RÍO NEGRO
UNIVERSIDAD NACIONAL

Licenciatura en Sistemas

Análisis del proceso de remake de un videojuego con mecánicas de realidad aumentada

Alumno: Wainmaier Sandro Leonel

Director: Ing. Garcia Martinez Nicolas

Año: 2021

Resumen

En el presente trabajo se expone el proceso de “Remake” de un videojuego, que consiste tomar como base un producto desarrollado previamente y utilizarlo para un nuevo proyecto que analice las características del original y tenga como objetivo realizar una mejor versión del videojuego. Se propone un desarrollo que toma como base el proyecto original “Invasión Aumentada”, desarrollado en el Laboratorio de Informática aplicada en 2014, y partir de este se desarrolla un nuevo proyecto con las mejores características del proyecto original y las adaptaciones necesarias para cumplir con los estándares actuales tanto a nivel tecnológico como jugable.

Inicialmente se analiza el proyecto original de 2014 en detalle, desde el enfoque del diseño actual de videojuegos, se rescatan las características que se mantendrán en el nuevo proyecto y se definen cuáles son las nuevas que se deben incluir. En la etapa de desarrollo se analizan qué técnicas de la ingeniería de software se adaptan mejor al desarrollo de videojuegos y luego se aplican estas mismas para desarrollar la solución.

Palabras claves: Realidad Aumentada (RA), Game Design, Desarrollo de videojuegos, Unity, Android, IOS, XR

Índice

I. Introducción

En el año 2015, como culminación de una investigación de las tecnologías de realidad aumentada y geolocalización en dispositivos móviles, se publicó el videojuego “Invasión Aumentada” en la plataforma Google Play (Android). El desarrollo presentaba mecánicas[1] Arcade[2] típicas de los juegos de esta plataforma pero con la integración de la realidad aumentada como parte de la propuesta, lo que permitía que la posición de la cámara y los espacios reales formarán parte de la experiencia del videojuego.

Este trabajo final de carrera se pone como objetivo tomar la idea original del juego “Invasión Aumentada” pero partir desde un proceso de diseño de videojuegos para obtener un producto final más terminado y aplicando las mejores prácticas del desarrollo de software. En el proyecto de 2015 existían varias inconsistencias en la estética y en el diseño del juego, pero también había mecánicas de juego innovadoras como el lanzamiento de proyectiles que, a partir de la dirección del dispositivo y la interacción con la pantalla, permitía definir la dirección y profundidad de dichos proyectiles y que el usuario interactuara con el entorno 3D del juego.

En el mundo de los videojuegos el proceso de tomar un juego publicado anteriormente y agregarle nuevas características y mejorarlo estéticamente para las plataformas actuales se llama “remake”. Implica un proceso nuevo de creación artística y técnica que va más allá de tomar el juego anterior y mejorarlo visualmente. Un remake se propone el rediseño y la actualización de las mecánicas y estética con el fin de que el juego se adapte a los tiempos actuales.

Este trabajo se propone llevar adelante el “remake” de “Invasión Aumentada” a partir del análisis del proceso de diseño, desarrollo y publicación de un videojuego. El resultado esperado es un videojuego de realidad aumentada que utiliza las características actuales de la tecnología, como la detección de planos, y que tiene un criterio de diseño más uniforme.

---

[1] Una mecánica de juego es una estructura que define la manera en la que el jugador interactúa con el juego, con él fin de brindar una experiencia de juego satisfactoria.

[2] Un videojuego estilo Arcade es aquel que está diseñado siguiendo los principios básicos de los antiguos juegos para máquinas Arcade. Sus características principales son un diseño sencillo y controles fáciles de asimilar, niveles cortos y de dificultad ascendente

II. Estado de la cuestión

II.1. Videojuegos

A principios de la década del 70, la aparición del videojuego Pong (Atari) causaba furor en los lugares donde se habían instalado las primeras máquinas para jugarlo (bares y tiendas) y posteriormente influiría en la aparición de salas de entretenimiento dedicadas exclusivamente a máquinas de videojuegos. Se dice que "así nació el videojuego como producto de cultura de masas" (Kent, 2001: 43-44).

Fig 1. Máquina de Pong (Atari 1972)

A partir de ese momento la industria tendrá un crecimiento casi constante. La aparición de las consolas permitió diversificar más la industria, ya que los juegos empezaron a venderse por separado, antes las máquinas tenían los juegos precargados y no había manera de adquirir nuevos. Con la posibilidad de vender los juegos por separado no sólo creció el mercado, sino que además el negocio pasó a ser vender los juegos y no la consola, lo que le permitió tener precios más competitivos.

Fig 2. La Consola Atari 2600 (1977) con cartuchos

Esto que parecía positivo en un principio terminó casi destruyendo toda la industria. En 1982 la ambición desmedida de Atari llevaría a lanzar más títulos de los que se demandaban y con un control de calidad muy bajo. El lanzamiento del videojuego E.T. (basado en la película homónima de Steven Spielberg) fue la gota que rebalsó el vaso. La gente perdió la fe en los videojuegos y los inversores le dieron la espalda a las empresas, aludiendo que los videojuegos eran una moda pasajera. Millones de copias de E.T. sin vender terminaron enterradas en el desierto, una metáfora ideal del estado de la industria a principios del 83. La era de Atari había llegado a su fin.

Durante casi todo el resto de la década del 80, el desarrollo de videojuegos quedó en manos de jóvenes entusiastas que, gracias a la aparición de computadoras personales básicas pero potentes como la Spectrum ZX, pudieron programar sus propios videojuegos. Así le dieron nueva vida a la industria y permitieron que aparecieran empresas independientes que experimentaron y crearon nuevos géneros y mecánicas innovadoras para la época. A finales de los 80 las consolas volvieron a dominar el mercado, con la aparición de Nintendo en la industria. La NES (Nintendo Entertainment System), gracias a componentes de hardware adicionales, como una pistola láser o juguetes interactivos, recuperó la confianza de la gente en la industria de las consolas. A partir de allí la industria creció constantemente, las consolas evolucionaron y adaptaron los avances tecnológicos a su formato, como la utilización de CDs o la conexión a internet y los juegos en PC también evolucionaron con cada avance de hardware..

La aparición de juegos móviles volvería a cambiar completamente la industria, si bien algunas empresas habían lanzado consolas portátiles, como el Gameboy de Nintendo o los dispositivos Tamagotchi de Bandai popularizados en

finales de los 90, ninguno de estos dispositivos tendría el impacto que tuvo la aparición de los teléfonos inteligentes. Estos dispositivos significaron la inclusión a la industria de millones de personas que normalmente no hubieran accedido a los videojuegos en los formatos tradicionales. El gran impacto que esto tuvo en el crecimiento de la industria se refleja en el hecho de que hoy en día es la industria del entretenimiento que genera más ingresos. En el siguiente gráfico podemos ver la comparación con la industria de la música y también se ve claramente el impacto de la aparición de los juegos para teléfonos inteligentes.

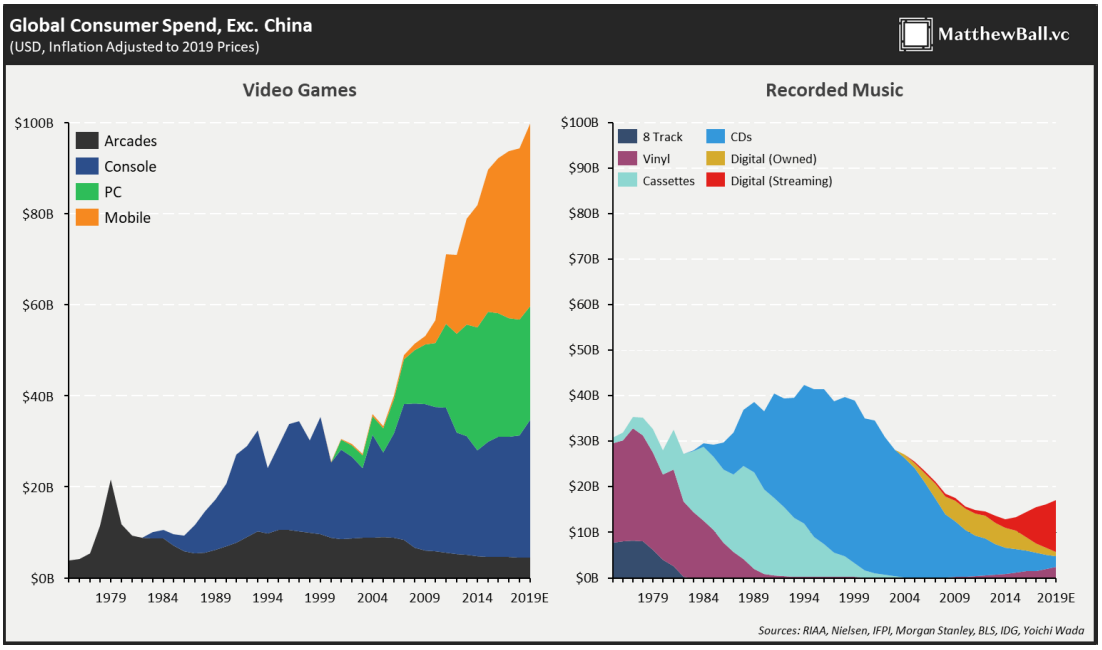

Fig 3. Comparación entre la industria de los videojuegos y la de la música

II.2. Motores de videojuegos

Como todo producto de software, en los videojuegos los componentes de un proyecto se reutilizan para otros proyectos con características similares. En el caso de los videojuegos, por ejemplo el motor de renderizado[3] o la librería de físicas[4] son potencialmente reutilizables. A finales de los 80 nacieron los primeros motores, que las empresas utilizaban para desarrollar videojuegos con las mismas bases. Un ejemplo puede ser SCUMM ("Script Creation Utility for Maniac Mansion", en español

[3] Motor de Renderizado: software que transforma información de una escena en imágenes, en el caso de los videojuegos se renderiza en tiempo real, es decir que el motor debe renderizar una tasa determinada de imágenes por segundo, para brindar una imagen fluida.

[4] Librería de Física: conjunto de funciones que le permiten al juego emular la física del mundo real, teniendo en cuenta, por ejemplo, gravedad, masa, fricción, restitución, etc.

"Utilidad de creación de scripts para Maniac Mansion"). Se encuentra a medio camino entre un lenguaje de programación y un motor para videojuegos de aventuras gráficas. SCUMM, considerado como una de las primeras aventuras gráficas de la historia de los videojuegos, fue creado por Lucasfilm Games (después conocida como LucasArts) y desarrollado por Aric Wilmunder y Ron Gilbert para su videojuego Maniac Mansion en 1987. Fue el motor de aventuras gráficas más popular en los ochenta y primeros noventa y tuvo en AGI y SCI de Sierra On-Line a sus principales competidores.

SCUMM permitía a los diseñadores de juegos crear lugares, objetos y secuencias de diálogos sin necesidad de escribir en el mismo lenguaje con el que se había escrito el código fuente del juego.

Fig 4. Dos juegos hechos en base a SCUMM Maniac Mansion (1987 Lucas Arts) e Indiana Jones and The Last Crusade (1989 Lucas Arts).

Con la aparición de los juegos 3D se volvió más común utilizar un mismo motor de renderizado para diferentes juegos y se empezaron a sumar herramientas, como editores de escenas, que permiten ubicar contenido del juego en el escenario 3D. Por ejemplo, el motor del videojuego Unreal (Epic Games 1998), agrupa las características y herramientas para hacer un juego de disparos en primera persona pero, al ser liberado, los desarrolladores encontraron la manera de hacer otros tipos de juegos utilizando las herramientas que brinda este motor.

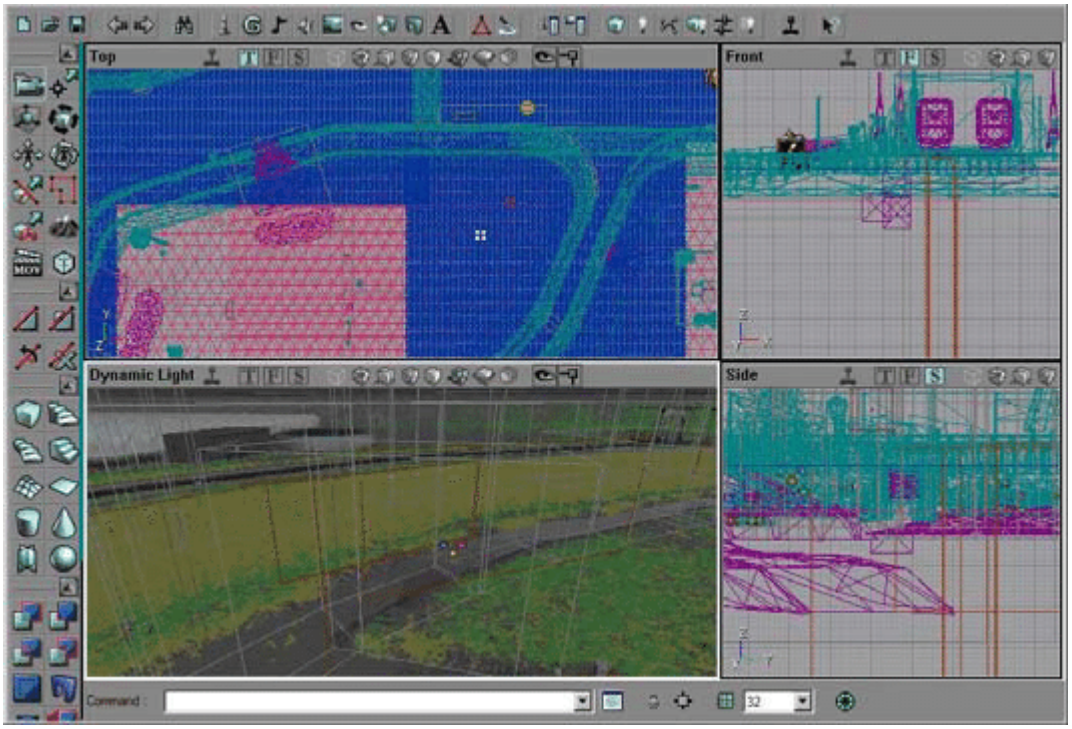

Fig 5. Primera versión del motor Unreal Engine (Epic Games 1998).

Actualmente existen muchos motores que se adaptan a las necesidades de los desarrolladores y brindan soporte para juegos 3D/2D en la mayoría de los lenguajes que se utilizan hoy en día. En algunos casos permiten crear juegos mediante bloques o interfaces, sin necesidad de conocer ningún lenguaje de programación en particular. Las herramientas que brindan también son mucho más específicas: un motor 2D generalmente tiene editor de sprites (Fig 6), tiles dinámicos (Fig 7) y un editor de niveles integrado (Fig 8), en cambio un motor 3D tiene editores de niveles que soportan modelos tridimensionales, editores de materiales y texturas y configuración de iluminación global.

Fig 6. Las Sprites son imágenes 2D que contienen las animaciones de los objetos/personajes. (The Legend of Zelda: A Link to the Past, Nintendo 1991)

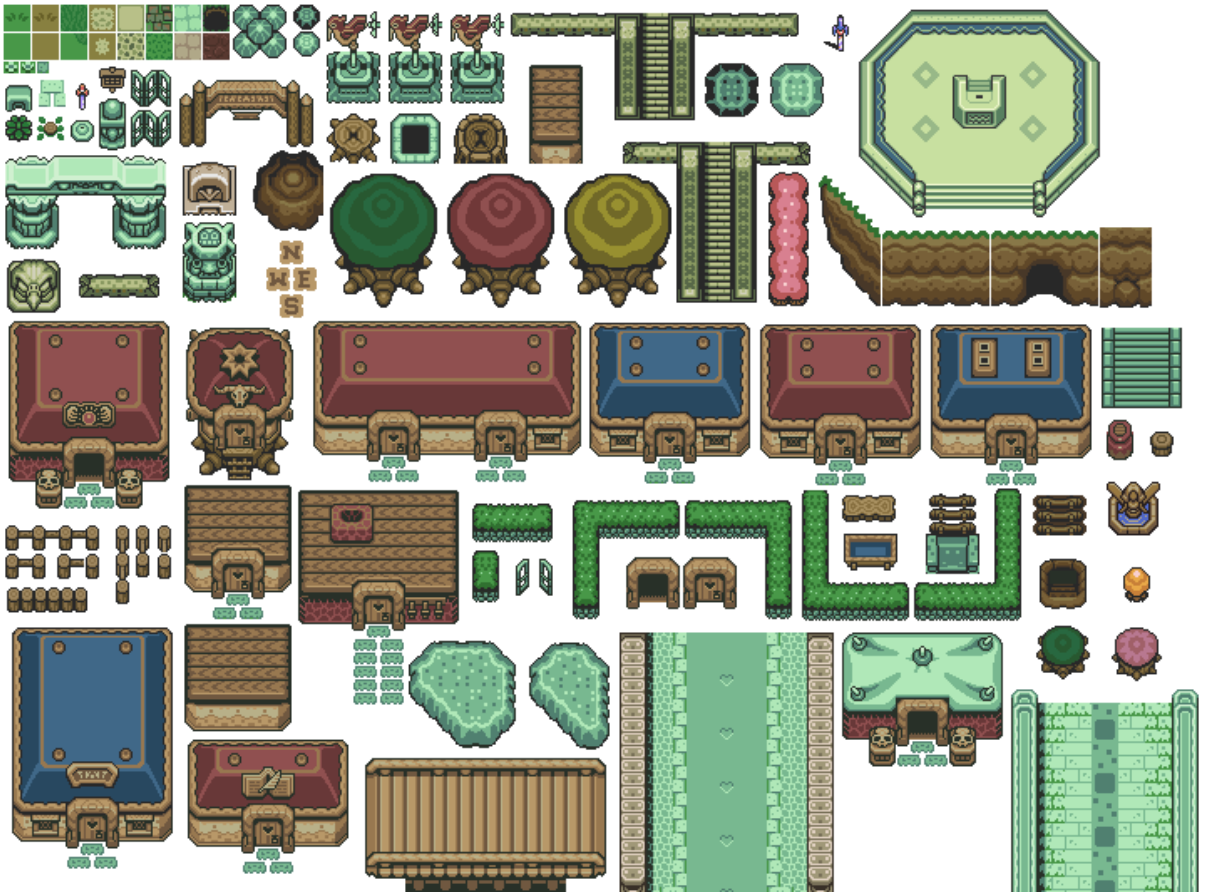

Fig 7. Los Tiles son imágenes 2D que contienen las partes del escenario y los fondos. (The Legend of Zelda: A Link to the Past, Nintendo 1991)

Fig 8: El editor de niveles permite crear niveles utilizando los sprites y tiles, estos se guardan en archivos internos que el juego puede administrar. (The Legend of Zelda: A Link to the Past, Nintendo 1991)

En resumen, un motor de videojuegos es un conjunto de piezas de software y herramientas integradas que brindan una base para realizar un videojuego. Actualmente los motores de videojuegos proveen soporte para exportar juegos a diferentes plataformas de una manera simple, teniendo en cuenta que los diferentes dispositivos tienen motores gráficos y hardware distintos. Un proyecto en un motor como Unity o Unreal puede ser explotado para Windows, consolas como PlayStation, Android e IOS, lo que permite ahorrar tiempo en implementar las especificaciones nativas de cada plataforma de forma individual. Podemos tomar como ejemplo las shaders[5] que en Windows utilizan la librería DirectX [6] y en Android deberían migrar a OpenGL[7] o Metal[8] para funcionar en IOS. La implementación de estas características en los motores de videojuegos permite que los desarrolladores lleven sus juegos a diferentes plataformas sin que esto implique una migración total o parcial del proyecto.

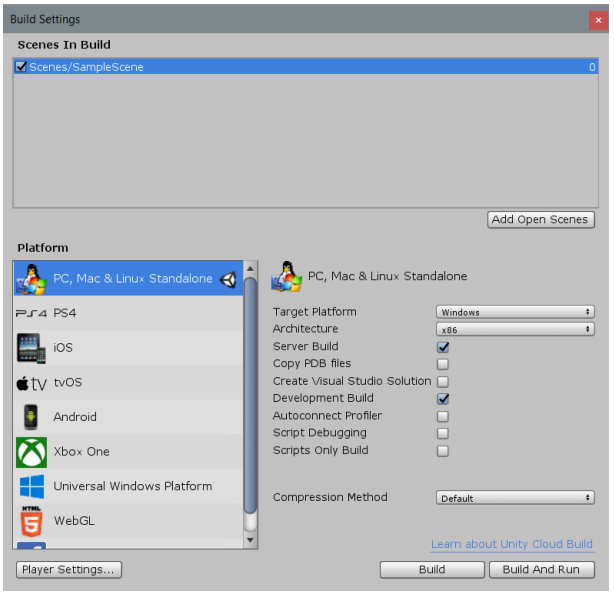

Fig 9. Configuración de exportación de Unity.

[5] Shaders: Los shaders son un tipo especial de programa que se ejecuta en unidades de procesamiento gráfico (GPU). Las GPU son especialmente útiles para el renderizado porque están optimizadas para ejecutar miles de instrucciones en paralelo.

[6]DirectX es un conjunto de componentes de Windows que permite al software, principal y especialmente juegos, funcionar directamente con el hardware de vídeo y audio.

[7] OpenGL es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D.(https://www.opengl.org/)

[8] Metal es una API de shaders y gráficos 3D acelerada por hardware de bajo nivel y baja sobrecarga creada por Apple.(https://developer.apple.com/metal/)

II.3 Realidad Aumentada

El término realidad aumentada fue acuñado en 1992 por Caudell y Mizell, investigadores de la compañía Boeing, para referirse a los sistemas de visualización que pueden añadir imágenes sintéticas a la imagen real. Más adelante, en 1995, Rekimoto y Nagao presentan los primeros marcadores[9] para Realidad Aumentada de matriz bidimensional. Desarrollan un sistema llamado NaviCam que reconocía marcadores en forma de barras y mostraba información adicional sobre la pantalla. Se puede tomar este antecedente como el punto inicial en el que se basan las librerías modernas que utilizan esta tecnología.

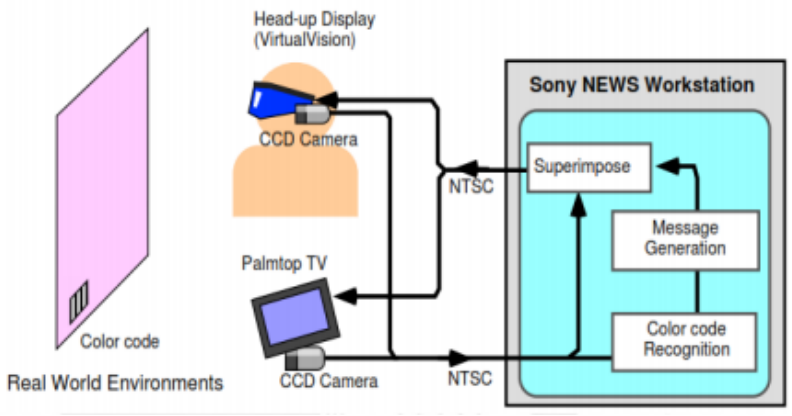

Fig 10. Arquitectura del sistema NaviCam. Rekimoto/Nagao 1995

Para definir realidad aumentada podemos tomar la definición de Azuma (1997), que lo describe como un sistema que reúne los siguientes requerimientos:

Combina la realidad y lo virtual. Al mundo real se le agregan objetos sintéticos que pueden ser visuales, como texto u objetos 3D (wireframe o fotorealistas), sonoros, táctiles y/o los relativos al olfato.

Es interactivo en tiempo real. El usuario ve, en tiempo real, un mundo real con objetos sintéticos agregados que le ayudarán a interactuar con la realidad.

Debe registrar las imágenes en espacios 3D. La información virtual tiene que estar vinculada, espacialmente, al mundo real de manera coherente.

[9] Un marcador le proporciona a la propia aplicación de Realidad Aumentada (RA) una clave visual o activadora sobre dónde posicionar el contenido de RA. Un marcador puede ser una imagen, un logo, o cualquier tipo de objeto 2D que pueda ser distinguido y reconocido por la cámara

Esta definición se adapta más al panorama actual de la tecnología, ya que la mayoría de las aplicaciones de realidad aumentada e incluso las redes sociales mediante filtros implementan las tres características definidas por Azuma.

Desde sus primeras implementaciones en ordenadores, la realidad aumentada necesitó de referencias específicas del mundo real donde anclar el contenido virtual. Todo sistema de realidad aumentada ejecuta de manera secuencial las siguientes cuatro tareas (Mazen Abdulmuslih, 2012)

1. Captura del escenario.
2. Identificación de la escena.
3. Fusión de la realidad con objetos sintéticos.
4. Visualización de escena aumentada.

Inicialmente se usaron códigos QR e imágenes fiduciales que eran fáciles de identificar por algoritmos de computación visual y las cámaras de esa época. A medida que avanzó el tiempo, las mejoras en los dispositivos, que se volvieron más potentes y con mejores cámaras, paralelamente al avance de la computación visual, permitieron que la realidad aumentada pudiera usar como referencias objetos del mundo real sin ningún patrón en particular, ya sea la detección de rostros con diferenciación de los sectores de la cara como la detección de espacios reales mediante la geolocalización o la identificación de características del entorno.

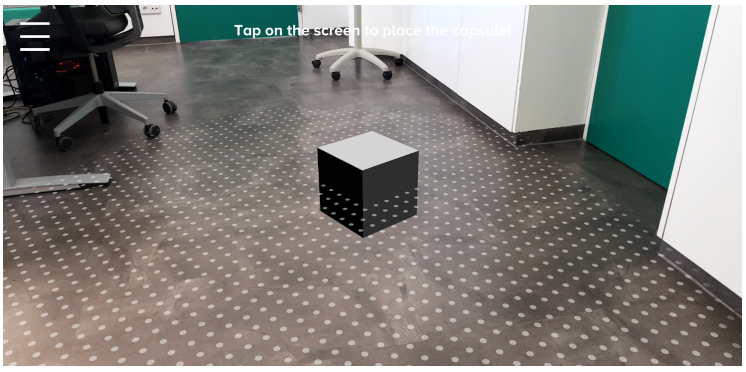

Fig 11. Detección de espacios reales con la librería Arkit

Hoy en día se dispone de muchas librerías de realidad aumentada, tanto para web como para dispositivos móviles, haciendo la tecnología cada vez más accesible. Tomemos el ejemplo, de la librería Spark AR de Facebook que atrae a muchos usuarios a esta tecnología permitiéndoles crear filtros para las redes sociales. Esta librería utiliza tecnologías como la detección de espacios o el face

tracking para crear contenido de realidad aumentada con el que además pueden interactuar en tiempo real y compartir con otros usuarios. Para dispositivos móviles las librerías explotan la calidad de la cámara y los sensores que tienen los smartphones actuales. De esta manera pueden detectar características de espacios reales gracias a la calidad de las cámaras y sensores como el giroscopio o detectar posiciones específicas del mundo real combinando el GPS con el sensor magnético (brújula) del dispositivo. Gracias a estas librerías podemos prescindir de marcadores específicos para colocar contenido virtual, lo que abrió el panorama para aplicaciones más creativas que interactúen con el mundo real de manera más fluida para el usuario.

II.4 Videojuegos de Realidad Aumentada

La constante mejora en los smartphones permitió que las aplicaciones de realidad aumentada puedan brindar experiencias cada vez más interactivas y visualmente interesantes. En este contexto empezaron a aparecer aplicaciones que aprovechan esta tecnología, entre ellas los videojuegos. La utilización de marcadores para ubicar objetos 3D y la integración de librerías de realidad aumentada con los motores de videojuegos permitieron que los desarrolladores pudieran crear videojuegos y llevar sus niveles al mundo real.

Además los sensores presentes en los dispositivos son esenciales para mejorar la experiencia o para expandir las posibilidades del jugador de interactuar con el mundo real. Por ejemplo la utilización del GPS permite que se desarrollen juegos que ubiquen contenido virtual en posiciones geográficas determinadas, de manera que se puedan crear narrativas y mecánicas de juego que trascienden lo virtual. Un ejemplo es Ingress (Fig 12) que dividía a los jugadores en dos facciones que competían por controlar distintos puntos ubicados en el mundo real y usaba realidad aumentada para interactuar con estos espacios. Posteriormente se sumaría el uso de sensores como la brújula y el giroscopio para mejorar la experiencia de interacción entre el contenido virtual de los puntos y el espacio real donde se encuentra el jugador.

Fig 12. Ingress (Niantic Lab 2012)

El desarrollo de la tecnología expande las posibilidades para los desarrolladores y les da más libertad a la hora de ubicar el contenido del videojuego sin preocuparse por las limitaciones asociadas al uso de marcadores mencionadas anteriormente. Un ejemplo de este tipo de juego es Angry Birds AR: Isle of Pigs (Fig 13) en el que tan solo con seleccionar un espacio real se ubicará el contenido del juego en ese lugar. Además, aprovecha esto para poner todo el contenido del juego especializado, desde los niveles hasta la interfaz del juego, distribuida en el espacio 3D proyectado en vez de estar fija en la pantalla del dispositivo .

Fig 13. Angry Birds AR: Isle of Pigs (Rovio Entertainment Corp. 2019)

III. Contexto del problema a resolver

Este proyecto plantea volver sobre los pasos del videojuego “Invasión Aumentada” desarrollado en el 2014 y publicado en 2015, como producto de la investigación de la tecnología realidad aumentada en el Laboratorio de informática aplicada de la Universidad Nacional de Río Negro. A partir del análisis de los fallos de diseño del proyecto original se plantea un proceso de “remake” con el fin de convertir la idea inicial en un videojuego que cumpla con las metas planteadas e implementar todas las características del proyecto que quedaron pendientes.

Para ello se pone el foco en del documento de diseño original, donde se definieron todas las características que se implementaron en el juego y a partir de esto elaborar un nuevo documento de diseño que tome las mejores características, corrija los errores y agregue nuevas posibilidades basadas en la tecnología actual y las mecánicas de juego más novedosas.

Los primeros pasos de esta fase consisten en desarrollar esa idea original del juego y describir minuciosamente cómo será cada detalle. Generalmente este proceso descriptivo se plasma en un documento conocido como “documento de diseño de videojuego o GDD” (Adams, E. 2009). En este caso tenemos el documento de diseño del proyecto anterior y se plantean una serie de revisiones y ajustes con el fin de que el nuevo proyecto no repita los errores del original.

III.1 Análisis del GDD (Game Design Document)

La comunidad de diseño de videojuegos generalmente prefiere el término mecánicas del juego al de “reglas del juego” porque las reglas se consideran instrucciones explícitas que el jugador conoce, mientras que la mecánica está oculta para el jugador, es decir, está implementada en un software con el cual cada jugador puede interactuar a través de la misma interfaz pero de maneras diferentes. “Los jugadores no tienen que saber cuáles son las reglas cuando comienzan a jugar; a diferencia de los juegos de mesa y cartas, el videojuego les enseña mientras juegan” (Game Mechanics: Advanced Game Design. Ernest W. Adams, 2012).

A partir del análisis del Documento de diseño original (adjunto en anexo) lo primero que destaca es que no se definen adecuadamente las mecánicas del juego, ya que si bien las describe, no hace un análisis exhaustivo de todas las

características e interacciones que suponen las mismas, por lo que podrían existir ambigüedades y distintos desarrolladores podrían implementar mecánicas totalmente diferentes por la falta de detalle del GDD original.

Si tenemos en cuenta la estructura de un GDD estándar debe contener al menos las siguientes secciones:

- Concepto: En esta sección se definen las características generales del juego como su título, sinopsis, género y público objetivo.
- Visión General del Juego: Describe los lineamientos básicos del juego y cómo interactúa el jugador con las mecánicas.
- Mecánicas: Se detallan exhaustivamente las mecánicas del juego, de manera que cualquier miembro del equipo que vea esta sección puede entender como el jugador interactúa con cada componente del juego.
- Interfaces: Se definen las pantallas que tendrá el juego (como menú, selección de niveles, pantalla de pausa), cómo se relacionan e interactúan entre sí, además define el lineamiento visual que tendrá la interfaz de usuario y cómo esta interactúa con las mecánicas del juego.
- Gráficos: Define la dirección artística del juego, los recursos visuales necesarios para cada componente del juego, esto puede variar bastante según el tipo de juego, por ejemplo en un juego 2D la mayoría de los recursos serán Sprites (imágenes 2D) pero en un juego 3D hay distintos tipos de recursos como modelos 3D, texturas, iluminación, etc.
- Música/Sonidos: Se listan los recursos sonoros del juego, tanto los sonidos para cada componente del juego que lo requiera como la música que acompaña cada escena/nivel. Esta sección también puede ser descriptiva, permitiendo que un compositor lo lea y pueda tener una idea de qué sensación debe transmitir cada pieza sonora del videojuego.

A partir de esta estructura podemos tomar como punto de partida el documento de diseño original y crear desde allí uno nuevo, que refleje detalladamente las características del juego y el lineamiento estético omitido en el original.

III.2 Análisis de la arquitectura del juego

Una característica importante en el desarrollo de software es tener una arquitectura flexible que pueda ser reutilizada para un proyecto similar, en este caso a partir del análisis de la arquitectura del proyecto original, vemos que la estructura del proyecto original no necesita sufrir grandes cambios para adaptarse al nuevo proyecto. Si tenemos esto en cuenta, más allá de los cambios que implica el proceso de “remake”, la arquitectura es reutilizable en su mayor parte.

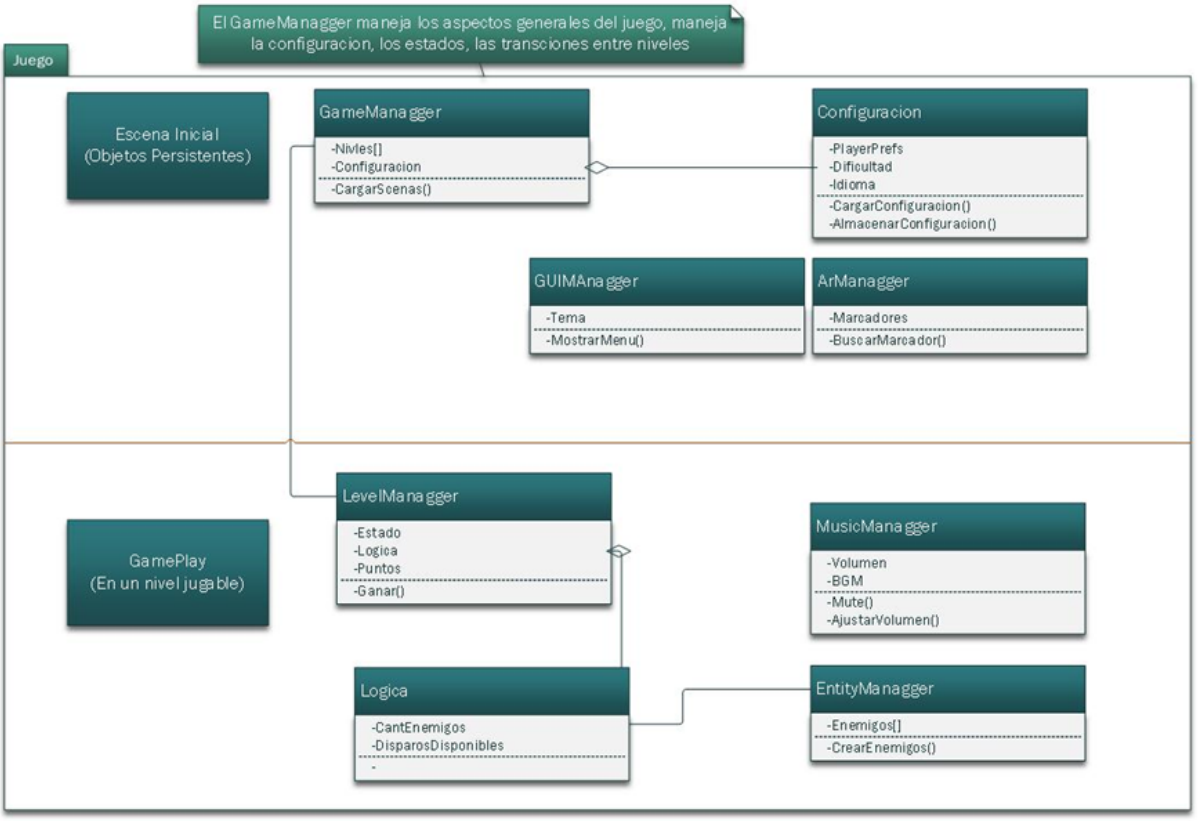

Fig 14. Documento de arquitectura original del proyecto

En este caso la arquitectura necesita adaptarse a la dinámica actual de desarrollo de videojuegos, pero en su mayoría es reutilizable, ya que por ejemplo separa la lógica del juego en Managers que administran sus propias características y se comunican entre sí es necesario. Esta estructura es muy utilizada en el desarrollo de videojuegos, por lo que si bien deben cambiarse algunas cosas la arquitectura sobrevive al paso del tiempo.

El componente más interesante de la arquitectura original es el ARManagger, que crea una interfaz entre el GameManagger que administra el estado general del juego y la librería de realidad aumentada. Esta capa intermedia permite que el juego no esté atado a una librería de realidad aumentada determinada. Esta fue una decisión acertada producto de la investigación de la tecnología, ya que en el momento se contaba con varias alternativas de librerías de RA. Este componente se desarrolló originalmente para el proyecto “Invasión Aumentada” y fue reutilizado en varios proyectos de realidad aumentada gracias a su flexibilidad y en el caso de este desarrollo nos permitirá trabajar con librerías actuales sin tener que modificar aspectos de la lógica del juego.

También tenemos componentes que deben ser modificados ya que originalmente no eran tan flexibles y estaban muy atados a características propias del motor que actualmente se encuentran depreciadas o cambiaron completamente. Esto lo podemos observar en el diseño de componentes como GUIManagger que maneja toda la interacción del jugador con la interfaz de usuario, este componente utiliza en su mayoría referencias a la librería de interfaz de usuario del motor, por lo que conviene redefinir este componente y los que tengan fallas de diseño similares.

III 3. Análisis de las mecánicas del juego

Las mecánicas se definen en el GDD y definen las características a implementar en la etapa de desarrollo del juego, en el proceso de remake se deberían mantener la mayoría de las mecánicas ya que son la “esencia” del juego. Si se modifican dichas mecánicas podríamos decir que se trata de un juego diferente o por lo menos no podríamos catalogarlo como “remake” sino como una interpretación o una versión diferente del juego. Las mecánicas principales son llamadas “mercancías núcleo” (core mechanics) y son las que definen las bases para el diseño de las demás mecánicas. Si tomamos como ejemplo un juego de plataformas 2D como Super Mario Bros la mecánica del salto es una de las más importantes e interactúa con otras mecánicas más simples como la de los bloques o los power-ups (objetos que aumentan las habilidades/características del personaje de forma temporal, como la estrella en Super Mario Bros), sin esta mecánica el juego sería totalmente diferente, cambiando incluso su género, ya que son sus

mecánicas de movimiento y salto los que lo definen como un juego de plataformas. Las mecánicas principales o núcleo son las que no deben ser alteradas durante el proyecto y las que definen al juego como tal. Podemos definir nuevas mecánicas que interactúan con estas o modificar mecánicas secundarias, siempre y cuando no interfieran con estas.

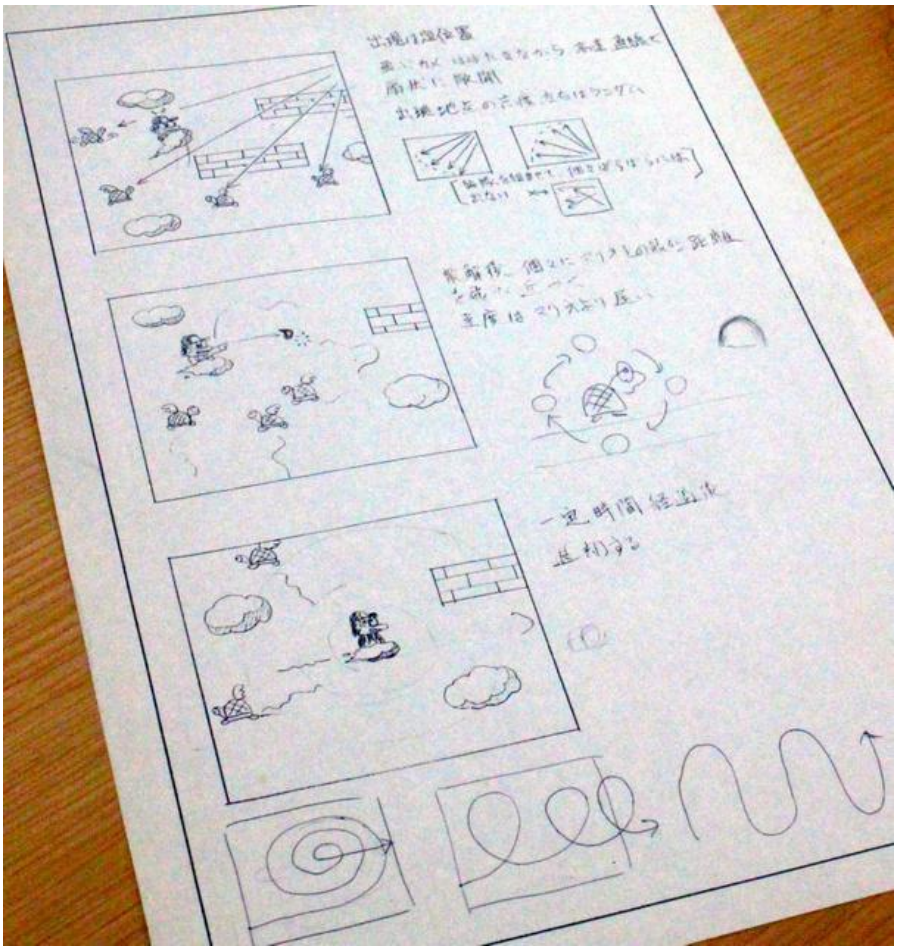

Fig 15. Documento de diseño de Super Mario Bros Shigeru Miyamoto Nintendo (1985)

En el caso de invasión aumentada muchas de las mecánicas núcleo se definieron a partir de las posibilidades que brindaba la realidad aumentada, ya que la posición real del jugador definía como interactuar con los objetos virtuales del juego. Se aprovechó esta característica y se decidió que los proyectiles que dispara el jugador partan desde la ubicación del dispositivo (teniendo en cuenta la posición de la cámara real en el entorno virtual) donde la dirección y fuerza serían determinadas por el trazo que realice el jugador en la pantalla del dispositivo, al comparar las posiciones de la pantalla táctil durante el mismo.

Fig

Fig 16. Mecánica de Disparo de invasión Aumentada (2014)

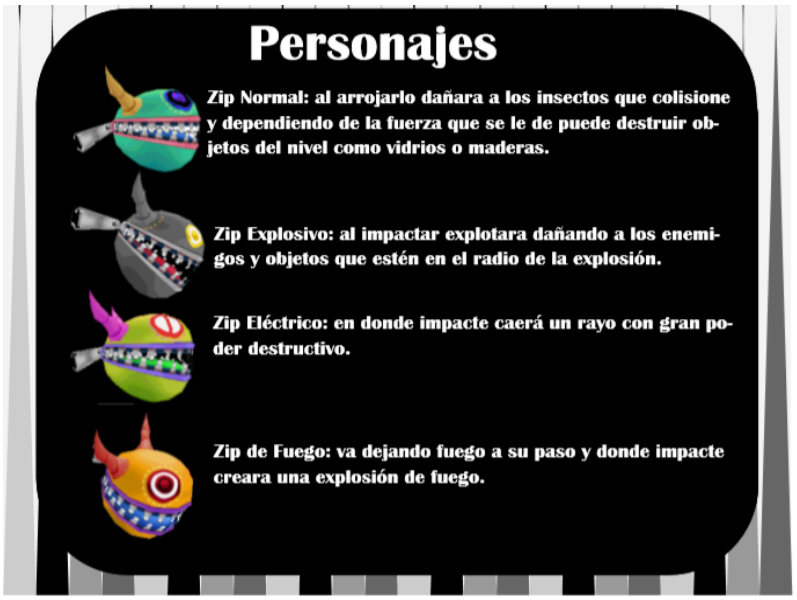

Fig 17. Diseño de personajes Invasión Aumentada (2014)

IV .Solución propuesta

En el desarrollo del presente capítulo, se detalla cómo fue implementada la solución a la problemática detallada anteriormente.
Asimismo, se explicará la metodología usada para planificar y estimar todo el trabajo realizado, la arquitectura y las diferentes tecnologías utilizadas.

IV.1. Metodología de trabajo y planificación

"La ingeniería de software es una ciencia de diseño que proporciona métodos para el desarrollo, implementación y mantenimiento de productos de software" (Pfleeger & Atlee, 2005). El desarrollo de videojuegos no es ajeno a esta disciplina, donde deben convivir una gestión del proyecto basado en las buenas prácticas que provee dicha ingeniería con las definiciones del diseño del juego detalladas en el GDD.

Para esto se toman algunos principios de las metodologías ágiles, que permiten probar las mecánicas del juego en etapas tempranas del desarrollo, donde las tareas se dividen en iteraciones que a cada ciclo aumentan su funcionalidad. Cada iteración es conocida como Sprint y al finalizar, el resultado obtenido es una pieza de software funcional, que fue previamente probada. De esta manera podemos ver si las mecánicas definidas funcionan en la práctica y validar el documento de diseño.

IV.2. Scrum y el proceso de desarrollo de videojuegos

En este caso, al tratarse de un remake, debemos tener en cuenta que el proyecto anteriormente pasó por una etapa de desarrollo, donde se aplicó la metodología ágil Scrum, es decir que ya se dividió en Sprints y se implementaron las historias de usuario definidas en el proyecto original. Para este proceso de rediseño se toman algunos de los principios de las metodologías ágiles, ya que muchos de los conceptos del método Scrum no se aplican del todo a la naturaleza del proyecto. La existencia de por ejemplo conceptos como rejugabilidad o diversión, que son difícilmente cuantificables influyen en los cambios de mecánicas

y estas generan nuevos requerimientos. Por otro lado la división en Sprints y el modelo iterativo/incremental se ajustan perfectamente a la naturaleza del proyecto y nos permite revisar uno por uno los ciclos del proyecto original y revisar qué cambios necesita para el proyecto actual.

Teniendo en cuenta lo mencionado anteriormente y en base la experiencia del desarrollo del proyecto original, podemos hablar de un “Scrum adaptado” al desarrollo de videojuego, ya que debemos tener en cuenta que si bien el resultado es una pieza de software, en el proceso de desarrollo conviven disciplinas heterogéneas.

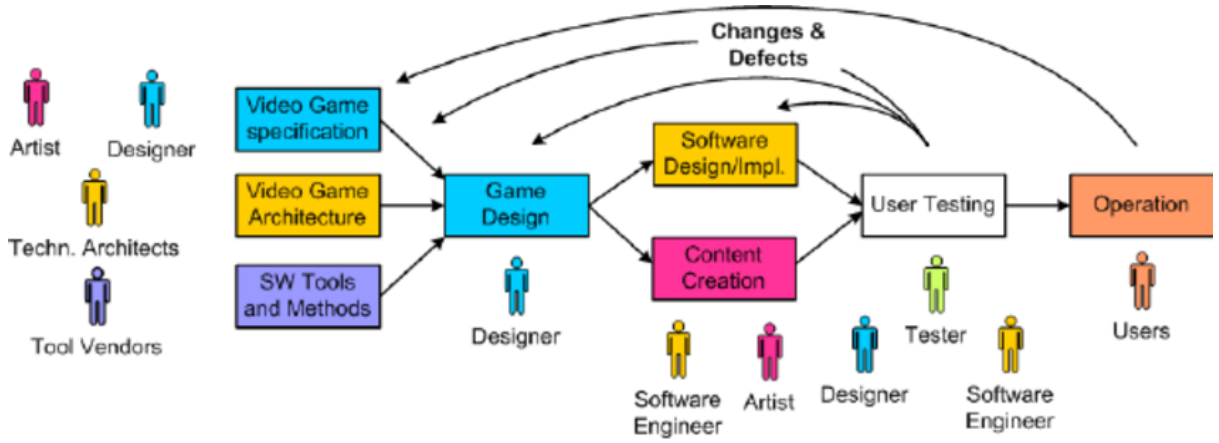

Fig 18. Disciplinas Heterogéneas en el proceso de desarrollo de videojuegos. Musil, J., Schweda, A., Winkler, D., & Biffl, S. (2010, September). Improving video game development: Facilitating heterogeneous team collaboration through flexible software processes. In European conference on software process improvement (pp. 83-94). Springer, Berlin, Heidelberg.

IV.3. Definición de los “Sprints”

Dentro de este “Scrum adaptado” definimos 4 Sprints que tienen como objetivo revisar cada una de las etapas del proyecto original, teniendo como resultado una pieza funcional del nuevo proyecto, con lo mejor del proyecto base y adaptando lo que se considere necesario para lograr que el juego esté a la altura de los estándares definidos anteriormente.

Sprint 1: Etapa de Rediseño	Inicio:01/04/20	Fin:15/04/20
1 - Análisis del GDD Original		
2 - Elaboración de un nuevo documento de diseño del juego		

Sprint 2: Prototipo del juego	Inicio:16/04/20	Fin:15/05/20
1 - Implementar las mercancías básicas del juego		
2 - Integrar la realidad aumentada al proyecto		
3 - Tener un prototipo funcional del juego		

Sprint 3: Recursos del Juego	Inicio:16/05/20	Fin:15/06/20
1- Elegir y/o elaborar los recursos gráficos del juego		
2 - Elegir y/o elaborar los recursos sonoros del juego		
3 - implementar la Interfaz de Usuario del Juego		

Sprint 4: Demo del juego	Inicio:15/06/20	30/06/20
1 - Implementar todas las mecánicas del juego		
2 - Crear las pantallas (escenas) definidas en el GDD		
3 - Integrar los recursos audiovisuales al juego		
4 - Versión demo del juego probado y validado		

IV.4. Herramientas de seguimiento del proyecto

Para gestionar los requerimientos del proyecto es necesario mantener un seguimiento activo de las tareas que componen los sprints y el estado de las mismas. Esto permite tener una visión global del proyecto y saber si se avanza según lo planificado y si las tareas cumplen el tiempo de sprint o es necesario ajustar los plazos o revisar el proceso de desarrollo. Para este proyecto se utilizará la herramienta Tape[10], que es un software que permite hacer un seguimiento de tareas, agruparlas en sprints y asignarle a cada estado a cada una. Los estados que permite asignar Tape son “Trabajando”, “Hecho”, “Prioritario”, “Validado”, “Entregado”.

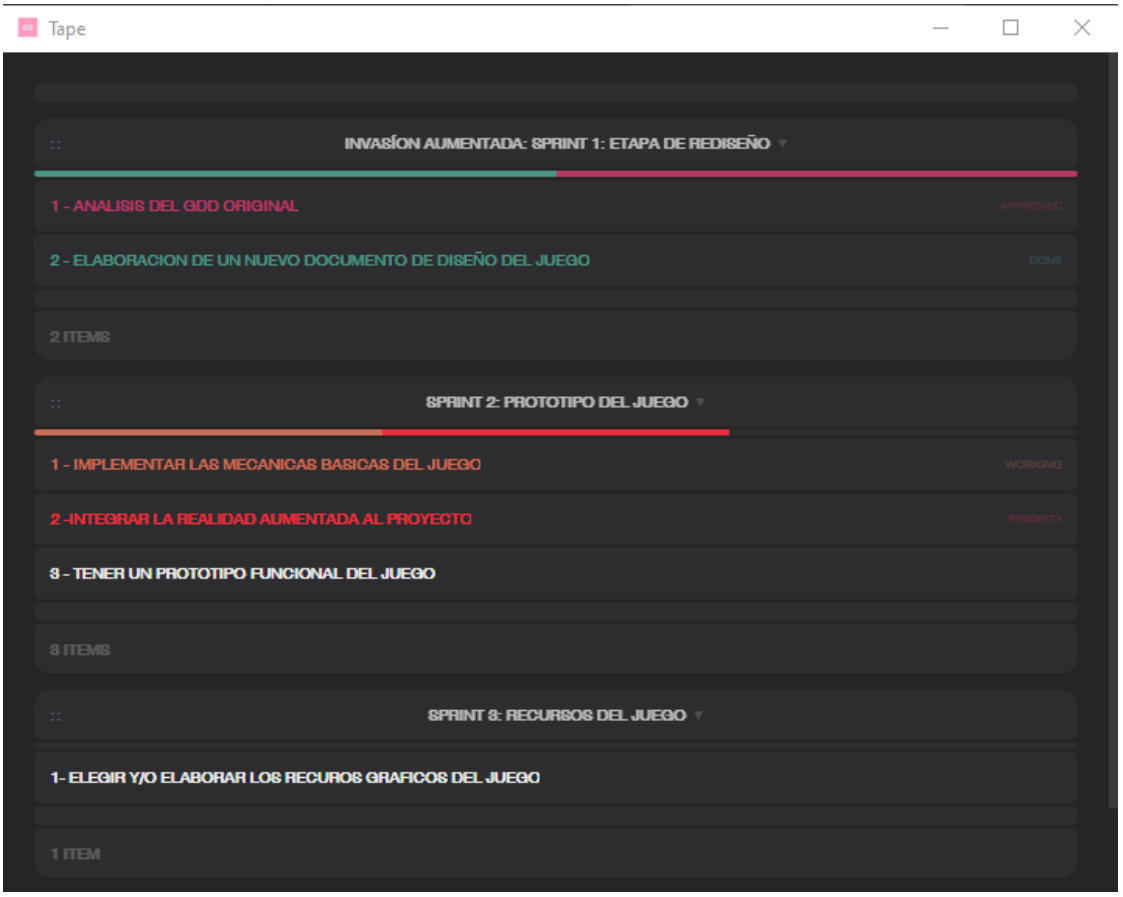

Fig 19. Vista de los Sprints en Tape

[10] https://aeriform.itch.io/tape

IV.5. Entornos de desarrollo de Videojuegos:

Para desarrollar un videojuego es necesario un conjunto de herramientas que comprenden desde las orientadas a crear una pieza de software como el motor del videojuego hasta las que permiten crear o editar los recursos multimedia asociado al proyecto (imágenes, sonido, modelos 3D), en el desarrollo de este proyecto se utilizarán las siguientes herramientas:

- Unity 3D 2019.3: Este motor de videojuegos tiene todas las herramientas necesarias para llevar adelante un proyecto de estas características. La utilización de un motor permite no tener que implementar el núcleo del proyecto desde cero. Teniendo como base una serie de herramientas integradas que permiten realizar la mayoría del proceso de desarrollo dentro del mismo.
- Visual Studio: Editor de código para C# que es el lenguaje que utiliza Unity, además gracias a un plugin el código está asociado dinámicamente al proyecto por lo que funciona de manera fluida con el motor.
- Blender 3D: programa gratuito para crear o editar modelos 3D, con herramientas que permiten trabajar sobre todas sus características.
- Audacity: Editor de audio gratuito que permite grabar/editar sonidos.
- GIMP: editor de imágenes digitales en forma de mapa de bits.

IV.6. Diseño del videojuego

Para el proceso de “remake” es importante analizar el documento de diseño original y decidir qué es lo que se quiere adaptar. Si analizamos remakes de juegos famosos podemos ver que se mantiene la esencia y las mecánicas del juego, pero se ajustan a la época, por ejemplo si analizamos el videojuego Pokemon Rojo/Verde (Pokemon Red/Green 1996 Gameboy) tuvo dos procesos de remake, el primero fue Pokemon Rojo Fuego/Verde Hoja (Pokemon Fire Red/Green Leaf, 2004 Game Boy Advance) y el segundo Pokemon Pokemon Let's Go, Pikachu!/Eevee! (Nintendo Switch 2018) en todos los juegos el jugador realiza el mismo progreso por las mismas areas, ya sea en un pantalla en blanco y negro en el Gameboy, en color en Gameboy advance o en 3D en Nintendo Switch. Lo que se ajusta más allá del

aspecto visual que tiene que ver con la potencia de cada consola, son las mecánicas del juego, por ejemplo las primeras dos ediciones tiene dificultad más alta que tiene que ver con el público objetivo. El último juego está orientado a un público más joven y actualmente no se valora tanto la dificultad como un factor decisivo. Producto de esto el juego agregó mecánicas que facilitan al jugador nuevo tener una experiencia de juego más accesible.

También podemos analizar los cambios que tienen que ver con las tecnologías de cada dispositivo y cómo influyeron estas en las mecánicas del juego, por ejemplo la forma de interactuar con los demás jugadores:

- Game Boy: utilizaba un cable que conectaba dos consolas para interactuar con otros jugadores.
- Gameboy Advance: tenía un sistema infrarrojo para comunicar consolas y un cable especial que admitía hasta 4 jugadores.
- Nintendo Switch: la consola se conecta por Wi-fi y el jugador puede interactuar con cualquier jugador del mundo. Además cuenta con bluetooth para conectarse con smartphones e interactuar con juegos de la misma saga en diferentes plataformas.

Tomando este ejemplo podemos concluir que además de actualizar las mecánicas del juego debemos tener en cuenta el contexto actual y las tecnologías disponibles para crear la mejor experiencia de juego posible.

Fig 20. Captura Superior: Pokemon Red (Nintendo, 1996)
Captura Intermedia: Pokemon Fire Red (Nintendo 2004)
Captura inferior: Pokemon Let's Go Pikachu (Nintendo 2018)

IV.7. Mecánicas de invasión Aumentada:

La principal mecánica de invasión aumentada consiste en poder disparar proyectiles desde la posición de la cámara de realidad aumentada (AR Camera), según la dirección y fuerza que le indique el jugador a partir de interactuar con la pantalla del dispositivo, siendo estas calculadas a partir de la dirección, sentido y magnitud del vector resultante de la interacción del jugador con la pantalla.

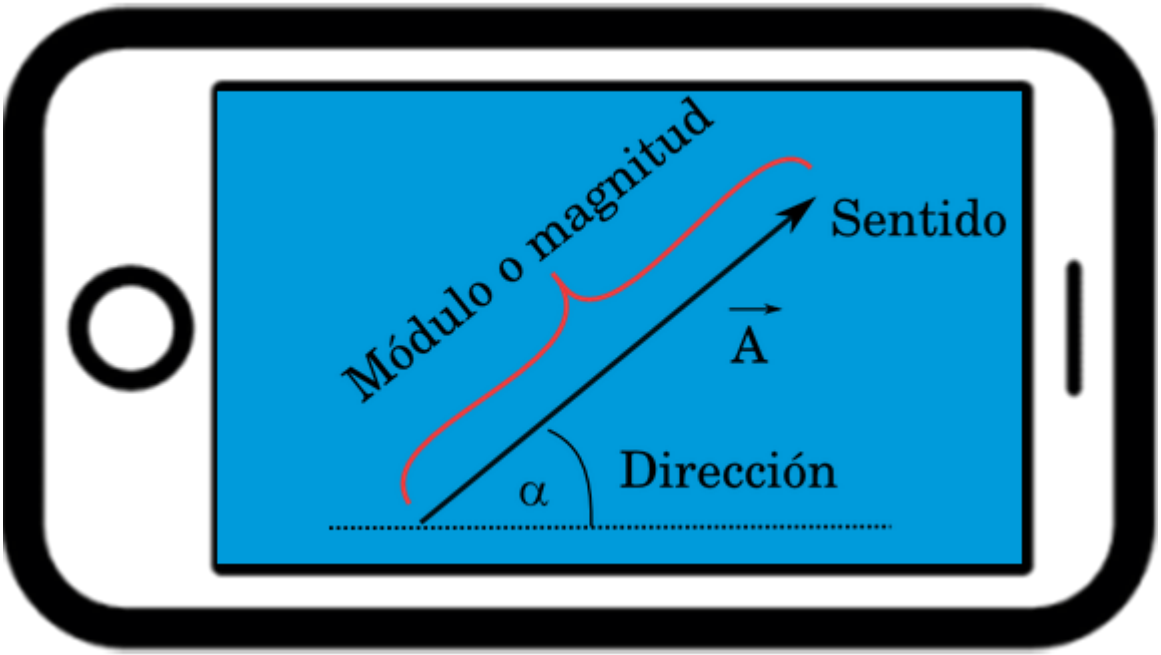

Fig 21. Vector generado a partir del trazo del dedo del jugador por la pantalla.

Este cálculo permite crear un proyectil que se mueve en el entorno 3D teniendo en cuenta la posición real del jugador pero permitiendo dirigirlo hacia cualquier lugar del nivel, simplificando la jugabilidad y permitiendo al jugador no tener que girar tanto para alcanzar a los enemigos. El algoritmo que crea el disparo utiliza la posición cámara de realidad aumentada como origen (que representa la cámara del dispositivo real en la escena 3D) y transforma dicho vector 2D en una fuerza tridimensional que es aplicada al proyectil al momento de crearse.

Fig 22. Algoritmo de disparo de Invasión Aumentada (2014)

Siendo esta la mecánica principal que define cómo va interactuar el jugador con el nivel, el juego tiene otras mecánicas secundarias que le agregan variedad como los tipos de proyectiles, que además de impactar y causar daño pueden explotar o estar cargados de electricidad, permitiendo diferentes formas de interactuar con el entorno. También se agregan al nivel objetos destruibles como vidrios o elementos apilados, para que el jugador pueda encontrar maneras creativas de superar el nivel y mejorar la sensación de inmersión.

Todos estos detalles están presentes en el documento de diseño nuevo, ya que en el original no se habían definido exhaustivamente cada uno de los elementos del juego. Esto dificulto en su momento definir qué elementos agregar primero y estimar el tiempo total del desarrollo del proyecto.

Además se definió claramente el diseño de niveles, ya que en la versión original no había ninguna relación ni estética ni mecánica en el orden de los niveles, por lo que desde el documento de diseño se elaboró un nivel prototipo que sirva de base para el diseño de niveles posteriores.

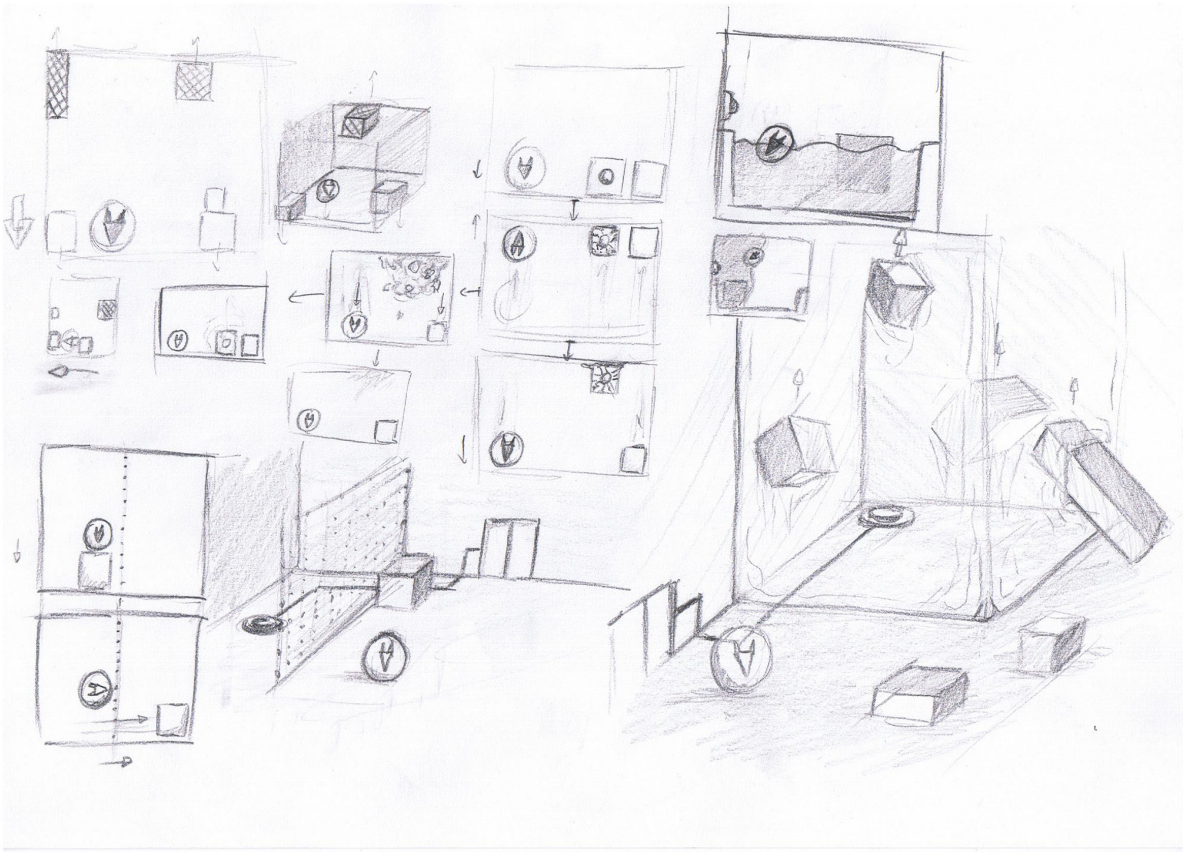

Fig 23. Boceto de un nivel

IV.8 Elaboración de un nuevo documento de diseño

El documento de diseño del videojuego o GDD (Game Design Document) es la pieza de documentación más importante para que el desarrollo del mismo sea exitoso, en él se definen todas las características a implementar en el juego detalladamente, desde las mecánicas y la narrativa hasta la dirección artística del juego y los el tipo de recursos necesarios para su desarrollo. Habiendo analizado previamente las mecánicas del proyecto original y definidos cuáles son las que van a ser reutilizadas se puede elaborar un nuevo documento de diseño que plantee a partir de estas cuales son las nuevas características a implementar en el proceso de remake.

El GDD tiene las siguientes secciones:

IV.8.1 Resumen

Invasión aumentada es un videojuego de realidad aumentada con mecánicas arcade cuya principal característica es poder utilizar dicha tecnología para proyectar los niveles del juego en el mundo real mediante la cámara del dispositivo, permitiendo que el jugador pueda interactuar con el nivel teniendo en cuenta su posición real.

IV.8.2 Historia

Invasión aumentada plantea un mundo invadido por criaturas extrañas que el jugador puede eliminar a través de los proyectiles que lanza con la cámara del dispositivo, como el juego se desarrolla en espacios reales cada vez que el jugador gana un nivel estaría “limpiando” de monstruos la zona en la que se encuentra

IV.8.3 Mecánicas

Proyectiles: se toma la mecánica de la versión original analizada en el capítulo previo, solo se cambian los “zips” por proyectiles de energía para simplificar la narrativa del juego.

También se conservan los tipos de proyectiles definidos originalmente:

- Proyectil normal: inflige daño al impacto
- Proyectil explosivo: causa daño de área y puede destruir estructuras
- eléctrico: donde impacta cae un rayo que causa daño extra.
- Proyectil de fuego: puede hacer explotar elementos del nivel y destruir estructuras.

Enemigos: los enemigos siguen el mismo concepto que en el original pero se amplía su comportamiento mediante una IA que les permite moverse y se agregan enemigos voladores.

Niveles: se mantiene el concepto de un diseño de niveles 3D que motive al usuario a moverse en el espacio real, lo que permite ocultar enemigos en lugares donde el usuario tenga que rotar alrededor del nivel para encontrar todo su contenido.

IV.8.4 Gráficos

Se abandona el estilo hiperrealista por una búsqueda de un estilo uniforme que evite la disonancia entre los objetos del nivel y los enemigos, se opta por una estética estilizada que además de permitir mejor rendimiento en dispositivos de gama media/baja se aleje del aspecto “aterrador” de las arañas de la versión original que podían ser un impedimento para el público objetivo del juego sea más joven.

IV.8.5 Sonidos y música

Se plantea el rediseño de los sonidos para cada interacción y se mantiene la decisión de no utilizar música de fondo, ya que al tratarse de una experiencia de realidad aumentada la música puede afectar la sensación de inmersión.

IV.8.6 Flujo de pantallas

Anteriormente el juego tenía un menú donde se elegía el nivel y al terminar se volvía al mismo y se desbloqueaban los niveles, en esta versión ya que la tecnología nos permite tener una sesión de realidad aumentada persistente y cargar contenido adicional sin interrumpir la experiencia. Por esto se plantea tener un menú en la escena principal de realidad aumentada donde se carga cada nivel de forma dinámica y se elige uno nuevo de forma aleatoria una vez que se completa el primer nivel, permitiendo que las sesiones de juego sean diferentes entre sí para favorecer la rejugabilidad.

IV.9. Configuración del Entorno de Trabajo

Como se mencionó anteriormente existen muchos motores de videojuegos con diferentes características y ventajas a la hora de desarrollar un proyecto, en este caso en su momento se eligió Unity porque tiene integración con varias librerías de realidad aumentada y por su posibilidad de poder exportar el proyecto a diferentes plataformas. Hoy a pesar del cambio tecnológico Unity sigue siendo la mejor opción, ya que logró crear su propia interface para integrar las librerías de realidad aumentada más utilizadas de forma nativa, lo que ahorra mucho tiempo de desarrollo y nos da un marco de trabajo que nos permite un único código para diferentes dispositivos que utilizan diferentes librerías de RA.

Para comenzar la etapa de desarrollo se creará el proyecto nuevo y se configurarán los módulos necesarios dentro del motor, en este caso Unity permite crear el proyecto a partir de una plantilla que tiene pre configurada algunos aspectos del motor de renderizado para proyectos orientados a dispositivos móviles. Esta característica permite ahorrar tiempo de configuración a la hora de exportar el proyecto, también se utilizó el gestor de paquete para instalar herramientas que ayudan a tener un prototipado rápido.

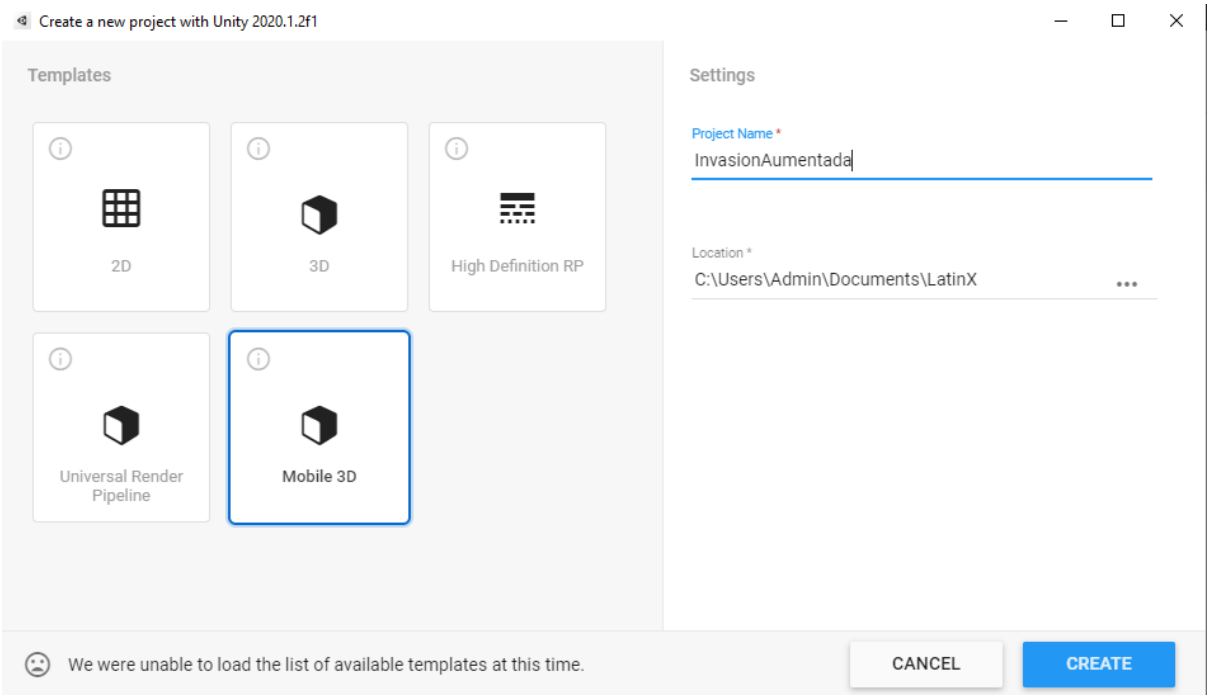

Fig 24. Plantillas de Unity 3D

Paquetes utilizados:

1. ProBuilder: permite crear formas 3D con facilidad dentro del motor, permitiendo prototipar los niveles dentro del motor y luego reemplazar por los modelos 3D finales.

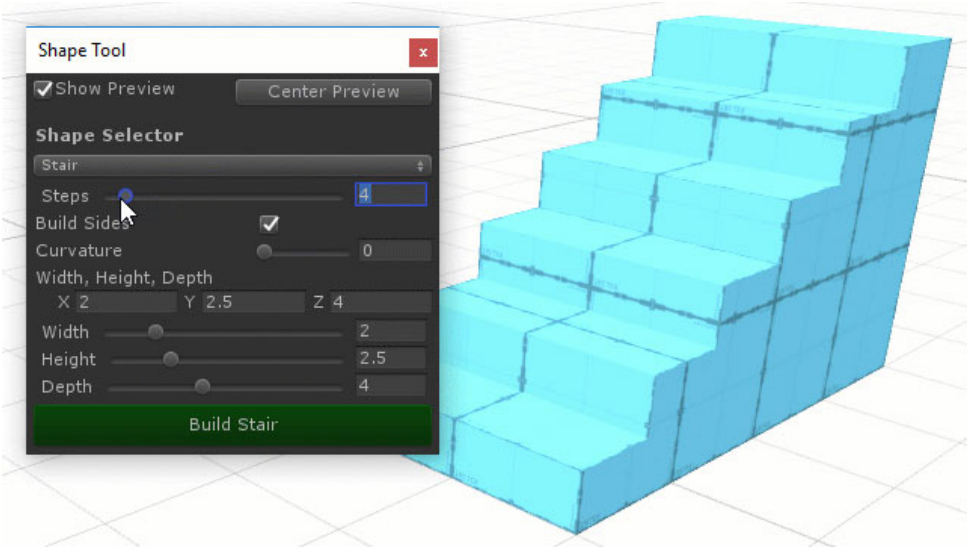

Fig 25. Herramientas de ProBuilder

2. ProGrids: Permite tener una grilla más exacta a la original de Unity, permitiendo ubicar de manera más rápida los elementos dentro de los niveles, esto acelera el tiempo de prototipado.
3. AR Foundation: Herramienta para utilizar realidad aumentada dentro de Unity independientemente de la plataforma a la que se va exportar el proyecto, AR Foundation trabaja como interfaz con ARCore (librería AR de Google para Android) y ARKit (librería AR de Apple para IOS) y trae herramientas para crear escenas de realidad aumentada con todas las características de cada liberia, como la detección de planos.

IV.10. Unity XR

El paquete AR Foundation forma parte del SDK de Unity XR, que es el módulo que tiene integrado el motor para trabajar con aplicaciones de Realidad Virtual Aumentada y Mixta, es importante conocer esta estructura antes de empezar

el desarrollo, para tener en cuenta el contexto global de las herramientas y de qué forma interactúa con el framework.

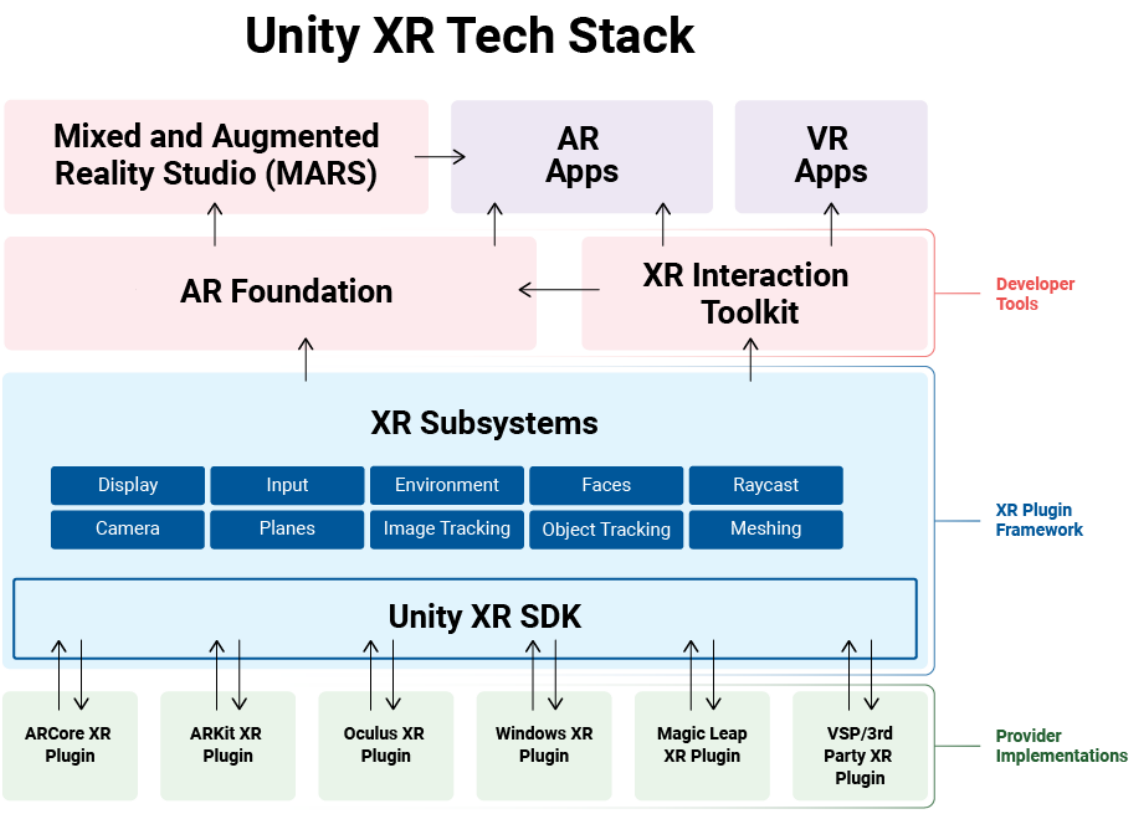

Fig 26. Estructura de Unity XR

Si analizamos la estructura de la librería (Fig 26) podemos ver que para una aplicación AR además de utilizar algunas herramientas de AR Foundation también utilizaremos parte del paquete XR Interaction Toolkit, que permite que distintos dispositivos interactúen con la aplicación sin tener que preocuparnos por la implementación específica de cada plataforma. Además gracias a una arquitectura basada en eventos permite asociar dinámicamente los scripts de la lógica del juego a las acciones propias de la realidad aumentada, por ejemplo asociar la acción de empezar la partida al evento de XR que se ejecuta cuando el jugador posiciona el contenido AR mediante el dispositivo.

Esta estructura permite separar la lógica del juego de las funciones específicas de la librería y reutilizar muchos de los componentes del proyecto original, en la arquitectura de dicho proyecto se utilizaba la clase Manager para separar estas características. Originalmente fue diseñado para que el proyecto no

sea dependiente de la librería que se usó originalmente, aunque fue pensado para trabajar con detección de marcadores, por lo que los eventos a los que referencia no nos sirven para trabajar con detección de planos. Para poder adaptar esta parte del código hay que refactorizar este componente para poder trabajar con los eventos de XR, si bien la librería plantea la posibilidad de utilizar marcadores, el objetivo de este proyecto es llevar el juego a las características más actuales y poder mejorar las mecánicas originales.

IV.11. Análisis del componente ARManagger

En el proyecto original de Invasión Aumentada se utilizó la librería Vuforia, que permite utilizar RA con marcadores, para facilitar la interacción con el usuario se implementó Custom Markets. Esta característica permite que el usuario cree sus propios marcadores a partir de tomar una foto de cualquier superficie una vez tomada esa foto la aplicación devuelve un valor de que tan eficiente era ese marcador y permite volver a tomar una foto hasta que el usuario pueda acomodar el contenido AR de la mejor forma posible. La clase ARManager se encarga de conectar los eventos de realidad aumentada con la lógica del juego, por ejemplo pone en pausa el juego si se pierde el marcador, esto permite que el jugador no afecte su partida mientras no esté viendo la escena.

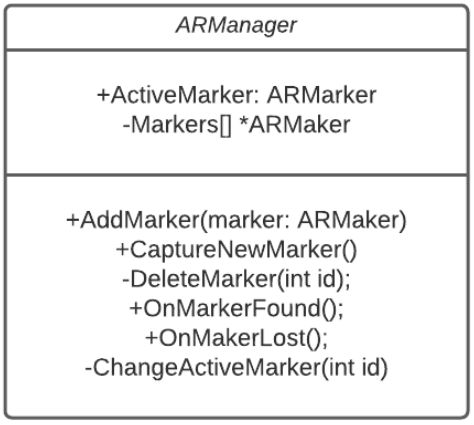

Fig 27. Diagrama de la Clase Manager

Para poder migrar de libreria, teniendo en cuenta que Vuforia no era una librería gratuita y que la tecnología era emergente y probablemente cambiaría

mucho en poco tiempo se pensó una arquitectura flexible donde ARManager se basaba en una interfaz que expone los eventos de forma genérica, permitiendo la posibilidad de volver a implementarla con otra libreria sin tener que cambiar ninguna linea del codigo del resto de las clases.

```
public void OnTrackableStateChanged(
    TrackableBehaviour.Status previousStatus,
    TrackableBehaviour.Status newStatus)
{
    m_PreviousStatus = previousStatus;
    m_NewStatus = newStatus;

    if (newStatus == TrackableBehaviour.Status.DETECTED ||
        newStatus == TrackableBehaviour.Status.TRACKED ||
        newStatus == TrackableBehaviour.Status.EXTENDED_TRACKED)
    {
        ARManager.OnMarkerFound();
    }
    else if (previousStatus == TrackableBehaviour.Status.TRACKED &&
             newStatus == TrackableBehaviour.Status.NO_POSE)
    {

        ARManager.OnMarkerLost();
    }
    else
    {
    // For combo of previousStatus=UNKNOWN + newStatus=UNKNOWN|NOT_FOUND
    // Vuforia is starting, but tracking has not been lost or found yet
    // Call OnTrackingLost() to hide the augmentations
    onMarkerLost.Invoke();
    }
}
```

Fig 28. Llamada a los métodos de ARManager en las Clases de Vuforia

Sin embargo el cambio en la modalidad de mostrar el contenido AR es bastante grande, ya que que hay grandes diferencias entre la detección de marcadores y la utilización de espacios reales para colocar el contenido de realidad aumentada, por lo que es necesario volver a pensar esta parte de la arquitectura antes de avanzar con la implementación del proyecto. La estructura de XR nos brinda acceso a los eventos necesarios para redefinir esta clase a partir de la clase ARPlacementInteractable que expone eventos similares a los implementados en ARManager pero para utilizar detección de planos en lugar de marcadores y además brinda herramientas para gestionar la interacción del contenido AR con el espacio real en el que será colocado. Esta clase hereda de ARBaseGestureInteractable que implementa todas las interacciones de realidad

aumentada de manera genérica y tienen varias implementaciones según cada tipo de detección de realidad aumentada disponible. Los eventos que antes fueron agregados manualmente dentro de las clases de Vuforia en XR están implementados como Unity Events, por lo que podemos vincular los eventos de la lógica del juego directamente dentro del editor de Unity, pudiendo prescindir de la clase ARManagger, si bien se podria dejar esta clase como una capa de la arquitectura seria redundante teniendo en cuenta que ahora XR ya implementa las tareas que realizaba esta clase de manera más eficientes y totalmente integrada al motor. Además XR se convirtió en un estándar de esta tecnología y se plantea soporte a largo plazo, por lo que la clase ARManagger ya no es necesaria en el proyecto.

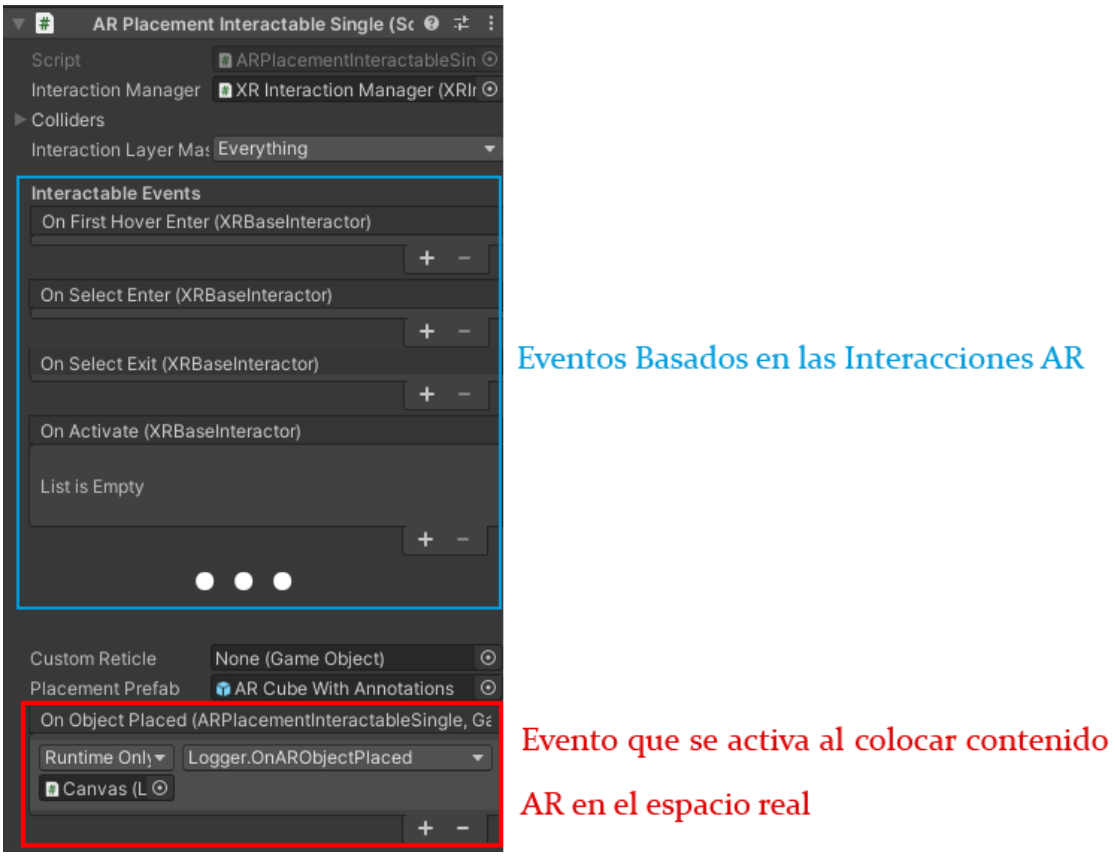

Fig 29. Clase ARPlacementInteractableSingle (hereda de ARPlacementInteractable)

IV.12. Configuración inicial de XR dentro de Unity

El Plug-in XR que será la base de nuestro proyecto y nos permite interactuar con las librerías de RA mediante componentes propios de Unity, permitiendo que el

mismo código corra en diferentes dispositivos, tanto de realidad aumentada como realidad virtual.

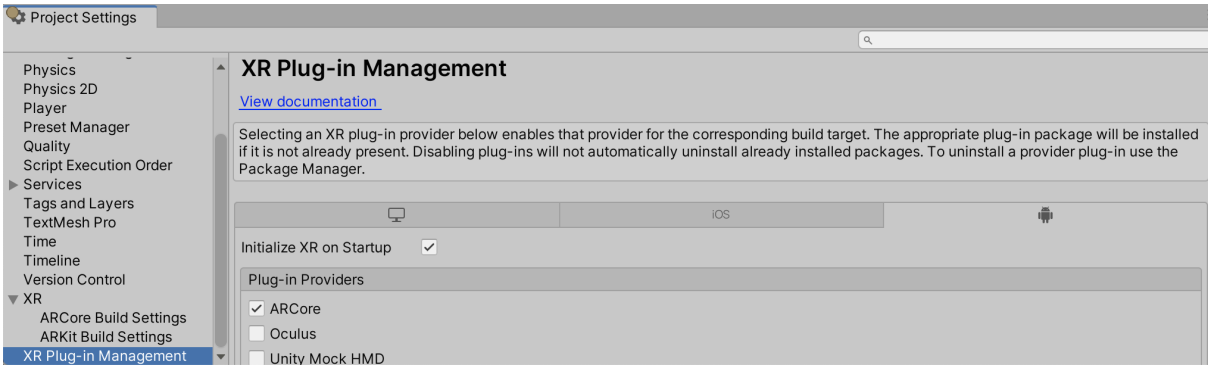

Fig 30. Ventana de administración de Unity XR

Una vez activada la librería podemos crear en nuestra escena de Unity componentes de XR que son necesarios para que nuestra escena utilice realidad aumentada. Para el correcto funcionamiento de la escena debemos crear una sesión de realidad aumentada, que inicializa los parámetros de la librería y crea una instancia en la escena que permite gestionar sus características.

La librería es muy amplia y soporta varios métodos de realidad aumentada, en el caso de este proyecto vamos a utilizar la detección de planos, que permite detectará superficies planas a través de la interacción de la cámara con el entorno, una vez detectado uno o varios planos el usuario puede interactuar con el dispositivo y seleccionar en qué lugar va a colocar el contenido virtual.

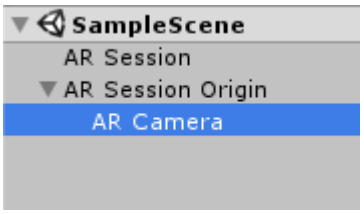

Fig 31. Componentes iniciales de la escena

El método de detección de planos se eligió por que es el más práctico y compatible con mayor cantidad de dispositivos una vez que se coloca el contenido virtual el algoritmo deja de escanear el espacio y deja fijo el punto de anclaje, lo que mejora el rendimiento. Otra característica que podría utilizarse es la nube de puntos (Point Clouds) que es un método más preciso pero que detecta todas las superficies que capta la cámara, para crear posibles puntos de anclaje más reales y de alguna

manera “virtualiza” las superficies que detecta la cámara. Este método además de ser más complejo, dejaría afuera muchos dispositivos, ya que se requiere un hardware más potente para poder utilizar esta característica.

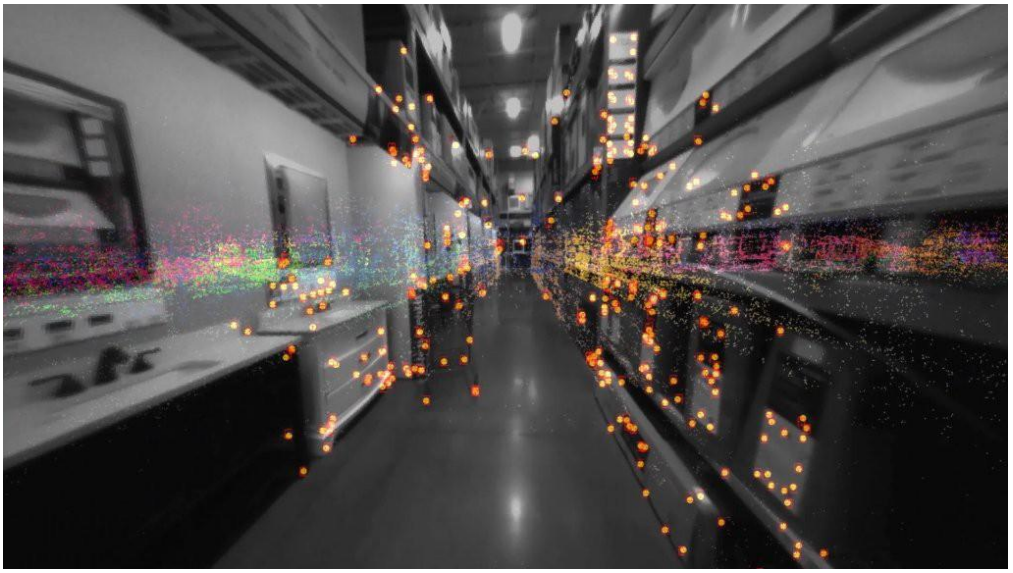

Fig 32. Método Cloud Points

Una vez seleccionado el método detección de planos se define el origen de la sesión, que contiene la cámara de AR, a través del componente ARPlaneManager, que permite definir qué objetos se va a dibujar cuando se detecte un plano y que tipo de plano nos interesa detectar (horizontales, verticales o todos).

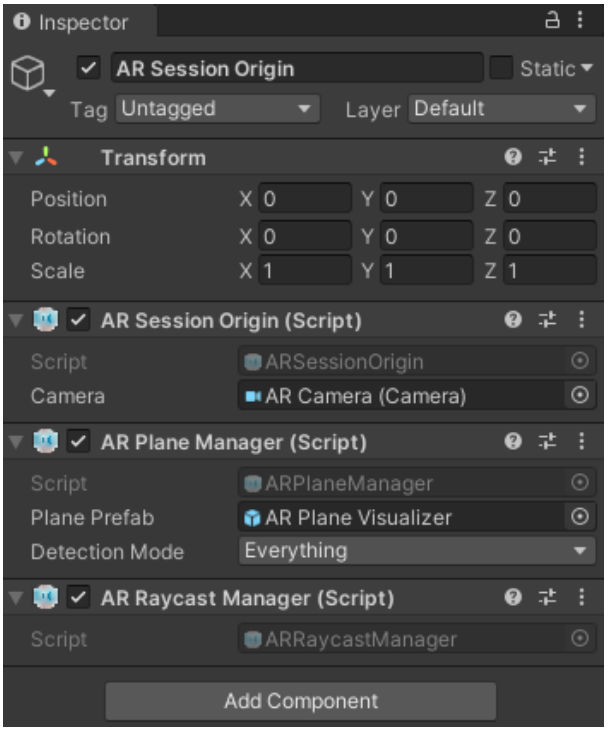

Fig 33. Componentes de la Sesión de Realidad aumentada

Además para que el usuario pueda interactuar con estos planos agregamos el componente ArRaycastManager, que permite que utilizar la posición de la cámara del dispositivo para generar interacciones con el entorno, es decir cuando el usuario toca la pantalla este componente crea un “rayo” que utiliza la información del entorno y la posición de la cámara para simular la posición real que tocó el usuario tocó en la pantalla. Un Raycast es un componente habitual en el desarrollo de videojuegos, ya que es un “rayo” que se desplaza por el entorno 3D y cuando colisiona con otro componente permite recuperar el punto de impacto y la información del otro objeto.

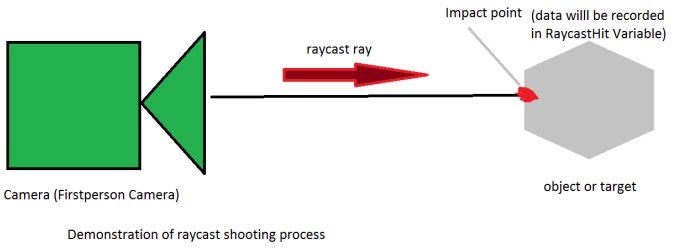

Fig 34. Utilización tradicional del Raycast en juegos de primera persona.

En un proyecto de realidad aumentada es importante que el usuario pueda interactuar con el entorno, en este sentido la librería XR tiene implementada varias características que facilitan el desarrollo de estas interacciones, por ejemplo tiene integrado una vista por defecto para que el usuario puede “corregir” la ubicación del contenido AR, permitiendo cambiar su posición, rotación y escala dentro del plano.

IV.13. Carga Dinámica de niveles

Una vez configurada la sesión de RA debemos empezar a construir el nivel del juego, en la versión anterior de invasión aumentada teníamos una escena por nivel por lo que cada vez que cambiamos de nivel debíamos detectar nuevamente el marcador. En este proyecto si bien se usa la detección de planos para la realidad aumentada persiste el mismo problema, ya que cambiar de escena reiniciará la sesión de RA y el usuario debería volver a buscar el sitio para colocar el contenido

del juego nuevamente. Para solucionar este problema tenemos que mantener la escena principal del juego activa y crear “sub-escenas” con el contenido individual de cada nivel, Unity nos permite cargar la escena dentro de otra y traer todos sus GameObjects y de la misma manera descargarla cuando ya no la necesitemos para liberar recursos. Es importante que la lógica del juego integre esta característica para que el juego se pueda jugar en una única sesión de AR y el GamaManager gestione de forma eficiente la carga de niveles, permitiendo tener una sesión de juego fluida y sin tiempos de cargas extensos que rompan la inmersión. Para lograr esto una vez que se carga un nivel el LevelManager debe tomar de forma dinámica las características del nivel cargado y refrescar el estado del juego, es importante definir claramente qué objetos serán persistentes entre escenas y cuales son eliminados cuando se carga un nuevo nivel. Todos los objetos de la lógica del juego, de la UI y de la sesión de realidad aumentada estarán cargados en la escena principal desde la cual se cargará el contenido del nivel que el usuario elija.

```
/*
* Cargar la scena en modo Aditivo permite que el contenido
* de la escena cargado se añada a la escena principal
*/
void LoadAsyncLevel(int LevelID)
{
    SceneManager.LoadSceneAsync(LevelID, LoadSceneMode.Additive);
}

//Destruye todos los Gameobjects de la escena a descargar
void UnloadAsyncLevel(int LevelID)
{
    SceneManager.UnloadSceneAsync(LevelID);
}
```

Fig 35. Funciones para administrar la carga asíncrona de niveles

IV.14. Conceptos a rediseñar

A partir de los definido en el documento de diseño se llevará adelante el proceso de rediseño de las siguientes características del juego:

IV.14.1. Diseño de niveles

Los niveles del juego son escenas donde está el entorno 3D y los componentes de dicho nivel, como enemigos, powers-ups y objetos interactuables. En la etapa de prototipado podemos usar recursos genéticos para representar la geometría del nivel y los actores de la misma, con el fin de tener de forma temprana un espacio donde poder probar todas las mecánicas del juego, luego es fácil reemplazar el contenido genérico por el contenido final del juego, una vez que sepamos que todos los componentes funcionen según las mercancías del juego.

IV.14.2 Diseño de enemigos

En la versión original los enemigos permanecen estáticos en sus posiciones, si bien estaban animados carecían de una inteligencia artificial real, para este proyecto además de diversificar los tipos de enemigos se pretende darles una IA básica que le permita interactuar con el entorno y darle al jugador una experiencia más inmersiva. Para esto debemos redefinir la clase de enemigo, representando su estado en un diagrama de comportamiento y permitiendo extender esta clase a distintos tipos de enemigos con distintas características.

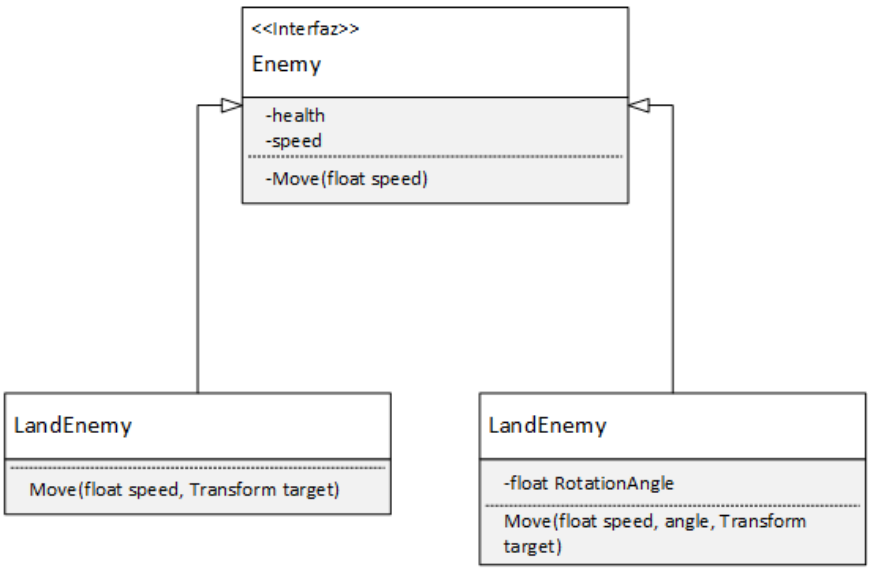

Fig 36. Diagrama de clase de los enemigos

IV.14.3. Diseño de proyectiles

Los proyectiles son una de las características principales del juego, en la versión original podíamos lanzar unos personajes llamados “Zips”, en esta revisión la idea es cambiar los proyectiles por “hechizos” para poder crear proyectiles con mejores efectos, tanto visuales como mecánicos para darle más variedad al juego. En la versión demo se va a tener una sola versión de cada disparo, pero lo ideal sería que puedan tener varios niveles y limitar el uso de proyectiles por cada nivel, añadiendo una capa de profundidad al gameplay, permitiendo que el jugador tenga que elegir estratégicamente qué proyectiles le convienen para cada nivel.

IV.14.4. Diseño de la Economía y monetización del juego

Esta característica va ser omitida en la versión demo, pero la idea es que el juego tenga una economía que le permita generar ingresos mediante anuncios y un sistema de recompensas. En una aplicación de realidad aumentada los anuncios pueden ser bastante molestos y rompen la sensacion de inmersion, para eso el juego debe tener un sistema de monetización bien pensado, mostrando solo anuncios temporales en momentos específicos, por ejemplo durante la carga de un nivel o luego de que el jugador pierde una determinada cantidad de veces. Además se pueden ofrecer recompensas al jugador por ver videos de forma voluntaria, estas recompensas suelen ser puntos u objetos cosméticos.

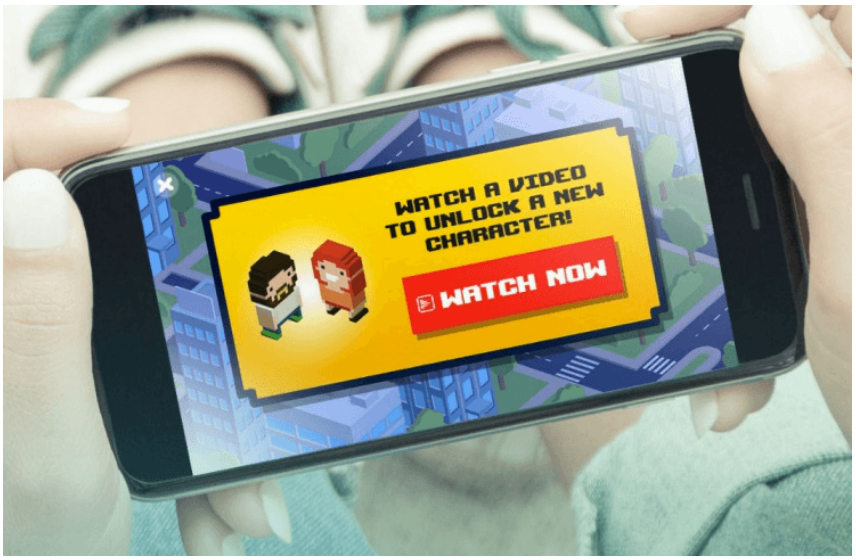

Fig 37. Ejemplo de monetización con videos que el usuario puede ver voluntariamente

IV.15. Desarrollo del prototipo

El desarrollo de un prototipo es muy importante para desarrollar un videojuego, ya que en este se verán implementadas todas las mecánicas del juego, centrándose en la funcionalidad del juego. En esta etapa no se profundiza en la parte estética del juego, generalmente se utilizan recursos genéricos que representan los objetos del juego y luego se los reemplaza por los recursos reales y se les agregan las animaciones, efectos y todos los componentes que requieren para la versión demo.

IV.15.1. Niveles de prueba

Es habitual en el desarrollo de videojuegos realizar niveles de prueba (Test Level) donde se puedan cubrir todos o la mayoría de los casos que puedan existir en los niveles reales del juego. Esto permite que se puedan probar las mecánicas del juego libremente y tener en cuenta las posibles interacciones entre los componentes, por ejemplo podemos ubicar varios enemigos de diferentes tipos y ver si hay colisiones entre sus comportamientos, permitiendo ajustar la IA.

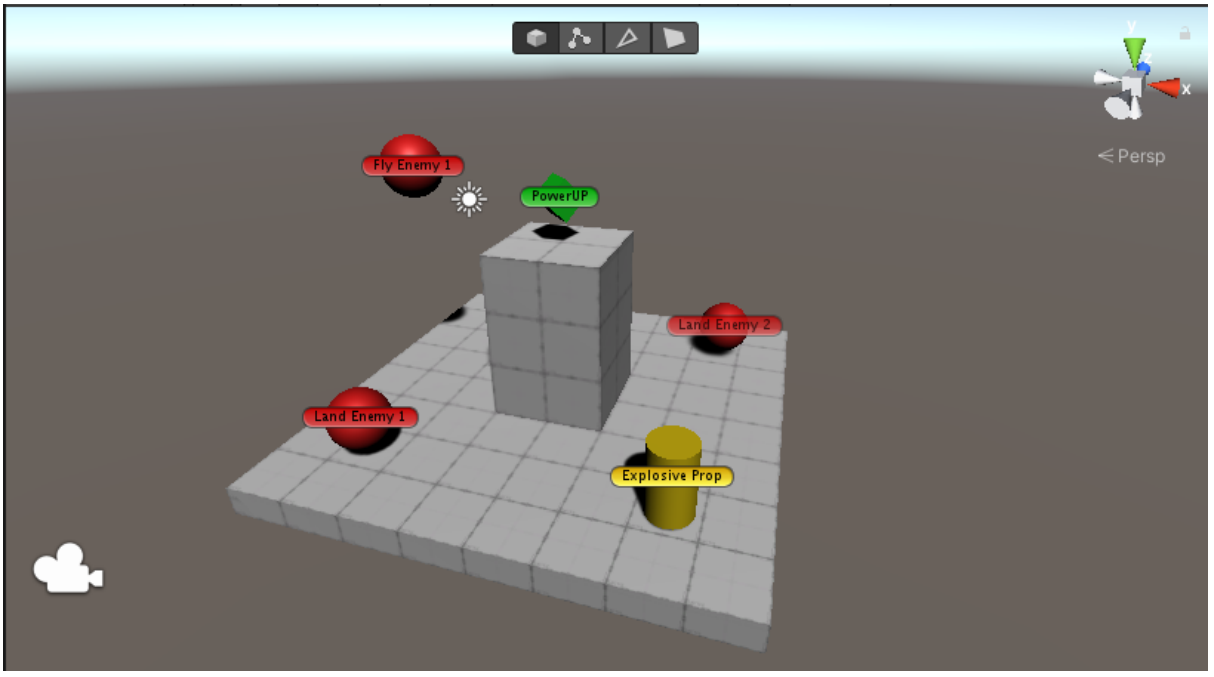

Fig 38. Nivel de prueba desarrollado

IV.15.2. Inteligencia artificial de los enemigos:

Como se definió anteriormente es necesario implementar un algoritmo de inteligencia artificial, para ello podemos aprovechar la herramienta de Pathfinding de Unity, que nos permite implementar NavMesh (mallas de navegación) que nos permite definir un mapa por el cual determinados objetos (Nav Mesh Agents) pueden desplazarse por dichos mapas y definir rutas por las cuales se puedan desplazar los enemigos.

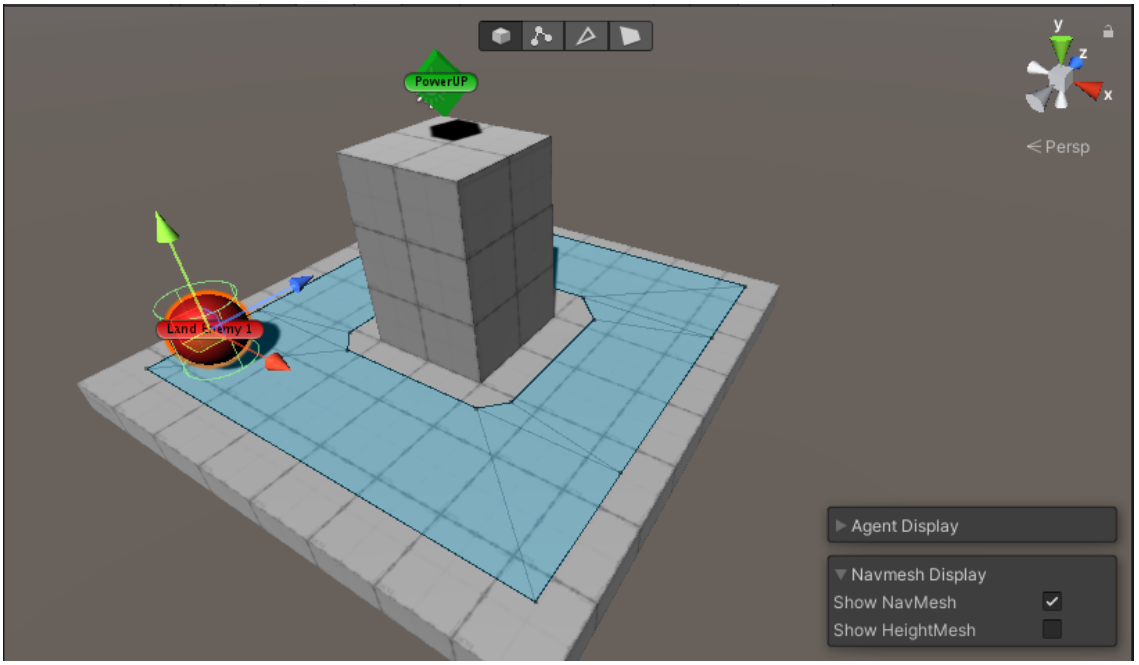

Fig 39. Mapa de navegación del nivel de prueba

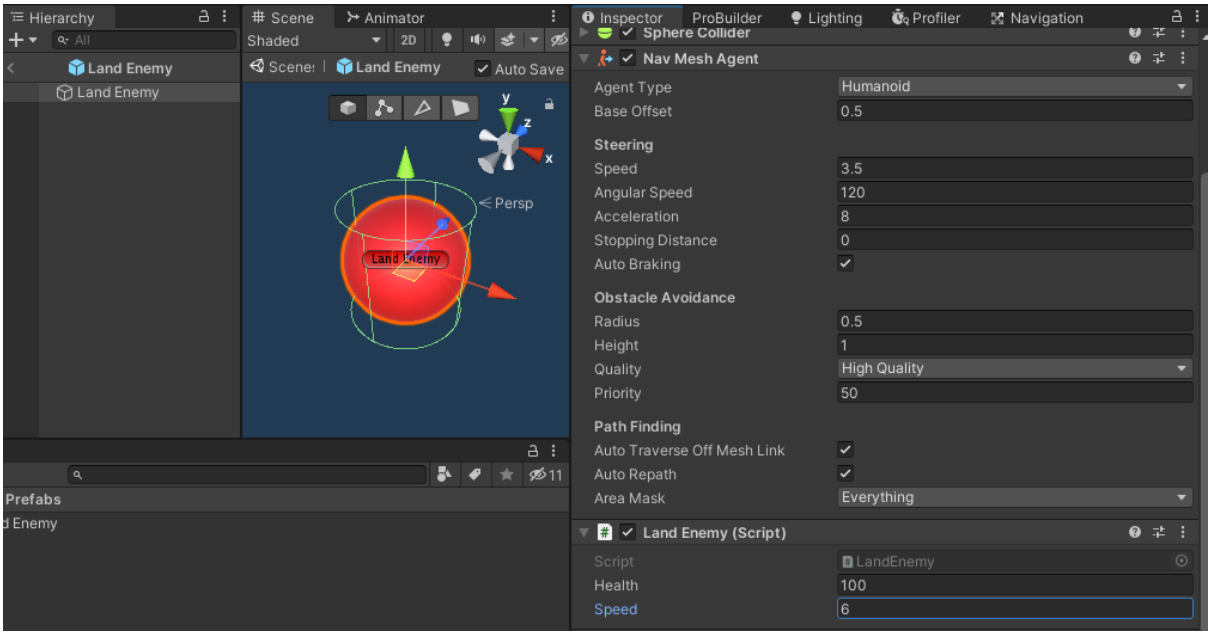

Fig 40. Prefab de LandEnemy

IV.15.3. Algoritmo de disparos:

Para poder probar todas las mecánicas implementadas con realidad aumentada es necesario que el jugador pueda interactuar con el nivel a través del dispositivo móvil, para eso el jugador puede lanzar los proyectiles al nivel, tanto como para golpear a los enemigos y objetos del nivel como para adquirir los powers-ups. Previamente se explicó la lógica de este algoritmo, que es totalmente reutilizable en esta versión, este algoritmo corre en una función Update, que es una función predefinida de Unity que se ejecuta una vez por cuadro (frame), es decir que se está ejecuta constantemente en la ejecución del juego. Teniendo esto en cuenta tenemos que asegurarnos que solo se ejecute cuando el jugador interactúe con la pantalla

```
//Inicializo las variables cuando el usario toca la pantalla
if (Input.touchCount > 0 && Input.GetTouch (0).phase == TouchPhase.Began)
{
    final = Vector3.zero;
    length = 0;
    angulo = 0;
    SW = false;
    Vector2 touchDeltaPosition = Input.GetTouch (0).position;
    startpos = new Vector3 (touchDeltaPosition.x, 0, touchDeltaPosition.y);
}
```

Fig 41. Extracto de la función Update del algoritmo de disparo

La clase Input maneja todas las entradas de Unity, en este caso comprobamos la entrada de la pantalla táctil para indicar los valores del algoritmo sólo si comienza a realizar un trazado en la pantalla.

Si el trazado se termina el algoritmo debe tomar el vector que formó la interacción del jugador con la pantalla y a partir de la posición del dispositivo y los valores del vector obtenido determinar la fuerza y dirección del proyectil que se va crear:

```
if (Input.touchCount > 0 && Input.GetTouch (0).phase == TouchPhase.Ended)
{
    if (SW)
    {
        coef = fuerzamax/(Screen.height / 2);
        Vector2 touchPosition = Input.GetTouch (0).position;
        endpos = new Vector3 (touchPosition.x, 0, touchPosition.y);
        final = endpos - startpos;
        Vector2 delta = endpos - startpos;


    float angle = Mathf.Atan (delta.y/delta.x) * (180.0f/Mathf.PI);
        length = final.magnitude;
        float fuerza = length*coef;
        angulo = angle;
        if(length > 100) {
            DisparoSimple(fuerza, angle);

        }
    }
}
```

Fig 42. Extracto de la función Update del algoritmo de disparo donde se calcula la dirección y fuerza del proyectil a partir de la interacción con la pantalla.

Parte final del algoritmo de disparo, donde se toma el vector del trazado y se lo convierte en fuerza y dirección para el disparo.

```
float SignedAngleBetween(Vector3 a, Vector3 b){
    // angle in [0,180]
    float dot = Vector3.Dot(a,b);
    // Divide the dot by the product of the magnitudes of the vectors
    dot = dot/(a.magnitude*b.magnitude);
    //Get the arc cosin of the angle, you now have your angle in radians
    var acos = Mathf.Acos(dot);
    //Multiply by 180/Mathf.PI to convert to degrees
    var angle = acos*180/Mathf.PI;

    return angle;
}
```

Fig 43 Función de apoyo para calcular el ángulo entre los vectores.

IV.16. Desarrollo de la versión Demo

IV.16.1 Selección de recursos Audiovisuales (Assets)

A partir del prototipo funcional que implementa todas las técnicas básicas del juego el siguiente paso es implementar una versión demo que sea un versión más aproximada al juego final, para esto es necesario agregar los recursos audiovisuales que forman parte del proyecto y reemplazar el contenido del prototipo por estos recursos. Para adquirir esos recursos se recurrió a la tienda de activos de Unity (Unity Assets Store):

IV.16.1.1 Modelos 3D y efectos visuales

- Escenario: necesitamos construir un escenario a partir de modelos 3D, se utilizó el siguiente paquete de recursos:

Fig 44. Cartoon Temple Building Kit Lite. A3D. Unity Assets Store

- Enemigos: para representar a los enemigos se utilizó el siguiente paquete de modelos 3D

Fig 45. Level 1 Monster Pack. PI Entertainment Limited. Unity Assets Store

- Efectos visuales: para representar los proyectiles, explosiones y demás efectos.

Fig 46. Toon Magic Collection. Unluck Software. Unity Assets Store

IV.16.1.2. Sonidos

Se eligió la librería de sonidos “FREE Casual Game SFX Pack” que cubre la mayor parte de sonidos necesarios para la versión Demo.

IV.16.2 Diseño de niveles

Utilizando el nivel de prototipo como base podemos utilizar los recursos mencionados para construir el primer nivel de la versión Demo, es importante respetar la estructura del nivel, que se diseñó con la intención de poder probar todas la variaciones de las mecánicas del juego.

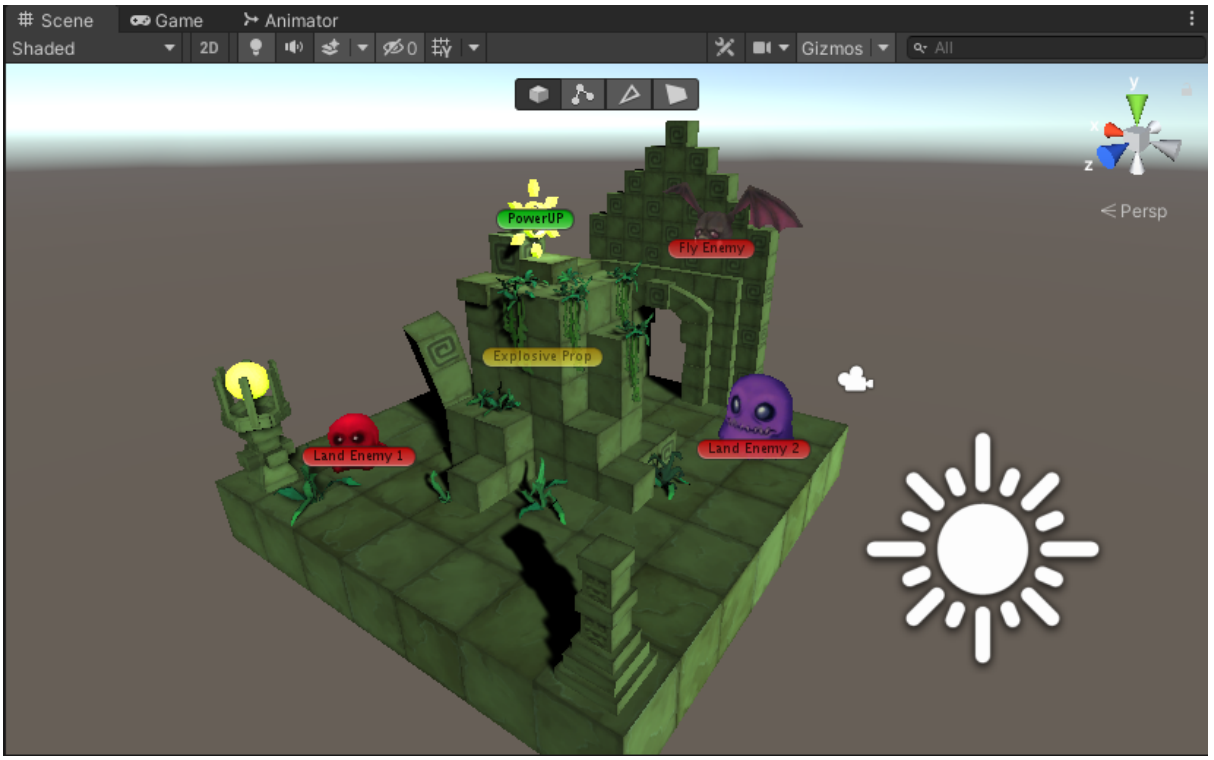

Fig 47 Nivel Demo basado en el nivel del prototipo

Además de tener en cuenta la distribución de los objetos en el nivel base, el nivel del demo tiene que mejorar estéticamente, por lo que se agregan objetos decorativos y efectos visuales, que aportan desde lo visual, pero sin alterar ninguna de las mecánicas del juego. Además es importante tener en cuenta parámetros como la iluminación y la calidad de la texturas, que pueden influir en el rendimiento del juego en dispositivos móviles.

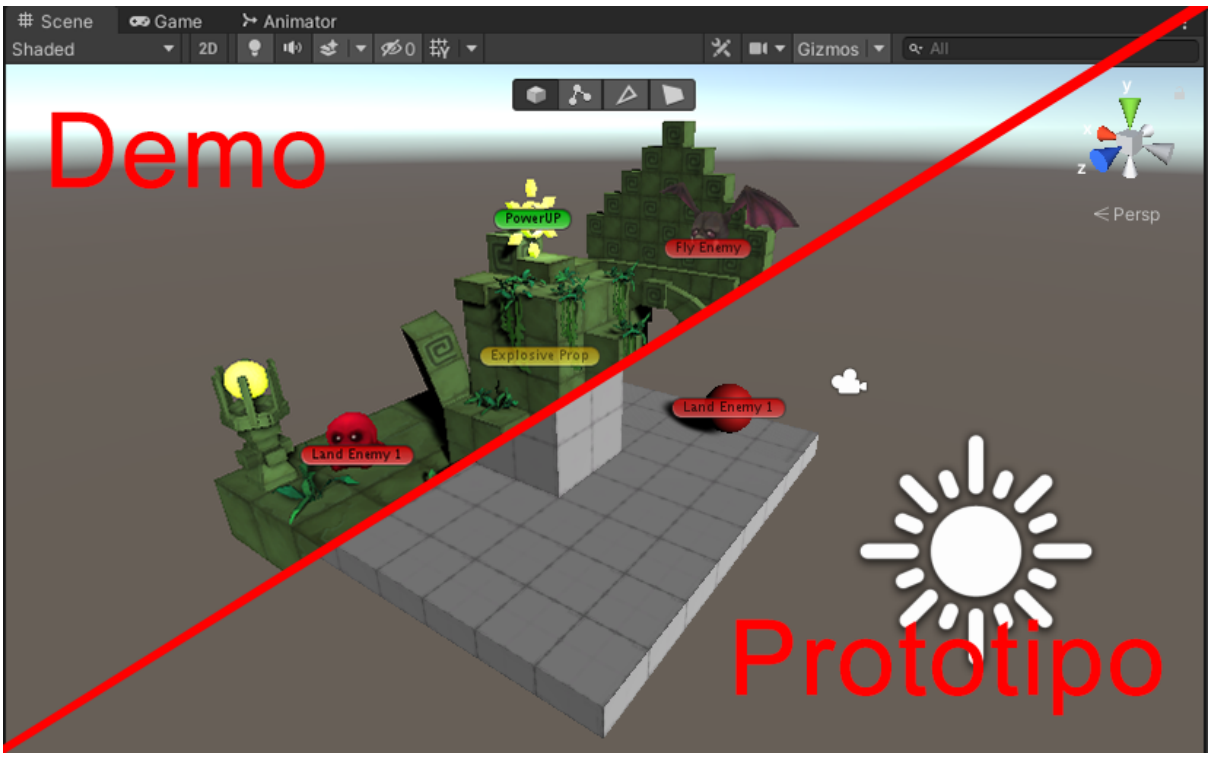

Fig 48. Vista comparativa de las escenas de Prototipo y Demo

IV.16.3 Enemigos:

Para poder integrar los modelos 3D descargados con la lógica de los enemigos ya existente es importante crear un componente animador (Animator) que nos permite relacionar el diagrama de estados de la inteligencia artificial con un árbol de animaciones que representen esas transiciones, por ejemplo cuando el enemigo es golpeado si sobrevive debe volver a componerse antes de volver a moverse, pero si no sobrevive la animación que debe resarcirse es la de la muerte.

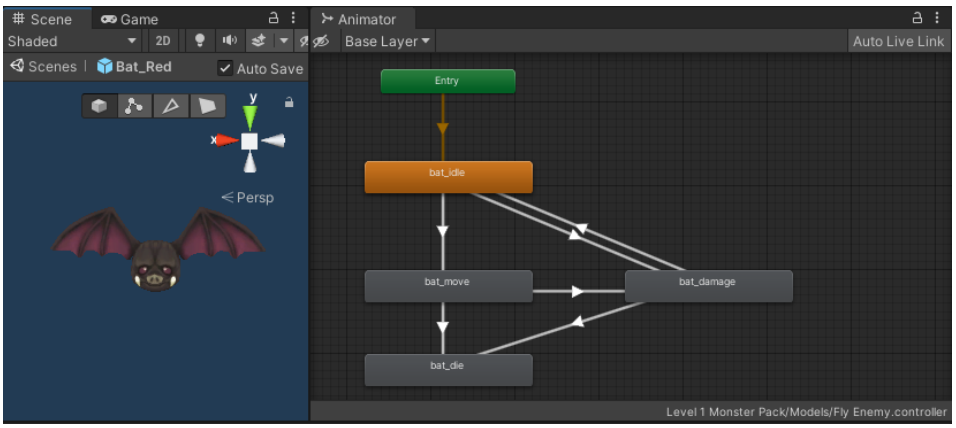

Fig 49. Diagrama de animación de un enemigo

IV.16.4. Disparos:

La lógica de los disparos se conservó íntegramente como estaba en el proyecto original ya que funcionan correctamente en la nueva versión, esto a nivel lógico, pero estéticamente debemos cambiarlos para unificar el juego estéticamente, en la versión original los disparos eran representados por modelos 3D estáticos, en esta implementación se utilizarán sistemas de partículas para tener disparos más dinámicos visualmente. Mediante la adaptación de los efectos de partículas descargados se logró que dejen una estela, que mejora la visualización del trayecto del disparo, lo que permite al usuario saber por donde pasa su disparo (al tratarse de realidad aumentada la estela iría desde la cámara hasta el punto de impacto) para tener una mejor idea de cómo interactuar con el algoritmo de disparo.

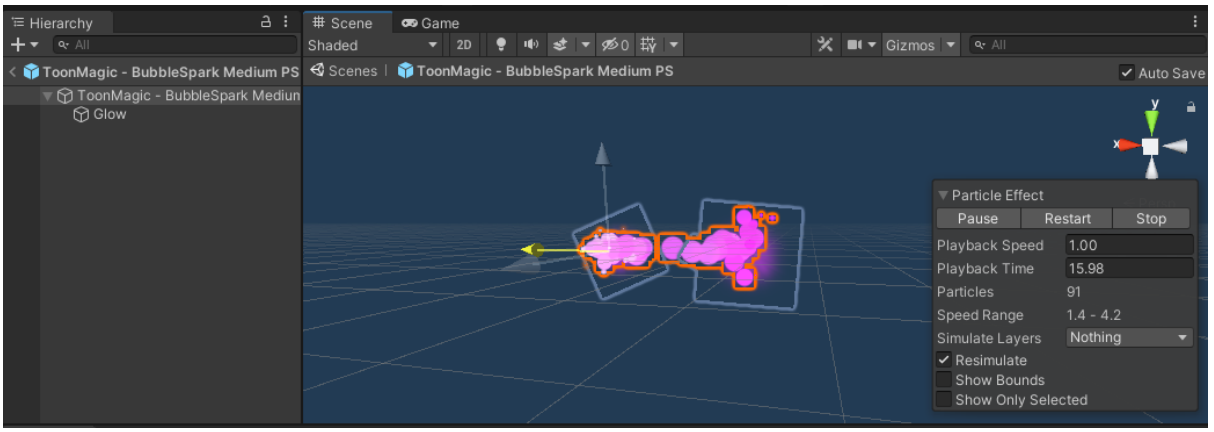

Fig 50. Partícula del disparo en movimiento

IV.16.5. Integración de Sonidos:

La cámara por defecto de Unity tiene un componente AudioListener que se encarga de captar los sonidos reproducidos por otros Gameobjects, en nuestro caso ese componente se encuentra en la cámara de la sesión de realidad aumentada, para que un GameObjects pueda reproducir sonidos necesita un componente Audio Source correctamente configurado. Este componente tiene todas las propiedades relacionadas al sonido y puede modificarse mediante Scripts, lo que permite cambiar el archivo de sonido, el volumen entre otras propiedades y para reproducir el sonido en el momento deseado. También podemos asociar los sonidos a los estados de animación con el editor Animator, lo que posibilita por ejemplo que en la animación “damage” del enemigo asociar un evento que reproduzca el sonido

elegido para cuando un enemigo es dañado. De todas maneras la mejor forma de evitar errores es poner los eventos de sonidos en la lógica del juego y no en las animaciones.

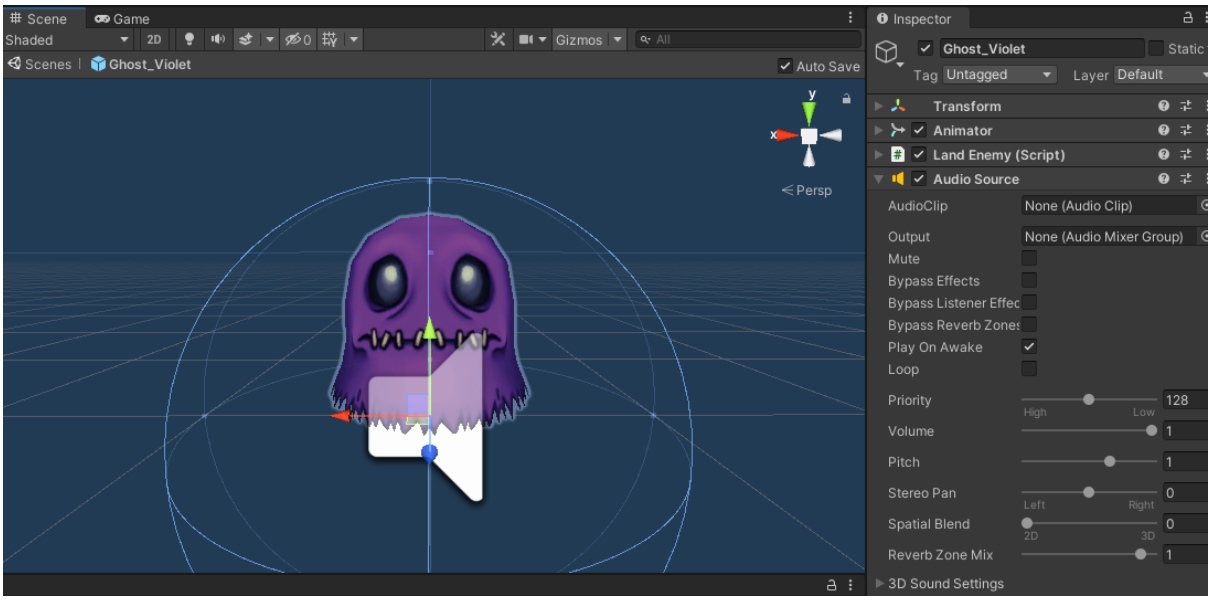

Fig 51. Componente Audio Source en enemigo

V. Verificación y análisis de los resultados.

En el presente capítulo, se llevarán a cabo las pruebas correspondientes sobre la solución desarrollada y se expondrán los resultados obtenidos.

Para ello se realizarán diferentes pruebas tanto desde el punto de vista tradicional del desarrollo de software como desde el punto de vista del diseño de videojuegos, mediante el análisis de parámetros como el balance del gameplay o la dificultad de los niveles.

V.1.Test de Unidad

Los test de unidad (Brian Robinson, Michael D. Ernst, Jeff H. Perkins 2011, pp. 23-32) son un método de prueba que verifica qué componentes individuales del código fuente funcionan correctamente. Para ello, el creador de los test implementa ejemplos de uso directamente en código fuente que es ejecutado durante el proceso

de pruebas. Si la ejecución no obtiene los resultados esperados se dice que el test falla o no pasa, y por lo tanto se considera que el código probado es incorrecto. A partir de este método podemos probar ciertas partes de la lógica del juego para comprobar que funciona correctamente, para ello Unity tiene una herramienta integradas para realizar los test de unidad, que permite correr las pruebas en tiempo real mientras se interactúa con el juego. Esto es importante por que si bien podemos simular la entrada del jugador siempre puede tener un comportamiento impredecible, por lo que conviene tener esto en cuenta a la hora de probar un videojuego.

Se habilitó la herramienta Unity Test Runner y podemos crear los test necesarios para validar la ejecución de las unidades implementadas en la solución, por ejemplo podemos un test para validar si el juego está balanceado. Por ejemplo enemigos otorgan un puntaje determinado, pero si un proyectil impacta a más de un enemigo el puntaje se multiplica, por lo que podemos correr un test que compare el puntaje esperado para este nivel, que evita que el jugador pueda obtener una cantidad de puntos fuera de lo esperado, lo que demostraría que el sistema de puntaje está desbalanceado y hay que ajustarlo.

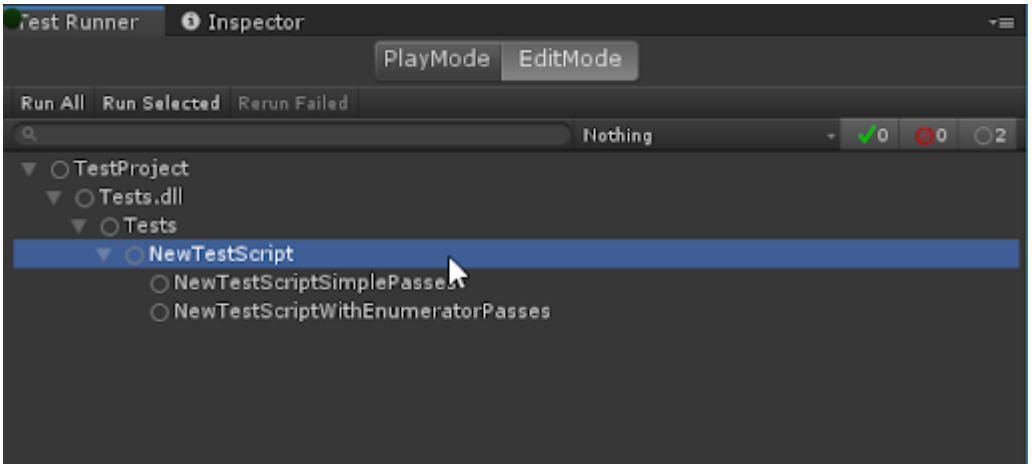

Fig 52: Vista de Test Runner en Unity

Test de Unidad implementados para probar la solución:

Script	descripción	estado
Level Manager	Comprobar puntaje obtenido con el esperado	fallo
LandEnemy	Comprobar daño esperado a enemigos	éxito
Projectile	Comprobar velocidad máxima	éxito
Fly Enemy	Animaciones asociadas a los estados de IA	éxito

Después de correr estas pruebas tenemos una mejor idea de cómo funciona la solución implementada, en el caso de test fallido significa que se debe revisar la forma en que el jugador obtiene los puntos, ya sea por la suma de un valor fijo por enemigo o la definición de un multiplicador de puntos más claro.

V.2. Pruebas de Rendimiento

Para un videojuego el rendimiento es un factor importante que influye en la experiencia que tiene el jugador al interactuar con el juego, hay diferentes factores que pueden influir negativamente la experiencia del usuario al interactuar con el proyecto, por ejemplo el Input Lag (Yuji, H. 1996). Este es un problema de rendimiento donde la interacción del usuario con los controles, sea en cualquiera de sus formas (teclado, mouse, joystick, pantalla táctil) llega con retraso al juego, lo que afecta la interacción en tiempo real, una de las características principales de un videojuego.

Generalmente el problema de rendimiento más conocido está relacionado con los FPS[11] (Orosy-Fildes, C, Allan, R. 1989) que influyen directamente en la fluidez visual que tiene el juego mientras se ejecuta, una tasa muy baja de

[11] Cuadros por segundo

fotogramas afecta mucho la fluidez de las animaciones y la calidad de imagen en general, en este caso a tratarse de una aplicación de realidad aumentada si el juego no se fluido además de los problemas antes mencionados hay una disonancia con lo que usuario interactúa con el juego a través de su cámara en tiempo real.

Hay mucho debate sobre cuál es la tasa de FPS “aceptable” pero el estándar está entre los 30 y 60 cuadros por segundo. Para mantener una tasa estable de FPS es importante tener en cuenta diferentes factores relacionados con la utilización de la unidad de procesamiento gráfico o GPU (graphics processing unit) que es el componente del dispositivo que se encarga de ejecutar los comandos gráficos y dibujarlos en la salida gráfica del dispositivo.

Para tener una idea del rendimiento en tiempo real del videojuego tenemos la herramienta Unity Profile, que nos permite ver el consumo de recursos de nuestra aplicación en tiempo real, mediante la grabación de los valores de rendimiento durante una sesión del juego para analizar en qué momentos hay altas o bajas en el consumo de recursos y a partir de su análisis poder validar el rendimiento de la solución o analizar qué cambios son necesarios para mejorar el rendimiento del juego en tiempo real.

A partir del análisis de la aplicación en tiempo real se notó que su rendimiento es aceptable y que gracias a la optimización de los modelos 3D (que tienen poca cantidad de polígonos) y de los efectos visuales simples pero visualmente agradables se mantiene una tasa estable de 48 fps, y con un consumo de recursos moderado, teniendo en cuenta que además de usar la cámara del dispositivo la librería de realidad aumentada utiliza otros sensores.

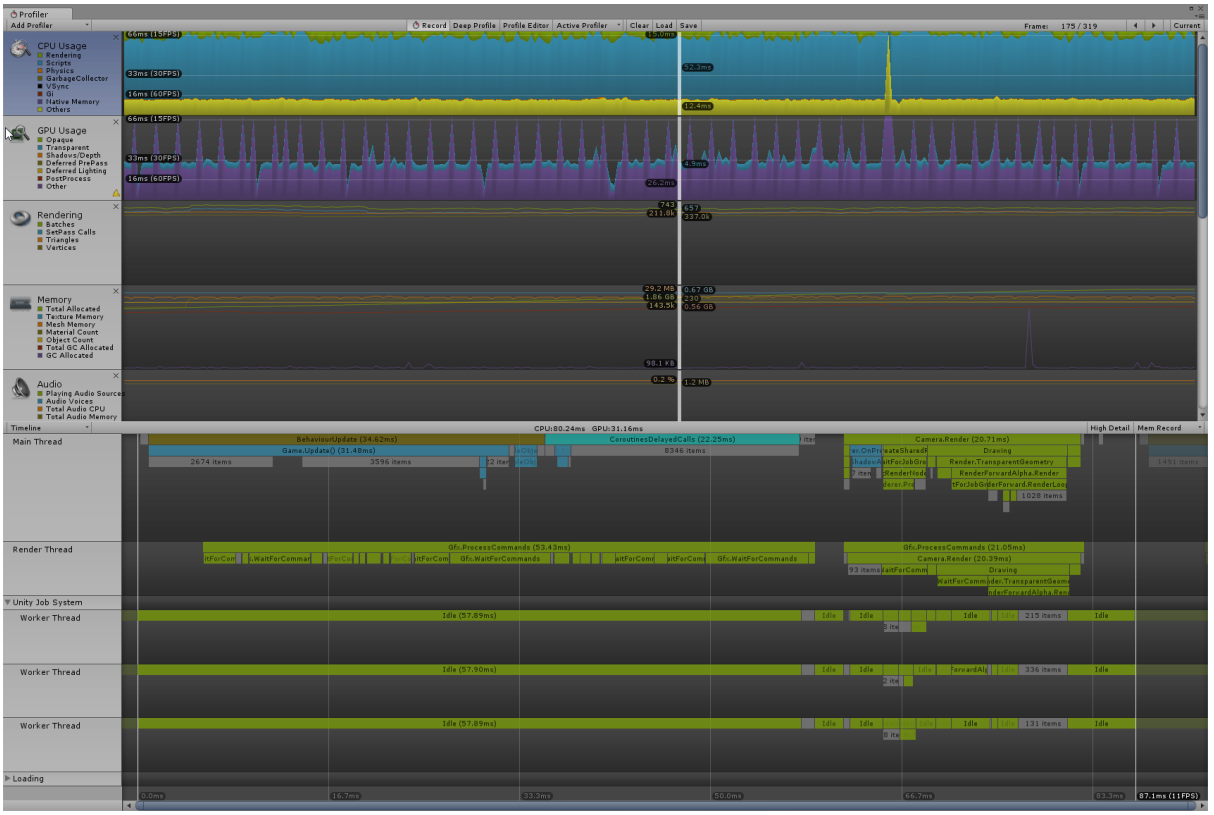

Fig 53. Vista de Unity Profiler

V.3. Pruebas de Realidad Aumentada

Una vez validado el funcionamiento de la lógica del juego y el rendimiento del mismo es importante comprobar que la solución brinda una forma simple de interactuar con la realidad aumentada, ya que la detección de planos (tecnología implementada en la aplicación) requiere una serie de pasos por parte del usuario para poder ubicar el contenido de realidad aumentada mediante su cámara. Para esto se propone hacer ciertas pruebas a la usabilidad de la aplicación en relación a la realidad aumentada, para determinar si en una versión futura sería necesario implementar un nivel tutorial o una serie de pasos que expliquen cómo utilizar la realidad aumentada de la aplicación.

En la versión original de invasión aumentada existía un tutorial que explicaba al usuario como sacar la foto que se utilizará como marcador, además de una serie de consejos de como capturar la mejor foto posible. En el caso de la detección de planos de la solución actual algunas aplicaciones utilizan una serie de pasos que guían al usuario durante los primeros pasos, desde detectar el espacio que rodea al

usuario hasta la selección del lugar donde se colocará el contenido de realidad aumentada y una guía para poder acomodar el contenido. Como observamos en la siguiente figura la interfaz es bastante simple pero carece de explicativos, por lo que en versiones futuras sería necesario sumar el tutorial mencionado anteriormente.

Fig 54. Colocando el nivel de prueba con realidad aumentada

VI. Conclusiones

Una vez desarrollada y validada la versión demo de la solución podemos concluir que se obtuvo el resultado esperado, se pudo realizar exitosamente el proceso de remake que consistió en tomar el proyecto original del 2014 como base para crear un nuevo proyecto que tome sus mejores características y utilice las nuevas tecnologías de realidad aumentada.

Desde el punto de vista de la arquitectura de software se pudo analizar la estructura original del proyecto que gracias a separar la lógica del juego de los componentes de realidad aumentada se pudieron reutilizar la mayoría de componentes para la versión nueva, lo que demuestra que la arquitectura original era flexible al cambio de librerías.

El proyecto también se vio beneficiado por las metodologías ágiles que permitieron un desarrollo enfocado en tener piezas funcionales en etapas tempranas del desarrollo, que permitió implementar primero las mecánicas básicas en el prototipo y después añadir las funcionalidades más específicas para la versión Demo.

En cuanto a la utilización de la Realidad Aumentada, teniendo en cuenta la tecnología que se utilizaba en la versión original podemos ver un gran avance tecnológico en un lapso de unos pocos años, gracias a la mejora constante de los dispositivos móviles. Esto se debe tanto en cámaras como en nuevos sensores permiten que hoy en día la tecnología de realidad aumentada pueda utilizar espacios físicos como anclaje y ya no necesite referencias específicas como marcadores o códigos QR. Este avance tecnológico permitió mejorar mucho la experiencia del juego con la tecnología e implementar nuevas características como la carga dinámica de niveles, ya que la estructura de sesión de Unity XR es mucho más flexible que la librería utilizada originalmente en el proyecto base.

Desde el punto de vista del diseño de videojuegos se respetaron todas las mecánicas definidas inicialmente, ya que es una buena práctica en un remake respetar la esencia del juego original. En esta versión se hizo un cambio a la parte estética del juego, para evitar la mezcla de distintos estilos que tenía la versión original, que buscaba tener niveles “hiper-realistas” pero tenía enemigos y disparos de diferentes estéticas. Si comparamos ambas versiones podemos observar la neuva además de estar unificada estéticamente simplifica la utilización de modelos 3D y efectos especiales para lograr ser compatible con la mayoría de los dispositivos que utilicen esta tecnología, y tiene como foco que el juego se vea fluido y se pueda disfrutar de una experiencia lo más inmersiva posible.

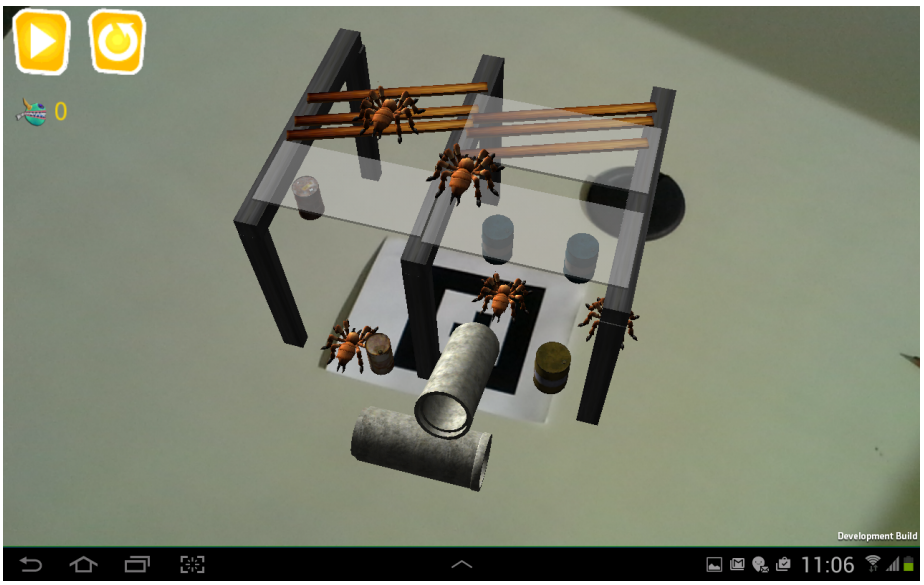

Fig 55. Invasión aumentada 2014

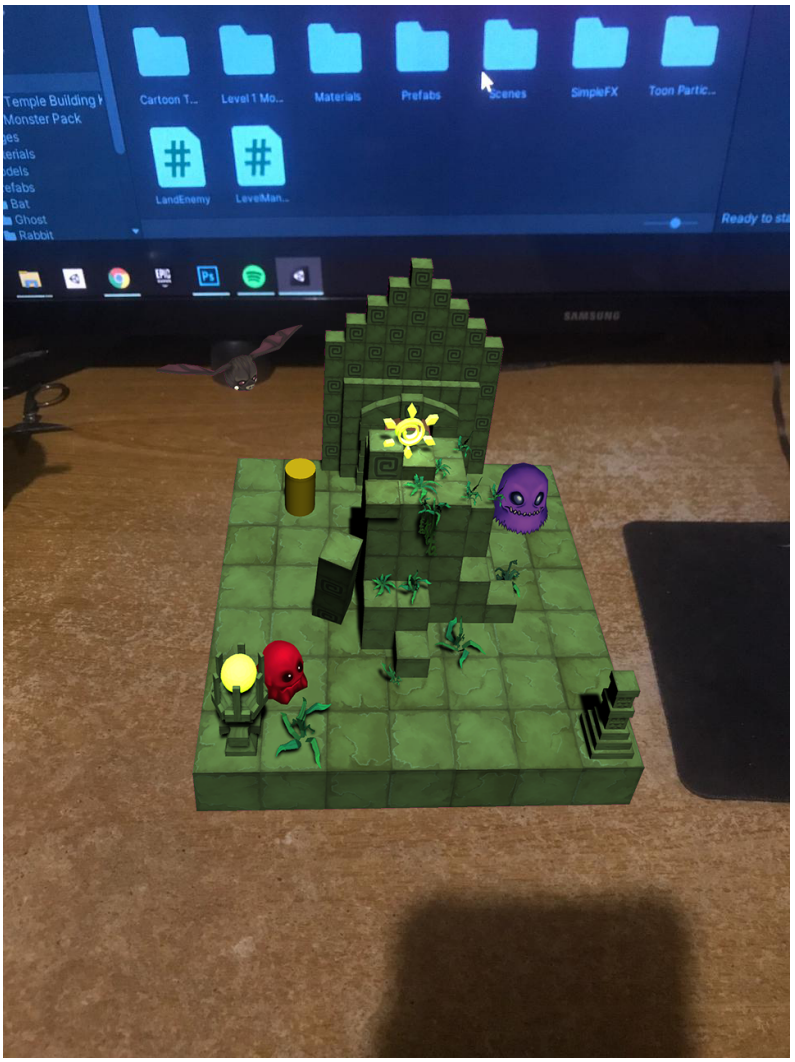

Fig 56. Invasión Aumentada 2020

VII. Referencias bibliográficas.

- Cawood, S., & Fiala, M. (2008). Augmented reality: a practical guide..
- Azuma, R. T. (1997). A survey of augmented reality. Presence: teleoperators & virtual environments, 6(4), p. 355-385.
- Carmigniani & Furht (2011). Augmented Reality: An Overview
- Adams, E. (2014). Fundamentals of game design. Pearson Education.
- Anthropy, A., & Clark, N. (2014). A game design vocabulary: Exploring the foundational principles behind good game design. Pearson Education.
- Adams, E., & Dormans, J. (2012). Game mechanics: advanced game design. New Riders.
- Kent, S. L. (2010). The Ultimate History of Video Games: from Pong to Pokemon and beyond... the story behind the craze that touched our lives and changed the world (Vol. 1). Crown.
- Sheff, D. (2011). Game over: How Nintendo conquered the world. Vintage.
- Kniberg, H. (2007). Scrum y XP desde las trincheras. Estados Unidos: C4Media.
- Google AR Core sitio oficial (2021) [En linea] https://developers.google.com/ar
- Apple ARkit sitio oficial (2021) [En linea] https://developer.apple.com/augmented-reality/
- Documetnacion de Unity (2021) [En linea] https://docs.unity3d.com/Manual/index.html
- Robinson, B., Ernst, M. D., Perkins, J. H., Augustine, V., & Li, N. (2011, November). Scaling up automated test generation: Automatically generating maintainable regression unit tests for programs. In 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011) (pp. 23-32). IEEE.
- Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn,+ Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland y Dave Thomas.(2001, noviembre) Manifiesto por

el Desarrollo Ágil de Software. Utah. (2021) [en línea]. Disponible en: http://agilemanifesto.org/history.html.

- Thomsen, Mike (February 23, 2010). "History of the Unreal Engine". IGN. Archived from the original on July 12, 2017. [En linea] https://www.ign.com/articles/2010/02/23/history-of-the-unreal-engine.
- Yuji, H. (1996). Computer games and information-processing skills. Perceptual and motor skills, 83(2), 643-647.
- Orosy-Fildes, C., & Allan, R. W. (1989). Psychology of computer use: XII. Videogame play: Human reaction time to visual stimuli. Perceptual and motor skills, 69(1), 243-247.
- Musil, J., Schweda, A., Winkler, D., & Biffl, S. (2010, September). Improving video game development: Facilitating heterogeneous team collaboration through flexible software processes. In European conference on software process improvement (pp. 83-94). Springer, Berlin, Heidelberg.