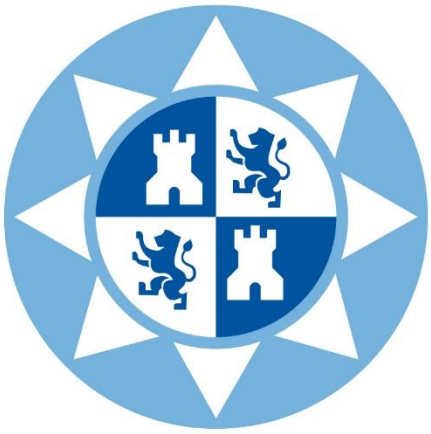
Universidad Politécnica de Cartagena



Escuela Técnica Superior de Ingeniería de Telecomunicación

Trabajo Fin de Grado:

Desarrollo de un juego RPG para prueba de algoritmos de IA con plataforma Unity e integración con la nube.

Autor:
Salvador Murcia Martínez

Director:
Dr. D Juan Ángel Pastor Franco

Autor	Salvador Murcia Martínez
E-mail del autor	Salvamm77@hotmail.com
Director	Dr D Juan Ángel Pastor Franco
E-mail del director	juanangel.pastor@upct.es
Título del TFG	Desarrollo de un juego RPG para prueba de algoritmos de IA con plataforma Unity e integración con la nube.
Descriptores	Unity, videojuego, RPG

Resumen

El TFG persigue el desarrollo de un juego RPG base sobre el que puedan realizarse otros TFGs relacionados con la IA de los personajes, las estrategias de evolución del propio juego, la distribución en red del juego y la conexión con la nube. El proyecto consta de de tres tareas principales, que se corresponden con sus fases: (1) estudio general de las estrategias de control del juego y de su evolución (alternativas), (2) selección e implementación de un conjunto de estas estrategias y (3) Pruebas de integración de estrategias de IA y conexión a la nube.

Titulación	Grado en Ingeniería Telemática
Departamento	Tecnologías de la Información y las Comunicaciones.
Fecha de Presentación	Octubre 2018

Índice de contenidos.

Índice de figuras.

Índice de tablas.

Abstract.

El proyecto consiste en el desarrollo de un videojuego RPG (Rol Play Game) para la plataforma Android preparado para el uso de diferentes algoritmos de IA y el tratamiento de los diferentes datos obtenidos por la aplicación en una web propia del videojuego.

Este proyecto ha requerido:

- Un estudio de la plataforma de desarrollo de videojuegos y aplicaciones **Unity**, así como de diferentes juegos ya existentes con el fin de conseguir una experiencia de usuario fluida en dispositivos Android y una experiencia RPG acorde.
- Estudio del funcionamiento de aplicaciones en dispositivos Android, así como su relación con Unity.
- Desarrollo de diferentes mecánicas de juego encarnadas por diferentes personajes y el propio terreno.
- Desarrollo y análisis de diferentes métodos para el tratamiento y visualización de datos de usuario.

El desarrollo se ha realizado íntegramente en **PC**, siendo únicamente necesario el dispositivo Android para realizar las sucesivas pruebas de funcionamiento.

Abstract (English).

The following project consists in the development of an RPG videogame for the Android platform prepared for the use of different AI algorithms and the treatment of the different data obtained by the application in a web of the video game itself.

- This project has required a study of Unity platform for videogames development, as well as of different existing games in order to achieve a fluid experience on Android devices and an appropriate RPG experience.
- Study of the operation of applications on Android devices, as well as their relationship with Unity.
- Development of different game mechanics embodied by different characters and the terrain itself.
- Development and analysis of different methods for the treatment and visualization of user data.

The development has been made entirely in PC, being only necessary the Android device to perform the successive tests of operation and quality.

1 Introducción.

Desde la aparición de los primeros videojuegos modernos en la década de los 60 de la mano de "*Pong*[1]", las empresas comenzaron a invertir en este tipo de sector junto a sectores como el del hardware para poder fabricar máquinas que pudieran soportar los videojuegos futuros. Desde entonces **el mundo de los videojuegos no ha dejado de crecer y desarrollarse** con el único límite que le ha impuesto la creatividad de los desarrolladores para adaptarlos a cualquier tipo de género imaginable.

El más inmediato reflejo de la popularidad que ha alcanzado el mundo de los videojuegos en las sociedades contemporáneas lo constituye una industria que da empleo a 120 000 personas y que genera unos beneficios multimillonarios que se incrementan año tras año.

Uno de los géneros que más destacó fue el *role-playing game* o RPG, su mayor exponente en sus comienzos fue ***Dungeons & Dragons***[2] *(Ver figura 1)*, juego que se inicia con una ya establecida tradición de interpretación de roles. Un juego de rol es un tipo de juego en el que los participantes asumen los roles de personajes y, a través de ellos, crean una historia. Los participantes determinan las acciones de su personaje basados en su caracterización y sus acciones pueden o no tener éxito en función de un sistema de reglas.

Figura 1: Captura del juego de mesa de ROL "Dungeons & Dragons"

[1] **Pong**: Pong fue un videojuego de la primera generación de videoconsolas, basado en el tenis de mesa.
[2] **Dungeons & Dragons**: juego de rol de fantasía heroica originalmente derivado de juegos de tablero jugados con lápiz, papel y dados.

El género comenzó en el sector de los videojuegos a mediados de los años 70, inspirado por juegos de rol de mesa como ***Dungeons & Dragons***. Otras fuentes de inspiración para los primeros videojuegos de rol fueron videojuegos de aventura, juegos de estrategia como el ajedrez o novelas de fantasía.

1.1 Objetivos

El objetivo del presente proyecto es crear un videojuego RPG de la forma más funcional, divertida y completa posible, A continuación, se explicarán cuáles son estos objetivos, seguidos de los puntos del videojuego que persiguen estos objetivos.

Los objetivos a cumplir son los siguientes:

- Presentación e **interfaz** atractiva, sencilla de comprender por el jugador y adaptación correcta a los dispositivos Android. Este punto es la piedra angular del proyecto, ya que los gráficos presentes en el juego y cómo son tratados por un dispositivo con una potencia de procesado limitada, como es un teléfono móvil, constituye la tarjeta de presentación del juego.

  Para conseguir este objetivo nuestro proyecto se ha centrado en el uso de gráficos **Low-poly**[3], y texturas caricaturizadas, que permiten un procesamiento más fluido en el dispositivo y ser más agradable para el jugador.

- Deberá tener un **modo para un solo jugador (Singleplayer)** para que el jugador no se vea limitado por tener que estar conectado y necesitar de más jugadores para poder jugar. Este apartado debe estar adaptado diferenciándose del multijugador en la cantidad de personajes, o dificultad.

  El proyecto cuenta con un modo para un solo jugador totalmente jugable sin necesidad de una conexión a internet, con una mecánica de juego adaptada al mapa, consiguiendo que el jugador tenga que viajar por todo el mapa para poder cumplir el objetivo.

- Cada personaje deberá proporcionar una **experiencia de Rol** completa, manteniendo cierto equilibrio entre todos los posibles personajes principales, de forma que los jugadores se vean incitados a jugar con todos.

  Cada personaje cuenta con tres habilidades claramente diferenciadas entre ellos, adaptadas a nivel de estadísticas con el fin de que ninguna clase sea superior al resto, pero sigan siendo únicas.

[3] **Low poly**: Nombre que se le da a aquellos modelos 3D que no cuentan con gran cantidad de polígonos.

- El **sistema de inventario** deberá ser fluido y dinámico.

  Mediante el sistema de inventario el jugador puede equipar, comprar, desequipar, vender diferentes objetos con diferentes características que le ayudarán en el transcurso de sus partidas. Esto le permite sentir una progresión en el juego, obteniendo cada vez mayor equipamiento y volviéndose más poderoso.

  Mediante esta mecánica el jugador se marcará objetivos que le inciten a seguir jugando y mejorar su personaje para obtener potenciadores de sus estadísticas.

- El jugador debe tener acceso a sus **estadísticas**, para comprobar su estado, y que combinación de ítems le es más beneficiosa en cada momento, además de como poder compararlas con las del resto de jugadores, a través de la web.

  Mediante la equipación de objetos, el jugador puede ver en tiempo real sus estadísticas de ataque y defensa modificados, con estas estadísticas puede además plantear una estrategia más ofensiva o defensiva.

1.2 Motivación

La creación de este videojuego viene motivada por las siguientes razones:

Actualmente los principales videojuegos de rol presentes en tiendas como Google Play presentan por un lado unos controles de movimiento basados en pulsación-desplazamiento, o combate por turnos. Queremos cambiar eso y darle un control clásico similar al que se obtendría en una consola o PC.

- La **plataforma Android** cuenta con diversos videojuegos RPG, pero en su mayoría en 2D por turnos, nosotros vamos a buscar recrear la experiencia vivida en consolas y PC de visualización de estos videojuegos en entornos 3D.

- Otro de los motivos es crear un videojuego que por un lado no te obligue a introducir saldo casi de forma obligada para poder avanzar, sino utilizar unas políticas de juego y recompensa no tan abusivas no tan abusivas como se hacía en antiguos juegos.

- La principal motivación sería la motivación por hacer un videojuego propio, seguir su metodología de diseño y lanzamiento y poder verlo finalmente funcionando.

1.3 Estructura de la memoria

Aparte de en esta introducción, la memoria se divide en los siguientes puntos:

- **Capítulo 2:**

En este capítulo pone en contexto la historia del videojuego y su evolución con el paso de los años y los diferentes avances tecnológicos utilizados en las diferentes consolas. Otro punto que se trata en este capítulo es la evolución del género RPG, género al que pertenece el videojuego objetivo de este proyecto.

Para finalizar este apartado enumeraremos las diferentes formas de clasificación del contenido de los videojuegos según el sistema PEGI.

- **Capítulo 3:**

El capítulo 3 describe las diferentes **herramientas utilizadas en el proyecto** y en que apartados o aspectos han sido utilizadas en el proyecto, haciendo hincapié en la herramienta principal, el motor gráfico Unity.

- **Capítulo 4:**

En este capítulo se habla de las **principales características con las que cuenta nuestro videojuego**, modos de juego, así como una puesta en contexto de este mediante un Storyboard.

- **Capítulo 5:**

Este capítulo **recoge un estudio sobre las principales características de Unity**, una explicación de alguno de **sus componentes** principales, así como el desarrollo a modo de orientación para el lector del uso de estos diferentes componentes en un desarrollo real como es el desarrollo de un personaje principal.

- **Capítulo 6:**

Este capítulo cuenta el **diseño de las diferentes mecánicas del juego**, empezando por el diseño del mapa, personajes principales y sus diferentes habilidades, la interfaz de usuario, los personajes enemigos, la gestión de recursos e inventario. También se describe el manejo de los diferentes menús e implementaciones con las que cuenta el juego y los métodos de optimización utilizados.

- **Capítulo 7:**

Expone el proceso con el cual **se compila nuestro proyecto** dejándolo listo para probar en un dispositivo móvil.

- **Capítulo 8:**

Describe el **proceso de desarrollo** seguido en el proyecto mediante la planificación de sprints.

- **Capítulo 9:**

Expone el **resultado final que nos ha arrojado el proyecto** y en base a ello, algunas **posibles mejoras** que pueden ser añadidas a éste, así como un breve análisis del mercado al que va destinado.

- **Capítulo 10:**

Finalmente, en este capítulo se exponen unas **conclusiones finales** sobre la experiencia, y las diferentes dificultades afrontadas durante el desarrollo del proyecto.

2 Antecedentes

A continuación, haremos un repaso a la historia, desde el origen de los videojuegos y su constante evolución hasta el día de hoy, en especial de aquellos pertenecientes al género de estudio, el RPG.

2.1 Historia de los videojuegos

Durante bastante tiempo ha sido complicado señalar cual fue el primer videojuego, principalmente debido a las múltiples definiciones de este que se han ido estableciendo, pero se puede considerar como primer videojuego el ***"Nought and crosses"***, desarrollado por Alexander S. Douglas, un profesor británico de ciencias de la computación en 1952 *(Ver figura 2)*. El juego era una versión computarizada del **tres en raya** que se ejecutaba sobre la **EDSAC**[4] y permitía enfrentar a un jugador humano contra la máquina.[5]

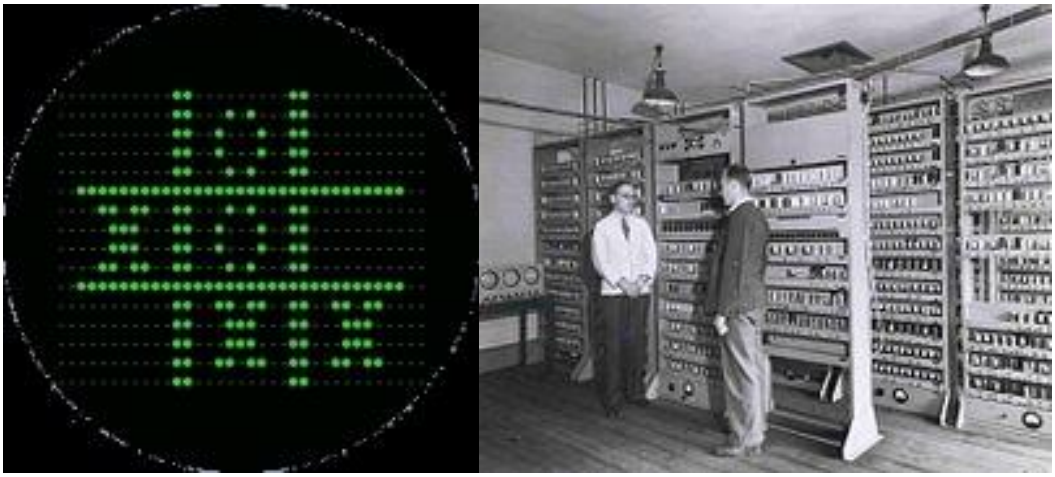


Figura 2: Fotografía de la EDSAC con el juego OXXO

De aquí en adelante se empezó a buscar el adaptar juegos clásicos de mesa, o simulaciones de la vida real al mundo digital.

En 1958 el físico estadounidense **William Higginbotham** creó, sirviéndose de un programa para el cálculo de trayectorias y un osciloscopio**, *"Tennis for Two" (tenis para dos):*** un simulador de tenis de mesa para entretenimiento de los visitantes de la exposición **Brookhaven National Laboratory**[6]. Este videojuego fue el primero en permitir el juego entre dos jugadores humanos

[4] **EDSAC**: Antigua computadora británica.
[5] **Fuente**: https://www.fib.upc.edu/retro-informatica/historia/videojocs.html
[6] **Brookhaven National Laboratory:** es un laboratorio nacional del Departamento de Energía de los Estados Unidos ubicado en Upton, en Long Island. Está especializado en investigaciones sobre física nuclear.

En 1972 sale al mercado "***Pong***" (primer nombre de la consola **Atari**[7]) que reestructuró por completo el negocio del entretenimiento. Desde la aparición del Atari la diversidad de consolas y videojuegos parece no terminar.

La década de los 80s fue un período de fuerte crecimiento en el sector del videojuego. Fue en estos años donde nuevas empresas comenzaron a desarrollar nuevas máquinas de sobremesa, juegos y máquinas recreativas.

Con el lanzamiento del famoso videojuego "**Super Mario Bros**" *(Ver figura 3),* el primer juego con una historia y varios niveles con principio y final, también llegaron nuevas ideas para implementar en los futuros juegos. Este lanzamiento supuso el mayor punto de inflexión en la historia del videojuego

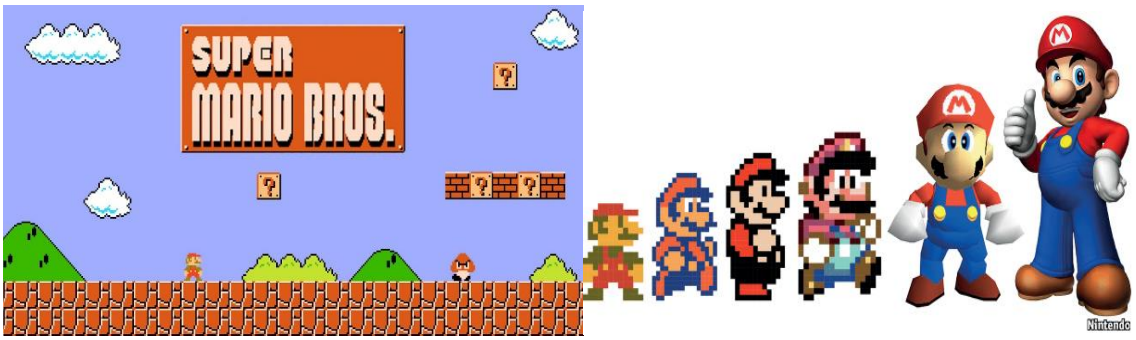


Figura 3: Imagen del primer Syper Mario Bros Junto a su evolución en sus diferentes entregas.

Exponencialmente la tecnología fue evolucionando y con ella las diferentes consolas. La evolución más destacable llegó con las **consolas de 32bits**[8] como la **PlayStation de Sony** y la **Nintendo 64** de **Nintendo** *(Ver figura 4)* que fueron pioneras en los primeros juegos que utilizarían **modelados 3D**[9]**.** La consola de **SONY** fue la primera que incorporó el **CD-ROM**[10] como medio de reproducción de los videojuegos, esto permitió crear juegos más largos y con más capacidad dentro de las restricciones permitidas por el hardware de la consola.

En esta década surgieron franquicias de videojuegos muy populares que siguen jugándose en la actualidad aprovechándose de la mayor potencia de las consolas de última generación. Algunos de estos títulos fueron: "***Resident Evil*"** (Capcom), "***Gran Turismo*" (**Polyphony Digital), "***Metal Gear Solid*" (**Konami), "***Super Mario*"** (Nintendo), etc. *(Ver figura 5)*

[7] **Atari**: es una de las productoras de videojuegos independientes más grande en Estados Unidos, comparte el mismo nombre que su primera consola.
[8] **Consola de 32 bits**: Nombre que se le da a la quinta generación de consolas, la cual supuso el paso de los 2D a los entornos tridimensionales 3D.
[9] **Modelo 3D**: Se llama al producto fruto del desarrollo de una representación matemática de cualquier objeto tridimensional (ya sea inanimado o vivo) a través de un software especializado.
[10] **CD-ROM:** (sigla del inglés Compact Disc Read-Only Memory) es un disco compacto con el que utilizan rayos láser para leer información en formato digital.

Figura 4: Consola Sony Playstation y Nintendo 64, fueron pioneras en el uso de modelos 3D.

Otro punto de inflexión en la industria fue a comienzos del siglo XXI, cuando la compañía japonesa, **SONY**, saco al mercado su nueva consola **PlayStation 2** revolucionando nuevamente el mundo de los videojuegos. La Playstation 2 apostó por la tecnología del momento, el **DVD**[11]**.** Sony dejó de lado el **CD-ROM** pues según la compañía se había quedado obsoleto. Además, el hardware de la consola había sido mejorado de tal forma que se empezaron a diseñar videojuegos con aspectos más realistas utilizando nuevas técnicas de iluminación y sombras.

[11] **DVD (*Digital Versatile Disc*)**: El **DVD** es un tipo de disco óptico para almacenamiento de datos.

Figura 5: Algunos juegos que explotaron el uso del CD-ROM para mejorar su aspecto gráfico.

A partir de este momento el desarrollo de videojuegos empieza a estar limitado a las prestaciones que ofrecen las consolas. Los nuevos videojuegos que empezaron a salir en aquel momento intentarían explotar al máximo todos los recursos de esta para poder realizar todo tipo de acciones que eran impensables solo unos pocos años atrás.

Hoy en día cada vez estamos más impactados con la realidad y calidad que ofrecen algunos videojuegos, siendo uno de estos el **producto cultural más rentable de la historia (*Grand Theft Auto V*)**[12] *(Ver figura 6)*, por encima incluso del cine. El mundo del videojuego ha sufrido un cambio exponencialmente, para mejor, que hace cuestionarse al jugador cual será los límites en un futuro.

[12]**Fuente**: https://www.marketwatch.com/story/this-violent-videogame-has-made-more-money-than-any-movie-ever-2018-04-06

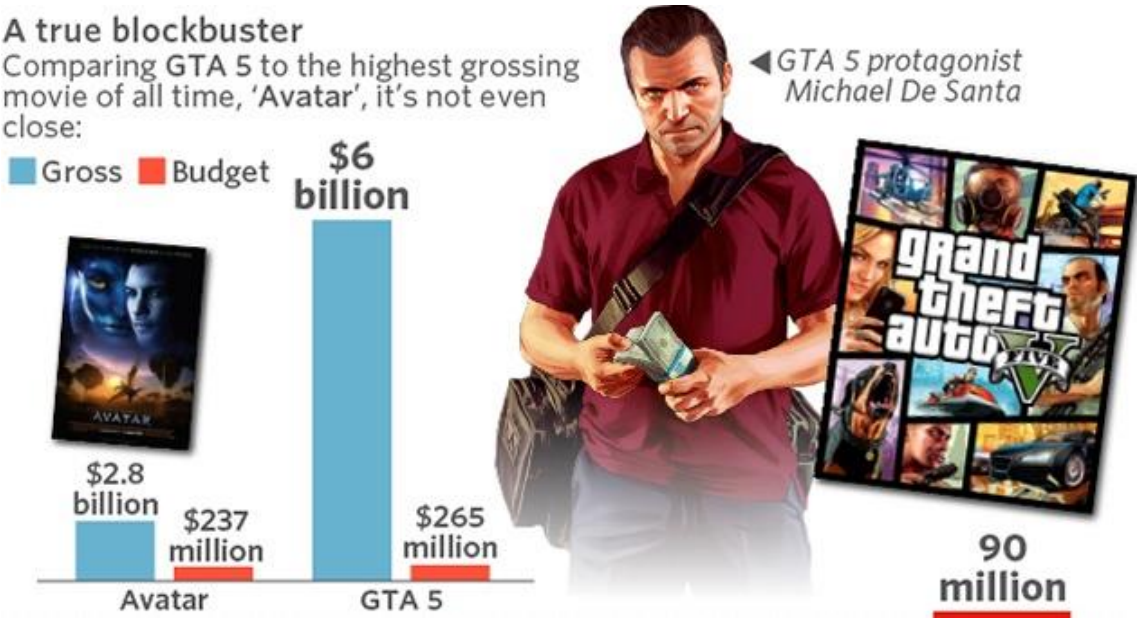


Figura 6: Grafico de los medios culturales con mayores ganancias

2.2 Género RPG en los videojuegos.

Los *Role Playing Games*, RPGs o comúnmente llamados juegos de rol, nacieron basándose fielmente en el estilo de *Dungeons & Dragons*, juego de rol del cual tomaron muchas de sus mecánicas como la creación de personajes, razas, **armamento**, niveles, **enemigos**, *quests*[13] y un largo listado de **mecánicas** y características. Los juegos de rol convencionales se juegan con hojas de papel, libros, dados y miniaturas donde todo es debidamente presencial, mientras que los RPG son los que asociamos comúnmente a los videojuegos.

2.2.1 Evolución de los videojuegos RPG

Los primeros videojuegos de género RPG nacieron en la década de los 80's con la salida de *Dungeons & Dragons*, llevado para la **computadora TRS-80**[14] y con un peso en total de sólo 16KB. El juego tuvo características únicas para esas fechas, las cuales hoy día son fundamentales en el género, tales como un analizador sintáctico por la línea de comandos, generación de personajes, una tienda para comprar equipamiento, combate, trampas y calabozos por explorar. De hecho, este juego desarrollaba un **sistema de combate a tiempo real**[15].

Para 1982 fue desarrollado el primer juego RPG en la Atari llamado "***Dragonstomper***" *(Ver figura 7)*. En 1984 salió a la venta "***Bokosuka Wars***" juego

[13] **Quest**: Nombre que se le da a las misiones o tareas dentro de un juego RPG
[14] **Computadora TRS-80:** designación para varias líneas de sistemas de microcomputadores producidos por Tandy Corporation.
[15] **Sistema de combate a tiempo real:** Sistema de juego consistente en que el tiempo transcurre de igual manera para todos los bandos en el combate, no existiendo parones.

con el que nacería un nuevo subgénero que se conoce como S-RPG o RPG de estrategia donde los jugadores deben de avanzar en un mapa por casillas y atacar a los enemigos del bando contrario.

Figura 7: Portada del videojuego "Dragon Stomper" junto a una captura del juego.

Durante los 80's y 90's los juegos del género se especializaban en ser juegos para un solo jugador. No fue hasta mitad y finales de los 90's con la masificación de Internet que empezaron a nacer los **MMORPG,** títulos como "*Runescape*", "*Lineage*", "***World of Warcraft*****"** *(Ver figura 8)* entre otros.

Pero, ¿qué define a un RPG? La fórmula clásica es la creación/optimización de un personaje, un sistema de inventario de objetos y/o habilidades de personajes, toma de decisiones y compraventa de objetos. Tomando como base la fórmula antes mencionada, hay muchos subgéneros de los RPG**: A-RPG, RPS, S-RPG y MMORPG**[16] ***(Ver Anexo: subgéneros de los RPG)***

Figura 8: Partida del videojuego RPG "World Of Warcraft"

Con la llegada de los dispositivos móviles el género de los videojuegos RPG se vio enormemente afectado puesto que adaptar un juego con tantas mecánicas y

[16] **Fuente**: https://revistayumecr.com/retro-gaming-poco-la-historia-los-rpg-opinion/

opciones concebido en su origen para jugar con teclado y ratón iba a ser una ardua tarea.

En estos dispositivos debido a limitaciones del Hardware no se ha explotado el género, aunque cuenta a día de hoy con cantidad de juegos de este género, en su mayoría estamos hablando de remasterizaciones o adaptaciones de juegos clásicos de 16 bits adaptados mediante emuladores.

2.3 PEGI

Uno de los factores a tener en cuenta cuando se desarrolla un videojuego es el público al cual va dirigido, es decir, los juegos están clasificados por edades y no todos los jugadores son aptos para jugar a todos.

Este control se lleva a cabo con **PEGI** (*Pan European Game Information*), que según la asociación española del videojuego (AEVI): *"Es el mecanismo de autorregulación diseñado por la industria para dotar a sus productos de información orientativa sobre la edad adecuada para su consumo."*[17]

El Sistema PEGI se puede dar de dos formas, una orientada a la edad mínima recomendada para su consumo por el consumidor y otra relativa al contenido del producto. *(véanse Figuras 9 y 10).*

Figura 9 Clasificación por edades PEGI

Los descriptores de contenido se clasifican en:

• **Lenguaje soez**: El juego contiene palabrotas o lenguaje soez.

• **Discriminación**: Contiene violencia o discriminación de razas.

• **Drogas**: Hace referencia a drogas o derivados.

• **Miedo**: Contiene escenas perturbadoras o aterradoras.

• **Juego**: Contiene apuestas con dinero real o ficticio dentro del juego.

• **Sexo**: Se pueden presenciar relaciones sexuales o desnudos.

• **Violencia**: Contiene escenas de muerte y violencia de cualquier tipo.

• **Online**: Contiene algún modo de juego en linea.

[17] **Fuente**: http://www.aevi.org.es/documentacion/el-codigo-pegi/

Figura 10 Clasificación PEGI por contenido.

Según PEGI, el videojuego objetivo de este proyecto entraría dentro de la categoría **Violencia y Online,** con una edad mínima recomendada de **12 años.**

3 Herramientas empleadas.

En este capítulo se va a hablar sobre las diferentes herramientas que han sido empleadas en el desarrollo del proyecto y las funciones y componentes con las que cuentan, así como en que partes han sido utilizadas para el presente proyecto.

3.1 Unity3D:

El proyecto ha sido desarrollado con el motor gráfico Unity3D en su versión 2018.1.1f1[18]. Unity 3D es un motor de desarrollo que el usuario puede obtener de manera gratuita con las funciones básicas que incluye el motor, pero si queremos sacar todo su potencial tenemos la posibilidad de utilizar la **versión PRO** que incluye funciones y herramientas especiales, que no eran necesarias para la elaboración de este proyecto.

Unity 3D es una de las **plataformas más completas** para desarrollar videojuegos. Permite el desarrollo de videojuegos para múltiples plataformas a partir de un único desarrollo.

Es junto a Unreal Engine, posiblemente la tecnología de mayor crecimiento en estos momentos para el desarrollo de videojuegos. Su principal limitación es el precio de su licencia completa, que puede alcanzar y superar los 4.500 $ por una licencia completa para una sola persona.

Unity 3D ha sido la plataforma elegida principalmente por su buen comportamiento y prestigio en el campo de los videojuegos en plataformas móviles como Android, en la cual se han creado e importado todos los elementos del proyecto: escenarios, modelos 3D, efectos de sonidos, luces, texturas, programación etc.

En las siguientes secciones se describen las características de este entorno, sus limitaciones y su estado actual.

3.1.1 Entorno de Unity3D

El editor de Unity 3D es uno de los más sencillos y potentes del mercado. Se divide en 5 vistas principales:

- **Proyecto**:

La ventana de Proyecto lista todos los elementos del proyecto y permite ordenar la aplicación. Desde esta ventana se pueden gestionar los diferentes **assets[19]:**

[18] **Notas de la versión 2018.1.1f1:** https://unity3d.com/es/unity/whatsnew/unity-2018.1.1
[19] **Asset**: Diferentes activos que pueden ser usados y modificados en el motor gráfico.

imágenes, **escenas**[20], **scripts**, **audios**, **prefabs**[21] y **texturas** que pertenezcan al proyecto *(Ver figura 11)*.

El panel izquierdo del navegador muestra la estructura de carpetas del proyecto como una lista **jerárquica**. Cuando una carpeta es seleccionada de una lista haciéndole click, su contenido se muestra en el panel a la derecha.

Los **assets individuales**[22] son mostrados como iconos que indican su tipo (script, material, subcarpeta, etc.). Los iconos se pueden redimensionar usando el deslizador que está en la parte inferior del panel; las diferentes posiciones del deslizador remplazan por diferentes tipos de previsualización.

El apartado Assets a la izquierda de la previsualización muestra el elemento actualmente seleccionado remarcado en azul, incluyendo la ruta completa al elemento si se está realizando una búsqueda.

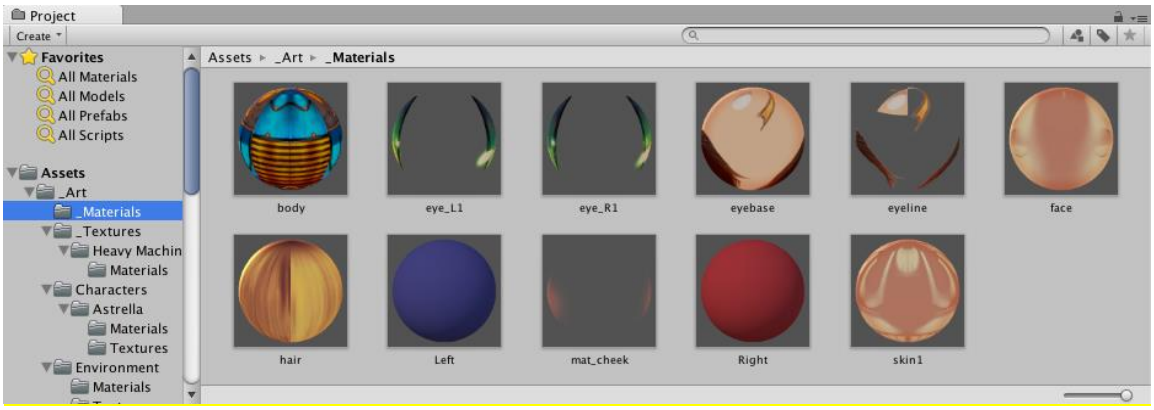


Figura 11 Aspecto de la ventana "Proyecto" de Unity

- **Inspector**: Muestra las propiedades de los elementos del proyecto. Sirve para modificar componentes o añadirlos, cambia texturas, añadir scripts, etc.

- **Jerarquía (Hierarchy)**: Lista jerárquica de los elementos de la escena*(Ver figura 12)*. La jerarquía contiene cada **GameObject**[23] de la escena que se esté mostrando en ese momento. Algunos de estos archivos de asset como modelos 3D, y otras son **Prefabs** que contienen una serie de **componentes invisibles** para el jugador como por ejemplo **colliders**. Desde la jerarquía es posible seleccionar los diferentes objetos para luego arrastrarlos uno dentro de otro para hacer uso de **Parenting**[24], consistente en hacer que un

[20] **Escena**: En Unity son los apartados que contienen todos los demás objetos del juego.
[21] **Prefab:** Tipo de asset de Unity que permite almacenar uno o varios GameObject completamente incluyendo sus componentes y propiedades.
[22] **Assets individuales:** Son aquellos que no pertenecen a ningún prefab.
[23] **GaeObject**: Objetos que componen una escena en Unity.
[24] **Parenting**: Concepto utilizado en Unity para asociar GameObjects en la jerarquía, consistente en establecer una relación objeto Padre-Hijo.

GameObject sea hijo de otro heredando alguna de las propiedades del GameObject padre como por ejemplo su movimiento y rotación, así como su estado. Podemos utilizar la flecha despegable del padre de un objeto para mostrar o esconder los hijos según sea necesario.

Por defecto los GameObjects se muestran en la ventana *Hierarchy* en el orden en el que son creados. Es posible reordenar los GameObjects arrastrándolos arriba o abajo, o haciéndolos hijos o padres mediante esta técnica de Parenting.

- **Escena**: Corresponde al diseño y maqueta del juego.
  . Cada escena representa un nivel o sección diferente del juego (portada, nivel 1, nivel 2, login, …). Para introducir elementos en la escena podemos arrastrar los assets desde el **Explorador** y editar sus variables desde el **Inspector***(Ver figura 13 y 14)***.** Desde la escena también podemos modificar ciertas **propiedades** como posición, tamaño y rotación de los elementos que contiene mediante las herramientas de edición rápida de la propia escena.

Figura 13: Herramientas de edición rápida de Unity.

- **Juego**: Aquí podemos visualizar el juego a distintas resoluciones. Es una vista **WYSIWYG**[25] de tu juego.*(Ver figura 14)*

[25] **WYSIWYG**: Acrónimo de "*What You See Is What You Get*" (en español, "lo que ves es lo que obtienes"), es una frase aplicada a los procesadores de texto y otros editores de texto con formato (como los editores de HTML) que permiten escribir un documento mostrando directamente el resultado final.

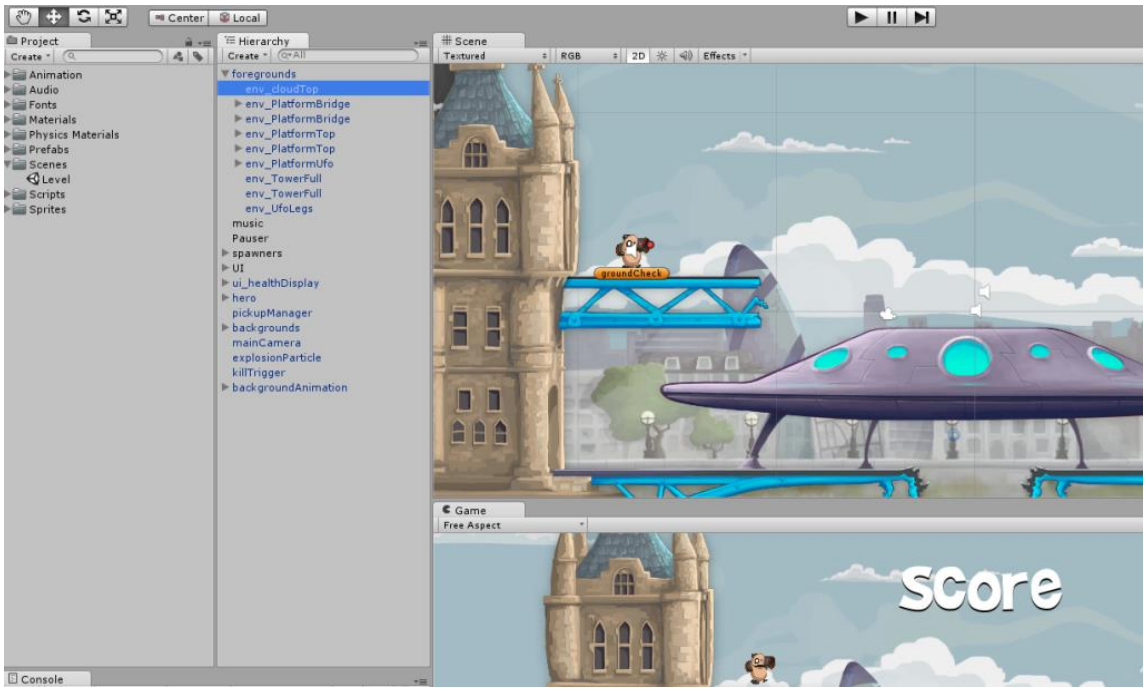


Figura 14: Ejemplo de entorno de trabajo en Unity

3.1.2 Desarrollo de juegos 3D y 2D

Inicialmente Unity era un entorno de desarrollo de juegos 2D. Se podían desarrollar entornos y juegos 2D ajustando los parámetros del juego para simular 2D (cámara ortográfica, texturas planas, etc). Pero contaba con grandes competidores y **algunos desarrolladores optaban por estas otras plataformas, como Cocos2D**, a priori mejor adaptadas a 2D.

A partir de una actualización Unity permitió desarrollos 2D de forma mucho más sencilla, incluyendo nuevos objetos y efectos para facilitar el desarrollo y el rendimiento del juego, como texturas 2D, efectos de física 2D o tipos de cámara específicos.

En el campo de los **juegos 3D**, existen muchos ejemplos de juegos desarrollados en Unity 3D. Actualmente más de 500 millones de usuarios juegan a juegos desarrollados con Unity 3D. De hecho la mayoría de los mejores juegos para Android y iOS 3D están hechos en Unity. En 2D el porcentaje es mucho menor, **siendo Cocos 2D todavía el motor más usado hasta la fecha**, ya que Unity 3D no había sido hasta ahora una opción robusta para el desarrollo de juegos 2D y por su precio, que es la principal barrera de Unity. Sin embargo algunas empresas como **Rovio** ya usan Unity 3D como motor 2D, habiendo desarrollado juegos de éxito como "***Bad Piggies".***

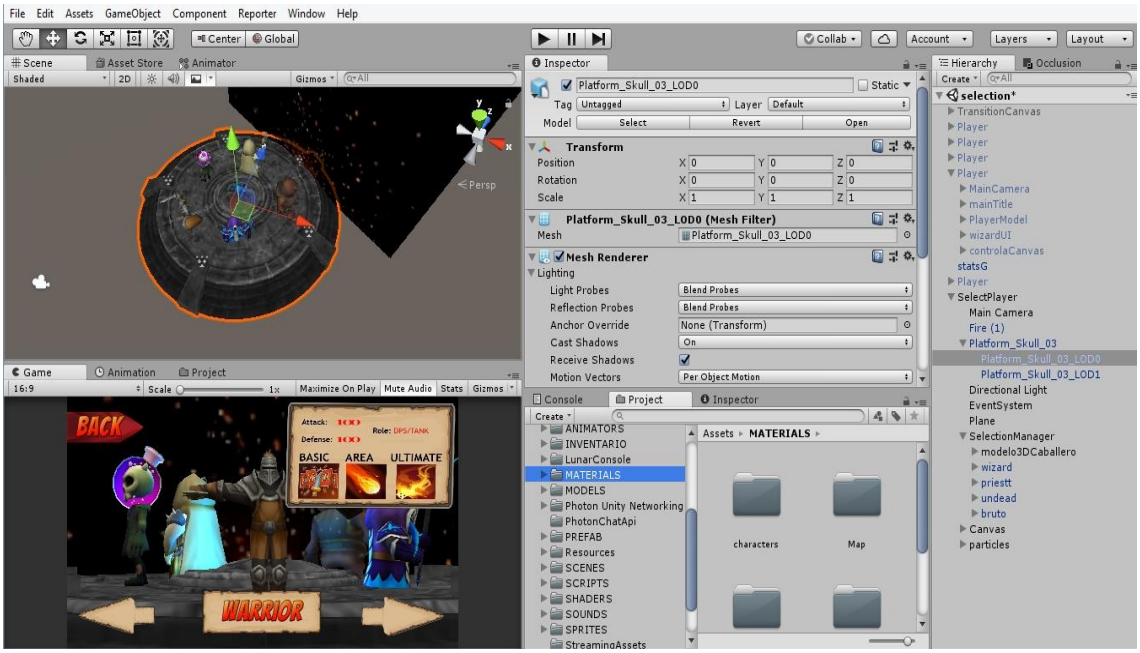


Figura 14: Captura del entorno de trabajo de Unity

3.1.3 Multiplataforma

Una de las mayores ventajas de Unity 3D es que permite desarrollar juegos para **muchas plataformas con muy poco trabajo extra** *(Ver figura 16)*:**,** siendo este trabajo en su mayoría compra de licencias para poder distribuir el videojuego. Unity permite importar los desarrollos de una plataforma a otra casi sin tener que hacer ajustes para adaptar el juego a esa nueva plataforma siempre y cuando estas cuenten con controles similares y no queramos usar funcionalidades específicas de alguna plataforma. Ejemplo de esto puede ser el desarrollo para diferentes consolas (***todas usan un mando con Joystick como controlador***), o entre plataformas móviles ***(el controlador es una pantalla táctil)*** *(Ver figura 15)*, ya que los controles que ambos emplean son similares.

Las plataformas que nos encontramos son prácticamente todas las implicadas de una forma u otra actualmente en el sector del videojuego y se pueden dividir en 4 grupos.

- **Consolas** (PlayStation, Xbox y Wii):

  Es posible desarrollar sin coste adicional juegos para **PS3/PS4, Wii U y Xbox 360**. A pesar de esto, todavía no es sencillo publicar juegos para estas consolas, al menos para empresas o equipos pequeños. Si bien Unity 3D no exige licencias adicionales, sí que son necesarios los equipos de desarrollo de las diferentes plataformas, que deben ser proporcionados por **Sony**, **Nintendo** y **Microsoft**, respectivamente.

  Actualmente los desarrollos para consola realizados con Unity son aquellos de menor presupuesto, enfocados a descargas desde las tiendas online de

las consolas. Para el desarrollo de juegos más potentes en estas consolas se usan otros motores, como **Unreal**[26].

- M**óviles y tabletas** (iOS, Android, Windows Phone y BlackBerry):

  Para Android, es necesario descargar e integrar el apk de Google para Android, así como hacer un pago único para una cuenta de desarrollador en Google Play de 25$ si queremos distribuir nuestro juego. iOS por su parte requiere un pago de 99$ al año a Apple en concepto de licencia de desarrollador.

Figura 15: Dispositivos móviles corriendo juegos desarrollados en Unity

  Además, se necesita un Mac para poder realizar la compilación final. Cuando se termine un proyecto para iOS, Unity no genera el fichero final sino un proyecto **XCode**[27] listo para compilar con el entorno XCode en Mac.

- **Ordenadores y navegador** (Linux, PC y Mac):

  Es posible compilar juegos para **PC, Mac y Linux**. En este caso el trabajo extra es prácticamente nulo, salvo en el caso de Mac que necesita un ordenador Mac para compilar el archivo XCode arrojado por Unity..

  Además, Unity 3D permite desarrollos para navegador y por tanto para Facebook. Todos los navegadores modernos como **Google Chrome**, **Firefox**, Internet Explorer o Safari permiten reproducir estos juegos.

  Anteriormente los juegos hechos con Unity 3D se reproducen con el reproductor oficial de Unity, Unity Web Player, que debe instalarse en el

[26] **Unreal Engine**: Es un motor de juego de PC y consolas creado por la compañía Epic Games
[27] **Xcode**: es un entorno de desarrollo integrado para macOS que contiene un conjunto de herramientas creadas por Apple destinadas al desarrollo de software para macOS, iOS, watchOS y tvOS.

navegador para poder ser usado. Finalmente Unity lanzó Unity 5 permitiendo la **reproducción de sus juegos con HTML 5**, haciendo mucho más sencillo la integración en páginas web.

Figura 16: Plataformas de desarrollo para Unity

3.1.4 Tarifas

Unity cuenta con varios planes de licencia *(Ver figura 17)*:

Licencia gratuita:

Cuenta con ciertas limitaciones La licencia gratuita incluye de forma obligada el logotipo de Unity en la primera pantalla de carga del juego y los proyectos bajo esta licencia solo pueden publicarse si la facturación total de la empresa encargada del desarrollo no supera los 100.000$ anuales.

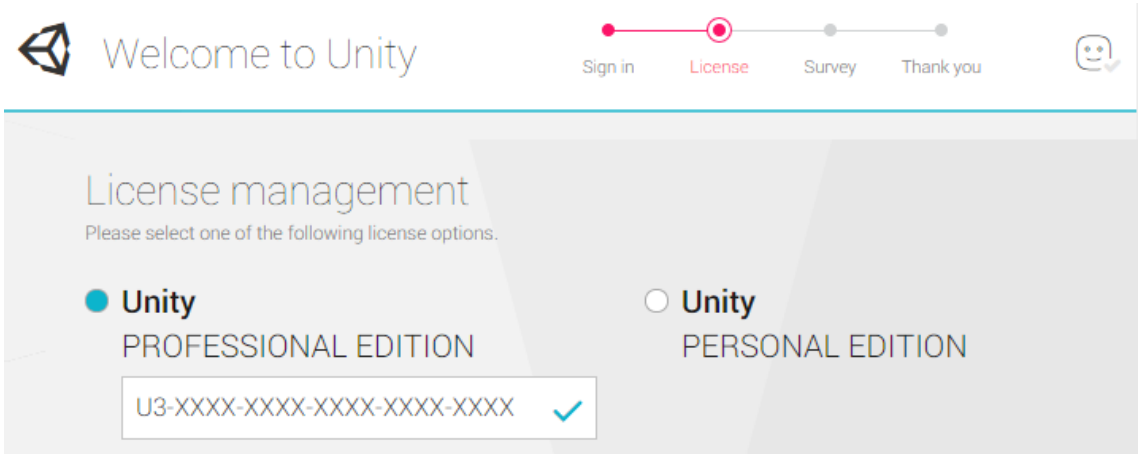


Figura 17: Pantalla de selección de Licencia de Unity

Licencia Profesional:

El precio de la licencia profesional **es de 1.500 $** más impuestos. Esta licencia permite el uso de todas las prestaciones de Unity Pro en hasta 2 ordenadores (de la misma persona). Las principales mejoras se encuentran en efectos, texturas y rendimiento 3D.

Esta licencia también es necesario si queremos hacer uso de Sockets .Net o para ciertas aplicaciones multijugador en tiempo real en plataformas móviles como Android o iOs.

Otras licencias como la licencia de equipo, **Team License**, o ciertos Assets también suponen un coste. La suma de estos costes es la principal barrera de esta tecnología, aunque **compensa enormemente por el tiempo de desarrollo ahorrado y la calidad del producto final.**

3.1.5 Asset Store

La Asset Store *(Ver figura 18)* es la **tienda de Unity.** En la Asset Store se pueden encontrar **modelos 3D de todo tipo**, efectos de sonido, sprites para los juegos, **animaciones**, **sistemas de partículas**, **extensiones** para el editor, y un largo etcétera.

Un tanto muy importante para Unity es esta tienda, cuenta con gran material audio visual que se actualiza constantemente permitiendo a los desarrolladores menos experimentados contar con gran cantidad de material para Utilizar en sus proyectos, consiguiendo una curva de dificultad y aprendizaje en el uso del motor mucho más sencilla.

Para mas detalles acerca de Unity, y su entorno de desarrollo consultar la siguiente fuente "https://www.yeeply.com/blog/desarrollo-de-juegos-con-unity-3d/ ", de la cual se han extraído ciertos fragmentos para explicar alguno de los puntos anteriormente desarrollados.

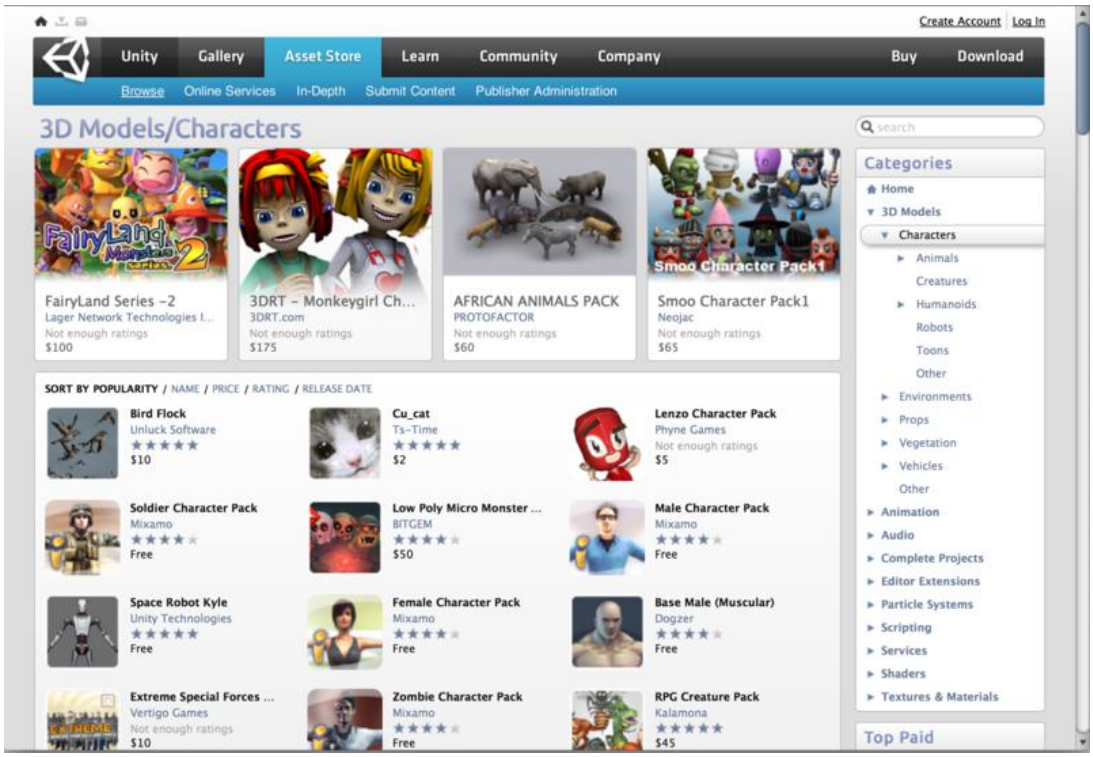


Figura 18: Captura de la Asset Store de Unity

3.2 Unreal Engine frente a Unity 3D

Tanto Unity 3D como Unreal Engine *(Ver figura 19)* ofrecen gran flexibilidad. Podemos hacer juegos para diferentes plataformas. En ambos como contamos con amplia documentación y abundantes tutoriales y en los dos casos tenemos acceso a una gran variedad de recursos creados por otros desarrolladores. Pero las similitudes se detienen ahí. A continuación, hablaremos de los pros y contras de cada motor, terminando con los motivos que propiciaron la elección por Unity.

Figura 19: Logos de Unreal Engine y Unity.

3.2.1 Pros de Unity

Respecto al desarrollo en el motor:

- **Versatilidad** para el prototipado.
- **Curva de aprendizaje** muy fácil tanto por la estructura de su editor como por el uso de un lenguaje de programación sencillo como puede ser C#.
- **Gran cantidad de documentación disponible** ya sean manuales y tutoriales.
- Ofrece una gran **variedad de plataformas** de publicación siendo el más completo en este aspecto.
- Gran **variedad y calidad de herramientas** para la realización de animaciones y de cinemáticas dentro de nuestro juego

Respecto a la comunidad:

- Cuenta con una **comunidad muy activa** que responde y plantea multitud de preguntas.
- **Gran variedad de contenido de terceros**, tanto para encontrar componentes, personajes, música y un largo etcétera en la tienda oficial (ya sean de pago o algunos gratuitos y de gran calidad) como para encontrar plugins de terceros que facilite la integración de características en nuestro juego como puede ser el uso de **Facebook, Google Play Services**, etc.

3.2.2 Contras de Unity

No todo es perfecto en el motor Unity, también existen ciertos inconvenientes que es necesario tener en cuenta antes de comenzar a trabajar con el motor:

- **Mala gestión de la memoria:** las librerías **.Net** pueden resultar lentas y para algunos casos la auto gestión de la basura, que es inevitable, no es eficiente.
- La **creación de terrenos** en Unity a veces se queda corta y no está muy optimizada.
- A lo largo del desarrollo Unity, se han ido generando diferentes versiones intermedias que en muchas ocasiones vienen con nuevos **bugs** para los que finalmente hay que volver atrás o esperar a que salga el parche correspondiente.

Por su parte **Unreal Engine:**

3.2.3 Pros de Unreal Engine

- **Control completo de su motor**, ya que se ofrece de forma gratuita su código abierto para que los usuarios puedan realizar mejoras.
- Una gran calidad y potencia en el apartado de la **iluminación global** y en el de la creación de **shaders y materiales**.
- Otro aspecto muy interesante y muy para tener en cuenta de Unreal Engine es la opción de programación **basada en nodos** y componentes, que facilita el trabajo a los no programadores y hace el prototipado muy rápido. Este sistema de **blueprints**[28] es muy parecido a crear diagramas de flujo y aporta a este motor gráfico un herramienta muy versátil y potente.

3.2.4 Contras de Unreal Engine

- El principal problema que nos podemos encontrar al empezar a acercarnos a Unreal Engine es la **dificultad de la curva de aprendizaje**.
- Algunos de los conceptos que podemos encontrar en Unreal Engine son confusos como la estructura de **Actors**[29].

Tras esta comparación y teniendo en cuenta el proyecto objetivo, el motor elegido para su desarrollo ha sido Unity.[30]

---

[28] **Blueprints Visual Scripting:** Sistema de Unreal Engine para hacer scripting basado en una interfaz de nodos unidos para crear elementos de juego.

[29] **Actors**: Nombre que se da en Unreal Engine a cualquier objeto en el juego.

[30] **Fuente**: https://www.deustoformacion.com/blog/diseno-produccion-audiovisual/pros-contras-programar-unity-vs-unreal-engine

3.3 Photon Unity Networking (PUN)

Photon*(Ver figura 20)* es un marco de desarrollo de juegos multijugador en tiempo real que tiene servicios de servidor en la nube. Una ventaja de Photon es que no requiere hosting, por lo que el jugador que creó la sala puede abandonar el juego sin causar bloqueos en los clientes. La comunicación rápida y (opcionalmente) confiable se realiza a través de servidores de Photon dedicados, por lo que los clientes no necesitan conectarse uno a uno. Además, su código fuente está disponible para solucionar cualquier problema que pueda tener.

Figura 20: Cabecera de Photon Unity Networking

Antes de nada, debemos instalar PUN. Podemos encontrarlo en la Asset Store de Unity.

PUN exporta básicamente a todas las plataformas compatibles con Unity y viene en dos formas: La versión de **prueba gratuita** permite hasta 20 jugadores simultáneos conectados, en el servidor, ya sea en la misma partida o en diferentes salas. Esta es la opción utilizada en este proyecto y el número de jugadores que permitimos en una misma sala será de 4. ***(Para instalar PUN véase Anexo).***

3.4 Herramientas de programación y lenguajes empleados.

En cuanto a la programación, el motor gráfico Unity cuenta por defecto con soporte para **MonoDevelop**, la herramienta de programación por defecto de Unity. Sin embargo, podemos utilizar cualquier otro compatible, en este proyecto se ha empleado Visual Studio, por comodidad y conocimiento de su funcionamiento.

3.4.1 Visual Studio

Microsoft Visual Studio es un entorno de desarrollo integrado para sistemas operativos **Windows**. Soporta múltiples lenguajes de programación, tales como C++, **C#**, Visual Basic .NET, F#, Java, Python, Ruby y PHP, al igual que entornos de desarrollo web, como ASP.NET

Visual Studio permite a los desarrolladores crear sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET. Además permite crear aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, dispositivos móviles, dispositivos embebidos y consolas, entre otros.

3.4.2 C#

Unity controla los diferentes comportamientos que se suceden en los videojuegos desarrollados en él con scripts asociados a los GameObjects. Los scripts pueden escribirse en cualquiera de los lenguajes que Unity soporta de forma nativa: **C#**, **UnityScript** y **Boo**.

En este proyecto se emplea C# s por varias razones*(Ver figura 21)*:

- Es el lenguaje que prima el propio motor de juegos **Unity**, ya que le da un **soporte especial** y basa toda su **documentación** en él. Además este lenguaje se ha utilizado durante el transcurso de la carrera.
- Es **el más utilizado**, con mucha diferencia (ver gráfico abajo), por la **comunidad de usuarios de Unity**, lo que facilita luego la obtención de información y el entendimiento entre la mayor parte de los usuarios de Unity.
- Su similitud con Java, lo convierte en muy fácil de utilizar por aquellos que trabajan en el entorno Android.

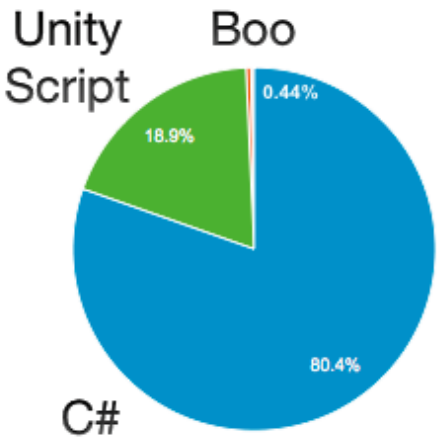


Figura 21: Lenguajes usados en Unity

3.4.3 JSON

Acrónimo de ***JavaScript Object Notation,*** es un formato de texto ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript. Actualmente se considera un formato de lenguaje independiente. debido a su adopción como alternativa a XML.

Una de las supuestas ventajas de JSON es que es mucho más sencillo escribir un analizador sintáctico para JSON que para XML como formato de intercambio de datos

En este proyecto JSON se ha empleado para mostrar las características de los diferentes objetos del inventario, todos estos datos se encuentran identificados por un ID único que será manejado por JSON.

3.5 Herramientas de edición de imagen:

Estas herramientas son: **Adobe Photoshop:** es un editor de imágenes usado principalmente para el **retoque de fotografías** y gráficos. Es líder mundial del mercado de las aplicaciones de edición de imágenes y domina este sector de tal manera que su nombre es ampliamente empleado como sinónimo para la edición de imágenes en genera**.**

Adobe Flash: un programa que se usa normalmente para crear animaciones para páginas de internet, aunque las posibilidades son muchas: crear dibujos animados, presentaciones multimedia, aplicaciones móviles o de escritorio.

Estas aplicaciones se han utilizado para realizar diferentes montajes con los **sprites** [31]que se ven en el juego*(Ver figura 22)*. Por un lado, **Photoshop** ha servido para darle una imagen más atractiva al inventario por poner un ejemplo, así como para unir todos los ítems en una sola imagen para después poder editarlos individualmente dentro de Unity.

Figura 22: Algunos ejemplos de edición con Photoshop para el inventario

Por otro lado, **Adobe Flash** se ha utilizado principalmente en el logo del juego *(Ver figura 23)*, así como para pequeños detalles en imágenes.

Figura 23:Miniatura del. Apk del videojuego

[31] **Sprites**: Son esencialmente unas texturas estándar o en algunos casos, texturas especiales combinadas por temas de eficiencia y conveniencia durante el desarrollo.

3.6 Herramientas de Assets

Unity es un motor gráfico muy potente, que hace uso de una gran cantidad y variedad de componentes para integrar al videojuego, tales como motores de físicas, animaciones, reproducción de audio, modelaje 3D, etc. A continuación, se destacarán algunas de las herramientas que se han utilizado para modificar o crear alguno de los componentes utilizados en el juego.

3.6.1 Audacity (Audio)

Audacity es una aplicación de software libre que se puede usar para grabación y **edición de audio.**

Entre sus principales características:

- Grabación de audio en tiempo real.
- Agregar efectos al sonido (eco, inversión, tono, etc).
- Edición archivos de audio tipo Ogg Vorbis, MP3, WAV, etc.
- Conversión entre formatos de tipo audio.
- Permite usar plug-ins para aumentar sus funcionalidades.

Alguno de los efectos de sonido, así como clips de música empleados en el juego han sido editados usando esta herramienta.

3.6.2 Mixamo (Animaciones):

Mixamo es una empresa de tecnología de gráficos 3D perteneciente a **Adobe**. Con sede en San Francisco, la compañía desarrolla y vende servicios basados en la web para la animación de personajes en 3D. Las tecnologías de Mixamo utilizan métodos de aprendizaje automático para automatizar los pasos del proceso de animación de personajes, incluido el modelado 3D.

Mixamo consiste en un servicio en línea que incluyen una tienda de animación con modelos 3D descargables y secuencias de animación *(Ver figura 24)*. Las animaciones presentes en Mixamo se desarrollan utilizan la captura de movimiento. Todas sus animaciones funcionan con personajes creados en **Fuse**[32] y / o equipados con **AutoRigger**[33] de Mixamo. Además de la galería de personajes creados en Adobe Fuse, el sitio web permite subir modelos 3D externos a la página y añadirle animaciones.

Debido al estilo grafico que presenta nuestro juego y el que presentan los modelos existentes en la página hemos seguido este segundo camino.

Además de dotarnos de animaciones, Mixamo se encarga mediante el AutoRigger comentado anteriormente de dotar al personaje de un esqueleto, sobre esos "huesos" predefinidos se ejecutarán las animaciones.

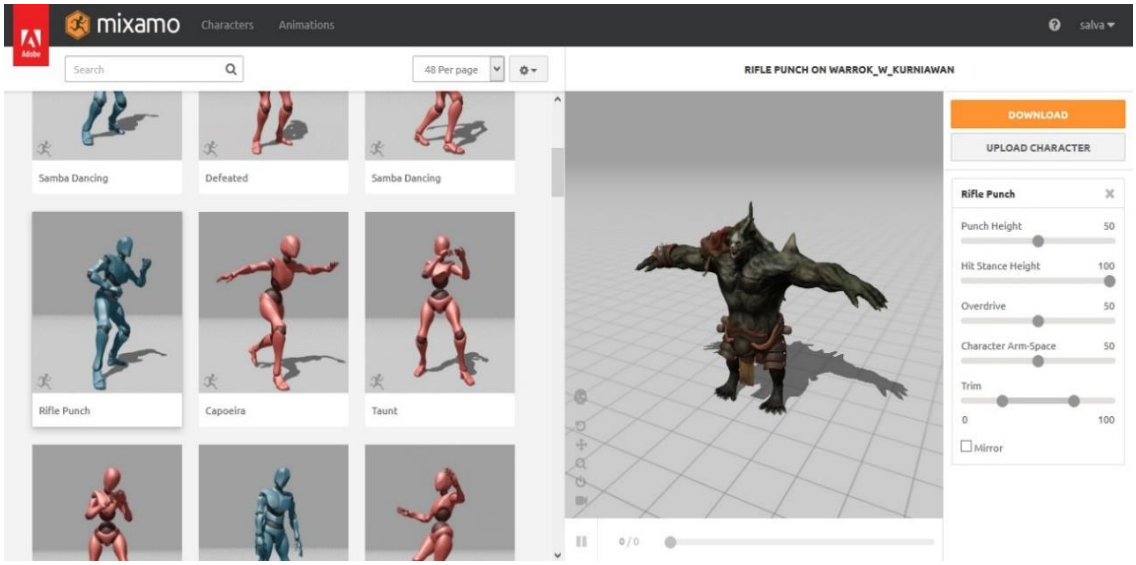


Figura 24: Espacio de trabajo de Mixamo.

[32] **Fuse**: Aplicación similar a Mixamo encargada del desarrollo asistido de modelos 3D de personajes.
[33] **Autorriger**: Proceso automatizado consistente en añadir "Huesos" a los modelos 3D con el fin de que puedan ser asociados a una animación.

4 Características del juego desarrollado.

En este apartado hablaremos de “Mr. Mahuhenhaim” *(Vease logo en figura 25)* el nombre del videojuego que ocupará este proyecto. Mr. Mahuhenhaim pertenece al estilo RPG, un género rico en opciones de jugabilidad de las cuales hablaremos. En este apartado se describirá brevemente su “gameplay”, y se comparará el estándar de jugabilidad en la industria del videojuego como a la jugabilidad que presenta este proyecto.

El diseño de las características del juego incluye:

- Determinar el argumento.
- Determinar los modos de juego y características principales con las que cuenta la app.
- Explicar el funcionamiento y desarrollo de una partida.
- Reforzar mediante un storyboard la explicación de este funcionamiento.
- Exponer los diferentes puntos de la jugabilidad que debemos buscar en un videojuego y enfrentarlos a los puntos de este videojuego concreto que atacan a cada uno de ellos.

En el siguiente punto se expondrán en mayor detalle los elementos anteriormente comentados, entrando en materia de programación, decisiones de diseño y assets empleados.

Figura 25:: Logo del videojuego “Mr. Mahuhenhaim”

4.1 Argumento

Un poder mágico se ha desatado sobre el reino de Mahuhenhaim liberando todo tipo de bestias y peligrosos seres para los aldeanos. Éste será el momento de que los héroes de todo el reino se agrupen para acabar con este mal

encabezados por “*Mr. Mahuhenhaim”,* un misterioso caballero al que nunca nadie ha visto sin su yelmo. ¿Podrás liberar el reino de Mahuhenhaim?

4.2 Características de juego principales

El desarrollo de este videojuego ha tomado 3 puntos importantes, el modo “**Un solo jugador**” (*single player*), el “**modo multijugador**” (*Multiplayer)* y el “**inventario/tienda**” (*Shop*). Éste último común para ambos modos, y base para el progreso del jugador en el juego.

4.2.1 Un solo jugador

Para este modo se ha buscado darle al jugador la libertad de jugar solo, con una dificultad razonable y con la posibilidad de no tener que estar conectado a la red. En esta modalidad el jugador elegirá un personaje y se enfrentará en solitario a los peligros que le esperan en el reino de Mahuhenhaim.

4.2.2 Multijugador

Para este modo será obligatorio contar con una conexión a internet. Aquí el jugador podrá unirse a otros jugadores en línea. Cada jugador selecciona un personaje y es lanzado al mundo del juego. Los personajes aquí tendrán que trabajar en equipo para lograr la victoria utilizando las habilidades únicas de cada personaje.

4.2.3 Tienda/Inventario

La tienda corresponde al inventario, el equipamiento y el mercader, la piedra angular de todo juego RPG. Aquí el jugador podrá comprar objetos para equipar a sus personajes y mejorar con ello sus estadísticas. También podrá adquirir consumibles para usar en el transcurso de las partidas, mejorando así su supervivencia. A su vez podrá comerciar con sus objetos en el transcurso del juego.

4.3 Desarrollo de una partida.

El objetivo del juego es muy simple: avanzar desde un punto A hacia un punto B *(Ver figura 26)* final en el cual tendremos que derrotar a un jefe de nivel para obtener una llave y así completar el nivel. Este proyecto cuenta con un único nivel, pero no se descarta ampliar la experiencia jugable a más niveles con diferentes desafíos.

El jugador siempre comenzará en una misma zona predefinida, a la que llamaremos "**Bosque (1)**". Aquí los personajes tendrán dos caminos, uno bloqueado por una neblina mágica que impide el paso y otro tras las puertas de un castillo a la que llamaremos "**Poblado (2)**". Los jugadores deberán abrirse paso a través del poblado y los enemigos para llegar a un jefe de zona que al ser eliminado dejará caer una llave.

Con esta llave los jugadores podrán desbloquear la neblina comentada anteriormente que les bloqueaba el paso y pasar a la "**Guarida del Troll (3)**", donde la mecánica se repetirá. El camino se verá bloqueado hasta conseguir la llave de este enemigo.

Tras pasar este nuevo obstáculo llegaremos al "**Bosque Encantado (4)**" donde nos espera el jefe de nivel, el cual es notablemente más complicado que los demás jefes de zona. Cuando los jugadores logren eliminar a este jefe, conseguirán una llave dorada, que al ser tocada dará por finalizado el nivel y nos mostrará las estadísticas propias en un panel, para que, pasados unos segundos, nos redirija al menú de un jugador.

A continuación, veremos el mapa, así como los puntos de interés que lo componen.

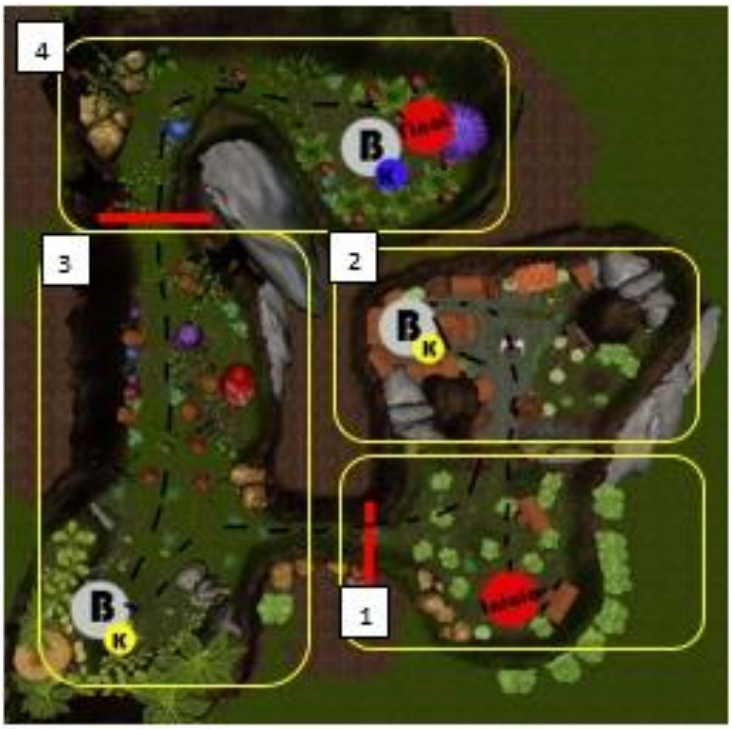


Tabla 1: Leyenda del mapa	
	**Punto de inicio de la partida:** al comenzar una partida todos los jugadores aparecerán aquí. Además, si alguno de ellos es eliminado en el transcurso de la partida reaparecerá en este mismo lugar.
	**Camino**: Líneas imaginarias que informan en este mapa del trayecto aproximado que los jugadores deberán seguir para progresar.
	**Jefe:** Enemigo de la categoría “jefe de zona”, el cual al ser eliminado dejará caer la llave con la que podremos continuar en el juego.
	**Llaves:** objetos arrojados por los jefes al ser derrotados. Estas llaves pueden **desbloquear caminos (K amarilla**) o desbloquear el **fin de la partida (K azul**)
	**Barrera mágica:** Barrera que nos bloqueará el paso. Necesitaremos una llave mágica, las cuales dejan caer los jefes de zona, para poder eliminarla.
	**Punto final de la partida:** Aquí se encuentra el ultimo jefe de zona, al derrotarlo y recoger su llave, la partida termina.

4.4 StoryBoard

A continuación, en la siguiente sucesión de imágenes se detallará brevemente el transcurso de una sesión de juego normal. Esta sucesión irá acompañada primero de una breve descripción de cada una de las funcionalidades que busca representar la imagen, seguido por la imagen en cuestión.

Pantalla de título:

Al arrancar el juego aparecerá una pantalla del título con la opción "**Play**" *(Ver figura 27).* Al pulsarla entraremos al menú principal del juego.

Figura 27: Pantalla del título (Storyboard)

Menú del juego:

El menú del juego nos presenta las tres opciones disponibles: el modo "***Single player***" para un solo jugador; el "***Multiplayer***", en línea con otros jugadores; y la tienda/inventario "***Shop***" *(Ver figura 28)*.

En este apartado también podremos modificar la cuenta con la que hemos iniciado sesión haciendo click en "***options***".

Llegados a este punto el jugador puede jugar directamente. Es más, su primera partida será sin pasar por la tienda ya que no cuenta con oro para comprar. Vamos a explicar el proceso de preparación diseñado para el juego, para proseguir, a continuación, con una partida cualquiera.

Figura 28: : Menú de inicio (Storyboard)

Inventario/Tienda

Si seleccionamos tienda tendremos que seleccionar un personaje para ver y editar su inventario exclusivamente *(Ver figura 29)*.

Figura 29: Selección de inventario (Storyboard)

Una vez seleccionado el inventario *(Ver figura 30),* si disponemos de dinero podremos comprar objetos de diferentes tipo, ya sea consumibles para usar durante la partida o equipamiento para mejorar nuestras habilidades.

Figura 30: Inventario de "Warrior" (Storyboard)

Será importante comprobar qué ganamos con cada objeto. Haciendo click en su icono desplegaremos un panel que nos mostrará el nombre del objeto, así como las características que nos otorga si lo equipamos *(Ver figura 31)*.

Figura 31: Panel de información de objetos (Storyboard)

Desarrollo de una partida

Independientemente del modo que seleccionemos, "***SinglePlayer***" o "***Multiplayer***" seremos redirigidos a la pantalla de selección de personaje *(Ver figura 32).* Aquí podremos ver además su modelo 3D, sus características totales (las características base más las que nos proporcionan los objetos equipados)

Figura 32: Pantalla de selección de personaje (Storyboard)

Ya en la partida *(Ver figura 33)* nos moveremos por el mundo mediante un **Joystick**[34] virtual y atacaremos con los botones representados en pantalla. Nos toparemos con enemigos que tendremos que eliminar para poder continuar.

Figura 33: Captura dentro de una partida (Storyboard)

[34] **Joystick**: es un periférico de entrada que consiste en una palanca que gira sobre una base e informa su ángulo o dirección al dispositivo que está controlando.

Si recibimos daño o queremos obtener **maná** [35]rápidamente podemos abrir el inventario de consumibles *(Ver figura 34).* En éste podremos encontrar los objetos de esta categoría que hayamos comprado en la tienda.

Figura 34 : Inventario de consumibles dentro de una partida (Storyboard)

Durante el juego encontraremos zonas cortadas por magias *(Ver figura 35).* Cuando nos topemos con ellas tendremos que buscar la llave, que posee el jefe de zona. Al derrotarlo y volver a este punto la barrera desaparecerá.

Figura 35: Portales bloqueando el camino (Storyboard)

[35] **Maná**: Nombre que se da en juegos de fantasía a los puntos necesarios para usar una habilidad.

Además de los jefes, los enemigos básicos o especiales soltaran objetos *(Ver figura 36).* Estos se consumen al instante y no se almacenan. Los objetos aparecen de forma aleatoria al derrotar a los enemigos y pueden ser desde consumibles a oro.

Figura 36: Los enemigos básicos pueden dejan caer objetos al ser eliminados (Storyboard)

Cuando encontremos un jefe de zona *(Ver figura 37)* encontraremos un mayor desafío que con un enemigo normal o especial. Al derrotarlo dejará caer una llave. Esta llave puede ser de **plata** para abrir portales que bloquean el camino o si es el jefe final soltará una **llave dorada.**

Cuando se coge la llave dorada finaliza la partida.

Figura 37 : Enfrentamiento con jefe de zona (Storyboard)

Es posible que durante la partida nos eliminen alguna vez. Si esto ocurre seremos transportados al punto inicial del mapa y recibiremos una penalización *(Ver figura 38)*.

Figura 38: Pantalla de muerte (Storyboard)

Al obtener la llave dorada la partida finalizará y en pantalla se mostrarán nuestros datos en esa partida *(Ver figura 39)*, se guardará todo y seremos devueltos al menú principal.

Figura 39: Estadísticas finales (Storyboard)

4.5 Jugabilidad

La jugabilidad es un factor muy importante en la creación de videojuegos, ya que es decisiva para su aceptación. Por ese motivo, en este proyecto se han tenido en cuenta los siguientes factores relacionados con la jugabilidad.

- **Satisfacción:** es el agrado del jugador ante el videojuego en algunos aspectos concretos de éste. La satisfacción es un atributo un tanto complejo ya que dependerá de los gustos del jugador.

Una buena implementación, fluidez, y ausencia de **bugs**[36] en el proyecto contribuirán a una buena satisfacción del usuario

- **Aprendizaje:** es la facilidad para dominar la mecánica del juego, es decir objetivos reglas y formas de interactuar con el juego. El aprendizaje debe ser secuencial, subiendo progresivamente de dificultad para no frustrar al jugador con una elevada dificultad ni aburrirlo con un juego demasiado fácil.

El juego explotará el aprendizaje en función de la posición de los diferentes tipos de enemigos, comenzando en la primera zona con aquellos más débiles, subiendo en dificultad conforme avancemos hasta llegar al jefe. A su vez, mediante el panel de información, el jugador obtendrá pistas de que está ocurriendo y qué debe hacer.

- **Inmersión:** es la capacidad de sumergir al jugador en el videojuego. Esta característica está estrechamente relacionada con el aprendizaje. El jugador sentirá que conforme aprende se va metiendo más en el personaje, presentándose desafíos que requieran de mayor atención y dificultad para ser superados.

- **Motivación:** es la característica que mueve al jugador a seguir jugando y persistir hasta conseguir un objetivo.

Este punto se ve reflejado en la búsqueda del aprendizaje de las mecánicas del juego, y en ver cómo su personaje se va haciendo más poderoso con los objetos que adquiere. La ecuación se convierte en: cuanto más juegas, más oro obtienes, más objetos compras y más poderoso te haces.

[36] **Bugs**: Errores inesperados dados durante el juego, que depende de una combinación concreta de factores para hacerse visibles.

- **Socialización:** es el factor de los juegos que fomentan la experiencia en grupo, que crea vínculos de compañerismo mediante las relaciones que se establecen con otros jugadores.

El sistema de progresión del juego y la posibilidad de comparar tu estado y logros con el de amigos u otros jugadores añade una motivación extra para ser mejor y estar un paso por delante en el juego.

5 Elementos del proyecto.

En este capítulo entraremos en detalle de algunos elementos implementados en el proyecto. El proyecto ha llevado un **desarrollo escalado**, separado por diferentes elementos diferenciados que componen el juego. Antes de comenzar con los elementos del juego y con el fin de mostrar una vista general de las diferentes **implementaciones** y el peso de cada una en cuanto a cantidad de programación y uso de assets se refiere. Se comenzará explicando en qué consisten los assets, de que elementos se componen y un ejemplo práctico de su uso en combinación con la **programación** necesaria para darles funcionamiento.

Posteriormente mediante unas **tablas** expondré la **carga de trabajo** del proyecto. En la primera de ellas *(Ver tabla 1)* se recoge para cada característica del juego su peso en uso de assets y programación y en la segunda tabla *(Ver tabla 2)* se presenta un resumen de los scripts que se encargan de cada característica tratada en la tabla anterior.

Al finalizar este punto y ya con una visión más clara del funcionamiento del motor gráfico Unity pasaremos a las implementaciones con las que cuenta el videojuego desarrollado.

5.1 Uso de programación y assets en Unity

Unity es un motor gráfico que integra gran cantidad de lógica de programación contenida en Assets. En el desarrollo de videojuegos se entiende por asset cada uno de los elementos que componen el juego (animaciones, modelos, IA, sonidos, etc).

El control de los diferentes assets y componentes se maneja a través de scripts. Estos componentes consistirán en algunos casos en una apariencia gráfica y visual de cara al usuario, y en otros serán los encargados de obtener un "**Input**"[37] para poder ser procesados por los diferentes scripts. Algunos de los componentes[38] más usados e importantes serán comentados a continuación.

5.1.1 El *GameObject* en Unity:

Los GameObjects son objetos fundamentales en Unity que representan personajes, componentes, y el escenario entre otros. Estos no logran nada por sí mismos, pero funcionan como contenedores para componentes, que implementan la verdadera funcionalidad. *(Ver figura 40)*

[37] **Input**: Valor de entrada obtenido por un componente o script

[38] ***Fuente***: Fragmentos de algunas de estas definiciones han sido tomadas de: https://docs.unity3d.com/Manual/index.html

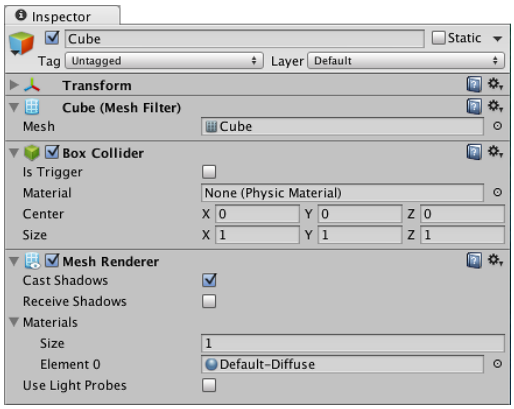


Figura 40: Ejemplo de GameObject Cube con algunos componentes

Un GameObject siempre tiene el componente **Transform**[39] adjunto (para representar la posición y orientación). Los otros componentes que le dan al objeto su funcionalidad pueden ser agregados del menú "***Component***" *(Ver figura 41)* del editor o desde un script. También hay muchos objetos útiles **preconstruidos**[40].

5.1.2 Los componentes.

Los **Components** constituyen la funcionalidad de los objetos y comportamientos de un juego. Son las piezas funcionales de cada **GameObject,** el cual es un contenedor para muchos componentes distintos. Estos pueden ser añadidos desde el menú "***Add Component".***

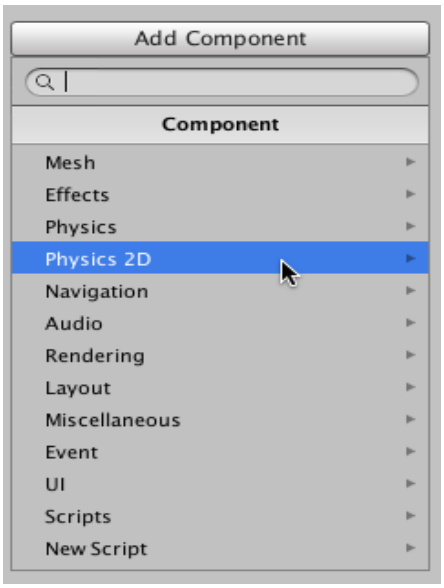


Figura 41: Menú de selección de componentes.

Uno de los grandes aspectos de los *components* es su flexibilidad. Cuando se adjunta un *component* a un GameObject, hay diferentes valores o ***Properties*** en

[39] **Transform**: Componente de Unity encargado de localizar en unas coordenadas concretas los objetos en el mundo.

[40] **Objetos preconstruidos:** Objetos básicos ofrecidos por el editor, tales como figuras geométricas simples en 3D, luces, o cámaras.

el *component* que se pueden ajustar en el editor mientras se construye el juego, o por scripts cuando esté corriendo el videojuego. *(Ver figura 42)*

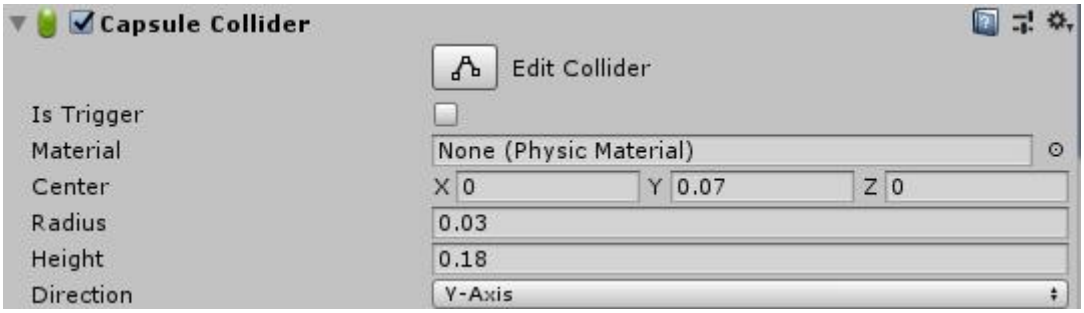


Figura 42: Ejemplo de componente.

A continuación, se habla sobre cada uno de los componentes asignados de los objetos en este proyecto, que se pueden agrupar en función de para qué estén destinados en físicas, animaciones, sonido, etc.

5.1.3 Animaciones:

El sistema de animación de Unity está basado en el concepto de ***Animation Clips***, estos contienen información acerca de cómo ciertos objetos cambian su posición, rotación, u otras propiedades en el tiempo. Cada clip puede ser pensado como una sola grabación lineal.

Los *Animation Clips* (Clips de animación) son organizados en un sistema con una estructura similar a la del diagrama de flujo llamada "***Animator Controller"*** *(Ver figura 43)***.** El ***Animator Controller*** funciona como una "***State Machine***" que mantiene un seguimiento de qué clip se está reproduciéndose actualmente, y cuando hay que cambiar de una animación a otra.

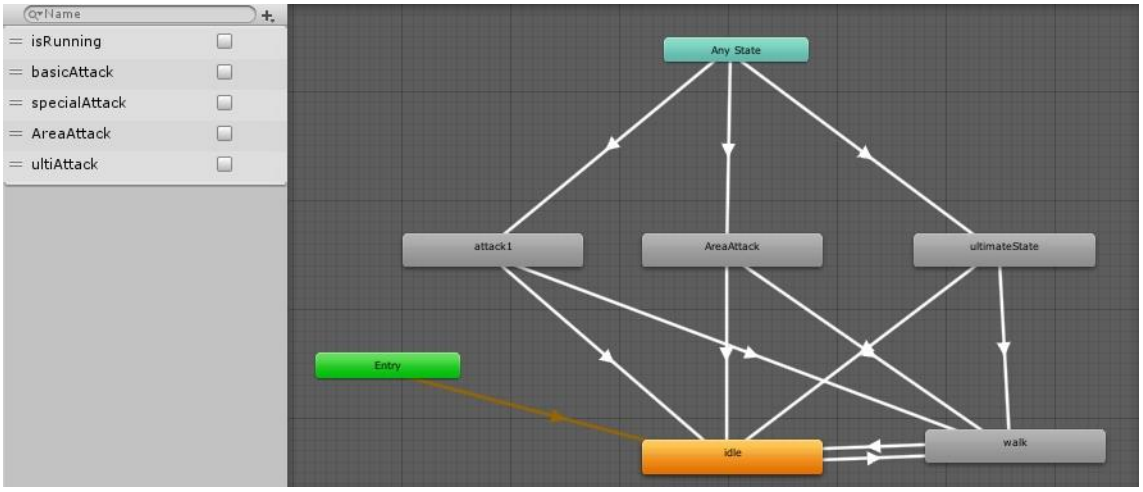


Figura 43: Animator Controller de los personajes

Un problema que se presentó durante las pruebas de las animaciones fue que estas se ejecutaban de forma muy errática y sin orden cada vez que pulsábamos un botón de ataque, lo que ocasionaba que se generaran ataques a destiempo

de la animación. Para solucionar este problema se han desarrollado una serie de temporizadores, de forma que sirva de ajuste para la velocidad de la animación frente a la velocidad del jugador al pulsar el botón.

5.1.4 Físicas:

Mediante las físicas podemos dotar a los modelos 3D de un cuerpo físico, y permitir que este reaccione al entorno o a otros elementos distribuidos por el escenario. Estas se pueden controlar principalmente mediante el **RigidBody**, y los **Colliders.**

- **Rigidbodies**

Los **Rigidbodies** *(Ver figura 44)* permite a un **GameObjects** actuar bajo el control de la física. Cualquier GameObject debe contener un Rigidbody para ser influenciado por la gravedad, actue bajo fuerzas agregadas vía scripting, o interactue con otros objetos a través del motor de física **NVIDIA Physx.**

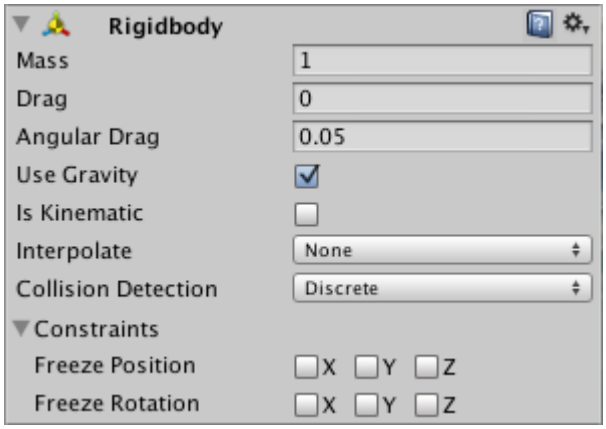


Figura 44: Componente Rigidbody.

- **Colliders**

Los componentes **Collider** definen la forma de un objeto a efectos de detectar colisiones físicas. Un collider, es invisible, no necesita estar con la misma forma exacta que el **mesh**[41] del objeto y, de hecho, una aproximación a menudo es más eficiente e indistinguible en el juego.

Cuando los colliders interactúan, sus superficies necesitan simular las propiedades del material que representan. Aunque la figura de los colliders no

[41] **Mesh**: Es la interfaz script básica a la geometría de un objeto. Utiliza arreglos para representar los vértices, triángulos, normales, y coordenadas de texturas.

es deformada durante las colisiones, su fricción y rebote pueden ser configurados utilizando **Physics Materials**[42].

El sistema de scripting detecta cuando se ocasionan colisiones e instancia acciones utilizando funciones propias de Unity para las físicas de colisiones, como, por ejemplo, la función **OnCollisionEnter**. Sin embargo, también se puede utilizar el motor de física simplemente para detectar cuando un collider entra al espacio de otro sin ocasionar una colisión. Un collider configurado como **Trigger** (utilizando la propiedad **Is Trigger**) no se comporta como un objeto sólido y simplemente le permitirá a otros colliders pasar a través de él.

5.1.5 Scripts

A su vez, los scripts pueden estar o no asociados a estos GameObjects *(Ver figura 45)* si van a necesitar algún tipo de **Input**: ya sea de estos componentes, o un Input desde el propio editor.

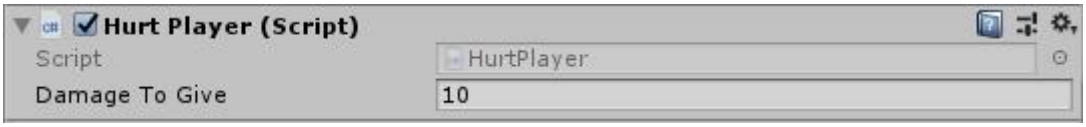


Figura 45: Script asociado a un gameObject con un input por editor

Podemos englobar la idea del uso de GameObjects y Componentes con una analogía:

El **gameObject** podría ser un cuerpo físico, una **cáscara sin vida**, los **componentes de assets** los diferentes **aspectos físicos** del cuerpo, como pueden ser el peso; **así como los sentidos**, vista, tacto, etc. Y finalmente los **Scripts** asociados a éste los diferentes **órganos vitales** del cuerpo, el corazón, cerebro, etc**, los cuales dan vida a todo lo demás.**

[42] **Physics Materials**: Componentes asociados a las texturas de un objeto 3D que le da propiedades al tacto tales como deslizamiento, rigidez, etc

5.2 Caso práctico: Desarrollo de un personaje.

Vamos a profundizar a continuación en el desarrollo de un personaje desde cero con el fin de tener una visión más cercana y centrada en este proyecto en cuanto al manejo de esta relación entre GameObjects y Components.

Cabe destacar que el proceso de desarrollo que se ha seguido en el proyecto **para cierta implementación** ha sido, primero **preparar los assets**, **disponerlos en el editor** y completarlos con la **programación.** En este caso práctico se va a agrupar primero la parte de preparación de assets que componen las distintas funcionalidades del personaje para finalizar comentando la adición de la lógica programada mediante scripts con el fin de mejorar la comprensión de los conceptos por el lector, no correspondiendo el desarrollo mostrado a continuación con ningún elemento del proyecto.

El primer paso será obtener un modelo 3D para este personaje junto con diferentes animaciones para el personaje obtenidas con **Mixamo**.

Puesto que nuestro personaje va a contener diferentes elementos asociados a él, modelo3D, interfaz de usuario, etc, el siguiente paso será crear un GameObject padre que contenga a todos los demás*(Ver figura 46)*. Una vez creado éste, colocaremos el modelo 3D del personaje como objeto hijo y a este le añadiremos los primeros componentes que necesitaremos para empezar a programar su comportamiento. Colocaremos un **RigidBody**, para establecer las propiedades físicas como el peso o el efecto de la gravedad, seguido por un **Collider**, para detectar cuando el personaje ha impactado con algo y comunicarlo a nuestros scripts. Finalizaremos con la adición de componentes del editor introduciendo un **AnimatorController**, estableciendo la máquina de estados con las diferentes animaciones, que programaremos más adelante.

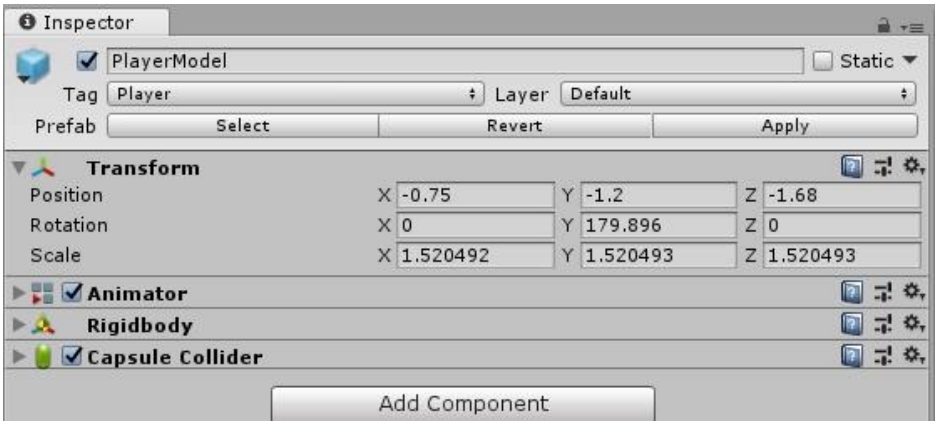


Figura 46: Estado del gameObject perteneciente al modelo de personaje con componentes del editor.

El siguiente componente por añadir será la **cámara**, creamos un componente cámara en un nuevo GameObject y dejamos su programación para más adelante.

Otro GameObject que tenemos que añadir es la **interfaz de usuario** *(Ver figura 47)*, esta a su vez se compone de varios GameObjects hijos que corresponden a cada una de las imágenes y barras que componen el panel de estado del personaje y sus botones. Todos los comportamientos de estas se programan en el objeto ***PlayerModel*.**

Algunos objetos serán programados cuando se produzca cierta acción en el juego, estos objetos permanecerán **desactivados** apareciendo con un color azul difuminado. Ejemplo de este panel puede ser el panel de **Muerte** o el que muestra el **inventario de consumibles**. Este comportamiento será **controlado por código**, y se explicará más adelante.

Llegados a este punto ya contamos con los assets preparados para su programación.

Empezaremos con la programación del movimiento del personaje*(Ver figura 48)*. Creamos un script que se encargará del control del personaje y lo asignamos, este script modificará varios componentes. Por un lado necesitará conocer el ***Animator controller*** para regir el cambio entre estados de las animaciones, el estado en el que se encuentra el **Sprite de la UI**, regido por otro **Script** y varios **parámetros** de velocidad que podemos editar desde código, finalmente se pasa el ***transform* del modelo 3D** para que sirva de output del código, traduciendo la combinación de todo esto en el **movimiento del personaje.**

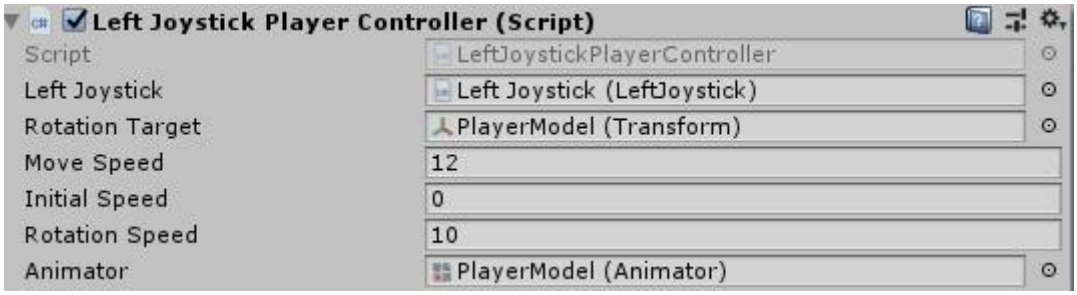


Figura 48: Script de Joystick con sus diferentes Inputs y Outputs

El **resto de script** tales como control de **ataques**, **cámara**, comunicación con la **UI**[43], reaparición y muerte, así como manejo de estadísticas siguen la misma premisa. En algunos casos una serie de componentes actúan de Inputs y otros de outputs.

Con este breve resumen ahora contamos con una visión más detallada del manejo de componentes, objetos y scripts, con esta premisa ahora pasamos al resumen en tablas de las diferentes implementaciones del videojuego, así como su distribución en asset y scripts.

[43] **UI**: Acrónimo en inglés de “User Interface”.

5.3 Esquemático de la implementación:

Tabla 2: Esquemático de la Implementación.

Elemento del juego	Característica	Scripts	Assets
Desarrollo **y diseño del mapa.**	Modelado del terreno.	-	Modelos 3D. Sistema de partículas. Collider.
	Objetos del mapa.	-	Modelos 3D. Sistema de partículas. Texturas. Collider. RigidBody.
	Ambiente.	-	Sistema de partículas. Luces.
Personajes principales.	Animaciones.	Control de animaciones.	
	Modelos personaje.	-	
	Ataques.	Realización de diferentes tipos de ataques. (distinción, lanzamiento de animación, detección de collider en caso de impacto, control de tiempo de animación)	
		Restar salud al personaje enemigo.	Animation Controller. Animación.

		Generación de un objeto a distancia con una dirección, fuerza velocidad dada.	Collider. Rigidbody. Modelos 3D. Imágenes de UI.
		Realización de daño instantáneo	
		Recuperación de vida de ciertos ataques	
		Potenciación de la estadística de ataque	
		Resta de salud del enemigo en función de un temporizador con ataques en área.	
	Movimiento.	Comunicación con el Joystick y traducción en movimiento del personaje.	
		Lanzamiento de animación de andar, detenerse.	
	Cámara.	Seguir al personaje.	Shader de RayosX.
		Cambiar entre escenas.	
		Localizar el modelo del personaje	
HUD	Panel de estado	Control del estado de la barra de maná	Retrato del jugador Adaptación a la pantalla
		Modificación de estados con la obtención de objetos, uso de habilidades.	Imagen de las barras de estado
		Control del estado de la barra de salud	
		Generación de maná mediante un temporizador	
	Botones de ataque	Ejecutar ataque	Identificar ataque pulsado

		Generar objeto y partículas de ataque	Adaptación a la pantalla
		Control de temporizadores de cooldown de los ataques	
		Generacíon de mascara y cuenta atrás del cooldown	
		Comunicación con las barras de estado del jugador	
	Joystick	Movimiento del personaje	Imagen de la UI Adaptación a la pantalla
		Movimiento del joystick en la UI	
Personajes enemigos	IA	Perseguir jugador	Collider
		Dañar jugador	RigidBody
		Salud personaje enemigo	Animator
		Dejar caer objetos aleatorios	Modelo 3D
Items	Coger	Generar ítem aleatorio	Modelo 3D Animation Sistema de partículas
		Coger ítem y eliminar de escena	-
	Estadísticas	Modificar estadísticas de salud	Imágenes de objetos en inventario
		Modificar estadísticas de maná	
		Modificar estadística de velocidad	
		Identificar tipo de objeto y lanzar modificador	

	Panel de información	Modificar panel	
Menú	Movimiento entre menús de una misma escena	-	
	Pantalla del titulo	Comprobación de primer inicio de sesión y comunicación con panel de registro	Adaptación a la pantalla Texto de botones
	Menú de registro	Gestión de registro y comunicación con nube	Componente button Paneles de imágenes
	Menú de inicio de sesión	Gestión de inicio de sesión y comunicación con la nube	
	Menú principal	Saludo al jugador conectado	
Pantalla de selección de personaje	Escena	Movimiento giratorio de la escena	Modelos 3D de personajes Sistema de partículas Adaptación a la pantalla
	Selección de personaje	Distinción de modo un jugador o multijugador	Componente button Lanzamiento de pantalla de carga
		Instanciación de personaje en caso de multijugador	Modelo 3D
	Estadísticas de personaje	Mostrar las estadísticas actuales de cada personaje	-
		Establecimiento del observador de estadísticas en el juego	
		Contador de la duración de la partida al comenzarla	

Inventario y tienda	Selección de inventario	-	Imágenes y textos de los personajes Componente Button
	Desplazamiento de objetos por la pantalla	Arrastrar y soltar objeto	-
	Lectura de Objetos	Lectura del JSON con información del objeto	-
		Localización del panel de información de objetos	
	Equipar objetos	Modificación de parámetros	
		Distinción de tipo de objeto	Adaptación del inventario a la pantalla Componente Button
		Permitir solo un objeto por stack de equipo	
	Funciones del inventario	Equipar objetos de un mismo tipo en un mismo stack	Imágenes de los objetos Imagen del personaje y fondo de inventario Movimiento de las celdas con scroll Imagen de paneles
		Guardar estado de las celdas del inventario	
		Guardar parámetros para comunicarlos al resto del juego	
		Lectura de usuario conectado y carga de inventario guardado	
		Distinción de consumibles comprados y comunicación con el inventario de consumibles	
		Distinción de parámetros por personaje y usuario en Playerprefs	
	Compra venta de objetos	Calculo del precios y saldo final	

Inventario de consumibles	Apertura del inventario	-	Componente Button
	Funcionamiento	Lectura de las estadísticas del objeto seleccionado	
		Comunicación con el HUD y modificación	Imagen/botón de objeto seleccionado
		Comunicación con las estadísticas de velocidad del jugador	
		Eliminación del objeto utilizado y comunicación con texto de UI.	
Muerte y reaparición	Eliminación de personaje	Destrucción del modelo de personaje e instanciación de uno nuevo al principio del mapa	
		Lanzamiento de panel de muerte y cuenta atrás	Modelo 3D Representación de la UI
		Búsqueda de la nueva instanciación por la cámara	
Estadísticas finales	Control de estadísticas	Comunicación con la UI para estadísticas de salud, maná y muerte	
		Comunicación con objetos recogidos por el mapa para obtención de estadísticas	
		Comunicación con estado de enemigos para estadísticas de eliminaciones	Panel de muestra de estadísticas Retrato del jugador

		Comunicación con objetos de inventario usados para obtención de estadísticas	
		Temporizador persistente a la muerte del personaje para control de duración de la partida	
	Comunicación con la nube	Envío de las estadísticas finales a la nube	
Pantalla de carga	Comunicación entre escenas	Carga de escena y representación en barra de progreso	Imagen de barra de progreso
Guardar partida	Estadísticas de jugador	Guardado de las estadísticas actuales del jugador para ataque y defensa	-
		Guardado de estadísticas de la última partida	
		Guardado y suma del oro obtenido en la partida	
	Inventario	Guardado y modificación de las estadísticas obtenidas con objetos equipados	
		Guardado de estado de las celdas de los inventarios.	
	Jugador	Guardado de información de inicio de sesión	
Multijugador	Conexión con el servidor	Establecimiento del juego con la nube de PUN	Panel de Lobby
		Lectura del número de jugadores en el lobby	
	Jugador	Instanciación del jugador en el mapa del servidor	

		Sincronización del movimiento del personaje	Modelos 3D Servidor de PUN Animaciones
		Variabilidad del comportamiento de la cámara entre modo multijugador y un solo jugador	
		Instanciación de ataques del personaje en el servidor	
	Múltiples jugadores	Comunicación del estado de los personajes en los diferentes dispositivos	Ayuda de PUN para la comunicación de GameObjects

5.4 Resumen de scripts

Tabla 3: Distribución de los scripts en las diferentes implementaciones.		
Implementación	**Características**	**Nombre(s) scripts implicados**
Personajes principales	Ataques	Caballero_Attack.cs
		Brute_Attack.cs
		Priest_Attack.cs
		Undead_Attack.cs
		Wizard_Attack.cs
		Destroy.cs
		Instancia.cs
		longRangeAttack
		timeAttack.cs
	Movimiento	leftJoystickPlayerController.cs
		leftJoystick.cs
	Cámara	cameraFollow.cs
		checkCamera.cs
HUD	Panel de estado	UIManager.cs
		playerHealthManager.cs
	Control de ataque	Priest_Attack.cs
		Undead_Attack.cs
		Wizard_Attack.cs
		Caballero_Attack.cs
		Brute_Attack.cs
	Joystick	leftJoystick.cs
Personajes enemigos	IA	IA.cs
		RDV.cs
		RDA.cs
		EnemyHealthManager.cs
		chase
		hurtPlayer.cs

Ítems	Coger	pickObject.cs
		Keys.cs
	Estadísticas	statsInGame.cs
		statsEndGame.cs
		manaBottle.cs
		leftJoystickPlayerController.cs
Menú	Inicio de sesión, registro y comprobaciones	initialLogin
		SignIn
		SingUp
Pantalla de selección de personaje	Movimiento y selección	SelectPlayer.cs
		warriorSelect.cs
	Estadísticas	StatsInGame.cs
		SelectPlayer.cs
		statsSelectionPanel.cs
Inventario y tienda	Funciones	CellsGroup.cs
		dummyInventoryControl.cs
		inventorySelect.cs
		BaseCelll.cs
		Highlight.cs
	Objetos	priceItem.cs
		stackItem.cs
		sortItem.cs
		buscarPanelInfo.cs
		itemCell.cs
		BDHandler.cs
	Equipar	DummyParameters.cs
		stackGroup.cs
		sortCell.cs
		Stackcell.cs
		Stackcell.cs
	Estadísticas	DummyParameters.cs
		saveLoad.cs

	Compra/venta	priceGroup.cs
		priceItem.cs
		stackItem.cs
Inventario de consumibles	Funcionamiento	statsInGame.cs
Muerte y reaparición	Funcionamiento	Respawn.cs
	Cuenta atrás del panel	reverseCount.cs
Estadísticas finales	Control estadísticas	statsInGame.cs
		statsEndGame.cs
	Comunicación con la nube	StatsEndGame.cs
Pantalla de carga	Comunicación entre escenas	Loading.cs
		loadScene.cs
Guardar partida	Estadísticas del jugador	stastEndGame.cs
	Inventario	saveLoad.cs
	Datos de jugador	statsInGame
		signIn.cs
		saveLoad.cs
Multijugador	Conexión con servidor	Sync.cs
		NetworkManager.cs
	Instanciación del jugador	NetworkManager.cs
		SelectPlayer.cs
	Comunicación entre jugadores	PhotonView.cs

6 Implementación.

En este punto se explicará cada una de las implementaciones con las que cuenta el juego, desde el diseño y uso del mapa del juego hasta el tratamiento de las estadísticas y datos del jugador, pasando por la programación de los personajes.

Este punto seguirá el mismo orden que se llevó a cabo para el propio desarrollo de estas implementaciones, en él se explicará en qué consisten, cuál es su funcionamiento, así como, que herramientas de Unity, Scripts y Assets se utilizan, acompañando la lectura con capturas de pantalla de cada una de ellas.

6.1 Desarrollo y diseño del mapa:

Antes de abordar la programación del juego es conveniente desarrollar un entorno en el cual realizar las pruebas y un mapa, con zonas bien diferenciadas, con una temática en cada una con el fin de llamar la atención visual del jugador y favorecer al "***gameplay***"[44].

El diseño se realizó sobre papel con la idea de tener varias zonas diferenciadas, cada una de ellas con un jefe de zona, para posteriormente definirlas en Unity con la herramienta "***Terrain***"[45] *(Ver figura 49)*. Mas adelante se introducen los diferentes modelos 3D decorativos del escenario obtenidos de la Unity Asset Store, y se rediseña el terreno base en función de los modelos utilizados.

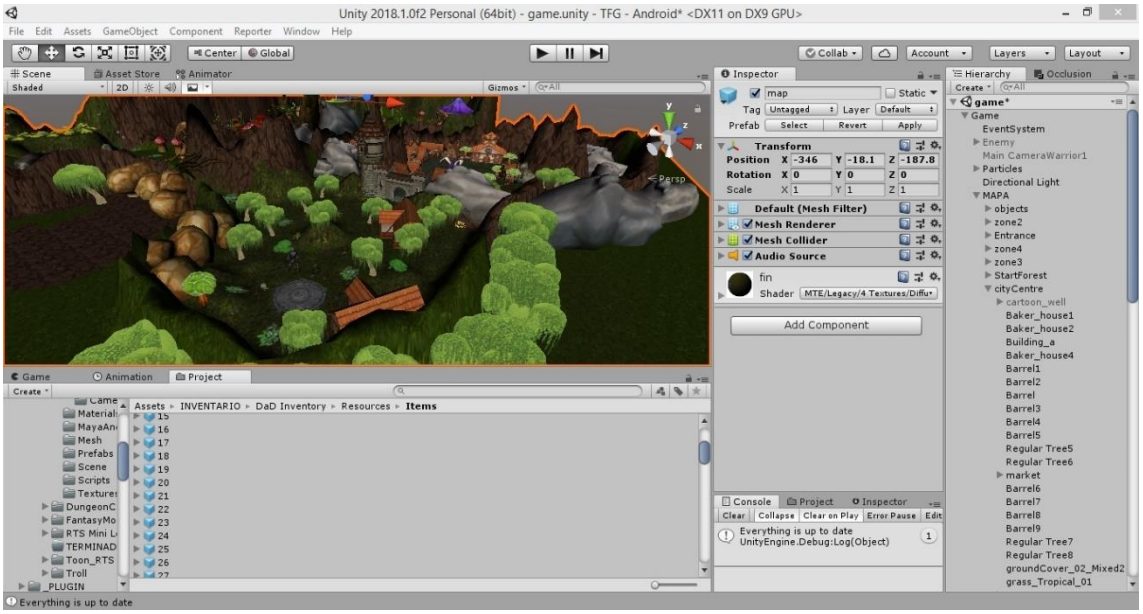


Figura 49: Unity durante el proceso de desarrollo del mapa.

[44] **Gameplay**: Conjunto de acciones que puede realizar un jugador para interactuar con juego o la forma en la que este interactúa con el propio jugador.
[45] **Terrain**: Herramienta de desarrollo de mapeado en 3D incluida en Unity

- **Colocación de los asset en el mapa.**
  Colocamos sobre los diferentes objetos los collider para detectar las colisiones con los personajes principales, que en función de su localización en el mapa agruparán varios objetos con el fin de reducir en procesamiento en el motor.

- Finalmente situamos algunos **sistemas de partículas[46], luces** y animaciones por el mapa con el objetivo de darle más vida.

.

- **Adaptación de la cámara:** El mapa está diseñado pensando en que en todo momento contaremos con dos cámaras, la **cámara principal** y la **cámara de rayos X** que veremos mas adelante. Ambas cámaras están agrupadas al mismo GameObject y actúan de forma similar, manteniendo una rotación fija solamente cambiando su posición en los **ejes X y Z** en función del movimiento del personaje**.**

- **Adaptación para Android:** La herramienta *terrain* es un método para creación de entornos 3D muy poderosa, nos permite modelar y texturizar un modelo 3D del paisaje, añadir árboles y hierba, así como zonas de aire. Esta herramienta ha tenido que ser modificada para su correcto uso en la plataforma Android.

6.2 Personajes principales.

Se ha realizado una búsqueda de personajes que encajasen bien en el estilo grafico del juego. El juego cuenta con 5 personajes principales, cada uno con sus habilidades únicas. Además, encontraremos personajes enemigos controlados por una inteligencia artificial, que a su vez se dividen en varias razas con sus ataques y aspectos característicos.

A continuación, hablaremos de los personajes principales que aparecen en el videojuego, su rol, comportamiento y habilidades, así como de los diferentes elementos que los componen.

6.2.1 Modelos.

Son aquellos que el jugador puede controlar *(Ver figura 50)*, cada uno representa una forma de juego diferente con el fin de que el juego conjunto de roles lleve a los jugadores a lograr la victoria. Estos son:

[46] **Sistema de partículas**: Asset de Unity consistente en generadores de pequeños sprites y luces para generar fluidos, o flujos animados, tales como el fuego o chorros de agua.

1.- Warrior: es el personaje más equilibrado, cuenta con unas estadísticas base de ataque y defensa muy similares, con ataques principalmente de cuerpo a cuerpo. Está pensado para ser una de las puntas de lanza en los ataques.

2.- Undead: es uno de los personajes en el que predomina el ataque a distancia, su rol será el de proporcionar cobertura a los personajes que ataquen cuerpo a cuerpo.

3.- Priest: Es el personaje más débil en comparación al resto. Está orientado a hacer de apoyo al resto de personajes con alguna de sus habilidades curativas. Puede dañar a enemigos, pero muy lentamente.

4.- Wizard: es una elección similar al Undead, pero orientado a hacer más daño, y soportar menos golpes.

5.- Brute: es el personaje con la salud más alta. No cuenta con un daño muy elevado por tanto está pensado para hacer de tanque y recibir el daño de los enemigos evitando así que sus compañeros más débiles lo reciban mientras los eliminan desde la distancia.

6.2.2 Funcionamiento de los ataques.

Los ataques constituyen la forma con la que los personajes aliados eliminan a los enemigos, y pueden ser: "**Cuerpo a cuerpo**", "**a distancia**" o "**Potenciadores**".

(1) Ataques cuerpo a cuerpo de personaje principal.

Los personajes Warrior y Brute son aquellos que cuentan con un ataque cuerpo a cuerpo, éste consiste en un ***ontriggerEnter***[47], el cual al colisionar con un objeto con el tag "***Enemy***", le ocasionará un daño.

Tabla 4: Ataques cuerpo a cuerpo		
ICONO		
	**Rol**	Warrior
	Efecto	Daño simple
	Tipo	Básico
	Descripción	Realiza un ataque básico con la espada, el cual causa daño al enemigo.

ICONO		
	**Rol**	Brute
	Efecto	Daño simple
	Tipo	Básico
	Descripción	Realiza un ataque básico con una maza, el cual causa daño al enemigo.

ICONO		
	**Rol**	Brute
	Efecto	Daño en área
	Tipo	Área
	Descripción	Realiza un salto sobre los enemigos, ocasionado un daño en área alrededor de la zona de caída del personaje.

[47] **OnTriggerEnter**: Llamada que hace un Collider a un script designado cuando choca con otro componente collider.

(2) Ataques lejanos de personaje principal

Los ataques lejanos son ideales para realizar daño a enemigos o grupos de ellos sin verse expuesto a ataques, estos ataques se realizan al frente de los personajes creando una estancia de un ***prefab*** previamente creado, el cual cuenta con un script encargado de procesar el impacto y el daño efectuado al enemigo. Puede ser manejado mediante el script "***longRangeAttack***" cuando es un daño instantáneo o con el script "***areaAttack***" si resta salud al enemigo si se mantiene en una posición fija. Este tipo de ataque es el que más abunda en el juego y se divide a su vez en dos tipos, "***lanzamientos"*** y "***áreas a distancia"***:

Tabla 5: Ataques a distancia		
ICONO		
	**Rol**	Warrior
	Efecto	Daño en área recta
	Tipo	Área
	Descripción	Invoca una serie de piedras delante del personaje que caen sobre los enemigos.

ICONO		
	**Rol**	Wizard
	Efecto	Daño a distancia
	Tipo	Básico
	Descripción	Lanza un orbe mágico que ocasiona daño a los enemigos.

ICONO		
	**Rol**	Undead
	Efecto	Daño a distancia
	Tipo	Básico
	Descripción	Lanza una bola de energía que ocasiona daños a los enemigos con los que impacta.

ICONO		
	**Rol**	Undead
	Efecto	DPS en área.
	Tipo	Área
	Descripción	Crea un portal de magia oscura que resta salud a los enemigos mientras estén en su área de efecto.

ICONO		
	**Rol**	Priest
	Efecto	Daño a distancia/ sanación
	Tipo	Básico
	Descripción	Lanza un haz de luz que daña a los personajes enemigos y sana ligeramente a los aliados.

ICONO		
	**Rol**	Wizard
	Efecto	Daño en área
	Tipo	Área
	Descripción	Impacta un rayo ocasionando daño a los enemigos que alcance.

ICONO		
	**Rol**	Undead
	Efecto	Daño en área
	Tipo	Definitiva
	Descripción	Desprende una nube de gas toxico que realiza daño a los enemigos dentro de su área de efecto.

(3) Potenciadores de jugador.

Los ataques potenciadores proporcionan una ventaja temporal al jugador que lo utiliza o al resto de jugadores, ya sea en forma de un incremento del daño que realizan al enemigo o una mejora de salud. Estos ataques son:

Tabla 6: Ataques potenciadores		
ICONO		
	**Rol**	Warrior
	Efecto	Potenciador
	Tipo	Definitiva
	Descripción	Cubre al personaje con un aura de fuego que le proporciona un incremento de 10 puntos de daño durante 25 segundos.

ICONO		
	**Rol**	Brute
	Efecto	Potenciador
	Tipo	Definitiva
	Descripción	El personaje entra en un estado de furia con el que realiza un ataque poderoso y se incrementa su ataque 10 puntos durante 25 segundos.

ICONO		
	**Rol**	Priest
	Efecto	Sanación
	Tipo	Área
	Descripción	Deja un orbe en el mapa que puede ser cogido por un aliado para recuperar salud, si en un tiempo de 10 segundos no lo ha recogido nadie se autodestruye.

ICONO		
	**Rol**	Wizard
	Efecto	Potenciador / daño
	Tipo	Definitiva
	Descripción	Crea un vórtice a su alrededor que origina un incremento del maná que obtiene por segundo tanto el como aliados cercanos, a su vez realiza daño a los enemigos.

ICONO		
	**Rol**	Priest
	Efecto	Sanador
	Tipo	Definitiva
	Descripción	Realiza una explosión de luz que sana 75 puntos de salud al personaje y origina un área a su alrededor que sana varios puntos de salud por segundo a los jugadores cercanos.

6.3 HUD

El HUD (Heads-Up-Display) de los videojuegos hace referencia a los elementos que se muestran en pantalla dando información al usuario de elementos como la cantidad de vida, la puntuación, número de armas etc. En el proyecto se han implementado una serie de elementos como son la barra de vida, el maná (o puntos de magia), un marco del personaje que se está utilizando y puesto que estamos en una plataforma móvil, los botones de control de ataque, así como el joystick de movimiento de personaje.

Para poder desarrollar estos elementos se ha utilizado un nuevo sistema que incorpora Unity 3D en las últimas versiones llamado ***Canvas***[48]. El Canvas es un GameObject con un componente Canvas donde tendremos que incluir todos los elementos de la UI.

6.3.1 Barra de estado del personaje

Para el desarrollo de esta parte de la UI de los jugadores se ha utilizado una serie de sprites, combinados y superpuestos obtenidos de la tienda de Unity Asset Store y modificadas con el retrato en función del personaje que esté manejando el jugador.

Esta se divide en:

Barra de salud: Representa la salud con la que cuenta el personaje en ese momento dado. Está realizada mediante un **slider**[49], que se reducirá cada vez que el jugador sufra daño.

Barra de mana*(Ver figura 51)*: Representa el poder mágico con el que cuenta el personaje. A excepción del ataque básico, los ataques de "**Área**" y "**Definitivo**" necesitarán hacer uso de "**maná**". Si tenemos suficiente maná podremos hacer uno de estos ataques. Podremos hacer uso de la habilidad "Área" si teníamos al **menos 50 puntos de mana** y "Definitiva" si contamos con **al menos 100 puntos de maná.**

Figura 51: Aspecto de la barra de maná después de usar la habilidad "Área"

Cada vez que utilicemos algunos de estos ataques, su coste se restará a esta barra. Ésta se regenerará lentamente con el paso del tiempo, con ayuda de unos temporizadores o recogiendo **"pociones de maná**" por el mapa.

[48] **Canvas**: Componente de Unity designado para mostrar la interfaz de usuario, actúa de GameObject padre para los elementos de la UI.
[49] **Slider**: Componente de Unity utilizado para realizar barras con aspecto y comportamiento variable mediante programación.

Retrato de personaje: Sprite meramente decorativo, en el que podemos ver el rostro del personaje que estamos controlando. *(Ver figura 52)*

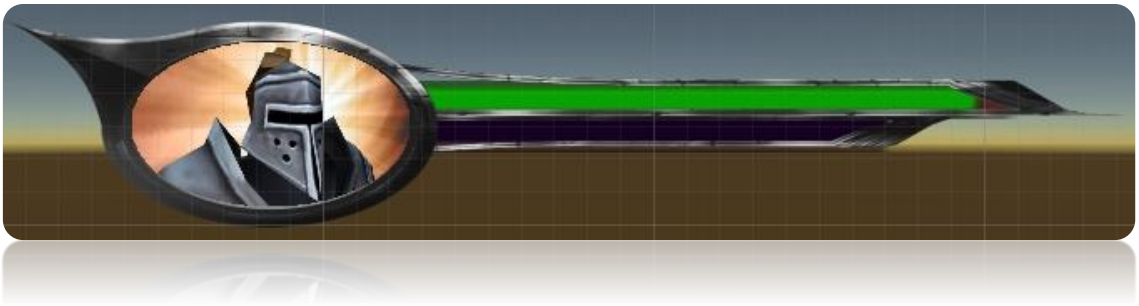


Figura 52: Barra de estado.

6.3.2 Ataques del personaje

Puesto que este videojuego está siendo desarrollado para plataformas móviles con pantalla táctil, todos los controles deben ir representados en la misma UI. Por un lado, tendremos en el extremo inferior derecho de la pantalla los tres botones representados por *sprites* que pertenecerán a **los ataques***(Ver figura 53)*, el más **básico** en un tamaño ligeramente superior, en la esquina, el **área** encima de este y finalmente la habilidad **definitiva resaltada con una exclamación** para que el jugador tenga más facilidad en identificarla, puesto que en cada personaje las imágenes, así como sus ataques varían.

Como comentaba en el punto anterior, estos ataques irán directamente relacionados con el uso de maná y su barra. Cuando usemos uno de los dos ataques especiales se activará una secuencia de "**enfriamiento**[50]" en el ataque utilizado, bloqueando su uso (aunque se cuente con mana suficiente) hasta que pase ese tiempo representado con una cuenta atrás. *(Ver figura 54)*

[50] **Enfriamiento(cooldown):** Estado en el que mediante una temporización se bloquea una acción.

Figura 54: Botón de ataque en proceso de enfriamiento

6.3.3 Joystick

El Joystick es el elemento encargado del movimiento del personaje, éste obtiene la posición en el espacio del personaje y la modifica, permitiendo el desplazamiento del mismo.

(Ver figura 55) Consta de dos ***sprites***, uno estático ***(1)*** simplemente decorativo simulando la base de un joystick real, y un nuevo Sprite que hace de joystick en sí, el cual se mueve sobre un eje ***(2),*** volviendo a su posición original cuando dejamos de pulsar en él.

6.3.4 Cámara

La cámara con la que cuenta el juego se divide en dos, la **cámara principal** *(Ver figura 56)* y la **cámara de rayos X**. Ambas cámaras están agrupadas al mismo GameObject y actúan de forma similar, manteniendo una rotación fija solamente cambiando su posición en los **ejes X y Z** en función del movimiento del personaje.

La cámara irá en todo momento siguiendo al personaje controlado por el jugador, manteniéndolo en una posición ligeramente centrada y a una misma altura.

Por otro lado, la cámara de Rayos X es similar a la cámara principal, compartiendo su movimiento. Sólo cambia que, en este caso, un ***shader***[51] "***XRayReplacement***" se encargará de dibujar una silueta del personaje mediante un ***Shader*** cuando éste detecte que hay un objeto cubriendo al personaje *(Ver figura 57)*. Para ello es necesario establecer a los objetos que pueden cubrir al personaje con una textura "***XRay Shader***".

[51] **Shader**: Componente que se asocia a las texturas y le añade ciertos efectos.

Figura 56: Vista de la cámara original.

Figura 57: Vista de la cámara de Rayos X funcionando.

Mediante la inclusión de esta segunda cámara de Rayos X solucionamos el problema de perder al personaje de vista al pasar por ciertas zonas del mapa.

Esta mecánica no se cumple con todos los objetos, puesto que las texturas que permiten "***XRay Shader***" son muy limitadas y en algunos casos empeoraban el aspecto gráfico del juego.

6.3.5 Panel de información.

Consta de un texto que nos va informando de diferentes acciones que tomamos en el transcurso de la partida *(Ver figura 58)*, por ejemplo, nos avisa de cuando hemos obtenido monedas y que cantidad de estas, así como de información útil

del mapa**,** esta información se obtiene **de diferentes scripts,** tales como los de ataque del personaje, estadísticas, objetos recogidos, etc.

Figura 58: Panel con información in-Game

6.4 Personajes enemigos.

Son aquellos que nos impedirán cumplir nuestro objetivo, cuentan con una IA desarrollada por un proyecto fin de grado paralelo titulado "***Definición y estudio de las estrategias de un videojuego RPG desarrollado en Unity".***

En este proyecto se diseñaron los diferentes scripts, así como la distribución de los colliders de Unity que se encargarán de obtener la información de sus proximidades para regir los diferentes comportamientos de los personajes en según qué situación.

Un collider esférico de gran tamaño en estado **trigger**[52] será utilizado como campo de visión del personaje, otro en el mismo estado, pero esta vez rectangular servirá como zona de ataque y un último collider esférico esta vez sin trigger servirá de receptor de daño y de pared para que el personaje no pueda ser atravesarlo *(Ver figuras 59 a 62)*.

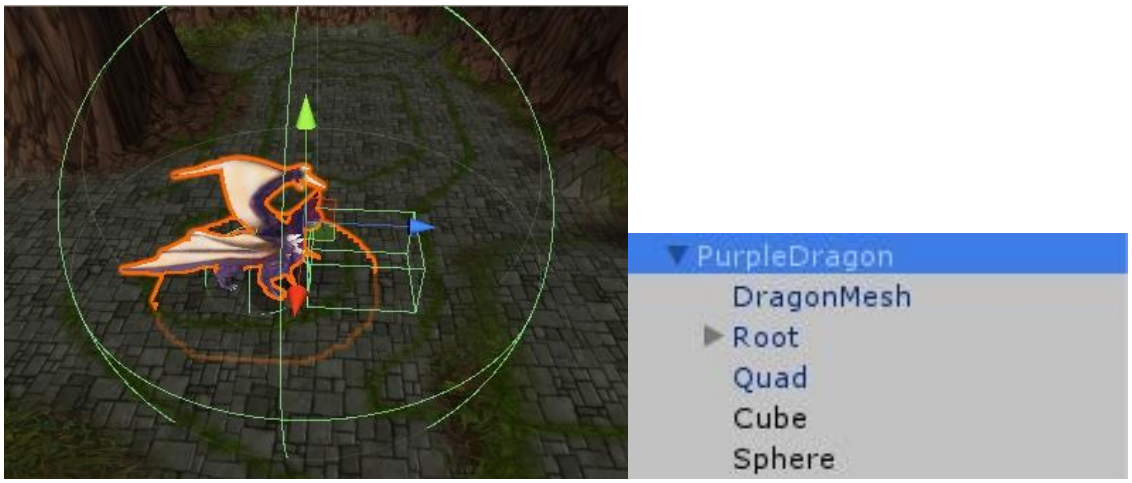


Figura 59: Vista de un enemigo en el editor con sus collider visibles junto a su jerarquía de GameObjects

[52] **Trigger**: Función de la clase collider que en función de si está activado o no, envía información sobre colisiones, o estados de RigidBody que esté en contacto con el collider en cuestión.

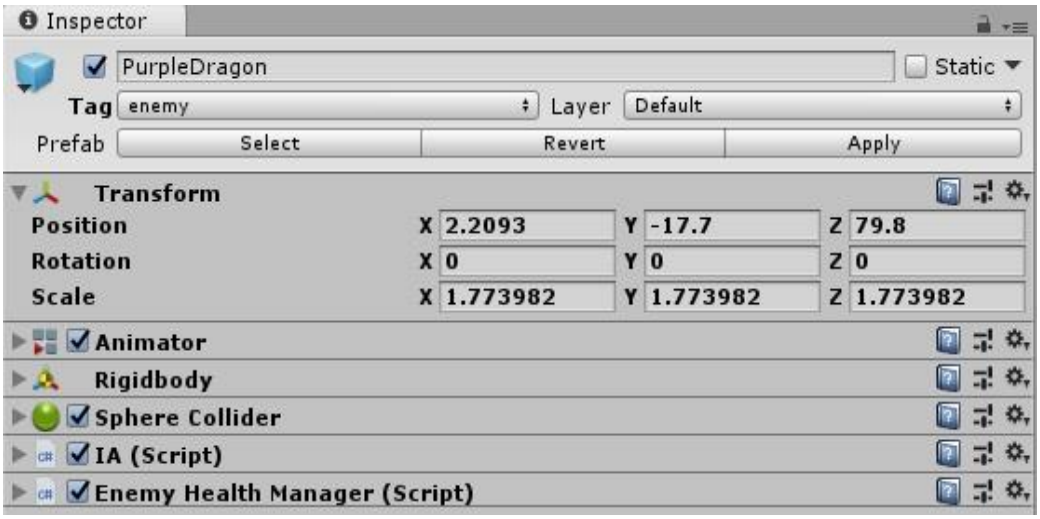


Figura 60: Gameobject perteneciente al modelo 3D del personaje y a la detección de daño.

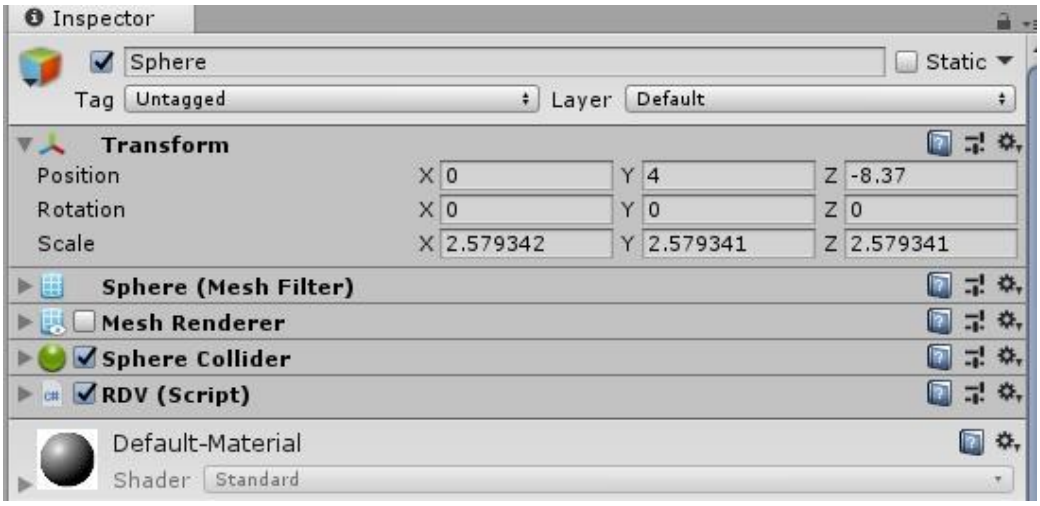


Figura 61: Componentes encargados del ángulo de visión y detección del enemigo.

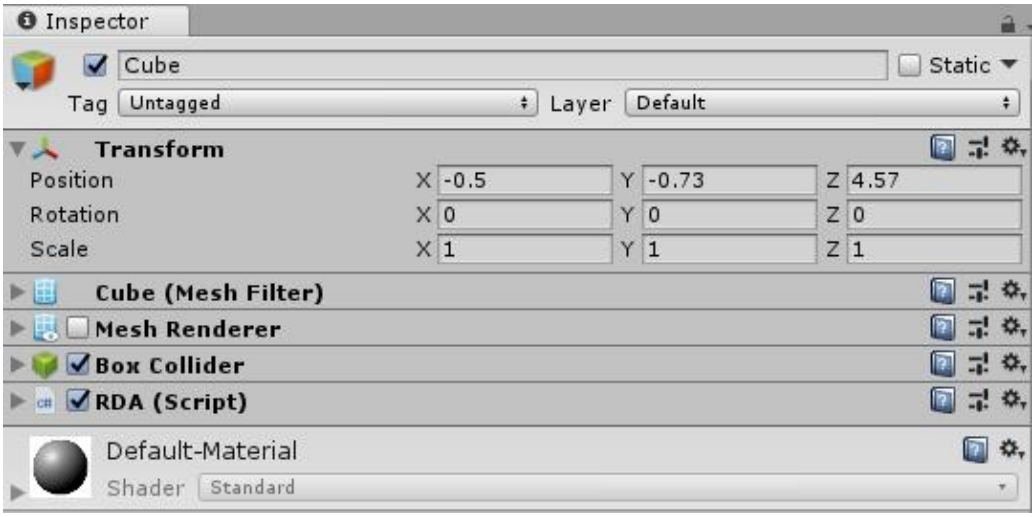


Figura 62: GameObject encargado del ataque del enemigo.

Enemigos básicos *(Ver figura 63)***:** Son los enemigos más abundantes y más débiles de los tres grupos, no presentan ninguna cualidad especial.

Figura 63: Enemigo básico

Enemigos especiales *(Ver figura 64)***:** Suelen ir acompañados de varios enemigos básicos, estos son más duros que los básicos y según cual sea puede darles órdenes a estos en función de la situación, así como realizar ataques más destructivos que un enemigo básico.

Figura 64: Enemigo especial

Jefes *(Ver figura 65)***:** Lideran una zona del mapa, requieren del trabajo en conjunto de los jugadores para poder ser vencidos con facilidad, estos al ser derrotados dejan caer una llave con la cual se puede avanzar a la siguiente zona.

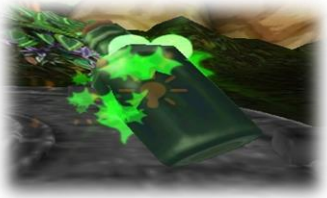
Figura 65: jefe

6.5 Ítems

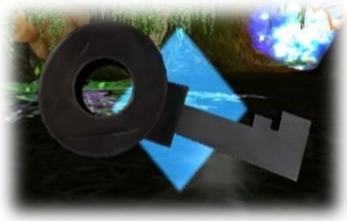
Además de los objetos con los que contamos en la tienda, podemos recoger otro tipo de objetos durante el juego, estos consistirán en un modelo 3D representativo con un collider esférico a su alrededor en modo *trigger*.

Mediante un script generamos un "***RandomDrop"***[53], los enemigos dejarán caer objetos durante la partida, estos variarán en probabilidad en función del poder que tengan dichos objetos. En ocasiones los enemigos no dejarán caer ningún objeto.

Mediante el script "***pickObject"*** y los **tags**[54] detectaremos cuando se produce una colisión con uno de estos objetos y distinguiremos cuál de ellos es, estos objetos son:

Tabla: Ítems		
Imagen	**Nombre**	**Uso**
	**Botella de maná**	Restaura 50 puntos de maná al jugador que la recoja.
	**Botella de salud**	Restaura 50 puntos de salud al jugador que la recoja.
	**Montón de oro**	Añade al jugador 25 monedas de oro, con las cuales podrá adquirir nuevos objetos en la tienda.

[53] **Random Drop**: Generación aleatoria de un objeto arrojado por un enemigo al morir.
[54] **Tag**: Nombre que se da a las etiquetas en Unity.

	**Cofre de oro**	Proporciona 50 monedas de oro para gastar en la tienda.
	**Llave de zona**	Llave mágica que dejan caer los jefes de zona, con esta llave se pueden abrir portales que bloquean el mapa.
	**Llave fin del juego**	Llave de la victoria, al obtenerla se da por finalizada la partida.

6.6 Menús

Los menús se componen por una serie de paneles y botones conectados entre sí, estos botones activan y desactivan otros paneles, así como activan scripts y funciones para desplazarnos entre los diferentes apartados y escenas del juego.

6.6.1 Menú de inicio.

El menú de inicio muestra un único botón de “Play” que al pulsarlo puede ocasionar dos cosas:

Si es la primera vez que se accede al juego aparecerá una ventana para hacer ***login*** o hacer un registro. Si se elige registrarse aparece un formulario *(Ver figura 66)*, que por un lado guarda en ***playerPrefs***[55] los datos del dispositivo y a su vez conecta con la página web del juego a la que envia un archivo **JSON** con los datos rellenados.

Estos datos guardados de inicio de sesión nos servirán para mantener la sesión iniciada, ahora cada vez que el usuario se ponga a jugar y pulse ***“Play”*** accederá directamente al menú principal del juego.

[55] **Playerprefs**: Herramienta empleada por Unity para programar guardado de datos en registros.

Figura 66: Menú de registro

Si por otro lado se quiere iniciar sesión con un usuario existente, haciendo *login* se conectará con la base de datos y de ser correcto el correo y contraseña se inicia sesión con ese usuario.

Por temas de rendimiento este proceso de ***login*** solo es válido para iniciar partida y guardar estadísticas con ese usuario, no se guardarán los objetos comprados, pero si el oro ganado**.**

6.6.2 Menú principal.

El menú principal presenta un panel de selección de modo, ***Single Player, Multiplayer*** y ***Shop*** en forma de imágenes acompañadas de un pequeño panel con una breve descripción. Desde aquí se manejan las diferentes partes del videojuego*(Ver figura 67)*.

Cada una de las imágenes anteriormente nombradas cuentan con la propiedad ***Button***, estos botones al pulsarlos activan el GameObject encargado de la pantalla de carga y le establecerá un nombre a este GameObject. En función del nombre que tenga la pantalla de carga se carga una escena u otra. La escena "***selection***" perteneciente al modo "Single Player", la escena *"**selection_multiplayer*** "perteneciente al modo "***Multiplayer***" y la escena "***Inventory***" la cual contiene la tienda, el equipo y el inventario del usuario para los diferentes personajes.

Figura 67: Menú principal

Además de estas opciones se cuenta con un pequeño botón "***Options***", el cual nos permite **volver a las pantallas de *login* y *register* anteriormente nombradas.**

6.7 Pantalla de carga

Cuando se pulsa un botón o sucede una acción que carga una nueva escena aparece una pantalla de carga que mediante una **corrutina**[56] carga la escena y por otro lado muestra una pantalla con una **barra de progreso de la carga** *(Ver figura 68)*, una vez terminada la carga transporta a la nueva escena. Para llamar a esta pantalla de carga la activamos en la jerarquía y cambiamos el nombre de su **GameObject** poniéndole de nombre la escena a cargar cuando se pulse cualquiera de los botones de modo del menú principal. Un script se encargará de mira el nombre que se ha dado al GameObject de la pantalla de carga para carga la escena correspondiente.

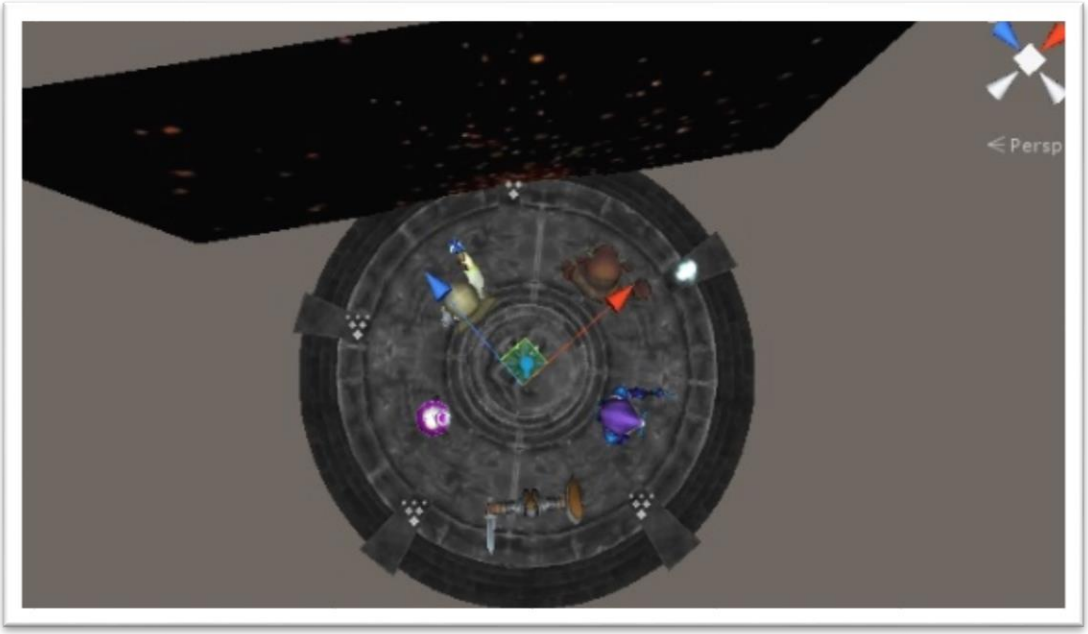
Figura 68: Pantalla de carga

[56] **Corrutina**: Al llamar a una función, la ejecutamos en su totalidad antes de retornar. Esto significa que cualquier acción tomando lugar en una función debe suceder en una sola actualización de frame.

6.8 Pantalla dinámica de selección de personaje

La pantalla de selección consiste en una **plataforma con los 5 personajes** *(Ver figura 69)*, la cámara se mantiene de forma fija mientras que los personajes orientados hacia un eje central giran colocándose frente a la cámara mostrándo sus características básicas.

Puesto que se cuenta con **diferentes personajes para elegir,** se necesitába una forma de representar este apartado, se ha elegido una representación en pentágono que gira sobre su centro por su dinamismo y belleza visual, donde cada personaje se coloca aproximadamente en un vértice de éste mediante **flechas colocadas en la UI** del jugador.



Al pulsar sobre las flechas de selección, el eje gira y mediante un script "***selectionPlayer***" se monitorea la posición en la que se encuentra este eje, cuando coincida con unas coordenadas concretas mostrará al personaje en cuestión, todo esto se realiza de forma transparente para el usuario, dando una sensación fluida de giro sin limitaciones.

Cuando se sitúa el modelo 3D del personaje en cuestión en el centro de la pantalla contaremos con una UI sencilla en la que se puede ver el nombre del personaje y un pequeño panel con una imagen representativa de sus tres ataques, información sobre su rol y estadísticas de ataque y defensa de este. Las estadísticas que se representan en esta pantalla se pueden ver modificadas en función del equipo con el que cuente el personaje en ese momento *(Ver figura 70)*.

Figura 70: Captura "in-Game" de selección de personaje.

6.9 Desarrollo del inventario / tienda.

El inventario y la tienda corresponde un elemento crucial en el desarrollo del videojuego y su evolución, pues el objetivo no es otro que ir mejorando a los personajes con **ítems** comprados en la tienda para así superar los retos más difíciles que va presentando el juego.

6.9.1 Proceso visual del desarrollo:

El aspecto visual del inventario al margen de ser adaptado para las plataformas móviles ha pasado por varias fases en función de lo que queríamos lograr, en un primer momento *(Ver figura 71)* el inventario contaba con una ventana común para seleccionar y ver el inventario de todos los personajes, diseño que resultaba confuso y en ocasiones molesto puesto que si pretendías comprar un objeto que no pertenecía a la clase seleccionada aparecería un aviso que te informaba de ello, por ello decidimos separar selección de inventario y los inventarios en dos escenas diferentes, mejorando así el procesamiento de los datos guardados logrando un manejo más fluido. A su vez, este primer diseño pretendía excluir de aquí la ventana de equipamiento y pasarla a un inventario **"*In Game*[57]"**, pero por cuestiones de dificultad y de comportamiento de JSON se decidió desechar la idea.

[57] **In Game**: Se dice de aquellos elementos que aparecerán dentro de una partida.

Figura 71: Primer diseño del inventario (Desechado).

Como resultado se ha agrupado equipamiento, inventario y tienda en una misma ventana*(Ver figura 72)*, ahora el jugador solo podría equiparse objetos aquí, así como comprar consumibles y realizar las compras que considere oportunas. Dando lugar al desarrollo de un segundo inventario llamado **"Inventario *inGame***" *(Ver punto 6.10)* para que el jugador pueda acceder a ciertos objetos de su inventario durante una partida de una forma más sencilla para el jugador y de procesar por el propio motor que su idea original.

Figura 72: Vista del inventario y tienda de Warrior.

Antes de entrar en detalle en el funcionamiento de cada uno de estos paneles vamos a ver el **uso general.**

6.9.2 Contenido y funcionamiento.

Se cuenta con gran variedad de objetos que proporciona al jugador diferentes **características.** Es necesario que el jugador pueda de alguna forma ver que ventajas o usos tiene cada uno, esto se logra haciendo una pulsación sobre las imágenes de los objetos, esto desplegará una nueva ventana con la información

de ese objeto *(Ver figura 73)*. Esta mecánica se logra debido a la propiedad **button** y el un nombre único de cada imagen. Cuando pulsamos en una imagen se llama a un archivo **JSON** que hemos convertido posteriormente en un **Text Asset**[58] que se encargará de buscar donde está el objeto con ese nombre y devuelve todos sus datos para representarlos con la propiedad **Text** de Unity.

Figura 73: Vista del panel de información de un objeto de la tienda

Para **comprar** y **guardar** en el inventario o equipar un objeto hay que **pulsar** y **arrastrar** el objeto a la casilla deseada. Cuando se arrastra el objeto se representa en todo momento siguiendo al dedo, cuando se suelta, un script comprueba **si tiene alguna celda debajo** y **si tiene permitido meterse en esa celda (esto se consigue mediante un script).** Si el objeto tiene permitido colocarse en esa celda aparece una ventana de intercambio en la que se muestra lo que cuesta el objeto. En la ventana de intercambio de oro si se tiene suficiente oro y se acepta la transacción se añade el objeto a la celda anteriormente descrita y se resta el importe de oro que ha costado, si no hay suficiente oro para realizar la transacción aparece un panel aviso avisando de esta falta de oro. Si cumplimos los anteriores requisitos de dinero, pero no cumplimos los de permisión de la celda comentado anteriormente (en el caso del panel "**Equipo**") no se equipará y pasará al inventario directamente.

6.9.3 Paneles del inventario

El inventario es independiente para cada personaje y consiste en tres paneles. Siguiendo el orden de izquierda a derecha de la imagen anterior podemos ver:

- **Equipo** *(Ver figura 74)*: Consiste en un retrato del personaje seleccionado, éste cuenta con varias casillas correspondientes a una prenda distinta: ***Casco, cuerpo, protección o joyas, cinturón, pantalones y guante*s**. Cuando se deja caer un objeto sobre una celda, un script se encargará de analizar si es de la categoría que él puede albergar, de no serlo, lo guarda en

[58] **Text Asset**: Función empleada para el manejo de texto de Unity.

inventario directamente, de ser así se equipará y se modifican los parámetros que se pueden observar en la parte inferior de este panel.
Esta distinción la realizamos mediante un Script llamado **"*Sort cell*"** el cual se encargará de comprobar que esa celda permite un objeto de X tipo. ***Sort Cell*** se comunica con **"*Sort Item*",** script asociado a cada objeto, el cual les comunicará a las celdas a que grupo pertenece.

Figura 74: Pantalla de equipo de personaje

Una vez que las celdas obtienen el grupo al que pertenece el objeto, se llamará a un script *"**HighLight***" que iluminará los bordes de las celdas, proporcionando al usuario una guía visual para equipar el objeto seleccionado.
Las estadísticas que obtiene cada personaje se obtienen con los objetos que le equipemos. Se compone de 7 slots, ***"Chest", "Helmet", "Pants", "Hand", "Belt"*** y dos que pueden contener **"*Jewel" o "Protection***" a elección del jugador.
Estos parámetros representan, por un lado, el ataque y la defensa base del personaje y por otro los puntos que ha obtenido por cada objeto equipado. Estos mismos parámetros se comunicarán a nuestro personaje para hacerlo más poderoso en el transcurso del juego.

- **Inventario:** Consiste en una serie de celdas originalmente vacías en las que se van depositando los objetos adquiridos por el usuario en la tienda que no estén equipados. Estos objetos se guardan en todo momento y siempre se pueden equipar o intercambiar con el que esté equipado en ese momento. Además, es posible vender un objeto a la tienda arrastrándolo a esta, con el

inconveniente de que el dinero obtenido por la venta será la mitad de su precio original.

- **Tienda:** La tienda puede cambiar ligeramente en su contenido entre los diferentes personajes, ya que algunos objetos son exclusivos de un determinado personaje. La tienda contiene dos tipos de cada una de las prendas que se comentaban más arriba a excepción de los guantes, las joyas y las protecciones, además de los objetos consumibles que pueden usarse durante el transcurso de una partida. Los objetos consumibles cuentan con un **stock**[59] limitado por cuenta de usuario y personaje siendo irrecuperables una vez usemos todos, con esto se busca que el jugador tome la decisión de usar un determinado objeto consumible o depender de los compañeros aliados o de los ***randomDrops*** para recuperar salud y maná.

  Cuando se quiera comprar un objeto aparecerá un panel *(Ver figura 75)*, en el cual se selecciona la cantidad de objetos que queremos comprar, así como una visualización del costo que tendría el total del objeto elegido y la cantidad seleccionada.

Figura 75: Pantalla de compra.

Si en algún momento se intenta comprar un objeto con un precio superior al saldo del jugador, aparecerá una ventana emergente avisando de que no se dispone de suficiente oro y no se producirá ningún cambio.

6.9.4 Pantalla de selección de inventario de personaje.

La ***Pantalla de selección de inventario de personaje*** *(Ver figura 76)* consiste en un gran panel con el retrato de cada uno de los personajes. Al hacer click en cualquiera de los retratos de un personaje aparece su respectivo **Canvas**, y se

[59] **Stock**: Cantidad de objetos de un mismo tipo almacenados en un mismo lugar.

desactiva este **Canvas de selección**. Al seleccionar un inventario se arranca un script "***saveLoad***" perteneciente a ese personaje seleccionado, que cargará desde ***Playerprefs*** los datos guardados del inventario: posición de los objetos comprados, cantidad, dinero total, objetos equipados, etc.

Siempre será posible volver a esta pantalla de selección pulsando el botón "*Exit*", en la parte inferior de cada inventario, cuando esto ocurre, el mismo Script "***saveLoad***" guardará los datos del último estado del inventario

Figura 76: Pantalla de selección de inventario

Las estadísticas de cada personaje son totalmente independientes entre personajes, obligando al jugador a jugar con cada uno de los personajes para obtener mejor equipo para ese personaje en concreto, evitando aislarse en un solo personaje e incitando al jugador a enfocarse en diferentes roles.

6.10 Inventario de consumibles.

Durante las partidas el jugador puede acceder a los objetos que ha comprado de la categoría "**consumibles**" y utilizarlos *(Ver figura 77)*, este seguimiento se consigue realizando una lectura mediante un bucle de los objetos del inventario que están contenidos **entre la celda 6 y la 22**,(aquellas pertenecientes al **inventario** del jugador, si algún objeto en estas casillas pertenece a algún consumible obtenemos su cantidad y lo guardamos en un ***playerPref*** que cargaremos dentro del juego al elegir ese personaje. Cuando se pulsa sobre uno de estos ítems se reduce su número y una vez que llegue a cero no se puede volver a utilizarlos en la partida en curso

Figura 77: Inventario de consumibles

Basándonos en la imagen anterior, los objetos en **verde otorgan salud** en diferente cantidad, mientras que los de la fila inferior en **azul** actúan igual, pero con el **maná** del jugador.

Mientras tanto los dos objetos remarcados en **amarillo son objetos especiales**, por un lado la zanahoria otorga una **mejora de velocidad** temporal y por otro lado el ojo restaurará en su totalidad tanto la salud como el maná

6.11 Muerte y reaparición del personaje

Cada personaje cuenta con un script "***PlayerHealManager***", el cual manteniene un seguimiento de la salud del personaje, actualizando su barra de vida. Si la barra de salud llega a 0 se activa un panel de muerte con un temporizador de diez segundos *(Ver figura 78)*, cuando este temporizador llegue a cero se elimina el modelo del jugador del mapa y se instancia uno nuevo al principio del mapa.

Figura 78: Panel de muerte

Los progresos no se verán afectados, morir solo retrasará la finalización de la partida y sumará una muerte a nuestras estadísticas. Cuando se **vuelve a instanciar** de nuevo el personaje todos los **GameObjects** presentes en la

jerarquía se asocian de nuevo a este mediante una búsqueda en funciones **Awake**()[60] y **Update**().

6.12 IA

Para este proyecto se han desarrollado una serie de scripts de inteligencia artificial *(Ver figura 79)*, tomando el trabajo fin de grado "***Definición y estudio de las estrategias de un videojuego RPG desarrollado en Unity".***

Estas actúan en función del enemigo al que vayamos a asociarlas, esta IA le comunica al enemigo cual es el personaje con el tag "**Player**" más cercano a su posición, si este personaje entra en un radio dado empezará a caminar hacia él, y a atacar cuando se encuentre en cierto rango de ataque.

Entre esta característica básica están presentes otras formas de ataque, ataque en grupo, cobertura a un enemigo "**Líder**", huida, etc.

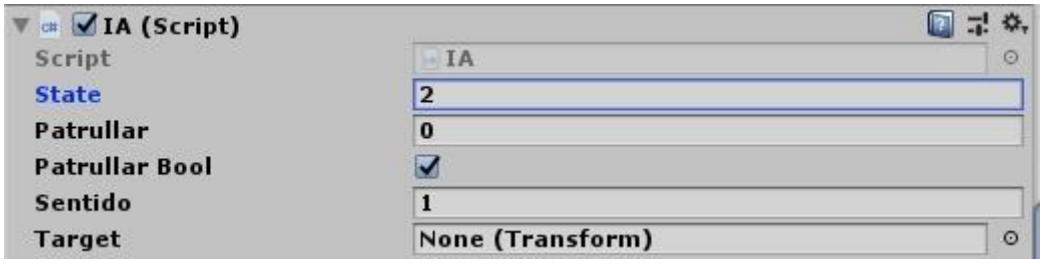


Figura 79: Muestra de un Script de IA

6.13 Estadísticas finales

Cuando se selecciona un personaje en la pantalla de selección le cambiamos el nombre a un objeto "***statsG***" que se encuentra omnipresente durante todo el transcurso de la partida observando al jugador, este objeto contiene un script que va monitorizando: salud perdida, maná utilizado, enemigos eliminados y el tipo de éstos, muertes, tiempo de juego, etc.

Este GameObject va asociado al **GameObject del personaje**, pero no se destruirá si el jugador muere, si esto ocurre, se pondrá a buscar la nueva instanciación del personaje para seguir monitorizando.

Cuando se obtiene la llave dorada, se termina la partida y aparece un panel mostrando todas estas estadísticas del jugador *(Ver figura 80)* durante un tiempo determinado por un temporizar, que cuando llegue a cero nos devolverá al menú del título.

[60] **Awake() y Update():** Funciones propias de la librería de Unity, encargados de ejecutar su contenido al principio de la ejecución y después de cada frame respectivamente.

6.14Conexión con la nube.

Esta conexión se realiza tomando el trabajo fin de grado **"Plataforma web para administración y análisis de un juego RPG".** Pueden darse dos ocasiones en las que es necesario conectarse a la nube.

- Por un lado, cuando el usuario se registra, se envían sus datos de sesión a la nube creando un nuevo usuario (en caso de que no exista uno igual). Si el usuario realiza ***login*** iniciará sesión como el usuario dado y se guardarán nuestras **estadísticas**.
- Finalmente, durante el proceso comentado en el apartado "**Estadísticas finales**" en el instante en el que se muestran las puntuaciones el juego tratará de conectarse a la nube para guardar nuestras estadísticas, creando un archivo **JSON** con los datos para posteriormente enviarlo y poder consultarlas desde la página web.

Nota: Con el objetivo de permitir un juego sin conexión continua a la red, en caso de no contar con conexión a internet esta opción se obvia y no habrá constancia de estas estadísticas.

6.15 Guardar partida.

Los datos del jugador, así como su progreso se guarda en ***PlayerPrefs***.

Esta una forma simple de almacenar y recuperar valores para usar dentro de los proyectos de Unity. Sus principales características son:

- Sus métodos son todos estáticos.
- Solo admiten los valores de string, int y float.
- Hay configuraciones independientes para los diferentes tipos siendo necesario especificar el tipo de un valor particular al obtenerlo y configurarlo.

Playerprefs se almacenan (persisten) en el dispositivo en diferentes registros según donde se esté ejecutando el juego o aplicación.

Para este proyecto ya que va a ejecutarse en Android los datos de *PlayerPrefs* se guardan en ***SharedPreferences***, una interfaz de Android que permite a la aplicación guardar y almacenar datos de forma nativa dentro del contexto de la aplicación. Son ejemplos de esto, bases de datos, parámetros de configuración, etc.

Shared Prefrences se almacena físicamente en: **/data/data/pkg-name /shared_prefs/pkg-name.xml.**

En este proyecto ***Playerprefs*** se utiliza para almacenar todas las estadísticas del jugador, que serán desarrolladas más adelante para crear un archivo JSON y enviarlo a la base de datos Web. Además, con ayuda de varios bucles anidados y *Playerprefs* se establecen que objetos y en que casillas están almacenados para cada personaje, de forma que, aunque el jugador cierre el juego, cuando vuelva tenga estos datos guardados.

6.16 Modo multijugador.

El modo multijugador utiliza el **plugin**[61] de Unity: ***Photon Unity Networking (PUN).*** La conexión llevada a cabo con PUN se basa en una **arquitectura Cliente-Servidor,** en esta arquitectura las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa que actúa de servidor y le da una respuesta. Esta idea suele presentarse en un sistema operativo multiusuario distribuido a través de una red de computadoras, aunque se puede aplicar a programas ejecutados sobre un mismo equipo. Con esta arquitectura se logra que si algún jugador abandona la partida durante su transcurso los demás jugadores no se vean afectados y la puedan terminar.

PUN cuenta con varios servidores a lo largo de todo el mundo, para nuestro juego el servidor más cercano y el que nos ofrece mejores prestaciones es aquel situado en la ciudad de **Amsterdam**.

PUN permite utilizar varios protocolos para la transmisión, **UDP**, **TCP**, **HTTP** o **Websockets**, este último solo con su versión de pago. Para este proyecto hemos empleado UDP, protocolo que no realiza retransmisión de paquetes y puesto que estamos ante un juego en tiempo real, es la mejor opción.

[61] **Plugin**: Herramienta que se puede adicionar al editor para obtener ciertas características extra.

La conexión con el servidor se realiza una vez el jugador selecciona un personaje en la ventana de multijugador. Esta acción activa el script **"*NetworkManager*"** *(Ver figura 81)*, que se encargará de conectar a los jugadores con el servidor, de crear un ***lobby***[62] y de tener un seguimiento de los jugadores que hay conectados a la partida o que quieren conectarse, limitando a cuatro el número máximo de jugadores por partida.

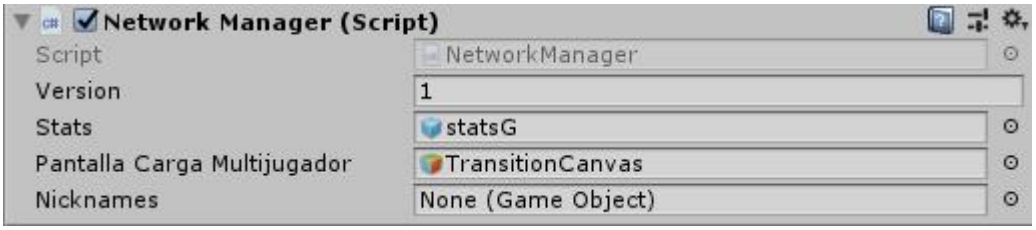


Figura 81: Panel de Network Manager

Por otro lado, se cuenta con el script **"*Photon View*"** *(Ver figura 82)***,** el cual se encarga de observar los objetos de la escena. Este script se asigna a cada objeto que tiene que ser monitorizado por el servidor para que le comunique los datos de sus componentes. Algunos de los componentes que se pueden transmitir al servidor son por ejemplo la posición del personaje, el estado del ***joystick*****,** o el propio "***Network Manager*****"**, entre otros.

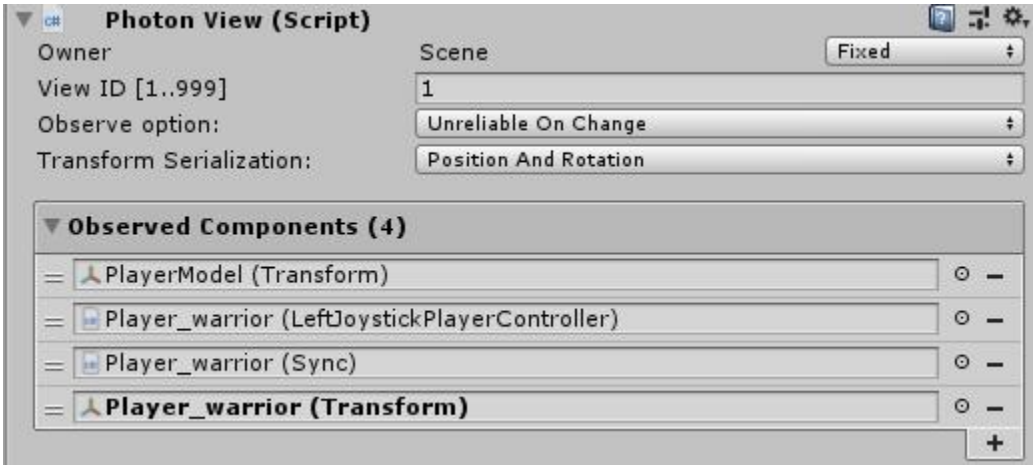


Figura 82: Script de Photon View en la jerarquía

Aparte de este **Photon View** se intenta mejorar la conexión ligeramente a través de un script **Sync** *(Ver figura 83)*, el cual envía **más información al servidor** reduciendo ligeramente el retardo cliente-servidor, y permitiendo enviar información perteneciente a **animaciones** u objetos instanciados en escena.

[62] **Lobby**: Espacio de reunión y espera de los jugadores antes de empezar una partida.

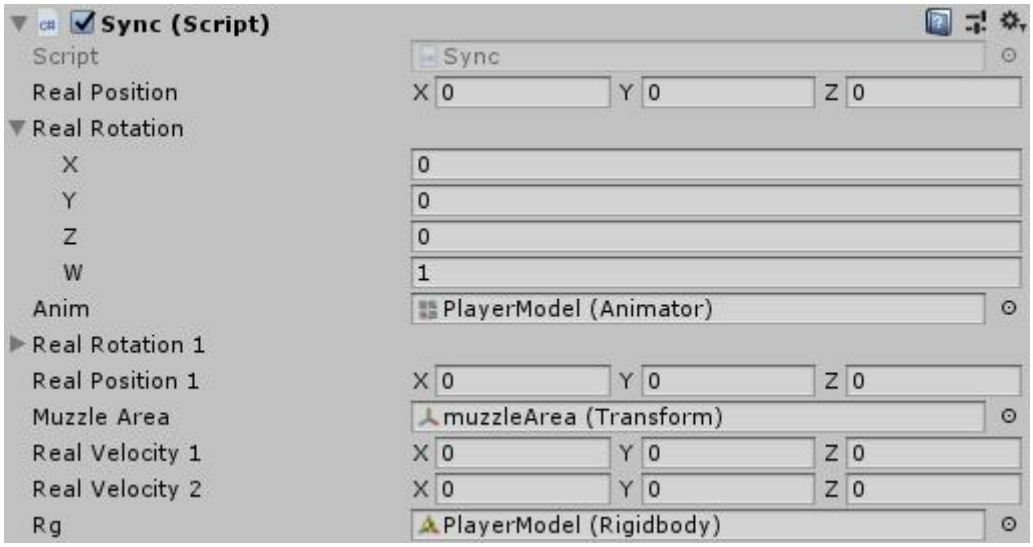


Figura 83: Script de Sincronización en la jerarquía

Aun así, puesto que estamos hablando de un servicio gratuito es muy dependiente de la **conexión** de la que dispongamos, así como la **hora** del día a la que intentemos conectarnos, pueden presentarse ***caídas de framerate***[63] cuando instanciamos algunos objetos, así como perdida de esta información.

6.17 Optimización

La optimización del juego resulta muy importante y crucial en el desarrollo de un videojuego, en este juego se ha realizado la optimización de varias formas:

- **Modelo del escenario:** El escenario esta desarrollado con la herramienta **"*Terrain*"** de Unity, esta herramienta permite moldear con diferentes características los terrenos de forma sencilla dentro del motor.
  Este mapa resultante en formato ***Terrain*** ha sido convertido a un modelo 3D en formato **.obj**[64]**,** ya que la herramienta *Terrain* de Unity no está del todo bien optimizada y realiza múltiples cálculos derivados de la iluminación que para este proyecto resultan innecesarios, numero de polígonos, zonas de aire, etc. La suma de todos estos cálculos resulta imposible de manejar para un dispositivo móvil de gama media-baja y causando problemas de rendimiento en los más potentes. Este problema se solucionó transformando el modelo de la topología del mapa **.terrain** a **.obj**, un objeto más optimizado para un dispositivo móvil que si puede tratar, además con buena tasa de imágenes por segundo.
- **Oclusión*:*** *Occlusion Culling* es una característica que desactiva el renderizado de objetos cuando actualmente no estén visibles por la cámara *(Ver figura 84)*. El proceso de la eliminación selectiva de oclusión *(occlusion*

[63] **Framerate**: Numero de imágenes por segundo que son mostradas en pantalla.
[64] **Formato .obj:** Formato para la representación de modelos 3D, consistentes en el guardado de coordenadas.

culling) va a través de la escena usando una cámara virtual para construir una jerarquía de un conjunto de objetos potencialmente visibles.
Esta información es usada en el tiempo de ejecución por cada cámara para identificar qué es visible y qué no es. Equipado con esta información, Unity asegura que solo los objetos visibles se van a enviar para ser renderizados.

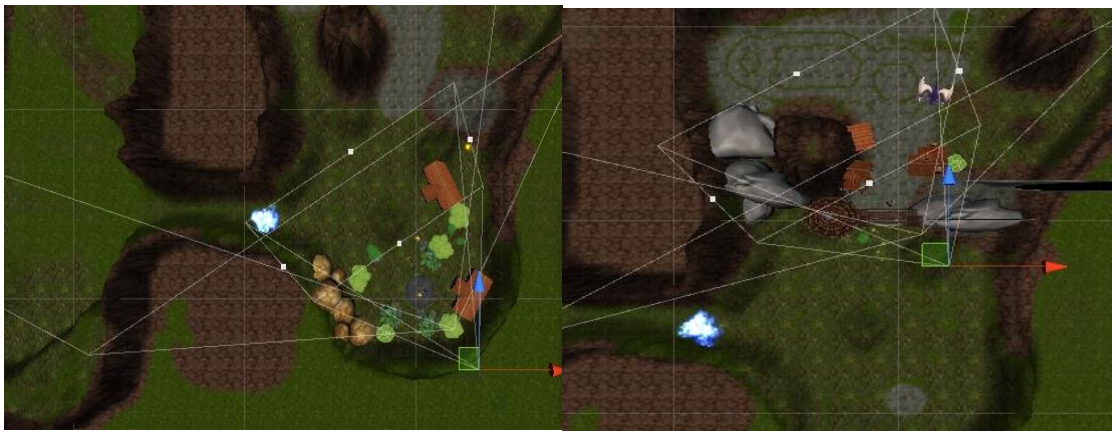


Figura 84: Demostración de cómo actúa la oclusión conforme el personaje se va desplazando.

Desde un primer momento, el mapa entero será dibujado, y serán los pequeños objetos que decoran las diferentes zonas, así como sus enemigos los que se irán cargando lentamente, así lograremos tener la memoria del dispositivo más liberada.

- **Compresión de texturas** *(Ver figura 85)***:**
  Las texturas están además de preparadas para su uso en Android, separadas en dos tamaños diferentes, aquellas pertenecientes a objetos más grande como por ejemplo las texturas del suelo tienen un tamaño máximo de 1024 mientras que aquellos objetos más pequeños tendrán un tamaño máximo de 256, con esto buscamos optimizar lo mayor posible la carga grafica cuando se tengan muchos objetos en pantalla.

  Además, como no contamos con texturas que se requieran muy nítidas el algoritmo que usaremos será el **Mitchell**[65], junto a una calidad de compresión más rápido y menos pesado para plataformas Android.

[65] **Mitchell**: Algoritmo de compresión de imágenes encargado de reducir el peso de las texturas más pequeñas de la escena para realizar un menor consumo de procesamiento.

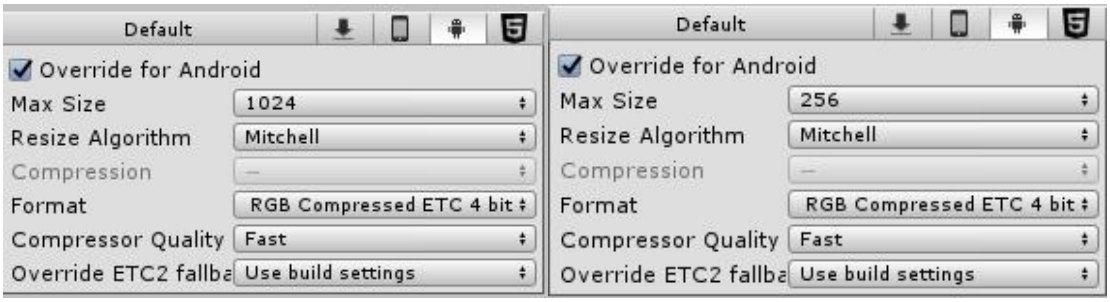


Figura 85: Sistema de optimización de sprites.

- **Gráficos** *(Ver figura 86)***:** Puesto que el videojuego está orientado para una plataforma móvil el juego cuenta con un estilo artístico ***Low Poly*** que proporciona un estilo más caricaturesco aunque menos realista. Este estilo, mejora el rendimiento para plataformas móviles ya que cuenta con el mínimo posible de polígonos en sus modelos 3D, cosa que reduce el peso y procesamiento de los modelos.

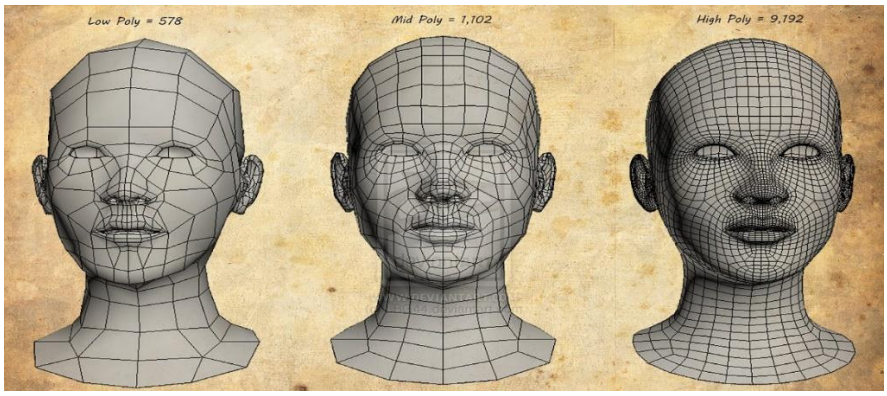


Figura 86: Comparación de los diferentes tipos de renderizados según su cantidad de polígonos.

- **Sombras:**
  Las luces de Unity pueden emitir sombras desde un objeto a otras partes de sí mismo u otros objetos cercanos. Las sombras agregan un grado de profundidad y realismo a una escena ya que estos traen la escala y la posición de objetos que de lo contrario se verían "planos". *(Ver figura 87).* Las sombras utilizadas en el proyecto son "***soft Shadows***", un tipo de sombra muy simple y poco detallada, ideal para plataformas móviles y para no sobrecargar la memoria.

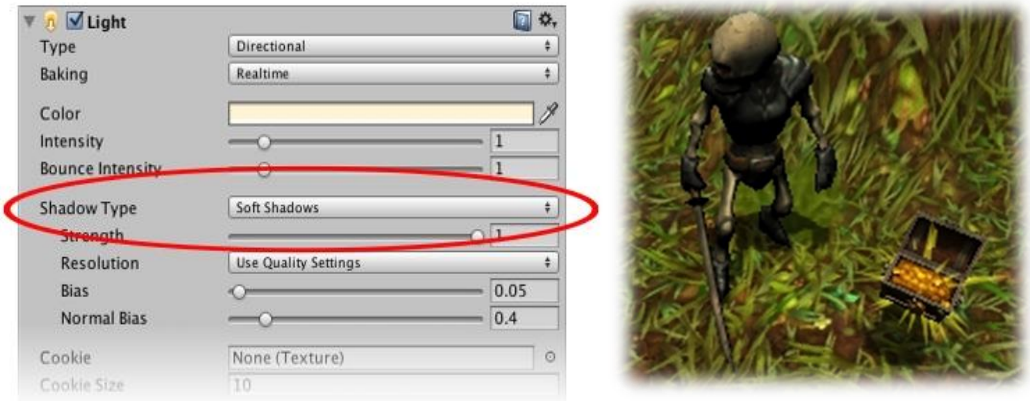


Figura 87: Optimización de la iluminación

- **Reducción del número de colliders:** Los colliders suponen los límites físicos de los objetos, con ellos se logran las colisiones entre los diferentes objetos. Los colliders ocasionan un uso de memoria, por ello en algunos casos se han agrupado diferentes objetos bajo un único collider, como por ejemplo varias casas cubiertas con un solo collider de mayor tamaño *(Ver figura 88)*, logrando un uso de memoria ligeramente menor.

Figura 88: Varios modelos 3D de casas y arboles usando un collider común.

7 Compilación del proyecto.

Cuando se dispone del proyecto terminado o se quiera realizar alguna prueba en Android es necesario compilar el archivo en formato .**apk**[66], para ello, es necesario tener instalado el **Java SDK y Android SDK**[67], para instalarlo seguimos los siguientes pasos:

7.1 Instalación SDK de Android.

Lo primero que hay que instalar es el **Java Development Kit** (conocido como JDK).

- Vamos a la web de Java (https://www.java.com/es/download/) para descargar el JDK más reciente.
- Una vez descargado lo instalamos.

A continuación, tenemos que instalar las herramientas de Android SDK.

- Para descargar el SDK Manager tenemos dos opciones: por un lado, podemos descárgalo por separado de la herramienta de programación de Android "***Android Studio"*** si nos dirigirnos a la dirección (https://android-sdk.uptodown.com/windows).

  O podemos descargarlo junto a Android Studio si nos dirigimos a la web para desarrolladores de Android (https://developer.android.com/studio/).

- En cualquiera de los casos, descargamos el Android SDK.
- Descomprimimos el archivo descargado, lo abrimos y vamos a **herramientas**.
- Buscamos el archivo llamado **Android** para ejecutarlo y proceder a su instalación.

Llegados a este punto, aparecerá una ventana emergente que muestra una lista de paquetes que se pueden instalar *(Ver figura 89)*. De forma predeterminada, se seleccionan ciertos paquetes para compilar y el paquete para la última versión del sistema operativo Android.

[66] **Apk**: Formato de compilación utilizado por las aplicaciones con sistema operativo Android
[67] **SDK**: (Software Development Kit) Constituye un conjunto de herramientas para el desarrollo de aplicaciones en una determinada plataforma.

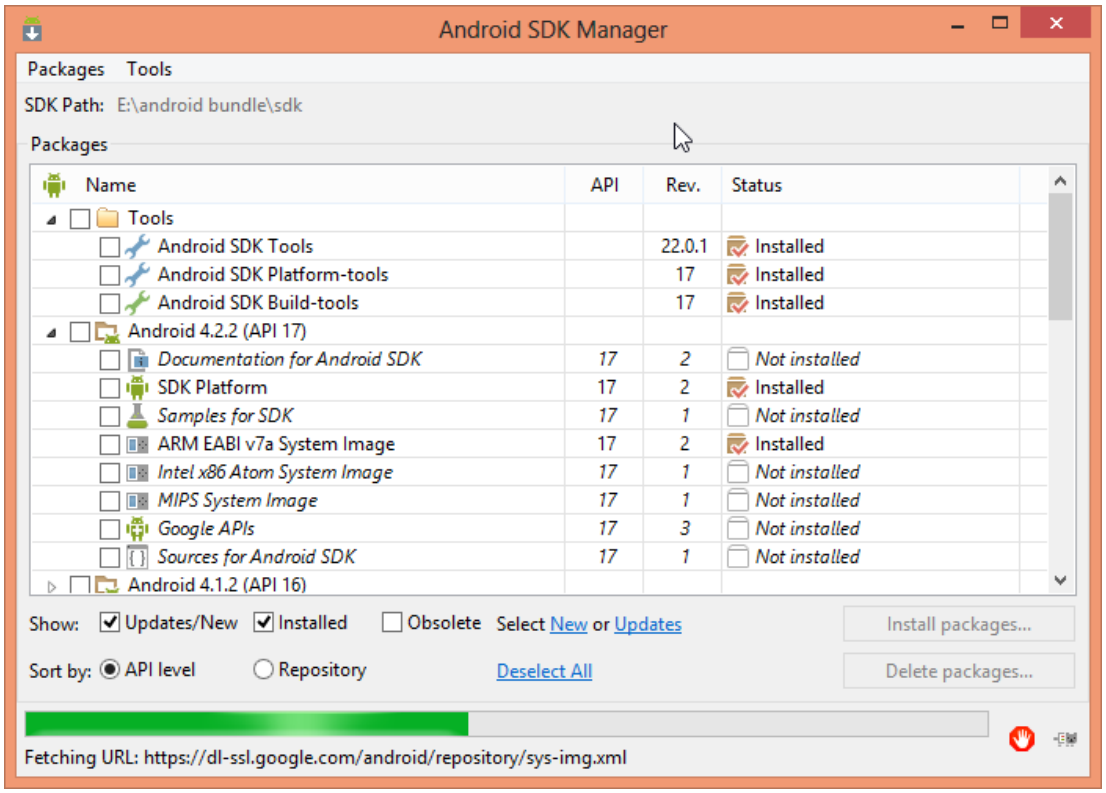


Figura 89: Instalación de las herramientas para las diferentes versiones de Android

Para este proyecto se quiere llegar a un público lo más amplio posible, pero a su vez se quiere que el videojuego se vea fluido, sin caídas de *frames* en la medida de lo posible, por ello, se establecera la versión de Android 4.4 (Kitkat) como la más baja permitida.

Para **Instalar paquetes,** se hace clic en los recuadros para comenzar el proceso de instalación. Y aceptamos las licencias de estos paquetes. Cuando terminemos aparecerá una ventana que nos muestra el proceso de instalación *(Ver figura 90)*.

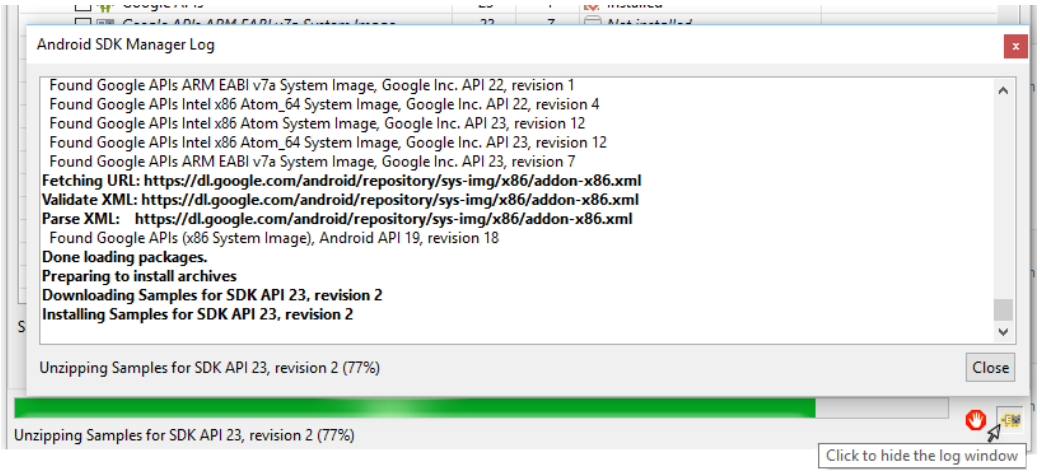


Figura 90: Proceso de instalación de Android SDK

7.2 Configuración del SDK de Android en Unity.

Finalmente, debemos decirle a Unity dónde instalamos las herramientas de Android SDK.

- Usando el menú superior, vamos a **Unity → Preferences** (en OSX) o **Edit → Preferences** (en Windows).
- Cuando se abra la ventana de Preferencias, seleccionamos **External Tools** *(Ver figura 91)*.
- Donde dice **ubicación de Android SDK**, hacemos clic en **Browse**, buscamos el directorio que contiene tanto el SDK como JDK respectivamente y hacemos clic en **Elegir**.

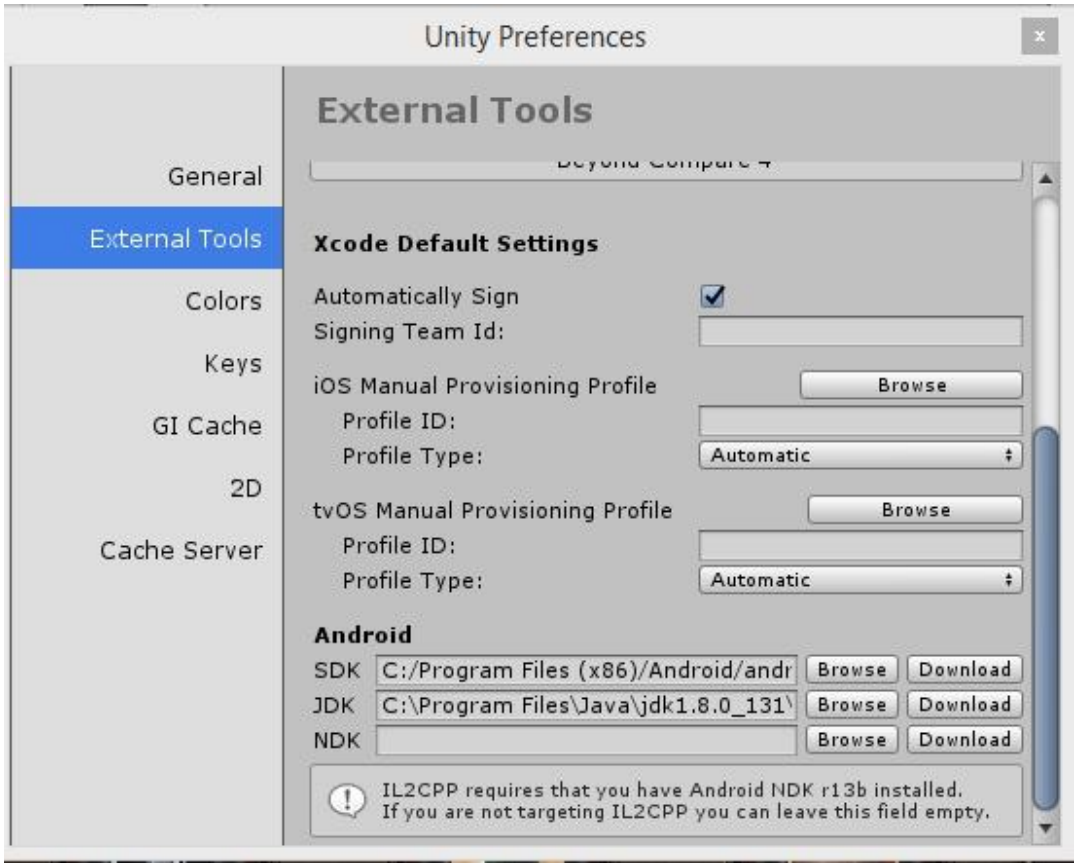


Figura 91: Asignación de las SDK y JDK en Unity

nos dirigimos a File→Build Settings *(Ver figura 92)*, aquí nos aparecerá una nueva ventana emergente en la cual tendremos que seleccionar la plataforma en la que queremos compilar nuestro proyecto *(Ver figura 93)*.

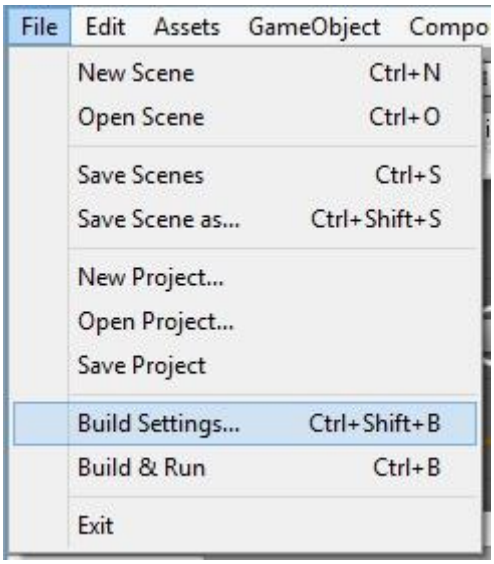


Figura 92: Menú File en Unity

Seleccionamos Android y añadimos todas las escenas de las que se compone nuestro proyecto. Y ajustamos las diferentes opciones en función de nuestras preferencias haciendo click en ***Player Settings***, aquí podremos ajustar el sistema de compresión, así como la dirección que tomará la aplicación en el dispositivo en el que se instale y la miniatura de esta cuando se instale.

Una vez que esté todo listo hacemos click en ***Build Settings*** y tras esperar unos minutos tendremos nuestra APK lista para instalar en un dispositivo Android.

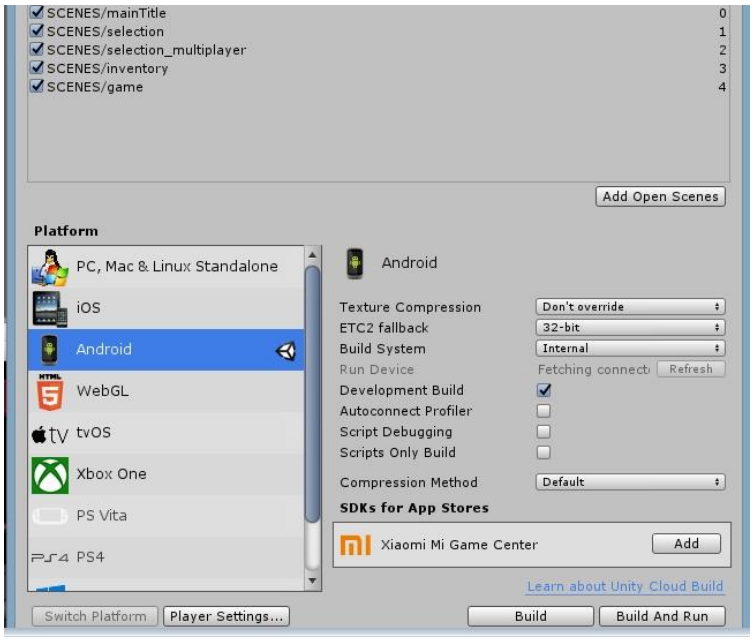


Figura 93: Ventana Build Settings

8 Planificación del proyecto.

En este capítulo se explicará el tipo de metodología que se ha escogido para realizar el trabajo y la planificación de las tareas del proyecto en formato de diagrama de **Gannt**[68]. La planificación de proyecto se divide en *sprints* y en cada sprint las tareas a realizar *(Ver figuras 94 y 95).*

En un primer momento se comentará el proceso de planificación inicial (planificación ideal) y después pasaré directamente al diagrama de Gantt que reflejará la planificación real de todo el proyecto.

También hablaremos del público al que va dirigido, así como su distribución y monetización.

Metodología Scrum

La metodología utilizada en el proyecto ha sido ***Scrum***. Scrum es una metodología de trabajo orientada en hacer *sprints* cada 3-4 semanas realizando una serie de tareas que deberán completarse. Al final de cada sprint se muestra un resumen del trabajo realizado para evaluar posibles errores y mejoras.

Planificación ideal

A continuación, se muestra de manera detallada toda la planificación que se creó al comenzar el proyecto.

Sprint 1 (7-11-17, 23-11-17)

•Documentación de Unity.

•Búsqueda de modelos 3D.

•Búsqueda de animaciones y efectos.

•Diseño del mapa del juego.

Sprint 2 (23-11-17, 5-12-17)

•Movimiento del personaje.

•Ataque básico del personaje.

•Diseño del HUD.

•Programación del HUD.

[68] **Diagrama de Gannt**: es una herramienta gráfica cuyo objetivo es exponer el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado

-Corrección de errores y bugs.

Sprint 3 (1-3-18, 20-4-18)

•Diseño del inventario.

•Control en pantalla táctil del inventario.

•Manejo de las celdas del inventario

-Corrección de errores y bugs.
-Backup[69]

Sprint 4 (22-4-18, 11-4-18)

•JSON de objetos.

•Equiparse objetos.

•Mejora de estadísticas.

-Corrección de errores y bugs.
-Backup

Sprint 5 (18-5-18, 30-5-18)

•Desarrollo de ataques a distancia.

•Ataques especiales de área.

•Ataques potenciadores.

•Ataques de sanación.

-Corrección de errores y bugs.
-Backup

Sprint 6 (1-6-18, 2-7-18)

•Tratamiento de datos con *PlayerPrefs*.

•Guardar estadísticas al finalizar partida.

•Guardado del inventario.

[69] **Backup**: Copia de seguridad realizada cada cierto tiempo con el fin de conservar los datos en caso de pérdida.

-Corrección de errores y bugs.
-Backup

Sprint 7 (5-7-18, 1-8-18)

•Inventario "*inGame*".

•Reaparición de personaje.

•Generación de objetos aleatorios.

-Corrección de errores y bugs.
-Backup

Sprint 8 (2-8-18, 16-8-18)

•Sistemas de optimización.

•Puesta a punto de assets desechados.

•Pruebas en Android.

-Corrección de errores y bugs.
-Backup

Sprint 9 (18-8-18, 7-9-18)

•Conexión a un servidor PUN

•Sala de espera de jugadores.

•Sincronización de movimiento de personaje.

•Sincronización de cámara de personaje.

-Corrección de errores y bugs.
-Backup

Sprint 10 (8-9-18, 18-9-18)

•Puesta a punto de la documentación.

•Pruebas finales.

•Presentación del proyecto.

Backups y control de implementaciones:

Puesto que el proyecto se ha realizado íntegramente a excepción de las IA en un mismo dispositivo al final de cada sprint se realizaba una copia de seguridad la cual quedaba almacenada en caso de que los archivos se corrompieran o pasara algo con el equipo.

Las implementaciones se revisaban al final de cada sprint, en un periodo de corrección de bugs, que en ocasiones se mezclaba con el propio desarrollo del sprint. Para ello se ha utilizado una escena o en ocasiones un proyecto dedicado al desarrollo para posteriormente implementar en el proyecto final.

Para sus pruebas en Android ha sido necesario el uso del Plugin ***"Lunar Console" (***ver anexo). Para poder observar la consola en busca de errores desde la propia aplicación ya instalada en el móvil, puesto que por sí mismo no contamos con consola en las aplicaciones.

El personaje principal para realizar estas pruebas ha sido el Warrior, para posteriormente una vez certificado su correcto funcionamiento implementar en el resto de los personajes.

		Nombre	Duración
1		Estudio de Unity 3D	7 days
2		Busqueda de modelos 3D y desarrollo del mapa del juego	4 days
3		Desarollo del movimiento del personaje	1 day
4		Funcionamiento de la camara	1 day
5		Desarrollo de los ataques del personaje principal y correcto	4 days
6		Diseño del HUD	1 day
7		Funcionamiento del HUD perdida de vida, manejo de maná	2 days
8		Desarrollo de un sistema de cooldown para las habilidades e	2 days
9		Sistema de inventario basado en casillas Drag and Drop	10 days
10		Separacion por tipo de objeto y casilla del inventario	8 days
11		Modificacion de la interfaz del inventario	5 days
12		Desarrollo de un menú principal y de inicio	3 days
13		Menú de login y registro	2 days
14		Menú de seleccion de personaje	3 days
15		Sistema de JSON del inventario	7 days
16		Adicion de las caracteristicas actuales al resto de personaje	7 days
17		Comunicacion de salud Jugador-Enemigo	3 days
18		Desarrollo de los ataques a distancia para todos los persona	3 days
19		Desarrollo de ataques potenciadores	3 days
20		Desarrollo de ataques en area	3 days
21		Pantalla de carga y comunicacion entre escenas	2 days
22		Modificacion de los parametros del personaje en el inventari	3 days
23		Guardado del estado de los inventarios	7 days
24		Guardado de las estadisticas de los personajes con Playerpr	3 days
25		Inventario de consumibles	7 days?
26		Sistema de reaparicion del personaje si muere	8 days
27		Objetos arrojados por enemigos al morir	4 days
28		Estadisticas "inGame"	7 days
29		Estudio de sistemas de optimizacion	1 day
30		Aplicacion de optimizaciones del juego	3 days
31		Pruebas en Android y analisis de errores	2 days
32		Correccion de errores en Android y nueva compilacion del pr	4 days?
33		Desarrollo del sistema online	15 days
34		Pruebas finales y puesta a punto	7 days

Figura 94: Planificación ideal del proyecto

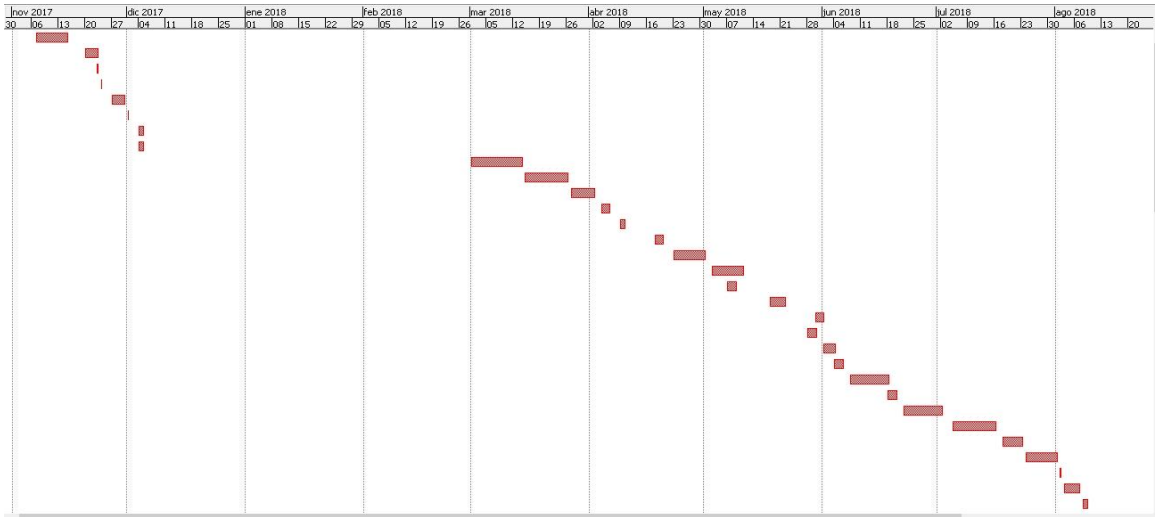


Figura 95: Diagrama de Gantt

9 Resultados finales.

El resultado final del proyecto ha sido una demostración del juego que con mayores recursos y tiempo buscamos lograr. Un juego que en un mercado tan amplio como es el de los videojuegos móviles podría abrirse camino de ampliarse su oferta jugable en forma de nuevos mapas y formas de personalización de personaje, su amigable apariencia grafica aparte de resultar llamativa para los usuarios ayuda a su mejor supervivencia en plataformas móviles donde el procesamiento y memoria es muy limitado.

Posibles mejoras y actualizaciones

En el futuro se desea ampliar el proyecto en gran medida añadiendo futuras actualizaciones entre las cuales:

- Mayor número de mapas.
- Mejora del servidor Web.
- Nuevos objetos.
- Nuevas mecánicas de juego.
- Nuevos héroes.
- Nuevos enemigos.
- Más habilidades
- Implementación de los servicios de Google Play Games.
- Añadir e invitar amigos.
- Personalización del personaje.
- Sistema de micro pagos para adquirir nuevas skins de personaje.
- Eventos en función del tiempo.
- Misiones diarias.

Mercado al que va destinado

El juego va destinado para plataformas móviles, inicialmente está pensado lanzarse para la plataforma Android en la tienda de aplicaciones de Play Store de Google.

Dependiendo del éxito de la aplicación se distribuirá hacia más plataformas como APP Store en IOS *o Windows Phone.*

El juego aún no se va a publicar, dado que hay muchas características y mejoras que quiero realizar, pero en tener una versión mejorada con algunas de las mejoras citadas posteriormente, se publicará la aplicación.

Para monetizar la aplicación se utilizarán espacios de publicidad, así como pequeños micro pagos que ofrezcan un "***Boost***[70]" de oro acorde a su precio.

[70]**Boost**: Se dice de aquellas ventajas que permiten ganar ciertos elementos en poco tiempo.

10 Conclusiones finales

Durante el desarrollo de este proyecto se han llevado a cabo un estudio del sector del videojuego, profundizando en el género RPG.

En capítulos posteriores, se explican las diferentes herramientas empleadas en el desarrollo, siendo la mas importante el motor gráfico Unity3D y sus diferentes funciones y componentes, motor que se comparará con su competencia directa Unreal Engine.

Por último, se describen todos los elementos del videojuego, estos elementos se dividen en tres puntos generales, éstos se han conseguido siguiendo una serie de objetivos:

- Modos de juego y su funcionamiento.

Los modos de juegos se fundamentan en el modo para un solo jugador, y tomando este modo como base se implementa el modo multijugador como modo secundario. Los objetivos perseguidos han sido:

-Objetivo 1: Presentar un mapa vivo, que presente objetos que recoger y visualizar que evolucione conforme se avanza en la partida.

-Objetivo 2: Permitir al jugador modificar su estado mediante la compra y uso de diferentes objetos durante el desarrollo de una partida.

- Desarrollo de una partida y jugabilidad según el personaje seleccionado.

Los videojuegos RPG se basan en partidas dinámicas que varían en función del personaje que esté usando el jugador, para ello se han perseguido los siguientes objetivos:

-Objetivo 1: Buscar mecánicas para que los jugadores recorran todo el mapa, sin saltarse ninguna zona y a su vez no dar la sensación de estar ante un videojuego lineal.

-Objetivo 2: Introducir varios personajes que modifiquen la forma de jugar y la reusabilidad del mismo fomenta la buena aceptación por el jugador.

- Tratamiento de los datos guardados.

El uso de datos es muy importante en el género RPG, por ello se han buscado los siguientes objetivos:

-Objetivo 1: Permitir al jugador editar y visualizar las características del personaje mediante el equipo de objetos.

-Objetivo 2: Recopilar información del progreso del jugador, permitiéndole guardar sus progresos y visualizarlos en la web.

Desde un punto de vista personal el desarrollo del proyecto ha sido muy constructivo y motivador, aunque también complejo, en especial para el modo multijugador. He podido aplicar conocimientos aprendidos sobre el sector del videojuego, tanto como jugador como desarrollador. En ocasiones se desecharon ideas por limitaciones de todo tipo, pero también se han sacado aspectos adelante que creía imposibles tiempo atrás. Acciones específicas, que en el modo para un solo jugador no requiere de grandes esfuerzos de programación, se convierten en todo un reto cuando se ejecuta en red, ejemplo de esto puede ser comunicarle al servidor cuando se ejecutan las diferentes animaciones del personaje.

Para concluir, decir que pese a lo complicado del proyecto poder verlo en pleno funcionamiento, ha supuesto para mí una gran gratificación que hace que haya merecido la pena el esfuerzo.

11 Glosario.

- **A**

  Asset: Diferentes activos que pueden ser usados y modificados en el motor gráfico.
  Atari: es una de las productoras de videojuegos independientes más grande en Estados Unidos, comparte el mismo nombre que su primera consola.
  Awake (): Función propia de la librería de Unity, encargados de ejecutar su contenido al principio de la ejecución y después de cada frame respectivamente.
  Apk: Formato de compilación utilizado por las aplicaciones con sistema operativo Android.
  Autorriger: Proceso automatizado consistente en añadir "Huesos" a los modelos 3D con el fin de que puedan ser asociados a una animación.
  Actors: Nombre que se da en Unreal Engine a cualquier objeto en el juego.

- **B**

  Backup: Copia de seguridad realizada cada cierto tiempo con el fin de conservar los datos en caso de pérdida.
  Bugs: Errores inesperados dados durante el juego, que depende de una combinación concreta de factores para hacerse visibles.
  Brookhaven National Laboratory: es un laboratorio nacional del Departamento de Energía de los Estados Unidos ubicado en Upton, en Long Island. Está especializado en investigaciones sobre física nuclear.
  Blueprints Visual Scripting: Sistema de Unreal Engine para hacer scripting basado en una interfaz de nodos unidos para crear elementos de juego.
  Boost: Se dice de aquellas ventajas que permiten ganar ciertos elementos en poco tiempo.

- **C**

  CD-ROM: (sigla del inglés Compact Disc Read-Only Memory) es un disco compacto con el que utilizan rayos láser para leer información en formato digital.
  Consola de 32 bits: Nombre que se le da a la quinta generación de consolas, la cual supuso el paso de los 2D a los entornos tridimensionales 3D.
  Computadora TRS-80: designación para varias líneas de sistemas de microcomputadores producidos por Tandy Corporation.
  Corrutina: Al llamar a una función, la ejecutamos en su totalidad antes de retornar. Esto significa que cualquier acción tomando lugar en una función debe suceder en una sola actualización de frame,
  Canvas: Componente de Unity designado para mostrar la interfaz de usuario, actúa de GameObject padre para los elementos de la UI.

- **D**

  Dungeons & Dragons: juego de rol de fantasía heroica originalmente derivado de juegos de tablero jugados con lápiz, papel y dados.
  DVD (Digital Versatile Disc): El DVD es un tipo de disco óptico para almacenamiento de datos.

Draw call: Llamado de dibujo del motor gráfico al API de gráficas del equipo en el que está corriendo.
Diagrama de Gannt: es una herramienta gráfica cuyo objetivo es exponer el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado.

- **E**

  EDSAC: Antigua computadora británica.
  Escena: En Unity son los apartados que contienen todos los demás objetos del juego.
  Enfriamiento(cooldown): Estado en el que mediante una temporización se bloquea una acción.

- **F**

  Formato ".obj": Formato para la representación de modelos 3D, consistentes en el guardado de coordenadas.

  Framerate: Numero de imágenes por segundo que son mostradas en pantalla.

  Fuse: Aplicación similar a Mixamo encargada del desarrollo asistido de modelos 3D de personajes.

- **G**

  GameObject: Objetos que componen una escena en Unity.
  Gameplay: Conjunto de acciones que puede realizar un jugador para interactuar con juego o la forma en la que este interactúa con el propio jugador.

- **I**

  Input: Valor de entrada obtenido por un componente o script.
  In Game: Se dice de aquellos elementos que aparecerán dentro de una partida.

- **J**

  Joystick: es un periférico de entrada que consiste en una palanca que gira sobre una base e informa su ángulo o dirección al dispositivo que está controlando.

- **L**

  Lobby: Espacio de reunión y espera de los jugadores antes de empezar una partida.
  Low poly: Nombre que se le da a aquellos modelos 3D que no cuentan con gran cantidad de polígonos.

- **M**

  Modelo 3D: Se llama al producto fruto del desarrollo de una representación matemática de cualquier objeto tridimensional (ya sea inanimado o vivo) a través de un software especializado.
  Maná: Nombre que se da en juegos de fantasía a los puntos necesarios para usar una habilidad.

Mitchell: Algoritmo de compresión de imágenes encargado de reducir el peso de las texturas más pequeñas de la escena para realizar un menor consumo de procesamiento.

- **O**

  Objetos preconstruidos: Objetos básicos ofrecidos por el editor, tales como figuras geométricas simples en 3D, luces, o cámaras.

- **P**

  Prefab: tipo de asset de Unity que permite almacenar uno o varios GameObject completamente incluyendo sus componentes y propiedades.
  Parenting: Concepto utilizado en Unity para asociar GameObjects en la jerarquía, consistente en establecer una relación objeto Padre-Hijo.
  Physics Materials: Componentes asociados a las texturas de un objeto 3D que le da propiedades al tacto tales como deslizamiento, rigidez, etc.
  Plugin: Herramienta que se puede adicionar al editor para obtener ciertas características extra.
  Playerprefs: Herramienta empleada por Unity para programar guardado de datos en registros.

- **Q**

  Quest: Nombre que se le da a las misiones o tareas dentro de un juego RPG.

- **R**

  Random Drop: Nombre del script encargado de generar objetos aleatorios al morir el personaje.

- **S**

  Sistema de combate a tiempo real: Sistema de juego consistente en que el tiempo transcurre de igual manera para todos los bandos en el combate, no existiendo parones.
  Sprites: Son esencialmente unas texturas estándar o en algunos casos, texturas especiales combinadas por temas de eficiencia y conveniencia durante el desarrollo.
  Sistema de partículas: Asset de Unity consistente en generadores de pequeños sprites y luces para generar fluidos, o flujos animados, tales como el fuego o chorros de agua.
  Slider: Componente de Unity utilizado para realizar barras con aspecto y comportamiento variable mediante programación.
  Shader: Componente que se asocia a las texturas y le añade ciertos efectos.
  Stock: Cantidad de objetos de un mismo tipo almacenados en un mismo lugar.

- **T**

  Transform: Componente de Unity encargado de localizar en unas coordenadas concretas los objetos en el mundo.
  Terrain: Herramienta de desarrollo de mapeado en 3D incluida en Unity

Trigger: Función de la clase collider que en función de si está activado o no, envía información sobre colisiones, o estados de RigidBody que esté en contacto con el collider en cuestión.
Tag: Nombre que se da a las etiquetas en Unity.
Text Asset: Función empleada para el manejo de texto de Unity.

- **U**

Unreal Engine: Es un motor de juego de PC y consolas creado por la compañía Epic Games.
UI: Acrónimo en inglés de "*User Interface*".
Update (): Función propia de la librería de Unity, encargados de ejecutar su contenido al principio de la ejecución y después de cada frame respectivamente.

- **W**

WYSIWYG: Acrónimo de *What You See Is What You Get* (en español, "lo que ves es lo que obtienes"), es una frase aplicada a los procesadores de texto y otros editores de texto con formato (como los editores de HTML) que permiten escribir un documento mostrando directamente el resultado final.

- **X**

Xcode: es un entorno de desarrollo integrado para macOS que contiene un conjunto de herramientas creadas por Apple destinadas al desarrollo de software para macOS, iOS, watchOS y tvOS.

12 ANEXO

12.1 •Géneros de los RPG

- **A-RPG o RPGs de acción:** Son aquellos donde podemos tomar el papel de un sólo personaje en tiempo real y el sistema de diálogo se mantiene en lo mínimo, tomando más énfasis en la acción y combate como en la serie "***Tales of"*** de Bandai Namco.
- **RPS o Rol Playing Shooter:** En esta categoría se encuentran los FPS que en su jugabilidad incorporan mecánicas de los RPG, tales como el diálogo, los NPC (personajes no jugables), sistema de inventario o habilidades, entre los cuales podemos clasificar a "***Deus Ex", "Mass Efect", "Borderlands"*** y ***"Fallout".***
- **S-RPG o RPG táctico:** Son juegos que se basan el combate en escenarios con cuadrículas, donde el valor táctico es fundamental forzando al jugador planear una estrategia previa para poder ganar cada combate desarrollado. En este subgénero podemos encontrar títulos como "***Final Fantasy Tactics"***, "***Fire Emblem"***, "***Disgaea"*** y "***Valkyria Chronicles".***
- **MMORPG o RPG multijugador:** Nacidos en la época de los 90's con la proliferación del Internet, aunque sus inicios fueron desarrollados en "***Secret of Mana***" que contaba con la opción de multijugador cooperativo. Luego este subgénero tendría el boom actual con Diablo, el cual permitía que los jugadores pudieran conectarse en grupos para derrotar monstruos y comprar objetos en línea.

A pesar de que el subgénero goza de una popularidad tremenda, tiene muchos detractores al decir que está diluida la esencia de los RPG, esto debido a que un RPG solitario nos invita a ser héroes de la historia, pero en los MMORPG no todos llegan a ser el héroe. De hecho, esto se vio muy retratado en el juego "***EverQuest***", en el cual los jugadores no podían obtener todos ítems poderosos, por lo cual los jugadores empezaron a hacer camping, robo de muertes y robo de botines. Aun así, en este subgénero podremos encontrar títulos como "*World of Warcraft",* "*Elder Scrolls Online*" y "*Ragnarok Online".*

12.2 Movimiento pulsación-desplazamiento.

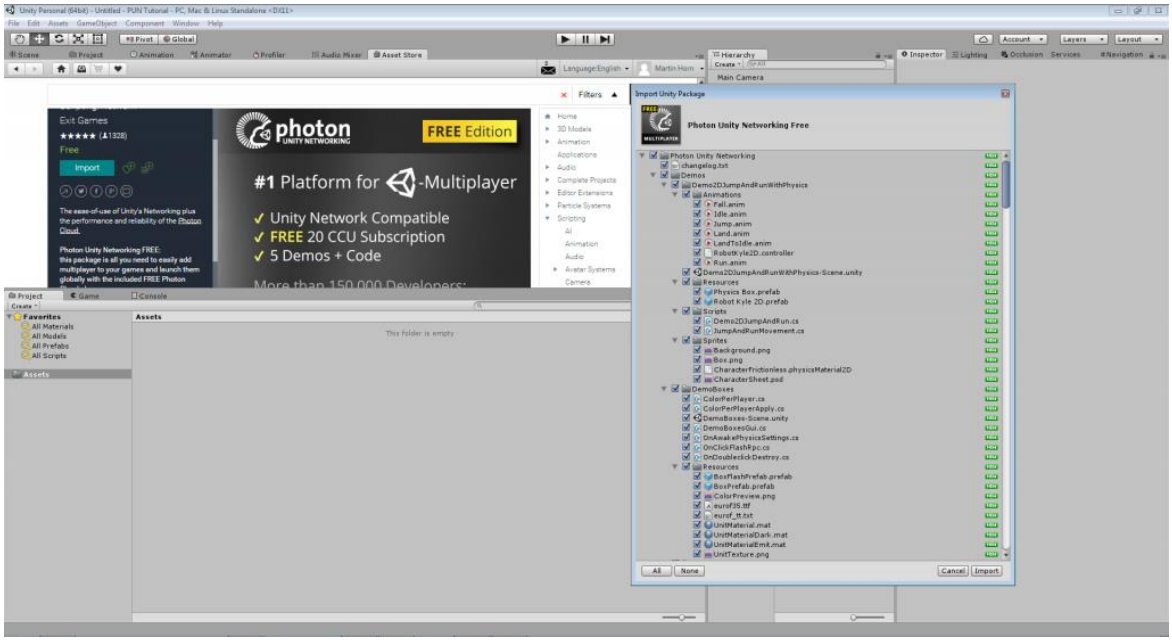
Forma de movimiento del personaje sin joysticks consistente en que el desplazamiento se realiza pulsando un punto de la pantalla al cual el personaje se desplazará y se pondrá en modo reposo esperando una nueva pulsación para desplazarse.

12.3 Instalación de Phtoton Unity Networking.

Parte 1a: crear el proyecto y descargar e instalar el SDK de PUN

Lo primero que debemos hacer es crear un nuevo proyecto.

Una vez que creado el nuevo proyecto y lo abramos en Unity, vamos a la Asset Store y buscamos ***Photon Unity Networking***. La primera coincidencia debe ser la versión gratuita de Photon Unity, por lo tanto, hacemos clic en el botón Descargar y, cuando termine la descarga, clic en el botón importar en la ventana del almacén de activos. Cuando se abre el cuadro de diálogo de importación en Unity, nos aseguramos de que todo está seleccionado y luego hacemos clic en el botón importar.



Dependiendo de la velocidad del PC, tomará uno o dos minutos importar todo, y una vez hecho esto, aparecerá el diálogo del asistente PUN de la siguiente manera:

Ingresamos el correo electrónico que deseamos utilizar en el cuadro etiquetado como **AppID** o **Correo electrónico** y hacemos clic en el proyecto de configuración. Debe responder con un mensaje que nos avise de que el correo electrónico no está registrado y le da la opción de abrir el panel de la nube.

Figura 96: Registro de cuenta de PUN

Hacemos clic en el botón Iniciar sesión en el ***Cloud Dashboard*** y se abrirá la página de inicio de sesión de Photon.

Hacemos clic en el botón ***Registrar***, luego ingresamos la dirección de correo electrónico que deseamos registrar y hacemos clic en el botón naranja *Registrar*. Se nos enviará un **correo electrónico** que contiene un **enlace de activación**, en el que debe hacer clic para configurar la contraseña de su cuenta y comenzar a usar Photon.

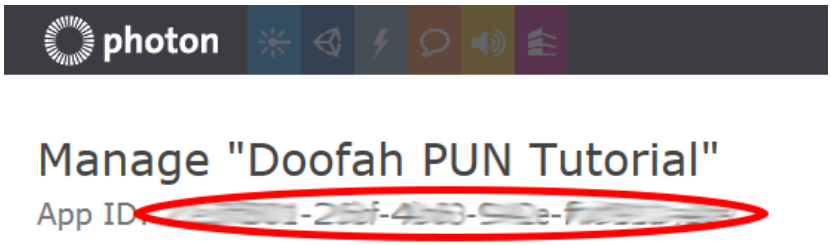
Después de haber creado su contraseña y haber activado su cuenta Photon, volvemos a Unity y hacemos clic en el botón *Iniciar sesión* en el *panel de* la *nube* en el cuadro de diálogo. Esto abrirá una ventana del navegador donde podemos iniciar sesión en Photon con el correo electrónico y la contraseña que especificamos en los pasos anteriores.

Una vez que iniciada sesión podremos agregar y administrar todas nuestras aplicaciones de red con PUN.

Hacemos clic donde aparece el nombre de nuestra aplicación e ingresamos un nuevo nombre. No tiene que ser el mismo nombre que el del proyecto de Unity, pero mantenerlo similar facilitará las cosas más adelante si tenemos muchos proyectos de red .

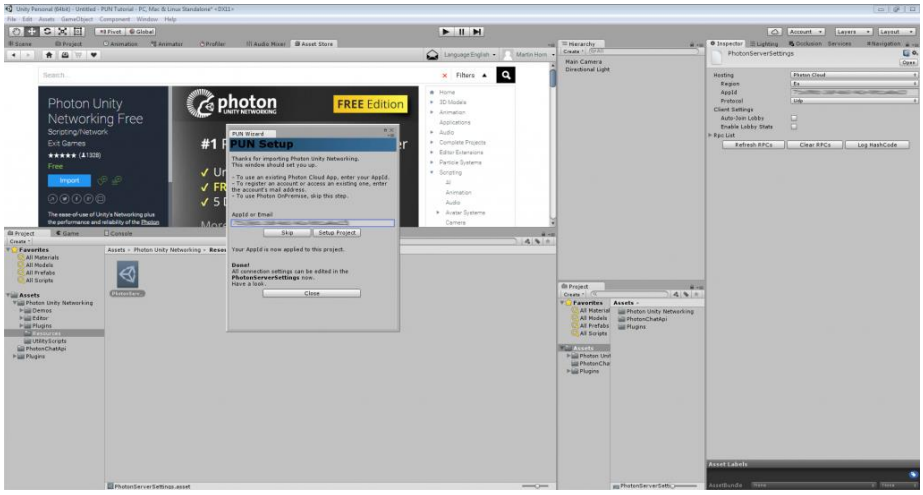
Una vez cambiado el nombre de la aplicación y hayamos hecho clic en guardar, seremos llevados a una nueva pantalla:

Casi hemos terminado en el panel de Photon por ahora, todo lo que tenemos que hacer es copiar en el portapapeles la **ID** de la aplicación, la cual es una ristra larga debajo del nombre de la aplicación, para que podamos pegarla en Unity.



Por lo tanto, al haber seleccionado y copiado el ID de la aplicación, volvemos a Unity y lo pegamos en el cuadro etiquetado como **AppID** o Email, luego hacemos clic en el botón: ***Configurar proyecto***

En este punto, el proyecto debería estar configurado para la conexión en red con Photon, y la configuración del servidor de Photon debería seleccionarse automáticamente, listo para que usted los edite. Por ahora, la única opción que queremos cambiar es la región, hacemos clic en el menú desplegable y seleccionamos Europa.



Así que eso es todo para la configuración del proyecto y para que esté listo para la creación de redes Photon.

12.4 Lunar console.

Este es un plugin consistente en una consola iOS / Android nativa para Unity de alto rendimiento y peso liviano para facilitar las pruebas y la depuración, este plugin es necesario puesto que en la plataforma Android pueden surgir diferentes problemas que no son reflejados en el proyecto mientras está en el editor.

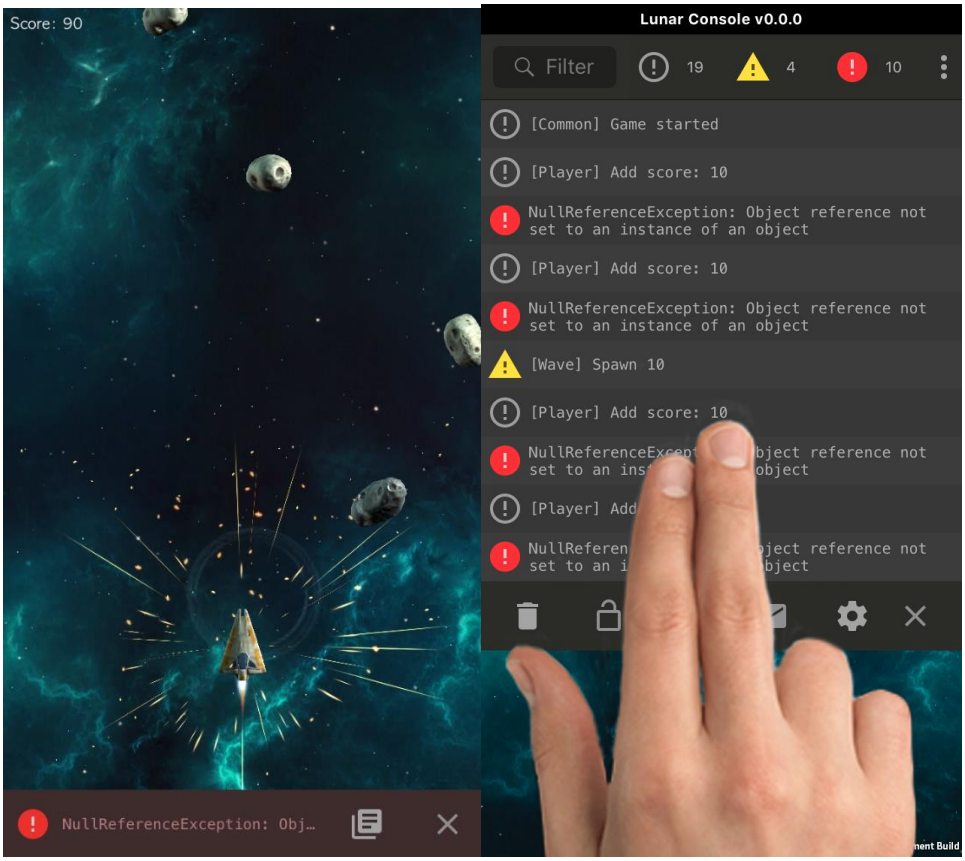


Figura 97: Captura del funcionamiento de Lunar Console en un dispositivo móvil.

12.4.1 Instalación

- **Automatic**:
  Unity Editor Menú: Window ▶ Lunar Mobile Console ▶ Install...
- **Manual**:
  Coger y arrastrar el prefab LunarConsole localizado en: **(Assets/LunarConsole/Scripts/LunarConsole.prefab)** en la jerarquía de la escena y guardar los cambios.

  Este proceso solo es necesario en la escena inicial del proyecto**.**

12.4.2 Activación:

- Comprobaremos que el **GameObject Lunar Console** se encuentra en la jerarquía de la aplicación.

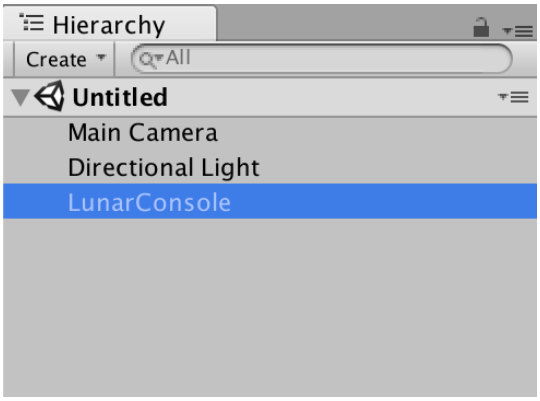


Figura 98: Jerarquía con lunar console.

- Finalmente, el apartado de compilación de la aplicación debemos marcar la casilla "***Development Build***" para que el plugin sea visible y pueda ser utilizado.

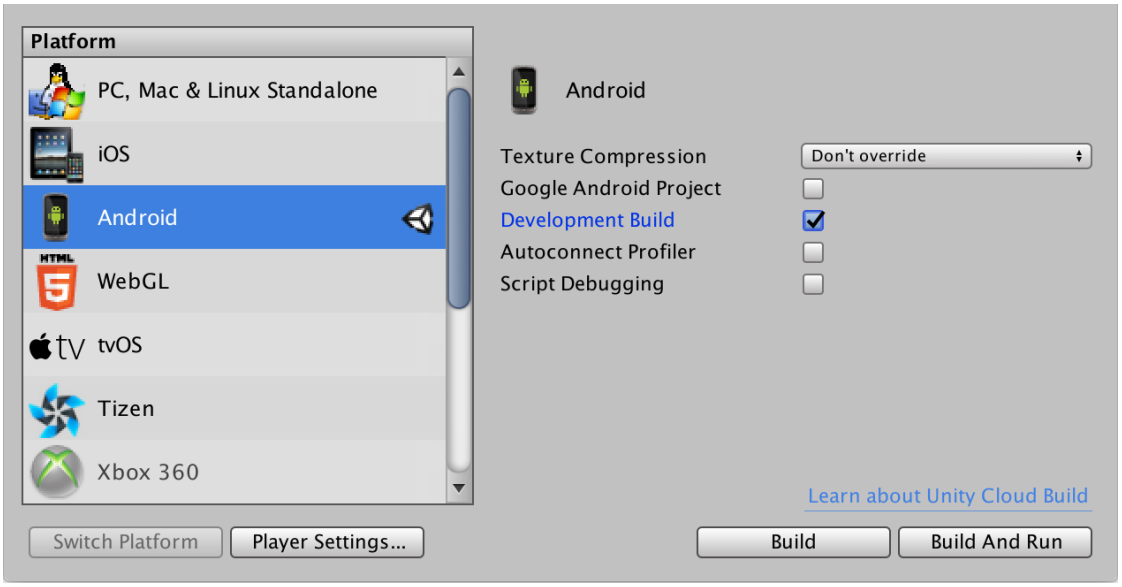


Figura 99: Activación de Development Build

- Llegados a este punto realizando el gesto de deslizar dos dedos sobre la pantalla, haremos aparecer la consola en un dispositivo Android o IOs para realizar el debug de la aplicación.

13 Bibliografía.

13.1 Bibliografía impresa.

Nieto Jiménez, Gabriel. Unity 4x Tutorial de iniciación al desarrollo de videojuegos.

Okita, Alex. Learning C# programing for Unity3D. CRC Press

Vallejo, David. Desarrollo de videojuegos, arquitectura del motor. CreateSpace Independent Publishing Platform,

Sapio, Francesco. Unity UI cookBook. Packt Publishing

13.2 Bibliografía digital.

https://www.yeeply.com/blog/desarrollo-de-juegos-con-unity-3d/

https://docs.unity3d.com/es/current/Manual/class-AudioSource.html

https://www.sacredseedstudio.com/tutorials/interacting-with-json-in-unity/https://www.udemy.com/mi-primer-juego-con-unity-5/learn/v4/

https://www.udemy.com/introduction-to-game-development-with-unity/learn/v4/overview

https://stackoverflow.com/questions/46003824/sending-http-requests-in-c-sharp-with-unity

http://doc-api.photonengine.com/en/PUN/current/general.html

https://doc.photonengine.com/en-us/pun/current/demos-and-tutorials/lagcompensation

https://docs.unity3d.com/es/current/Manual/ScriptingSection.html

https://answers.unity.com/questions/131511/playerprefs-storage-location-on-android-or-ios.html

https://unity3d.com/es/learn/tutorials/topics/scripting/testing-unity-games-android-visual-studio

https://answers.unity.com/questions/13047/proper-way-to-export-a-terrain.html

http://www.unityspain.com/topic/37429-shader-outline/

https://www.assetstore.unity3d.com/

https://github.com/SpaceMadness/lunar-unity-console#about

http://www.doofah.com/tutorials/networking/pun-tutorial-part-1/

https://www.fib.upc.edu/retro-informatica/historia/videojocs.html