



Universidad de Jaén

Escuela Politécnica Superior de Jaén

Plataforma para visualización interactiva de Realidad Aumentada. Caso de uso: videojuego móvil LandmARK.

Autor: Juan Carlos Fernández Pérez

Máster: Ingeniería Informática

Director: Lidia Ortega Alvarado - David Jurado Rodríguez

Departamento del director: Informática

Fecha: 05/09/2024

C R E A



UNIVERSIDAD DE JAÉN

D./D^a Lidia Ortega Alvarado y D./D^a David Jurado Rodríguez, tutor(es) del Trabajo Fin de Máster titulado: **Plataforma para visualización interactiva de Realidad Aumentada. Caso de uso: videojuego móvil LandmARk.**, que presenta Juan Carlos Fernández Pérez, autoriza(n) su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, septiembre de 2024

El estudiante

Los tutores

Juan Carlos Fernández Pérez

Lidia Ortega Alvarado

David Jurado Rodríguez

Agradecimientos

Redactar unos agradecimientos puede que sea la parte más difícil de cualquier trabajo que los incluya, no por la propia calidad de la redacción, sino por aquello que lo compone. Para muchos, una mención a las personas más cercanas es suficientemente satisfactorio; para otros, quizá necesiten incluir algo más que personas. Incluso hay quienes optan por omitir el capítulo y abordar el esfuerzo que supuso su trabajo.

Tras un largo planteamiento, aprovecho este espacio para agradecer a los tutores Lidia y David por su paciencia y, en especial, su comprensión con las incontables dudas que han rodeado al proyecto incluso antes de su comienzo. Por supuesto, gracias también a todo el Grupo de Gráficos y Geomática de Jaén por su acogida y brindarme la oportunidad de formar parte de sus proyectos como estudiante y como profesional.

Sería evidente mencionar ahora al resto de personas que tengo en mente mientras plasmo estas palabras, pero pienso que un simple cúmulo de letras no es significativo de los sentimientos que busco expresar. Todos vosotros sabéis lo importante que habéis sido no solo durante este proyecto, sino durante mi desarrollo como persona.

Con este trabajo he afrontado muros que parecían inexpugnables, barreras que cambiaron drásticamente el rumbo, tramos ocluidos por el desconocimiento y la falta de experiencia. En su mayoría, trabas mentales que cualquier creador debe desafiar en algún momento. Quizá el resultado final, la cima o la meta, no sea la solución anhelada, pero es el camino, la manera en que sorteamos todas esas trabas, eso que nos hace humanos y nos dota de lo necesario para continuar creando y creciendo.

Tabla de contenidos

1. Introducción	1
1.1. Contexto	2
1.2. Motivación	4
1.3. Objetivos	5
1.4. Normativa	6
1.5. Recursos del proyecto	6
1.6. Estructuración de la memoria	7
2. Documento de Diseño del Juego	9
3. Antecedentes	25
3.1. Origen del videojuego	25
3.1.1. Nuevos sistemas de juego	29
3.1.2. <i>Smartphones</i> y videojuegos móviles	32
3.1.3. Desarrollo de videojuegos	34
3.2. Géneros de videojuego: Estrategia	35
3.2.1. Historia de los videojuegos de estrategia	35
3.2.2. Otros subgéneros	36
3.3. Realidad Aumentada y videojuegos	39

3.3.1. Uso de marcadores	40
4. Especificación del proyecto	41
4.1. Especificación de requerimientos software	42
4.1.1. Requerimientos funcionales	43
4.1.2. Requerimientos no funcionales	43
4.2. Metodología de desarrollo	44
4.2.1. Metodología <i>Scrum</i>	46
4.2.2. Historias de usuario iniciales	48
4.3. Gestión de riesgos	49
4.4. Planificación	53
4.4.1. Descomposición del trabajo	53
4.4.2. Planificación temporal	55
4.4.3. Asignación de roles	56
4.5. Presupuesto	57
5. Desarrollo iterativo	61
5.1. Incremento 0. Estudio de alternativas	61
5.1.1. Estación digital de audio	62
5.1.2. Gráficos 3D	63
5.1.3. Gráficos 2D	64
5.1.4. Motor gráfico	65
5.2. Incremento 1. Puesta en marcha y marcadores de Realidad Aumentada	70
5.2.1. Proyecto en <i>Unity</i> y control de versiones	71
5.2.2. Entorno de Realidad Aumentada con <i>Vuforia</i>	73

5.2.3. Diseño de marcadores de Realidad Aumentada	75
5.2.4. Resultados	77
5.3. Incremento 2. Sistema de turnos y tablero	78
5.3.1. Sistema de turnos	78
5.3.2. Tablero de juego	82
5.3.3. Resultados	84
5.4. Incremento 3. Interacción y fichas de juego	85
5.4.1. Tipos de fichas de juego	86
5.4.2. Estructura de datos del tablero	89
5.4.3. Seguimiento de marcadores y posición en el tablero	91
5.4.4. Acciones de fichas	96
5.4.5. Ataque entre fichas	98
5.4.6. Resolver el turno	99
5.4.7. Recursos del jugador	102
5.4.8. Preparación del tablero	104
5.4.9. Resultados	104
5.5. Incremento 4. Recursos audiovisuales e interfaz gráfica	105
5.5.1. Indicadores y efectos visuales en la partida	106
5.5.2. Interfaz gráfica en la partida	108
5.5.3. Menú principal y opciones	111
5.5.4. Efectos sonoros y música	114
5.5.5. Mezclador de sonido	115
5.5.6. Resultados	116
5.6. Incremento 5. Despliegue del prototipo	117

5.6.1. Navegación entre escenas	117
5.6.2. Generación del ejecutable	118
5.6.3. Resultados	120
5.7. Incremento 6. Pruebas y mantenimiento	120
5.7.1. Feedback de los jugadores	120
5.7.2. Etapa de pruebas. Problemas encontrados	122
5.7.3. Solución de problemas y segunda versión	123
5.7.4. Resultados	125
6. Conclusiones	127
6.1. Trabajo futuro	127
6.2. Plan de mercado	128
6.3. Valoración personal	130
Bibliografía	x

Lista de figuras

1.1.	Evolución del diseño de controladores para videojuegos.	2
1.2.	Gráfico comparativo entre los conceptos Realidad Virtual, Realidad Aumentada, Realidad Mixta y Realidad Extendida	3
1.3.	Imagen promocional de <i>Half-Life: Alyx</i>	3
1.4.	Resultados de la encuesta “2023 Gaming Trends Report” de <i>Google</i>	4
2.1.	Imagen promocional de <i>Sid Meier's Civilization VI</i>	10
2.2.	Marcadores de <i>LandmARK</i> para Realidad Aumentada	14
2.3.	Concepto del movimiento de las fichas sobre el tablero	14
2.4.	Concepto de interacción mediante pantalla táctil	18
2.5.	Paquete de modelos 3D del paquete <i>KayKit - Medieval Hexagon Pack</i> .	20
2.6.	Imágenes promocionales de los paquetes de recursos visuales de <i>Kennedy</i> utilizados	21
2.7.	Efectos sonoros compuestos para <i>LandmARK</i> mediante <i>LMMS</i>	22
2.8.	Diseño de la interfaz gráfica y experiencia de usuario de <i>LandmARK</i> . .	23
3.1.	Sistemas domésticos de juego <i>Magnavox Odyssey</i> y <i>Atari 2600</i>	26
3.2.	Captura de pantalla de <i>E.T. the extraterrestrial</i>	27
3.3.	Sistema <i>Sega Dreamcast™</i> , edición temática de <i>Sonic The Hedgehog</i> .	28
3.4.	Imagen promocional de <i>Nintendo DS™</i>	30

3.5. Imagen promocional de <i>Steam Deck™</i> y estación de conexión	31
3.6. Imagen promocional de <i>Meta Quest 2</i>	32
3.7. Listado de juegos con más ingresos en <i>Google Play</i> en agosto de 2024	33
3.8. Captura de pantalla de <i>Dune II</i>	36
3.9. Captura de pantalla de <i>DOTA 2</i>	37
3.10. Captura de pantalla de <i>Bloons™ TD 6</i>	38
3.11. Imagen promocional de <i>Minecraft Earth</i>	39
4.1. Esquema de funcionamiento de la metodología ágil <i>Scrum</i>	47
4.2. Diagrama de descomposición del trabajo, con varios niveles jerárquicos de especialización.	53
4.3. Diagrama de Gantt del proyecto	55
4.4. Estimación de factura de consumo energético.	58
5.1. Captura de pantalla de la aplicación <i>LMMS</i>	63
5.2. Captura de pantalla de la aplicación <i>Blender</i>	64
5.3. Captura de pantalla del motor gráfico <i>Godot</i>	67
5.4. Sistema <i>Blueprints</i> de <i>Unreal Engine</i> aplicado en una escena	68
5.5. Herramientas software utilizadas durante el desarrollo del proyecto	70
5.6. Ventana de creación de un nuevo proyecto en <i>Unity Hub</i>	71
5.7. Listado de paquetes del proyecto de <i>Unity</i> utilizados	72
5.8. Jerarquía de escena en <i>Unity</i> con los objetos de <i>Vuforia</i> necesarios	74
5.9. Captura de pantalla del editor <i>Unity</i> ejecutando el simulador de <i>Vuforia</i>	75
5.10. Diseños de marcador evaluados durante el desarrollo	76
5.11. Marcadores de <i>LandmARK</i> para Realidad Aumentada	77

5.12. Capturas de pantalla de la aplicación tras el primer incremento	78
5.13. Secuencia básica de movimiento entre turnos alternados	82
5.14. Jerarquía de escena para el tablero de juego	83
5.15. Posicionamiento del tablero de juego a partir del marcador principal . .	85
5.16. Diagrama UML de las clases para las fichas de juego	86
5.17. Creación de una instancia de objeto <i>Token</i> como <i>ScriptableObject</i>	87
5.18. Inspector de parámetros de una instancia de un objeto <i>Token</i>	87
5.19. Jerarquía de un <i>prefab</i> para las fichas de juego	89
5.20. Ejemplo de contenido de las estructuras de datos del tablero	90
5.21. Diagrama de clases centrado en la clase <i>TokenMoveRegister</i>	91
5.22. Marcadores <i>ImageTarget</i> de la escena y colecciones organizadas en un objeto <i>TokenTargetsManager</i>	93
5.23. Sombreador para el indicador de movimientos de fichas	94
5.24. Diagrama de clases para el movimiento de marcadores y fichas	94
5.25. Diagrama de secuencia para el movimiento de marcadores y fichas . .	95
5.26. Jerarquía y resultado de la interfaz gráfica para las acciones de fichas .	96
5.27. Diagrama de clases para la gestión de acciones de fichas	97
5.28. Diagrama de secuencia para la realización de acciones de las fichas .	98
5.29. Diagrama de clases del control de movimientos, acciones y ataques ac- tualizado	99
5.30. Diagrama de clases actualizado del movimiento de fichas y estado del tablero	101
5.31. Inspector de la escena sobre la instancia de la clase <i>BoardSetup</i>	104
5.32. Capturas de pantalla del prototipo en funcionamiento tras el tercer in- cremento	105

5.33. Indicadores visuales para acciones durante la partida	107
5.34. División de una imagen en nueve segmentos desde <i>Unity</i>	108
5.35. Panel de acciones decorado con texturas e iconos	109
5.36. Interfaz gráfica de usuario visible durante la partida	110
5.37. Jerarquía de escena para el menú principal del prototipo	111
5.38. Animación de inicio del menú principal del prototipo	112
5.39. Jerarquía de escena para la ventana de opciones	113
5.40. Mezclador de sonido con los canales y filtros definidos	115
5.41. Ajustes de compilación del proyecto <i>Unity</i>	118
5.42. Ajustes gráficos del proyecto <i>Unity</i>	119
5.43. Textos indicativos del cambio de turno durante la partida	124

Capítulo 1

Introducción

¿Qué impulsa a una persona a jugar un videojuego? Para la mayoría de jugadores, la justificación queda dada mediante la búsqueda de **entretenimiento**, como un método más de ocio. Otros individuos buscan un aspecto más **educativo** en los videojuegos, tratando de alcanzar metas o conocimientos concretos, generalmente dedicando su tiempo a lo que hoy en día conocemos como juegos serios o *Serious Games*. Algo que comparten estos dos perfiles es el uso de propiedades intrínsecas del videojuego: un escenario o contexto, un objetivo definido y una serie de reglas, todo ello bajo un conjunto de mecánicas que motiven al jugador a alcanzar esos objetivos.

Desde el punto de vista del desarrollador, independientemente de su especialidad, las razones para involucrarse con los videojuegos también dan lugar a dos perfiles. En primer lugar, un videojuego es un producto software, comúnmente comercial o público, cuyo objetivo es alcanzar el máximo número de jugadores posible, maximizando así el **beneficio** percibido (ya sea económico o social). Simultáneamente, el mero interés personal sobre el medio incita a aquellos desarrolladores más interesados a continuar involucrados en la industria; el videojuego puede compararse a otras expresiones artísticas como la música o el cine, surgiendo debates sobre la calificación del **videojuego como un arte** [1] [2] [3] [4].

Independientemente de su clasificación artística, el videojuego ha conseguido tomar un hueco en la sociedad, constituyendo **uno de los pilares más robustos del entretenimiento** a nivel nacional [5] e internacional [6] [7].

1.1. Contexto

Entre las múltiples disciplinas que componen un videojuego, destacamos el uso de métodos de interacción persona-ordenador. Desde su nacimiento, los videojuegos han empleado periféricos comunes (teclado, ratón, monitor, etc.), e incluso han insertado el uso de controladores de juego, cuyo diseño continúa en evolución (véase la Figura 1.1). Diversos enfoques buscan llevar la interacción a un escenario más próximo al usuario: especialmente por parte de *Nintendo* y sus controladores para *Wii™* o *Nintendo Switch™*, que fomentan la realización de acciones más complejas que la pulsación de botones o teclas al introducir los **gestos como método de interacción**.

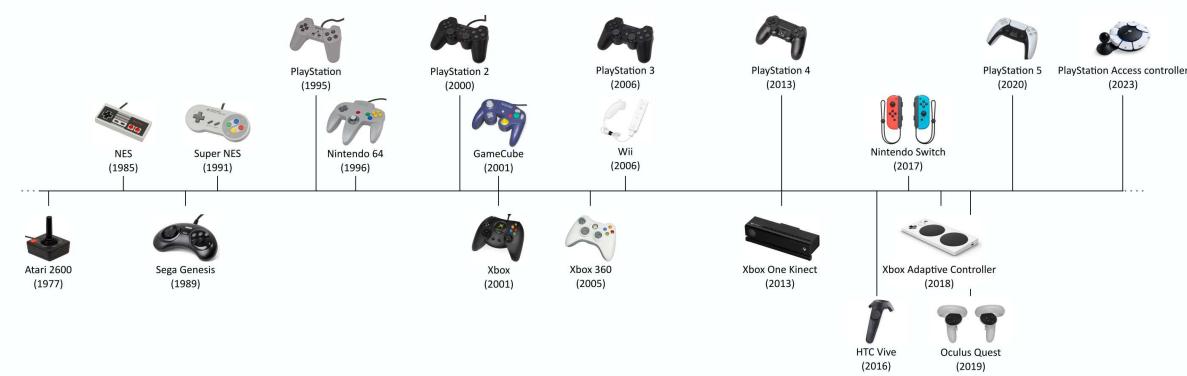


Figura 1.1: Evolución del diseño de controladores para videojuegos.

Indagando aún más en estos métodos próximos al usuario, pueden encajarse los conceptos de **Realidad Virtual (RV)**, **Realidad Aumentada (RA)** y **Realidad Mixta (RM)** como una forma de incorporar el propio entorno en que se encuentra el usuario a la interacción. Pese a ser muy similares, cada paradigma mencionado consta de una serie de características propias que lo diferencian con claridad (Figura 1.2): la RV sumerge al usuario dentro del entorno virtual, dotando al sistema del control total del mismo; la RA parte del entorno real del usuario, añadiendo ciertos elementos sobre este sin alterar la percepción del usuario; la RM se encuentra entre ambos extremos, alterando en cierta medida la percepción del entorno real con elementos virtuales.

Estos paradigmas innovadores se adentran cada vez más en aplicaciones a nivel general y bajo un amplio repertorio de disciplinas: desde soluciones muy especializadas para entornos industriales o sanitarios, hasta productos más estandarizados y adaptables como el gemelo digital. Los videojuegos no quedan al margen en su inclusión; múltiples empresas dedicadas al desarrollo de videojuegos han apostado por lanzar títulos completamente dedicados a estos paradigmas de interacción.

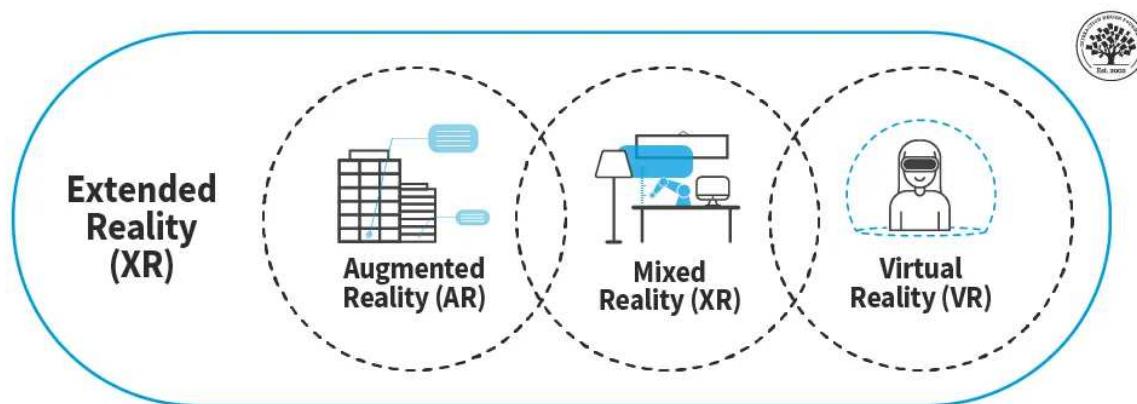


Figura 1.2: Gráfico comparativo entre los conceptos Realidad Virtual, Realidad Aumentada, Realidad Mixta y Realidad Extendida. Extraído de [Interaction Design Foundation](#)

La RV constituye el mayor eje para estos enfoques al habilitar una mayor inmersión para el usuario. Si bien aún continúa la exploración del concepto (y la estandarización del hardware necesario), títulos casuales como **Beat Saber** demuestran la posibilidad de mantener cierto nivel de confort durante la sesión de juego, a la par que entregas como **Half-Life: Alyx** ejemplifican el potencial de la tecnología y el paradigma de interacción. La Figura 1.3 muestra una captura promocional de este último título en la que pueden verse las manos del personaje, vinculadas a los controladores del hardware.



Figura 1.3: Imagen promocional de *Half-Life: Alyx*. Extraída de su [página web oficial](#).

Respecto a la RA, la pérdida del control total del entorno supone la necesidad de adaptar otras estrategias durante el diseño. Sin lugar a dudas, el mayor exponente de la RA aplicada a videojuegos se encuentra en el éxito mundial **Pokémon™ GO**, combinando en su jugabilidad la información geográfica como puntos de interés con

recompensas y mecánica principal de juego. Finalmente, la RM se encuentra muy por detrás en desarrollo al requerir cierta integridad en el entorno real. Como ejemplo, ***Mario Kart™: Live Circuit*** propone la construcción de circuitos de carreras de karts, estableciendo un camino entre diversos marcadores físicos, tras lo que será poblado con objetos virtuales durante la partida.

En la actualidad, cualquier aspecto relacionado con los videojuegos no puede ignorar su fuerte presencia en los dispositivos móviles, pues suponen la principal fuente de jugadores y beneficios. Por esto, muchos desarrolladores optan por dedicar sus esfuerzos a la creación de títulos ideados para estos dispositivos o en preparar una versión adaptada a ellos. Para la RV, el hardware aún no ha alcanzado suficiente madurez para proporcionar soluciones tan robustas como las mencionadas anteriormente, pero la RA obtiene el gran beneficio de disponer de una o varias cámaras integradas de alta calidad. Ante esto, ambos conceptos resultan interesantes en un ámbito móvil, como demuestran las estadísticas de la figura [Figura 1.4](#), en la cual los usuarios buscan experiencias innovadoras en estos paradigmas.

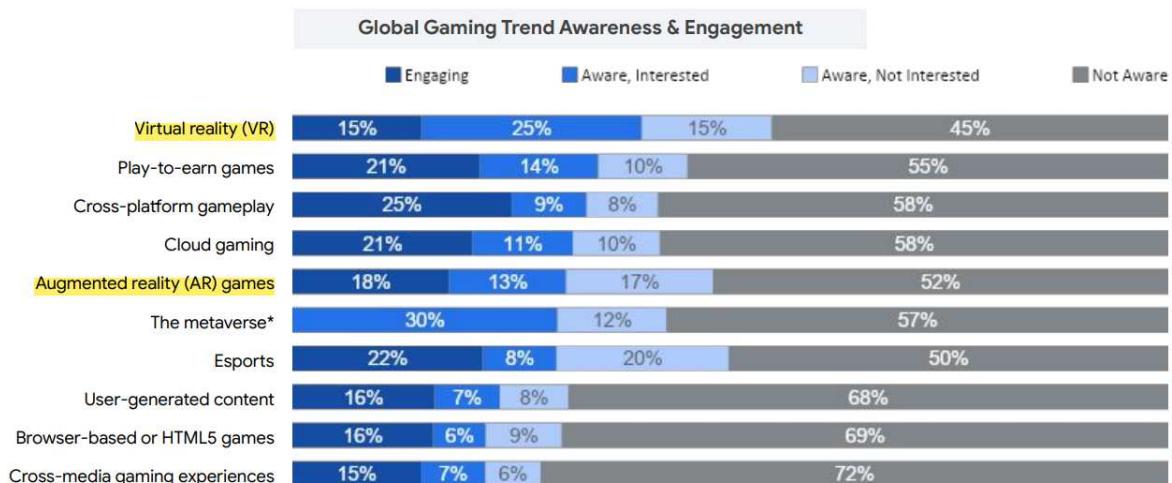


Figura 1.4: Resultados de la encuesta “2023 Gaming Trends Report” de *Google*. Extraída de [Google for Games](#).

1.2. Motivación

Siguiendo las tendencias revisadas, el desarrollo de un videojuego para dispositivos móviles empleando el concepto de la RA viene respaldado por un **interés suficiente por parte del público general**. Adicionalmente, al comparar el esfuerzo necesario en el desarrollo de un videojuego bajo estas características, **el uso de la RA simplifica en gran medida el trabajo frente a la RV y la RM** al requerir una menor

conexión con el entorno físico. Esta simplificación viene también acompañada de unos requisitos mínimos en cuanto a hardware: basta con recurrir a la cámara y sensores del dispositivo para obtener la información necesaria del entorno que rodea al jugador.

Esto último también limita en gran medida el alcance del producto, pues será el dispositivo móvil el encargado de manejar todos los procesos involucrados no solo con la RA, también con el funcionamiento y gestión del juego. Por tanto, deben tenerse en cuenta las **limitaciones del hardware móvil**, considerando la capacidad de cómputo limitada y la minimización del uso de batería.

Con las bondades y limitaciones especificadas, este proyecto persigue el diseño y desarrollo de un prototipo de videojuego para dispositivos móviles que aproveche el uso de la RA en su jugabilidad. El prototipo queda, por tanto, dirigido a **explorar ese interés por parte de los usuarios** de dispositivos móviles en el uso de nuevos paradigmas de interacción, alcanzando un amplio abanico de perfiles de usuario potencialmente interesados en el producto final. Además, la **experiencia previa** en el desarrollo de aplicaciones interactivas con RA, móviles y videojuegos simplificará diversos aspectos del proyecto durante las fases de diseño y desarrollo, garantizando cierta calidad durante las mismas.

1.3. Objetivos

Teniendo en cuenta las necesidades y puntos a explotar del proyecto, el objetivo principal será el **diseño y desarrollo de un prototipo de videojuego con Realidad Aumentada**. El prototipo permitirá a los jugadores realizar el flujo básico de cualquier videojuego, desde que se inicia una partida hasta que finaliza bajo cualquier condición. Con el fin de alcanzar tal objetivo se plantean varios objetivos específicos, que velarán por la correcta obtención del objetivo general, listados a continuación:

- **O1:** Generación de un prototipo de videojuego de estrategia por turnos, interactivo en tiempo real para dispositivos móviles Android y compatible con las funcionalidades de *ARCore* para el uso de la Realidad Aumentada.
- **O2:** Evaluación continua de la jugabilidad, reglas de juego y experiencia de usuario, considerando continuamente el factor de la diversión (*fun factor*), buscando motivar a los jugadores a finalizar la partida y seguir con el juego.
- **O3:** Uso de recursos audiovisuales con un estilo constante a lo largo de la aplicación, siguiendo una temática bélica medieval, pero recurriendo a enfoques

estilizados o caricaturescos y geometría simple (*low-poly*) para los elementos visuales (imágenes, iconos, modelos 3D y efectos).

- **O4:** Recopilación de estadísticas durante la partida que permitan al jugador revisar el rendimiento y calidad de su estrategia.
- **O5:** Preparación de tutoriales y/o manuales de usuario que guíen a jugadores novatos sobre la jugabilidad y reglas de juego hasta obtener la capacidad de manejar la aplicación al completo independientemente.
- **O6:** Localización de los textos presentes en la aplicación para, al menos, los idiomas inglés y español, favoreciendo la facilidad de uso de la aplicación ante un mayor abanico de usuarios potenciales.

1.4. Normativa

Los contenidos del proyecto y la memoria redactada siguen los conceptos básicos establecidos en la normativa nacional UNE 157001:2014, utilizada como guía en la estructuración de los contenidos de este documento. La redacción del mismo queda delegada al único desarrollador del proyecto (autor del Trabajo Fin de Máster), y su revisión y validación parcial e incremental a los tutores asignados a la convocatoria del Trabajo Fin de Máster.

Como herramienta de escritura, se hace uso del servicio en línea *Overleaf* para la compilación de código *LaTex* con los contenidos del documento, generando un fichero en el formato común y portable *PDF* para su distribución, a partir de la plantilla para Trabajos Fin de Grado facilitada públicamente por Francisco Charte Ojeda¹.

1.5. Recursos del proyecto

Todos los recursos generados y recopilados para el desarrollo del proyecto se encuentran disponibles en [este repositorio remoto público de GitHub](https://github.com/fcharte/TFGLatexTemplate/tree/master). La estructura que mantiene este repositorio es la misma que utiliza el motor gráfico *Unity* para sus proyectos en disco, con los directorios *Assets* que mantiene los recursos generales, *Packages* para los metadatos de paquetes externos incluidos y *ProjectSettings* con la

¹<https://github.com/fcharte/TFGLatexTemplate/tree/master>

configuración y parámetros globales del proyecto, así como ficheros de *CSPROJ* dedicados a la compilación del código.

1.6. Estructuración de la memoria

1. **Introducción:** Detalla la visión global del proyecto, además de proveer un primer acercamiento y resumen genérico de la industria del videojuego y los paradigmas más complejos de interacción persona-ordenador. Especifica también los objetivos planteados para alcanzar las metas del proyecto.
2. **Antecedentes:** Revisión de la literatura, estado del arte y actualidad de la industria del videojuego y la RA, explorando las soluciones de mayor repercusión y la sinergia entre ambos campos.
3. **Especificación del proyecto:** Descripción del trabajo a realizar, desde un punto de vista técnico. Abarca la ingeniería de requisitos y metodología de desarrollo, así como estimaciones de coste y esfuerzo necesarios para el desarrollo.
4. **Desarrollo iterativo:** Especificación del desarrollo del trabajo, exponiendo las herramientas y recursos empleados. Queda dividido en los incrementos definidos, aportando la información detallada relevante para cada uno de ellos.
5. **Conclusiones:** Análisis del trabajo realizado tras su finalización y opiniones generales. Exposición de aspectos por resolver, limitaciones y trabajo futuro de cara a la continuidad del proyecto.

Capítulo 2

Documento de Diseño del Juego

Introducción

Un problema que afronta cualquier proyecto de desarrollo de un videojuego es su naturaleza multidisciplinar, juntando ideas, necesidades y objetivos. Esta amalgama de conceptos debe, por tanto, quedar redactada y ser accesible por todos los individuos relevantes en el proyecto, permitiendo añadir sus aportaciones, evaluar el alcance y contenido del juego o simplemente consultar aquellas decisiones tomadas ante cualquier duda durante el desarrollo. La experiencia previa en dar solución a esta necesidad origina el denominado **documento de diseño del juego** o **GDD** (*Game Design Document*) [8]

Este documento incorpora todas aquellas decisiones, definiciones y justificación de las distintas partes del juego, desde jugabilidad, hasta reglamento, incluyendo también temática, narrativa y estilo visual, entre otros aspectos. Desde su aparición, a pesar de no disponer de una estructura normalizada, el *GDD* ha sido definido en múltiples ocasiones por autores de la literatura del sector del videojuego. Un primer trabajo destacado en este aspecto propone aplicar ciertas mejoras al *GDD* a partir de los principios del análisis de requerimientos [9]; por otro lado, también se consideran propuestas para la redacción orientadas mucho más a desarrollos para dispositivos móviles [10].

Tomando esta última propuesta, queda redactado un *GDD* para este proyecto en este capítulo, donde se definirán todos los puntos necesarios durante el desarrollo. El objetivo principal de este documento es su utilidad como fuente principal de información al encontrar cualquier duda durante la producción, asegurando que el enfoque se mantenga sobre los objetivos e ideas propuestas.

Resumen del juego

LandmARK es un videojuego de estrategia por turnos entre dos jugadores para dispositivos móviles Android que aprovecha los conceptos de la Realidad Aumentada como método principal de interacción, estableciendo un tablero virtual sobre el que desplazar distintas fichas mediante marcadores físicos.

El objetivo principal consiste en **dominar el área controlada por el jugador rival**, destruyendo su punto de inicio para alcanzar la victoria. Toda la partida sucede en una misma escena, visualizada en el **entorno de Realidad Aumentada**, con elementos 3D decorativos y representativos del tipo de objeto y jugador al que pertenece.

Como referentes en juegos de estrategia similares a *LandmARK*, títulos pertenecientes a sagas como *Age of Empires*, *StarCraft* o *Warcraft* suponen los mayores exponentes de estrategia en tiempo real competitiva entre varios jugadores, mientras que, en cuanto a estrategia por turnos, la saga *Sid Meier's Civilization* ([Figura 2.1](#)) es una de las más relevantes en la industria por parte de los jugadores del nicho.

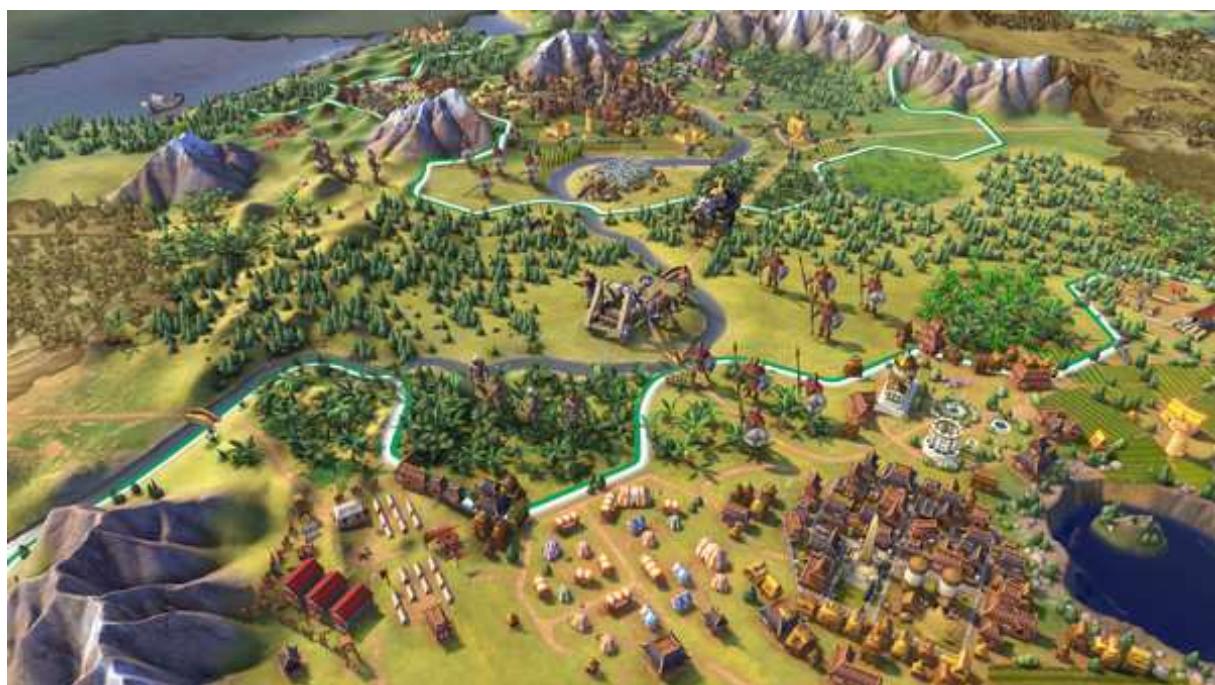


Figura 2.1: Imagen promocional de *Sid Meier's Civilization VI*. Extraída de [Steam](#).

LandmARK queda enfocado, por tanto, en la misma dirección que propone *Sid Meier's Civilization* con sus mecánicas, implementando un sistema de turnos (alternos en lugar de concurrentes), un despliegue de unidades o fichas simulando el ejército que controla cada jugador y un tablero o escenario en el cual se desarrolle la acción y estrategia durante la partida.

Simultáneamente, el uso de la Realidad Aumentada es un pilar central de *LandmARk*, un campo con mucha menos fuerza en cuanto a productos disponibles en el mercado. Actualmente, *Pokémon GO* constituye la principal figura del uso de RA en dispositivos móviles al simplificar la información del mundo real a únicamente la posición geográfica. Añadiendo el uso de marcadores físicos, no es un sector que presente suficiente fuerza en el mercado actual; hay que recurrir a trabajos de investigación que fomentan su aplicación con videojuegos, como *Stackable Music* [11] o *Enmity* [12].

LandmARk no contará con una historia argumental y estará centrado en el desarrollo de partidas cortas. Aunque se presentarán algunos personajes con identidad genérica y sin personalidad, no se incluirán conversaciones en el juego, relevando el cargo de aportar información de utilidad (tutoriales, guías, consejos, etc.) a textos en ventanas emergentes y controles de la interfaz gráfica. Si bien la falta de narrativa impone un gran punto negativo al título, permite dedicar los esfuerzos del desarrollo a mejorar **la experiencia de juego (*game feel*), que constituye el atractivo principal de *LandmARk*.**

Características

- **Género:** Estrategia por turnos.
- **Jugadores:** 2 jugadores, competitivo, en el mismo dispositivo local.
- **Plataformas:** Dispositivos móviles Android compatibles con *ARCore*.
- **Temática:** Época medieval: armamento de acero, uso de caballos, arcos y armas de asedio a distancia. Edificaciones de gran tamaño de piedra y madera.
- **Historia argumental:** Sin historia ni contexto narrativo.

Jugabilidad principal

Aprovechando el uso de un entorno de Realidad Virtual, mediante una serie de marcadores físicos, el jugador podrá posicionar el mundo virtual y un conjunto de fichas sobre un tablero de juego. Toda la acción del juego se llevará a cabo sobre este tablero haciendo uso de los marcadores mencionados; desplazar un marcador provocará el movimiento de la ficha en el mundo virtual.

LandmARk contará con 2 supertipos de ficha y un total de 6 tipos concretos. Como supertipos, habrá fichas de *Unidades*, representando personajes y tropas capaces de

moverse y atacar, junto a fichas de *Edificio* como una serie de construcciones estáticas que aportar un beneficio pasivo o activo al jugador.

Durante la partida, ambos jugadores obtendrán una serie de recursos desde varias fuentes, necesarios para la generación de nuevas fichas, añadiendo un factor adicional a su estrategia. La combinación de una buena planificación de recursos y movimiento de sus fichas ayudarán al jugador a conseguir la victoria.

Progresión

Además de no incorporar una historia argumental, misiones o tareas que guíen al jugador hacia unos objetivos concretos, *LandmARK* tampoco incluye ningún sistema de puntuación explícitamente. En su lugar, el propio factor estratégico supone el progreso durante la partida hasta la victoria, mediante la creación de nuevas fichas y la realización de acciones relevantes.

Idealmente, un jugador que dedique un mayor esfuerzo a generar y controlar más fichas conseguirá una ventaja sobre un rival que opte por un juego pasivo que no optimice el uso de sus recursos. Todas las decisiones tomadas deberán desembocar en el objetivo final intrínseco de *LandmARK*, la destrucción de un Edificio central del jugador rival (definido en las siguientes secciones) mediante ataques con fichas de Unidades.

Alcance

Partidas competitivas entre 2 jugadores reales en un mismo dispositivo, de 5 a 10 minutos. Cada jugador cuenta con un número máximo de fichas según su tipo, que se crearán y destruirán durante el transcurso de la partida. Dicha creación se basa en un sistema básico de economía mediante dos recursos representados por valores numéricos; contarán con un incremento pasivo en cada turno que podrá alterarse con varias mecánicas y se descontará al crear una nueva ficha, siendo también un factor limitante como condición previa a la creación de una ficha. Sobre la destrucción, algunas fichas tendrán la capacidad de realizar un “ataque” con el que enfrentarse a una ficha adversaria. El resultado de este enfrentamiento puede dar lugar a que alguna de las fichas (o ambas) sean destruidas y salgan del juego.

Público objetivo

Cualquier usuario de un dispositivo móvil Android con interés en los videojuegos y la tecnología de Realidad Aumentada. No se incluirán escenas sensibles ni de violencia explícita, pero los objetivos del juego supondrán un tono bélico que limitará el acceso a jugadores menores de edad.

Para conseguir que *LandmARk* resulte atractivo al mayor rango de jugadores, contará con una interfaz gráfica y con un estilo visual simple, empleando pocos controles de gran tamaño, iconos estilizados con la temática medieval y un conjunto de colores reducido, tomando el rojo y el azul como colores principales de los jugadores.

Con la finalidad de mantener un nivel suficiente de comodidad para el jugador, el dispositivo en que se ejecute *LandmARk* deberá contar con unas prestaciones suficientes que permitan alcanzar una tasa de, al menos, 30 fotogramas por segundo de forma estable. Cualquier dispositivo que no alcance este mínimo se considerará no compatible. Salvo en dispositivos más antiguos, la gran mayoría de smartphones recientes podrán incluso alcanzar tasas de 60 fotogramas por segundo, considerando los avances en la tecnología móvil. Para respaldar esto, *LandmARk* se mantendrá con una carga mínima en cuanto a elementos visuales y sistemas en funcionamiento.

Mecánicas del juego

Elementos de juego

LandmARk hará uso principalmente de una colección de marcadores similares a códigos QR ([Figura 2.2](#)), con un tamaño de 25x25 milímetros, identificables por la aplicación desde una distancia considerable y cómoda para su uso durante la partida (de 30 a 40 centímetros). Cada marcador será único en la colección y estará ligado a un tipo de ficha disponible en el juego. Esto servirá también para limitar el número de fichas simultáneas en la partida. También se dispondrá de un marcador de mayor tamaño, 50x50 milímetros, empleado para posicionar el entorno de Realidad Aumentada y el tablero de juego virtual.

Sobre el marcador de mayor tamaño quedará posicionado el mundo virtual, constituido por el tablero de juego con una serie de celdas distribuidas en una cuadrícula de 6x9 casillas. Todas las fichas se posicionan sobre una casilla y se desplazan según sus propiedades individuales; no pueden coexistir dos fichas en una misma casilla. El

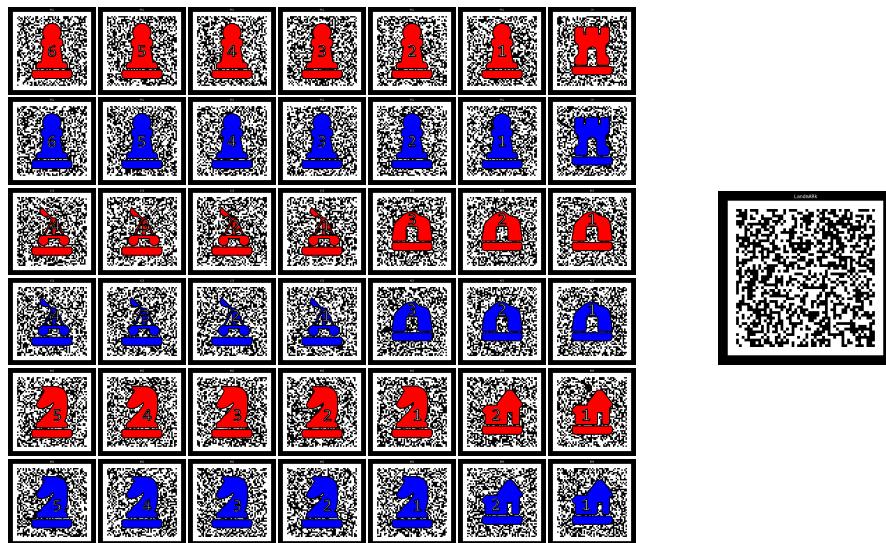


Figura 2.2: Marcadores de *LandmARK* para Realidad Aumentada

movimiento será realizado desplazando físicamente los marcadores hasta que aparezcan centrados en su casilla, mostrando los recursos visuales de la ficha en esa posición en la pantalla, similar a lo que muestra la Figura 2.3.

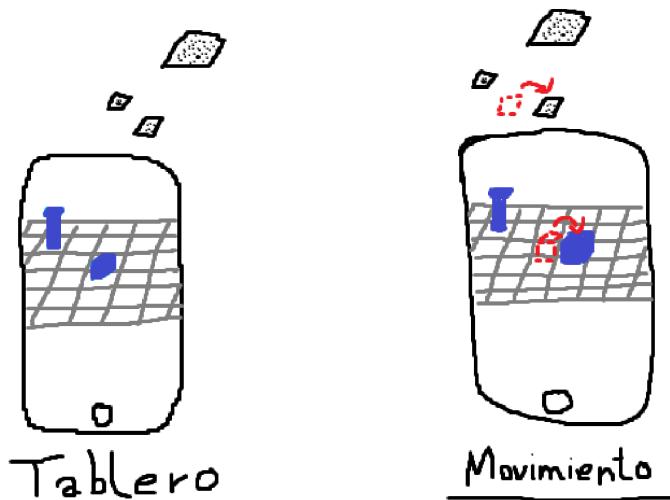


Figura 2.3: Concepto del movimiento de las fichas sobre el tablero

Cada jugador cuenta con un contador de dos recursos: madera y piedra. Ambos se obtendrán de forma pasiva y serán utilizados en la creación de nuevas fichas durante la partida. Los recursos disponibles de cada jugador serán visibles en todo momento en la interfaz de usuario, formando parte de la estrategia.

Coexistirán dos tipos de ficha. Primero, las fichas de **edificio** (Tabla 2.1) representan construcciones creadas por el ejército del jugador, y proporcionan ventajas pasivas y activas. Estas fichas no se desplazan en el tablero y deben construirse tras una serie de turnos.

	Ayuntamiento
	Resistencia 50
	Construcción -
	Función Punto inicial del jugador. Genera Milicia.
	Cuartel
	Resistencia 35
	Construcción 3 turnos. 2 Madera, 4 Piedra
	Función Genera Caballería y Asedio.
	Recolector
	Resistencia 15
	Construcción 1 turno. 3 Madera, 1 Piedra
	Función Genera recursos adicionales

Tabla. 2.1: Fichas de edificio de *LandmARk*

Por otro lado, las fichas de **unidad** (Tabla 2.2) pueden desplazarse por el tablero, saltando de una casilla a otra en cada turno. Cuentan con propiedades y funciones especiales, así como ventaja al enfrentarse a otra ficha como daño adicional cuando sucede un combate. El movimiento sobre el tablero es una propiedad única de cada ficha y no es posible su desplazamiento fuera del patrón definido.

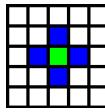
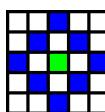
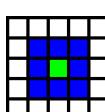
	Milicia
	Fuerza 3
	Resistencia 15
	Generación 2 turnos. 2 Madera, 2 Piedra
	Ventaja Asedio
	Función Construye Cuartel y Recolector
	Caballería
	Fuerza 4
	Resistencia 8
	Generación 3 turnos. 4 Madera, 1 Piedra
	Ventaja Milicia
	Función -
	Asedio
	Fuerza 5
	Resistencia 20
	Generación 3 turnos. 4 Madera, 4 Piedra
	Ventaja Edificios
	Función -
	Movimiento
	
	Movimiento
	
	Movimiento
	

Tabla. 2.2: Fichas de unidad de *LandmARk*

Reglas del juego

■ Inicio de partida

- Cada jugador comienza con un Ayuntamiento y una ficha de Milicia en posiciones opuestas del tablero.
- Cada jugador comienza con 1 de Madera y 1 de Piedra.
- El turno inicial será siempre del jugador azul.

■ Turno

- Solo podrá interactuarse con las fichas del jugador poseedor del turno.
- El jugador activo cede en cualquier momento su turno, mientras no haya realizado movimientos ilegales.
- El jugador activo puede ceder su turno sin realizar movimiento.
- En un turno puede hacerse un único movimiento sobre cada una de las fichas disponibles.
- Un turno no puede finalizar si queda pendiente posicionar una nueva unidad al generarse.

■ Movimiento

- El desplazamiento de unidades estará limitado por su patrón de movimiento.
- Los edificios interactúan en una de las cuatro direcciones cardinales.
- Al desplazarse y resolver el turno, la ficha se desplazará si la casilla destinataria está vacía.
- Al desplazarse y resolver el turno, se producirá un combate si la casilla destinataria contiene una ficha rival.
- Un combate supone un intercambio de daño entre las dos fichas involucradas, reduciendo los puntos de salud restantes de ambas.
- Algunas fichas disponen de una ventaja frente a otro tipo de ficha, otorgando un incremento del 50 % de daño adicional al aplicar el daño de combate.
- Un combate, tras aplicar el daño infligido por ambas fichas, puede resultar en cuatro situaciones:
 - **Ninguna es derrotada:** Las dos fichas reciben el daño y no se ven alteradas en otro aspecto.
 - **Defensora derrotada:** La ficha defensora sale del juego y la atacante ocupa su posición.

- **Atacante derrotada:** La ficha atacante sale del juego y la defensora mantiene su posición.
- **Ambas son derrotadas:** Las dos fichas salen del juego.

■ Recursos

- Los recursos serán añadidos al inicio de cada turno para el jugador que lo recibe.
- De forma fija, cada jugador recibe 1 unidad de Madera y 1 unidad de Piedra.
- Estos ingresos pasivos podrán ser incrementados con la posesión de ciertos tipos de ficha, que aporten un aumento de los recursos generados.

■ Fichas

- La construcción de edificios resulta en una nueva ficha de edificio situada en la casilla donde se inició el proceso.
- El contador de construcción será reducido al inicio del turno del jugador propietario.
- La generación de unidades resulta en una nueva ficha situada en una casilla adyacente de las cuatro direcciones cardinales.
- El contador de generación será reducido al inicio del turno del jugador propietario.
- Los recursos necesarios para construcción de edificios o generación de unidades serán descontados al seleccionar la acción y resolver el turno.
- Un edificio no será construido si la unidad no puede desplazarse a otra casilla.
- Una unidad no será generada si las casillas adyacentes al edificio en las cuatro direcciones cardinales están ocupadas.
- No habrá reembolso de recursos en caso de no poder generar una ficha, ya sea edificio o unidad.

■ Fin de partida

- La partida finalizará cuando un jugador destruya el Ayuntamiento de su rival, otorgándole la victoria.
- La partida finalizará en tablas cuando un jugador seleccione la opción desde el menú o se cierre la aplicación.

Dinámicas del juego

Metas y desafíos

El objetivo principal del juego se centra en alcanzar la victoria tras derrotar la ficha de Ayuntamiento del jugador rival, siguiendo una estrategia a partir de la información disponible del escenario. Como mayor obstáculo durante la partida, cada jugador afrontará el reto que propone la estrategia de su rival, así como las restricciones propuestas en las reglas del juego. De forma natural, en el género de estrategia, los jugadores buscarán tácticas que permitan optimizar el uso de sus recursos y minimicen el tiempo necesario hasta la victoria.

Controles

Además del manejo de marcadores físicos, la funcionalidad específica de cada ficha, así como la navegación y uso de los distintos elementos de la interfaz gráfica, será accesible mediante la pantalla táctil del dispositivo. Acciones como la selección de una ficha para obtener información adicional o marcar su objetivo quedarán realizadas por este método para facilitar el flujo de la partida. El resto de elementos de interfaz serán controles típicos de la interacción persona-ordenador, como botones, barras deslizantes, interruptores, campos de texto o numéricos, etc.

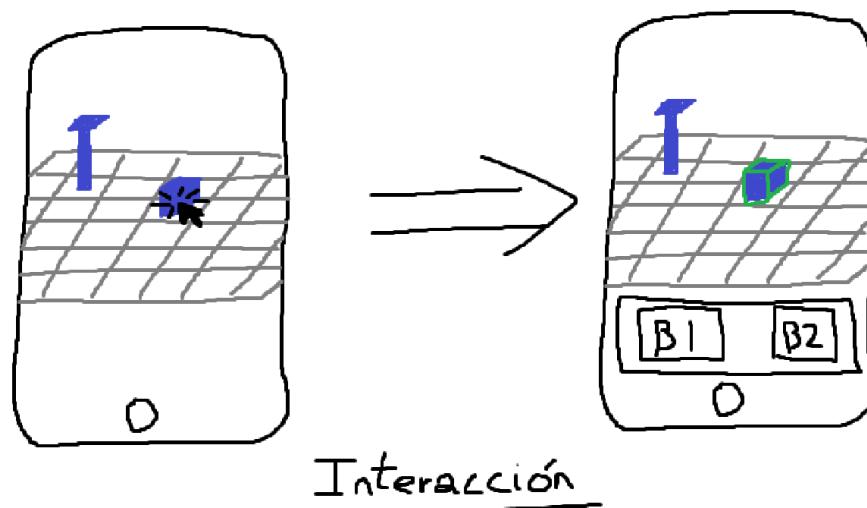


Figura 2.4: Concepto de interacción mediante pantalla táctil

Equilibrio del juego

Los puntos críticos a vigilar para asegurar un correcto equilibrio entre los dos jugadores son las **propiedades de las fichas** (puntos de fuerza, resistencia, movimiento y coste), así como la **generación de recursos**. En primer lugar, se evitarán establecer valores desorbitados o muy distantes al del resto en los parámetros de las fichas para prevenir el abuso de los mismos tipos durante toda la partida, lo que a su vez promueve el planteamiento de distintas estrategias más variadas.

La generación de recursos puede ser algo más delicada al tener que encontrar un punto intermedio entre no permitir construir un ejército masivo en pocos turnos, pero a la vez habilitar a los jugadores a generar las fichas que deseen sin ser un obstáculo a largo plazo. De manera estática, desde el inicio de la partida, **cada jugador recibirá una unidad de cada recurso** al inicio de su turno, **salvo el primer turno del primer jugador** para tratar de compensar la ventaja de comenzar con iniciativa de movimientos. Mediante la construcción de fichas de Recolector, cada jugador podrá incrementar la producción de un recurso en dos unidades adicionales; cada recolector puede generar un tipo de recurso de forma independiente e intercambiable sin coste adicional.

Estilo artístico

Modelos 3D

- “**KayKit - Medieval Hexagon Pack**” por **Kay Lousberg**: Obtenido desde Itch.io¹. La estética que aporta el uso de pocos polígonos para los modelos (estilo *low-poly*) aporta un factor caricaturesco al juego sin perder la temática medieval y militar. Las texturas, además, utilizan colores planos y sencillos, tomando varios colores primarios que permitirán distinguir las fichas de cada jugador.

Imágenes, iconos y fuentes

Los primeros recursos utilizados a continuación ([Figura 2.6](#)) cuentan con la licencia *Creative Commons CC0*, siendo así de dominio público y ofertando un uso libre y

¹<https://kaylousberg.itch.io/kaykit-medieval-hexagon>



Figura 2.5: Paquete de modelos 3D del paquete *KayKit - Medieval Hexagon Pack*

sin restricciones. Este es el principal motivo por el cual se recurre únicamente a los paquetes de recursos de *Kenney*, pues todos se ofertan bajo esa misma licencia. Es posible descargar los paquetes independientemente desde su página web² o desde su página de *Itch.io*³ como un conjunto de todos sus recursos mediante una compra única.

- “**Game Icons**” por **Kenney**: Colección de iconos generales, comunes en videojuegos. Cuentan con un estilo muy neutro, asegurando que aquello representado por cada ícono sea fácilmente reconocible en multitud de situaciones diversas.
- “**Board Game Icons**” por **Kenney**: Colección de iconos temáticos de juegos de mesa. Mantienen el mismo estilo neutro que el paquete anterior. Se hará uso únicamente de los iconos de piezas de ajedrez y construcciones para los marcadores de Realidad Aumentada.
- “**UI Pack (RPG Expansion)**” por **Kenney**: Recursos visuales para interfaz gráfica de usuario, con una temática más rústica bajo un estilo similar a los anteriores, con colores planos y llamativos.
- **Iconos personalizados**: Se generan dos iconos adicionales para los marcadores de Realidad Aumentada, propios de las fichas de Asedio y Recolector. Quedan diseñados tratando de seguir una estética similar al paquete *Board Game Icons* anterior.

²<https://kenney.nl>

³<https://kenney.itch.io/kenney-game-assets>

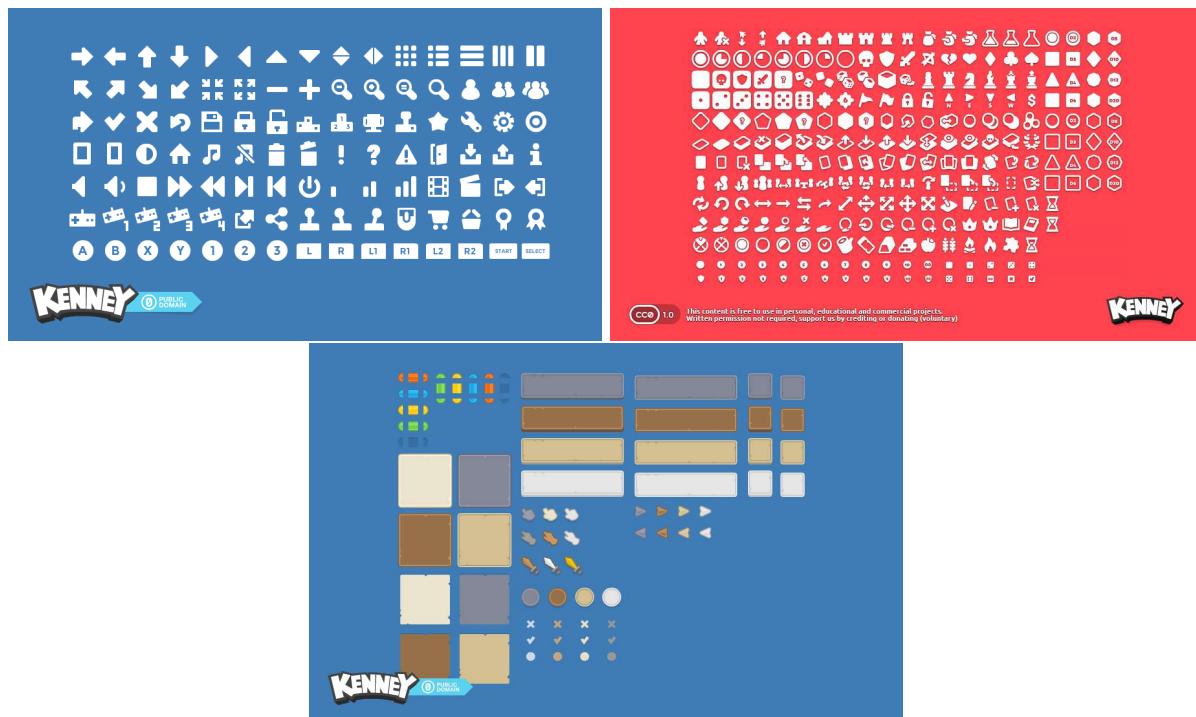


Figura 2.6: Imágenes promocionales de los paquetes de recursos visuales de Kenney utilizados

Efectos visuales

Todos los efectos visuales serán producidos como una combinación de los recursos visuales anteriores o con las herramientas que facilite el software de desarrollo (Unity, Blender, Inkscape, etc.).

- **Indicador de posición destacada:** emisor de partículas (cubos de un color prefijado) en un patrón circular hacia arriba, con un efecto sobre el tiempo que reduce lentamente el tamaño de cada partícula hasta desaparecer.
- **Marcador de movimiento:** Línea representada con un plano orientado hacia la cámara (*billboard*) con un sombreador personalizado que genera un efecto de movimiento transversal en la dirección del movimiento. Varía entre un color de validez (verde) y color de error (rojo).
- **Marcador de ataque:** Combinación de dos instancias de un modelo 3D como un cruce de espadas, empleando un sombreador de color plano y transparencia habilitada. Consta de un color previo a ser seleccionado (rojo) y otro una vez haya sido seleccionado (verde).
- **Marcador de ocupación:** Modelo 3D del símbolo de exclamación (!") con un sombreador que le proporciona un color plano rojo con cierta transparencia.

- **Marcador de daño:** Texto enriquecido posicionado en la escena 3D con una animación de aparición y desaparición alterando su posición en altura y tamaño.

Efectos de sonido

Efectos sonoros de elementos de interfaz gráfica obtenidos del paquete *Kenney - UI Audio*. El resto de efectos sonoros reproducidos durante el juego serán compuestos en el software *LMMS*, manteniendo un estilo similar a las pistas musicales, tomando como único instrumento el tambor y tonalidades graves.



Figura 2.7: Efectos sonoros compuestos para *LandmARk* mediante *LMMS*

Pistas musicales

Pistas de bucles musicales extraídas del paquete *Music Loops Mini Set 2* de *Andrey Sitkov*, adquirido en *Humble Bundle*⁴. La música de fondo se reproducirá una vez al entrar al menú principal, y se reproducirá uno de los bucles seleccionados cada cierto tiempo durante la partida.

Experiencia de juego

El prototipo de *LandmARk* contará con tres vistas principales. Comenzando por el **menú principal**, contará con la presentación al juego mediante una pequeña animación que use los mismos modelos 3D y muestre el título, así como tres botones principales para salir del juego, abrir el menú de opciones, y comenzar una partida. El **menú de opciones** incluirá barras deslizantes para establecer el volumen maestro y de efectos y música, junto a un selector de idioma, un botón con enlace a una encuesta externa y un botón para volver al menú desde la partida. Al contar un redu-

⁴<https://www.humblebundle.com/software/big-royaltyfree-music-2-software>

cido número de controles y vistas, el flujo entre ellas puede resumirse con un único esquema presentado en la figura [Figura 2.8](#).

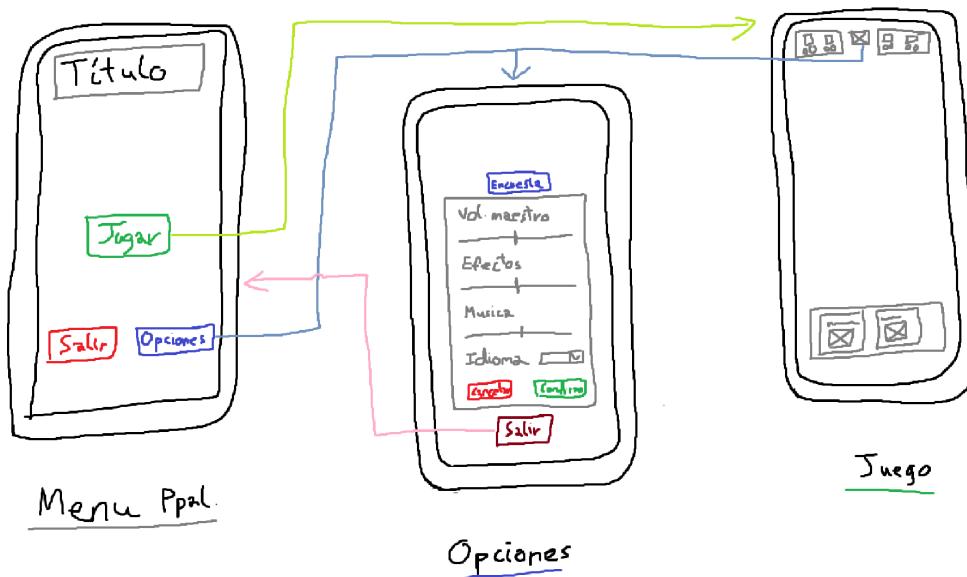


Figura 2.8: Diseño de la interfaz gráfica y experiencia de usuario de *LandmARK*

Limitaciones

Restricciones técnicas

LandmARK es un juego para **dispositivos móviles**, motivo por el cual debe establecerse un límite en la calidad visual de las escenas al tratar con un dispositivo ligero, portable y con **batería**. Esta calidad no solo considera el uso de texturas y técnicas de iluminación altamente demandantes, también se tienen en cuenta la cantidad de elementos visibles de forma simultánea y los sistemas que interconectan todos los objetos activos. Mantener un bajo nivel de carga asegurará también una **temperatura baja** durante el juego que no incomode al usuario.

Adicionalmente, la calidad de la **cámara trasera** del dispositivo influye directamente en la jugabilidad, pues compromete la visibilidad y detección de los marcadores de Realidad Aumentada, ante lo que se preparan marcadores con un tamaño suficiente y amplio número de características únicas tal que sean perceptibles por el sistema incluso en imágenes de menor calidad.

Restricciones de negocio

Contar con un equipo de una única persona encargada del desarrollo impone un proceso de iteración muy extenso en el tiempo, con incrementos lejanos temporalmente que permitan validar la progresión del proyecto. Además, cualquier videojuego requiere una cantidad inmensa de tiempo de desarrollo hasta completar todas sus fases. En general, *LandmARK*, como producto, sufrirá de un extenso periodo de desarrollo hasta que pudiese estar operable y público en cualquier sistema de distribución.

También queda considerada la nula experiencia previa del desarrollador en las labores de publicación y marketing para videojuegos. Esto se traduciría en errores de estimación a medio y largo plazo, así como a los ingresos potenciales que pueda generar el producto.

Capítulo 3

Antecedentes

El videojuego ha supuesto, sin duda alguna, una revolución en el entretenimiento y, cada vez más, en el desarrollo de software; gran parte de los conceptos que definen al videojuego quedan adaptadas a otras industrias o son aplicadas como mejora para la experiencia de los usuarios de un producto o servicio.

Desde los inicios de la industria hasta la actualidad, el videojuego ha pasado por diversas etapas en términos de popularidad, calidad y alcance, pero con un **crecimiento continuo año tras año**, llegando a tomar el puesto de sector del entretenimiento con mayor facturación anual [13] [14]. Tal magnitud impone un aumento en los dos ejes sobre la industria: cada año el número de títulos publicados crece exorbitadamente, y las producciones más serias cuentan con un alcance y presupuesto descomunales, llevados a cabo por una amplia plantilla de desarrolladores, productores, artistas, directores, etc.

Gracias al esfuerzo común de los participantes de la industria del videojuego, hoy en día contamos con un sinfín de recursos bibliográficos y audiovisuales que narran historias, proponen conceptos y, en general, aportan conocimiento tanto en el desarrollo de videojuegos como su aplicación en diferentes campos. Las siguientes secciones de este capítulo abarcarán diversos pilares de la industria del videojuego, con un mayor énfasis en aquellas partes relevantes con la temática de este proyecto.

3.1. Origen del videojuego

La aparición del concepto de videojuego resulta algo difusa e imprecisa, pero puede establecerse en la década del 1940 junto a las primeras máquinas de entreteni-

miento con componentes electrónicos y el origen de las máquinas recreativas [15] [16]. Esta época destaca también por **los primeros usos de la computación sobre juegos de mesa clásicos**, con la participación de grandes figuras como Alan Turing y Claude Shannon en busca de una máquina capaz de jugar competentemente al ajedrez; un avance de esta magnitud era, indudablemente, el futuro de la computación y la inteligencia artificial.

Poco a poco, el mercado americano y japonés vivieron la proliferación de los juegos recreativos, con salas cada vez más extensas llenas de máquinas y juegos únicos. En los primeros años de la década del 1970 encontramos nombres conocidos en la industria: *Nintendo*, *Sega*, *Atari* y *Magnavox* daban sus primeros pasos hacia el videojuego y las máquinas de juego domésticas con la aparición de *Pong* por parte de Al Alcorn y la gran presencia del lanzamiento de *Magnavox Odyssey™* (primera imagen de la [Figura 3.1](#)) en 1972 [17]. Años después, **en 1977, Atari desveló su sistema de juego doméstico con *Atari™ 2600*** (segunda imagen de la [Figura 3.1](#)) junto al título *Combat* acompañando su salida al mercado [18].



Figura 3.1: Sistemas domésticos de juego *Magnavox Odyssey* y *Atari 2600*. Imágenes obtenidas de [17] y [18] respectivamente.

En paralelo a múltiples disputas legales entre *Atari* y *Magnavox*, los videojuegos continuaron su desarrollo tanto en máquinas recreativas como en sistemas domésticos, hasta llegar al año 1980. En este mismo año, no solo *Atari* lanzaría su exitoso *Space Invaders™* para la *Atari 2600*, pues **Namco revolucionaría el mundo de las recreativas con la salida de *Pac-Man™***. Además, justo al año siguiente, *Nintendo* también formaría un consistente competidor en recreativas con su título *Donkey Kong™*. Entre unos años con grandes lanzamientos, resulta difícil pensar en el **gran desastre** que le esperaba al sector del videojuego próximamente.

El año 1983 marca uno de los grandes puntos de inflexión de la industria del videojuego americano como una de las mayores catástrofes que ha sufrido en su historia [19] [20]. El lanzamiento de **títulos clónicos y de pésima calidad**, la aparición de

cada vez más dispositivos de juego y la pérdida del control y regulación de los procesos de publicación inundaron el mercado hasta que, inevitablemente, colapsó hasta niveles críticos. El caso más conocido de este suceso se le otorga al lanzamiento del juego *E.T. the extraterrestrial*, inspirado en la película de 1982 *E.T., el extraterrestre* ([Figura 3.2](#)), famoso por su nefasta calidad, incluso odiado por los desarrolladores. *Atari* contaba con estanterías llenas de copias listas para su venta, pero que nadie estaba dispuesto a adquirir, lo que llevó a la empresa a deshacerse del exceso de producto enterrando cientos de cartuchos en el desierto de Nuevo México [21].



Figura 3.2: Captura de pantalla del juego *E.T. the extraterrestrial*. Extraída de [Retrogaming](#)

Aunque *Atari* no consiguió mantenerse a flote tras el desastre, *Nintendo* preparaba el lanzamiento de *Family Computer (Famicom)* en las tierras niponas en 1984, y sería en 1986 cuando recibe un lanzamiento mundial bajo el nombre de ***Nintendo Entertainment System*** o ***NES***, mientras que el matemático ruso **Alex Pajitnov** diseñaba un videojuego que nombraría **Tetris™**. Enseguida, la aparición de grandes títulos que hoy en día siguen presentes en la industria (*Super Mario Bros.™*, *The Legend of Zelda™*, *Final Fantasy™*, *Sonic the Hedgehog™* y más), volvieron a tomar el foco de atención y permitieron una segunda oportunidad al videojuego.

Con el tiempo, la industria continuó estandarizándose y alcanzando metas cada vez mayores, surgiendo nuevos competidores. Recién entrados en el siglo actual, en 2001 el mercado contaba con la consola *Nintendo 64™* por parte de *Nintendo*, *PlayStation™ 2* de la mano de *Sony*, y la recién llegada *Xbox™* de *Microsoft*. Estos años marcan, además, el final de la competencia de *Sega* por ofrecer su propio sistema de juego doméstico tras discontinuar la venta de *Dreamcast™* ([Figura 3.3](#)), dedicando su esfuerzo a la publicación de videojuegos en el resto de sistemas.



Figura 3.3: Sistema *Sega Dreamcast*™, edición temática de *Sonic The Hedgehog*. Extraída de [ConsoleVariations](#)

Sin embargo, la retirada de *Sega* no fue algo repentino: tras la crisis del 1983, inició un rifirrafe contra *Nintendo* en busca de la posición dominante del mercado de consolas; este periodo suele clasificarse como **la primera guerra de consolas** [22] [20]. Previo a *Dreamcast*™, *Sega* trató de competir en el mercado contra la titánica *NES*™ de *Nintendo* mediante su consola *Mega Drive*™, logrando cierta ventaja en Estados Unidos pero con un impacto mucho menor en Japón. De esta competición, como conocemos hoy día, *Nintendo* acabó superando a *Sega* a largo plazo, estableciéndose como un pilar central del videojuego.

El futuro tras la primera guerra de consolas marcaría el paso a la tercera dimensión, con hardware mucho más avanzado y con capacidades gráficas excepcionales para la época. Desde entonces hasta la actualidad, la industria ha seguido un progreso relativamente lineal: con la salida de nuevas generaciones de consolas, la calidad visual, artística y la profundidad de mecánicas se intensifican gradualmente, muchas veces en busca del fotorrealismo.

Un punto de inflexión destacable del periodo más reciente de la industria del videojuego es **la aparición y extensión del sector independiente (indie)**: títulos desarrollados por estudios pequeños y con escasos recursos [23]. Lentamente, la industria se ha masificado, adoptado estrategias de menor riesgo a largo plazo que garanticen unos beneficios estables durante el máximo tiempo posible.

Entre esas estrategias, la reutilización de mecánicas y diseños simples prolifera tanto en los estudios de desarrollo como en los usuarios finales, y desmotivan la exploración de nuevas ideas y conceptos arriesgados. Aquí destacan los títulos *indie*: ya que no pueden competir en calidad y cantidad de contenido dada su limitación de recursos, deben tomar el riesgo de ofrecer una experiencia única.

Son muchos los videojuegos *indie* que han supuesto una revolución en la industria, retomando géneros abandonados o con una base de jugadores muy escasa, e incluso como génesis de nuevos géneros. El mejor ejemplo de esto lo encontramos en los géneros *rogue-like/rogue-lite* y *metroidvania*¹, popularizados principalmente gracias a los incontables títulos independientes lanzados año tras año.

The Binding of Isaac™: Rebirth, *Hades* o *Balatro* dominan las listas recientes de popularidad y jugadores para el género *rogue-like* de Steam², todos ellos siendo producciones independientes desarrolladas por equipos muy limitados. Como otros ejemplos, etiquetados con el género *metroidvania*, el éxito del título español *Blasphemous* y su secuela *Blasphemous 2*, o el excelente *Hollow Knight* también forman parte de los más jugados en Steam³.

3.1.1. Nuevos sistemas de juego

Otro aspecto que destaca en el videojuego actual es la presencia de un amplio abanico de plataformas de juego, no en forma de consolas alternativas por parte de múltiples empresas, sino como **dispositivos que ofrecen experiencias únicas** y distinguibles entre sí. La normalización de los dispositivos móviles y *smartphones* no tardó en recibir un gran interés por parte de los desarrolladores de videojuegos, surgiendo títulos con enfoques mucho más casuales adaptados a las pequeñas pantallas y dispositivos limitados en hardware y por batería.

Junto al *smartphone*, las nuevas versiones de consolas portátiles continuaron con un hueco en el mercado. Tras el éxito de *GameBoy™*, *Nintendo* apostaría por otro sistema portátil en 2004, *Nintendo DS™* (Figura 3.4), que volvería a convertirse en un referente de la industria. Simultáneamente, *Sony* también se lanzaría al mercado portátil con *PlayStation Portable* o *PSP™*, marcando fuertemente su presencia en el mercado junto a *Nintendo*.

¹La etiqueta “metroidvania” surge de una mezcla de los títulos *Castlevania™* y *Metroid™*, juegos casi idénticos y originarios del género.

²<https://steamdb.info/charts/?tagid=1716>

³<https://steamdb.info/charts/?tagid=1628>



Figura 3.4: Imagen promocional de *Nintendo DS*™. Extraída de la [página web oficial de Nintendo](#)

Ambas consolas recibieron, con el tiempo, nuevas versiones en forma de hardware mejorado y adaptado a las novedades: *Nintendo* continuó con un enfoque idéntico mediante *3DS*™ y *2DS*™, que añadían un efecto de profundidad mediante estereoscopía⁴ a las imágenes generadas; por otro lado, *Sony* apostaría por una doble pantalla táctil en *PlayStation Vita*™, que no consiguió la misma fuerza que su predecesora.

En la actualidad, estas consolas portátiles no continúan en la industria, pues han dado paso a sistemas algo distintos. *Nintendo*, como referente en innovación del videojuego, apostó por un **sistema híbrido** con *Nintendo Switch*™, ofreciendo a los jugadores una consola de menor tamaño y peso, pero con suficiente capacidad hardware para funcionar a mayores resoluciones una vez conectada a una base incluida en el paquete. Pese a su salida en 2017, *Nintendo Switch*™ sigue siendo el sistema principal de *Nintendo* en 2024, casi duplicando el tiempo de vida común de cada generación de consolas.

A la par que *Nintendo Switch*™ continuaba su extensión, *Valve*, principal desarrollador y distribuidor de videojuegos en ordenador gracias a su plataforma *Steam*, propuso un sistema similar en 2022, *Steam Deck*™ (Figura 3.5), con un enfoque mucho más orientado al factor portátil, pero con posibilidad de conexión a un entorno de escritorio al uso. Permitiendo a los jugadores acceder a casi todos los títulos de *Steam* gracias a *Proton*⁵, esta consola ha encontrado su espacio en el mercado y, poco a poco, son más los competidores que buscan acaparar parte de ese espacio con sistemas similares (*Asus ROG Ally*™ o *MSI Claw*™).

⁴La estereoscopía consiste en colocar una imagen ajustada a cada ojo, con una modificación mínima para que, al ser alineadas, se obtenga una sensación de profundidad.

⁵*Steam Deck* emplea un sistema operativo basado en Linux, mientras que la mayoría de títulos solo son compatibles con Windows. *Proton* es una herramienta de compatibilidad que traduce los ejecutables de Windows a sistemas Linux.



Figura 3.5: Imagen promocional de *Steam Deck*™ y estación de conexión. Extraída de su [página web oficial](#)

Tomando otro enfoque en cuanto a jugabilidad e inmersión, en 2012 aparece una campaña de micromecenazgo⁶ respaldada por profesionales de la industria del videojuego [24] presentando un dispositivo de **Realidad Virtual para el consumidor general**, *Oculus Rift*. Si bien desde la década del 1990 se daban los primeros pasos en el concepto de Realidad Virtual, *Oculus* adentraba el concepto al mercado general del videojuego, como una plataforma dedicada a nuevas experiencias mucho más inmersivas. Poco después, *Sony* adoptaría la misma idea con *PlayStation VR*.

Siendo una tecnología novedosa y fuera de lo común para la mayoría de jugadores, no se acopló con tanta fuerza, pero resultaba excepcionalmente prometedora y continuaron los avances tecnológicos en busca de una mejor comodidad durante su uso y software dedicado (videojuegos y aplicaciones de entretenimiento en general). A los pocos años, Valve seguiría explorando sus opciones en la fabricación de hardware junto a HTC en el mercado de la Realidad Virtual, lanzando *Valve Index* después de que su sistema software *ValveVR* se estableciese.

Algo en lo que pecan estos dispositivos es la **necesidad de mantener una conexión al dispositivo**, pues la tecnología requiere una gran capacidad hardware para mantener una experiencia de juego fluida: al hacer uso de estereoscopia para añadir la sensación de profundidad, deben generarse dos fotogramas simultáneamente, a la par que debe alcanzarse una tasa elevada de fotogramas por segundo para evitar problemas comunes de percepción, estabilidad y náuseas.

⁶En el micromecenazgo, los usuarios aportan económicamente una cantidad determinada, comúnmente empleada en proyectos pequeños y con recursos limitados, ofreciendo recompensas como contenido adicional, acceso anticipado, etc.

No tardaron en salir al mercado nuevas versiones de los dispositivos que solventan este problema, como *PlayStation Gear VR* u *Oculus Go*, pero es fácil atribuir la popularización del concepto a *Oculus* desde el lanzamiento de *Oculus Quest*. Desde esta iteración, *Meta*, propietaria de *Oculus*, dispone de dos revisiones adicionales, *Meta Quest™ 2* (Figura 3.6) y la más reciente *Meta Quest™ 3*, siendo uno de los mayores representantes de la Realidad Virtual; ***Meta Quest™ 2 es, con diferencia, el dispositivo más utilizado en videojuegos*** de ordenador en la plataforma Steam⁷.



Figura 3.6: Imagen promocional de *Meta Quest™ 2*. Extraída de su [página web oficial](#).

3.1.2. Smartphones y videojuegos móviles

Entre las muchas plataformas que han adoptado al videojuego, hoy en día el sector móvil toma el papel principal como mayor productora de ingresos y, por tanto, principal fuente de innovación y mejora.

La portabilidad y utilidad del *smartphone* ha supuesto una revolución en la población mundial, adaptando nuestro día a día sobre tareas cada vez más centralizadas en estos dispositivos. Naturalmente, el ocio también ha acabado formando parte de su utilidad, dotando a las personas de nuevas experiencias mediante un sistema portable, de menor escala, y con limitaciones considerables.

Pese a nacer con los mismos objetivos que los videojuegos clásicos, las plataformas móviles han tornado hacia soluciones mucho más **comerciales y con enfoques a muy corto plazo**, que debutan y caen en el olvido en cuestión de semanas o meses [25]. Esta ideología de desarrollo ha sido combinada a un modelo comercial catalogado como “oscuro” por muchos usuarios y desarrolladores: la facilidad de acceso a los dispositivos móviles ha originado la expansión de títulos gratuitos, que implementan mecánicas estrictamente estudiadas para la generación de beneficios.

⁷<https://store.steampowered.com/hwsurvey/Steam-Hardware-Software-Survey-Welcome-to-Steam>

En la [Figura 3.7](#) se muestra un listado de los juegos con mayores ingresos lanzados en *Google Play* para *Android*, todos ellos gratuitos y centrados en micropagos orientados a ciertas ventajas en la partida o elementos cosméticos. Muchas de estas mecánicas surgen con el único objetivo de producir beneficio económico, dejando poco a poco a un lado el bienestar de los jugadores y el factor diversión.

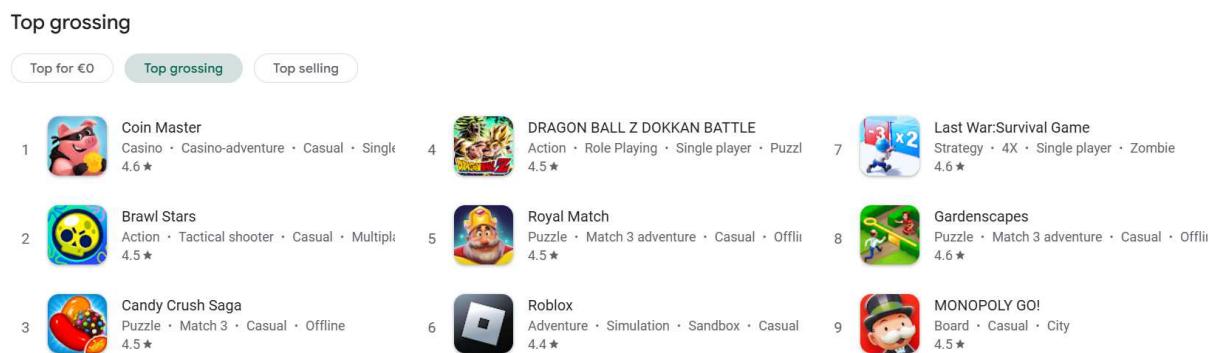


Figura 3.7: Listado de juegos con más ingresos en *Google Play* en agosto de 2024. Extraído de [Google Play Store](#)

El exponente actual más arraigado en la industria lo componen las cajas de botín o *loot boxes*, objetos que otorgan recompensas aleatorias, generalmente cosméticas, adquiridas con dinero real. Este concepto resulta inmediatamente comparable a **estrategias y técnicas inherentes de los juegos de apuestas** o *gambling*, con todas sus connotaciones negativas y perjudiciales [26].

El caso de los videojuegos móviles es mucho más preocupante: la accesibilidad de los dispositivos móviles en menores de edad crece anualmente, exponiendo a las nuevas generaciones mucho antes a la tecnología. Lógicamente, combinar estrategias lucrativas e inspiradas en el juego y apuestas con productos fácilmente accesibles por menores marca una necesidad de revisión sobre las regulaciones del videojuego [27]. Afortunadamente, este es un problema conocido y tratado individualmente o en conjunto por multitud de países; en Europa, encontramos soluciones estrictas como restricción de edad o prohibición, o alternativas menos impactantes como la necesidad de indicar el valor monetario y todos los detalles relativos a la probabilidad de obtención de los premios [28] [29].

Aun con tales perjuicios ligados a los videojuegos móviles, se mantienen con una amplia oferta de ventajas para los jugadores, más allá del simple pasatiempo: la **salud física, mental y social** son las principales beneficiadas en los jugadores de todas las edades [30] [31] [32]. Aplicado a individuos, jóvenes y entornos escolares, también resultan de utilidad frente a los problemas comunes de inactividad física y desarrollo cognitivo; por supuesto, **siguen existiendo riesgos** asociados que deben considerarse continuamente [31] [33].

3.1.3. Desarrollo de videojuegos

Cambiando el enfoque de este estudio sobre la industria del videojuego, el punto de vista de los desarrolladores también merece una dedicación especial. Cualquier videojuego es un producto masivo, con numerosas vertientes del conocimiento humano en un punto común hasta alcanzar una armonía que se traduce en su jugabilidad.

Por esto, el desarrollo de videojuegos impone la necesidad de un amplio equipo de trabajo multidisciplinar, lo que, a su vez, induce el requisito de figuras intermedias capaces de gestionar, organizar y dirigir a todos los miembros. Utilizando el ejemplo de “*The Door Problem*” por Liz England [34], cada rol asignado responde de maneras muy variadas y especializadas a cuestiones tan simples como la utilidad y funcionamiento de una puerta.

Aunque Liz England expone varios roles, el funcionamiento interno de cada organización o grupo varía enormemente según múltiples factores, que en ocasiones supondrá la ausencia de algún rol o la necesidad de otros puestos especializados. Pese a no seguir una estandarización o convenio común, muchas figuras aparecen de forma constante a lo largo de la industria, independientemente del ámbito social de los desarrolladores [35] [36] [37]: **directores y diseñadores creativos, artísticos y de sonido** plantean las ideas apropiadas para el juego y su contenido; los **programadores, artistas y productores** implementan todas esas ideas hasta formar un producto funcional, interactivo, y atractivo; el **equipo de control de calidad o QA (Quality Assurance)** garantizan que el producto alcance los objetivos propuestos y no incluya errores de funcionamiento.

Una cualidad que comparten la mayoría de creadores, diseñadores y desarrolladores es su **pasión por el videojuego y la posibilidad de ofrecer al público una experiencia única** y sobrecogedora. En paralelo al desarrollo de este proyecto, la industria se encuentra en una de sus peores etapas para los creadores: algo evidente cuando la producción se masifica y la calidad aumenta es el incremento en los costes, especialmente tras la situación de pandemia global COVID-19.

En los últimos años, miles de personas han perdido sus puestos de trabajo en la industria a raíz del excesivo crecimiento de las organizaciones y los aumentos en cantidad y calidad del contenido de los nuevos títulos. 2023 supuso un fuerte impacto con más de 6.000 despidos en grandes empresas de todo el mundo [38], pero **escasos meses tras el comienzo de 2024, esa cifra ha sido superada** con un total superior a los 8.000 despidos [39] [40].

Si bien no podemos conocer a ciencia cierta lo que deparará el futuro a la industria, todo apunta a que las rondas de despidos masivos continuarán hasta que se alcance cierta estabilidad. Algo que sí es seguro, independientemente de la situación, es que esa pasión por el medio continuará motivando a más personas y desarrolladores globalmente e inspirando el nacimiento de nuevos títulos.

3.2. Géneros de videojuego: Estrategia

Una propiedad común a cualquier obra artística es la posibilidad de distinguir agrupaciones que asemejen varios productos en un género, como si de una etiqueta se tratase. Los objetivos de estas etiquetas tratan de dar a entender al público qué buscan expresar antes de que trate directamente con el producto o la obra.

Para un videojuego, su género representa **qué puede esperar el jugador sobre la jugabilidad y mecánicas que incluye**, e incluso cuestiones más detalladas como una duración aproximada del juego o las partidas, o la profundidad de la interacción con el mundo y los personajes [41] [42].

3.2.1. Historia de los videojuegos de estrategia

El origen de los videojuegos de estrategias, como es de esperar, proviene de juegos clásicos de tablero (Ajedrez o Go) y *wargames*⁸ trasladados al formato digital. La primera entrega dedicada completamente en videoconsolas puede asociarse a *Invasion* para *Magnavox Odyssey* en 1972 [43], replicando al conocido juego de mesa *Risk™*, al cual le sucedieron otros títulos similares en concepto, todo bajo la misma ideología que define al género de estrategia.

En la década posterior, surgieron algunas entregas que se atrevieron a modificar las mecánicas comunes del género, dejando de lado la división en turnos para las acciones de los jugadores y propiciando una jugabilidad **en tiempo real** simultánea. Este nuevo enfoque acabó destacando entre el público con la salida de *Dune II* en 1992 (Figura 3.8), y acabó por ser uno de los puntos fuertes de la industria al cabo de un tiempo, gracias a los títulos de *Blizzard Entertainment StarCraft™* y *Warcraft™ III: Reign of Chaos* (1998 y 2002 respectivamente).

⁸Los juegos de guerra o *wargames* son juegos de mesa en que los jugadores compiten entre sí controlando ejércitos representados con figuras en miniatura, simulando un conflicto bélico.



Figura 3.8: Captura de pantalla del juego *Dune II*. Extraída de [Retrogames](#)

3.2.2. Otros subgéneros

Desde los orígenes hasta la actualidad del videojuego de estrategia, han surgido incontables subgéneros que heredan los conceptos de planteados en sus raíces. Cada vez resulta más complicado encontrar títulos etiquetados simplemente como estrategia, a favor de estos subgéneros que especifican aún más su jugabilidad, pues han sido estandarizados en el público general y recibidos positivamente por los jugadores.

- **4X: eXplore, eXpand, eXploit, eXterminate**

La vertiente más conocida dentro de los videojuegos de estrategia, popularizada casi desde el mismo nacimiento del género, se otorga al subgénero 4X, simplificando los cuatro conceptos clave de **exploración, expansión, explotación y exterminación**. Su origen se remonta a 1993, donde el periodista Alam Emrich introdujo el concepto en su análisis de *Masters of Orion* para la revista *Computer Gaming World* [44].

En las entregas etiquetadas como 4X, los jugadores toman el cargo de un imperio que evolucionará durante la partida, resaltando avances militares, tecnológicos, culturales o científicos, entre otros, que habiliten diversas rutas hacia la victoria. Con este concepto, la condición única de victoria por dominación militar convive con otras alternativas: suele ser posible conseguir una supremacía mediante avances tecnológicos sobrecededores o el uso de estrategias políticas y culturales que doten a un jugador de una ventaja o la victoria.

Actualmente, las sagas *Total War* y *Sid Meier's Civilization* suponen los grandes

referentes del subgénero, con nuevos títulos concurrentemente. Aunque disponen de versiones para consolas, estas sagas y, en general, los juegos 4X destacan en ordenador al plantearse con un esquema de controles basado en teclado y ratón en lugar de los controladores típicos de las consolas.

■ **MOBA: Massive Online Battle Arena**

Possiblemente, el caso más famoso, no solo en cuanto al género de estrategia, sino respecto a la historia del videojuego, que produjo el nacimiento de un subgénero, sea *Warcraft™ III: Reign of Chaos* como una modificación del famoso *Warcraft™ III: Reign of Chaos* [45]. En *DOTA*, dos equipos de cinco jugadores se enfrentan controlando personajes ficticios en una competición en busca de dominar el territorio enemigo, con el objetivo final de destruir su base principal. Este nuevo género acabó bautizado como *MOBA*, procedente de las siglas en inglés *Multiplayer Online Battle Arena*.

Con el tiempo, los autores originales de *DOTA* partieron por caminos distintos, pero surgieron dos estudios altamente relevantes en la actualidad para retomar las riendas del género. Por un lado, Valve mostró interés por el concepto y la propiedad intelectual [46], originando un nuevo proyecto que alcanzaría el mercado como *DOTA 2* ([Figura 3.9](#)). A la vez, varios entusiastas del título decidieron aventurarse a establecer su propia empresa de videojuegos, Riot Games, estrenando *League of Legends* después de reclutar a algunos desarrolladores de *DOTA* [47].



Figura 3.9: Captura de pantalla de *DOTA 2*. Extraída de [Steam](#)

■ *Auto battler o Auto Chess*

Desde el mismo *DOTA 2* volvió a repetirse, en menor escala, una historia similar con la aparición de *Dota Auto Chess* en 2019. Un juego del género *Auto Chess*, también denominado *Auto battler*, propone enfrentamientos entre varios jugadores definiendo las posiciones de fichas en un tablero, seguido de una etapa de batalla uno contra uno donde las unidades combaten de forma automática [48].

Enseguida, más desarrolladores apostaron de nuevo por este subgénero con títulos como *TeamFight Tactics*, *Dota Underlords* y *Hearthstone Battlegrounds*.

■ *Tower Defense*

Un último subgénero destacado en esta sección corresponde a un tipo de juegos con menos impacto, pero ciertamente destacados por su comunidad de jugadores. La defensa de torres o *Tower Defense* hace referencia a títulos de estrategia donde el jugador debe defender un punto concreto del escenario de enemigos controlados por el juego, recurriendo a la construcción de “torres” (algunos títulos no recurren explícitamente torres en cuanto a edificios, algunas entregas utilizan personajes o elementos del entorno) capaces de atacar automáticamente.

En un *Tower Defense*, los enemigos suelen aparecer por oleadas y recorren un camino establecido y visible por el jugador, fomentando la construcción de las defensas en posiciones estratégicas a lo largo del mismo. Un buen ejemplo de este subgénero lo compone la saga *Bloons™* (Figura 3.10), utilizando un estilo visual muy caricaturesco y colorido, donde las “torres” son figuradas como animales y artilugios con habilidades variopintas.



Figura 3.10: Captura de pantalla de *Bloons™ TD 6*. Extraída de [Steam](#)

3.3. Realidad Aumentada y videojuegos

Continuando los contenidos mencionados en el [Capítulo 1](#), este último apartado detallará algunos aspectos adicionales sobre el concepto de Realidad Aumentada y su aplicación en los videojuegos. **Los conceptos de Realidad Aumentada, Virtual y Mixta empiezan a propagarse desde la década de los 90**, pero el uso de la estereoscopia se remonta a varias décadas antes de estos conceptos. En 1968, Ivan Sutherland, junto a Bob Sproull, desarrollaron el primer prototipo de casco de Realidad Virtual, nombrado cómicamente “Espada de Damocles” por su tosca apariencia encima de la cabeza del usuario [49] [50]. Hoy en día, la Realidad Aumentada deja a un lado el hardware especializado a favor del uso de dispositivos móviles o cámaras web, aunque hay algunos intentos que han tratado de reintroducir accesorios con hardware para la generación de un entorno de Realidad Aumentada.

En cuanto a aplicaciones, es común dar con un amplio interés por la Realidad Aumentada en los ámbitos de **educación, entrenamiento, medicina e industria** [51] [52] [53], con soluciones y avances continuos ante problemas cada vez más complejos. Aplicado a videojuegos, basta con recordar el éxito del título *Pokémon™ Go* durante su salida en 2016 para afirmar la popularidad del concepto entre los jugadores móviles.

Salvo casos extremadamente concretos como el anterior, los videojuegos no muestran un uso recurrente de la Realidad Aumentada, más allá de diversos intentos por replicar el impacto de *Pokémon™* por parte de otras marcas altamente conocidas como *Minecraft* ([Figura 3.11](#)), *The Witcher™*, e incluso *Harry Potter™*, que acabaron cancelando sus proyectos o deteniendo el funcionamiento de las entregas.



Figura 3.11: Imagen promocional de *Minecraft Earth*. Extraída de la [página web oficial de Minecraft](#)

3.3.1. Uso de marcadores

Donde sí tiene algo más de éxito la combinación de Realidad Aumentada y videojuegos, o la gamificación en general, es al introducir marcadores físicos que representen un punto del entorno real. Estos marcadores permiten a la aplicación realizar el seguimiento de objetos e incluso el cálculo de profundidad [54], facilitando, además, la alineación del mundo virtual sobre el entorno del usuario.

En la literatura, muchos investigadores continúan dedicando sus esfuerzos a explotar el concepto y su utilidad, implementando aplicaciones orientadas como juegos educativos o *serious games*. A modo de ejemplo, *EduPARK* [55] es una aplicación educativa centrada en un espacio natural, donde los estudiantes pueden navegar ese entorno mientras resuelven cuestionarios y son presentados con curiosidades y detalles adicionales. Por otro lado, *Stackable Music* [11] aprovecha los marcadores y su orientación como instrumentos individuales, que el jugador deberá apilar formando un grupo musical y una melodía.

Capítulo 4

Especificación del proyecto

En un mundo digital donde predominan los desarrollos software cada vez más extensos y complejos, **el concepto original de Ingeniería del Software queda, en gran medida, obsoleto**. Con el incremento incesante en los costes de producción, esta disciplina prioriza alcanzar un software de calidad bajo un presupuesto establecido, dentro de un límite temporal y con los requisitos necesarios [56]. Esto ha supuesto la evolución del concepto a lo largo de las décadas, como demuestra [57], incorporando nuevos puntos de vista basándose en la experiencia acumulada de proyectos software, tanto éxitos como fracasos, tratando de asentar unos pilares más robustos y aplicables de la forma más general posible.

Gran parte del contenido de este capítulo queda fundamentado en esas bases de la Ingeniería del Software, tomando algunos de los recursos bibliográficos dedicados a la especificación y enseñanza en este campo, propios de los inicios del siglo actual [58] y más próximos a la actualidad [59] [60] [61] [62]. Aun partiendo de estos recursos, el desarrollo de videojuegos ha alcanzado una magnitud suficiente para considerarse una especialización de la disciplina al conjuntar múltiples ámbitos de conocimiento y capacidades artísticas.

El trabajo realizado por [63] justifica la utilidad de los métodos de Ingeniería del Software al aplicarse durante el desarrollo de videojuegos, dando lugar a productos de mayor calidad y solventando gran parte de los desafíos que presentan estos proyectos, a la par que [64] indica la creciente necesidad de métodos de Ingeniería del Software adaptados a los requisitos necesarios por los proyectos de videojuego. Estudios más recientes [65] [66] continúan exponiendo esta necesidad de especializar la Ingeniería del Software en una rama dedicada a los videojuegos, pues suponen una serie de desafíos nuevos y cambiantes ligados a la rápida evolución de la industria.

Pese a que esta necesidad ha comenzado a ser especialmente relevante en proyectos más recientes, su estudio consta de trabajos publicados durante los inicios del siglo actual [67] [68], si bien algunos quedan enfocados como una generalización aplicada a mundos virtuales [35] [69]. Simultáneamente, otra especialización del videojuego con alta popularidad consiste en el desarrollo de juegos serios (*Serious Games*), priorizando el aprendizaje sobre una materia o problema concreto desde un punto de vista totalmente educativo. Este ámbito también presenta multitud de fuentes bibliográficas indagando en su etapa de diseño, aunque, dado los objetivos educativos, muestran algunas diferencias en estas fases [70] [71] [72].

Siguiendo el contenido de los trabajos mencionados, y considerando el alcance del proyecto como Trabajo Fin de Máster, este capítulo queda dedicado a exponer los aspectos de Ingeniería del Software seleccionados como más relevantes. Se incluirán, por tanto, secciones dedicadas a: la **especificación de requerimientos**, definiendo con precisión aquello que el sistema debe realizar y bajo qué limitaciones; el desglose de la **metodología de desarrollo** a seguir durante el transcurso del trabajo y cómo se estructurarán los avances y metas obtenidas; la **planificación y asignación de tareas** planteadas para alcanzar los objetivos establecidos a lo largo de la duración del proyecto; la identificación de los **riesgos** que podrán afrontarse durante el desarrollo; y una estimación económica del **presupuesto** requerido para el proyecto.

4.1. Especificación de requerimientos software

El concepto de requisito suele ser visto como algo que debe cumplirse necesariamente, pero en el contexto del software consiste en una necesidad del cliente que puede variar con el tiempo [62]. En general, un requerimiento software define qué debe hacer el sistema y cómo debe hacerlo, mientras que, en paralelo, existen una serie de restricciones de diseño que rigen la manera en que se implementa ese sistema. Cualquier desarrollo que no establezca formalmente los requisitos o donde no queden expuestas con claridad las necesidades del cliente sufrirá de un elevado número de cambios durante su transcurso, influyendo considerablemente sobre los costes [61].

Aplicado a un videojuego, el análisis de requisitos mantiene su utilidad al formalizar qué debe hacerse [73], con una diferencia crítica: en el software clásico, el cliente tiene claro su objetivo, mientras que un videojuego se compone de ideas, necesidades y objetivos en constante modificación. Dada esta inestabilidad, los equipos de desarrollo recurren al uso de más herramientas para definir todo lo necesario, convergiendo en un **documento de diseño del juego** o **GDD** como el redactado en el Capítulo 2.

El objetivo principal de este documento es su utilidad como fuente principal de información al encontrar cualquier duda durante la producción, asegurando que el enfoque se mantenga sobre los objetivos e ideas propuestas. A partir del *GDD* elaborado, los siguientes apartados incluyen la colección de requisitos establecidos para el proyecto, recopilados en una única sección dedicada.

4.1.1. Requerimientos funcionales

Un **requisito funcional** queda definido como aquel requisito que define **lo que debe hacer un sistema** software según las necesidades del cliente. De cara al resultado final del proyecto, pueden plantearse como un listado de funcionalidades que el sistema puede o no puede realizar, sin dejar lugar a estados intermedios. A continuación se enumeran los requisitos funcionales recopilados para este proyecto:

- **RF1:** El sistema utilizará el hardware móvil (cámara y sensores) y metáforas propias de la plataforma (uso de gestos, interacción sobre pantalla táctil, etc.).
- **RF2:** El sistema hará uso de la Realidad Aumentada como método principal de interacción durante la partida.
- **RF3:** El sistema garantizará la simplicidad y facilidad de uso de los elementos de la interfaz gráfica.
- **RF4:** El usuario podrá dar comienzo o finalizar una partida en curso en cualquier momento.
- **RF5:** El sistema contará con diversas opciones de configuración de la aplicación, accesibles en cualquier momento.
- **RF6:** El sistema garantizará el cumplimiento de las normas de juego establecidas durante el transcurso de la partida.

4.1.2. Requerimientos no funcionales

Como soporte a los requisitos funcionales, un **requisito no funcional** establece **la forma en que deben alcanzarse**, en ocasiones estableciendo unos valores significativos. En el siguiente listado quedan redactados los requisitos no funcionales planteados para el proyecto:

- **RNF1:** El sistema se ejecutará en dispositivos *Android* con soporte para *ARCore*
- **RNF2:** El sistema no hará uso de conexión a Internet durante su uso.
- **RNF3:** El sistema funcionará a una tasa de fotogramas superior a los 30 fotogramas por segundo (*FPS*) con un límite superior de 60 FPS.
- **RNF4:** El sistema contará con un tiempo de respuesta mínimo en tiempo real ante la interacción del usuario.
- **RNF5:** El sistema hará uso de múltiples escenas y ventanas que guíen al usuario sobre todo el contenido.
- **RNF6:** El sistema deberá ser escalable frente a nuevo contenido bajo las mismas mecánicas básicas.
- **RNF7:** El sistema permitirá visualizar los textos en dos idiomas, español e inglés, según las preferencias del usuario.

4.2. Metodología de desarrollo

A la hora de preparar los procesos de desarrollo en proyectos de videojuegos, predomina el uso de metodologías ágiles frente a otras alternativas clásicas [63] [65], dada la inestabilidad del entorno y la frecuencia de cambios en la planificación u objetivos. En cuanto a metodologías concretas, existe gran variedad de uso en la industria del videojuego: es común dar con desarrollos basados en *Scrum*, Programación Extrema o metodologías personalizadas para los equipos de trabajo encargados. Además, las plataformas móviles han supuesto otra motivación por el uso de estas metodologías ágiles para encarar sus requisitos cambiantes y con un enfoque a multitud de usuarios y dispositivos [74] [75].

Un enfoque ágil puede resultar muy dispar al implantarse en proyectos de desarrollo, originando multitud de metodologías definidas y documentadas bajo ciertas propiedades. El **Desarrollo Orientado a Características** (*Feature Drive Development* o **FDD**) centra los esfuerzos en las características necesarias por el cliente, definiendo un ciclo de vida en cinco pasos: definición del **modelo general**, especificando el contexto y alcance del desarrollo; construcción de la **lista de características** según las necesidades del cliente; **planificación**, por característica, con las metas del incremento y asignación de las tareas; **diseño de cada característica** mediante documentación y/o diagramas destinados al equipo de trabajo; y la **producción de características**,

tras lo que serán implementadas y acopladas al sistema. Aunque permite centrar el desarrollo en las necesidades definidas, presenta dificultades en entornos más inestables donde dichas necesidades pueden ser alteradas [76].

Por otro lado, encontramos el **Desarrollo Orientado a Pruebas** (*Test Drive Development* o **TDD**) como una metodología que busca garantizar la calidad del código, definiendo una serie de pruebas sobre las que realizar la implementación. Su ciclo de vida consiste en la **definición de pruebas** concretas (*unit test*) que describen el comportamiento del código, implementando posteriormente la funcionalidad requerida. Tras la implementación, se **ejecutan las pruebas** en busca de errores: si el código supera la prueba, el desarrollo procede a la siguiente funcionalidad; **en caso de error, el código debe repararse** hasta cumplir los requisitos de las pruebas. Las principales desventajas que presenta *TDD* se encuentran en la interconexión de funcionalidades, limitada por el alcance de las pruebas unitarias, y la necesidad de conocimientos adicionales en el desarrollo de pruebas por parte de los programadores. Por esto, suele adoptarse parcialmente durante el uso de otras metodologías [76].

La **Programación Extrema** (*eXtreme Programming* o **XP**) toma distintas prácticas comunes e intensifica su aplicación. Esta metodología comienza con una **recopilación continua de requisitos** e historias de usuario, permitiendo **adaptar las iteraciones** y centrar el esfuerzo del equipo; al decidir el contenido de un incremento, comienza su implementación empleando **esas prácticas de desarrollo extremas**, tales como programación en pareja (*Pair Programming*, integración continua, o el diseño de pruebas (adaptando a la metodología *TDD*). Conforme los incrementos quedan finalizados, el sistema pasa a una **fase de producción** en que se involucra al cliente para realizar ajustes menores, seguido de una **etapa de mantenimiento** donde se evalúa el producto en su totalidad con los reajustes mayores, hasta dar con una última versión que marque la finalización del proyecto [77] [76].

Empleando un enfoque visual, *Kanban* hace uso de un tablero de tareas que muestra, en todo momento, las características que componen un proyecto, tratando de simular el enfoque de entrega industrial *Just In Time* en el desarrollo de software. Sus principios básicos consisten en la **limitación del trabajo**, la correcta gestión del **flujo de trabajo**, la **mejora continua** y, como aspecto principal, la **visualización del trabajo**. Esta visualización se compone como un tablero dividido en columnas, asignadas a distintos estados de una tarea (por ejemplo: “planificación”, “en desarrollo” y “terminado”), que además son los elementos que pueblan el tablero. Mediante este tablero, todo el equipo de trabajo conoce en todo momento el estado de las tareas que componen el desarrollo [78] [79].

Como una de las metodologías más empleadas, *Scrum* trata de aportar el mayor valor posible en la menor cantidad de tiempo. Al no definir aspectos técnicos del desarrollo y mantener cierta simplicidad, puede adaptarse a multitud de proyectos. Su funcionamiento consta de una primera etapa donde se definen **historias de usuario**, representando las necesidades del cliente, que serán implementadas iterativamente en **sprints** (ciclos de desarrollo esparcidos en varias semanas). Para ser implantada, *Scrum* requiere la asignación de tres roles: el **Propietario del Producto** como el responsable de los objetivos establecidos según los requisitos evaluados, además de revisar las mejoras de cada *sprint*; el **Maestro de Scrum** encargado de gestionar el proyecto y el cumplimiento de las historias de usuario, además de mantener la comunicación entre el equipo y el cliente o cualquier entidad externa; por último, el **equipo de trabajo** queda compuesto por los desarrolladores encargados de la gestión, análisis, diseño e implementación del sistema tal que cumpla con los requisitos de la historia de usuario en cuestión [80] [79].

Las metodologías anteriores cuentan con sus ventajas y desventajas propias, especialmente al adaptarse a proyectos concretos. La [Tabla 4.1](#) agrega, para las cinco metodologías, las cualidades más ventajosas (filas junto al símbolo de suma “+”) y los defectos (filas junto al símbolo de resta “-”) bajo el contexto del proyecto a desarrollar. A partir de esta comparativa, **se decide aplicar una metodología ágil incremental** inspirada mayoritariamente por *Scrum* al desarrollo, pues cuenta con una gran adaptabilidad a las necesidades concretas y requiere de amplia experiencia para los roles que asigna. En cuanto al resto, *TDD* y *XP* son descartadas al suponer un esfuerzo adicional para la incorporación de pruebas unitarias y, en el caso de *XP*, no disponer de un grupo de trabajo sobre el cual delegar los distintos roles. Esto último también justifica el descarte de *FDD*, añadido a la falta de experiencia en la recopilación y análisis de requisitos para garantizar la fluidez de la metodología. Por último, aunque *Kanban* pueda ajustarse al proyecto, la ausencia de roles y definición de incrementos podría nublar la visión del proyecto sin establecer objetivos claros durante el desarrollo.

4.2.1. Metodología *Scrum*

Indagando en el funcionamiento de *Scrum*, el ciclo de desarrollo comienza con la definición de un listado del producto, que agrupa los requisitos identificados en historias de usuario para su implementación. Al seleccionar la historia a desarrollar en el *sprint*, queda realizada una **planificación** que subdivide el trabajo en tareas concretas, asignadas al equipo de desarrollo que se encargará del **seguimiento e implementación** hasta alcanzar una **versión entregable**. Finalmente, sobre este entregable, se

	FDD	TDD	XP	Kanban	Scrum
+	Adaptable a cambios de requisitos	Código robusto y validado	Incrementos rápidos y de menor tamaño	Organización visual del trabajo	Adaptable a cualquier entorno
	Entregables en 1 a 4 semanas	Mejor diseño e integración	Pruebas unitarias garantizan calidad	Flexibilidad del flujo de trabajo	Entregables en 1 a 4 semanas
-	Gran cantidad de roles	Falta de experiencia en diseño de pruebas	Gran cantidad de roles	Sin incrementos definidos	Sin responsabilidades específicas por miembro
	Falta de experiencia en roles clave	Poco ajustable al tipo de aplicación a desarrollar	Requiere un grupo de desarrollo	No define roles ni responsabilidades	No propone prácticas de desarrollo
	Carga de trabajo adicional para las pruebas	Carga de trabajo adicional para las pruebas	Carga de trabajo adicional para las pruebas		

Tabla. 4.1: Ventajas y desventajas de metodologías de desarrollo ágil

realiza una **revisión** validando los avances que propone el *sprint* y una **retrospectiva** de cara a planificar posibles mejoras en el sistema. Este ciclo queda expuesto visualmente en la Figura 4.1.

SCRUM FRAMEWORK

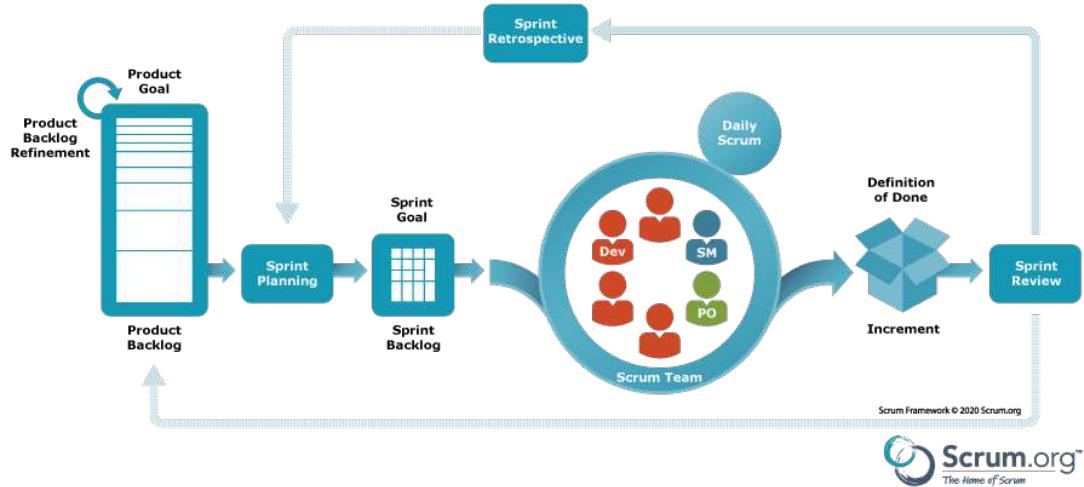


Figura 4.1: Esquema de funcionamiento de la metodología ágil *Scrum*. Extraído de: <https://www.scrum.org/resources/scrum-framework-poster>

A pesar de que la metodología *Scrum* define tres roles a lo largo de su ciclo de vida, ya que el equipo de desarrollo está constituido únicamente por el autor de este Trabajo Fin de Máster, tendrá todos estos roles asignados. Por esto, será responsable del análisis de requisitos y definición de los objetivos como **propietario del producto**, de la gestión de tareas y validación de los avances del proyecto como **maestro de Scrum**, y del diseño e implementación de las partes del sistema en cada *sprint* como **equipo de Scrum**.

Como principal ventaja de este enfoque unipersonal se destaca la simplicidad del seguimiento y realización de reuniones que propone *Scrum*: por ejemplo, las reuniones diarias quedan reducidas a una revisión breve del trabajo realizado, resultando en un nuevo estado del proyecto en el sistema de gestión de versiones empleado (comúnmente denominado *commit*)¹. Durante la revisión del *sprint*, será mediante los *commit* realizados donde se verificará la implementación de los avances propuestos.

4.2.2. Historias de usuario iniciales

Partiendo del análisis de requisitos y el *GDD* redactado, es posible establecer una serie de **historias de usuario** que definan varios aspectos a implementar del sistema. Será desde este listado del que partirán los distintos *sprints* que compondrán el desarrollo del proyecto, pudiendo alterar en cierta medida la colección de historias de usuario en caso de que los requisitos se vean modificados.

Cada historia cuenta con, además de su descripción, un **valor de prioridad** que indica la relevancia sobre el sistema final, sirviendo como método de elección de los objetivos en cada *sprint*. Este valor varía desde 1, indicando menor prioridad, hasta 5, como prioridad máxima. La [Tabla 4.2](#) muestra el listado de historias de usuario definidas para el proyecto.

Identificador	Historia de usuario	Prioridad
U.S. 1	Como jugador, quiero iniciar, pausar y terminar una partida en cualquier momento.	5
U.S. 2	Como jugador, quiero tener acceso a un menú con opciones generales de la aplicación.	3
U.S. 3	Como jugador, quiero ver el tablero de juego con mi dispositivo mediante la Realidad Aumentada.	5
U.S. 4	Como jugador, quiero interactuar en tiempo real con el entorno de Realidad Aumentada y la aplicación.	4
U.S. 5	Como jugador, quiero obtener un resumen con estadísticas de la partida al finalizar	2
U.S. 6	Como jugador, quiero leer los textos en mi idioma de preferencia.	1

Tabla. 4.2: Historias de usuario definidas para el proyecto.

¹Si bien el sistema concreto se detallará en capítulos posteriores, todos los sistemas de versionado hacen uso de los mismos conceptos y flujo de funcionamiento, aunque utilicen nomenclaturas dispares.

4.3. Gestión de riesgos

Un aspecto común de las metodologías de desarrollo ágiles es la ausencia de un enfoque relevante sobre los riesgos presentes en el transcurso del proyecto [81] [82]; mientras que las metodologías tradicionales tratan de mantener cierta estabilidad, el desarrollo ágil se basa principalmente en **aceptar el cambio y adaptarse**. Adicionalmente, un videojuego presenta situaciones de riesgo no aplicables en otros ámbitos y que pueden comprometer los resultados del proyecto [83]. En el ámbito software, un **riesgo** es un evento que puede perjudicar al proyecto, con la distinción entre riesgos “controlados” si pueden manejarse y mitigarse directamente y “no controlados” si surgen de fuentes externas.

Todo riesgo identificable supone cierto impacto sobre un activo concreto involucrado en el proyecto. Antes de realizar un análisis de riesgos, quedan identificados los activos relevantes del proyecto y los riesgos asociados a cada uno de ellos:

- **Producto:** Corresponde al videojuego (prototipo) desarrollado durante el proyecto como una aplicación software para dispositivos móviles *Android*.
 - **Factor diversión:** El videojuego puede no resultar atractivo para el usuario final, debido a problemas en la implementación de sus mecánicas, mal ajuste de sus reglas o simplemente como factor personal del usuario [84].
 - **Alcance excesivo:** Un descontrol en el alcance del proyecto añade complejidad a las labores de gestión y ralentizará el desarrollo, así como suponer una pérdida de calidad al reducir el esfuerzo aplicado a pruebas.
 - **Errores de funcionamiento:** Cualquier problema encontrado durante el uso de la aplicación, desde un comportamiento inesperado hasta bloqueos del sistema o problemas de rendimiento.
- **Software del proyecto:** Engloba el código fuente desarrollado y los recursos adicionales empleados (modelos 3D, imágenes, efectos sonoros y visuales...).
 - **Pérdida de información:** Extravío de cualquier tipo de información relacionada con el desarrollo. Supone la realización de procesos de recuperación que interrumpen el desarrollo del proyecto.
- **Herramientas software:** Todas las herramientas y utilidades software empleadas para la producción como aplicaciones del sistema o en línea (sistema operativo, motor gráfico, manejo de imágenes/sonido, servidor de gestión de versiones, servidores de almacenamiento remoto, etc.).

- **Pérdida de información:** Extravío de credenciales, borrado de ficheros remotos, etc. que imposibiliten el acceso a los datos de la herramienta. Varía desde errores de guardado en sistemas locales hasta el borrado imprevisto o revocación de acceso en servidores y repositorios remotos.
 - **Falta de acceso:** Impedimento del acceso a servidores por situaciones imprevistas, principalmente debido a caídas de servidores remotos.
 - **Incompatibilidad de formatos:** Uso de herramientas con resultados incompatibles entre sí. Resultan en el uso de otras herramientas alternativas.
- **Dispositivo de pruebas:** Dispositivo móvil *Android* empleado para la realización de pruebas con el prototipo.
- **Compatibilidad hardware:** Obsolescencia al hacer uso de tecnología novedosa, o no alcanzar los requisitos mínimos para un rendimiento aceptable.
 - **Extravío:** Pérdida del dispositivo o limitación de su uso, evitando la realización de pruebas de uso.
 - **Fallo de funcionamiento:** Errores que imposibiliten el uso del dispositivo, como problemas del sistema operativo, software conflictivo o degradación de la batería y/o sensores.
- **Equipo hardware:** Dispositivo de escritorio y periféricos empleados para el desarrollo del proyecto, así como utensilios comunes adicionales.
- **Corte en la conexión a la red:** Detención del acceso a Internet por parte del dispositivo, originado por problemas en el hardware de comunicación (tarjeta de red, cable Ethernet o router) o fallos de configuración software.
 - **Compatibilidad hardware:** Obsolescencia al hacer uso de tecnología novedosa, o no alcanzar los requisitos mínimos para un rendimiento aceptable.
 - **Fallo de funcionamiento:** Errores que imposibiliten el uso del dispositivo por motivos como hardware defectuoso o fallos de configuración software.
 - **Malware:** Software malicioso que amenace la continuidad del desarrollo.
- **Lugar de trabajo:** Espacio físico habitable dedicado para el equipo hardware y desarrollador principal, supliendo las necesidades básicas.
- **Corte del suministro eléctrico:** Detención del suministro eléctrico, producido por problemas del proveedor o de la instalación local.
 - **Siniestro:** Catástrofes que supongan la pérdida total o parcial del lugar de trabajo y las instalaciones y el material contenido, provocado por incendios, averías o desastres naturales.

- **Desarrollador principal:** Persona encargada de la realización del proyecto y las tareas que lo componen.
 - **Estrategia de desarrollo:** Falta de precisión o claridad al especificar metodologías, guías e instrucciones para el desarrollo del proyecto.
 - **Personal poco cualificado:** Falta de conocimiento o experiencia previa en las labores a realizar.
 - **Calendario y presupuesto:** Imprecisiones y retardos en la estimación de entregables y costes del proyecto.
 - **Motivación:** Falta de motivación para la continuidad del desarrollo de una forma ágil y eficiente.

Mediante la [Tabla 4.3](#), quedan analizados los riesgos planteados mediante tres valores numéricos: la **probabilidad de ocurrencia** de la situación, con valores comprendidos entre 1 para menor probabilidad (a lo sumo, una vez al año) y 5 para mayor probabilidad (una o más veces al día); el **impacto** del riesgo sobre el sistema, valorado entre 1 cuando su influencia es negligible y 5 cuando puede suponer problemas críticos o quede amenazada la continuidad del proyecto; por último, un **valor de riesgo** establecido con la suma de los anteriores y empleado como escala a la hora de priorizar los riesgos involucrados.

Activo	Riesgo	Probabilidad	Impacto	Valor
Producto	Factor diversión	5	5	10
Producto	Alcance excesivo	4	3	7
Producto	Errores de funcionamiento	3	5	8
Software del proyecto	Pérdida de información	2	5	7
Herramientas software	Pérdida de información	2	4	6
Herramientas software	Falta de acceso	2	4	6
Herramientas software	Incompatibilidad de formatos	1	2	3
Dispositivo de pruebas	Compatibilidad hardware	1	3	4
Dispositivo de pruebas	Extravío	1	5	6
Dispositivo de pruebas	Fallo de funcionamiento	1	5	6
Equipo hardware	Corte en la conexión a la red	1	3	4
Equipo hardware	Compatibilidad hardware	1	2	3
Equipo hardware	Fallo de funcionamiento	1	5	6
Equipo hardware	Malware	1	3	4
Lugar de trabajo	Corte del suministro eléctrico	2	5	7
Lugar de trabajo	Siniestro	1	5	6
Desarrollador principal	Estrategia de desarrollo	4	5	9
Desarrollador principal	Personal poco cualificado	5	3	8
Desarrollador principal	Calendario y presupuesto	4	4	8
Desarrollador principal	Motivación	4	3	7

Tabla. 4.3: Análisis de riesgos del proyecto.

Para concluir el estudio y preparación ante los riesgos especificados, queda planteada una planificación de la respuesta ante la ocurrencia de cada riesgo, tratando de **mitigar** su impacto en el proyecto, **prevenir** la ocurrencia del evento imprevisto o **asumir** los daños que origine. La planificación frente a los riesgos se presenta mediante la [Tabla 4.4](#), que indica la acción y el plan de actuación ante el suceso.

Activo	Riesgo	Valor	Acción	Plan
Producto	Factor diversión	10	Mitigar	Revisión continua de calidad.
Desarrollador principal	Estrategia de desarrollo	9	Prevenir	Selección de metodología de desarrollo conocida.
Producto	Errores de funcionamiento	8	Prevenir	Realización de pruebas y uso de sistemas de tratamiento de errores críticos
Desarrollador principal	Personal poco cualificado	8	Mitigar	Uso de tutoriales, documentación y ejemplos.
Desarrollador principal	Calendario y presupuesto	8	Prevenir	Planificación de tareas.
Producto	Alcance excesivo	7	Prevenir	Planificación de tareas.
Software del proyecto	Pérdida de información	7	Mitigar	Uso de servicios de almacenamiento en la nube.
Lugar de trabajo	Corte del suministro eléctrico	7	Mitigar	Uso de aplicaciones instaladas localmente.
Desarrollador principal	Motivación	7	Mitigar	Planificación de tareas.
Herramientas software	Pérdida de información	6	Asumir	
Herramientas software	Falta de acceso	6	Mitigar	Uso de aplicaciones instaladas localmente.
Dispositivo de pruebas	Extravío	6	Asumir	Adquisición de un nuevo dispositivo.
Dispositivo de pruebas	Fallo de funcionamiento	6	Prevenir	Actualizaciones del software del dispositivo.
Equipo hardware	Fallo de funcionamiento	6	Prevenir	Actualizaciones del software del dispositivo y pruebas de funcionamiento (benchmarks)
Lugar de trabajo	Siniestro	6	Asumir	Uso de servicios de almacenamiento en la nube.
Dispositivo de pruebas	Compatibilidad hardware	4	Prevenir	Estudio previo de alternativas de desarrollo.
Equipo hardware	Corte en la conexión a la red	4	Asumir	Uso de aplicaciones instaladas localmente.
Equipo hardware	Malware	4	Prevenir	Uso de antivirus, análisis del dispositivo y uso responsable.
Herramientas software	Incompatibilidad de formatos	3	Prevenir	Estudio previo de alternativas de desarrollo.
Equipo hardware	Compatibilidad hardware	3	Prevenir	Estudio previo de alternativas de desarrollo.

Tabla. 4.4: Planificación de acciones ante riesgos del proyecto.

4.4. Planificación

Una vez definida la metodología de desarrollo a emplear, esta sección tratará la planificación del desarrollo a lo largo de la duración del proyecto, desglosando los incrementos y tareas para alcanzar los objetivos definidos. Este desglose será empleado a la hora de establecer cierto orden en las tareas y *sprints* durante el desarrollo ágil, pero puede ser ligeramente alterado según se considere necesario.

4.4.1. Descomposición del trabajo

En primer lugar, como punto de partida para el desglose, el objetivo principal del proyecto queda descompuesto en diversas tareas y subtareas que deben completarse hasta cumplir con tal objetivo. Siguiendo la [Figura 4.2](#), el objetivo principal es el desarrollo del prototipo del videojuego *LandmARk*; para ello, se plantean una serie de tareas parciales: la **revisión bibliográfica**, indagando en explorar el estado actual de la industria del videojuego, así como su proceso de desarrollo, y el uso de la Realidad Aumentada; la **planificación y el diseño** del prototipo a desarrollar, especificando cuestiones de jugabilidad y mecánicas, temática y estilo del entorno, y primeros diseños de la interfaz de usuario e interacción; el proceso iterativo de **implementación**, desde la selección de alternativas y recursos para el desarrollo, hasta la generación de la aplicación; y, finalmente, la realización de un **mantenimiento** en forma de pruebas con jugadores y la corrección de errores encontrados tras el despliegue.

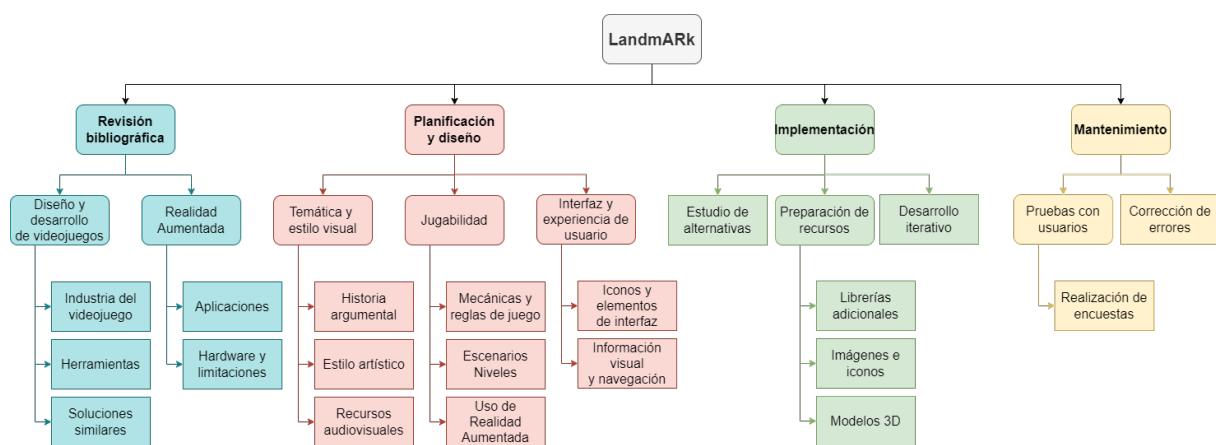


Figura 4.2: Diagrama de descomposición del trabajo, con varios niveles jerárquicos de especialización.

A continuación queda detallada la división del trabajo en varios incrementos, mediante un título identificativo que resuma brevemente sus objetivos, junto a una descripción de los contenidos que abarca cada uno. Con el siguiente listado, además, se

establece el orden en que se tratarán de seguir durante el desarrollo, priorizando en *sprints* tempranos aquellas tareas críticas para el prototipo:

- **Incremento 0. Estudio de alternativas:** Recopilación, análisis y pruebas con diferentes alternativas de herramientas para el desarrollo de videojuegos, principalmente motores gráficos y librerías adicionales. Quedarán valoradas en términos de facilidad de uso, funcionalidad, documentación, ejemplos y familiaridad por parte del desarrollador principal. Selección de otras herramientas como editores de imagen, estaciones de sonido o servicios en línea.
- **Incremento 1. Puesta en marcha y marcadores de RA:** Preparación del proyecto y librerías a emplear durante el desarrollo. Incorporación de un entorno en Realidad Aumentada mediante marcadores, habilitando la interacción en tiempo real sobre dicho entorno. Diseño y producción de los marcadores a utilizar.
- **Incremento 2. Sistema de turnos y tablero de juego:** Reglas y mecánicas esenciales de juego, respetando el orden de los turnos para las jugadas y movimientos básicos sobre el tablero. Generación de un tablero virtual sobre el que se desarrolla el juego.
- **Incremento 3. Interacción sobre el tablero y fichas de juego:** Definición de la interacción durante la partida mediante el dispositivo y el entorno de RA en tiempo real, permitiendo el desplazamiento de fichas sobre el tablero bajo las reglas especificadas. Diferenciación entre tipos de ficha disponibles, con propiedades diversas que influyen en su comportamiento.
- **Incremento 4. Recursos audiovisuales e interfaz gráfica:** Población de la escena de juego con modelos 3D, elementos y controles de interfaz gráfica y efectos audiovisuales. Vista inicial al abrir la aplicación de un menú con opciones de inicio de partida y ajustes de la aplicación.
- **Incremento 5. Despliegue del prototipo:** Ciclo principal de juego y conexión entre las escenas y ventanas implementadas, desde que el usuario presiona la opción de iniciar partida hasta que finaliza de cualquier forma. Generación del ejecutable, instalación en el dispositivo de pruebas y comprobación de su correcto funcionamiento.
- **Incremento 6. Pruebas y mantenimiento:** Etapa de pruebas con usuarios y revisión de calidad del prototipo. Arreglo de errores encontrados y estudio de las valoraciones de usuarios.

4.4.2. Planificación temporal

Este apartado se dedica a establecer la planificación de las tareas que componen el proyecto a lo largo del tiempo, mediante el diagrama de Gantt de la [Figura 4.3](#). Cada barra del diagrama representa una tarea general, y su posición y longitud indica el lapso de tiempo en el que se espera resolver. El tramo de finales del año 2023 sin dedicación corresponde a las vacaciones navideñas del mismo.

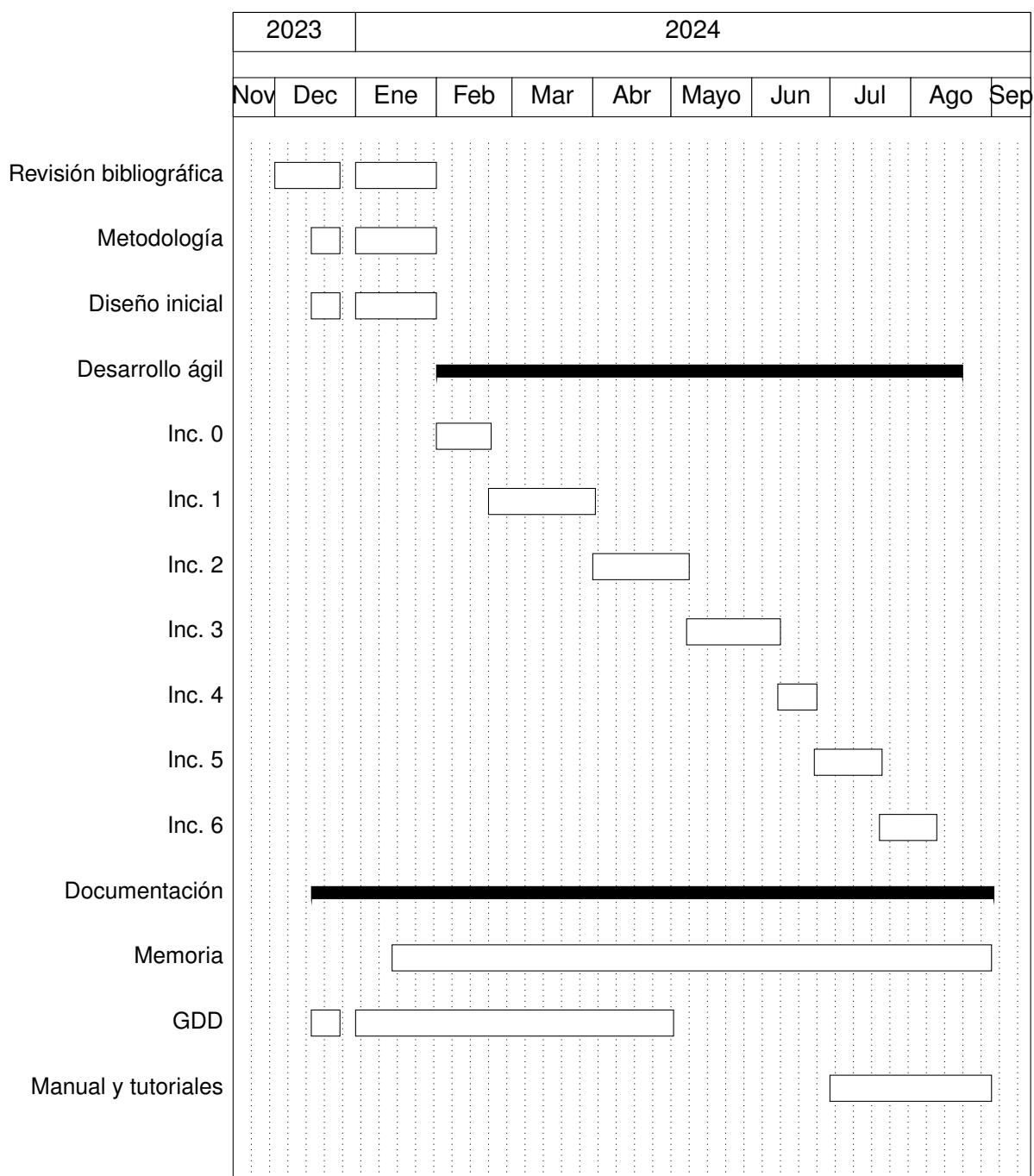


Figura 4.3: Diagrama de Gantt del proyecto

4.4.3. Asignación de roles

Teniendo en cuenta que el equipo de desarrollo está formado por una única persona y el amplio abanico de roles involucrados en el desarrollo de un videojuego, se propone la asignación dada en la [Tabla 4.5](#), presentando los roles considerados para el proyecto junto a una medida de dedicación sobre cada uno de ellos. Esta medida indica el esfuerzo dedicado a las tareas propias del rol en cuestión; por ejemplo, en cuanto a modelado 3D, la mayoría de modelos se obtienen de librerías o paquetes externos, reduciendo las tareas de modelado a un mínimo y, por tanto, con un porcentaje bajo para esta medida.

Rol	Descripción	Dedicación
Director creativo	Propone ideas que forman el juego y toma las decisiones más relevantes del proyecto.	100 %
Diseñador	Formaliza las ideas en documentos para los desarrolladores especializados y supervisan el trabajo que realizan los diversos equipos.	100 %
Programador	Implementa el código necesario para obtener los comportamientos propuestos en el diseño. Puede especializarse en jugabilidad, interfaz de usuario, efectos audiovisuales, etc.	100 %
Modelador 3D	Crea objetos virtuales formados por polígonos en tres dimensiones y los dota de color y propiedades mediante el uso de texturas.	20 %
Artista 2D	Crea ilustraciones sobre la temática del juego. Puede especializarse en artista conceptual, ilustrador, artista de entorno o artista de interfaz de usuario.	10 %
Animador 3D + <i>Rigging</i>	Añade movimiento a un modelo 3D sobre un esqueleto (<i>rig</i>). En ocasiones, la creación de este esqueleto queda ligada al propio animador o a un desarrollador especializado en <i>rigging</i> .	100 %
Artista VFX	Diseña y crea efectos visuales, caracterizando las acciones o sucesos del juego.	60 %
Diseñador de sonido	Diseña y crea las pistas de audio para diversos fines: efectos sonoros, banda sonora o voces.	10 %
Ingeniero de sonido	Incorpora las pistas de audio al juego bajo unas cualidades específicas: sonido 3D, filtros de audio, canales de audio, etc.	100 %
Diseñador UI	Diseña la interacción y elementos de la interfaz mostrados al jugador. Asegura la facilidad de uso y simplicidad de la interacción con la interfaz.	100 %
QA Tester	Evaluá el producto resultante en busca de problemas de funcionamiento, rendimiento, o dinámicas del juego. Comunica estos problemas al resto del equipo para su subsanación.	100 %

Tabla. 4.5: Roles involucrados en el desarrollo del proyecto

4.5. Presupuesto

Partiendo de la estimación temporal y considerando el periodo de vacaciones de Navidad del año 2023, el proyecto quedaría desarrollado en, aproximadamente, 35 semanas divididas en nueve meses. Con una dedicación semanal de 10 horas, la extensión del desarrollo aumentará hasta las **350 horas**. Consultando el XVIII Convenio Colectivo estatal de empresas de consultoría, tecnologías de la información y estudios de mercado y de la opinión pública de 2023 [85], estableciendo una jornada laboral de 1.800 horas anuales, y considerando el rol de maestro de *Scrum* como el de más cargo de conocimiento y experiencia, su oficio puede encajarse en el grupo A del área 3, con un total de 28.850,23 € anuales (salario base más plus de convenio). Adicionalmente, se incluye un 35 % de este salario como costes añadidos de Seguridad Social, suponiendo un gasto total de **38.947,81 € anuales**, o 3.245,65 € mensuales.

En cuanto a recursos hardware, se dispone de un equipo de escritorio de las siguientes características: procesador Intel i7 de doceava generación; 32 GB de memoria RAM DDR4; Disco duro SSD Sata 3 con capacidad de 1 TB; tarjeta gráfica Nvidia RTX 3060. Un equipo de estas prestaciones, **añadiendo los periféricos y recursos hardware adicionales básicos** (como cableado o unidades de almacenamiento externo), puede **valorarse en 1.200 €**. Adicionalmente, la aplicación será desplegada para su uso en un dispositivo móvil POCO F5 12+128 GB, **valorado en 365 €**.

Para las herramientas software a emplear, se tuvo en cuenta la existencia de licencias de uso gratuitas durante la evaluación de alternativas, favoreciendo software libre o de uso gratuito. Los recursos multimedia que empleará el prototipo serán adquiridos en librerías digitales especializadas, dedicando a la adquisición de los mismos un **máximo de 50 €**. En cuanto a licencias de software, el ordenador de trabajo contará con el sistema operativo *Windows 11 Pro*, cuya licencia está valorada en **260 €**. Como aspecto destacable, Unity ofrece una licencia de uso para proyectos personales y organizaciones con ganancias inferiores a 100.000 dólares [86], gracias a lo que su coste se mantiene a cero en esta estimación; el resto de software a emplear contará también con versiones o licencias de uso gratuitas.

Finalmente, deben añadirse una serie de gastos indirectos relativos a otros puntos necesarios durante el transcurso del proyecto. Partiendo por la conexión a la red, al comparar las distintas ofertas de los proveedores nacionales, puede aproximarse el coste de una conexión mediante fibra óptica con un **ancho de banda de 300 MB sobre los 35 € mensuales**. Respecto al consumo energético, es posible emplear herramientas en línea capaces de aproximar la factura basándonos en ciertos parámetros

del servicio contratado. Considerando un consumo medio de 270 kWh mensuales², tomamos un valor menor para las tareas del proyecto de 250 kWh, así como la contratación de 3 kW de potencia, resultando en la estimación de la [Figura 4.4](#). También se incluirá un añadido del 5% del coste del proyecto para cualquier otro gasto imprevisto y gastos indirectos.

Simulador Factura Luz		Tu tarifa - PVPC 2.0TD (PVPC 2023/Ene,2024/Ene)	
		01/12/2023 - 01/09/2024 (276 días)	
POTENCIA ... 66,76 €		Se han aplicado 2 cambios de normativa de PVPC, y lo que ves son los precios medios aplicados.	
Potencia facturada punta	P1: 3 kW x 276 días x 0,069384 €/kW día	57,45 €	
Potencia facturada valle	P2: 3 kW x 276 días x 0,002729 €/kW día	2,26 €	
Margen comercializadora	PMC: 3 kW x 276 días x 0,008514 €/kW día	7,05 €	
Potencia facturada TOTAL	P1+P2+PMC: 276 días ... 66,76 €		
CONSUMO ... 283,84 €			
Consumo facturado punta	E1: 744 kWh x 0,171210 €/kWh (p:0,076606 + e:0,094604)	127,38 €	
Consumo facturado llano	E2: 668 kWh x 0,115704 €/kWh (p:0,027954 + e:0,087750)	77,29 €	
Consumo facturado valle	E3: 926 kWh x 0,085497 €/kWh (p:0,002821 + e:0,082676)	79,17 €	
Consumo facturado TOTAL	E1+E2+E3: 2338 kWh x 0,121403 €/kWh ... 283,84 €		
Total peaje	p: 2338 kW x 0,033477 €/kWh ... 78,27 €		
Total energía	e: 2338 kW x 0,087926 €/kWh ... 205,57 €		
Fórmula	[A] Basado en PVPC		
OTROS CONCEPTOS ... 27,17 €			
Financiación Bono Social	276 días x 0,006300 €	1,77 €	
Alquiler equipos de medida	276 días x 0,026630 €	7,38 €	
Impuesto eléctrico	5.11269632% de 352,37 €	18,02 €	
TOTAL			
BASE IMPONIBLE		377,77 €	
IVA	10% de 377,77 €	37,78 €	
TOTAL FACTURA		415,55 €	

Figura 4.4: Estimación de factura de consumo energético, obtenida de la herramienta en línea [Simulador Factura Luz](#)

Los costes anteriores quedan, además, sujetos a una **amortización lineal** de su valor a lo largo de su vida útil y la duración del proyecto, salvo los recursos humanos cuyo salario ajustado ya se ha desglosado al inicio de esta misma sección. Siguiendo un esquema de amortización lineal, los bienes reciben un coeficiente de amortización anual establecido por la Agencia Tributaria [87], que se utilizará respecto al valor de adquisición para obtener el coste ajustado. Además, ya que el proyecto consta de una duración menor a un año, se ajustarán los valores obtenidos a una escala mensual en lugar de anual. Las ecuaciones a utilizar se listan a continuación, junto a la tabla [Tabla 4.6](#) que resume los gastos amortizables del proyecto.

²Información extraída de <https://www.endesa.com/es/blog/blog-de-endesa/luz/calcular-consumo-electrico-casa>

Origen del gasto	Valor compra	Coeficiente	Amortización anual	Amortización mensual	Amortización de proyecto
Ordenador de escritorio	1.200 €	20 %	240,00 €	20,00 €	180,00 €
Dispositivo móvil	365 €	20 %	73,00 €	6,08 €	54,75 €
Licencia W11 Pro	260 €	33 %	85,80 €	7,15 €	64,35 €

Tabla. 4.6: Amortización de los gastos del proyecto

$$\text{AmortizacionAnual} = \text{ValorCompra} * \text{CoeficienteAmortizacion}$$

$$\text{AmortizacionMensual} = \frac{\text{AmortizacionAnual}}{12\text{meses}}$$

$$\text{AmortizacionProyecto} = \text{AmortizacionMensual} * \text{DuracionProyecto}$$

Acumulando estos gastos, el salario destinado al personal ($3,245,65 \frac{\text{EUR}}{\text{mes}} * 9\text{meses} = 29,210,86\text{EUR}$) y los gastos adicionales en recursos adicionales ($35 \frac{\text{EUR}}{\text{mes}} * 9\text{meses} = 315\text{EUR}$ de internet y $415,55\text{EUR}$ para la factura de luz) y costes imprevistos (equivalente al 5 % del coste total, $30,240,51\text{EUR} * 1,05 = 31,752,53\text{EUR}$), el coste total del proyecto queda establecido en **31.752,53 €**.

Capítulo 5

Desarrollo iterativo

Este capítulo queda totalmente dedicado al proceso de desarrollo del prototipo de *LandmARK*. Cada sección contendrá todos los avances realizados en los múltiples incrementos especificados, detallando aquellos aspectos más relevantes del diseño (diagramas, patrones de programación o recursos a emplear, entre otros), y la implementación (fragmentos de código, capturas de pantalla, librerías y utilidades, etc.).

5.1. Incremento 0. Estudio de alternativas

Previo al desarrollo de cualquier producto software, y una vez definidos los objetivos, requisitos y metodología que rigen esta etapa, deben evaluarse las alternativas de herramientas que permitan la implementación. En este incremento se detallarán los motivos por los que, una a una, se han seleccionado o descartado las herramientas planteadas. Este paso es especialmente importante en un videojuego, pues migrar el código fuente y los recursos audiovisuales puede generar problemas de incompatibilidad entre plataformas o herramientas diferentes.

Algo indiferente a la selección final queda presente en el uso de software libre y/o gratuito, pues no suponen un incremento adicional al coste del proyecto y es posible dar con herramientas tan competentes como sus alternativas de pago o privadas. El orden que seguirá el contenido de este incremento pasará, en primer lugar, por las alternativas más claras o con menos opciones en el abanico de posibilidades, como el modelado 3D o composición sonora, desembocando en una comparativa de algunos motores gráficos que formarán el esqueleto del prototipo y requieren un estudio mucho más profundo.

5.1.1. Estación digital de audio

El sonido suele quedar en segundo plano durante el desarrollo de aplicaciones al uso. En un videojuego se convierte en una prioridad, pues constituye uno de los mejores medios, junto al visual, para presentar una respuesta al usuario de sus acciones o del entorno en que se encuentra. Desde sonidos simples como el clic de un botón hasta la banda sonora que se escuchará de fondo durante la partida, la solución a producir esas pistas sonoras pasa por el uso de estaciones digitales de audio, o comúnmente denominadas *DAW*, por sus iniciales en inglés *Digital Audio Workstation*.

Estas estaciones pueden considerarse como un instrumento más [88] al generar sonidos generalmente asociados al género *electrónico*, definiendo cada nota musical como una onda y aplicando numerosos filtros que perfilan el resultado final. Además, cualquier *DAW* permite tomar como entrada la salida de instrumentos musicales típicos, aunque con cierto nivel de complejidad adicional para asegurar suficiente calidad del sonido. Por último, es posible hacer uso de la interfaz *MIDI* (*Musical Instrument Digital Interface*) estandarizada que asemeja los controladores de instrumentos típicos al procesamiento digital.

A la hora de comparar varias *DAW*, es común encontrar la recomendación de usuarios expertos sobre *Reaper*, pues cuenta con un amplio repertorio de funcionalidades a la vez que mantiene un uso simple y fácil de comprender. La razón que provoca su descarte para el proyecto es la necesidad de adquirir una licencia de uso una vez finalice un periodo de prueba gratuito para continuar con su uso. La siguiente opción viable, dado el conocimiento previo en la herramienta, es el uso de *LMMS*, otra *DAW* con una licencia de software libre y de uso gratuito, también con una interfaz simple y fácil de usar, dividida en ventanas especializadas por funcionalidad.

El uso concreto que se dará a *LMSS* será detallado en incrementos posteriores, pero, a modo de introducción, la Figura 5.1 muestra la vista principal al crear un nuevo proyecto, presentando las ventanas más relevantes con los canales de sonido y las pistas de cada instrumento o efecto. Mediante un clic sobre el nombre del instrumento aparecerá el editor de onda, si es compatible, y un pequeño teclado de piano con el que comprobar el sonido producido. Seleccionado, por otro lado, la barra de tiempo de un instrumento aparecerá otra ventana con la partitura que seguirá tal instrumento al reproducir la pista. Sobre esta partitura es posible marcar en cada instante las notas y acordes hasta formar la composición deseada.

Existe un sinfín más de funcionalidades propias de cualquier *DAW*, pero las mencionadas serán las mínimamente necesarias para el proyecto, originando algunos

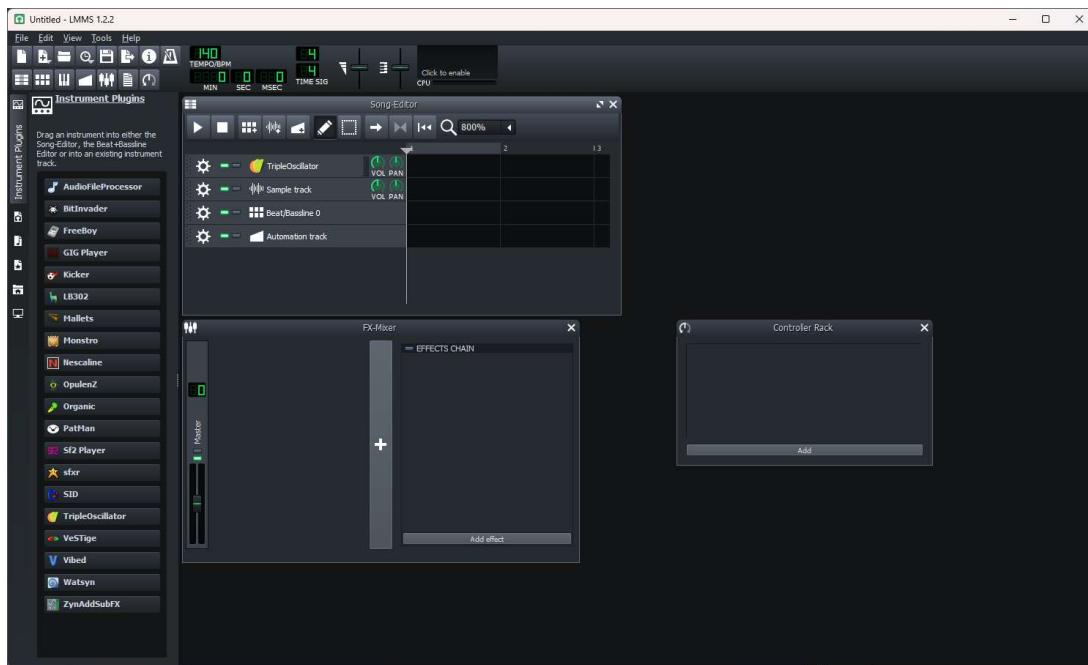


Figura 5.1: Captura de pantalla de la aplicación *LMMS*

efectos de sonido para momentos concretos del juego. Con esto, **queda consolidado el uso de *LMMS*** como estación de audio digital para producción y gestión del aspecto sonoro del proyecto.

5.1.2. Gráficos 3D

Una escena en una aplicación gráfica tridimensional se compone de una serie de modelos que representan objetos diferentes, tratando de simular la realidad. Estos modelos son formados por varias primitivas: vértices (puntos dispersos en el espacio), aristas (que conectan esos puntos) y caras (que rellenan el área entre varias aristas y vértices). Evitando entrar en excesivo detalle sobre el funcionamiento interno de los gráficos por ordenador, la aplicación gráfica se encarga de comunicar al hardware qué objeto debe pintar en pantalla y con qué propiedades, combinando la información dada en el modelo 3D y los materiales asociados al mismo.

Generar estos modelos puede tomar dos puntos de partida distintos: por un lado, recrear un objeto real mediante, por ejemplo, imágenes, o producir un objeto sintético desde la nada, añadiendo manualmente todos sus vértices, aristas y caras. Es aquí donde encaja el rol de escultor 3D en la actualidad, como aquellos especialistas en fabricar modelos 3D mediante software especializado. Este software presenta una escena 3D virtual y multitud de funcionalidades para el manejo de las primitivas mencionadas, así como métodos que asemejan las técnicas de escultura real.

Ya que este tipo de software abarca muchas funcionalidades, es posible encontrar multitud de alternativas de gran calidad, aunque suele percibirse cierta tendencia sobre *Blender* al ser una aplicación de código libre capaz de suplir las necesidades de muchos especialistas. Resulta especialmente relevante que se comercialice de forma gratuita al encontrar otras alternativas con licencias de pago continuo. Por este motivo, y el aprovechamiento de cierta experiencia con su uso, ***Blender* será la herramienta a usar** en cuestiones relativas a la generación de modelos 3D y sus propiedades visuales.

Al inicio, *Blender* resulta extremadamente caótico al presentar casi toda su funcionalidad en una misma vista (ver [Figura 5.2](#)), pero es altamente accesible una vez el usuario se familiariza con aquellas utilidades que desea emplear. En este proyecto, nos limitaremos principalmente a la vista de la escena 3D, la jerarquía de objetos que pueblan la escena y el editor de objetos, que permite gestionar las primitivas anteriores. Ya que los modelos 3D más complejos se obtienen de paquetes de recursos externos, no será necesario, a priori, recurrir a la funcionalidad avanzada de *Blender*.

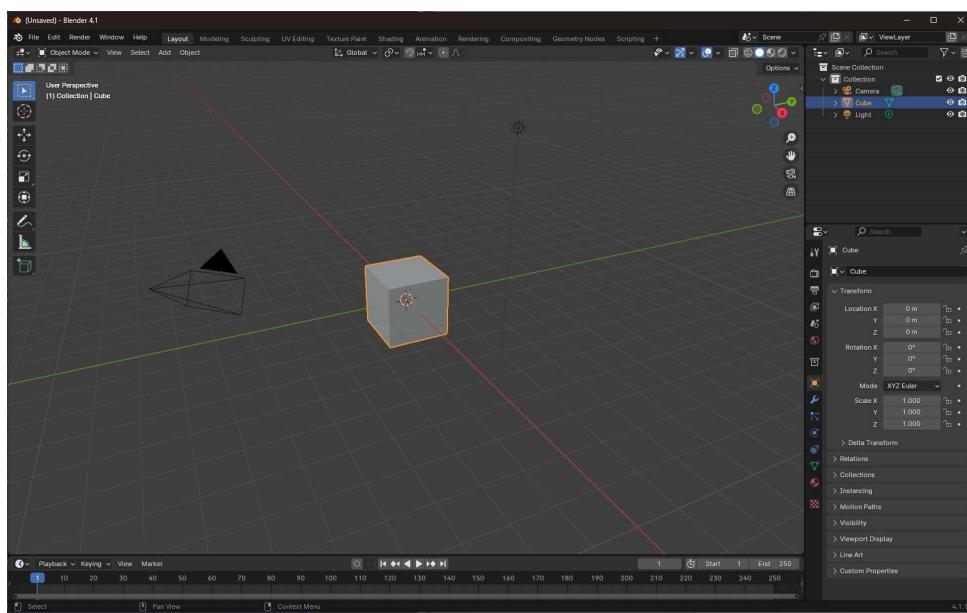


Figura 5.2: Captura de pantalla de la aplicación *Blender*

5.1.3. Gráficos 2D

Pese a que pueda asemejarse al apartado anterior, la producción de gráficos 2D sigue un enfoque muy distinto al 3D. El objetivo de esta producción es generar imágenes 2D a utilizar en multitud de aspectos y aplicaciones, lo que genera una distinción sobre el tipo de imagen a tratar. El tipo más común y que cualquier persona acostumbra a usar es el conocido como imagen raster, una colección de píxeles de colores

ordenados de cierta forma generando la imagen final. Estas imágenes resultan fáciles de manejar y dan resultados de gran calidad a cambio de una mayor ocupación de espacio para su almacenamiento; además, al estar limitada a cierto número de píxeles, surgen problemas al aumentar o reducir su calidad.

En paralelo, las imágenes vectoriales no se componen internamente de píxeles de color: hacen uso de formulaciones matemáticas para definir las figuras que forman la imagen. Pese a ser más complejas de manejar, permiten obviar completamente los problemas de calidad, ya que la imagen final se genera una vez conocidas las dimensiones deseadas. Ambos tipos de imagen presentan utilidad, y, por tanto, aparecen alternativas que permiten su manejo.

Comenzando por imágenes raster, no cabe duda que el editor por excelencia es *Adobe Photoshop* y su incontable repertorio de funcionalidades. El factor que induce al claro descarte de esta herramienta es el requisito de una licencia de uso para la suite de *Adobe*, lo que excede considerablemente los costes estimados para el proyecto. Afortunadamente, *GIMP* surge como software libre y de código abierto con un gran abanico de utilidades, que además queda extendido por los colaboradores y usuarios de la propia aplicación. ***GIMP* será la herramienta a utilizar** en cuanto a imagen raster, ya sea para edición o dibujo.

Respecto a las imágenes vectoriales, la situación es similar, con *Adobe Illustrator* como referente pero bajo una licencia de uso comercial. De igual forma, *Inkscape* aparece como una aplicación con los mismos objetivos, ofertado como software libre y de código abierto. Con esto y la experiencia previa en su uso, ***Inkscape* es la herramienta seleccionada** para las necesidades de dibujo vectorial.

5.1.4. Motor gráfico

Por último, resta elegir la que constituirá la herramienta principal de trabajo para el desarrollo del prototipo. Un motor gráfico es una aplicación destinada al desarrollo de aplicaciones gráficas, generalmente videojuegos, simplificando múltiples procesos comunes como la visualización, gestión de eventos y entrada/salida, carga de recursos externos, etc. Considerando la premisa de utilizar software con licencia de uso gratuita y con el alcance y objetivos del proyecto, aparecen tres alternativas principales para el desarrollo: *Unreal Engine*, *Godot* y *Unity*.

Las tres opciones serán evaluadas en cuanto a funcionalidades, facilidad de uso, calidad visual, y, especialmente, capacidades para el desarrollo de aplicaciones con

Realidad Aumentada. Algo común a estos motores es la presencia de extensiones y/o librerías por parte del equipo tras su desarrollo y de la comunidad que extienden la funcionalidad del motor hacia áreas que no consideraba previamente. Estas extensiones se tendrán en cuenta durante la comparativa al evaluar la capacidad de los motores.

Godot

Pese a ser un motor gráfico con menos impacto que otras alternativas, *Godot* ha comenzado a adquirir fuerza y una posición considerable entre la comunidad de desarrolladores desde el lanzamiento de su cuarta versión. Su principal característica se encuentra en el uso de un lenguaje de programación propio, *GDScript*, integrado en el mismo motor, con una capacidad muy alta de interconexión entre sus componentes internos; mientras otros motores recurren a lenguajes generales, *GDScript* habilita lo mejor de ambos mundos, un control preciso de la información que maneja con un gran rendimiento al estar directamente integrado.

Por otro lado, *Godot* sigue un esquema de funcionamiento **fuertemente basado en componentes**, donde los distintos objetos que componen una escena están, a su vez, subdividido en componentes menores. A esto queda añadido el concepto de escena, que varía un poco en *Godot* respecto a sus alternativas. Cada escena no es más que la unión de una serie de componentes, independientemente de su tipo, agregadas entre sí para formar otras escenas de mayor magnitud. A modo de ejemplo, un objeto que represente al jugador estará compuesto por varias piezas: el modelo 3D con sus texturas y animaciones, un componente de controlador que gestione la entrada/salida para el movimiento y acciones, otro componente con los elementos de interfaz gráfica asociados como la salud o puntos de experiencia y componentes dedicados a la detección de colisiones con el escenario y que limiten el movimiento.

Como ventaja adicional nativa en *Godot*, implementa la “recarga en caliente” (*hot reload*) en la gestión del código, mediante la que cualquier cambio realizado queda directamente aplicado sobre el juego en funcionamiento desde el editor, sin necesidad de detener la ejecución y volver a lanzar el proceso. Esto implica un flujo de trabajo y pruebas mucho más acelerado, pues proyectos de mayor magnitud suelen tomar un tiempo de preparación alto únicamente para lanzar el juego contenido en el editor.

La [Figura 5.3](#) muestra la interfaz gráfica del editor, tras abrir una escena perteneciente a un proyecto¹. En la jerarquía situada a la izquierda puede apreciarse la

¹ *Screequare* es un título desarrollado en paralelo, con un alcance mucho menor, en una única semana. Se encuentra disponible en <https://imjustjc.itch.io/screequare>.

composición de varios elementos agrupados según su función. Por ejemplo, el control de volumen, icono y reproductores se agregan bajo un controlador de audio.

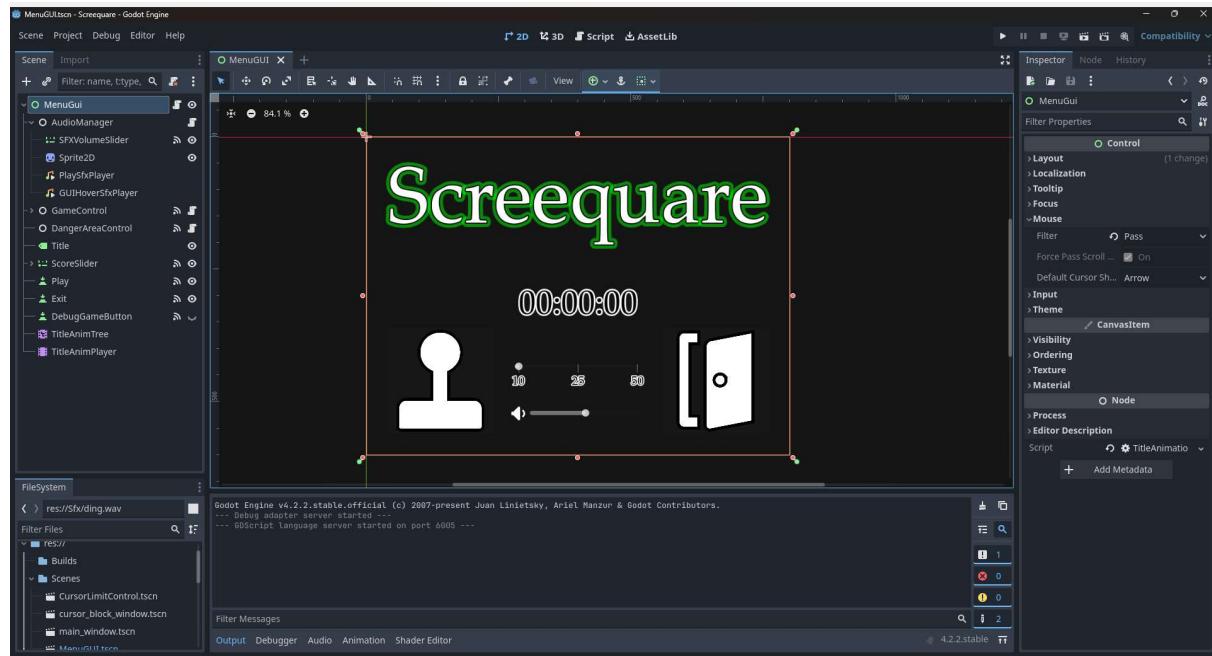


Figura 5.3: Captura de pantalla del motor gráfico *Godot*

En cuanto a extensiones, su naturaleza de código abierto permite a cualquier usuario aportar nueva funcionalidad directamente trabajando con los entresijos del motor, y es posible acceder a una librería con un amplio repertorio de extensiones. Indagando en las necesidades del proyecto, *Godot* no destaca por su compatibilidad con el manejo de la Realidad Virtual, Aumentada ni Mixta; forman un conjunto de funcionalidades que aún continúa en desarrollo².

Pese a ser un fuerte candidato que permite un alto nivel de calidad visual y rendimiento, y disponer de experiencia previa en su uso, ***Godot* queda descartado** al no disponer de tanto desarrollo en el ámbito de la Realidad Aumentada como otras alternativas. Aun así, es una herramienta con mucho potencial y de gran utilidad para casi cualquier proyecto de videojuego, y puede convertirse en el mejor candidato una vez reciba suficientes actualizaciones por parte de su comunidad.

Unreal Engine

De la mano de *Epic Games*, *Unreal Engine* permite alcanzar los mayores niveles de detalle en la industria respecto a cualquier otro motor de uso gratuito. Respaldado

²Extensiones como <https://github.com/GodotVR/godot-xr-tools> se integran poco a poco en la versión principal de *Godot*.

por grandes profesionales de la informática gráfica, el motor cuenta con novedades continuas que buscan empujar los límites de la calidad visual a la par que optimizan notablemente el rendimiento de las aplicaciones. Un ejemplo de esto último viene dado en una actualización reciente del título *Fortnite*, mediante la cual quedaron implementadas técnicas novedosas del motor que mejoraron su rendimiento y calidad visual³.

Aunque cuenta con una barrera de entrada algo compleja para desarrolladores nóveles y hace uso del lenguaje de programación *C++*, incorpora el sistema propio ***Blueprints***, mediante el cual pueden implementarse **scripts** a través de programación visual por bloques: basta con añadir algunos nodos e interconectarlos para obtener una funcionalidad concreta.



Figura 5.4: Sistema *Blueprints* de *Unreal Engine* aplicado en una escena. Extraído de la documentación oficial de *Unreal Engine*

Es posible añadir más funcionalidades al motor mediante su sistema de plug-ins, facilitados por diversos distribuidores. Entre estos, encontramos utilidades de RA, que no se incluyen por defecto en el motor y solventan cuestiones como el desplazamiento del dispositivo, la captura de vídeo e incluso el reconocimiento de imágenes. A pesar de las múltiples ventajas que presenta el motor, ***Unreal Engine* acaba siendo descartado** del proyecto por la falta de experiencia y el soporte limitado para dispositivos *Android*: ciertos aspectos en el uso de sombreadores (*shaders*) más específicos no son compatibles a la hora de compilar y exportar el ejecutable al dispositivo móvil.

Unity

Sin duda alguna, la opción principal año tras año de la inmensa mayoría de desarrolladores es *Unity*. Con el paso del tiempo se ha ganado la confianza de sus usuarios con la inclusión de nuevos sistemas y mejoras sobre los previos, facilitando las labores de desarrollo y permitiendo expandir las posibilidades. Esta sinergia converge en

³<https://www.fortnite.com/news/drop-into-the-next-generation-of-fortnite-battle-royale-powered-by-unreal-engine-5-1>

la existencia de un amplio repertorio de recursos, tutoriales (tanto genéricos como específicos), manuales de programación y ejemplos aptos para casi cualquier proyecto.

Su funcionamiento sigue las mismas ideas que cualquier otro motor, agregando distintos objetos en escenas 3D o 2D. A diferencia de *Godot*, *Unity* **separa mucho más el concepto de escena** en sí mismo, considerándolas como agrupaciones separadas de objetos, similar, por ejemplo, a los niveles que tendría un videojuego: varias escenas pueden compartir objetos, pero no hay realmente un traspaso de información entre escenas. Utiliza *C#* como lenguaje de programación para *scripting*, simplificando ciertas funciones y uso de estructuras de datos y con un enfoque orientado a objetos.

Cuenta, además, con un catálogo de novedades bastante ágil, incorporando métodos que habilitan un mayor nivel de calidad visual y aseguran un rendimiento optimizado. Actualmente, muchas novedades están siendo preparadas para la siguiente gran versión del motor, *Unity 6*. Sin embargo, otro punto clave viene dado por una serie de decisiones empresariales realizadas por la empresa tras el motor gráfico.

A escasos meses del final de 2023, *Unity* anunció un **cambio drástico en sus términos y licencias de uso**. El aspecto que más repercusión tuvo en la comunidad fue la intención de añadir una tasa por descarga a todos los productos que se adaptaran a la nueva licencia, a partir de cierto rango de ingresos y de forma incremental [89]. Aunque esto originó bastante confusión y tomó gran parte de la polémica, la manera en que los responsables de la organización gestionaron la nueva información retorció aún más la situación: se encontró una modificación con la que **desaparecieron los cambios de licencia** seguida de una actualización que **desligaba los términos de la versión usada en el desarrollo con los que aplican al producto final** [90]. Tras calmarse toda la polémica y aclarar la situación, gran parte de los cambios se deshicieron hasta alcanzar un estado más neutro para los extremos, aunque este tipo de situaciones perjudican notablemente la imagen de la marca [91].

Especificando la funcionalidad relativa al proyecto, *Unity* cuenta con soporte para Realidad Virtual, Aumentada y Mixta mediante diversas librerías y módulos de forma nativa. Sobre la Realidad Aumentada, cuenta con abstracciones de las librerías *AR-Core* y *AR Kit* para trabajar tanto en *Android* como *iOS*, pero existe otra alternativa a modo de extensión que abstrae las anteriores: *Vuforia*, una extensión que abstrae todas las utilidades necesarias para entornos de Realidad Aumentada y Mixta.

Vuforia cuenta con la capacidad de mantener el contexto del mundo virtual, realizando el seguimiento de los movimientos del dispositivo, detección de imágenes y objetos, así como establecimiento y gestión del entorno. Con los objetivos del proyecto, la simplificación del uso de marcadores con la detección de imagen solventada

promete ser un buen camino a tomar para el desarrollo, pues permitirá centrarse en la implementación del juego en sí. Además, ya que la detección de imagen se realiza mediante características únicas, el uso de *Vuforia* facilita ajustar los marcadores a cualquier diseño y forma.

Tras explorar *Unity* y los motores gráficos anteriores, **se decide desarrollar el proyecto con *Unity* y la extensión *Vuforia***, dadas las facilidades que aportarán sobre las necesidades del proyecto y la experiencia previa en su uso. A modo de resumen, la [Figura 5.5](#) muestra los logotipos de las distintas herramientas software utilizadas para el desarrollo del proyecto.



Figura 5.5: Herramientas software utilizadas durante el desarrollo del proyecto

5.2. Incremento 1. Puesta en marcha y marcadores de Realidad Aumentada

Una vez definidos los contenidos del juego y seleccionadas las herramientas para el desarrollo, este primer incremento supone el inicio del prototipo como proyecto en *Unity*, y solventará el sistema de gestión de versiones utilizado, así como la puesta en marcha del entorno de Realidad Aumentada y los marcadores a utilizar para el prototipo desarrollado.

5.2.1. Proyecto en *Unity* y control de versiones

Empezando por la generación del proyecto, *Unity* facilita un pequeño ayudante capaz de gestionar instalaciones, proyectos y manuales de referencia antes de lanzar el editor, *Unity Hub*, mediante el cual es posible generar un nuevo proyecto a partir de plantillas, especificando unos pocos parámetros. La [Figura 5.6](#) presenta este ayudante con la selección de la plantilla 3D y los parámetros necesarios, como nombre del proyecto, ubicación en disco y opciones de conexión a la nube y control de versiones.

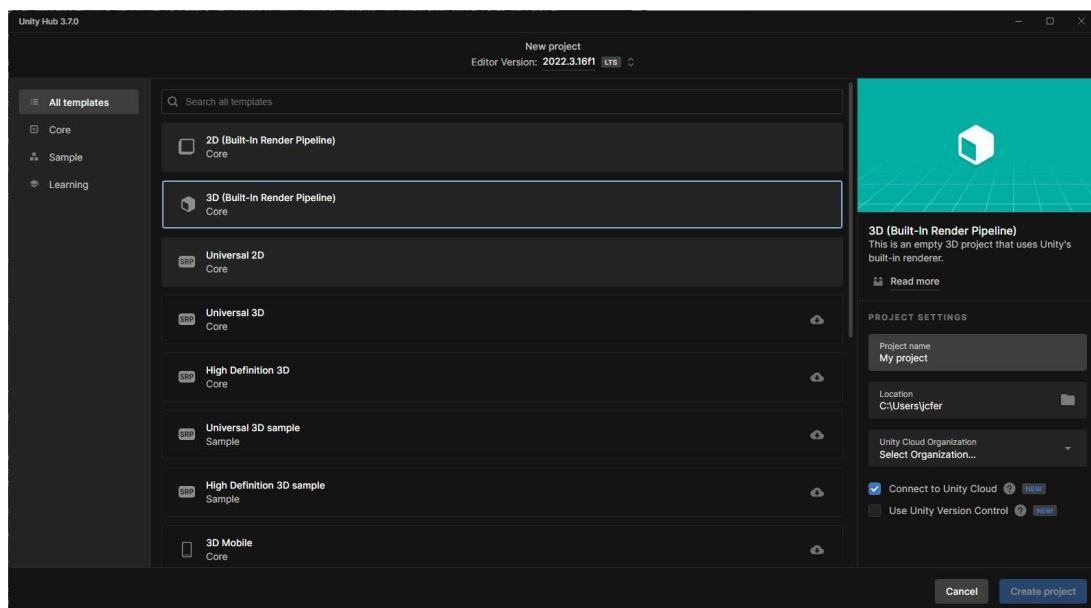


Figura 5.6: Ventana de creación de un nuevo proyecto en Unity Hub

Unity cuenta con un sistema de control de versiones propio, *Unity Version Control*, integrado en el propio motor, que facilita compartir el proyecto entre distintos desarrolladores y equipos de trabajo. Como cualquier sistema de este tipo, su funcionamiento se basa en la agrupación de cambios (comúnmente esta agrupación recibe el nombre de *commit*) y la subida de tales cambios a un servidor remoto de sincronización. Al detectar cambios en el servidor remoto, el sistema informa al usuario y permite la actualización de sus ficheros locales, gestionando también cualquier conflicto entre la descarga y las modificaciones realizadas localmente (en casos más complejos, es tarea del usuario resolver estos conflictos).

Si bien el proyecto no sigue una metodología *Scrum* que define una revisión diaria de los avances, el uso de un sistema de control de versiones facilita esta revisión al mantener el servidor remoto actualizado con los avances diarios o, a lo sumo, cada dos o tres días. Aunque no se comparta el desarrollo con un amplio grupo de desarrolladores, mantener **una copia remota de los avances** aporta cierta seguridad ante cualquier catástrofe que implique la pérdida o corrupción del proyecto localmente.

Al generar el proyecto a partir de la plantilla 3D, solo se incorporan las librerías y extensiones esenciales por el motor. *Unity* cuenta con un gran repertorio de paquetes oficiales no incluidos por defecto y que simplificarán diversas partes del desarrollo; la Figura 5.7 expone el listado con todos los paquetes incorporados al proyecto, su versión, y agrupados por autor.

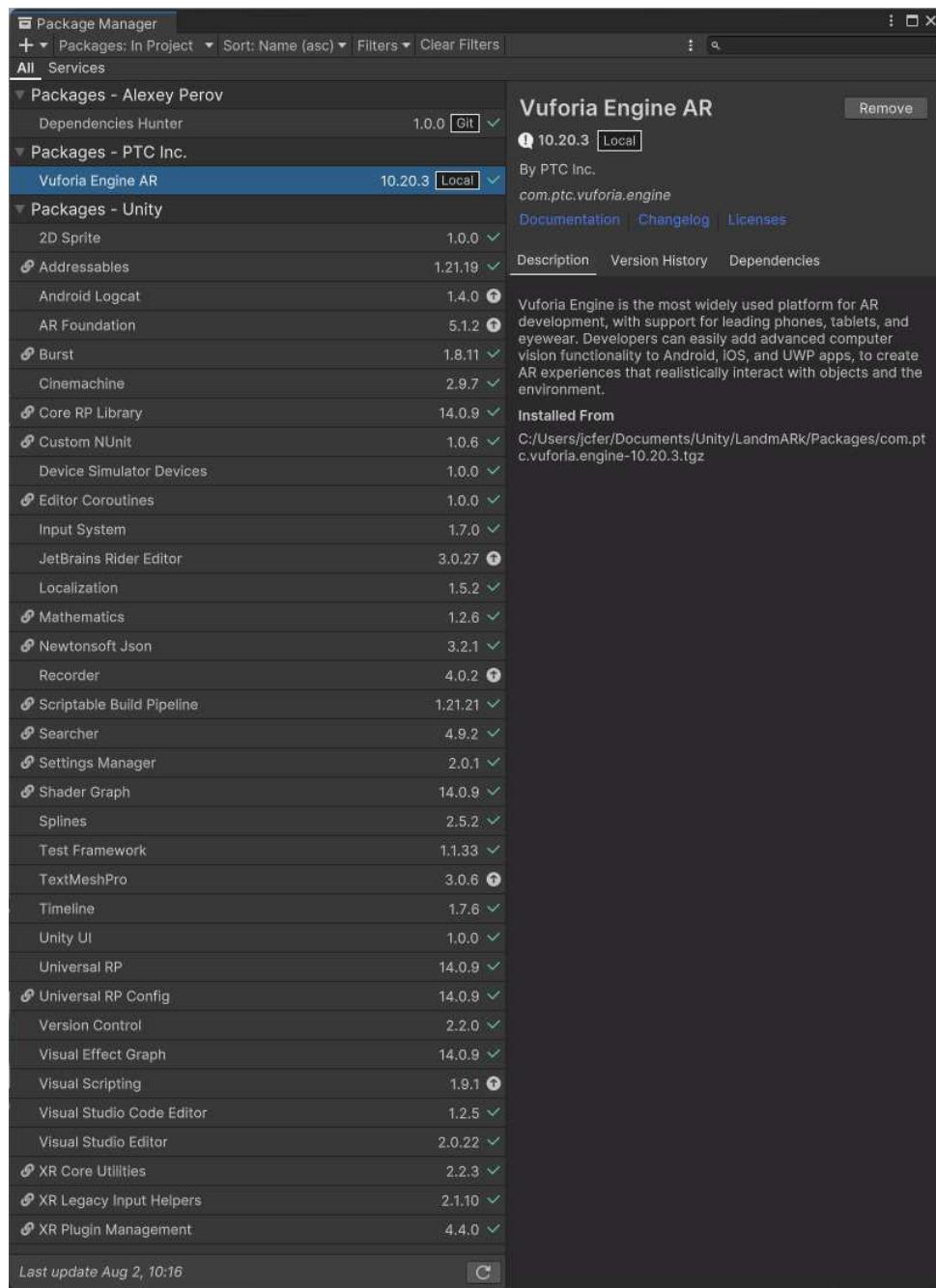


Figura 5.7: Listado de paquetes del proyecto de *Unity* utilizados

Entre todos los paquetes anteriores, gran parte de ellos se incorporan automáticamente junto a la plantilla seleccionada al generar el proyecto, pero otros se añaden manualmente desde un enlace a repositorio remoto o desde un fichero de paquete

Unity. Estos paquetes adicionales se mencionan a continuación, explicando brevemente su utilidad:

- **Dependencies Hunter**: Permite generar un listado las referencias de cada recurso, resaltando aquellos no utilizados en el proyecto para facilitar su limpieza.
- **Vuforia Engine AR**: Paquete con la extensión *Vuforia* para el entorno de Realidad Aumentada, se detallará en el siguiente apartado.
- **AR Foundation**: Agrupa las extensiones y librerías comunes de Realidad Aumentada para las distintas plataformas y dispositivos compatibles.
- **Localization**: Sistema de localización de recursos capaz de gestionar modificaciones entre zonas, como textos, dimensiones, iconos e incluso modelos 3D.
- **Recorder**: Herramienta de captura y grabación integrada en el motor gráfico, permite extraer capturas y vídeos de la aplicación desde el propio editor.
- **Shader Graph**: Herramienta de creación de sombreadores mediante programación visual por bloques.
- **Universal RP**: Canal universal de renderizado o *URP* altera la generación de los gráficos interna para dar soporte a un gran repertorio de plataformas con un alto nivel de calidad visual.

5.2.2. Entorno de Realidad Aumentada con *Vuforia*

Vuforia cuenta con una amplia documentación y ejemplos de uso disponibles en su web oficial⁴, destacando un tutorial para su puesta en marcha como parte de un proyecto en *Unity*. Basta con añadir el paquete al proyecto para encontrar el contenido que ofrece *Vuforia*, comenzando por la cámara virtual para Realidad Aumentada. Antes de comenzar a usar *Vuforia*, deben establecerse ciertos parámetros y ajustes en la configuración de la extensión, priorizando la **clave de licencia para su uso**. Esta licencia puede obtenerse de forma gratuita al registrarse en su web oficial, y cuenta con todas las funcionalidades disponibles de forma local, restringiendo el acceso únicamente a servicios en la nube. La configuración también permite establecer ciertos valores como el escalado de la escena virtual, el modo de enfoque de la cámara del dispositivo (automático o manual) o el número de imágenes simultáneas.

⁴<https://developer.vuforia.com/library/>

Después de configurar las opciones de *Vuforia*, preparar una escena básica para Realidad Aumentada resulta bastante sencillo: basta con insertar un objeto de cámara virtual de Realidad Aumentada (*AR Camera*) y objetos con imágenes como objetivo (*Image Target*). Al iniciar la ejecución, *Vuforia* se encargará de inicializar el entorno, del seguimiento del dispositivo, y la detección de las imágenes asignadas. Asociando otros objetos como descendientes de cada imagen en la jerarquía (ver [Figura 5.8](#)), serán posicionados en el mundo en la posición en que se detecte dicha imagen.

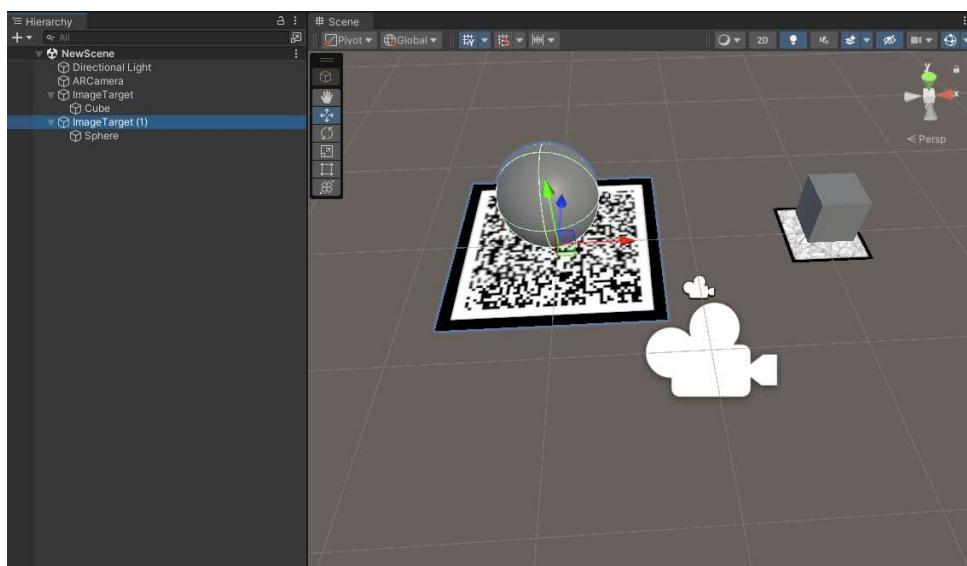


Figura 5.8: Jerarquía de escena en Unity con los objetos de Vuforia necesarios

En este punto surge una necesidad derivada de la plataforma objetivo para el prototipo. **Al distribuirse en dispositivos móviles, es necesario hacer las pruebas de funcionamiento sobre el mismo**, pero encontramos dos obstáculos principales. En primer lugar, y más sencillo de solventar, las dimensiones de la pantalla del dispositivo afectan directamente a los contenidos que pueden mostrarse, pero *Unity* permite redimensionar su simulador de ejecución a las dimensiones deseadas. El dispositivo móvil cuenta con una pantalla de 2400 píxeles de altura por 1080 píxeles de anchura en formato de retrato, que será la resolución a utilizar como base para el desarrollo.

Como segundo obstáculo, el uso de la Realidad Aumentada requiere de la cámara incorporada en el dispositivo y los sensores internos para su posicionamiento en el mundo virtual; una primera opción sería generar el fichero con la aplicación móvil (en *Android* estos ficheros son del tipo *APK* o *Android Package*, denominación que se utilizará en menciones futuras al mismo fichero), transferirlo al dispositivo, instalarlo y ejecutarlo. Claramente, este flujo de trabajo es excesivamente complejo para las necesidades de depuración de un proyecto software. Afortunadamente, ***Vuforia* ofrece una alternativa directamente desde el editor de *Unity***, simulando el dispositivo mediante una cámara web conectada al equipo.

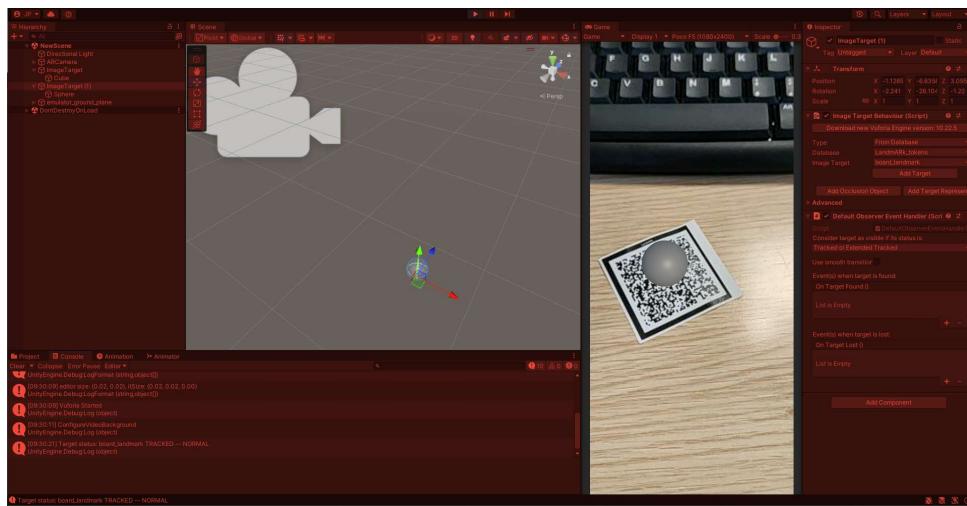


Figura 5.9: Captura de pantalla del editor Unity ejecutando el simulador de Vuforia

Para comenzar el uso de este simulador, se hace uso del propio dispositivo móvil como cámara web mediante la aplicación *DroidCam*. De esta forma, es posible controlar la cámara virtual con el dispositivo móvil en mano, mientras que el juego se ejecuta directamente en *Unity*, sin necesidad de exportar el proyecto. Un ejemplo de este funcionamiento viene dado en la [Figura 5.9](#), donde se ha detectado el marcador de prueba y se han posicionado los objetos asociados en la jerarquía.

5.2.3. Diseño de marcadores de Realidad Aumentada

El último avance contenido en este incremento será el diseño de las imágenes utilizadas como marcadores para el juego. Como se mencionó en la sección del incremento anterior, *Vuforia* realiza la detección de imágenes a partir de una serie de características únicas propias de la imagen, comúnmente desde los puntos que forman esquinas o ángulos cerrados. Desde su página web, *Vuforia* ofrece un servicio de creación de una pequeña base de datos de imágenes, capaz de extraer dichas características y evaluar la validez de la misma como marcador. Siguiendo lo establecido en el [Capítulo 2](#), los marcadores tendrán unas dimensiones físicas de 25 milímetros cuadrados, algo a tener en cuenta al considerar que deben ser captados por la cámara del dispositivo.

Con un tamaño tan reducido, debe encontrarse un **equilibrio en la cantidad de características detectables** de la imagen, pues un número reducido supondrá errores en su detección, mientras que el propio tamaño induce una limitación en esa cantidad de características manteniendo suficiente calidad visual. La mayoría de aplicaciones que recurren a la Realidad Aumentada hacen uso de marcadores especializados, da-

dos por librerías o herramientas como *ARUco* o *AprilTag*. Este tipo de marcadores no ofrecen buenos resultados con *Vuforia* al contar con muy pocas características generales (están diseñados para sistemas especializados), por lo que se harán pruebas de funcionamiento con marcadores generados en las herramientas en línea *ARMarker*⁵ y *AR-Marker-Generator*⁶, que producen imágenes con un alto número de características. Los cuatro tipos de marcador valorados se presentan en la [Figura 5.10](#).

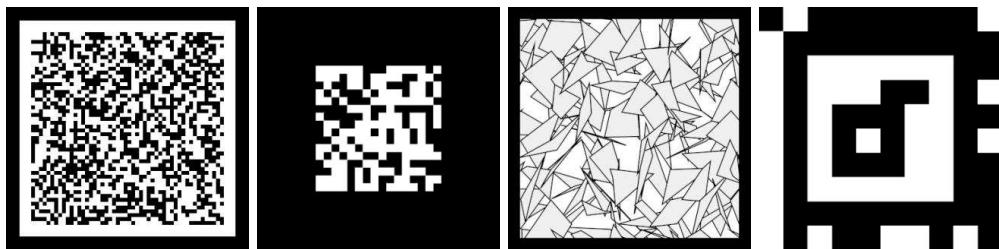


Figura 5.10: Diseños de marcador evaluados durante el desarrollo

La evaluación de cada marcador consiste en subir la imagen al servicio en línea de *Vuforia*, asignar las dimensiones apropiadas, y comprobar la medida de validez dada por el sistema. Salvo los marcadores *AprilTag*, el resto reciben la máxima valoración posible del sistema, implicando que son buenos candidatos para el uso en la aplicación. Pese a esto, **se decide utilizar el primer diseño que muestra la Figura 5.10**, como una cuadrícula extensa en toda la imagen, pues debe también aplicarse un elemento diferenciador que facilite a los jugadores controlar los marcadores. Este elemento será el ícono de la ficha que representa cada marcador, ocupando un espacio en el centro de la imagen junto a un número identificativo del índice asociado.

De los seis tipos de ficha planteados para *LandmARk*, cuatro de los iconos se obtienen del paquete *Kenney - Board Game Icons*, los correspondientes a las fichas de Ayuntamiento, Cuartel, Milicia y Caballería, mientras que las dos fichas restantes, Recolector y Asedio, se diseña un ícono propio tratando de mantener el mismo estilo visual mediante dibujo vectorial en *Inkscape*. El resultado final del conjunto de marcadores a utilizar se presenta en la [Figura 5.11](#), que será impreso en papel a color y recortado para su uso a lo largo del proyecto.

Al generar una base de datos de marcadores en la página web de *Vuforia*, puede descargarse e incorporarse a *Unity* como un paquete de expansión (puede añadirse simplemente con un doble clic mientras el proyecto se encuentra abierto en el editor), que se encarga de vincular las imágenes a las estructuras internas de *Vuforia* para su asignación directa desde cada objeto *Image Target*.

⁵<https://shawnlehner.github.io/ARMarker/>

⁶<https://danilogr.github.io/AR-Marker-Generator/>

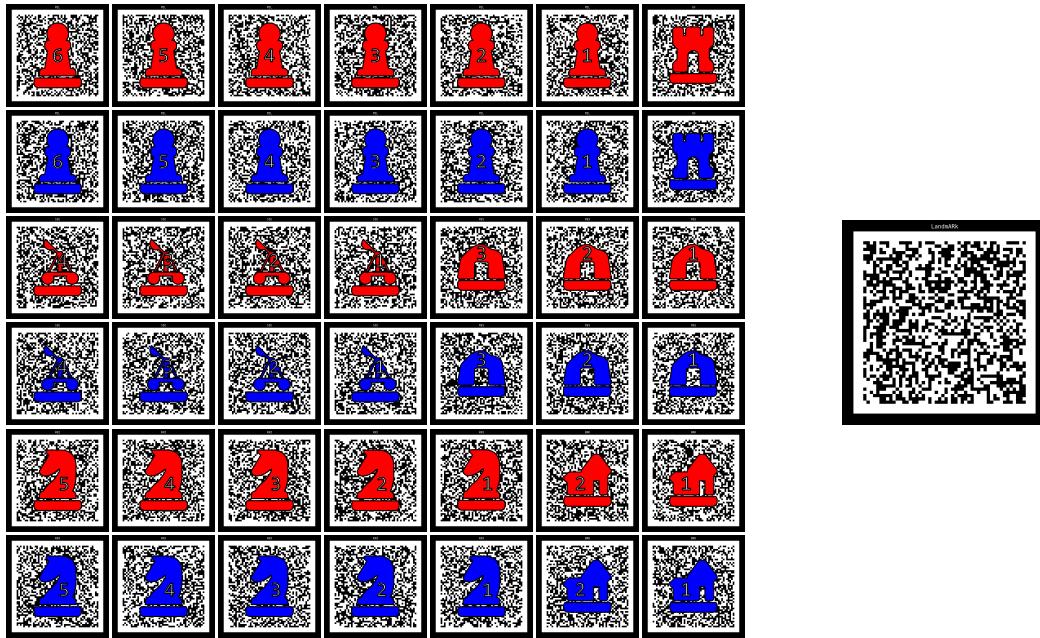


Figura 5.11: Marcadores de *LandmARK* para Realidad Aumentada

5.2.4. Resultados

Al finalizar este incremento, se realiza la generación de un primer ejecutable para dispositivos Android con objeto de comprobar el funcionamiento del entorno de Realidad Aumentada, habilitando la detección de marcadores y el posicionamiento de modelos 3D sobre ellos. En este punto, la aplicación es capaz de esta funcionalidad con una precisión suficiente, lo cual habilita la continuación del proyecto siguiendo la planificación inicial.

La [Figura 5.12](#) muestra varias capturas de pantalla realizadas desde el dispositivo con la aplicación en funcionamiento, donde pueden apreciarse los marcadores con el modelo asociado correctamente posicionados en el espacio. **Cada marcador identificado cuenta con un seguimiento independiente del resto;** de una captura a otra se ha desplazado uno de los marcadores y el objeto asociado queda también recolocado en el mundo virtual manteniendo su posición sobre el propio marcador. En próximas iteraciones, este seguimiento se utilizará como punto de posicionamiento sobre el tablero, colocando las fichas asociadas a cada marcador según la celda en que se coloque.

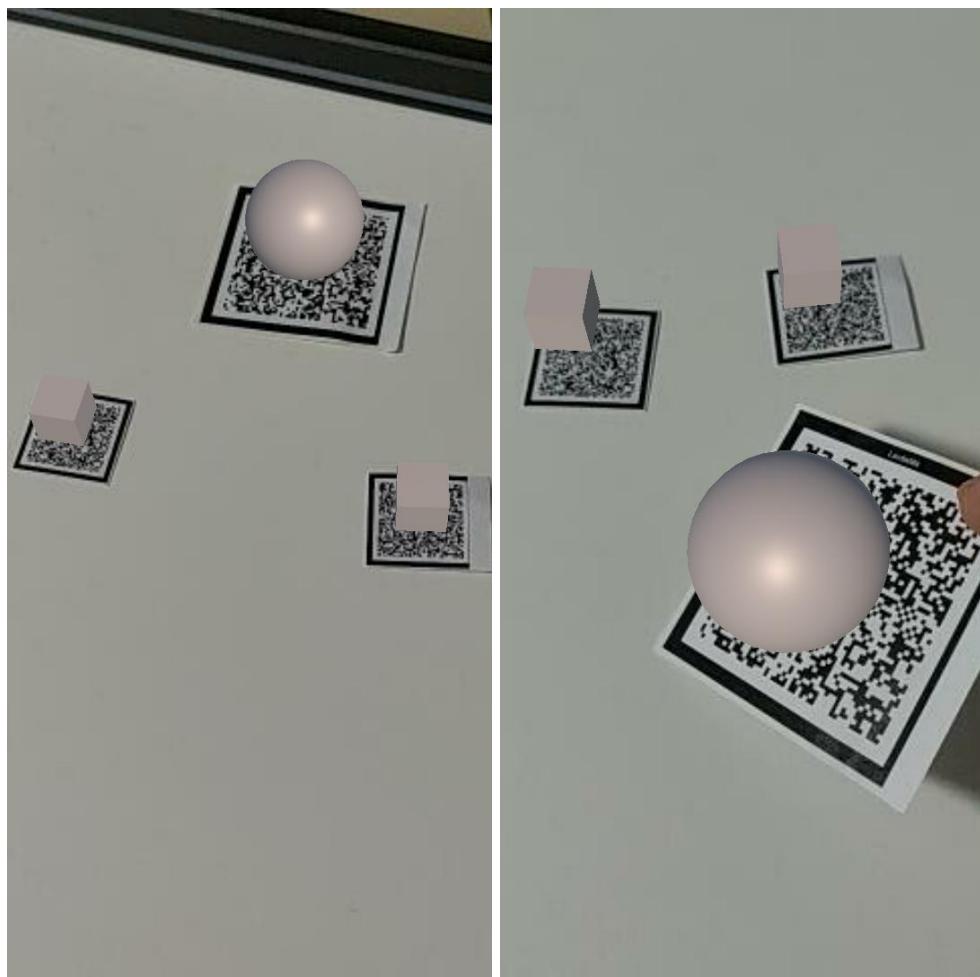


Figura 5.12: Capturas de pantalla de la aplicación tras el primer incremento

5.3. Incremento 2. Sistema de turnos y tablero

Partiendo del proyecto y el entorno de Realidad Aumentada funcionando correctamente, las primeras mecánicas y reglas a implementar del juego será el tablero sobre el que se desarrollará la partida y el sistema de turnos que alterne el jugador activo. A la finalización de este incremento, el prototipo permitirá colocar el tablero en torno a un marcador principal y alternar el control sobre un objeto de muestra desplazable sobre el mismo tablero entre turnos.

5.3.1. Sistema de turnos

La decisión de implementar primero el sistema de turnos surge de su simplicidad y la preparación de un modelo apropiado para el desarrollo del tablero, ya que **en este punto del desarrollo aún se debatían cuestiones básicas** del mismo: número

de celdas, formato (cuadrícula o hexagonal) y posiciones iniciales de cada jugador. En este punto, además, comienza el uso de *scripts* para controlar el funcionamiento del juego y sus componentes; aquellos *scripts* más específicos y de menor longitud quedarán incorporados al documento y explicados, mientras que otros serán mencionados de una forma más genérica, detallando los atributos y métodos más relevantes (omitiendo, por ejemplo, variables de control, contadores, o métodos de consulta) así como comportamientos clave mediante pseudocódigo.

Un aspecto básico del desarrollo con *Unity* es la definición de clases, pues es la base sobre la que opera el lenguaje de programación *C#*. Cualquier *script* escrito definirá una clase y, generalmente, heredará de *MonoBehaviour* para permitir su vinculación a un objeto de la escena. Todas las clases comparten el mismo espacio de nombres y son accesibles desde cualquier otra clase por defecto, lo que habilita, en ciertas situaciones, el uso de clases generales con un comportamiento global que no quiera asociarse a un objeto de la escena activa. Para facilitar la asignación de valores a los atributos de una clase, la etiqueta “[*SerializeField*]” indica a *Unity* que el atributo debe serializarse y manejarse directamente desde el propio editor, pudiendo asignar su valor sin necesidad de modificar el fichero.

El sistema de turnos parte de una premisa básica: **solo habrá dos jugadores en la partida**. Por tanto, el turno comenzará en un jugador y pasará al adversario una vez el mismo jugador haya decidido finalizar su turno. El *script* *PlayerTurnManager.cs* presentado en el [Listado 5.1](#) contiene la clase del mismo nombre encargada de mantener el jugador activo y notificar cuando se produzca un cambio de turno. Esta notificación será enviada mediante el patrón *Publicador-Suscriptor*. En *C#*, es posible utilizar este patrón mediante el uso un atributo de evento y funciones delegadas (líneas 3 y 4 del listado). En el método de cambio de turno, *PlayerTurnManager::NextPlayer()*, el atributo de evento se invoca con el parámetro asociado, llamando a todas las funciones que se hayan suscrito al mismo.

```
1 public class PlayerTurnManager : MonoBehaviour {
2     public delegate void TurnChanged(int playerTurn);
3     public static event TurnChanged TurnChangeEvent;
4
5     [SerializeField]
6     protected int MaxPlayers = 2;
7     public static int PlayerTurn = 0;
8
9     public void StartTurns() {
10         PlayerTurn = 0;
11     }
```

```

12     public void NextPlayer() {
13         PlayerTurn = (PlayerTurn + 1) % MaxPlayers;
14         TurnChangeEvent?.Invoke(PlayerTurn);
15     }
16 }
```

Listado 5.1: Clase PlayerTurnManager para el control de turnos

Aunque no quede implementado en su totalidad el tablero de juego, se prepara un pequeño escenario de prueba dividido en celdas individuales a modo de tablero, que permita colocar un objeto para cada jugador y **desplazarlo mediante movimientos de una celda a otra adyacente**. Este tablero de prueba simplemente consta de un conjunto de planos alineados entre sí, y con un color plano alternando de una celda a otra. Los objetos desplazables contarán con un comportamiento definido por un *script UnitMovement.cs* que asigne el **identificador numérico del jugador al que pertenecen** y con unos marcadores de validez del movimiento. Estos marcadores serán modelos 3D colocados sobre las celdas adyacentes ortogonalmente, y tendrán un color representando movimientos válidos (verde) y no válidos (rojo). Esta validez viene dada por el posicionamiento dentro de los límites del tablero, y su actualización cambia el material asociado a la malla de triángulos del modelo.

Por último, los mismos objetos desplazables contarán con otro *script* más, dado en el [Listado 5.2](#), que defina el comportamiento al ser seleccionados por el puntero (ya que esta parte será cambiada en el futuro, aún no se considera el uso de pantalla táctil). El *script* hace uso de la vinculación de acciones mediante el paquete *Input System*⁷ y lanzamiento de rayos mediante el sistema de físicas: comprobando el impacto de **un rayo desde la posición del puntero y con dirección igual a la vista de la cámara**, se harán dos pruebas de colisión según capas de físicas distintas, una exclusiva para el objeto a desplazar y otra para los marcadores de posición objetivo.

Si el objeto intersectado es el propio objeto a desplazar, se actualizarán sus marcadores de movimiento indicando las posiciones válidas para desplazarse. Cuando se intersecte a un marcador de movimiento, si es válido, el objeto será desplazado y cambiará el turno al otro jugador. En ambos casos, solo se considerará la intersección con objetos pertenecientes al jugador activo.

```

1 public class PointerUnitManager : MonoBehaviour {
2     protected PlayerTurnManager turnManager;
3     [SerializeField] protected LayerMask unitLayerMask;
4     [SerializeField] protected LayerMask moveLayerMask;
```

⁷<https://docs.unity3d.com/Packages/com.unity.inputsystem@1.10/manual/index.html>

```

5   protected UnitMovement selectedUnit;
6   protected Vector2 cursorPosition = Vector2.zero;
7   private void Awake() {
8       turnManager = FindFirstObjectByType<PlayerTurnManager>();
9   }
10
11  public void OnPointerMove(InputValue inputValue) { cursorPosition = inputValue.Get
12 <Vector2>(); }
13
14  public void OnPointerClick(InputValue inputValue) {
15      Ray clickRay = Camera.main.ScreenPointToRay(cursorPosition);
16      float rayDist = Camera.main.farClipPlane;
17      RaycastHit hitInfo;
18      if (Physics.Raycast(clickRay, out hitInfo, rayDist, unitLayerMask)) {
19          if (hitInfo.collider.TryGetComponent(out targetUnitMovement)) {
20              if (turnManager.playerTurn == targetUnitMovement.GetOwnerPlayer()) {
21                  if (selectedUnit != null)
22                      selectedUnit.SetMoveProjectionsVisible(false);
23                  targetUnitMovement.UpdateMoveProjections();
24                  targetUnitMovement.SetMoveProjectionsVisible(true);
25                  selectedUnit = targetUnitMovement;
26              }
27          } else if (Physics.Raycast(clickRay, out hitInfo, rayDist, moveLayerMask)) {
28              Transform targetObject = hitInfo.collider.transform;
29              Transform targetUnit = targetObject.transform.parent.parent;
30              UnitMovement targetUnitMovement = targetUnit.GetComponent<UnitMovement>();
31              if (turnManager.playerTurn == targetUnitMovement.GetOwnerPlayer()) {
32                  targetUnit.position = new Vector3(targetObject.position.x, targetUnit.
33 position.y, targetObject.position.z);
34                  targetUnitMovement.SetMoveProjectionsVisible(false);
35                  turnManager.NextPlayer();
36              }
37              if (selectedUnit != null) {
38                  selectedUnit.SetMoveProjectionsVisible(false);
39                  selectedUnit = null;
40              }
41          } else {
42              if (selectedUnit != null) {
43                  selectedUnit.SetMoveProjectionsVisible(false);
44                  selectedUnit = null;
45              }
46          }
47      }
}

```

Listado 5.2: Clase PointerUnitManager para la interacción con objetos desplazables

Montando la escena básica con el tablero temporal, dos modelos 3D desplazables y un objeto para el control de turnos, el funcionamiento sigue el comportamiento esperado, mostrando los marcadores de posición únicamente al seleccionar el objeto del jugador activo y cambiando su posición al seleccionar uno de esos marcadores. La [Figura 5.13](#) resume este comportamiento en el que primero se desplaza el objeto azul, posteriormente el objeto rojo y acaba volviendo el turno al primer jugador, completando el ciclo de turnos.

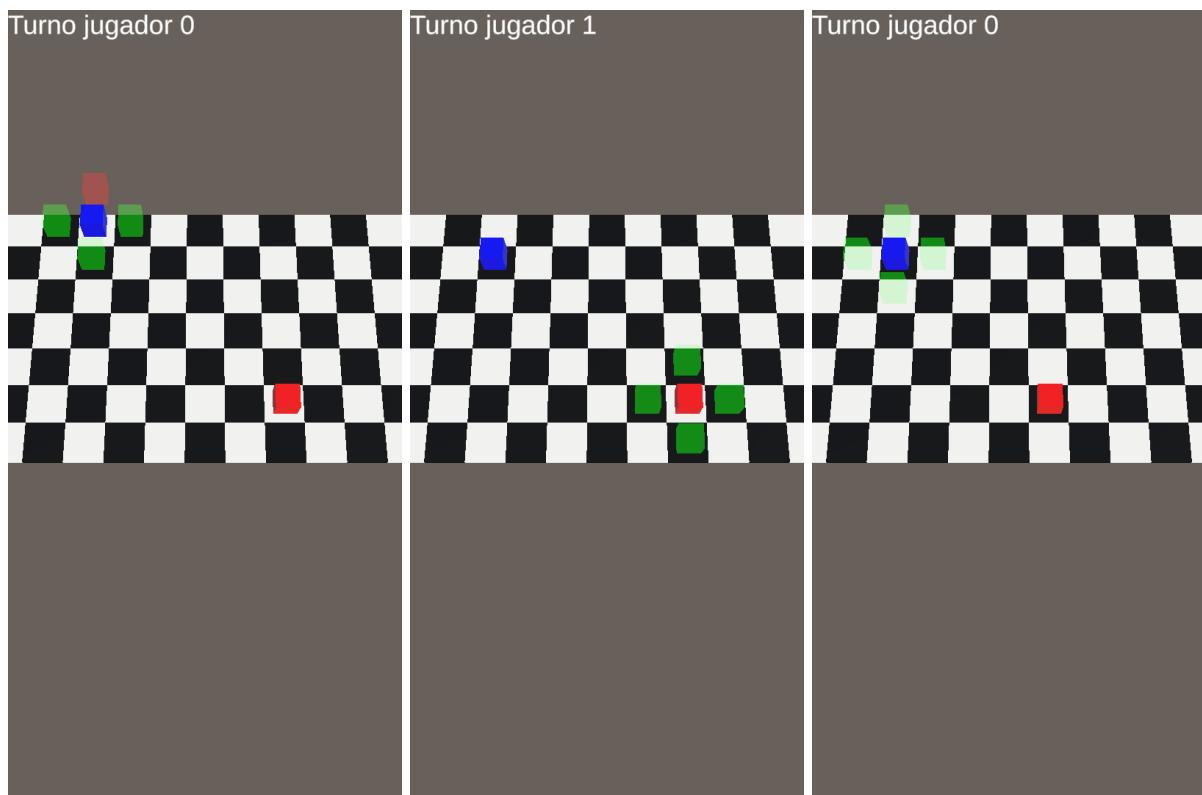


Figura 5.13: Secuencia básica de movimiento entre turnos alternados

5.3.2. Tablero de juego

Con el sistema de turnos establecido y en correcto funcionamiento, el siguiente paso consiste en definir cómo estará formado el tablero de juego. Al inicio, se planteó la posibilidad de generar las celdas de forma dinámica, basándose en unas dimensiones predefinidas, que permitirían al jugador decidir iniciar una partida en tableros pequeños o grandes según la duración deseada de juego. Esta idea quedó rápidamente descartada al tratar de mantener un equilibrio en el juego para un mayor control sobre la duración de la partida y las propiedades de cada ficha: por ejemplo, en tableros grandes, aquellas fichas con movimientos de más alcance aportarían mucho más beneficio que el resto.

La decisión final consiste en establecer **una a una las celdas** que componen el tablero, con unas dimensiones de 9 celdas de anchura y 6 de altura, colocadas horizontalmente sobre el plano ajustando a cada jugador en un extremo opuesto. Una consideración esencial durante el uso de un entorno de Realidad Aumentada es permitir al usuario **visualizar aquello que captura la cámara**, pues será el método básico para correlacionar la posición en el mundo virtual con el real. Ante esto, se prepara un modelo 3D en *Blender* como un plano hueco con bordes de color negro, permitiendo visualizar a través de él mismo, ya que el posicionamiento de cada ficha se realizará con un marcador en el entorno real del jugador.

Pese a resultar ciertamente tedioso, asignar cada celda manualmente permite un mayor control posterior y facilita el acceso a cada celda de forma individual. Además, la incorporación al entorno de Realidad Aumentada simplemente requiere colocar la jerarquía como descendiente del objeto *Image Target* asociado al marcador principal, como muestra la [Figura 5.14⁸](#).

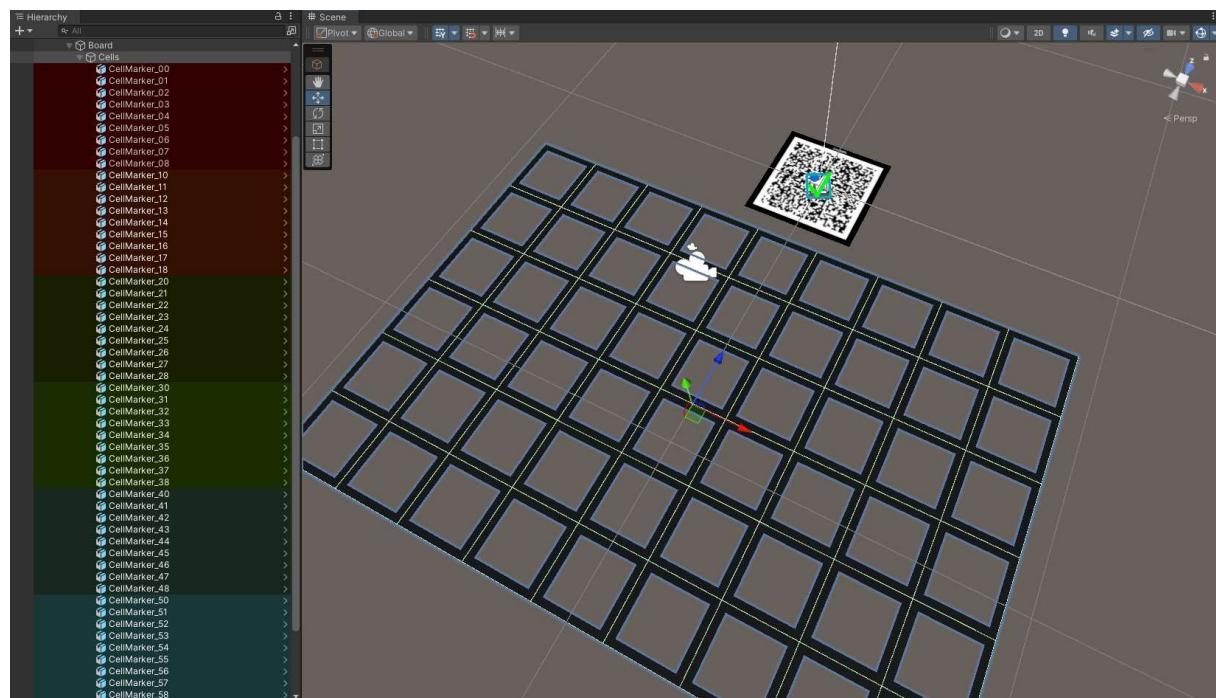


Figura 5.14: Jerarquía de escena para el tablero de juego

Cada una de las celdas colocadas en la escena tendrán también asociadas una instancia de la clase *BoardCell* detallada en el [Listado 5.3](#), conteniendo su posición como índice para los ejes vertical y horizontal. Estos mismos índices forman parte del nombre asignado a la celda, que simplificará la búsqueda del objeto en la escena a partir de su posición. Esta clase, además, implementa la interfaz interna *IEquatable*⁹

⁸El recurso [Pretty Objects](#) permite decorar los elementos de la jerarquía del editor

⁹<https://learn.microsoft.com/en-us/dotnet/api/system.iequatable-1?view=net-8.0>

asociada con ella misma, sobreescribiendo la comparación de igualdad a la comparación entre sus índices de posición, considerando iguales dos celdas que se encuentren la misma posición.

```

1 public class BoardCell : MonoBehaviour, IEquatable<BoardCell> {
2     [SerializeField] protected int X;
3     [SerializeField] protected int Y;
4
5     public Vector2Int GetCellIndexVector() { return new Vector2Int(X, Y); }
6
7     public Vector2Int GetVectorToOther(BoardCell o) {
8         return new Vector2Int(o.X - X, o.Y - Y);
9     }
10
11    public bool Equals(BoardCell other) {
12        if (ReferenceEquals(null, other)) return false;
13        if (ReferenceEquals(this, other)) return true;
14        return base.Equals(other) && X == other.X && Y == other.Y;
15    }
16
17    public override bool Equals(object obj) {
18        if (ReferenceEquals(null, obj)) return false;
19        if (ReferenceEquals(this, obj)) return true;
20        if (obj.GetType() != this.GetType()) return false;
21        return Equals((BoardCell)obj);
22    }
23
24    public override int GetHashCode() {
25        return HashCode.Combine(base.GetHashCode(), X, Y);
26    }
27 }
```

Listado 5.3: Clase *BoardCell* para el control de celdas del tablero

5.3.3. Resultados

Este incremento no produce una versión ejecutable del prototipo, pero prepara el entorno para la inclusión del resto de mecánicas esenciales. El sistema de turnos se mantendrá sin cambios considerables durante los siguientes incrementos, pero el tablero temporal con el que se comprobó el funcionamiento de los turnos queda sustituido por el tablero compuesto por las celdas personalizadas, que hacen uso de la clase *BoardCell* y cuentan con un modelo 3D generado en *Blender*.

Al establecer el tablero como descendiente del marcador principal de *LandmARK*, la aplicación coloca las celdas una vez se haya detectado el mismo, **y seguirá los movimientos que realice el propio marcador**, permitiendo que el jugador desplace el entorno según sus necesidades, representado en la [Figura 5.15](#).

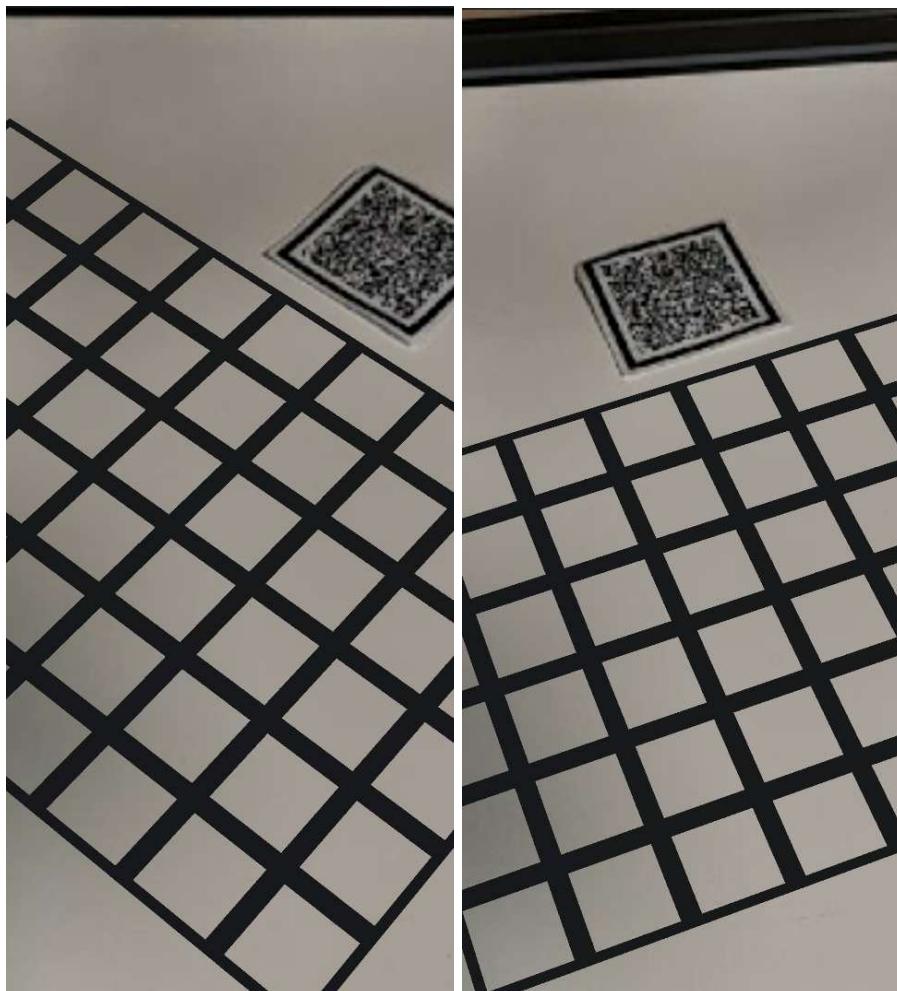


Figura 5.15: Posicionamiento del tablero de juego a partir del marcador principal

5.4. Incremento 3. Interacción y fichas de juego

El siguiente incremento tratará el contenido principal de juego, implementando el resto de mecánicas planteadas y estableciendo la jugabilidad sobre el tablero mediante Realidad Aumentada. Además, quedarán especificadas las distintas fichas ideadas con comportamientos individuales, tratando de mantener un sistema escalable que habilite la continuidad del proyecto. Cuando finalice este incremento, el prototipo contará con un flujo básico de juego, donde pueden moverse fichas y utilizar sus acciones específicas en el turno de cada jugador.

5.4.1. Tipos de fichas de juego

El prototipo de *LandmARK* cuenta con múltiples fichas de juego, que toman unas propiedades y comportamiento diferentes entre sí. Ya que este es uno de los principales puntos de extensión en cuanto a continuidad del proyecto, la implementación de estas fichas debe ser genérica y adaptable a nuevas ideas y mecánicas.

Ante esto, queda construida una **jerarquía básica de clases**: *Token*, *Building* y *Unit*. Su implementación aprovecha el concepto de **herencia de la programación orientada a objetos** tal como muestra la [Figura 5.16](#), estableciendo como padre común la clase *Token* que contendrá las propiedades comunes a todas las fichas; el resto de clases, si bien no contarán con contenido propio por el momento, podrán utilizarse durante comparaciones de tipo dinámicas durante el juego. También se incluye una enumeración, *TokenType*, como identificador concreto del tipo de ficha y un listado con las ventajas de ataque frente a otros tipos.

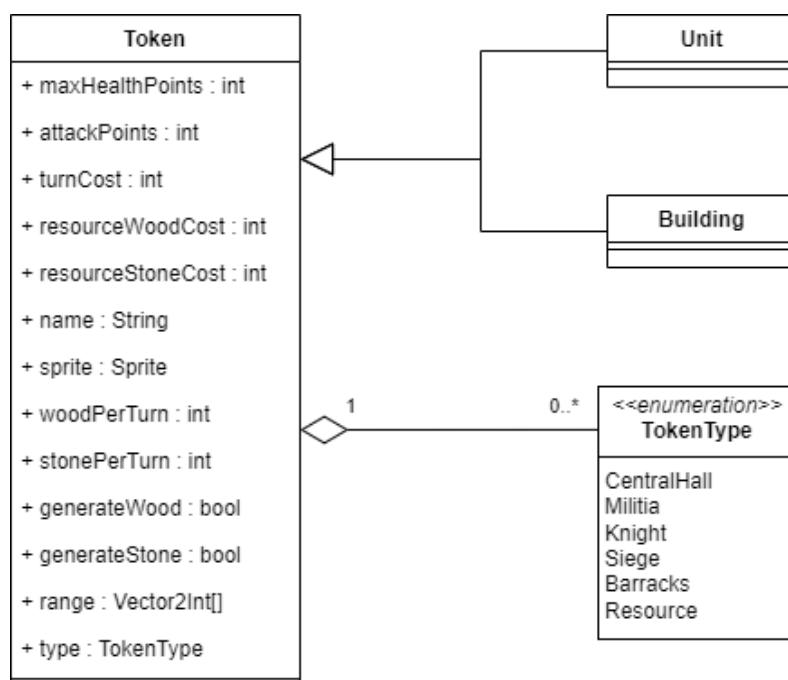


Figura 5.16: Diagrama UML de las clases para las fichas de juego

Una vez establecida la jerarquía y propiedades comunes de las fichas, la implementación de los seis tipos planteados para el prototipo hace uso del sistema **ScriptableObjects**¹⁰ de *Unity*. Este sistema permite definir clases completamente ajenas a *GameObject*, el elemento principal que puebla las escenas, con un enfoque orientado a datos y reutilizable. Añadiendo la clase *Token* como derivada de *ScriptableObject*, pueden generarse instancias especializadas desde el menú contextual del propio edi-

¹⁰<https://docs.unity3d.com/ScriptReference/ScriptableObject.html>

tor ([Figura 5.17](#)); es posible controlar cómo aparecen estas opciones en dicho menú con algunas etiquetas, como el nombre del grupo o el orden de aparición.

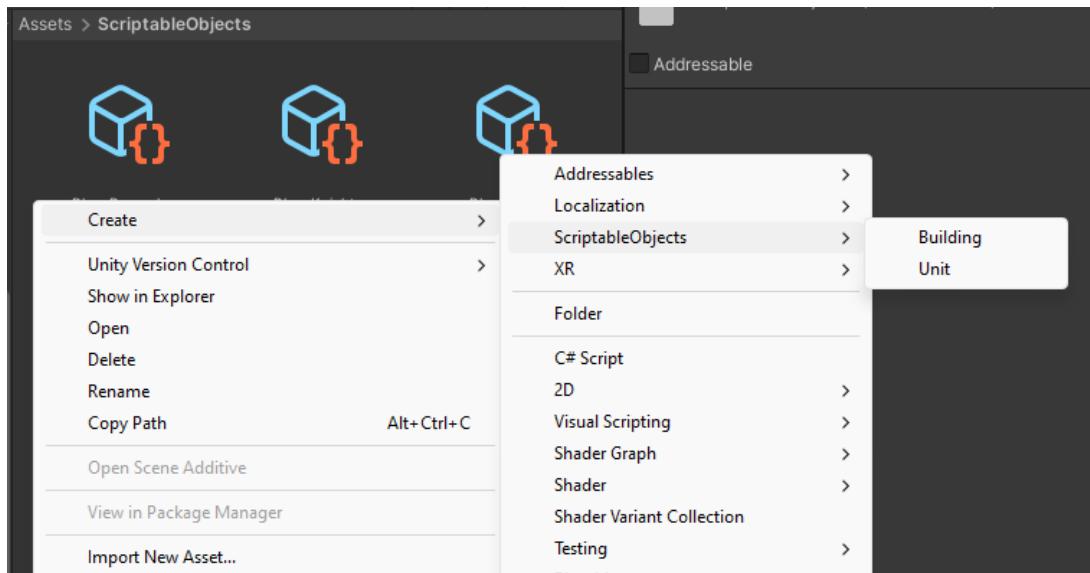


Figura 5.17: Creación de una instancia de objeto *Token* como *ScriptableObject*

Por cada tipo de ficha, y para ambos jugadores, se generará un objeto de este tipo con sus atributos concretos según los valores establecidos en el [Capítulo 2](#). La [Figura 5.18](#) presenta uno de estos objetos en el inspector de *Unity*, mostrando los atributos definidos en la clase general.

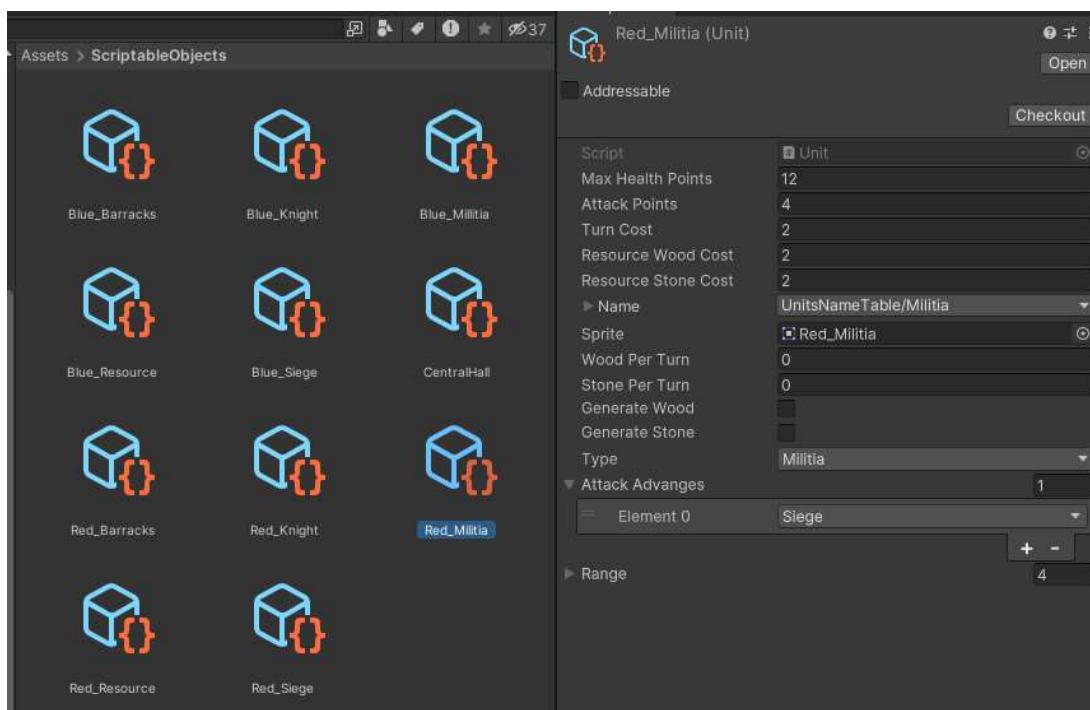


Figura 5.18: Inspector de parámetros de una instancia de un objeto *Token*

A continuación, las distintas fichas deben asociarse a los objetos correspondientes que las representen, para lo cual se prepara la clase *BoardToken* del [Listado 5.4](#). Esta clase abstracta una ficha del juego, conteniendo información sobre su estado, jugador propietario, y acceso a sus propiedades, junto a varios métodos que simplifiquen ese acceso: el método *BoardToken::ReceiveDamage* restará puntos de salud a la ficha considerando los puntos de ataque de la ficha rival y el bonus de ventaja (50 % de daño adicional); por otro lado, el método *BoardToken::CheckMovementAllowed* comprobará si la ficha puede desplazarse de una celda a otra según sus movimientos.

```
1 public class BoardToken : MonoBehaviour {
2     [SerializeField] protected Token tokenProperties;
3     [SerializeField] protected int playerOwner;
4
5     public float RemainingHP;
6     public bool CanMove;
7
8     private void OnEnable() { RemainingHP = tokenProperties.maxHealthPoints; }
9     public int GetPlayerOwner() { return playerOwner; }
10    public Token GetProperties() { return tokenProperties; }
11    public Vector2Int[] GetTokenRange() { return tokenProperties.range; }
12
13    public float GetRemainingHp01() {
14        return RemainingHP / tokenProperties.maxHealthPoints;
15    }
16
17    public bool ReceiveDamage(BoardToken other, out float damageDealt) {
18        Token otherProps = other.tokenProperties;
19
20        float multiplier = 1.0f;
21        if(otherProps.attackAdvanges.Contains(tokenProperties.type))
22            multiplier = 1.5f;
23
24        damageDealt = otherProps.attackPoints * multiplier;
25        RemainingHP -= damageDealt;
26        return RemainingHP <= 0.0f;
27    }
28
29    public bool CheckMovementAllowed(BoardCell from, BoardCell to) {
30        if (!CanMove) return false;
31        var diffVec = from.GetVectorToOther(to);
32        return tokenProperties.range.Any(move => move == diffVec);
33    }
34 }
```

Listado 5.4: Clase *BoardToken* para el control de las fichas de juego

Por último, cada ficha se prepara en el editor como un *prefab*, una colección de objetos de escena generalizados. La división en *prefabs* permite replicar el uso de los objetos que representan como objetos enlazados, donde un cambio sobre el *prefab* aplicará tales cambios a todas sus instancias por defecto. De nuevo, cada tipo de ficha se representará como un *prefab* para los dos jugadores, ya que contarán con diferencias visuales en los modelos 3D utilizados.

Aunque los modelos varíen, todas las fichas mantienen una estructura común, con un objeto vacío como padre, que tiene asociada una instancia de *BoardToken* y un objeto *Canvas* destinado a la interfaz gráfica básica, con una barra de salud y un botón invisible sobre la ficha (se utilizará más adelante al implementar la interacción). A modo de ejemplo, la [Figura 5.19](#) muestra la jerarquía construida para la ficha de Milicia del jugador rojo con los elementos mencionados y modelos 3D colocados espacialmente.

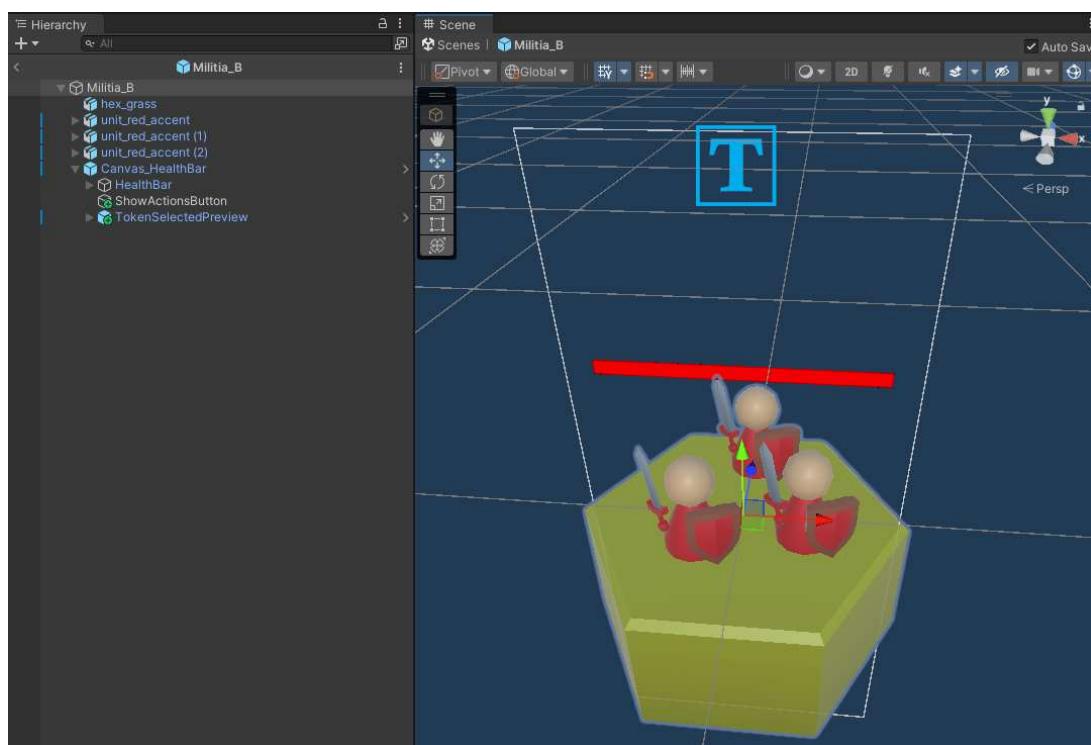


Figura 5.19: Jerarquía de un *prefab* para las fichas de juego

5.4.2. Estructura de datos del tablero

Hasta ahora, el tablero de juego se encuentra únicamente en la jerarquía de objetos de la escena del editor, pero, ya que cada celda contará con cierta información adicional, resulta necesario **mantener una estructura de datos** que permita manejar tal información en cualquier momento de la partida. Mediante esta estructura quedará facilitado el acceso a todo el estado de un turno y el cumplimiento de las reglas.

Al disponer de un identificador único para cada celda (su índice en ambos ejes), es posible diferenciarlas entre sí y, además, asignar un código de identificación *hash*, para lo cual queda sobreescrita la función *GetHashCode* en la implementación de la clase *BoardCell* ([Listado 5.3](#)). Con este código, la clase definida puede emplearse como clave en estructuras de datos organizadas, como los diccionarios o conjuntos. Estas estructuras mantienen su información organizada de tal forma que el acceso futuro resulte lo más eficiente posible, evitando operaciones de búsqueda de alta complejidad temporal y próximas al acceso inmediato¹¹.

Como principal diferencia entre diccionario y conjunto, el primero consta de un par de elementos clave-valor, mientras que el segundo solo recurre a la clave, siendo así ella misma el valor a almacenar. Para el tablero de juego, se decide utilizar un **diccionario donde las claves serán todas las fichas activas de la partida**, y tomarán como valor la celda en que se encuentre cada ficha.

Para facilitar la gestión de turnos y los movimientos realizados, la misma estructura se duplica con dos fines diferentes: una de ellas queda dedicada a mantener el estado del tablero al inicio del turno, antes de que el jugador realice cualquier acción; simultáneamente, la otra estructura se dedica a los movimientos realizados. Un ejemplo de estas estructuras viene dado en la [Figura 5.20](#), donde el estado actual contiene cuatro fichas en casillas distintas y se ha realizado un movimiento de una de ellas a otra casilla distinta y no ocupada.

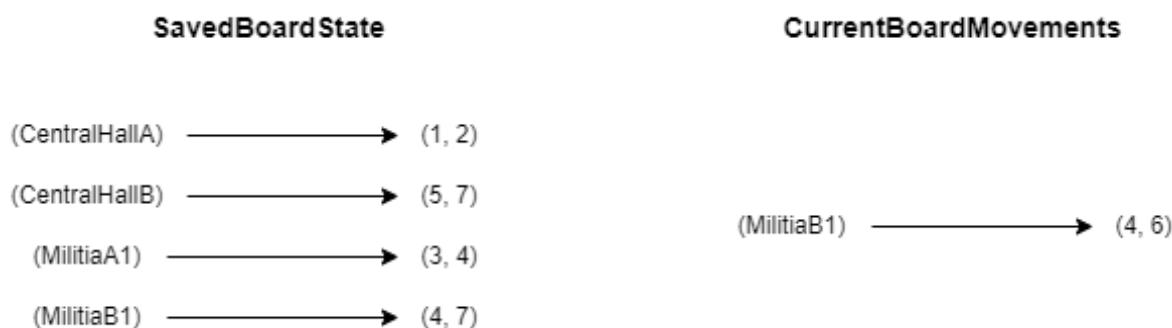


Figura 5.20: Ejemplo de contenido de las estructuras de datos del tablero

Ambas estructuras son gestionadas a **nivel global mediante una clase estática**, *TokenMoveRegister*, accesible desde cualquier punto de la aplicación y con varios métodos que facilitan el acceso a la información. La [Figura 5.21](#) presenta el diagrama de clases correspondiente a esta nueva clase, incluyendo las relaciones al resto de clases presentadas anteriormente. En un diagrama de este tipo, cada caja representa una clase, conteniendo una primera sección de atributos, seguida de los métodos

¹¹<https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2?view=net-8.0>

que implementa. La conexión entre dos clases indica una dependencia entre ambas; en este caso, el inicio de línea con un rombo hueco corresponde a una agregación, donde la clase origen contiene una o varias instancias de la clase a la que apunta, pero no controla su creación ni destrucción.

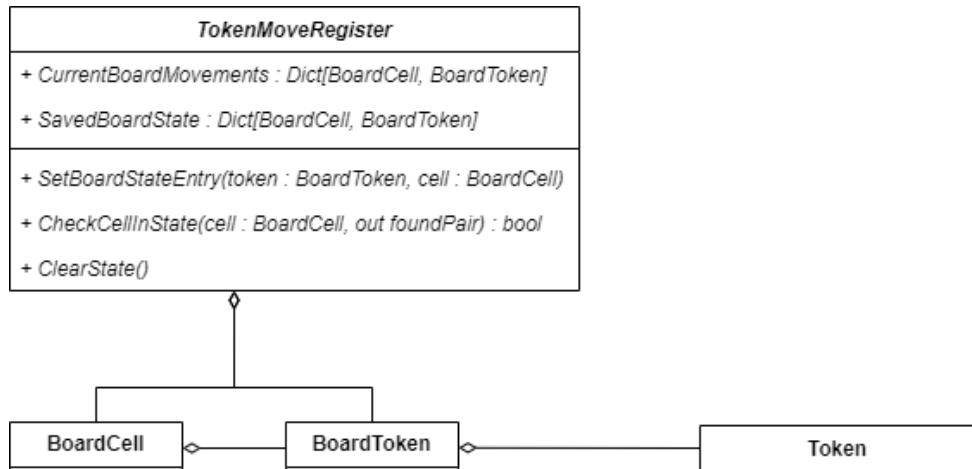


Figura 5.21: Diagrama de clases centrado en la clase *TokenMoveRegister*

5.4.3. Seguimiento de marcadores y posición en el tablero

Tras establecer la estructura que mantiene el estado del tablero de juego, el siguiente paso del incremento queda centrado en colocar las fichas sobre las distintas celdas que ocupan. Cada ficha contará con un marcador asignado para facilitar la gestión interna de los objetos, y además permite añadir un indicador sobre el mismo marcador representativo de la ficha correspondiente. Al igual que las celdas del tablero, la escena de juego dispone de todos los marcadores añadidos manualmente, utilizando objetos *ImageTarget* de *Vuforia* con la imagen asociada.

Como componente adicional de cada marcador, a estos objetos se les asocia una instancia de la nueva clase *TokenCellPositioner*, detallada en el [Listado 5.5](#) cuyo objetivo será **colocar las fichas en una de las celdas del tablero**. Su funcionamiento recurre al **uso de físicas mediante lanzamiento de rayos** que comprobarán si en la dirección hacia la que apunta la cara del marcador hay una celda, tras lo que el objeto visible, asignado mediante *BoardToken*, se incorpora en los movimientos del turno. Para asegurar que únicamente se comprueben colisiones con el tablero, las celdas son asignadas a una capa¹² específica. En caso de no encontrar un objetivo en el sentido positivo de la dirección, se realiza la misma comprobación en el sentido contrario.

¹²Las capas permiten separar las interacciones de físicas entre objetos: <https://docs.unity3d.com/ScriptReference/LayerMask.html>

```

1 public class TokenCellPositioner : MonoBehaviour {
2     [SerializeField] protected LayerMask GroundLayerMask;
3     public BoardToken AssignedTokenObject;
4
5     private RaycastHit[] _hits;
6
7     private void Start() {
8         _hits = new RaycastHit[1];
9     }
10
11    private void FixedUpdate() {
12        Ray testRay = new Ray(transform.position, transform.up);
13        int hitCount = Physics.RaycastNonAlloc(testRay, _hits, float.MaxValue,
14        GroundLayerMask, QueryTriggerInteraction.Collide);
15
16        if (hitCount == 0) {
17            hitCount = Physics.RaycastNonAlloc(new Ray(transform.position, -transform.
18            up), _hits, float.MaxValue, GroundLayerMask, QueryTriggerInteraction.Collide);
19        } if (hitCount > 0) {
20            BoardCell hitCell = _hits[0].collider.GetComponent<BoardCell>();
21            TokenMoveRegister.CurrentBoardMovements[AssignedTokenObject] = hitCell;
22        }
23    }
24 }
```

Listado 5.5: Clase *TokenCellPositioner* para el posicionamiento de las fichas sobre el tablero

Ya que todos los marcadores se encuentran simultáneamente en la escena, mantener **el seguimiento para todos a la vez puede perjudicar gravemente al rendimiento de la aplicación**. Ante esto, queda generada una clase dedicada a la agrupación de los marcadores, *TokenTargetsManager*, con varias colecciones según lo que representa cada marcador ([Figura 5.22](#)). En esta clase se mantienen listados de objetos *TokenCellPositioner* por cada tipo de ficha y para ambos jugadores, facilitando el acceso según el tipo necesario en cada momento. Internamente, se utiliza una estructura simple, *TokenState*, que controle también el uso del marcador.

Una vez quedan gestionados los movimientos de las fichas, su aplicación en la escena virtual se realiza en otro *script* que controlará en todo momento el estado del tablero visible, *BoardState*. Mediante esta clase, **los modelos 3D de la escena serán activados, reubicados y desactivados según el transcurso de la partida** y el turno de juego; al comienzo, únicamente gestiona el movimiento de las fichas y un indicador visual de esos movimientos.

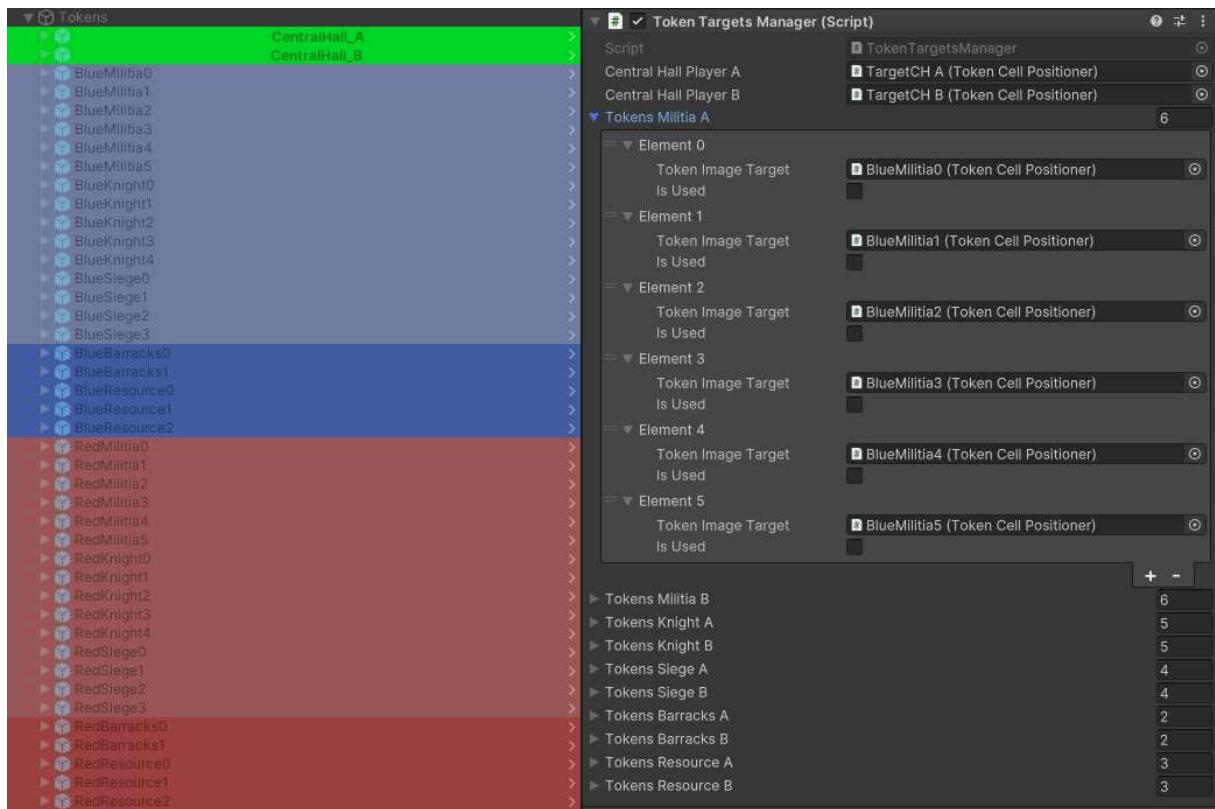


Figura 5.22: Marcadores *ImageTarget* de la escena y colecciones organizadas en un objeto *TokenTargetsManager*

En un turno, **cada ficha del jugador activo puede realizar una acción**: moverse, atacar o generar otra ficha. Cuando la partida avance y el jugador disponga de un amplio número de fichas, resulta especialmente complejo recordar todas las acciones establecidas en el turno. Por esto, todas las acciones estarán acompañadas de un indicador visual que ayude a diferenciar qué hará cada ficha al finalizar el turno actual. Para el movimiento, este indicador será una línea desde la celda origen hasta la celda destino con una animación representando la dirección de movimiento.

Un movimiento puede variar entre dos estados de validez, permitiendo o no ser aplicado según dos condiciones: el movimiento entra en el rango de acción de la ficha y la celda destino no está ocupada por otra ficha. Definiendo la clase *MovePositionMarker*, la posición y estado del movimiento pueden modificarse según el jugador desplace los marcadores. Esta clase, junto a *BoardState*, formarán el bucle principal de la lógica del juego, donde quedará resuelto el movimiento de fichas y el paso de turnos.

Las clases anteriores y su función son algo más extensas y complejas, por lo que serán expuestas mediante el uso de **diagramas de clase** que detallen su contenido y **diagramas de secuencia** para especificar su utilidad. Comenzando por *MovePositionMarker*, cuenta con un atributo del tipo *LineRenderer* propio de *Unity*, encargado

de dibujar una línea en la escena, acompañado de dos materiales para los dos posibles estados del movimiento. La animación que representa el movimiento se resuelve con un sombreador, desarrollado mediante el paquete *ShaderGraph*, con el cual pueden definirse sombreadores con programación visual por bloques. Este sombreador ([Figura 5.23](#)) utiliza el valor de una onda triangular en un punto dado por el tiempo transcurrido para animar una franja de un color especificado a lo largo del objeto.

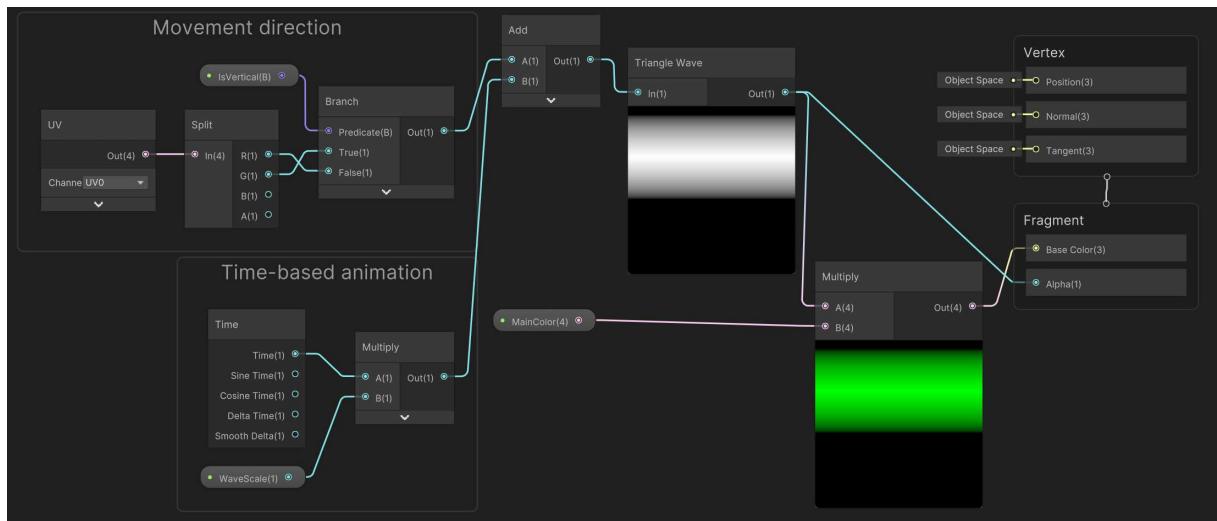


Figura 5.23: Sombreador para el indicador de movimientos de fichas

En el caso de *BoardState*, al inicio cuenta con dos métodos principales: *HandleTurnMovements* para el posicionamiento de las fichas en el tablero tras ser desplazadas, y *TurnEnded* como método de respuesta al finalizar el turno. Las clases involucradas en controlar el movimiento de los marcadores y colocar las fichas en el tablero son detalladas en la [Figura 5.24](#) junto a sus relaciones entre sí. Cabe destacar que, en ocasiones, estos diagramas omitirán algunos atributos y métodos básicos de *Unity*: por ejemplo, son omitidos los métodos *Update* y *Start* utilizados para ejecutar código en cada fotograma y al inicio de la ejecución, respectivamente, comunes a todos los objetos de una escena.

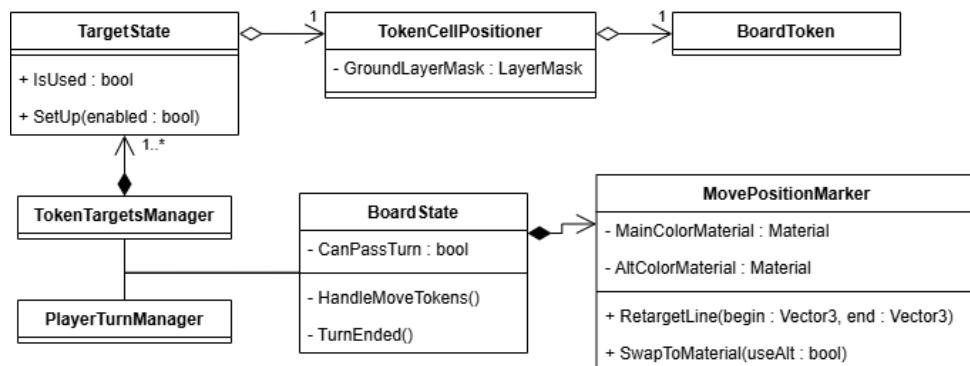


Figura 5.24: Diagrama de clases para el movimiento de marcadores y fichas

Para terminar, el comportamiento de las clases anteriores desde que es detectado un marcador hasta que la ficha es posicionada en el tablero queda resumido en el diagrama de secuencia de la [Figura 5.25](#). En cuanto se detecta un marcador y *Vuforia* activa los objetos asociados, aquel que cuenta con la clase *TokenCellPositioner* comienza a comprobar, mediante lanzamiento de rayos, su posición en el tablero. Al colisionar con una celda, queda actualizada la estructura interna de *TokenMoveRegister* con la ficha y celda del movimiento.

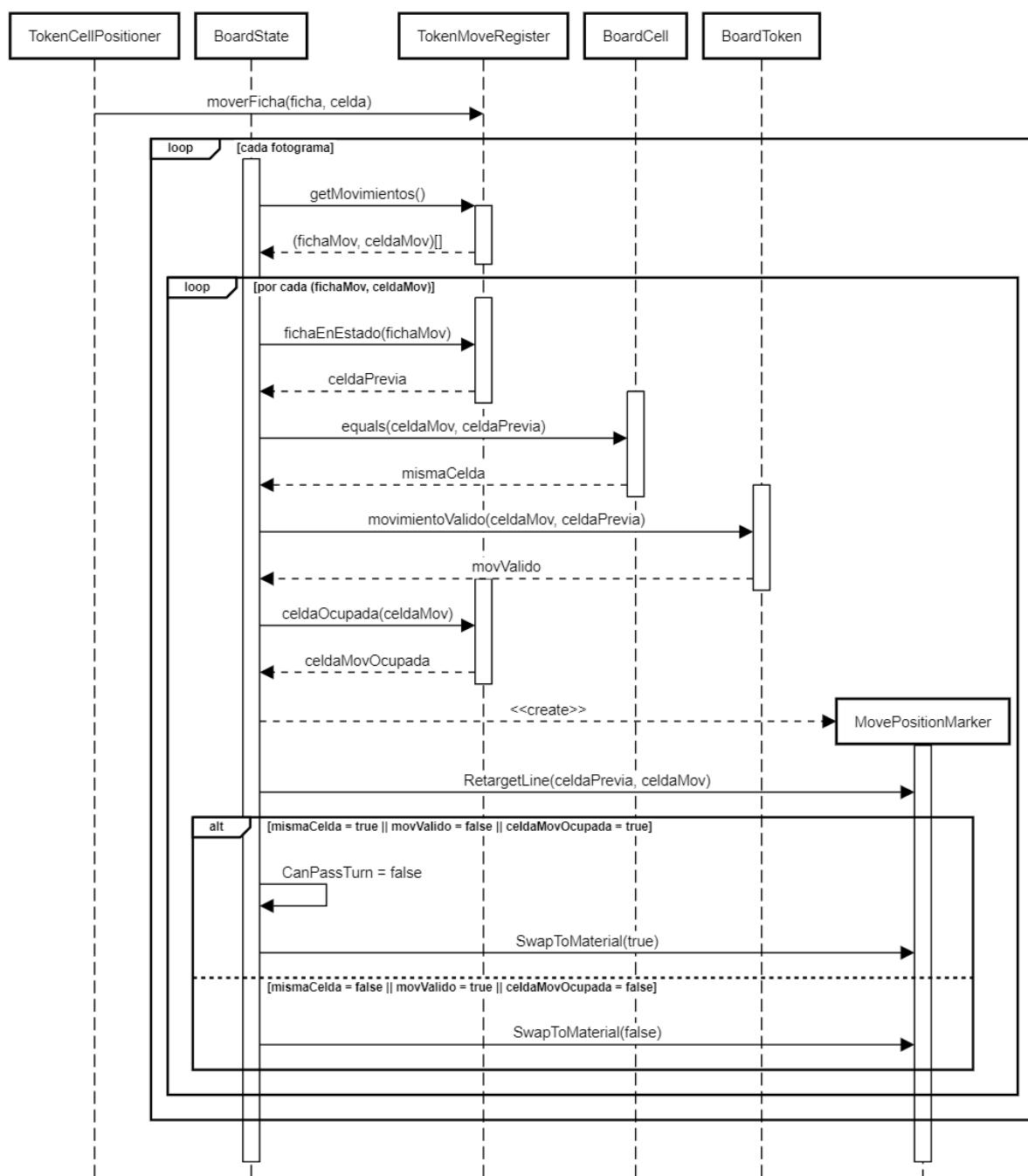


Figura 5.25: Diagrama de secuencia para el movimiento de marcadores y fichas

Posteriormente, en cada fotograma, *BoardState* recorre esa misma estructura de movimientos para, por cada entrada, comprobar la validez del movimiento (es decir, no es la misma celda, es un movimiento válido según el rango de la ficha, y el destino no está ocupado por otra ficha). En este bucle también se encarga de crear y actualizar el indicador del movimiento, estableciendo el origen y destino que tendrá la línea. Por último, dependiendo del resultado de la validez, se bloqueará o no el cambio de turno y el material del indicador de movimiento.

5.4.4. Acciones de fichas

Además del movimiento, algunas fichas pueden realizar otro tipo de acciones: **todas las unidades pueden atacar** y otras permiten **generar nuevas fichas**. El acceso a estas acciones se hará desde la interfaz gráfica de usuario, que mostrará un menú con las acciones de una ficha al presionar sobre ella (utilizando el botón incorporado a cada ficha mencionado al comienzo del incremento), siguiendo el diseño planteado en el [Capítulo 2 \(Figura 2.8\)](#).

Esta parte de la interfaz gráfica contará con un panel en la parte inferior de la pantalla poblado con varios botones cuadrados, uno por acción disponible de la ficha. Estos botones mostrarán la información relevante a la acción que contienen: para el ataque, basta con un texto indicativo junto a un ícono; la generación de fichas contará con más parámetros, como el nombre, previsualización y recursos necesarios. Los botones quedan agrupados en un área desplazable horizontalmente, dando solución a situaciones en que no disponga de espacio para mostrar todas las acciones. *Unity* facilita enormemente la implementación de una interfaz gráfica, ofreciendo múltiples controles preparados para las necesidades comunes: botones, áreas desplazables¹³, texto, barras deslizantes, desplegables, etc. El resultado de componer esta parte de la interfaz gráfica, junto a la jerarquía en la escena, se muestra en la [Figura 5.26](#).

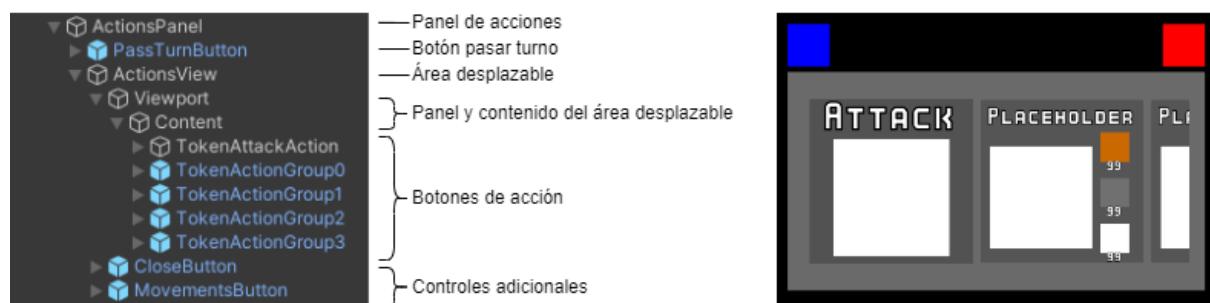


Figura 5.26: Jerarquía y resultado de la interfaz gráfica para las acciones de fichas

¹³<https://docs.unity3d.com/Packages/com.unity.ugui@2.0/manual/script-ScrollRect.html>

La gestión de las acciones en el código resulta algo más compleja al tener que enlazar información procedente de la ficha (clase *Token*) con la interfaz gráfica y adaptarla según la disponibilidad de acciones. Tal vinculación entre los dos extremos será resuelta con **varias clases nuevas, dedicadas a controlar ciertos aspectos de la relación, traspaso de información y visualización**. Primero, todo el panel y botones dedicados a las acciones son manejados por *ActionPanelControl*, con las referencias a objetos de la interfaz gráfica y un listado de hasta cuatro acciones.

Cada acción queda representada con *ActionGroupControl* para la gestión del contenido en cada botón (título, costes, imagen, etc.): este *script* se encarga de mantener actualizados los controles en todo momento, ajustando sus valores a la acción representada. Esta clase se asigna a un *prefab* al ser objetos que comparten atributos y comportamiento en la escena (objetos *TokenActionGroup* de la [Figura 5.26](#)).

Simultáneamente, cada ficha también recibe un componente en forma de un nuevo *script*, *TokenActionsControl*, dedicado a controlar las acciones realizadas por cada ficha individualmente. Mediante esta clase, son asignadas las acciones de cada ficha desde su *prefab* en un listado de objetos *TokenAction* con atributos intrínsecos de las fichas. Este cúmulo de clases, presentado en el diagrama de la [Figura 5.27](#), divide la responsabilidad en partes concretas del código independientes, con objeto de facilitar, en un futuro, posibles continuaciones al introducir nuevas acciones o mecánicas.

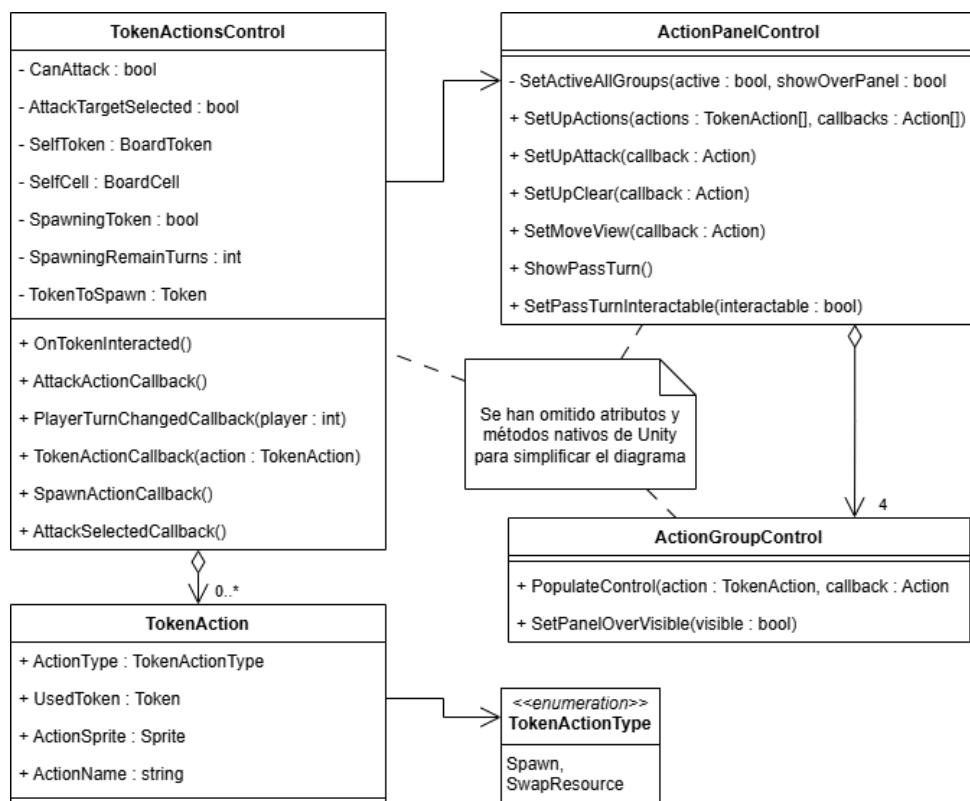


Figura 5.27: Diagrama de clases para la gestión de acciones de fichas

Al igual que en el apartado anterior, el comportamiento y traspaso de información queda resumido mediante un diagrama de secuencia, en la [Figura 5.28](#). Primero, al presionar sobre una ficha, la propia ficha notifica al controlador *ActionPanelControl* con las acciones propias de la ficha, que ajustará la interfaz gráfica (objetos *ActionGroupControl*) al contenido de estas acciones. En cuanto el jugador seleccione la acción, el método de respuesta asignado se invocará para resolverla.

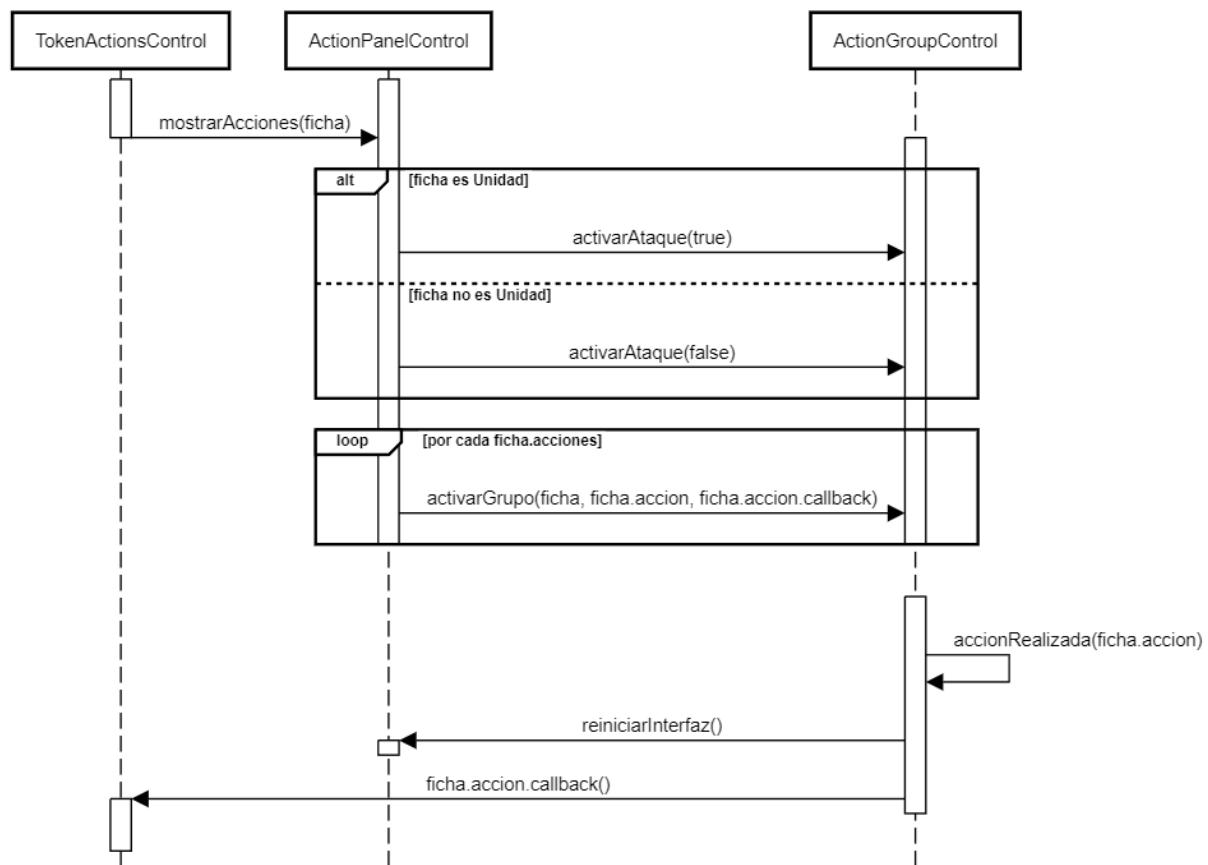


Figura 5.28: Diagrama de secuencia para la realización de acciones de las fichas

5.4.5. Ataque entre fichas

De forma ajena al resto de acciones, la realización de un ataque requiere un paso adicional. Tras seleccionar el botón de ataque, se muestran indicadores en el tablero sobre las fichas que pueden ser objetivo según el rango de la ficha atacante. Para esto, *TokenActionsControl* incorpora atributos y métodos dedicados a la acción de ataque, que controlarán estos indicadores. Este comportamiento recurre, de nuevo, a **métodos de respuesta asignados al seleccionar la ficha y el objetivo**.

El ataque solo puede ser realizado por fichas de unidad: Milicia, Caballería y Asedio. Siguiendo sus atributos definidos en la estructura *Token* definida al inicio de este

incremento, todas las fichas tienen asociado un listado con pares de enteros (atributo *Token::range*) que representan los movimientos o el rango de acción de la ficha como el número de casillas en los dos ejes direccionales.

Dada esta restricción, el botón asociado a la acción quedará oculto o se mostrará al seleccionar la ficha según su tipo. Además, el ataque se considera como una acción más: no es posible que una misma ficha en un mismo turno realice otra acción o un movimiento si se establece su ataque, o viceversa. Cuando pueda atacar, y el jugador presione uno de los indicadores del tablero sobre el objetivo, queda vinculada la acción y método de respuesta para que, al finalizar el turno, sea resuelto y se aplique el daño.

5.4.6. Resolver el turno

Después de establecer todo lo que es posible realizar durante el turno, el acto de **finalizar turno aplicará las acciones, movimientos y ataques indicados por el jugador**. Extendiendo la clase *TokenMoveRegister*, originalmente ideada para los movimientos sobre el tablero, gestiona también el resto de acciones posibles mediante colecciones adicionales que mantengan lo realizado por cada ficha en el turno.

De cara a simplificar el manejo de información entre varios puntos del código, se incorporan dos estructuras adicionales: *AttackState*, con referencias apuntando a la ficha atacante y objetivo y sus celdas correspondientes; y *SpawnState* para la generación de nuevas fichas, también con referencias a la ficha y celda origen, pero incorporando la ficha a generar y celdas donde puede ser colocada una vez aparezca. La [Figura 5.29](#) presenta el diagrama de clases actualizado con estas nuevas estructuras, que serán actualizadas conforme el jugador realice acciones y movimientos.

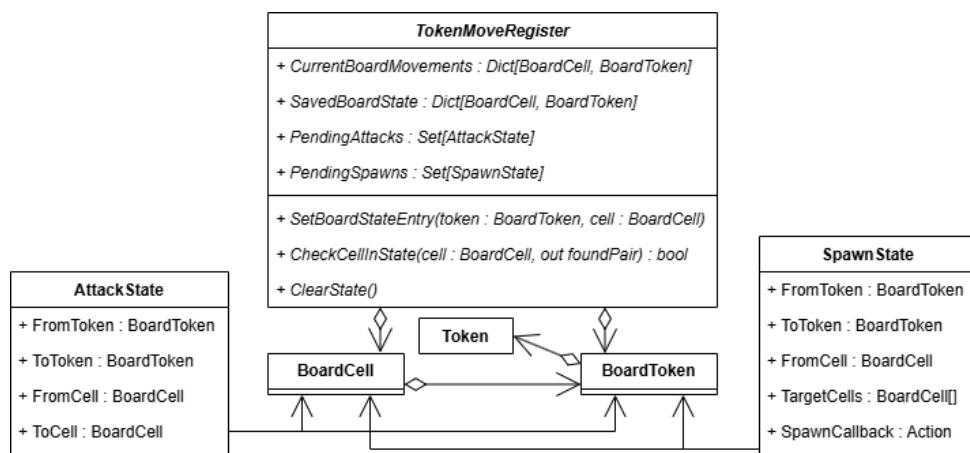


Figura 5.29: Diagrama de clases del control de movimientos, acciones y ataques actualizado

Estos cambios suponen una actualización del funcionamiento del script *BoardState*, añadiendo soporte a las nuevas colecciones. En este punto deben diferenciarse las partes de un turno: el **inicio**, donde se generarán nuevas fichas; la **actividad del jugador**, donde definirá qué hace cada ficha disponible; y el **final de turno**, en el que resolver esas acciones definidas. La etapa intermedia queda resuelta con lo implementado hasta ahora, y la finalización del turno es considerablemente simple.

Si, durante el turno, el jugador no ha realizado movimientos o acciones no permitidas o incompatibles con el estado de sus fichas, no podrá pasar el turno. Usando el mismo panel para las acciones, mientras no haya selección de ficha este panel mostrará un botón para el paso de turno (objeto con nombre *PassTurnButton* visible en la [Figura 5.26](#)), que será deshabilitado cuando no pueda pasar y se ocultará al seleccionar una ficha (cuando la acción sea resuelta, vuelve a mostrarse). Como respuesta a la pulsación del botón, será ejecutado el método *BoardState::PassTurn()* aplicando las acciones necesarias y cediendo el turno al jugador rival. En cuanto un turno acabe, ocurren un conjunto de operaciones que pueden agruparse en cuatro apartados:

1. **Movimientos:** Manteniendo el mismo comportamiento, las fichas que hayan cambiado su posición en el tablero se desplazan visualmente y las estructuras de datos son actualizadas.
2. **Ataques:** Por cada ataque establecido (accesible desde las nuevas colecciones de *TokenMoveRegister*), las fichas involucradas reciben el daño según sus parámetros, considerando el daño adicional cuando la ficha atacante cuenta con ventaja contra el tipo de la ficha objetivo.
3. **Limpieza:** Todos los indicadores visuales se eliminan y las estructuras de datos internas son vaciadas.
4. **Cambio de jugador:** Produce el cambio de turno desde *PlayerTurnManager* y establece los valores iniciales para el próximo turno (estado del turno y aumentar o disminuir contadores).

Respecto al estado del turno mencionado en la última etapa, su uso se centra en distinguir también las distintas etapas del inicio de un turno. En este caso, serán tres etapas que limitan inicialmente al jugador hasta resolver diversas cuestiones sobre el estado de sus fichas:

1. **Mover fichas generadoras:** Las fichas de Milicia son capaces de construir fichas de edificio (Cuartel y Generador) en su posición tras varios turnos. Al tener

ambas la misma posición, antes de introducir la nueva ficha es necesario desplazar la generadora a una casilla accesible en su rango. Cuando un movimiento de esta ficha a otra celda válida se detecte, la ficha se traslada y, en caso de no quedar otras fichas de este tipo pendientes, el turno avanza al siguiente estado.

2. **Generar fichas:** Todas las fichas en proceso de generación que hayan agotado su último turno de espera se activan y deben colocarse en el tablero. De nuevo, una vez sea detectado un movimiento de alguna de estas nuevas fichas a alguna de sus celdas objetivo, será posicionada y, cuando no queden más fichas pendientes, el turno continuará.
3. **Movimientos del turno:** Con las nuevas fichas disponibles en el tablero, el turno continúa su desarrollo normal, habilitando que el jugador establezca movimientos y acciones de sus fichas.

En cada fotograma, en lugar de únicamente comprobar los movimientos realizados, *BoardState* comprobará las acciones necesarias del turno según el estado en que se encuentre, evaluando su valor en el método *Update*, cuyo resultado provocará la llamada a los métodos correspondientes. Tras incorporar los nuevos métodos y atributos, la Figura 5.30 incluye el diagrama de clases actualizado a los cambios que extienden la clase *BoardState*; por simplicidad, en dicho diagrama se han omitido las clases relacionadas desde *TokenTargetsManager* (en la Figura 5.24 pueden verse esas clases), pero no son alteradas internamente.

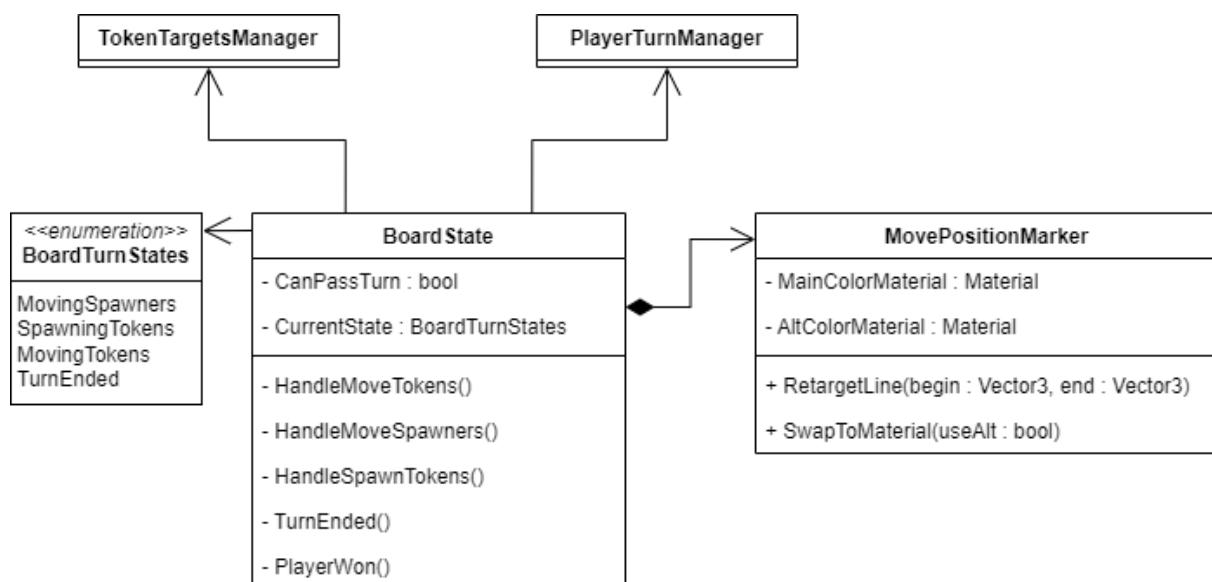


Figura 5.30: Diagrama de clases actualizado del movimiento de fichas y estado del tablero

5.4.7. Recursos del jugador

La última mecánica principal por implementar abarca la generación y uso de recursos durante la partida, formando un **factor limitante a la hora de generar nuevas fichas**. Los dos recursos disponibles son madera y piedra, ambos empleados como coste a la hora de realizar acciones que generen fichas; si el jugador no cuenta con suficientes recursos, no podrá realizar la acción. Buscando simplificar el acceso al conteo de recursos, serán gestionados en una clase estática y pública, *PlayerResourcesState*, correspondiente al [Listado 5.6](#), con atributos que mantengan el conteo de recursos y la cantidad generada por turno, junto a varios métodos para establecer estos valores y acceder a su valor actual.

```
1 public static class PlayerResourcesState {
2     public static readonly int WoodPerTurn = 1, StonePerTurn = 1;
3     public static int PlayerAWoodCount, PlayerBWoodCount, PlayerAStoneCount,
4         PlayerBStoneCount;
5     public static int PlayerAWoodIncome, PlayerBWoodIncome, PlayerAStoneIncome,
6         PlayerBStoneIncome;
7
8     public static void TurnSwitch(int newPlayer) {
9         if (newPlayer == 0) {
10             PlayerAWoodCount += WoodPerTurn + PlayerAWoodIncome;
11             PlayerAStoneCount += StonePerTurn + PlayerAStoneIncome;
12         } else if (newPlayer == 1) {
13             PlayerBWoodCount += WoodPerTurn + PlayerBWoodIncome;
14             PlayerBStoneCount += StonePerTurn + PlayerBStoneIncome;
15         }
16     }
17
18     public static bool IsTokenSpawnable(int player, Token token) {
19         if (player == 0) {
20             return PlayerAWoodCount >= token.resourceWoodCost && PlayerAStoneCount >=
21             token.resourceStoneCost;
22         } else if (player == 1) {
23             return PlayerBWoodCount >= token.resourceWoodCost && PlayerBStoneCount >=
24             token.resourceStoneCost;
25         }
26         return false;
27     }
28
29     public static void RemoveResources(int player, Token token) {
30         if (player == 0) {
31             PlayerAWoodCount -= token.resourceWoodCost;
32             PlayerAStoneCount -= token.resourceStoneCost;
33         }
34     }
35 }
```

```

29         } else if (player == 1) {
30             PlayerBWoodCount -= token.resourceWoodCost;
31             PlayerBStoneCount -= token.resourceStoneCost;
32         }
33     }
34
35     public static void RefundResources(int player, Token token) {
36         if (player == 0) {
37             PlayerAWoodCount += token.resourceWoodCost;
38             PlayerAStoneCount += token.resourceStoneCost;
39         } else if (player == 1) {
40             PlayerBWoodCount += token.resourceWoodCost;
41             PlayerBStoneCount += token.resourceStoneCost;
42         }
43     }
44
45     public static void SetPlayerIncome(int player, int woodIn, int stoneIn) {
46         if (player == 0) {
47             PlayerAWoodIncome = woodIn;
48             PlayerAStoneIncome = stoneIn;
49         } else if (player == 1) {
50             PlayerBWoodIncome = woodIn;
51             PlayerBStoneIncome = stoneIn;
52         }
53     }
54 }
```

Listado 5.6: Clase PlayerResourcesState para el control de recursos de los jugadores

Ambos jugadores comienzan la partida sin recursos, con un ingreso constante en cada turno de una unidad de cada recurso, sumado a una cantidad variable según el estado de la partida. Al crear una ficha de Recolector, la producción del recurso seleccionado incrementará en dos unidades. Cada recolector puede generar cualquiera de los dos recursos independientemente, sin ninguna limitación; la única acción que pueden realizar estas fichas será cambiar el recurso que generan.

Al finalizar el turno, en la etapa de cambio de jugador, el jugador que recibe el turno obtiene la cantidad indicada de recursos, obtenida como la suma de la cantidad fija más los ingresos adicionales por recolectores. Utilizando el método *PlayerResourcesState::IsTokenSpawnable()* puede comprobarse, al mostrar las acciones de una ficha, si son del tipo que genera otras fichas, que el jugador cuente con los recursos suficientes para la generación, activando o desactivando la acción según el resultado.

5.4.8. Preparación del tablero

Antes de dar por finalizado el incremento y que el prototipo sea jugable de inicio a fin, el tablero debe iniciarse en un estado determinado en el que cada jugador contará con su ficha principal de Ayuntamiento y una ficha de Milicia. Las fichas iniciales de los jugadores estarán en **posiciones opuestas del tablero y simétricas respecto al centro** para mantener la igualdad.

Previo a que *BoardState* tome el control de los turnos, otra nueva clase, *BoardSetup*, toma un listado de las fichas y celdas que deben colocarse antes de iniciar la partida. Comprobando la estructura que mantiene los movimientos realizados con los marcadores, en cuanto una de las fichas objetivo esté en su posición indicada quedará fija a la misma. Una vez todas las fichas hayan sido colocadas, la gestión del turno pasará a *BoardState* y la partida dará comienzo.

Debido a que esta preparación es solventada usando métodos y estructuras ya mencionadas anteriormente, no se incluyen diagramas que describan su contenido ni comportamiento. En la escena, la [Figura 5.31](#) corresponde a los atributos asignados en el inspector de *Unity*, con referencias al resto de clases necesarias y el listado con las celdas iniciales.

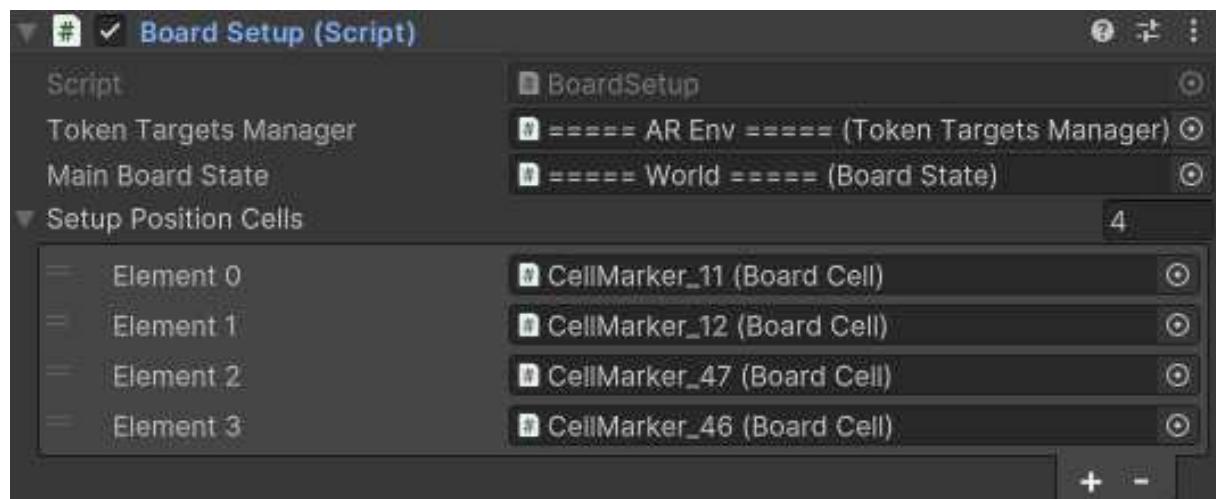


Figura 5.31: Inspector de la escena sobre la instancia de la clase *BoardSetup*

5.4.9. Resultados

Los resultados alcanzados en este incremento suponen el **flujo principal de juego**, incorporando las mecánicas principales y posibles acciones del jugador bajo las reglas establecidas. En la [Figura 5.32](#) se muestra el prototipo en funcionamiento du-

rante una partida, en la que se han desplazado las fichas iniciales y el jugador azul ha generado una ficha de Milicia adicional. Los siguientes incrementos refinarán estas mecánicas y mejorará la presentación del prototipo en su totalidad al usuario.

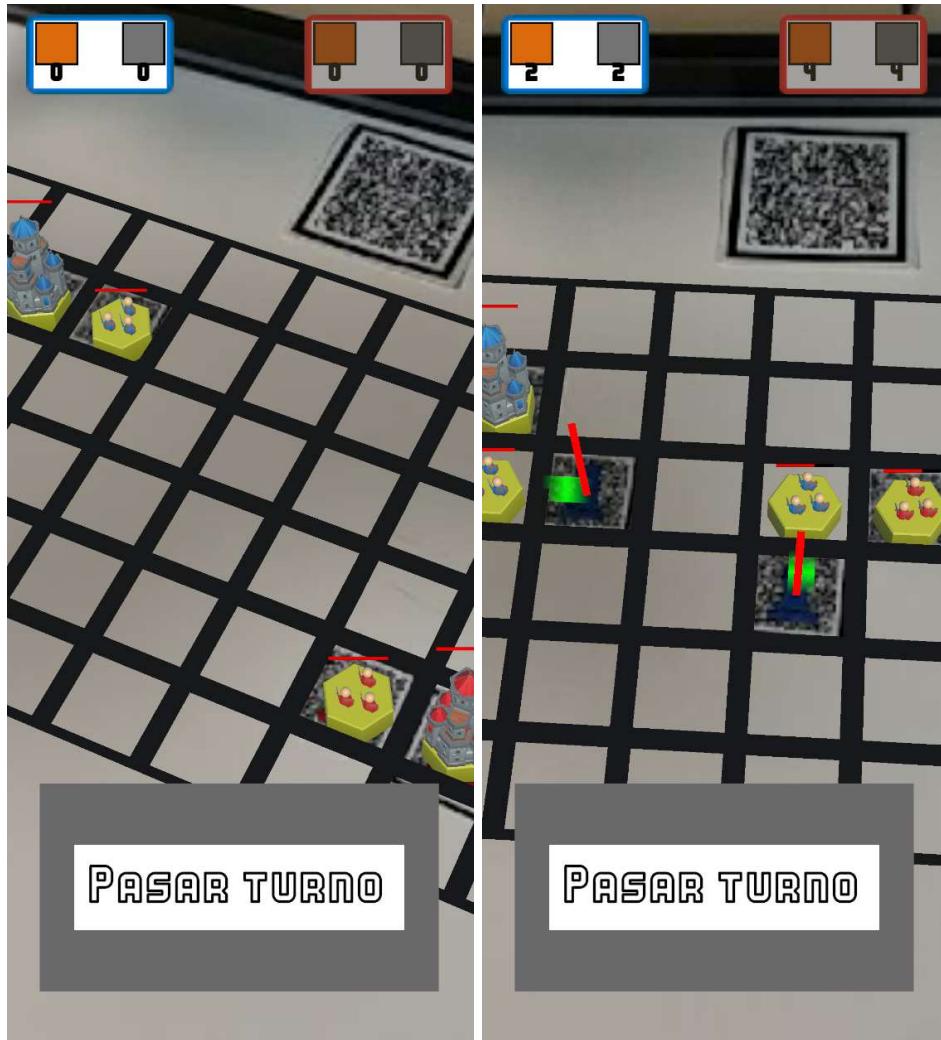


Figura 5.32: Capturas de pantalla del prototipo en funcionamiento tras el tercer incremento

5.5. Incremento 4. Recursos audiovisuales e interfaz gráfica

Después de contar con las mecánicas principales del juego implementadas en el prototipo, este cuarto incremento está dedicado a decorar la presentación mediante efectos visuales, menús y ventanas de interfaz gráfica de usuario y efectos sonoros. Como objetivo principal del incremento, se busca ofrecer al usuario una **mayor comodidad a la hora de presentar la información relevante de la partida**.

Al acabar este incremento, el prototipo permitirá comenzar y finalizar una partida desde un menú principal, así como configurar diversas opciones de la aplicación; también serán añadidas ciertas facilidades en el turno de cada jugador, con elementos visuales que indiquen acciones a realizar (ataque o mover ficha, por ejemplo), y efectos sonoros que resalten la interacción del usuario con la aplicación.

5.5.1. Indicadores y efectos visuales en la partida

Para comenzar el incremento, algunas acciones y el estado del turno resultan abstractas cuando el usuario no conoce el juego, dando lugar a confusiones sobre la manera de continuar con la partida o resolver ciertas acciones. Con esto, usando varios modelos 3D, animaciones y materiales, se prepararán varios indicadores destinados a momentos concretos de la partida, indicando qué acciones deben resolverse o posiciones relevantes en el tablero (por ejemplo, al generar una nueva ficha).

A continuación son listados los tres indicadores preparados, especificando la forma en que han sido generados y acompañados por la [Figura 5.33](#) con capturas de pantalla de los mismos en el editor:

- **Indicador de ataque:** Construido con el modelo 3D de dos espadas cruzadas entre sí. Utiliza un material transparente con el color rojo, usado como marca de los objetivos válidos cuando una ficha ataca. Cuenta con un botón mediante el que, tras ser presionado, el ataque tomará como objetivo la ficha sobre la que se coloca. Una vez seleccionado el objetivo, el color del material cambiará a verde.
- **Indicador de conflicto:** Símbolo de exclamación 3D construido mediante una esfera y un cilindro. Hace uso también del mismo material transparente de color rojo. Su uso estará reservado a marcar aquellas fichas que hayan generado una ficha nueva en su misma posición (principalmente fichas de Milicia al construir edificios), indicando que deben moverse para poder colocar la nueva ficha.
- **Indicador de posición:** Sistema de partículas como un efecto de espiral, utilizando cubos de color verde a modo de partículas, desplazadas en sentido positivo del eje vertical. Es generado mediante el propio editor de *Unity* para sistemas de partículas, usando un patrón de generación en forma de donut y una emisión constante de partículas a lo largo del recorrido. El desplazamiento de partículas recurre a la asignación de velocidad a cada partícula, ignorando la constante de gravedad del sistema de físicas simulado.

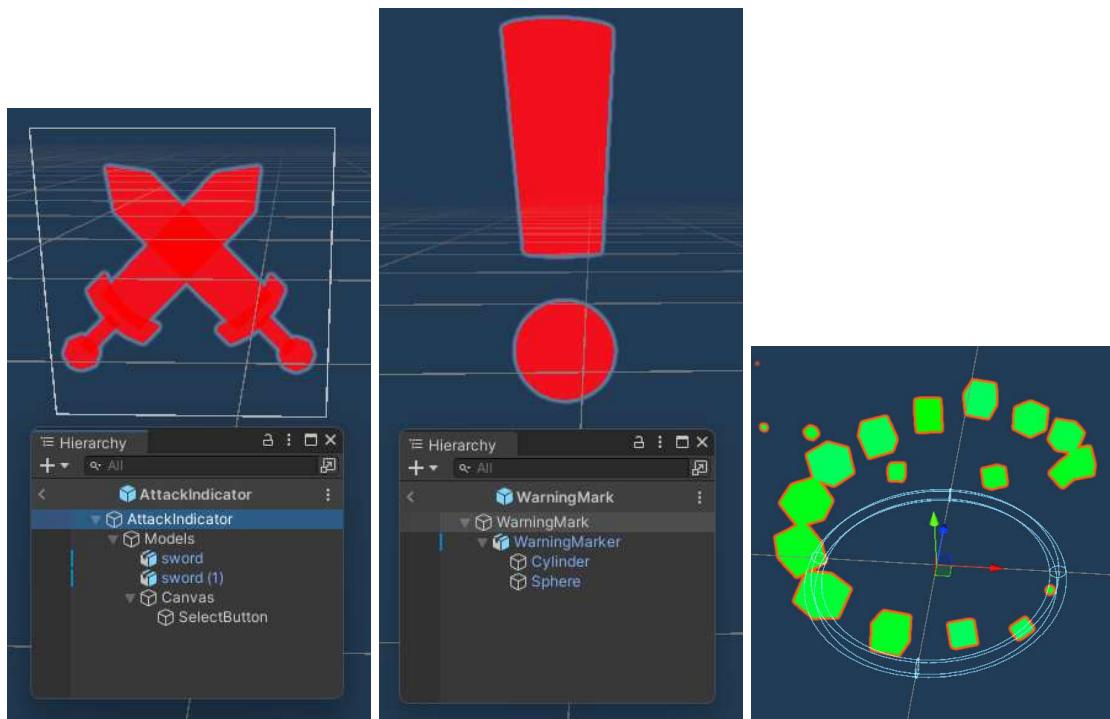


Figura 5.33: Indicadores visuales para acciones durante la partida

Estos indicadores son generados como *prefabs*, y se instancian en la escena desde los *scripts* relevantes. Al seleccionar una acción de ataque, el método *TokenActionsControl::AttackActionCallback()* de la ficha atacante comprueba, según su rango de movimiento, las fichas válidas como objetivo del ataque, colocando un indicador sobre cada una de las que se consideren válidas; a la vez, el botón de estos indicadores recibe como método de respuesta un conjunto de instrucciones que, al ser presionado, actualizan la estructura para ataques de *TokenMoveRegister*.

Por otro lado, cuando *BoardState* comprueba si alguna ficha ha terminado de generar otra ficha da con una unidad que haya generado un edificio en su misma posición, y, por tanto, espere que dicha unidad cambie de posición, colocará un indicador de conflicto sobre la ficha, marcando que debe ser desplazada para permitir colocar la nueva ficha. Finalmente, cada vez que una ficha deba colocarse en alguna celda del tablero, el *script* correspondiente generará un indicador de posición en esas celdas: esto ocurre al preparar el tablero, desde *BoardSetup*, o al generar una nueva ficha, gestionando los indicadores desde *BoardState*.

Los *scripts* mencionados se encargan tanto de **generar y colocar los indicadores en la escena como de destruirlos una vez no sean necesarios**: al resolver el ataque, todos los indicadores de ataque se eliminan; al mover una ficha generadora en conflicto, su indicador se elimina; y al colocar una ficha en alguna de las celdas destino, todos los indicadores de posición para esa ficha se eliminan.

5.5.2. Interfaz gráfica en la partida

En el incremento anterior, la interfaz gráfica de usuario quedó generada con elementos y colores planos, priorizando la lógica y comportamiento de los controles frente a la estética. Es ahora cuando todos esos elementos recibirán texturas, colores y animaciones que favorezcan la estética y resalten su comportamiento. Todos los objetos y controles de interfaz gráfica utilizarán los recursos obtenidos en los paquetes de *Kenny*, tal como se indicaba en el [Capítulo 2](#), especialmente con los recursos del paquete “*UI Pack (RPG Expansion)*”.

Comenzando por el panel de acciones situado en la parte inferior de la pantalla, al propio panel de fondo se le asigna una textura con un borde marcado, representando una ventana con cierto contenido. Un concepto esencial a la hora de diseñar y decorar una interfaz gráfica es la escala de los objetos: al enlazar un panel con una textura, es fácil encontrar problemas de visualización si las dimensiones y razón de aspecto no coinciden, originando un alargamiento y pérdida de calidad de la textura. La solución más común y generalizada consiste en la **división en 9 segmentos**¹⁴ de la textura, tal que los fragmentos de las cuatro esquinas queden fijos y únicamente se escalen o repitan las secciones centrales. En *Unity*, la [Figura 5.34](#) muestra el editor de imágenes y la asignación de los nueve segmentos utilizados al asignar la imagen a algún elemento visual.

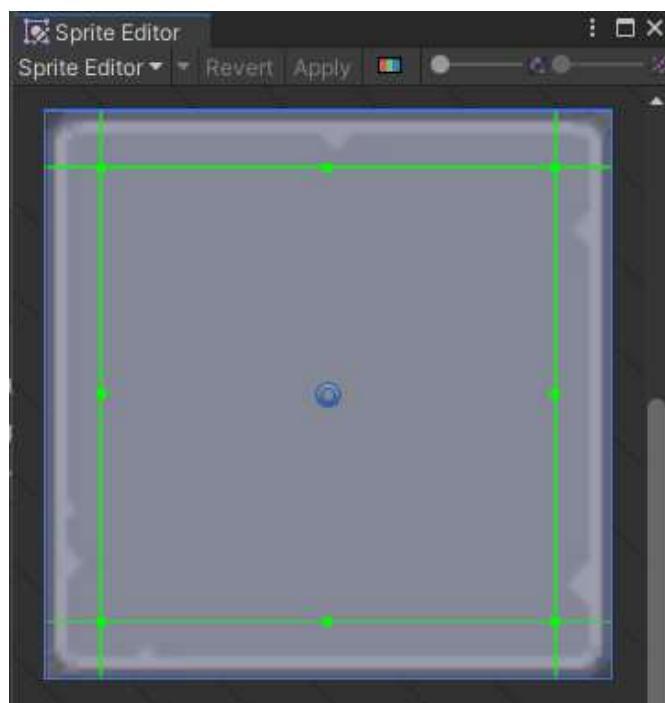


Figura 5.34: División de una imagen en nueve segmentos desde *Unity*

¹⁴Más conocido como *9-Slice* o *9 Patch*

Todos los objetos de la interfaz gráfica con una textura de fondo utilizarán esta solución de nueve segmentos: paneles, botones, agrupaciones, etc. En el caso de los botones, la mayoría utilizarán un ícono representativo de la función que cumplen, tomados principalmente del paquete “*Game Icons*” también de *Kenney*. En el mismo panel inferior, aparecen dos botones al seleccionar una ficha para realizar una acción: a la izquierda, un botón con el que visualizar el rango de movimientos o interacción de la ficha; a la derecha, un botón de limpieza que deshará la acción elegida. El primero utiliza un ícono con cuatro flechas, tratando de representar el movimiento, mientras que el segundo recurre a una cruz de color rojo; el resultado de aplicar los cambios queda presentado en la [Figura 5.35](#).



Figura 5.35: Panel de acciones decorado con texturas e iconos

Para terminar este panel, son generados **cuatro iconos para la acción de ataque, los dos recursos del jugador (madera y piedra), y otro más representativo de una cantidad de turnos**. El primero será una simple captura del modelo 3D de una espada (la misma utilizada en el indicador de ataque), obtenida del mismo paquete utilizado para las fichas, “*KayKit - Medieval Hexagon Pack*” por *Kay Lousberg*. Los dos siguientes también se obtienen como capturas de modelos 3D presentes en el mismo paquete, agrupando varios troncos de árbol para el recurso de madera. Respecto al ícono de tiempo, este paquete cuenta con una torre de reloj en uno de sus edificios; utilizando *Blender*, la geometría del reloj se separa del edificio para, de nuevo, tomar una captura, originando como ícono un reloj circular.

Con esto, el panel de acciones se encuentra completamente poblado y decorado (el ícono de cada acción es dinámico y se asigna al seleccionar una ficha, usando una captura del modelo de las fichas asociadas a la propia acción). Otra parte de interfaz gráfica implementada en el incremento anterior es el **panel superior con el conteo**

de recursos de ambos jugadores. Reutilizando los iconos de recursos y asignando una textura a los paneles, queda suficientemente decorado con una estética común junto al panel inferior. Además, a cada panel contador de recursos se le coloca un borde del color primario del jugador (azul y rojo), así como un panel transparente de color negro colocado sobre el jugador que no tenga el turno, usando así estos mismos paneles como indicador del turno.

Por último, entre los dos paneles superiores, se añade un botón más con un ícono de engranaje, que se utilizará como acceso para el menú de opciones de la aplicación. El resultado final de la vista durante la partida viene dado en la [Figura 5.36](#), que se actualizará según transcurra la partida y los jugadores realicen acciones.

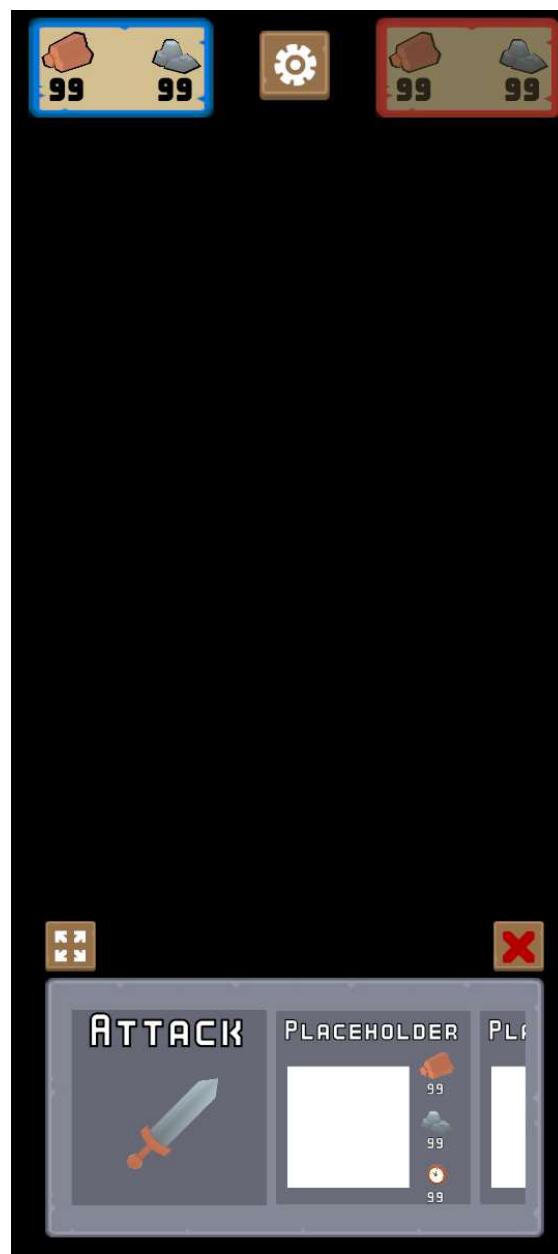


Figura 5.36: Interfaz gráfica de usuario visible durante la partida

5.5.3. Menú principal y opciones

Nada más abrir la aplicación, el jugador dispondrá de un **menú principal con varios botones que permita acceder a una nueva partida, abrir la ventana de opciones y salir del juego**. Este tipo de menú suele implementarse como parte de la interfaz gráfica, mediante controles 2D, pero, tratando de conseguir una estética más atractiva, el prototipo de *LandmARk* usará una escena 3D con varios elementos decorativos tomados del mismo paquete de modelos del que se obtienen las fichas. La [Figura 5.37](#) muestra la jerarquía de objetos en la escena del menú principal, con multitud de modelos 3D, botones, reproductores de audio y un controlador de animaciones.

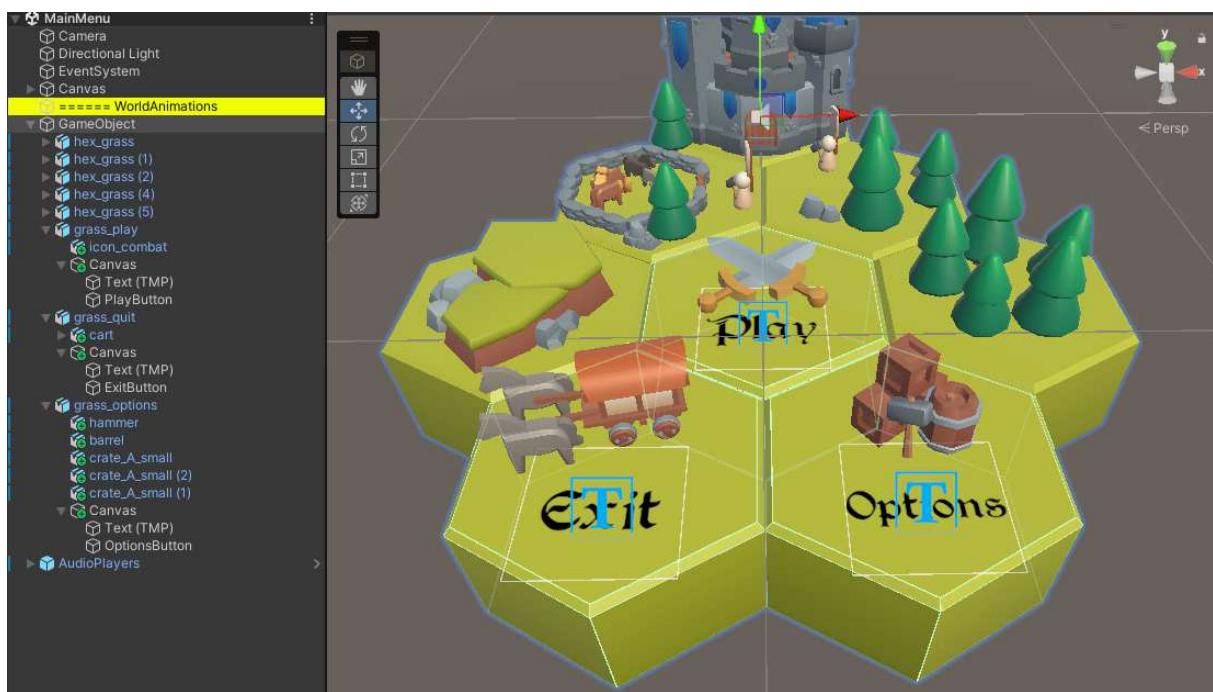


Figura 5.37: Jerarquía de escena para el menú principal del prototipo

En lugar de mostrar directamente la escena, se reproduce una corta animación en la que los distintos modelos aparecen y se colocan en la posición final. Esta animación hace uso de fotogramas clave definidos para cada modelo, directamente desde *Unity*, utilizando una interpolación mediante curvas: cada fotograma clave corresponde a un punto de una curva para el parámetro que representa (por ejemplo, escala en el eje X), asignando en el tramo intermedio el valor que toma la curva en cada instante. El resultado parcial de la animación se muestra en la [Figura 5.38](#), tomando tres instantáneas mediante las cuales se aprecia la aparición de los distintos elementos decorativos.

Para permitir al jugador configurar algunos aspectos de la aplicación, las dos escenas del prototipo (menú principal y partida), contarán con **una ventana compuesta por varios controles de interfaz gráfica**: tres controles deslizantes para cada

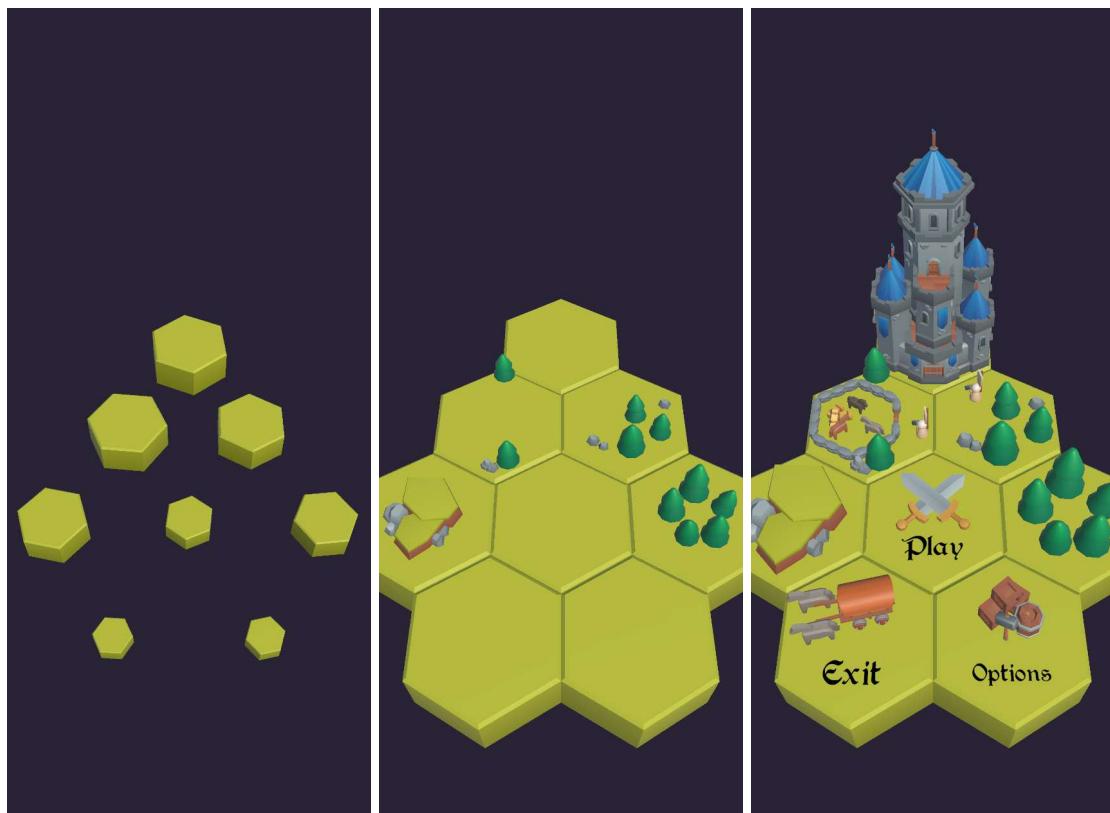


Figura 5.38: Animación de inicio del menú principal del prototipo

mezclador de sonido, que se detallarán en el próximo apartado; un selector desplegable con los idiomas disponibles para los textos; y varios botones con lo que confirmar o cancelar los cambios, mostrar un enlace (en forma de código QR) a una encuesta de retroalimentación, y salir de la partida o de la aplicación. La composición final de esta ventana se presenta en la [Figura 5.39](#).

Desde el menú principal, uno de los tres botones centrales mostrará la ventana de opciones, también con una animación en la que el panel emerge desde el centro de la pantalla. En este caso, la animación recurre a la técnica de ***tweening*** o ***inbetweening***, generando los fotogramas intermedios de una animación a partir de, generalmente, curvas o segmentos definidos matemáticamente. *Unity* no cuenta con una librería interna para *tweening*, pero es posible recurrir a paquetes externos con soporte a estas operaciones, como *DOTween*¹⁵.

Con esto, las animaciones de la ventana de opciones serán ejecutadas en un *script* dedicado, *OptionsMenuControl*, con el cual pueden, además, definirse varios ajustes y atributos de la animación. El paquete utilizado incorpora un amplio conjunto de ajustes adicionales a cada animación, pero en este caso únicamente se utilizará la función de suavizado o *easing*. Con estas funciones, la velocidad en que varía el atributo animado

¹⁵<https://dotween.demigiant.com>

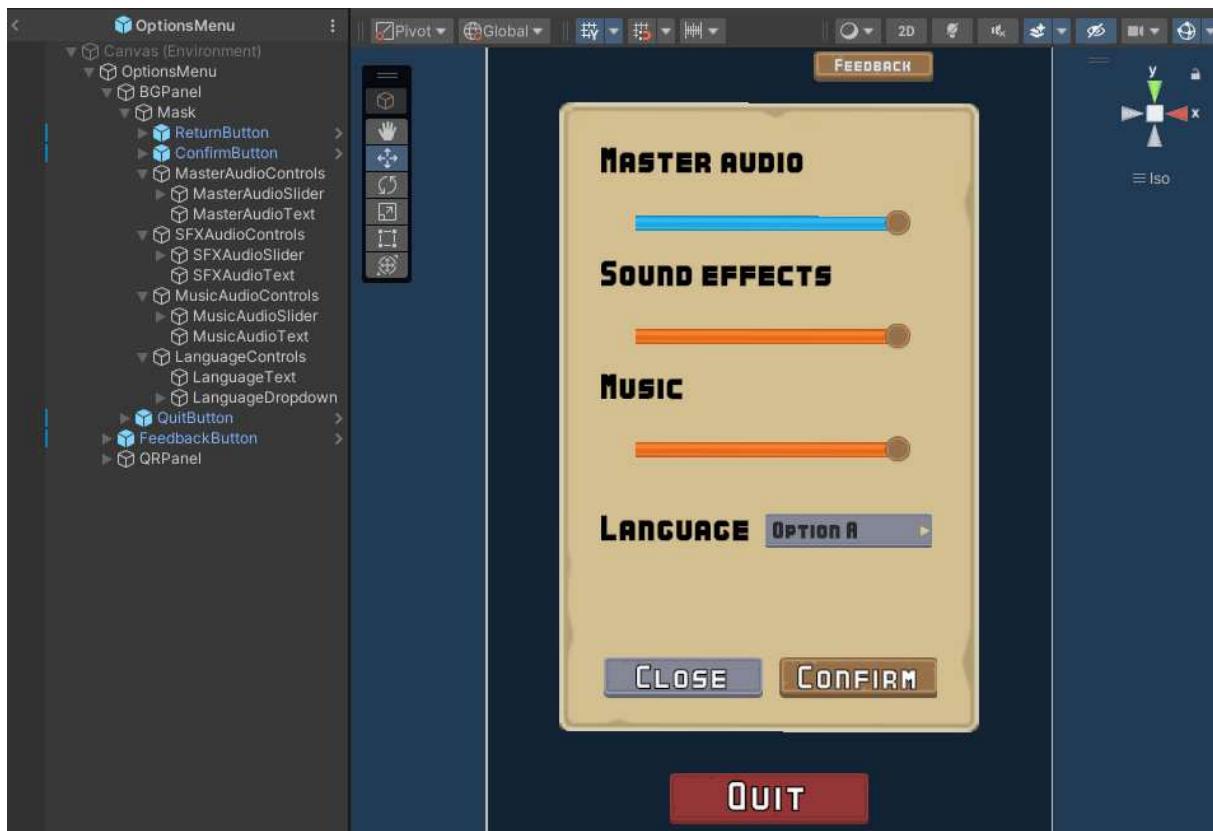


Figura 5.39: Jerarquía de escena para la ventana de opciones

variará a lo largo del tiempo¹⁶. Para la ventana de opciones, la animación de apertura escalará la ventana hasta su tamaño original, ocupando el centro de la pantalla, con un suavizado *ease-in*, mientras que, al cerrarse, será escalada hasta cero (y, por tanto, no sea visible), con un suavizado *ease-out*. El código que aplica ambas animaciones mediante *DOTween* se lista a continuación, en el [Listado 5.7](#):

```

1 public void ShowOptionsMenu()
2 {
3     OptionsParentPanel.DOScale(Vector3.one, AnimTime).SetEase(OpenEasing).Play();
4 }
5
6 public void CloseOptionsMenu()
7 {
8     OptionsParentPanel.DOScale(Vector3.zero, AnimTime).SetEase(CloseEasing).Play();
9 }
```

[Listado 5.7: Métodos para animación de la ventana de opciones](#)

¹⁶Según el tipo de suavizado, es posible obtener multitud de efectos distintos. Existen recursos en línea que muestran, con ejemplos generales, el comportamiento del suavizado (<https://cubic-bezier.com>) o algunas funciones comunes (<https://easings.net>)

5.5.4. Efectos sonoros y música

Además de los distintos efectos visuales implementados en el prototipo, **ciertos momentos de la interacción tendrán asociados un efecto sonoro**. Sobre la interfaz gráfica, cuando el jugador presione sobre algún control, sonará un efecto de clic obtenido del paquete *Kenney - UI Audio*; en el caso concreto de los controles deslizantes, cada vez que su valor sea actualizado, el efecto sonoro será similar, pero con una tonalidad suficientemente distinta al anterior como para ser distinguibles de inmediato.

Durante la partida, se componen cinco efectos sonoros con la estación digital *LMMS*, utilizando uno de sus clips de audio de golpe de tambor. Todos estos efectos están incluidos en el [Capítulo 2](#) y mostrados en la [Figura 2.7](#) del mismo capítulo. Su uso se reserva a situaciones concretas durante la partida: dos efectos quedan dedicados al ataque, reproducidos cuando el jugador pasa su turno si ha realizado un ataque; otros dos se reproducen cuando una ficha es destruida, con una variación para las fichas de unidad y otra para los edificios; el último efecto está dedicado a la finalización de una partida, cuando un jugador consiga destruir el Ayuntamiento rival.

Junto a los efectos de sonido, el prototipo contará con **música de fondo en ambas escenas (menú principal y partida)**. Las pistas musicales utilizadas son obtenidas del paquete *Music Loops Mini Set 2* de *Andrey Sitkov*, como pistas de corta duración ideadas para reproducirse en bucle. En lugar de esta reproducción en bucle, durante la partida se reproducirá una de las pistas elegidas de forma aleatoria cada cierta cantidad de tiempo¹⁷. Para el menú principal, únicamente la pista musical de mayor duración entre las seleccionadas se reproduce al inicio.

La gestión de efectos sonoros y música de fondo se vincula a la clase *AudioPlayerControl*, que cuenta con referencias como atributos a todos los efectos y clips de sonido utilizados en el prototipo, junto a múltiples métodos que permiten su reproducción. Con esto, la implementación queda reducida a asignar el reproductor de audio y realizar las llamadas correspondientes a dichos métodos. El control de la música aleatoria durante la partida es realizado con otro *script*, *SceneBGMPlayer*, detallado en el listado [Listado 5.8](#) cuyo comportamiento consiste en invocar continuamente al método de reproducción tras cierto retardo generado aleatoriamente.

```
1 public class SceneBGMPlayer : MonoBehaviour {  
2     [SerializeField, Min(0.0f)] protected float PlayDelaySeconds;  
3     [SerializeField] protected UnityEvent PlayBGMAction;
```

¹⁷El caso más famoso de esta técnica fue popularizada por *Minecraft*. El siguiente vídeo del creador de contenido Jaime Altozano explica excepcionalmente su funcionamiento y motivación en el contexto de *Minecraft*: https://www.youtube.com/watch?v=9rLc_brC8DY

```

4 [SerializeField] protected bool LoopBGM;
5 [SerializeField] protected float MinLoopDelay, MaxLoopDelay;
6
7 private void Start() { StartCoroutine(PlayBGM()); }
8 private IEnumerator PlayBGM() {
9     if(!LoopBGM) {
10         yield return new WaitForSeconds(PlayDelaySeconds);
11         PlayBGMAction.Invoke();
12     } else {
13         yield return new WaitForSeconds(Random.Range(MinLoopDelay, MaxLoopDelay));
14         PlayBGMAction.Invoke();
15         StartCoroutine(PlayBGM());
16     }
17 }
18 }
```

Listado 5.8: Clase SceneBGMPlayer dedicada a la reproducción de música de fondo

5.5.5. Mezclador de sonido

Unity permite definir múltiples canales de sonido y asignar filtros o modificadores individualmente. Si bien la solución más simple es tratar todo el audio en un mismo canal, facilitar al jugador un mayor control sobre la intensidad del volumen mejora en gran medida la experiencia de juego. Para ello, son generados tres canales en el mezclador de sonido, **Figura 5.40: volumen maestro**, correspondiente a todos los sonidos de la aplicación; **SFX** para los efectos sonoros; y **BGMusic** dedicado a la música de fondo. Una vez generados, cada objeto que emita sonido cuenta con un atributo referente al canal de audio en que emitirá sus sonidos.

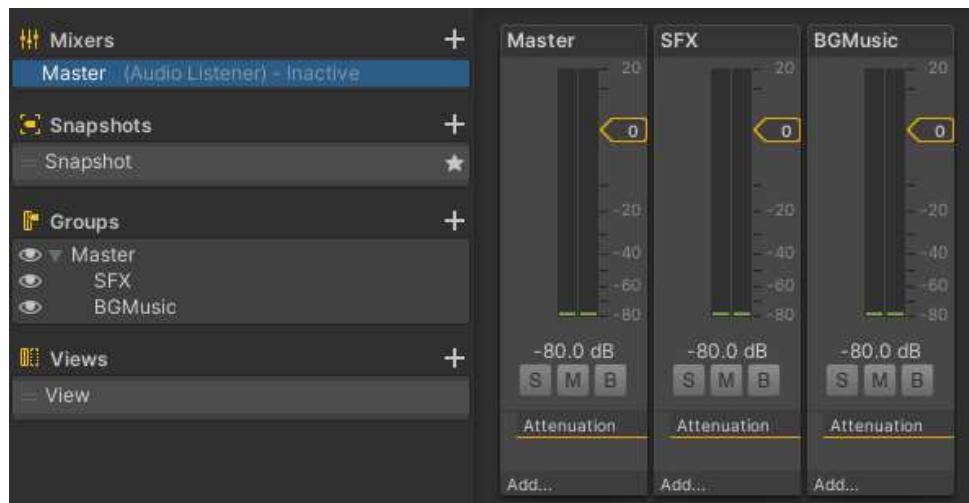


Figura 5.40: Mezclador de sonido con los canales y filtros definidos

Al vincular el valor del volumen de estos canales con el control deslizante de la interfaz de usuario, debe considerarse la escala numérica que utilizan, ya que el decibelio sigue un comportamiento logarítmico y la interfaz gráfica usa un control lineal. El *script VolumeSlider* del [Listado 5.9](#) se encarga de transformar el valor lineal obtenido del control deslizante a esa escala logarítmica, asignando el valor final al canal de audio y almacenando ese mismo valor en las preferencias internas de la aplicación¹⁸, permitiendo mantenerlo entre ejecuciones de la aplicación.

```
1 public class VolumeSlider : MonoBehaviour {
2     [SerializeField] protected AudioMixer audioMixer;
3     [SerializeField] protected string audioChannelName;
4
5     private void Start() {
6         Slider selfSlider = GetComponent<Slider>();
7         selfSlider.value = PlayerPrefs.GetFloat(audioChannelName);
8         audioMixer.SetFloat(audioChannelName, Mathf.Log10(selfSlider.value) * 20.0f);
9     }
10
11    public void OnValueChanged(float value) {
12        audioMixer.SetFloat(audioChannelName, Mathf.Log10(value) * 20.0f);
13        PlayerPrefs.SetFloat(audioChannelName, value);
14    }
15 }
```

Listado 5.9: Clase VolumeSlider dedicada a la asignación del volumen al mezclador de audio

5.5.6. Resultados

Con este incremento, el prototipo cuenta con una presentación temática y favorece una mejor experiencia de juego desde que se inicia hasta que finaliza una partida. Todos los controles de la interfaz gráfica han sido actualizados con texturas, animaciones y efectos sonoros, y varios momentos clave de la partida reciben efectos visuales que indiquen con claridad su estado.

El menú principal utiliza una de las pistas musicales al ser presentado como un elemento más de la escena, mientras que la escena de juego recurre a múltiples pistas de música cortas reproducidas en bucle indefinidamente.

¹⁸<https://docs.unity3d.com/ScriptReference/PlayerPrefs.html>

5.6. Incremento 5. Despliegue del prototipo

Hasta ahora, los incrementos desarrollados se centran en incluir mecánicas de juego, implementar la interfaz gráfica de usuario, y buscar una buena experiencia de usuario. Tras disponer de todas las partes diseñadas inicialmente en el prototipo, el siguiente paso, y el objetivo principal de este incremento, será generar el ejecutable de la aplicación a desplegar en dispositivos *Android*. Para ello, todas las escenas, vistas y ventanas deben enlazarse hasta ser accesibles, y el juego tendrá que probarse para verificar su funcionamiento antes de comenzar las pruebas con usuarios finales.

5.6.1. Navegación entre escenas

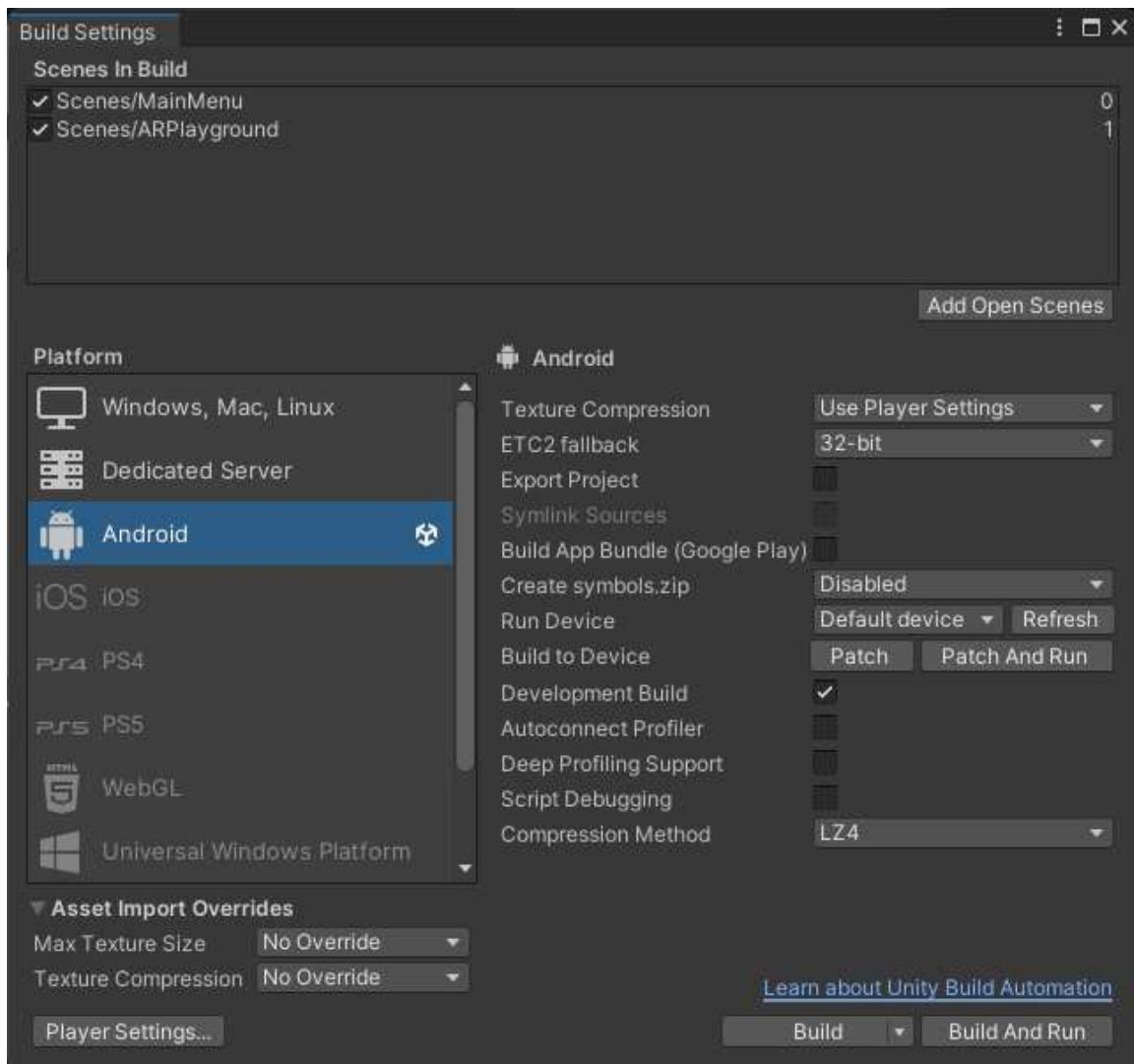
Siguiendo el diseño de la experiencia de usuario planteado en el [Capítulo 2](#), en que, además de la interfaz gráfica, se idea la navegación entre las vistas de la aplicación, el principal enlace será entre el menú principal y la escena de juego, dejando la ventana de opciones como menú contextual accesible desde ambas.

En *Unity*, y generalmente en todos los motores gráficos destinados a videojuegos, los distintos niveles son agrupados en escenas que representan el mundo virtual en el cual se desarrolla el juego. Este enfoque permite **repartir la carga de contenido** y poder manejar con mayor precisión el **uso de los recursos del sistema**: tener todo un mundo con personajes, interacciones, efectos y más, cargados y en funcionamiento simultáneamente, imposibilita en gran medida conseguir un rendimiento estable.

Aunque el prototipo de *LandmARk* no cuenta con una alta carga gráfica, ni entornos masivos, el uso de escenas permite, más que aportar al rendimiento, separar la funcionalidad de la aplicación y gestionar cada una independientemente. Por esto, el menú principal cuenta únicamente con la presentación y permite el acceso directo a la partida y los ajustes de la aplicación, mientras que la escena de juego controla todo el contenido y jugabilidad implementados. Ya que ambas escenas cuentan con un contenido considerablemente bajo, podrán ser cargadas y descargadas en su totalidad durante la transición sin perjudicar al rendimiento.

Desde *Unity*, la ventana de ajustes de compilación ([Figura 5.41](#)) permite seleccionar qué escenas estarán incluidas al compilar la aplicación y establecer un orden; cargar cualquiera de ellas se reduce a una única llamada a la función *SceneManager.LoadScene()* indicando el índice asociado y el modo de carga¹⁹.

¹⁹<https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.html>

Figura 5.41: Ajustes de compilación del proyecto *Unity*

5.6.2. Generación del ejecutable

Aunque *Unity* facilita enormemente la compilación de proyectos sobre distintas plataformas, en ocasiones, sobre todo al usar librerías y paquetes externos, resulta necesario configurar varios parámetros antes de obtener el resultado. En primer lugar, *Vuforia* necesita algunos ajustes establecidos en valores concretos para asegurar su compatibilidad accesibles desde los ajustes del proyecto:

- **API (Application Programming Interface) gráfica automática:** Con objeto de asegurar compatibilidad para un mayor rango de dispositivos y versiones de *Android*, además del soporte con las librerías externas, debe desactivarse y, en el listado de interfaces utilizadas, mantener únicamente *OpenGL ES3*, además de marcar el requisito de uso de *ES3.2* y desmarcar el resto ([Figura 5.42](#)).

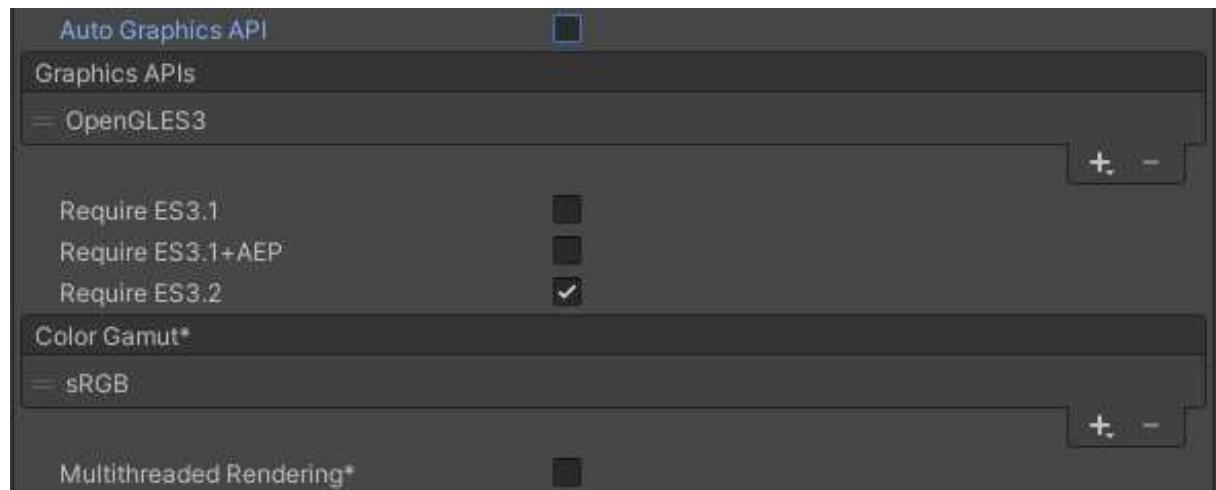


Figura 5.42: Ajustes gráficos del proyecto *Unity*

- **Renderizado multihilo:** *Vuforia* toma el control del renderizado a partir de lo que captura la cámara del dispositivo, por tanto, el renderizado no puede dividirse en varios hilos, ya que haría imposible que *Vuforia* pueda gestionarlo.
- **Nivel mínimo de API (Android):** Según la versión de *Unity* y *Vuforia* utilizadas, el requisito de la versión de *Android* mínima varía. En este caso, al utilizar *Unity 2022.3.16f1* y *Vuforia 10.20.3*, se establece en *Android 7.0*, el nivel mínimo que permite usar todas las librerías y extensiones necesarias.
- **Back-end de programación (Scripting):** Por defecto, *Unity* compila el código generado usando *Mono*, capaz de compilar sobre la marcha y bajo demanda, pero es común encontrar problemas de compatibilidad y rendimiento con ciertas librerías y plataformas. Ante esto, *IL2CPP* traduce el código a C++ para generar un ejecutable o paquete de la aplicación²⁰, ampliando el soporte a cambio de una compilación de todo el proyecto frente a cualquier cambio; *IL2CPP* será el *backend* utilizado en el proyecto.
- **Nivel de compatibilidad de la API:** Este nivel define la compatibilidad entre el lenguaje C# y las librerías de la plataforma .NET²¹. En busca de una mayor compatibilidad entre plataformas, se asigna a .NET Standard 2.1.
- **Generación de código IL2CPP:** Internamente, *IL2CPP* puede configurarse para generar el código con un enfoque para mayor eficiencia o un tamaño menor de aplicación. De cara a una mayor comodidad del desarrollo, el proyecto hace uso de la segunda opción, generando versiones más rápido y de menor tamaño.

²⁰<https://docs.unity3d.com/Manual/IL2CPP.html>

²¹<https://docs.unity3d.com/Manual/dotnetProfileSupport.html>

5.6.3. Resultados

Al terminar este incremento, el prototipo es compilado para generar el paquete de aplicación *Android* (fichero *APK*) que será distribuido e instalado sobre el dispositivo de pruebas. Antes de comenzar el siguiente incremento, se realizan algunas pruebas básicas de funcionamiento en busca de posibles errores críticos que imposibiliten las pruebas con usuarios (iniciar una partida, realizar algunos movimientos a lo largo de varios turnos o modificar algunos ajustes principalmente).

Dado que durante estas primeras pruebas básicas no se encontraron problemas críticos con el funcionamiento del prototipo, este incremento se considera finalizado para dar paso al último incremento con el cual finalizará el desarrollo.

5.7. Incremento 6. Pruebas y mantenimiento

Tomando como punto de partida el prototipo generado al final del incremento anterior, serán realizadas varias pruebas con usuarios en las que dos jugadores se enfrentarán a lo largo de una partida. Antes de comenzar cada partida, los conceptos básicos, objetivos y la jugabilidad de *LandmARK* se explicará a los jugadores, tras lo cual iniciarán la partida hasta que finalice o decidan detenerla.

5.7.1. Feedback de los jugadores

Los jugadores seleccionados para las pruebas iniciales serán personas conocidas, algo que no es recomendable al buscar opiniones imparciales. Siendo *LandmARK* un videojuego que requiere algo más de preparación que cualquier otro título disponible en dispositivos móviles (debido al uso de marcadores para Realidad Aumentada), su distribución a más personas y sobre un mayor rango de dispositivos complicaría el desarrollo y la preparación de pruebas de mayor calidad.

Por esto, el feedback de los jugadores y el comportamiento del prototipo será registrado manualmente, comprobando en vivo el transcurso de la partida y anotando comentarios de los jugadores, así como errores que surjan de forma imprevista. Además, se prepara una encuesta mediante *Google Forms* mediante la cual recopilar las opiniones de los jugadores al finalizar la partida, accesible desde la ventana de ajustes como un código QR. Esta encuesta contará con las siguientes cuestiones:

1. **¿Cómo valorarías tu experiencia general con LandmARK?:** Valor entero entre 1 (indiferente) y 10 (inolvidable).

2. **¿Qué aspecto te ha gustado más de LandmARK?:**

- Jugabilidad y reglas de juego
- Uso de la Realidad Aumentada
- Interfaz gráfica y facilidad de uso
- Gráficos y elementos visuales
- Sonido y efectos especiales

3. **¿Cuánto tiempo han tomado tus partidas en LandmARK?:**

- Menos de 5 minutos.
- De 5 a 10 minutos.
- De 10 a 15 minutos.
- Más de 15 minutos.

4. **¿Consideras que LandmARK es un juego fácil de comprender?:**

- Sí, he podido pensar varias estrategias y adaptarlas al transcurso de la partida
- Sí, pero me ha costado plantear varias estrategias o de mayor complejidad
- A medias, no he comprendido algunas cosas y no me he visto motivado a usarlas (por ejemplo, tipos de ficha)
- No, me ha costado entender el objetivo de la partida y he tenido dificultades para jugar
- No, no he podido jugar sin ayuda y/o supervisión constante

5. **¿Consideras cómodo el uso de Realidad Aumentada y marcadores en LandmARK?:** Valor entero entre 1 (Muy incómodo, preferiría no usarlo) y 10 (Muy cómodo, resulta natural durante la partida).

6. **¿Has encontrado alguno de los siguientes tipos de error durante la prueba?:**

- Errores críticos que han impedido continuar jugando
- Errores graves que no impiden continuar jugando
- Errores menores que no impiden continuar jugando

7. **¿Puedes dar más detalles sobre los errores encontrados? (Cualquier detalle, por nimio que parezca, puede ser de utilidad para solucionar el error):** Respuesta larga sin límite de caracteres. Opcional.

8. **¿Qué aspecto mejorarías de LandmARK?:**

- Jugabilidad: por ejemplo, nuevas reglas y mecánicas de juego que den profundidad a la partida
- Contenido: por ejemplo, nuevas fichas para un mayor repertorio de estrategias
- Modos de juego: por ejemplo, modo un jugador contra la máquina
- Efectos visuales: por ejemplo, uso de partículas o modelos de mayor calidad
- Sonido: por ejemplo, añadiendo efectos a ciertos eventos o incorporar más música de fondo
- Opciones y accesibilidad: por ejemplo, más ajustes de aplicación u opciones de accesibilidad (texto a voz, vibración, etc.)
- Otro

9. **¿Quieres añadir algún comentario más? ¡Tus opiniones, consejos e ideas son bienvenidas!: Respuesta larga sin límite de caracteres. Opcional.**

5.7.2. Etapa de pruebas. Problemas encontrados

A lo largo de varios días, un total de 11 personas probaron el prototipo de *LandmARK*, de los cuales únicamente 3 respondieron a la encuesta proporcionada. Aun así, **la evaluación en vivo permitió recopilar suficientes comentarios y errores** encontrados durante las partidas a solventar, formulando una versión actualizada.

En primer lugar, las cuestiones y errores menores, pero molestos en cierta medida, encontrados incluyen: incongruencia de elementos de la interfaz gráfica con su estado (botones deshabilitados visiblemente iguales que los botones activos); los indicadores de ataque se mantienen en la escena pese a resolver el ataque una vez finaliza un turno; varios jugadores indicaron la necesidad de contar con un indicador al pasar el turno a su rival.

Por otro lado, entre los errores más graves y críticos para la experiencia de juego destaca un problema con las fichas de Recolector, mediante el cual los jugadores no obtenían recursos al inicio de su turno, impidiendo generar nuevas fichas. Además,

cuando se selecciona la opción de volver al menú principal desde la ventana de opciones durante la partida, el estado interno del tablero no era reiniciado correctamente y se mantenía el estado de la partida anterior. Otro problema detectado permite al jugador realizar el movimiento de una ficha pese a seleccionar otra acción, rompiendo las reglas de juego establecidas.

Todas las pruebas realizadas cuentan con un registro de mensajes de consola internos, obtenido mediante *Android Logcat*. Este registro incluye todos los mensajes de estado generados por la aplicación, junto a varias etiquetas y una marca de tiempo, facilitando en gran medida la depuración de errores. El siguiente apartado detallará la solución a los problemas percibidos, resultando en una segunda versión del prototipo con mejor estabilidad y experiencia de juego.

5.7.3. Solución de problemas y segunda versión

Comenzando con los errores menores, la incongruencia de elementos de la interfaz gráfica ocurría al dar con varias excepciones en los *scripts* encargados de deshabilitar los botones involucrados, originadas por una asignación nula del parámetro que mantiene la referencia al botón de dichos *scripts*. Tras revisar y asegurar que los parámetros están correctamente asignados, el error queda solventado. Para los indicadores de ataque, una depuración del bucle de limpieza permitió determinar que el error ocurría únicamente cuando la ficha atacante era destruida, deshabilitando así sus *scripts* encargados de eliminar esos marcadores antes de hacer su función. Reordenando el orden de destrucción y ejecución de los eventos al resolver el ataque, el error pudo ser solucionado.

Respecto al cambio de turno, la solución implementada incluye un texto en la pantalla durante varios segundos indicando el jugador que recibe el turno ([Figura 5.43](#)). Este texto se presenta mediante una animación por *tweening* desde ambos laterales de la pantalla: para el jugador azul, aparecerá el texto de color azul desde la izquierda; para el jugador rojo, aparecerá desde la derecha y de color rojo.

Sobre los errores críticos restantes, el primero en solucionarse es el reinicio del estado interno del tablero al salir de la partida después de revisar los métodos de respuesta al presionar el botón correspondiente; en estos métodos no se realizaba la llamada necesaria al *script* que mantiene el estado del tablero para reiniciarlo a su estado original. Simultáneamente, la generación de recursos contaba con un problema grave cuando el jugador construye una ficha de Recolector, deshabilitando la producción de recursos totalmente. Una depuración y la revisión del registro interno concluyó

en una excepción originada por estas fichas. Durante la generación, no eran asignadas correctamente en algunas colecciones internas, lo que induce el error al tratar de buscar la ficha internamente.

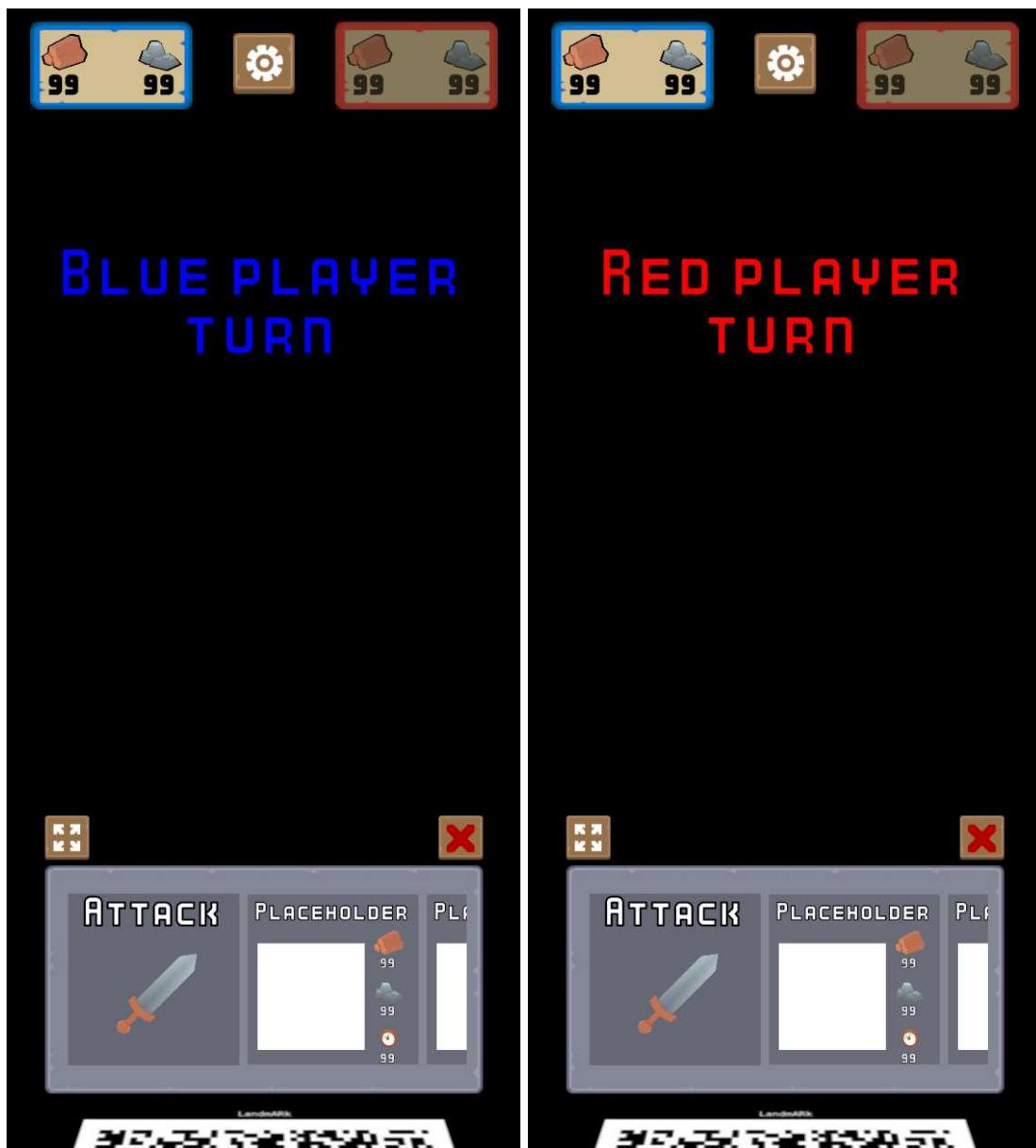


Figura 5.43: Textos indicativos del cambio de turno durante la partida

El último error, y el más costoso de resolver, permitía al jugador desplazar una ficha de Milicia a la vez que asignaba la construcción de un edificio, originando una incoherencia entre el estado visible e interno del tablero. Desde el *script BoardState*, encargado de comprobar si el jugador puede pasar el turno, el orden de las operaciones de comparación lógicas ignoraba resultados concretos, devolviendo valores como falsos positivos y falsos negativos. Después de reordenar y rehacer parte de esas comparativas, el código muestra el funcionamiento esperable.

5.7.4. Resultados

Con los contenidos de este último incremento, el prototipo de *LandmARk* ha pasado por una primera etapa de pruebas con usuarios finales, que ha permitido encontrar multitud de problemas y cuestiones no consideradas durante el desarrollo. Tales problemas se han solventado hasta obtener una segunda versión del título con mayor estabilidad y mejoras en su experiencia de usuario.

Este punto del desarrollo marca, bajo la planificación del proyecto, el final del mismo. Originalmente, *LandmARk* fue planteado con mucho más contenido y mecánicas, que fueron descartadas ante la limitación temporal del proyecto. Sin embargo, esas ideas podrán retomarse en una hipotética continuación del desarrollo, que son expuestas en el próximo y último capítulo de este documento.

Capítulo 6

Conclusiones

Este último capítulo incorpora otras cuestiones relativas a la continuidad del proyecto y el desarrollo realizado. Será, por tanto, un capítulo con menos contenido y con un enfoque más subjetivo, ligado a las intenciones e ideas para el futuro de *LandmARK*.

6.1. Trabajo futuro

El contexto en el cual ha sido desarrollado el prototipo de *LandmARK* implica ciertas limitaciones sobre el alcance del proyecto, dado el reducido marco temporal y las necesidades adicionales de un Trabajo Fin de Máster. Originalmente, *LandmARK* se ideó como un título muy distinto, buscando cierta compatibilidad con los intereses del Grupo de Gráficos y Geomática de Jaén, que poco a poco fue limitado y parte de las ideas descartadas para este primer prototipo.

Entre esas ideas descartadas, la mayoría favorecen al contenido del juego, incluyendo principalmente más tipos de ficha, acciones y mecánicas durante la partida. Un primer objetivo para el futuro del juego consiste en realizar el diseño y la implementación de este nuevo contenido con el mismo enfoque incremental. Algunas de las propuestas más planificadas se listan a continuación:

- **Nuevas fichas:** Extender el conjunto de fichas actuales con más unidades y edificios, como Arqueros (capaces de atacar a distancia sin recibir daño) o Murallas (bloqueando celdas del tablero para limitar el movimiento).
- **Puntos de interés del tablero:** Buscando fomentar la producción de recursos y navegación por el tablero, incluir puntos de interés que aporten algún beneficio

al jugador parece una decisión de interés. Estos puntos pueden aportar recursos adicionales de forma inmediata, o aplicar ciertas mejoras a la ficha que las visite (potenciar sus atributos o restaurar salud).

- **Mejora de fichas:** Una de las mecánicas que más potencial tuvo para ser incluida en el prototipo (pero acabó descartándose) es la mejora de fichas durante la partida, tomando como referente el funcionamiento de esta misma mecánica en *Sid Meier's Civilization*. Dedicando una serie de turnos y recursos para mejorar un tipo de ficha, las futuras fichas generadas contarán con sus atributos básicos aumentados. Así, los jugadores dispondrán de una jugabilidad con mayor profundidad, lo cual se traduce en más posibilidades para su estrategia.
- **Decoración del tablero:** La mayor limitación de la Realidad Aumentada reside en mantener la coherencia del entorno real del jugador. Aunque no sería apropiado decorar la totalidad de la pantalla, sí sería posible alterar el tablero y las celdas tal que resulten más atractivas, buscando replicar un escenario. Con esto, además, podrían prepararse variaciones y temáticas que mantengan un estilo común entre escenario, y fichas (por ejemplo, escenarios de llanuras, campos de batalla, o tierras nevadas).

Cualquier cambio planteado desembocaría nuevamente en más etapas de pruebas, tanto con nuevos jugadores como aquellos que experimentaron con *LandmARk* en su primera versión de prototipo jugable. Idóneamente, disponer de tiempo adicional de desarrollo permitiría ajustar el proyecto a más dispositivos móviles (soporte de Realidad Aumentada, diseño y escalado de la interfaz gráfica, rendimiento, etc.) y que estas pruebas puedan llevarse a cabo de forma remota, facilitando a los jugadores la aplicación y los marcadores necesarios.

6.2. Plan de mercado

Actualmente, los videojuegos disponen de un impacto masivo en la tecnología de consumo como el mayor sector en cuanto a facturación anual y uno de los más relevantes frente a continuas mejoras y avances tecnológicos, tanto hardware como software. Considerando la magnitud de un videojuego, independientemente de su contenido o alcance, resulta especialmente difícil adquirir una estabilidad suficiente en su desarrollo a medio y largo plazo; muchos desarrolladores y equipos independientes o con recursos limitados adoptan soluciones de micromecenazgo, donaciones voluntarias o introducir micropagos en el juego.

LandmARk no queda exento de estas cuestiones, y, si bien en su estado actual puede considerarse como una prueba de concepto, la decisión de continuar su desarrollo debe respaldarse de una planificación sobre posibles **ingresos a percibir una vez sea publicado**. Las labores de marketing y publicación de videojuegos y software en general requieren un estudio intensivo y elaborado por profesionales, pero cualquier desarrollador debe conocer nociones básicas y, más importante, entender el mercado en el que compite.

Los videojuegos móviles son, sin duda, el foco principal de atención de la mayoría de jugadores globalmente, pero los títulos de esta plataforma recurren a soluciones variadas para su comercialización. La respuesta más directa, aunque con peores resultados, es la venta directa de acceso al juego, donde los usuarios deben abonar una suma monetaria para adquirir y acceder al producto. Por otro lado, los lanzamientos con mayor repercusión suelen ofrecerse con acceso gratuito al juego y gran parte de su contenido (modelo *free to play*), incorporando micropagos que incorporen o desbloqueen contenido adicional. Otra alternativa que también parte del acceso gratuito es la presentación de publicidad y pausas comerciales durante el juego, mostrando otros productos inspirados en los intereses individuales del jugador.

Evitando indagar en la moralidad de estos modelos de negocio para videojuegos móviles, *LandmARk* se plantea como un **juego gratuito**, con todo su contenido principal accesible sin restricciones de pago. Como fuente de ingresos se haría uso de **elementos cosméticos que no alteren la jugabilidad o las mecánicas** establecidas: tableros y fichas temáticos, con diseños similares inspirados en escenarios o estilos artísticos concretos; música de fondo, ofertando variaciones de la banda sonora a reproducir durante la partida; o alteraciones de los efectos especiales y partículas que aparecen durante una partida.

A la vez, previo a un supuesto lanzamiento al mercado, se haría **uso de redes sociales que den a conocer *LandmARk* al público general**, publicando avances del desarrollo, tutoriales o guías de juego e interactuando con la audiencia objetivo. Por último, el lanzamiento del título sería realizado en la tienda digital de *Android Google Play*, lo que impone ciertas restricciones en el contenido ante sus términos de uso, pero facilita enormemente la gestión de versiones públicas y el contacto con los jugadores. Además, *Google Play* (y cualquier tienda o librería digital dedicada a la publicación de software), incluye en sus términos una tasa porcentual del 15 al 30 % de los ingresos generados¹ como pago por sus servicios.

¹<https://support.google.com/googleplay/android-developer/answer/112622?hl=en>

6.3. Valoración personal

Aprovechando esta sección final para expresar el punto de vista como desarrollador único del proyecto, *LandmARK* ha supuesto, frente a cualquier otra cosa, un reto personal mayor a cualquier otra experiencia anterior. Desarrollar un videojuego, para muchos, puede parecer algo simple o que requiera poco esfuerzo o tiempo, pero no es hasta que uno comienza a explorar este campo internamente que no conoce la difícil realidad que lo rodea.

Pensar ideas, plantear mecánicas o diseñar cómo interactúan los objetos puede surgir de cualquier persona, independientemente de su conocimiento o experiencia previa; donde reside la dificultad es en comprender la validez de esas ideas y planteamientos. El error más común de muchos desarrolladores nóveles acaba siendo el alcance de su producto, tratando de conseguir un sinfín de mecánicas, niveles, personajes, diálogos o cualquier otro elemento de un videojuego.

Es en esos momentos donde el papel de diseñador y director destaca, descartando todo aquello que suponga un mayor perjuicio que beneficio al producto final o a su desarrollo. Figuras extremadamente relevantes en la industria han demostrado la importancia de estos roles: como ejemplo, Fumito Ueda (*Ico*, *Shadow of the Colossus*) es ampliamente conocido por su estrategia de “diseño por sustracción” [92], reduciendo las mecánicas de sus proyectos al mínimo necesario para una experiencia gratificante.

***LandmARK* no es, ni jamás será, un juego perfecto**, y quizá quede estancado como un prototipo sin llegar a ser publicado. Pese a esto, el proyecto ha supuesto una gran experiencia personal, o, en otras palabras, un muro a superar que da lugar a una nueva etapa y oportunidades hacia el futuro próximo en la industria del videojuego.

Bibliografía

- [1] Grant Tavinor. *The Art of Videogames*. John Wiley & Sons, November 2009. ISBN 978-1-4443-1018-4. Google-Books-ID: LM3hnwGb8xUC.
- [2] Aaron Smuts. Are Video Games Art? *Contemporary Aesthetics (Journal Archive)*, 3(1), January 2005. URL https://digitalcommons.risd.edu/liberalarts_contempaesthetics/vol3/iss1/6.
- [3] Jonathan Jones. Sorry MoMA, video games are not art. *The Guardian*, November 2012. ISSN 0261-3077. URL <https://www.theguardian.com/artanddesign/jonathanjonesblog/2012/nov/30/moma-video-games-art>.
- [4] Ellie Gibson. Games aren't art, says Kojima | Eurogamer.net, January 2004. URL <https://www.eurogamer.net/news240106kojimaart>.
- [5] Desarrollo Español de Videojuegos DEV. *Libro Blanco del Desarrollo Español de Videojuegos 2023*. June 2024. URL <https://dev.org.es/es/publicaciones/libroblancodev2023>.
- [6] Benjamin Engelstätter and Michael R. Ward. Video games become more mainstream. *Entertainment Computing*, 42:100494, May 2022. ISSN 1875-9521. doi: 10.1016/j.entcom.2022.100494. URL <https://www.sciencedirect.com/science/article/pii/S1875952122000180>.
- [7] James Batchelor. GamesIndustry.biz presents... The Year In Numbers 2023, December 2023. URL <https://www.gamesindustry.biz/gamesindustrybiz-presents-the-year-in-number-2023>.
- [8] Scott Rogers. *Level Up! The Guide to Great Video Game Design*. John Wiley & Sons, April 2014. ISBN 978-1-118-87719-7. Google-Books-ID: UT5jAwAAQBAJ.
- [9] Mario Gonzalez Salazar, Hugo A. Mitre, Cuauhtémoc Lemus Olalde, and José Luis González Sánchez. Proposal of Game Design Document from software engineering requirements perspective. In *2012 17th International Conference on Computer Games (CGAMES)*, pages 81–85, July 2012. doi: 10.1109/CGames.2012.6314556. URL <https://ieeexplore.ieee.org/abstract/document/6314556>.

- [10] Sana Akhtar, Rabia Latif, Faiza Iqbal, and Ayesha Altaf. exGDD: Extended Game Design Document template for Mobile Game Design and Development. 6, 2021.
- [11] Max Chen, Shano Liang, and Gillian Smith. Stackable Music: A Marker-Based Augmented Reality Music Synthesis Game. In *Companion Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, CHI PLAY Companion '23, pages 22–28, New York, NY, USA, October 2023. Association for Computing Machinery. ISBN 9798400700293. doi: 10.1145/3573382.3616071. URL <https://dl.acm.org/doi/10.1145/3573382.3616071>.
- [12] Santiago Bedoya-Rodriguez, Cristian Gomez-Urbano, Alvaro Uribe-Quevedoy, and Christian Quintero. Augmented reality RPG card-based game. In *2014 IEEE Games Media Entertainment*, pages 1–4, October 2014. doi: 10.1109/GEM.2014.7118433. URL <https://ieeexplore.ieee.org/abstract/document/7118433>.
- [13] Newzoo. Top countries and markets by video game revenues, 2022. URL <https://newzoo.com/resources/rankings/top-10-countries-by-game-revenues>.
- [14] Simon Read. Gaming is booming and is expected to keep growing. This chart tells you all you need to know, July 2022. URL <https://www.weforum.org/agenda/2022/07/gaming-pandemic-lockdowns-pwc-growth/>.
- [15] Steven L. Kent. *The Ultimate History of Video Games, Volume 1: From Pong to Pokemon and Beyond . . . the Story Behind the Craze That Touched Our Lives and Changed the World*. Crown, 2001. ISBN 978-0-7615-3643-7. Google-Books-ID: C2MH05ogU9oC.
- [16] Tristan Donovan. *Replay : the history of video games*. East Sussex, England : Yellow Ant, 2010. ISBN 978-0-9565072-0-4. URL <http://archive.org/details/replayhistoryofv0000dono>.
- [17] Ralph H. Baer. Magnavox Odyssey Video Game Unit, 1972, 2006. URL https://americanhistory.si.edu/collections/nmah_1302004.
- [18] Science Museum Group. Atari 2600 Video Computer System | Science Museum Group Collection, 2007. URL <https://collection.science museum group.org.uk/objects/co8094237/atari-2600-video-computer-system-games-console>.
- [19] Gaming Historian. The Video Game Crash of 1983 - Gaming Historian, October 2009. URL <https://www.youtube.com/watch?v=kv7DJrLAZus>.
- [20] Amanda Onion, Missy Sullivan, Matt Mullen, and Christian Zapata. Video Game History - Timeline & Facts, October 2022. URL <https://www.history.com/topics/inventions/history-of-video-games>.

- [21] Drew Robarge. From landfill to Smithsonian collections: E.T. the Extra-Terrestrial Atari 2600 game, December 2014. URL <https://americanhistory.si.edu/explore/stories/landfill-smithsonian-collections-et-extra-terrestrial-atari-2600-game>.
- [22] Blake J. Harris. *Console wars : Sega, Nintendo, and the battle that defined a generation*. New York, NY : It Books, An imprint of HarperCollins Publishers, 2014. ISBN 978-0-06-227669-8 978-0-06-227670-4. URL <http://archive.org/details/consolewarssega0000harr>.
- [23] Manuel Delgado and Jonathan León. *Revolución Indie: La subversión cultural del videojuego*. Héroes de papel, 2019. ISBN 978-84-17649-16-6.
- [24] Oculus Rift virtual reality headset gets Kickstarter cash. *BBC News*, August 2012. URL <https://www.bbc.com/news/technology-19085967>.
- [25] Jisu Yi, Youseok Lee, and Sang-Hoon Kim. Determinants of growth and decline in mobile game diffusion. *Journal of Business Research*, 99:363–372, June 2019. ISSN 0148-2963. doi: 10.1016/j.jbusres.2017.09.045. URL <https://www.sciencedirect.com/science/article/pii/S0148296317303636>.
- [26] Erica L. Neely. Come for the Game, Stay for the Cash Grab: The Ethics of Loot Boxes, Microtransactions, and Freemium Games. *Games and Culture*, 16(2):228–247, March 2021. ISSN 1555-4120. doi: 10.1177/1555412019887658. URL <https://doi.org/10.1177/1555412019887658>. Publisher: SAGE Publications.
- [27] David Zendle, Rachel Meyer, Paul Cairns, Stuart Waters, and Nick Ballou. The prevalence of loot boxes in mobile and desktop games. *Addiction*, 115(9):1768–1772, 2020. ISSN 1360-0443. doi: 10.1111/add.14973. URL <https://onlinelibrary.wiley.com/doi/abs/10.1111/add.14973>. _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/add.14973>.
- [28] Dr CERULLI-HARMS, Frithjof Michaelsen, Marlene MÜNSCH, Christian THORUN, and Pierre Hausemer. *Loot boxes in online games and their effect on consumers, in particular young consumers*. July 2020.
- [29] Leon Y. Xiao. Loot box State of Play 2023: A global update on regulation, December 2023. URL <https://www.gamesindustry.biz/loot-box-state-of-play-2023-a-global-update-on-regulation>.
- [30] Yemaya J. Halbrook, Aisling T. O'Donnell, and Rachel M. Msetfi. When and How Video Games Can Be Good: A Review of the Positive Effects of Video Games on Well-Being. *Perspectives on Psychological Science*, 14(6):1096–1104, November

2019. ISSN 1745-6916. doi: 10.1177/1745691619863807. URL <https://doi.org/10.1177/1745691619863807>. Publisher: SAGE Publications Inc.
- [31] Dilsad Coknaz, Ayse Dilsad Mirzeoglu, Halil Ibrahim Atasoy, Seval Alkoy, Hakkı Coknaz, and Kemal Goral. A digital movement in the world of inactive children: favourable outcomes of playing active video games in a pilot randomized trial. *European Journal of Pediatrics*, 178(10):1567–1576, October 2019. ISSN 1432-1076. doi: 10.1007/s00431-019-03457-x. URL <https://doi.org/10.1007/s00431-019-03457-x>.
- [32] Hannah R. Marston and Rachel Kowert. What role can videogames play in the COVID-19 pandemic? *Emerald Open Research*, 1(2), January 2020. ISSN 2631-3952. doi: 10.1108/EOR-02-2023-0011. URL <https://doi.org/10.1108/EOR-02-2023-0011>. Publisher: Emerald Publishing Limited.
- [33] Daniel Alanko. The Health Effects of Video Games in Children and Adolescents. *Pediatrics In Review*, 44(1):23–32, January 2023. ISSN 0191-9601. doi: 10.1542/pir.2022-005666. URL <https://doi.org/10.1542/pir.2022-005666>.
- [34] Liz England. “The Door Problem”, 2014. URL <https://lizengland.com/blog/2014/04/the-door-problem/>.
- [35] Richard A. Bartle. *Designing Virtual Worlds*. New Riders, 2004. ISBN 978-0-13-101816-7. Google-Books-ID: z3VP7MYKqalC.
- [36] Heather Maxwell Chandler. *The Game Production Handbook*. Jones & Bartlett Publishers, 2014. ISBN 978-1-4496-8809-7. Google-Books-ID: MUOG6CE08ucC.
- [37] Sakurai. Masahiro Sakurai on Creating Games - YouTube. URL https://www.youtube.com/@sora_sakurai_en.
- [38] Megan Farokhmanesh. Mass Layoffs Are Causing Big Problems in the Video Games Industry. *Wired*, January 2024. ISSN 1059-1028. URL <https://www.wired.com/story/the-video-game-industry-is-just-starting-to-feel-the-impacts-of-2023s-layoffs/>. Section: tags.
- [39] Zack Zwiezen and Kenneth Shepard. 89 Days Into 2024 And 8,800+ Video Game Layoffs Have Been Announced, January 2024. URL <https://kotaku.com/game-industry-layoffs-how-many-2024-unity-twitch-1851155818>.

- [40] Nicole Carpenter. 2024 has already had more video game industry layoffs than all of 2023 — and it's only June, June 2024. URL <https://www.polygon.com/24177290/video-game-industry-layoffs-studio-closures-record>.
- [41] David Clearwater. What Defines Video Game Genre? Thinking about Genre Study after the Great Divide. *Loading...*, 5(8), May 2011. ISSN 1923-2691. URL <https://journals.sfu.ca/loading/index.php/loading/article/view/67>. Number: 8.
- [42] Rachel Ivy Clarke, Jin Ha Lee, and Neils Clark. Why Video Game Genres Fail: A Classificatory Analysis. *Games and Culture*, 12(5):445–465, July 2017. ISSN 1555-4120. doi: 10.1177/1555412015591900. URL <https://doi.org/10.1177/1555412015591900>. Publisher: SAGE Publications.
- [43] *Invasion - Game Manual*. 1972. URL http://archive.org/details/Invasion_1972_Magnavox_US.
- [44] Alan Emrich. MicroProse's Strategic Space Opera is Rated XXXX. In *Computer Gaming World*, volume 110, pages 92–93. 1993. URL <https://www.cgwmuseum.org/galleries/index.php?year=1993&pub=2&id=110>.
- [45] Ryan Atkinson. How Warcraft 3's modding community paved the way for League of Legends and Dota 2, March 2018. URL <https://www.pcgamesn.com/warcraft-iii/warcraft-3-mods-dota-league-of-legends>.
- [46] Christian Nutt. Gamasutra - The Valve Way: Gabe Newell And Erik Johnson Speak, October 2016. URL https://web.archive.org/web/20161026180847/http://www.gamasutra.com/view/feature/6471/the_valve_way_gabe_newell_and_.php.
- [47] Brian Crecente. League of Legends is now 10 years old. This is the story of its birth. *Washington Post*, October 2019. ISSN 0190-8286. URL <https://www.washingtonpost.com/video-games/2019/10/27/league-legends-is-now-years-old-this-is-story-its-birth/>.
- [48] Matt Cox. Spawn Point: What on earth is an auto battler? *Rock, Paper, Shotgun*, August 2019. URL <https://www.rockpapershotgun.com/what-is-an-auto-battler>.
- [49] Rick Van Krevelen. *Augmented Reality: Technologies, Applications, and Limitations*. April 2007. doi: 10.13140/RG.2.1.1874.7929.
- [50] John Werner. Catchup With Ivan Sutherland - Inventor Of The First AR Headset, 2024. URL <https://www.forbes.com/sites/johnwerner/2024/02/23/>

[catchup-with-ivan-sutherlandinventor-of-the-first-ar-headset/](#). Section: AI.

- [51] Hidayat Hidayat, Sukmawarti Sukmawarti, and Suwanto Suwanto. The application of augmented reality in elementary school education. *Research, Society and Development*, 10(3):e14910312823–e14910312823, March 2021. ISSN 2525-3409. doi: 10.33448/rsd-v10i3.12823. URL <https://rsdjournal.org/index.php/rsd/article/view/12823>. Number: 3.
- [52] Tadatsugu Morimoto, Takaomi Kobayashi, Hirohito Hirata, Koji Otani, Maki Sugimoto, Masatsugu Tsukamoto, Tomohito Yoshihara, Masaya Ueno, and Masaaiki Mawatari. XR (Extended Reality: Virtual Reality, Augmented Reality, Mixed Reality) Technology in Spine Medicine: Status Quo and Quo Vadis. *Journal of Clinical Medicine*, 11(2):470, January 2022. ISSN 2077-0383. doi: 10.3390/jcm11020470. URL <https://www.mdpi.com/2077-0383/11/2/470>. Number: 2 Publisher: Multidisciplinary Digital Publishing Institute.
- [53] Fabio Arena, Mario Collotta, Giovanni Pau, and Francesco Termine. An Overview of Augmented Reality. *Computers*, 11(2):28, February 2022. ISSN 2073-431X. doi: 10.3390/computers11020028. URL <https://www.mdpi.com/2073-431X/11/2/28>. Number: 2 Publisher: Multidisciplinary Digital Publishing Institute.
- [54] Salin Boonbrahm, Poonpong Boonbrahm, and Charlee Kaewrat. The Use of Marker-Based Augmented Reality in Space Measurement. *Procedia Manufacturing*, 42:337–343, January 2020. ISSN 2351-9789. doi: 10.1016/j.promfg.2020.02.081. URL <https://www.sciencedirect.com/science/article/pii/S2351978920306466>.
- [55] Lúcia Pombo and Margarida M. Marques. The Potential Educational Value of Mobile Augmented Reality Games: The Case of EduPARK App. *Education Sciences*, 10(10):287, October 2020. ISSN 2227-7102. doi: 10.3390/educsci10100287. URL <https://www.mdpi.com/2227-7102/10/10/287>. Number: 10 Publisher: Multidisciplinary Digital Publishing Institute.
- [56] Stephen R. Schach. *Software engineering*. Homewood, IL : Aksen Associates : Irwin, 1990. ISBN 978-0-256-08515-0. URL http://archive.org/details/softwareengineer0000scha_k3z4.
- [57] Barry Boehm. A view of 20th and 21st century software engineering. In *Proceedings of the 28th international conference on Software engineering*, pages 12–29, Shanghai China, May 2006. ACM. ISBN 978-1-59593-375-1. doi: 10.1145/1134285.1134288. URL <https://dl.acm.org/doi/10.1145/1134285.1134288>.

- [58] K. K. Aggarwal. *Software Engineering*. New Age International, 2005. ISBN 978-81-224-1638-1. Google-Books-ID: dx2C9Zkez5YC.
- [59] Eric J. Braude and Michael E. Bernstein. *Software Engineering: Modern Approaches, Second Edition*. Waveland Press, March 2016. ISBN 978-1-4786-3303-7. Google-Books-ID: kILICwAAQBAJ.
- [60] Walt Scacchi. Practices and Technologies in Computer Game Software Engineering. *IEEE Software*, 34(1):110–116, January 2017. ISSN 1937-4194. doi: 10.1109/MS.2017.20. URL https://ieeexplore.ieee.org/abstract/document/7819395?casa_token=aFaW69QOUqkAAAAA:iXHJ-mRUUka7BmSpefmFEZ9IrOKT_VmZ9FnvSI8Gv2qDsu2DqfMUsl1BIPiG_NIDRO_kCiuz. Conference Name: IEEE Software.
- [61] MALL RAJIB. *FUNDAMENTALS OF SOFTWARE ENGINEERING, FIFTH EDITION*. PHI Learning Pvt. Ltd., September 2018. ISBN 978-93-88028-03-5.
- [62] Frank Tsui, Orlando Karam, and Barbara Bernal. *Essentials of Software Engineering*. Jones & Bartlett Learning, January 2022. ISBN 978-1-284-22899-1. Google-Books-ID: IN9XEAAAQBAJ.
- [63] Christopher M. Kanode and Hisham M. Haddad. Software Engineering Challenges in Game Development. In *2009 Sixth International Conference on Information Technology: New Generations*, pages 260–265, April 2009. doi: 10.1109/ITNG.2009.74. URL https://ieeexplore.ieee.org/abstract/document/5070627?casa_token=SK2d290EQBIAAAAA:auwBbXnxjTuxoP7TGLp7ap-N-mBGmc79wg8YuTv4wFMfe4NifRKGw-kiQAOUArvmLa-Dc1Lo.
- [64] Apostolos Ampatzoglou and Ioannis Stamelos. Software engineering research for computer games: A systematic review. *Information and Software Technology*, 52(9):888–901, September 2010. ISSN 0950-5849. doi: 10.1016/j.infsof.2010.05.004. URL <https://www.sciencedirect.com/science/article/pii/S0950584910000820>.
- [65] Saiqa Aleem, Luiz Fernando Capretz, and Faheem Ahmed. Game development software engineering process life cycle: a systematic review. *Journal of Software Engineering Research and Development*, 4(1):6, November 2016. ISSN 2195-1721. doi: 10.1186/s40411-016-0032-7. URL <https://doi.org/10.1186/s40411-016-0032-7>.
- [66] Jorge Chueca, Javier Verón, Jaime Font, Francisca Pérez, and Carlos Cetina. The consolidation of game software engineering: A systematic literature review of

- software engineering for industry-scale computer games. *Information and Software Technology*, 165:107330, January 2024. ISSN 0950-5849. doi: 10.1016/j.infsof.2023.107330. URL <https://www.sciencedirect.com/science/article/pii/S0950584923001854>.
- [67] Andrew Rollings and Dave Morris. *Game Architecture and Design with Cdrom*. Coriolis Group Books, USA, October 1999. ISBN 978-1-57610-425-5.
- [68] Jessica Mulligan and Bridgette Patrovsky. *Developing Online Games: An Insider's Guide*. New Riders, 2003. ISBN 978-1-59273-000-1. Google-Books-ID: mvuUPxXyB7AC.
- [69] Ann Latham Cudworth. *Virtual World Design*. CRC Press, July 2014. ISBN 978-1-4665-7961-3. Google-Books-ID: U9vhAwAAQBAJ.
- [70] Diego Avila Pesantez and Luis Rivera. Approaches for Serious Game Design: A Systematic Literature Review. *Computers in Education Journal*, 8, September 2017.
- [71] Petros Lameras, Sylvester Arnab, Ian Dunwell, Craig Stewart, Samantha Clarke, and Panagiotis Petridis. Essential features of serious games design in higher education: Linking learning attributes to game mechanics. *British Journal of Educational Technology*, 48(4):972–994, 2017. ISSN 1467-8535. doi: 10.1111/bjet.12467. URL <https://onlinelibrary.wiley.com/doi/abs/10.1111/bjet.12467>. _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/bjet.12467>.
- [72] Laura Romero Rodríguez. Engaging future engineers: the case study of a serious game implementation. *Education and Information Technologies*, 28(3): 2909–2939, March 2023. ISSN 1573-7608. doi: 10.1007/s10639-022-11279-y. URL <https://doi.org/10.1007/s10639-022-11279-y>.
- [73] Jussi Kasurinen, Andrey Maglyas, and Kari Smolander. Is Requirements Engineering Useless in Game Development? In Camille Salinesi and Inge van de Weerd, editors, *Requirements Engineering: Foundation for Software Quality*, pages 1–16, Cham, 2014. Springer International Publishing. ISBN 978-3-319-05843-6. doi: 10.1007/978-3-319-05843-6_1.
- [74] Vaishnavi Patil, Sanjana Panicker, and Maitreyi Kv. Use of Agile Methodology for Mobile Applications. *International Journal of Latest Technology in Engineering, Management & Applied Science*, V, November 2016.
- [75] Shakira Banu Kaleel and S sowjanya Harishankar. Applying Agile Methodology in Mobile Software Engineering: Android Application Development and

- its Challenges. report, Toronto Metropolitan University, September 2022. URL https://rshare.library.torontomu.ca/articles/report/Applying_Agile_Methodology_in_Mobile_Software_Engineering_Android_Application_Development_and_its_Challenges/14637270/2.
- [76] Samar Alsaqqa, Samer Sawalha, and Heba Abdel-Nabi. Agile Software Development: Methodologies and Trends. *International Journal of Interactive Mobile Technologies (iJIM)*, 14(11):246, July 2020. ISSN 1865-7923. doi: 10.3991/ijim.v14i11.13269. URL <https://online-journals.org/index.php/i-jim/article/view/13269>.
- [77] Patricio Letelier. M todolog as  giles para el desarrollo de software: eXtreme Programming (XP), April 2006. URL http://www.cyta.com.ar/ta0502/b_v5n2a1.htm. Publisher: T cnica Administrativa issn:1666-1680.
- [78] Rajat B Wakode, Laukik P Raut, and Pravin Talmale. Overview on Kanban Methodology and its Implementation. 3(02), 2015.
- [79] Hamzah Alaidaros, Mazni Omar, and Rohaida Romli. The state of the art of agile kanban method: challenges and opportunities. *Independent Journal of Management & Production*, 12(8):2535–2550, December 2021. ISSN 2236-269X. doi: 10.14807/ijmp.v12i8.1482. URL <http://paulorodrigues.pro.br/ojs/ijmp/index.php/ijmp/article/view/1482>. Number: 8.
- [80] Ken Schwaber and Jeff Sutherland. Scrum Guide | Scrum Guides, November 2020. URL <https://scrumguides.org/scrum-guide.html>.
- [81] Alan Moran. Agile Risk Management. In Alan Moran, editor, *Agile Risk Management*, pages 33–60. Springer International Publishing, Cham, 2014. ISBN 978-3-319-05008-9. doi: 10.1007/978-3-319-05008-9_3. URL https://doi.org/10.1007/978-3-319-05008-9_3.
- [82] Muhammad Hammad, Irum Inayat, and Maryam Zahid. Risk Management in Agile Software Development: A Survey. In *2019 International Conference on Frontiers of Information Technology (FIT)*, pages 162–1624, December 2019. doi: 10.1109/FIT47737.2019.00039. URL <https://ieeexplore.ieee.org/abstract/document/8991647>. ISSN: 2334-3141.
- [83] Marc Schmalz, Aimee Finn, and Hazel Taylor. Risk Management in Video Game Development Projects. pages 4325–4334. IEEE Computer Society, January 2014. ISBN 978-1-4799-2504-9. doi: 10.1109/HICSS.2014.534. URL <https://www.computer.org/csdl/proceedings-article/hicss/2014/2504e325/120mNAoUTxj>. ISSN: 1530-1605.

- [84] Linda K Kaye and Jo Bryce. Putting the fun factor into gaming: The influence of social contexts on the experiences of playing videogames. *International Journal of Internet Science*, 7(1):24–38, August 2012. ISSN 1662-5544.
- [85] Ministerio de Trabajo y Economía Social. Resolución de 13 de julio de 2023, de la Dirección General de Trabajo, por la que se registra y publica el XVIII Convenio colectivo estatal de empresas de consultoría, tecnologías de la información y estudios de mercado y de la opinión pública, July 2023. URL [https://www.boe.es/eli/es/res/2023/07/13/\(5\).](https://www.boe.es/eli/es/res/2023/07/13/(5).)
- [86] Unity Technologies. Unity plans and pricing, 2024. URL <https://unity.com/products>.
- [87] Agencia Tributaria. Agencia Tributaria: 1. Por coeficientes de amortización lineal, July 2024. URL <https://sede.agenciatributaria.gob.es/Sede/ayuda/manuales-videos-folletos/manuales-practicos/irpf-2023/c07-rendimientos-actividades-economicas-estimacion-directa/fase-1-determinacion-rendimiento-neto/amortizaciones-dotaciones-ejercicio-fiscal-requisitos-generales/coeficientes-amortizacion-lineal.html>.
- [88] Colby N. Leider. *Digital Audio Workstation*. McGraw-Hill, Inc., USA, 1 edition, May 2004. ISBN 978-0-07-142286-4.
- [89] Keith Stuart. Game developers furious as Unity Engine announces new fees. *The Guardian*, September 2023. ISSN 0261-3077. URL <https://www.theguardian.com/games/2023/sep/12/unity-engine-fees-backlash-response>.
- [90] Darkfrost. Unity silently removed their Github repo to track license changes, then updated their license to remove the clause that lets you use the TOS from the version you shipped with, then insists games already shipped need to pay the new fees., September 2023. URL www.reddit.com/r/gamedev/comments/16hnibp/unity_silently_removed_their_github_repo_to_track/.
- [91] Tom Gerken. Unity: Gaming boss quits after furious pricing backlash. October 2023. URL <https://www.bbc.com/news/technology-67063855>.
- [92] Vítor M. Costa. The Definition of Design by Subtraction, June 2022. URL <https://medium.com/super-jump/the-definition-of-design-by-subtraction-a051e127f171>.