapa que se colocan según los filtros aplicados y modifican el entorno de la escena. Estos elementos ya han sido explicados en el punto 7.4.4.

Sin embargo, a partir de las imágenes, podemos detectar un cambio de color visible en las texturas aplicados en el *studio* de *Mapbox*. Estos cambios se deben a dos factores:

- Iluminación: Se trata del elemento que permite hacer visibles los objetos en escena. La iluminación es de tipo direccional, orientada en perpendicular hacia el mapa. El color de la fuente de luz es amarillo claro (*#FFF4D6*), que al proyectarse sobre la superficie del mapa, modifica ligeramente los colores originales.
- *Skybox*: Se trata de uno de los elementos principales en cuanto a ambientación se refiere. Consiste en una imagen empleada como mapa de texturas, que se aplica al cubo que envuelve a la totalidad de la escena. Este cubo, con la textura aplicada, genera una proyección de entidades al fondo de la escena, representando imágenes a lo lejos. Estas imágenes generan el efecto de cielo y el horizonte que puede verse en las siguientes *figuras 69 y 70*:



Figuras 69 y 70: Capturas del Skybox en el juego y mapa de textura empleado
Fuente: Elaboración propia (69) y http://bit.ly/30APAeO (70)

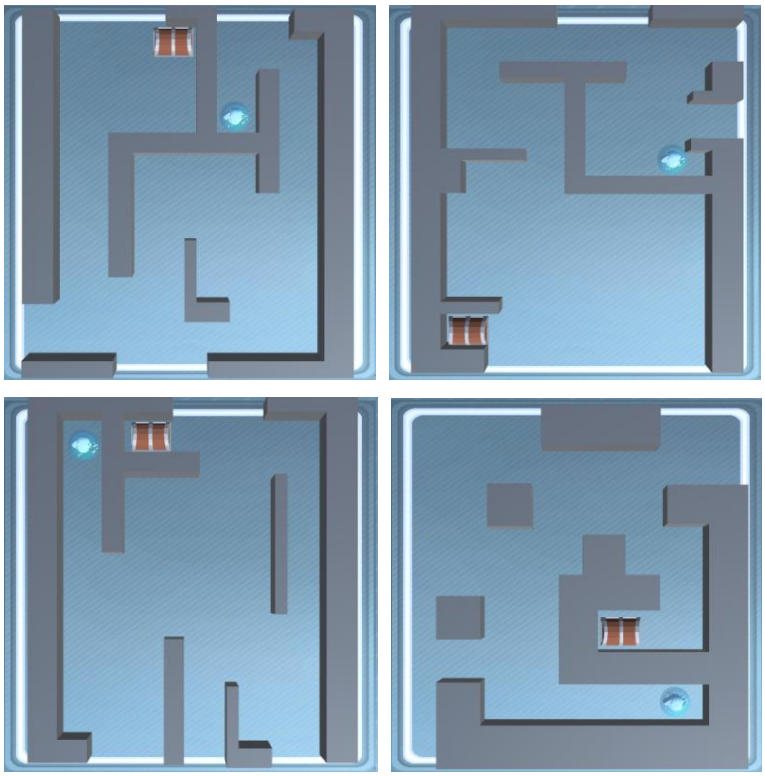
Como se puede ver, se trata de una textura con unos colores muy azules, que tienen una gran influencia cuando se reflejan en la escena. Esto justifica el cambio de color en la textura del mapa.

Esta textura se ha obtenido a partir de un *asset* gratuito en la tienda de Unity. Además de permitir su integración en el juego, dispone de un *shader* (script de comportamiento especial), que permite modificar distintos parámetros del skybox, entre ellos: color, efecto de niebla y rotación de la textura activa. La rotación permite que durante la partida el *skybox* gire, generando una sensación de dinamismo en la escena. Este giro únicamente se aprecia por las nubes representadas en la textura, pues el resto de los elementos se mantiene homogéneo.

Por lo que respecta a las escenas que no son el mapa, la ambientación no es un aspecto muy relevante, pues se utilizan técnicas de realidad aumentada basadas en la cámara. Este es el caso para la escena de batalla, en la que los únicos aspectos relevantes se encuentran en la interfaz.

Sin embargo, podemos destacar algunos puntos de la escena puzle que afectan a la ambientación, como es el diseño de los laberintos. Se han realizado un total de 4 laberintos, que se alternan de forma aleatoria entre ellos cuando se inicia la escena.

En todos los casos, se han realizado en *Blender* para ser importados posteriormente a Unity. Podemos verlos a continuación (de la *figura 71* a la *74*):



Figuras 71, 72, 73 y 74: Capturas de los puzles diseñados en el juego.
Fuente: Elaboración propia

En las figuras también destaca el uso de una textura azul aplicada en el plano base, que contrasta con los muros del laberinto. Los otros dos elementos son la esfera que controla el jugador y el cofre que representa el final del laberinto, a alcanzar.

7.9. Diseño de interfaces

En este apartado se va a explicar del diseño de interfaces desarrollado y la importancia que tiene dentro de cada una de las escenas del juego. Para ello, vamos a analizar por escenas los distintos bloques que hacen uso de interfaz y la interacción que realiza el jugador de ellas.

Todos los assets utilizados a continuación han sido importados a Unity, transformándolos en un tipo de textura llamado “Sprite”, que se utiliza con propósitos 2D y UI.

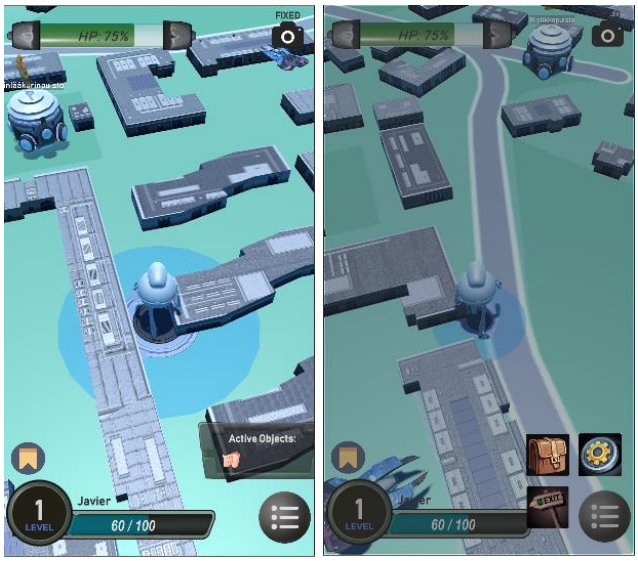
7.9.1. Interfaz en la Escena Mapa

Es la escena con mayor presencia de interfaces de cara al usuario. Existe una multitud de menús y secciones basadas en este punto. Aunque los elementos se han ajustado para ser

"*responsive*" (adaptable a distintas resoluciones de pantalla), se han realizado las pruebas principalmente en dispositivos con las resoluciones más empleadas en la actualidad: 1080x1920 y 1080x2160 (disposición *portrait* o vertical) y 1920x1080 y 2160x1080 (disposición *landscape* u horizontal), con un aspecto ratio correspondiente a: 16:9 y 16:10.

Todas las secciones o bloques de interfaz se encuentran incluidos en el *GameObject* "GUI", que analizamos con anterioridad en el punto 5.2. Este objeto tiene un script de comportamiento asignado como componente, llamado "GUI Manager". Permite controlar todas las entidades de la interfaz en el juego, centralizando muchas funciones de apertura de ventanas y diversas secciones. Cada una de estas secciones clave, tiene un script o clase propia asignada, que permite controlar sus propias funciones desde dentro, siempre y cuando el GUI Manager así lo permita. Analizamos a continuación estos bloques:

- **Interfaz activa en el mapa (disposición vertical)**: Se trata de la interfaz que se muestra constantemente mientras jugamos en la escena mapa. Es aquella que proporciona información adicional, mientras vemos los distintos objetivos repartidos por el mapa. Estos elementos podemos identificarlos en las siguientes figuras (75 y 76):



Figuras 75 y 76: Capturas de la escena mapa, en las que se ve la interfaz empleada

Fuente: Elaboración propia

Los elementos se detallan a continuación:

- Vida: Posicionada arriba a la izquierda, representa la salud del jugador en tiempo real. Para acceder a combates y puzles debe ser mayor que cero. Se emplea un diseño con tono de ciencia ficción, al introducir la barra de color verde dentro de un frasco, que combina metal y cristal. No se puede interactuar con ella.
- Botón de cámara: Situado arriba a la derecha, permite alternar con un solo click el tipo de cámara que sigue al jugador en cada momento. Está representada por un icono de una cámara y un texto que indica si la cámara se encuentra fija ("*fixed*") o libre ("*free*").
- Botón medalla: Situado bajo a la izquierda de la pantalla, justo encima del perfil. Permite acceder a la lista de medallas del jugador. El icono trata de representar una medalla, en colores grises y amarillos.
- Botón perfil/nivel: Situado en el borde abajo a la izquierda, que indica el nivel actual del jugador. Si pulsamos sobre él accedemos al perfil del jugador, con más detalles de este. Dispone de un icono circular metálico y el texto asociado con el nivel.
- Barra de experiencia: Posicionada justo a la derecha del botón de perfil, indica el nivel de experiencia del jugador actual. Además de expresarlo en cifras, se puede ver como la barra de color azul aumenta consecuentemente. No dispone de botón de interacción.
- Barra de objetos activos: Barra que se sitúa en el lado derecho de la pantalla, justo encima del botón del menú. Por defecto se encuentra oculto, hasta que el jugador utiliza un objeto que se mantiene activo durante un tiempo tras su uso.
- Botón de menú: Se posiciona justo en el borde de abajo a la izquierda de la pantalla. Su icono representa una lista de ítems o "menú de hamburguesa". Al interactuar con él se despliegan tres iconos, oscureciendo el resto de los elementos en pantalla. Los 3 iconos son:
    - Inventario: Permite acceder al inventario, asociado al jugador.
    - Opciones: Permite acceder al apartado de opciones.
    - Salir: Permite salir del juego con ventana de confirmación.

- **Inventario**: Sección en la que podemos consultar las armas y objetos adquiridos por el jugador a lo largo de la partida. Podemos ver su resultado en las siguientes figuras 77 y 78:

Figuras 77 y 78: Capturas del inventario completo en la escena mapa.
Fuente: Elaboración propia

Los distintos elementos que encontramos son:

- Botón para volver: Situado en el margen arriba a la izquierda, permite cerrar el inventario y volver al mapa.
- *Switch* de *Equipment/Objects*: Centrado en la parte superior, permite alternar entre la lista de armas y la lista de objetos del jugador. La sección se activará al pulsar sobre el botón con las letras correspondientes, con un color con fondo naranja.
- Casillas de objetos: Situados en el centro de la pantalla, rellenando el espacio disponible. Son las casillas que contienen los ítems recogidos y se representan mediante iconos asociados a cada uno de ellos. Si no contienen ningún objeto, se mostrarán vacías. Al pulsar sobre ellas, si contienen algún elemento, se selecciona en caso de ser un arma y se utilizarán en caso de ser un objeto.

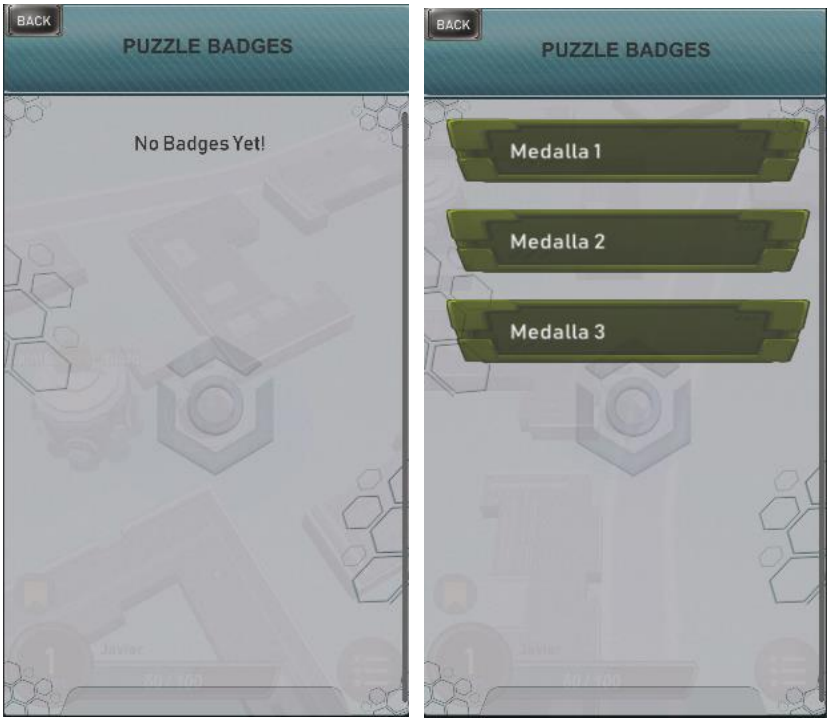
- Botón de información: Se posiciona en el margen superior derecho. Al pulsar sobre él, se mostrarán unos iconos de información sobre las casillas de los objetos que contengan elementos. Estos nos darán acceso al detalle de ítem.

- Botón de borrado: Se posiciona en el margen superior derecho con un icono de papelera. Al pulsar sobre él, se mostrarán unos iconos en esta vez de borrado en forma de "X" sobre las casillas de objetos. Si pulsamos sobre alguno de ellos, procederemos a borrar el ítem de la casilla correspondiente.

- **Detalle de ítem**: Sección que nos aporta más información sobre el ítem seleccionado para consultar. En caso de ser un arma, se desglosan todos los parámetros del arma en combate, así como la vida, la calidad y el tipo de arma. Cuando en un objeto, se nos indica únicamente si tiene uso instantáneo o no y si tiene algún valor de incremento asociado. En ambos casos existe una descripción asociada, almacenada en la clase del tipo de objeto en cuestión. También aparece el botón de atrás para volver al inventario (figuras 79 y 80 a continuación).

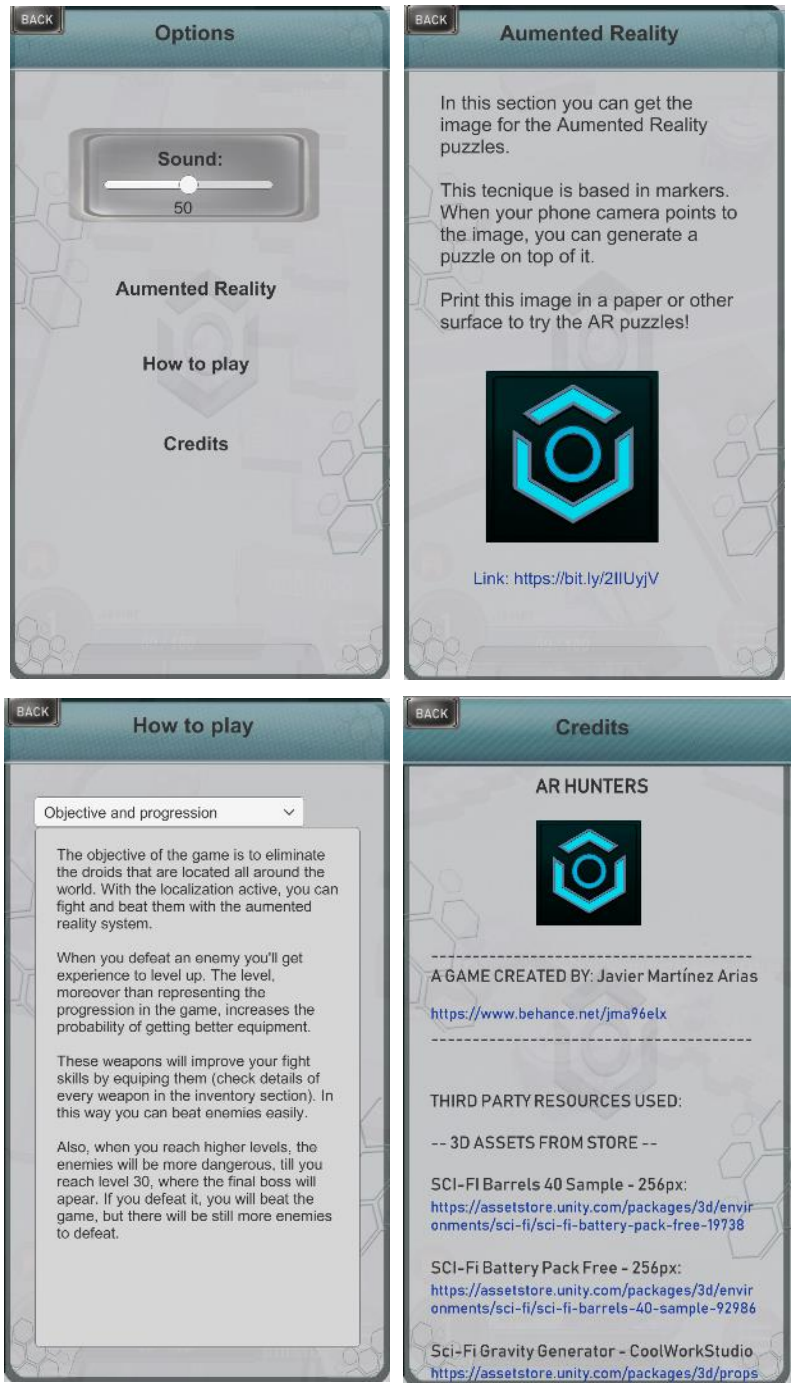
Figuras 79 y 80: Capturas del detalle de ítem en el mapa, en las que se ve la interfaz empleada para los objetos y las armas
Fuente: Elaboración propia

- **Medallas**: Sección del juego en la que podemos consultar las medallas obtenidas por el jugador a lo largo de la partida, por superar los distintos puzles. En su interfaz solo podemos destacar 2 elementos: El botón para volver al mapa (al pulsar sobre él) y la sección principal en la que aparecen las distintas medallas que no permite interactuar. Por tanto, se trata de una sección meramente informativa. Podemos ver las capturas en las figuras 81 y 82 a continuación:



Figuras 81 y 82: Capturas en la escena mapa de la interfaz de medallas
Fuente: Elaboración propia

- **Opciones**: El apartado de opciones es aquel que permite modificar y consultar varios ajustes como: el sonido, cómo jugar, etc. Puede verse reflejado en las siguientes figuras (de 83 a 86):



Figuras 83, 84, 85 y 86: Capturas en la escena mapa, en las que se ve la interfaz empleada en el apartado de opciones
Fuente: Elaboración propia

A partir de ellas podemos identificar:

- Ajuste del sonido: Permite modificar los sonidos de forma general en el juego. La forma en la que el jugador interactúa es mediante un “Slider” horizontal, que permite regular los valores de intensidad de la música.

- Realidad aumentada: Sección encargada de explicar en que consiste la realidad aumentada basada en marcadores, que puede utilizarse durante la resolución de los puzles. Además, nos ofrece un enlace de descarga de la imagen objetivo a la que debemos apuntar. Simplemente debemos pulsar sobre la imagen o el enlace en cuestión y se nos abrirá la página de descarga. Siempre podemos volver al menú de opciones con el botón de "atrás".
- Cómo jugar: Apartado de las opciones en el que podemos consultar el funcionamiento y objetivo de varias secciones del juego. Para ello se hace uso de un desplegable, que al pulsar en él podemos consultar todas las descripciones, según se necesite. Estos apartados son: Objetivos y progresión (generales del juego), inventario, batalla y puzles (cómo funcionan).

- **Perfil**: Al igual que ocurría en la sección de medallas, el perfil de jugador es una sección de la interfaz con función meramente informativa, por lo que no disponemos de entidades con las que interactuar (tales como botones). La vista de este apartado se refleja en la siguiente figura 87:

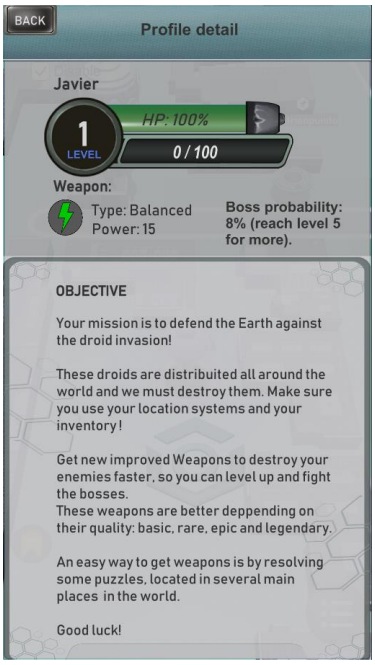
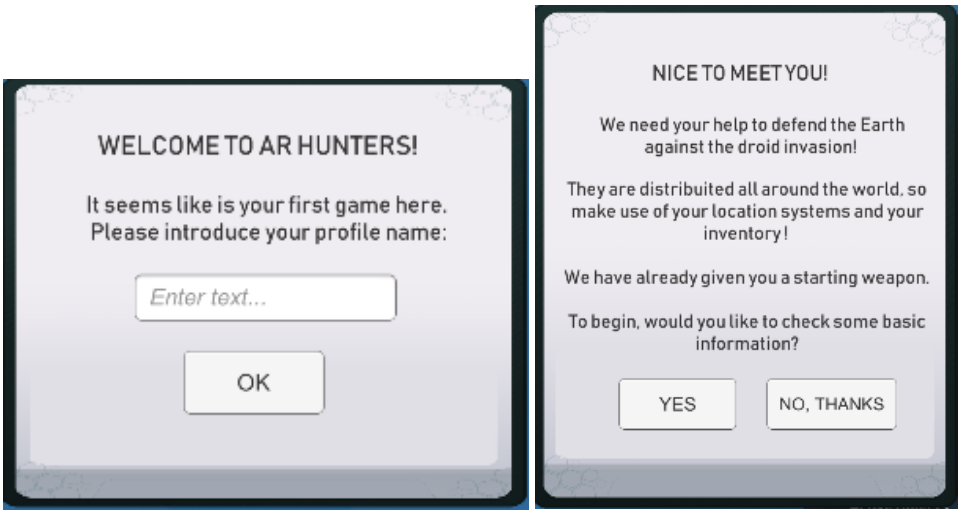


Figura 87: Captura de la interfaz con el perfil de jugador
Fuente: Elaboración propia.

- **Panel de transición**: Se trata de un panel que aparece únicamente cuando el jugador pulsa sobre algún elemento que fuerza la transición de escena, como son: puntos de puzle y naves enemigas. No es interactuable.
- **Ventana de bienvenida**: Se trata de un tipo de ventana emergente, que aparece en la escena mapa cuando no se encuentra una partida guardada con anterioridad. Permite establecer el nombre del jugador mediante un campo "*Textbox*" así como un botón. Tras confirmar aparece un texto de bienvenida, dando la opción para acceder al apartado de cómo jugar. Esta parte es opcional, teniendo dos botones para ello. Puede verse el resultado en las figuras 88 y 89:



Figuras 88 y 89: Capturas de las ventanas de bienvenida
Fuente: Elaboración propia

7.9.2. Interfaz en la Escena Batalla

La interfaz en la escena de batalla (disposición horizontal), es mucho más simple que en la escena anterior, pues en este caso, no hay ninguna sección oculta más allá de lo que se muestra durante el combate. Además, la cantidad de elementos que se pueden encontrar es mucho más reducida en este caso. Sin embargo, este hecho no significa que su relevancia sea inferior, pues en este caso, se trata de uno de los componentes principales que hacen posible el combate en primera persona (basado en lo que captura la cámara). En la siguiente captura (*figura 90*) se pueden ver los distintos elementos de la interfaz que aparecen en la escena:

Figura 90: Captura de la escena batalla, donde se pueden identificar los diferentes elementos empleados
Fuente: Elaboración propia

Como vemos la disposición del dispositivo pasa a ser en horizontal en esta escena, al contrario que la anterior. Los elementos que existen se explican a continuación:

- **Botón para volver**: Situado en el margen superior izquierdo. Al pulsarlo permite ir al mapa de nuevo.
- **Texto de enemigos**: Centrado en la parte superior, indica cuantos enemigos quedan en la escena en todo momento, por si no somos capaces de localizarlos. No tiene interacción con el usuario.
- **Información de arma**: Se posiciona en el margen superior derecho y permite conocer el arma equipada. En el caso de la figura, nos aparece el arma por defecto, pues no hemos equipado ninguna previamente en la escena mapa. El jugador no puede interactuar con ella, pues es únicamente informativa.
- **Botón de escudo**: Situado en la parte izquierda de la pantalla, permite defendernos ante los ataques de los enemigos. Al pulsar sobre él, se activará el escudo, mostrando una pantalla azul semitransparente y protegiéndonos temporalmente de los ataques.
- **Botón de ataque**: Posicionado en la parte derecha de la pantalla, permite atacar a los enemigos en la escena al ser pulsado. Crea una entidad de tipo bala en el centro de la pantalla, con una fuerza que la empuja hacia delante.
- **Barra de vida**: Se encuentra centrado en la parte inferior y permite dar información en tiempo real de la vida del jugador.

- **Pantalla de jugador (escudo y daño):** Se trata de una pantalla semitransparente que se superpone al resto de elementos de interfaz. Permite dar respuesta al jugador ante ciertas acciones, entre ellas:
    - Cuando es atacado, se muestra en color rojo, sincronizada con el decremento de la barra de salud.
    - Cuando se activa el botón de defensa, se muestra azul, representando el escudo activo.
- **Elementos de ambientación**: Los elementos de ambientación se han realizado a partir de un asset de terceros, con un objetivo meramente decorativo en la escena. Se han realizado algunos retoques en Photoshop para adaptar esta imagen a la escena. Por tanto, no tiene ninguna función para interactuar con el usuario.
- **Mirilla**: Posicionada en el centro de la pantalla, se emplea como referencia a la hora de disparar.

7.9.3. Interfaz en la Escena Puzle

Se trata de la escena en la que los elementos de la interfaz tienen una menor relevancia, pues no son esenciales para lograr el objetivo de la escena. Su disposición en este caso es horizontal. El número de elementos visibles de interfaz se limita a 4 elementos. Estos elementos pueden verse en la figura 91, a continuación:

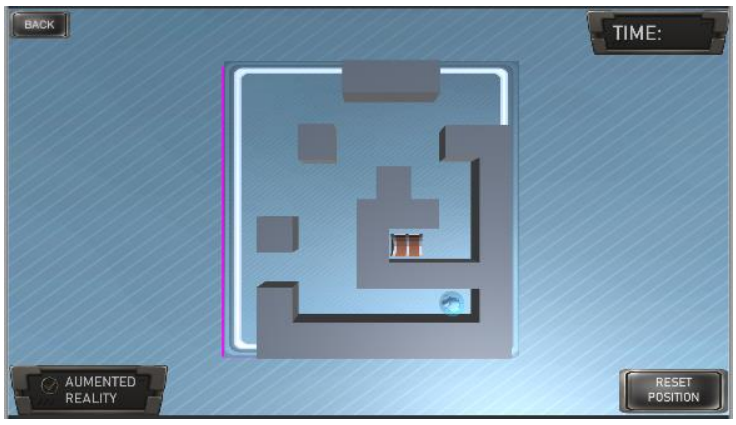


Figura 91: Captura de la escena Puzle en la que se pueden identificar los distintos elementos de la interfaz.
Fuente: Elaboración propia

Como ocurre en las escenas anteriores, todos los elementos se encuentran dentro de un *GameObject* padre llamado "*GUI*", que dispone de un script de comportamiento para

centralizar las funciones de los distintos botones u elementos de interfaz. Por tanto, podemos diferenciar los siguientes elementos:

- **Botón para volver**: Se posiciona en el margen superior izquierdo y permite volver a la escena mapa en cualquier momento al ser pulsado.
- **Panel con tiempo restante**: Contiene la información con el tiempo restante para superar el puzle. Se encuentra en el margen superior derecho.
- **Toggle/botón de realidad aumentada**: Permite activar o desactivar la realidad aumentada en la escena, cuando el jugador pulsa sobre él. Se encuentra en el margen inferior izquierdo.
- **Botón para restablecer posición**: Al pulsar sobre él, el jugador puede reposicionar la esfera en la posición de origen de la escena. Se encuentra en el margen inferior derecho.

7.9.4. Elementos comunes de Interfaz

Para terminar, se va a describir un elemento de interfaz que encontramos en las diferentes escenas por igual. Se trata de la ventana de notificación/confirmación, que se corresponde con el Game Object "Alert", que se citaba en el punto 5.4. Permite mostrar alertas informando al jugador en distintas facetas:

- **Victoria o derrota** en alguna escena de acción: tanto en el puzle cono en la escena de batalla, el jugador puede ganar o perder, por lo que la escena se encarga de llamar a la instancia de esta ventana para notificar al jugador en cada caso
- **Subida de nivel**: En cualquier escena, el gestor de niveles del jugador puede controlar la experiencia de esta entidad. Por tanto, cuando entra en la función de subida de nivel, muestra una alerta informando al jugador del nuevo nivel y el arma asociada obtenida (*figura 92*, más adelante).
- **Confirmación de borrado o uso de objeto**: Cuando el jugador se encuentra en el inventario y realiza una de estas dos opciones, siempre aparece una ventana para confirmar la acción del usuario.
- **Notificación de salud en escena mapa**: Cuando se pulsa sobre un puzle o enemigo para entrar en la escena, se comprueba previamente si la salud es superior a cero. De no ser así, se mostrará una ventana informando al jugador que debe curarse previamente.

- **Confirmación de cierre**: Si el jugador pulsa en el botón para salir del juego, aparece una ventana para confirmar esta acción, como ocurría con el borrado/uso de objetos.

Figura 92: Captura de la ventana emergente, en este caso, subiendo de nivel
Fuente: Elaboración propia.

7.10. Desarrollo de sonidos y música

En este apartado se van a explicar los distintos sonidos utilizados a lo largo del juego, contrastando la información planificada en el GDD.

En primer lugar, cabe decir que todos los sonidos introducidos en el juego pertenecen a fuentes externas, es decir, no han sido compuestas o creadas durante el desarrollo del proyecto. Sin embargo, en todo momento se ha buscado cumplir con lo redactado en el GDD, intentando encajar los distintos sonidos en las distintas partes del juego. Además, se han escogido buscando transmitir las sensaciones descritas, con una ambientación futurista y con un tono más relajado o serio en cada caso.

Para manejar y gestionar los diferentes sonidos en el juego, se ha hecho uso de dos componentes que nos ofrece Unity: *Audio Source* y *Audio Clip*.

- ***Audio Source***: Se trata del manejador de archivos de audio, incluido dentro de un *GameObject* como componente, que permite gestionar la reproducción de un clip de audio asociado. Aunque este limitado a un solo clip, la referencia del sonido a la que apunta puede ser modificada en tiempo de ejecución, por lo que siempre podemos alternarlos/modificar el sonido asociado.
- ***Audio Clip***: Es el tipo de variable que permite almacenar la referencia al archivo de audio contenido en la carpeta de *assets*.

Al igual que en el punto anterior, se van a dividir los distintos elementos en base a las escenas del juego.

7.10.1. Sonidos en la Escena Mapa

Empezamos describiendo la música o pieza principal utilizada en esta escena. Tras buscar en distintas fuentes, se ha escogido una pieza musical que encajara con las sensaciones descritas en el GDD, teniendo en cuenta además la ambientación creada en el juego. Una vez encontrada, se seleccionó frente al resto por transmitir un tono alegre, sin tensión y que pretende crear en el jugador un ambiente de exploración futurista.

Por otro lado, en esta escena se pueden encontrar sonidos distribuidos en las diferentes entidades, según la acción llevada a cabo por el jugador en cada momento. Los enunciamos a continuación en base a 2 tipos:

- **Sonidos de interacción con enemigos y puntos de misión en el mapa**: En este caso se ha empleado los mismos sonidos en ambos casos: el de transición entre escenas cuando sí podemos acceder a ellos (cumpliendo los requisitos) o el sonido asociado cuando se abre un mensaje de alerta, notificando al jugador que no cumple algún requisito.
- **Sonidos de interacción con botones de la interfaz**: En este caso encontramos varios sonidos en base a la función o situación de la escena, entre ellos:
    - Menú principal: Se reproduce cuando pulsamos sobre el menú principal y se despliegan las diferentes opciones.
    - Botón genérico: Se utiliza en la mayoría de botones de interacción en la interfaz. Algunos ejemplos de botones: atrás, medallas, detalle de objeto, inventario, opciones, etc.
    - Añadir objeto en inventario: Se reproduce cuando recogemos un objeto del mapa y se inserta correctamente en el inventario.
    - Seleccionar equipamiento: Se escucha al elegir una de las armas en el inventario.
    - Desplegar ventana: Cuando se muestra un mensaje emergente (casos comentados en el punto 5.9 de interfaz), se reproduce un sonido asociado.
    - Botón cancelación: Una vez desplegada la ventana, si permite seleccionar entre dos opciones y se indica que “no”, se reproducirá.

Por lo que respecta a los “Sonidos de los enemigos al aparecer cerca”, se ha descartado por motivos de implementación, pues los enemigos en la partida, por lo general, se encuentran posicionados en la escena una vez iniciamos el juego. Por tanto, para evitar la repetición de este sonido cada vez que se inicia la partida, se ha decidido no incluirlo.

7.10.2. Sonidos en la Escena Batalla

De nuevo, comenzamos describiendo la música empleada en esta escena. Tras buscar en distintas fuentes, se ha escogido una pieza musical opuesta al caso anterior en cuanto a tono o sensaciones se refiere. Sin dejar de tener un tono futurista, se seleccionó frente al resto por transmitir un tono serio, con la tensión de la batalla y que pretende crear en el jugador un ambiente de confrontación. Se utiliza para todos los tipos de enemigos.

Los sonidos con mayor presencia se encuentran en la **escena batalla**, y son:

- Disparos: Se emplea el mismo para los enemigos y para el jugador (función de ataque). Sin embargo, para el caso de los enemigos avanzados y jefe, existe un tipo de disparo fuerte, que genera otro sonido que no se había contemplado.
- Ataque cuerpo a cuerpo: Los enemigos cuando atacan cuerpo a cuerpo generan, avanzan más rápido, por este motivo se ha decidido asociarles un sonido de turbo.
- Defensa: Cuando se activa un escudo (función de defensa de entidades en batalla), se ha decidido diferenciar los sonidos de los enemigos del jugador, para evitar confusiones, por tanto no son el mismo.
- Curación (habilidad): Se reproduce cuando algún enemigo jefe se cura (pues el resto de las entidades no dispone de esta mecánica/habilidad). Cabe decir que la curación comparte sonido con el resto de las habilidades de los enemigos: escudo, reducir visibilidad del jugador e invisibilidad.
- Daño al jugador: Como se planteaba en el GDD, se diferencia del asociado a lso enemigos.
- Daño al enemigo.
- Daño al escudo del jugador: Se ha añadido para informar al jugador en estas situaciones, de forma que pueda reconocer cuando está haciendo su efecto el escudo.
- Canción victoria y derrota: se reproducen a la vez que se muestra la ventana emergente, en cada caso.

7.10.3. Sonidos en la Escena Batalla

En esta escena la presencia de sonidos es muy limitada, tal y como se planteaba en el GDD. Sin embargo, se han tenido que desechar algunos de estos sonidos, por la complejidad a la hora de manejar las interacciones entre algunas entidades. Se explica más adelante.

La música que suena en esta escena se encuentra en un término intermedio entre la escena batalla y escena mapa. Aunque en temática se encuentra más cercana a la de batalla, no tiene un tono tan serio. El tema escogido se puede describir como un tema "técnico", de misión delicada y que requiere concentración. Por tanto, cumple con lo indicado en la planificación.

Para los sonidos, se han desechado todos los sonidos en relación con las colisiones con las paredes/obstáculos, pues se han encontrado muchos problemas al tratar de ajustar el volumen con la velocidad que llevaba la esfera. En muchos casos, se generaban sonidos repetitivos que empeoraban la experiencia de juego, por lo que se ha decidido descartarlos.

La canción de victoria se reproduce al llegar a la meta (unificando estos sonidos planteados originalmente) y la de derrota cuando la cuenta atrás llega a 0. Son los mismos sonidos que para la escena de batalla.

7.11. Componente de realidad aumentada

En este punto se va a explicar cómo se ha desarrollado la componente de realidad aumentada para las escenas de batalla y puzle. Para el caso de la escena mapa, ya se ha mencionado con anterioridad, pues el proceso de obtención y recogida de los sistemas de ubicación se encuentra automatizado por la librería *Mapbox*, con los diferentes objetos y componentes ya descritos. Por tanto, pasamos a analizar de nuevo estas dos escenas, para explicar la generación de este efecto.

7.11.1. Realidad Aumentada en la Escena Batalla

La realidad aumentada desarrollada en esta escena se conoce como "*Markerless*", es decir, que no precisa de marcadores para su funcionamiento y generación de entidades. En este caso, son los enemigos que se encuentran distribuidos alrededor de la escena, los encargados de crear este efecto cuando apuntamos en las distintas direcciones con la cámara. Aunque existen otros tipos de realidades aumentadas basadas en *markerless* (por ejemplo, que reconocen superficies en el mundo real donde posicionarse), genera un efecto bastante realista. Pasamos a analizar cómo se ha desarrollado, teniendo como referencia la *figura 93*:

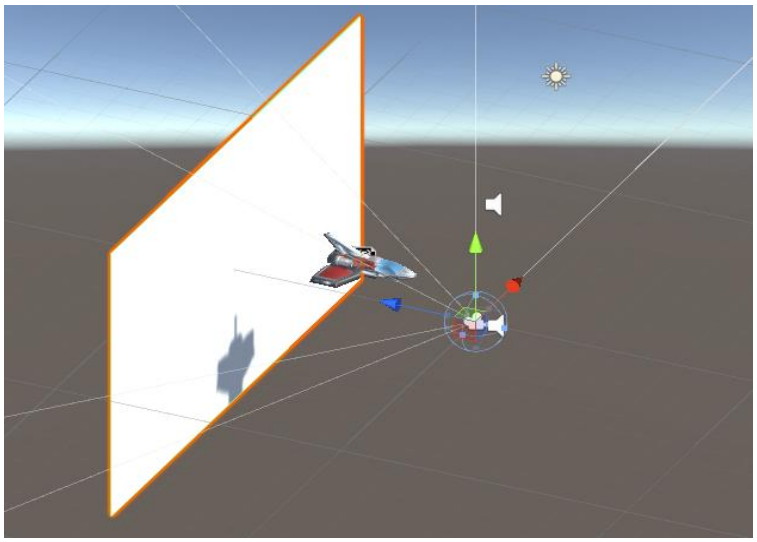


Figura 93: Representa la distribución de los elementos en la escena, para crear el efecto de realidad aumentada
Fuente: Elaboración propia

Para crear este efecto, se han empleado dos *GameObjects* en escena.

- Por un lado, tenemos la **cámara principal**, cuyo script de comportamiento está muy relacionado con la realidad aumentada para la batalla. El script de esta cámara recoge la orientación del dispositivo mediante el uso del giroscopio, por lo que en todo momento rota en la escena a la vez que el jugador mueve el dispositivo.
- Por otro lado, tenemos el **plano principal** de la escena. Se trata de un plano asociado a la cámara de la escena, con un script que altera su comportamiento normal. Este script permite convertir todo lo que captura la cámara del dispositivo, en una textura que se aplica al plano en cuestión. En Unity este tipo de objeto que recoge la información de la cámara y la transforma en textura se conoce como "*WebCamTexture*". En combinación con el giroscopio, la cámara de escena rota cuando el jugador mueve el dispositivo, al mismo tiempo que lo hace el plano en cuestión. Se consigue así sincronizar el giro de la cámara del dispositivo con el de la escena, proyectando las imágenes visibles por la cámara en todo momento.

  Hecho esto, simplemente se debe colocar el plano a una distancia de la cámara, de forma que todo lo que se interponga entre la cámara de escena y el plano, genera un efecto de realidad aumentada. En este caso los enemigos se encuentran siempre justo delante de las imágenes de la cámara (proyectadas en el plano), con distinta profundidad en base a las posiciones de la escena.

Los resultados de esta técnica pueden verse en la *figura 94*:

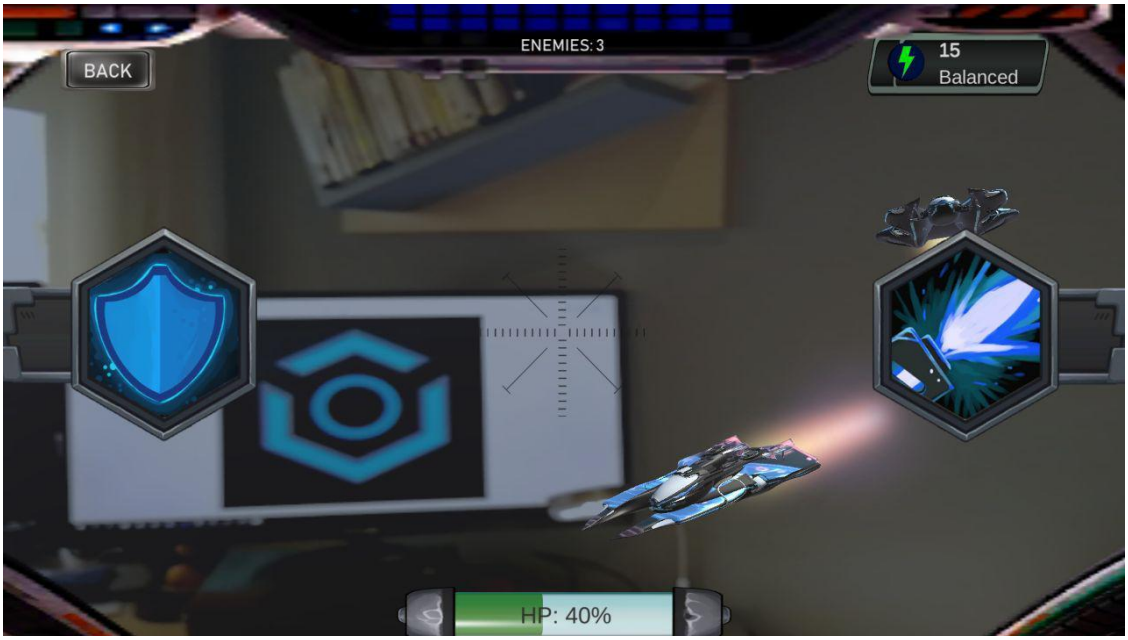


Figura 94: Realidad aumentada sin marcadores en la escena Batalla
Fuente: Elaboración propia

7.11.2. Realidad Aumentada en la Escena Puzle

Para la escena puzle se ha hecho uso de la librería *Vuforia*, como ya se ha comentado anteriormente. También se ha explicado que la técnica empleada en esta escena es conocida como "basada en marcadores", por depender de una imagen para poder crear el efecto deseado en superficies. Sin embargo, no se ha detallado el proceso seguido para realizar este efecto, por lo que a continuación se especificará por pasos:

- En primer lugar, para configurar un proyecto con *Vuforia* se deben descargar las librerías/packages correspondientes. Pueden obtenerse tanto desde la página oficial de *Vuforia* como en el propio entorno de Unity, ya que viene integrado en la instalación.
- Una vez instalado en el proyecto, se debe acceder a la página oficial de la librería y crear un usuario para empezar a gestionar el proyecto.
- Hecho esto, lo primero que se debe hacer es vincular la licencia asociada a la cuenta, con el proyecto de Unity en cuestión (casilla de licencia correspondiente).

- El siguiente paso es configurar el marcador, para ello, la página nos permite crear bases de datos que almacenen la información de las imágenes objetivo, por tanto, una vez tengamos la imagen preparada la cargaremos en la base de datos. Podemos cargar varias imágenes, pero en este caso solo se tratará una (*logo_ar1* en la *figura 95*).

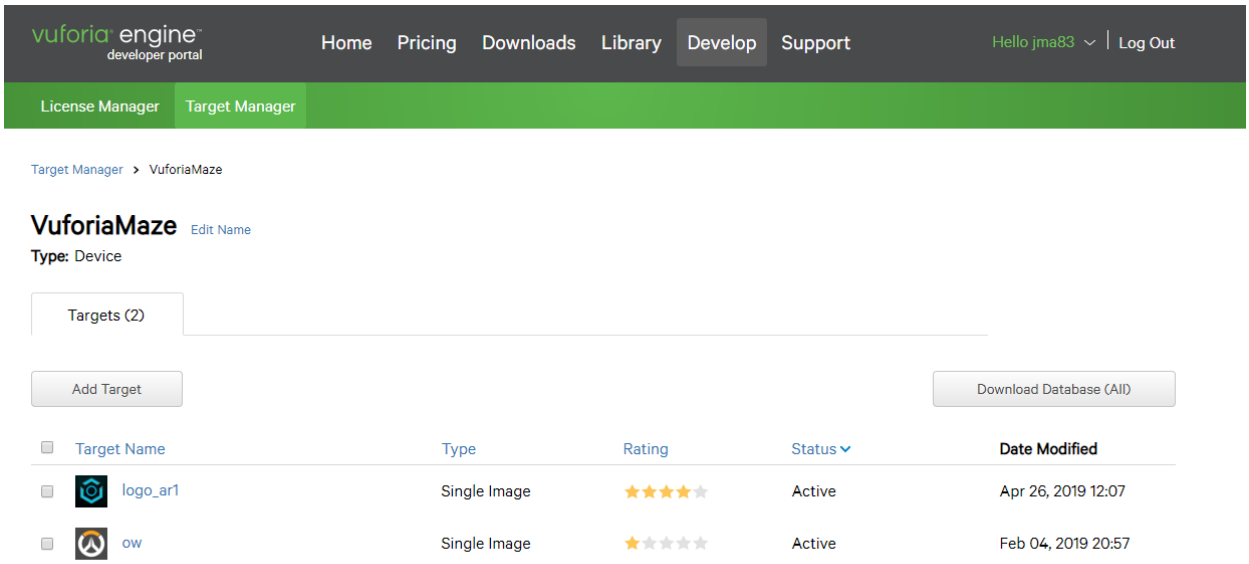


Figura 95: Captura de la web de Vuforia, configurando marcadores
Fuente: Elaboración propia, en la web de Vuforia

- Finalizado este proceso, ya podemos descargar la base de datos como *asset* para Unity, que permita importarlo sin problemas. Cuando esté importado, únicamente debemos añadir 2 objetos a la escena: la cámara de realidad aumentada que viene como *prefab* en la librería y el marcador o imagen objetivo. A esta imagen objetivo debe asociarse la base de datos y el marcador correspondiente que hemos importado.
- Para terminar, podemos añadir varios elementos visibles como objetos hijos de la imagen objetivo, de forma que sean visibles cuando la imagen sea reconocida.
- Tras ajustar algunos parámetros de la librería ya tenemos lista una aplicación de realidad aumentada basada en marcadores, a la que debemos añadir muchos elementos y scripts de comportamiento para que tenga un buen acabado.

Los resultados de esta técnica pueden verse en las siguientes figuras 96 y 97:

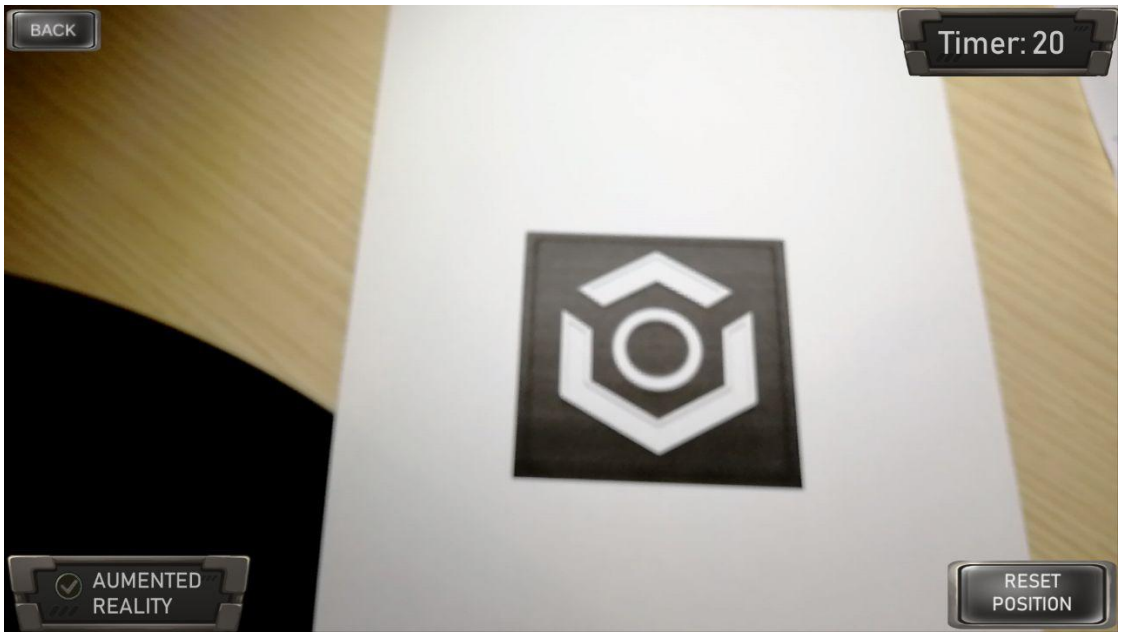


Figura 96: Reconocimiento de Imagen en la escena Puzle.
Fuente: Elaboración propia

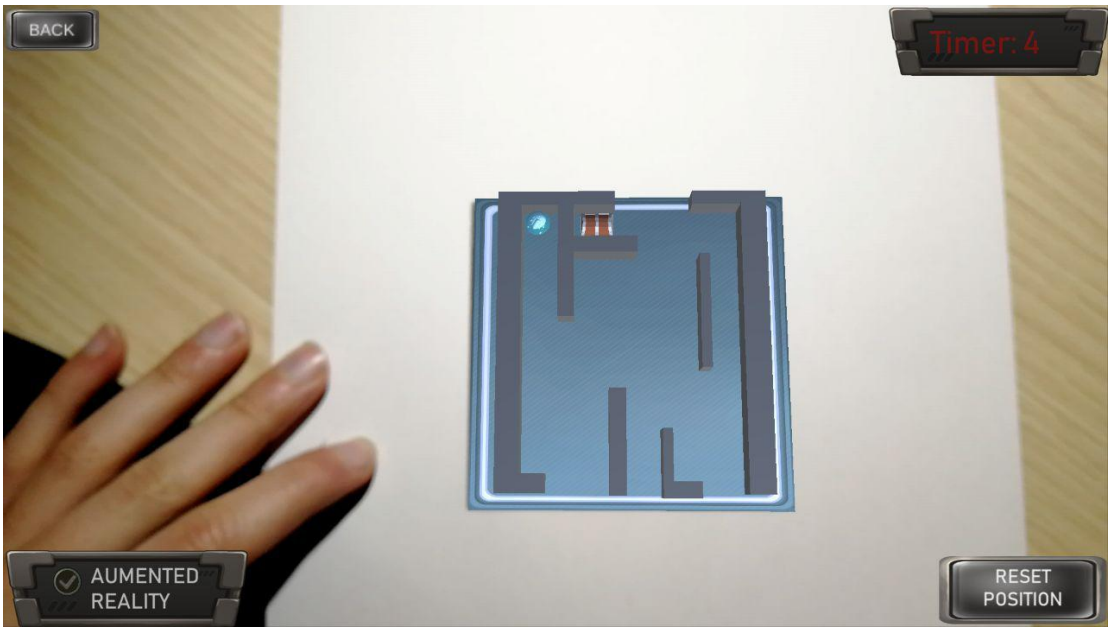


Figura 97: Realidad aumentada generada con imagen en la escena puzle
Fuente: Elaboración propia

8. CONCLUSIONES

Con tal de analizar las conclusiones con mayor detalle, se ha decidido dividir el punto en dos secciones: Revisión de objetivos y conclusiones de la planificación, que analizamos a continuación.

8.1. Revisión de objetivos

Para empezar, las conclusiones del proyecto son buenas, pues se ha logrado realizar un videojuego acabado y jugable que cumple con gran parte de los objetivos planteados originalmente. La idea de este proyecto de videojuego en todo momento ha sido explorar las grandes posibilidades que ofrece el uso de realidad aumentada, teniendo en cuenta su realización en el entorno de desarrollo Unity y orientado a la plataforma más extendida en la actualidad, es decir, los dispositivos móviles. Haciendo una valoración general, se puede decir que se ha alcanzado este objetivo principal, ya que el juego integra todos los aspectos clave mencionados con anterioridad. Pasamos a desglosar a continuación los distintos objetivos propuestos originalmente y contrastando con los resultados finales:

- **Idea y requerimientos:** Este objetivo agrupa aquellos puntos necesarios para el planteamiento inicial del juego. Se ha cumplido, reflejando las ideas base en el GDD (sección 6 del documento).
- **Planificación de tareas:** Las tareas se han establecido durante la planificación inicial del juego, una vez analizados los requerimientos. En combinación con la sección anterior, se corresponde con el apartado de diseño. Puede verse reflejado en el punto 5.1 del documento.
- **Aprendizaje de Unity:** Se ha superado la curva de aprendizaje requerida para desarrollar el videojuego en Unity, pues previamente no se había trabajado en profundidad en este entorno. Puede verse reflejado en los puntos 7.1 y 7.2.
- **Implementación de juego (mecánicas, IA, etc.):** Se trata de la lógica que integra el juego, estableciendo su comportamiento tal y como estaba previsto inicialmente. Se detalla en los apartados 7.3,7.4, 7.5, 7.6, 7.7 y 7.11.
- **Diseño visual de juego (modelos, entorno, interfaz, etc.):** Se trata de los elementos visuales que integra el producto final. Lo hacen más llamativo y agradable para la experiencia del jugador. Se detalla en los apartados 7.8, 7.9 y 7.10.

- **Mejoras y correcciones:** Aunque este apartado no se encuentra reflejado en la memoria final, ha sido necesario su cumplimiento para solucionar errores y ajustar elementos, obteniendo un resultado con mejor acabado.

Sin embargo, uno de los objetivos no alcanzados es la adaptación del juego a multiplataforma dentro del entorno de móvil. El caso más importante lo encontramos para los dispositivos con IOS que, con el resultado alcanzado, se encuentran fuera de los *SmartPhones* compatibles. Este objetivo no se ha podido cumplir por falta de tiempo y recursos, que permitieran probar el juego en estas plataformas.

Aunque no se trate de un objetivo como tal, cabe destacar que el videojuego se encuentra publicado en la actualidad en la tienda de Google "Play Store" con el nombre "***AR Hunters***", permitiendo así su libre descarga en los dispositivos Android.

Puede descargarse además desde fuera de la plataforma a través del siguiente enlace: http://bit.ly/318gwTh.

8.2. Conclusiones de la planificación

Por lo que respecta a las conclusiones de la planificación, debemos analizar las horas dedicadas y los desajustes, respecto a aquello planteado originalmente. Como se indicaba en el punto 5.1 del documento, el control de las horas totales dedicadas en el proyecto se ha realizado con la herramienta *Toggl* (descrita con anterioridad), de la cual se ha obtenido la siguiente captura (*figura 98*):

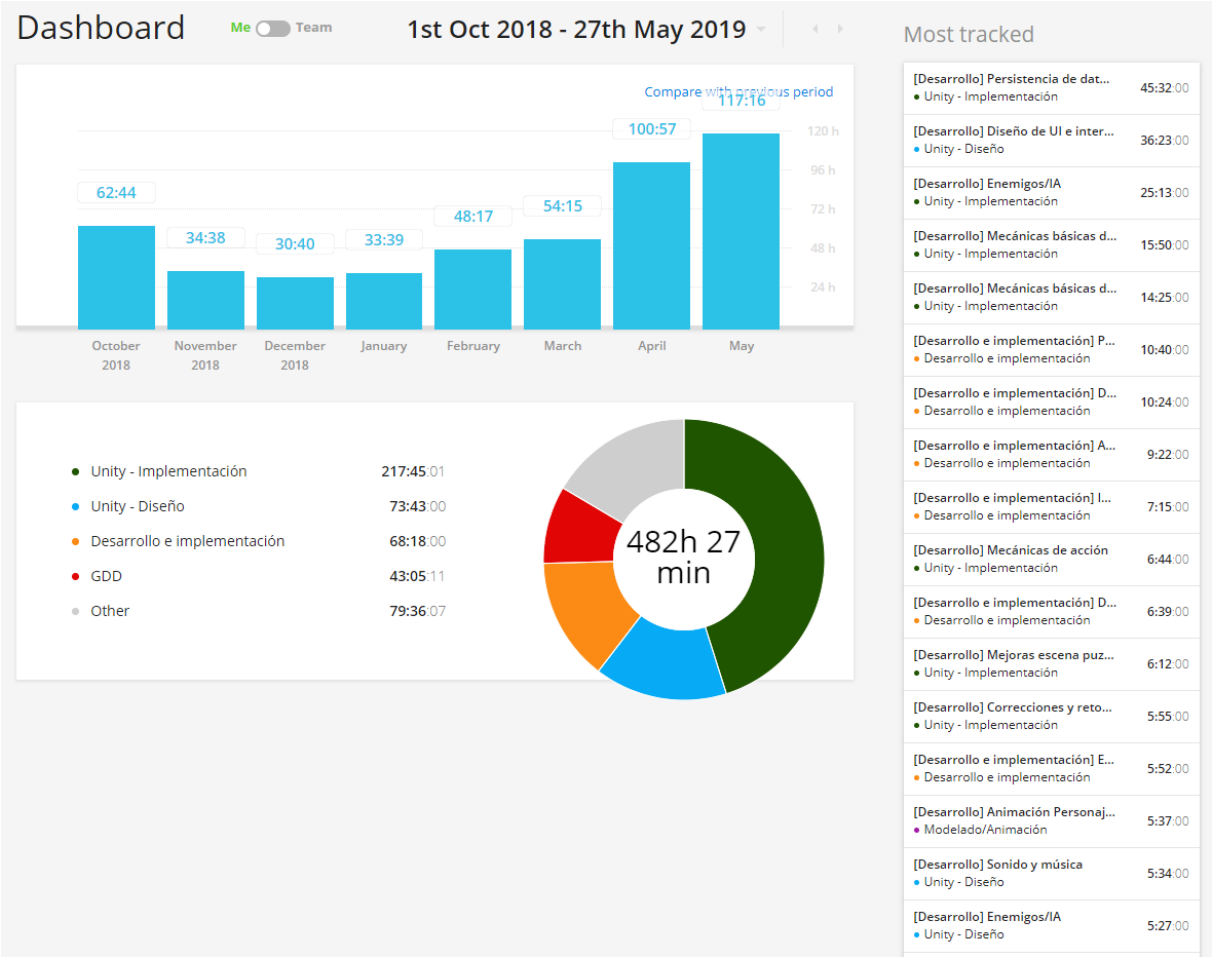


Figura 98: Reporte de horas totales del proyecto, realizado con Toggl
Fuente: Elaboración propia en Toggl

Las conclusiones que podemos extraer a partir de estos gráficos es la disponibilidad de dedicación al proyecto, que no ha sido la misma en todo momento por causas externas al mismo. A partir del mes de abril, además de tratarse de la recta final del proyecto, ha sido clave el tiempo disponible para poder dar por finalizado el desarrollo del proyecto, motivo por el que podemos ver un notable crecimiento de las horas empleadas.

La tarea que más ha afectado a las horas totales dedicadas es la “Persistencia de datos”, apartado que no se tuvo suficientemente en cuenta durante la planificación del proyecto y que ha obligado a reestimar tareas y reajustar tiempos. Otro aspecto que no se tuvo en cuenta durante la planificación fue que no se podían hacer la mayor parte de los puntos de “*Desarrollo e Implementación*” del documento (punto 7), a la vez que se desarrollaba el juego en Unity, pues este último se encontraba constantemente sometido a cambios. Por tanto, se decidió posponer este punto del documento al mes de mayo, para poder describir

todos los aspectos una vez terminado el videojuego. A pesar de estos inconvenientes, se han alcanzado los objetivos principales.

Para finalizar, este proyecto de videojuego, como muchos otros, se encuentra con la necesidad de actualizarse en el futuro, pues la tecnología avanza a ritmos estremecedores y las técnicas de realidad aumentada pueden verse sometidas a grandes cambios. Por tanto, aunque este proyecto pretenda ser ambicioso en las técnicas de realidad aumentada utilizadas, solo puede llegar a abarcar una parte de lo que ofrece la tecnología a día de hoy. Si quiere mantenerse vivo y competitivo en el mercado, debe actualizarse al mismo tiempo que lo hagan estas herramientas. En caso contrario, está destinado a ser superado por sus sucesores/competidores, por las fórmulas y posibilidades que serán capaces de ofrecer los programas de desarrollo y las librerías en el futuro.

Bibliografía

3D Formats, Unity. (2019). *Unity Files Documentation*. Obtenido de https://docs.unity3d.com/es/current/Manual/3D-formats.html

ARCore supported devices, Google. (2019). *Google Developers*. Obtenido de https://developers.google.com/ar/discover/supported-devices

ARCore, Wikipedia. (18 de Febrero de 2019). *ARCore*. Obtenido de https://en.wikipedia.org/wiki/ARCore

Aumented Reality, Xinreality. (2019). *Xinreality*. Obtenido de https://xinreality.com/wiki/Augmented_Reality

Behavior tree, Wikipedia. (15 de Mayo de 2019). *Behavior tree*. Obtenido de https://en.wikipedia.org/wiki/Behavior_tree_(artificial_intelligence,_robotics_and_control)

Decision Tree, Wikipedia. (31 de Mayo de 2019). *Decision Tree*. Obtenido de https://en.wikipedia.org/wiki/Decision_tree

Desarrollo basado en funcionalidades, Wikipedia. (5 de Abril de 2019). *Desarrollo basado en funcionalidades (FDD)*. Obtenido de https://en.wikipedia.org/wiki/Feature-driven_development

Documentación, Unity. (2019). *Unity Documentation*. Obtenido de https://docs.unity3d.com/Manual/UnityOverview.html

Documentation, Mapbox. (2019). *Mapbox Documentation*. Obtenido de https://docs.mapbox.com/

GanttProject. (2019). *GanttProject*. Obtenido de https://www.ganttproject.biz/

GDD, Wikipedia. (7 de Mayo de 2019). *Documento de Diseño de Videojuegos*. Obtenido de https://es.wikipedia.org/wiki/Documento_de_dise%C3%B1o_de_videojuegos

Github. (2019). *Github*. Obtenido de https://github.com/jma83/TFG_AR

GitKraken. (2019). *GitKraken*. Obtenido de https://www.gitkraken.com/

Herramientas realidad aumentada, Estudio Alfa. (21 de Marzo de 2017). *Estudio Alfa*. Obtenido de https://estudioalfa.com/top-herramientas-crear-apps-realidad-aumentada

Ingress, Wikipedia. (9 de Mayo de 2019). *Ingress*. Obtenido de https://es.wikipedia.org/wiki/Ingress

Invizimals, Wikipedia. (21 de Noviembre de 2018). *Invizimals (video game)*. Obtenido de https://en.wikipedia.org/w/index.php?title=Invizimals_(video_game)&oldid=869925373

Kanban, Wikipedia. (26 de Marzo de 2019). *Kanban (desarrollo)*. Obtenido de https://es.wikipedia.org/wiki/Kanban_(desarrollo)

Mapbox. (2019). *Mapbox*. Obtenido de https://www.mapbox.com/

Mapbox, Wikipedia. (16 de Noviembre de 2018). *Mapbox*. Obtenido de https://es.wikipedia.org/wiki/Mapbox

Mixamo, Adobe. (2019). *Mixamo*. Obtenido de www.mixamo.com

Motor de videojuegos, Wikipedia. (30 de Mayo de 2019). *Motor de videojuegos*. Obtenido de https://en.wikipedia.org/wiki/Game_engine

Panda BT Documentation, Eric Begue. (2019). *Panda BT Documentation*. Obtenido de http://www.pandabehaviour.com/?page_id=23

Panda BT, Eric Begue. (2019). *Panda BT*. Obtenido de http://www.pandabehaviour.com/

Pokémon GO, Wikipedia. (9 de Mayo de 2019). *Pokémon GO*. Obtenido de https://es.wikipedia.org/wiki/Pok%C3%A9mon_GO

Requisitos técnicos, Unity. (2019). *Requisitos técnicos Unity*. Obtenido de https://unity3d.com/es/unity/system-requirements

Studio, Mapbox. (2019). *Mapbox Studio*. Obtenido de https://studio.mapbox.com/

Toggl. (2019). *Toggl*. Obtenido de https://toggl.com/

Trello. (2019). *Trello*. Obtenido de https://trello.com/javiermartinez155/boards

Unity. (2019). *Unity*. Obtenido de https://unity.com/es

Unity, Wikipedia. (07 de Mayo de 2019). *Unity (motor de juego)*. Obtenido de https://es.wikipedia.org/wiki/Unity_(motor_de_juego)

Unreal Engine, Wikipedia. (8 de Mayo de 2019). *Unreal Engine*. Obtenido de https://es.wikipedia.org/wiki/Unreal_Engine

Videojuego, ASALE, RAE. (2019). *Diccionario de la lengua española (RAE)*. Obtenido de https://dle.rae.es/?id=bmnbNU7

Videojuego, Wikipedia. (27 de Mayo de 2019). *Videojuego*. Obtenido de https://es.wikipedia.org/wiki/Videojuego

Vuforia. (2019). *Vuforia*. Obtenido de https://engine.vuforia.com/engine

Vuforia, Wikipedia. (9 de Noviembre de 2018). *Vuforia Augmented Reality SDK*. Obtenido de https://en.wikipedia.org/wiki/Vuforia_Augmented_Reality_SDK

Fuente de los recursos utilizados

SCI-FI Barrels 40 Sample - 256px:

https://assetstore.unity.com/packages/3d/environments/sci-fi/sci-fi-barrels-40-sample-92986

SCI-Fi Battery Pack Free - 256px:
https://assetstore.unity.com/packages/3d/environments/sci-fi/sci-fi-battery-pack-free-19738

Sci-Fi Gravity Generator - CoolWorkStudio:
https://assetstore.unity.com/packages/3d/props/electronics/sci-fi-gravity-generator-82847

Workplace Tools - ILIAS KAP:

https://assetstore.unity.com/packages/3d/props/industrial/workplace-tools-86242

Sci-Fi Gun - GRASBOCK:

https://assetstore.unity.com/packages/3d/sci-fi-gun-30826

Low Poly RPG Item Pack - FI SILVA:

https://assetstore.unity.com/packages/3d/props/low-poly-rpg-item-pack-76088

FREE Skybox - Cubemap Extended - BOXOPHOBIC:

https://assetstore.unity.com/packages/vfx/shaders/free-skybox-cubemap-extended-107400

Panda BT Free - ERIC BEGUE:

https://assetstore.unity.com/packages/tools/ai/panda-bt-free-33057

Standard Assets:

https://assetstore.unity.com/packages/essentials/asset-packs/standard-assets-32351

Nytro OST - Crystalline Drift - isaiah658 (public domain):

https://opengameart.org/content/crystalline-drift

Psycho Punch Song (public domain):

https://opengameart.org/content/psycho-punch-loop-fps-shoot-em-up

Convert Song (public domain):

https://opengameart.org/content/convert

UI Sound Library (Attribution 3.0 Unported):

https://opengameart.org/content/ui-sound-effects-library

Sci-fi_computer_beeps (Attribution 3.0 Unported):

https://opengameart.org/content/9-sci-fi-computer-sounds-and-beeps

Sci-Fi Sound Library (Attribution 3.0 Unported):

https://opengameart.org/content/sci-fi-sound-effects-library

JRPG_UI (Attribution 3.0 Unported):

https://opengameart.org/content/jrpg-style-ui-sounds

Sci-fi-sfx (public domain):

https://opengameart.org/content/50-cc0-sci-fi-sfx

UI pack 1 (Attribution 3.0 Unported):

https://opengameart.org/content/ui-sound-effects-pack

Enemy spaceship models – Unnamed y Herminio:

https://opengameart.org/content/5-space-ships

https://free3d.com/3d-model/wraith-raider-starship-22193.html

Big Potion – Locad:

https://www.iconfinder.com/icons/2730331/bottle_colour_harry_magic_potion_potter_icon

Small Potion – Locad:

https://www.iconfinder.com/icons/2730329/bottle_colour_harry_magic_potion_potter_icon

Extend Capture Range - Chanut is Industries:

https://www.iconfinder.com/icons/2913111/fantasy_game_magic_magician_parchment_spell_scroll_wizard_icon

Laser Weapon:

https://icon-icons.com/es/icono/l%C3%A1ser-pistola/96178

Sword Weapon:

https://www.iconfinder.com/icons/2913105/battle_fantasy_medieval_steel_sword_war_weapon_icon

Badge Icon:

https://www.iconfinder.com/icons/1054992/badge_ribbon_icon

Durability Up:

http://clipart-library.com/clipart/533088.htm

Defensive Weapon:

https://pixabay.com/es/vectors/espada-escudo-caballero-152804/

Balanced Weapon:

https://game-icons.net/1x1/delapouite/shoulder-armor.html

Paper Bin Icon:

https://icon-icons.com/es/icono/basura-reciclaje/102620

Timer Icon:

https://icon-icons.com/es/icono/historico-tiempo-reloj-de-arena/54175

Spaceship Cockpit:

https://www.seekclipart.com/clipart/iRJhJwx_spaceship-clipart-cockpit-spaceship-cockpit-png-transparent-png/

FANTASY ICONS MEGAPACK - REXARD:

https://assetstore.unity.com/packages/2d/gui/icons/fantasy-icons-megapack-97000

TCG CARD DESIGN - REXARD:

https://assetstore.unity.com/packages/2d/textures-materials/tcg-card-design-42367

SCI-FI SKILL ICON PACK - REXARD:

https://assetstore.unity.com/packages/2d/gui/icons/sci-fi-skill-icon-pack-52852

GUI MEGAPACK - REXARD:

https://assetstore.unity.com/packages/2d/gui/icons/gui-megapack-101517

ENGINEERING CRAFT ICONS - REXARD:

https://assetstore.unity.com/packages/2d/gui/icons/engineering-craft-icons-71838