UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática

Universitat Politècnica de València

Desarrollo de un juego multiplataforma con realidad aumentada en Unity

Proyecto Final de Carrera

Ingeniería Informática

Autor: Adrián Ciborro Montes

Director: Juan Carlos Ruiz García

6 Septiembre 2015

Índice

1. Introducción **7**

1.1 Motivación 7

1.2 Desafíos 9

1.3 Estructura de la memoria 10

2. Estado del arte **11**

2.1 Videojuegos 12

2.1.1 Factores de diseño 13

2.1.2 Modding 16

2.2 Dispositivos móviles 17

2.3 Plataforma de desarrollo 18

2.4 Realidad aumentada 20

2.5 Tipo de juego para el dispositivo móvil 22

3. Análisis de requisitos **23**

3.1 Glosario 23

3.2 Objetivos 24

3.3 Casos de uso 26

4. Diseño e implementación **31**

4.1 Desarrollo con Unity 31

4.2 Mecánicas de juego 33

4.3 Interfaz gráfica 36

4.4 Realidad aumentada 41

4.5 Comunicación 43

5. Test **49**

6. Conclusiones **55**

7. Bibliografía **57**

Lista de figuras 1

Figura 1. Age of Empires II Map Editor.

Figura 2. Ciclo de vida de un producto software.

Figura 3. Campus Party Londres 2013, Oculus Rift y Unity.

Figura 4. Trine, tiempo real, acción, plataformas, puzzle.

Figura 5. Hearthstone, Blizzard.

Figura 6. Unity Logo.

Figura 7. Continuo Realidad-Virtualidad.

Figura 8. Augmented Reality Logo.

Figura 9. Comparativa tabletas gama alta.

Figura 10. Caso de uso inicio de sesión.

Figura 11. Caso de uso búsqueda de usuarios.

Figura 12. Caso de uso desafiar usuario.

Figura 13. Caso de uso realizar acción.

Figura 14. Máquina de estados de MonoBehaviour.

Figura 15. Diagrama de clases del sistema.

Figura 16. Diagrama de clases mecánicas de juego.

Figura 17. Máquina de estados de animaciones.

Figura 18. Jerarquía de controladores.

Figura 19. Interfaz gráfica de juego.

Figura 20. Núcleo del sistema de Raycast.

Figura 21. Menú de inicio de sesión.

Figura 22. Menú principal.

Figura 23. Menú de búsqueda.

Lista de figuras 2

Figura 24. Diagrama de clases interfaz gráfica.

Figura 25. Diagrama a alto nivel de Vuforia.

Figura 26. Diagrama de clases mensaje de red.

Figura 27. Task Scheduler.

Figura 28. Broker pattern.

Figura 29. Captura en PC.

Figura 30. Captura en tableta.

Figura 31. Captura de juego.

1 Introducción

La generación a la que pertenezco ha vivido muy de cerca el nacimiento de los videojuegos, pero desde siempre ha existido el concepto de juego y no hay rincón en el mundo donde no se juegue.

Este proyecto no hubiera sido posible sin la motivación y diversión que me han dado, pues si los juegos tienen una característica principal esa es la diversión y como dice Ian Livingstone, escritor y fundador de Games Workshop, son un verdadero poder para hacer el bien.

Los avances tecnológicos llegan a cualquier rincón de la vida humana y los juegos no son una excepción. Los videojuegos se han convertido en el sector audiovisual que más factura a nivel mundial y que atrae a un amplio abanico de perfiles profesionales, tanto técnicos como artísticos.

Como todo proyecto software lo he desarrollado iterando sobre las fases de análisis de requisitos, diseño, implementación y test. Una fase muy importante a la hora de desarrollar un videojuego es la fase de diseño. En esta fase se toman una serie de decisiones que son muy subjetivas y que tienen una gran importancia a la hora de que el producto sea competitivo y tenga éxito. Por este motivo los procesos de seleccionar objetivos y requisitos, y posteriormente diseñarlos, son las que más amplitud tienen.

1.1 Motivación

La idea de realizar un videojuego como proyecto final de carrera surge de diferentes lugares.

Por una parte siempre me han gustado mucho los juegos RTS (*real-time strategy games*) y la mayoría de ellos incluían un motor de generación de mapas con herramientas gráficas y programación con un lenguaje de scripting con el cual he creado mods de esos juegos. El proceso de trabajar con recursos artísticos y técnicos para construir un entorno interactivo es algo que siempre me ha gustado. De hecho, muchos de los profesionales que actualmente trabajan en el sector como programadores, diseñadores de proyecto, de personajes o del mundo empezaron su carrera haciendo modding sobre este tipo de videojuegos ya existentes en el mercado.

Figura 1. Age of Empires II Map Editor.

Por otra parte en el verano del 2012 hice un curso ofertado por el centro de formación permanente de la UPV de desarrollo de juegos en Unity. Durante el curso aprendí los conceptos básicos de la informática gráfica moderna (mallas, texturas, materiales, shaders...), la arquitectura de los motores y el lenguaje C# (lenguaje de scripting usado en Unity).

Cuando terminé el curso estaba muy interesado en conocer mejor las arquitecturas de motores de juegos. Por ello me compré el libro Game Engine Architecture [1] y empecé a estudiar los distintos subsistemas de Unity (renderizado, animación, IA, etc.).

El empujón definitivo a la idea de crear un videojuego como proyecto final de carrera surgió cuando me fui de Erasmus a Dinamarca el curso 2013-2014. Allí cursé varias asignaturas que influyeron en la elección del proyecto.

La asignatura de programación en C# usando .NET me dio la experiencia e información necesaria sobre el lenguaje y sobre otras tecnologías de Microsoft. Por otra parte con la asignatura de diseño de juegos aprendí algunos conceptos básicos de diseño de reglas. Otra asignatura que también influyó fue programación en Android, en esta asignatura usé una tableta para hacer pruebas sobre las diferentes aplicaciones que íbamos implementando en el curso, así como pruebas más específicas que hacía yo con Unity.

Con toda esta nueva información y con bastante tiempo libre empecé a hacer pruebas sobre qué podía y qué no podía hacer yo solo con Unity como proyecto.

Un día haciendo pruebas con la realidad aumentada surgió la idea de por dónde enfocarlo. No necesitaba muchos recursos gráficos ya que la mayor parte del videojuego se jugaba usando el mundo real como base, quitando la mayor limitación que tiene un programador solo ante este tipo de proyectos, el arte.

1 2 Desafíos

Los desafíos que han surgido a lo largo del desarrollo del proyecto han sido muy diferentes y todos ellos han ido ligados a los objetivos que debía de cumplir el videojuego, siendo estos **multiplataforma**, **multijugador** y usar **realidad aumentada**. Estos objetivos se analizarán en más profundidad en el apartado de análisis de requisitos.

La elección de los tipos de dispositivos a los que va a ir enfocado el videojuego, tratando con el problema de la fragmentación, la integración de la realidad aumentada en el proyecto teniendo en cuenta sus limitaciones computacionales o la comunicación entre jugadores son algunos de los más importantes.

El primer reto fue encontrar una plataforma de desarrollo que permitiera implementar el videojuego en el mayor número de dispositivos posible, abstrayendo gran parte del problema de la **fragmentación** [20], existe un gran número de dispositivos en los que sería interesante implementar el videojuego.

Otro reto importante fue resolver cómo integrar la realidad aumentada en el conjunto del proyecto. Una característica muy importante que tiene que cumplir la solución de realidad aumentada elegida es que sea portable para diferentes dispositivos.

Todo videojuego necesita recursos gráficos para su desarrollo, por tanto fue necesario buscarlos para el prototipo. Al hacer uso de la realidad aumentada, la cantidad de modelos y texturas que se necesita es mucho menor que un videojuego que cree su mundo totalmente virtual. A pesar de esto es difícil encontrar recursos que estén dentro de una misma temática y encajen bien en el juego.

Otro desafío es el desarrollo de la comunicación por red para que los clientes puedan jugar entre sí. La búsqueda de jugadores o poder atravesar los NAT [6] (*Network Address Translation*) de los routers sin problemas son algunos de los requisitos que el sistema de red debía de tener.

Teniendo en cuenta todos estos desafíos, el más importante ha sido la integración del diseño de reglas del juego al mismo tiempo que se tiene en cuenta las limitaciones de las tecnologías que se está usando. Todo ello engloba una serie de decisiones diseño que hacen que el videojuego vaya cambiando a lo largo del desarrollo.

1 3 Estructura de la memoria

La estructura de esta memoria sigue un modelo similar al de cualquier proyecto tecnológico.

En primer lugar se presentará el estado de las diferentes tecnologías usadas, así como las decisiones que llevaron a elegirlas para el tipo concreto de juego. También se explicarán las diferentes tecnologías usadas como la realidad aumentada o el mundo de los dispositivos móviles y como estos ofrecen muchas oportunidades de éxito a los pequeños equipos de desarrollo.

Una vez seleccionadas las tecnologías y el tipo de juego se describirán cada una de las diferentes etapas que tiene cualquier proyecto software, analizar los requisitos del sistema, realizar el diseño que hará de guía para la implementación de las diferentes versiones del producto y por último la fase de testeo de las versiones en la que se pueden obtener resultados que cambien el diseño e implementación de futuras versiones para así volver a cerrar un ciclo de iteración del producto.

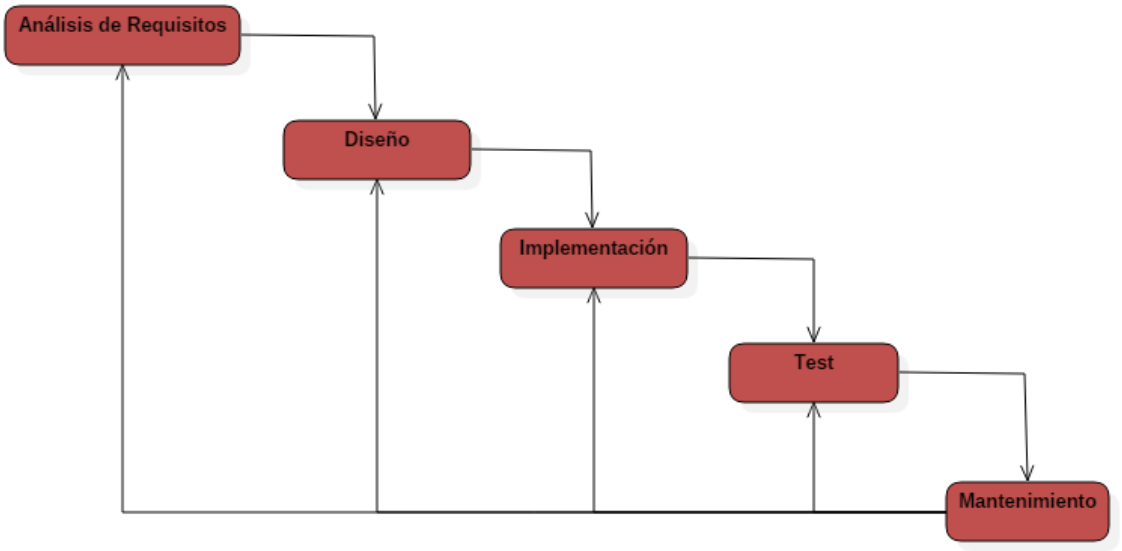

Figura 2. Ciclo de vida de un producto software.

Para finalizar se hará una conclusión del proyecto donde se analizará si las tecnologías usadas han sido las adecuadas y en qué aspectos es posible mejorar el ciclo de vida y las iteraciones para alcanzar un producto de mayor calidad y con viabilidad de mercado.

2 Estado del arte

En esta sección se analizarán las diferentes tecnologías que se han seleccionado para la realización del proyecto, el estado en el que se encuentran y como se integran todas ellas en él.

Se empezará hablando sobre el sector de los videojuegos, contextualizando como ha llegado a ser la industria que es hoy en día y como el avance de la tecnología ha influido en el sector tanto en la parte técnica como artística. Se hará un análisis sobre las diferentes decisiones que influyen en que un videojuego tenga éxito, decisiones como la forma de gestión del tiempo del juego, la elección del género y subgéneros a los que pertenecerá, los modelos de negocio o la localización del producto.

A continuación se hablará sobre las oportunidades que ofrecen los nuevos dispositivos móviles en el desarrollo de videojuegos, de manera muy similar a como lo hacen las consolas portátiles, poniendo de relieve las oportunidades que pueden llegar a ofrecer para los equipos de desarrollo pequeños.

Después se darán argumentos sobre la decisión de elegir Unity como motor de videojuegos para resolver el problema de la fragmentación de dispositivos, repasando sus características que hacen que sea una plataforma de desarrollo interesante.

Constantemente aparecen nuevas tecnologías que buscan crear nuevas experiencias en los usuarios. Una de ellas es la realidad aumentada, sus diferentes aplicaciones y cómo es usada en un amplio número de sectores serán los contenidos del apartado de realidad aumentada.

Por último se describirá el tipo de videojuego elegido, los argumentos de diseño que han llevado a elegir las tecnologías y los objetivos de diseño que debe de cumplir.

2 1 Videojuegos

El sector de los videojuegos se encuentra en constante crecimiento, siendo ahora mismo una industria que se estima que facturará en 2015 alrededor de los 80.000 millones de dólares a nivel mundial. Según AEVI [10], la *Asociación Española de Videojuegos*, la facturación en España del sector en 2013 fue de 762 millones de euros contando videojuegos, consolas y periféricos. Esto supone unos ingresos muy superiores al resto de sectores audiovisuales.

Con estos datos presentes y teniendo en cuenta la gran expansión de las empresas tecnológicas en los últimos años, se puede afirmar que es un sector muy globalizado, en el que cada vez entran en juego más actores. Se trata de un sector multidisciplinar que engloba muchos perfiles profesionales (programadores, artistas, diseñadores, etc.) y que atrae a otros como músicos o escritores.

En el contexto actual de crecimiento están empezando a surgir las primeras escuelas de enseñanza de desarrollo de videojuegos. En sus inicios el desarrollo se hacía casi totalmente por parte de programadores, las limitaciones gráficas de las plataformas hacían que la parte artística del videojuego estuviera muy limitada, por ello el mayor peso residía en el diseño e implementación de las reglas. A medida que la informática gráfica fue avanzando el peso del equipo artístico fue aumentando, por ello los equipos de desarrollo también lo hicieron, creando así empresas más grandes. Con este modelo las empresas eran las que directamente se ocupaban de la enseñanza, haciendo muy difícil el acceso a nuevas personas al sector.

A lo largo de la corta historia del sector han ido apareciendo nuevas tecnologías que han aumentado su proyección de crecimiento.

Nuevas tecnologías de distribución como Steam, la amplia gama de dispositivos móviles y tabletas, o nuevos periféricos como el famoso Oculus Rift son algunos ejemplos del rumbo que está tomando.

Un rumbo en el que la gama de plataformas para desarrollar es muy amplia, los perfiles de los jugadores son muy diferentes y cada vez más gente entra en el sector con grupos de desarrollo pequeños.

Figura 3. Campus Party Londres 2013, Oculus Rift y Unity.

Con la masificación de los dispositivos móviles se ha abierto un gran mercado para que los videojuegos ocupen cualquier rincón de ocio disponible, para jugar ya no es necesario un ordenador o una consola, cualquiera en cualquier lugar puede hacerlo.

Pero todo este avance tiene un coste, el tamaño de los proyectos es cada vez más grande por lo que los equipos de desarrollo se vuelven más complejos. Para solucionar esto se está apostando por la creación de motores de videojuegos como producto final en sí mismo, que sirven de middleware de desarrollo y de abstracción para el proyecto. En este momento estamos en un punto en el que las alternativas de motores son muy variadas y ofrecen muchas características de abstracción que hacen que merezca la pena usarlos frente a comenzar el desarrollo desde cero.

Cada vez los videojuegos son más sociales, interactuar con otros jugadores añade experiencias que no son posibles de lograr con una inteligencia artificial. Por ello es muy importante diseñar sistemas distribuidos que tengan características como la escalabilidad, usabilidad o portabilidad. Desde la llegada de internet la explosión del uso y desarrollo de estos sistemas ha sido enorme por lo que no cabe imaginar otro escenario que no sea que sigan creciendo. Dada esta gran diversificación es cada vez más importante el uso o desarrollo de middlewares que permitan comunicar a los usuarios.

Gracias a todas estas tendencias positivas hoy en día es posible dedicarse al desarrollo de videojuegos, pero ahora más que nunca se requiere una gran formación en informática, programación y tecnologías en general.

2 1 1 Factores de diseño

A medida que la industria avanzaba, el mercado se abría y el número de usuarios aumentaba, fueron definiéndose las formas de clasificar los videojuegos, los géneros.

Existen muchos estereotipos sobre los gustos de jugadores de diferentes países. Estos estereotipos, aunque exagerados, siguen unos patrones que pueden jugar a favor si se toman las decisiones de diseño adecuadas. Decisiones como la temática, la gestión del tiempo, las mecánicas del juego, el modelo de negocio o la localización del producto son factores claves para que el producto tenga éxito. Por tanto es muy importante saber cuál es el mercado al que está dirigido.

La clasificación más general que se puede hacer de videojuegos es separarlos según como se administra el flujo de tiempo del juego. De aquí nacen 2 categorías, los juegos en tiempo real y los juegos por turnos.

Los videojuegos por turnos han estado muy presentes desde el comienzo. A diferencia de los juegos en tiempo real, el flujo de juego está dividido en turnos bien definidos en los que los jugadores realizan sus estrategias y acciones.

A pesar de la aparente reducción de complejidad que puede presentar diseñar un videojuego por turnos frente a uno en tiempo real no hay una relación de aparición del uno frente al otro a lo largo de la historia.

Estos juegos presentan un tiempo de toma de decisiones fijo y en el que una acción tiene una alta relevancia en la partida, por tanto, la pericia a la hora de crear la estrategia y administrarla en el tiempo es lo que suele decantar las partidas.

La clasificación mediante géneros se hace en función de las mecánicas que son características en el videojuego. La mayoría de juegos pertenece un género claro (juegos de lucha, arcade, simulaciones, plataformas, etc.) pero también poseen características de otros subgéneros.

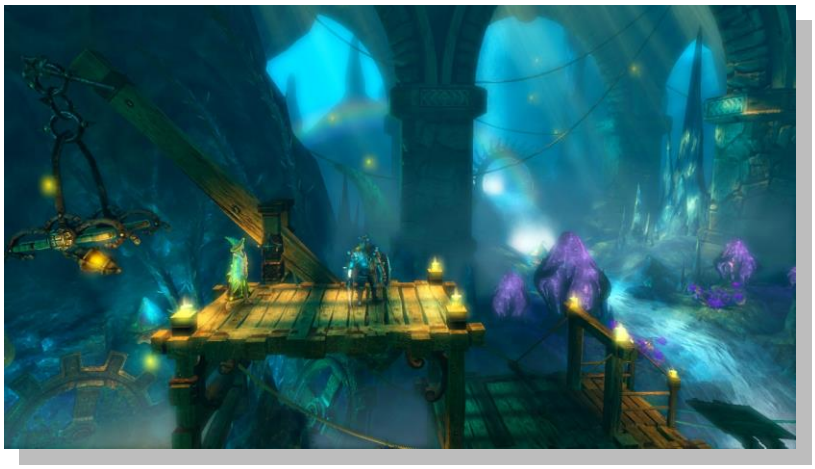

Figura 4. Trine, tiempo real, acción, plataformas, puzzle.

A la hora de la elección del género hay que tener en cuenta su masificación en el mercado. Géneros como los MMORPG (*massively multiplayer online role-playing game*) o videojuegos de deportes están muy masificados. Por contra clásicos como los RTS no tienen ningún título actual muy destacable.

Otra decisión importante que hay que tomar es elegir entre un juego 2D o 3D. A pesar de que los videojuegos en sus inicios eran todos 2D y se dio un gran paso hacia los 3D, este es un género muy seguido en la actualidad. Esta elección implica un importante cambio en las herramientas software que se emplean para el desarrollo del arte.

Una decisión muy importante a la hora de diseñar el videojuego es la elección del **modelo de negocio**. Existen tres modelos de negocio dominantes en el sector, aunque suelen combinarse para obtener mejores beneficios.

- **Pay to Play** (pagar por jugar) [12]. Es el modelo más clásico y extendido, el jugador compra el juego una vez y tiene acceso del producto al completo.

- **F2P** (*free to play*) [13]. Es un modelo que cada vez se usa más (principalmente con juegos casuales). El jugador adquiere el juego gratis y paga por expansiones, modificaciones o nuevas funcionalidades.

- **Publicidad**. El juego contiene publicidad por la que el desarrollador obtiene ingresos. La publicidad puede estar dentro del juego, en los menús principales, al inicio...

Según el mercado al que nos dirijamos es importante tener en cuenta los modelos de negocio. Por ejemplo el mercado norteamericano es muy grande y rico, y tiene el mayor número de títulos de diferentes géneros, pero no es el único mercado al que hay que mirar.

Mercados como Rusia o China han sido hasta hace poco muy cerrados y por tanto la piratería era y sigue siendo un factor muy importante a la hora de elegir el modelo de negocio en estos países. En estos países el F2P o la publicidad son modelos que pueden funcionar mejor que la compra directa del videojuego.

Otro factor decisivo y que cada vez toma un papel más importante en el diseño es la **localización** [17]. Al abrirse el sector a todo el mundo se pone de relieve las zonas en las que las nuevas tecnologías no han llegado con tanta fuerza o la apuesta por la docencia de otras lenguas es muy pequeña. Por tanto hay zonas en las que la salida del videojuego únicamente en inglés hace que la comercialización pueda ser un fracaso.

Pero no solo la variedad de lenguas es importante en la localización. Factores como la zona horaria, el sistema de medidas o la opción de pago con la moneda local son muy importantes a la hora de localizar el producto.

Un ejemplo de la toma de decisiones correcta es el videojuego Hearthstone de Blizzard. Su género es de cartas coleccionables, un género poco masificado. La temática gira en torno a otros de sus videojuegos ya desarrollados con éxito. Su modelo de negocio es F2P con microcompras de contenido digital como cartas o extensiones y su localización incluye inglés, ruso, chino y muchas lenguas europeas además de permitir el pago con diferentes divisas.

Figura 5. Hearthstone, Blizzard.

2.1.2 Modding

Modding es la técnica mediante la cual se extiende el contenido de un videojuego [14]. Desde siempre las empresas de videojuegos para ordenador han dado cierto grado de personalización y modificación en sus juegos, permitiendo a los usuarios participar en una pequeña parte del ciclo de vida como algo más que jugadores.

En los inicios del sector estas comunidades de usuarios eran muy reducidas y actuaban sin la ayuda de las compañías de videojuegos. En muchos casos esto solía traer problemas legales a algunos modders.

Con el paso del tiempo las compañías han visto como el hecho de crear comunidades que no sean exclusivamente de jugadores alarga enormemente la vida del producto y su calidad, y crea una gran variedad de oportunidades para otros usuarios.

Debido a esto cada vez se apuesta más por publicar herramientas de desarrollo, herramientas para que la comunidad con menos conocimiento técnico pueda hacer mods. Algunos ejemplos de estas herramientas son plugins de editores gráficos como 3ds Max o Maya, o programas para exportar los recursos del mismo videojuego para modificarlos y posteriormente reemplazarlos.

Dos géneros han sido los principales pioneros del modding, los shooters y los RTS. En ambos géneros las partidas se juegan en un mapa acotado que los modders pueden construir. Hay muchos ejemplos de mods que han triunfado y después han dado lugar a nuevos géneros y videojuegos separándose de la plataforma original, géneros como el famoso MOBA (*multiplayer online battle arena*).

Esta característica es un factor que hay que tener en cuenta en el desarrollo ya que si se apuesta lo suficiente por la comunidad que pueda crear, la vida y calidad del producto crecen enormemente.

2 Dispositivos móviles

La elección de los dispositivos en los que se jugará al juego es una de las que más influyen en el éxito del videojuego, se han creado muchos videojuegos muy buenos que no han cumplido sus objetivos por elegir plataformas de juego erróneas.

La naturaleza del juego hace que sea necesario un dispositivo que pueda ser transportado con libertad. Se trata de un juego casual que puede ser jugado en cualquier parte sin necesidad de que el dispositivo tenga que estar conectado a la red eléctrica. Además es obligatorio que el dispositivo tenga una conexión a internet estable.

Dados estos requisitos se barajan como plataformas los dispositivos móviles, *smartphones* y tabletas. Su rápida expansión en los últimos años hace que sean unos dispositivos muy interesantes como plataforma de juego. El modelo es parecido al que tienen las consolas portátiles como la PS Vita o la antigua Game Boy, jugar donde quieras cuando quieras. Sin embargo los *smartphones* y tabletas tienen una ventaja frente a estas plataformas, su difusión. Han conseguido llegar a muchos lugares del mundo con precios competitivos donde las consolas portátiles no lo han hecho.

Una característica importante a la hora de elegir el dispositivo es el sistema operativo que utiliza. Actualmente los sistemas operativos Android e iOS abarcan más del 90% de la cuota de mercado, siendo la cuota del resto marginal.

Otro aspecto importante para elegir plataforma es el sistema de distribución del producto. Android e iOS tienen tiendas digitales donde poner a la venta tus aplicaciones a cambio de un porcentaje de los beneficios (Play Store y App Store respectivamente). Gracias a estas plataformas de distribución se puede delegar esta fase del desarrollo por un precio razonable, centrando esfuerzos en el desarrollo del videojuego. Además ambas tiendas tienen mucho tráfico de aplicaciones y descargas, haciendo inmediata su elección como plataforma de distribución.

Con la gran variedad de modelos de smartphones y tabletas que existen aparece el problema de la **fragmentación** [20]. Tanto Android como iOS son sistemas operativos muy fragmentados, son sistemas operativos que se usan en una gran variedad de modelos hardware y tienen un gran número de versiones del propio sistema operativo, por ello esta es una barrera que hay que vencer. Como se analizará más adelante en el apartado multiplataforma del análisis de requisitos, el tamaño de la pantalla y la calidad de la cámara de video son además algunas de las características que determinaran la elección más concreta de los dispositivos finales.

Así pues debido al problema de la fragmentación hay que elegir una plataforma de desarrollo que lo resuelva y no sea muy costosa para el desarrollo. En los siguientes apartados se analizará la plataforma elegida y sus características.

2 3 Plataforma de desarrollo

Una de las decisiones y desafíos iniciales fue elegir una plataforma de desarrollo que permitiera desarrollar el videojuego para el mayor número de dispositivos posibles y hacer frente al problema de la **fragmentación** [20]. Además, era necesario que la plataforma tuviera una curva de aprendizaje rápida para adaptar el proyecto a las fechas.

Muchas plataformas de desarrollo para dispositivos móviles hacen frente al problema de la fragmentación y la portabilidad para diferentes sistemas operativos. Un ejemplo es Titanium Appcelerator, su IDE permite implementar aplicaciones usando JavaScript que luego son traducidas a código nativo dependiendo del dispositivo.

Pero este tipo de plataformas tiene un problema muy ligado al desarrollo de videojuegos, carecen de un motor de renderizado potente que permita crear escenas e interfaces gráficas más complejas. Por ello se buscó una solución más estándar en el sector de los videojuegos que cumpliera los objetivos iniciales. La plataforma elegida fue Unity [8].

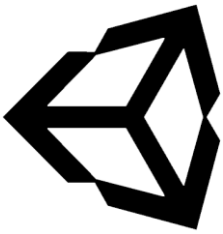

Figura 6. Unity Logo.

Unity es un motor que permite crear videojuegos para más de 20 plataformas diferentes, entre ellas están Android, iOS, Playstation 3, XBox 360, PC, etc.

El hecho de que este motor se haya convertido en el favorito de la mayoría de equipos de desarrollo pequeños o indie es resultado de un proceso que en Unity Technologies llaman **democratización**.

Este proceso busca extender y hacer más accesibles las herramientas necesarias para el desarrollo de un videojuego a cualquier persona o equipo. Un claro ejemplo de este modelo se puede ver en la versión 5 de Unity. En esta versión la edición personal y profesional tienen las mismas características, con la única diferencia de que si tu aplicación supera un umbral de beneficios debes de comprar la edición profesional. Esto hace que las inversiones iniciales de los equipos de desarrollo sean más pequeñas, centrándose más en el diseño y la creación del videojuego, abstrayendo gran parte de la tecnología.

La primera versión de Unity fue lanzada en 2005 y desde entonces no ha hecho más que crecer. Con cada versión fueron añadiendo subsistemas típicos de los motores de videojuegos punteros del mercado. Algunos ejemplos son el sistema de partículas (Shuriken), el sistema de animación (Mecanim), sombras reales en dispositivos móviles, el nuevo sistema de diseño de interfaces gráficas, la incorporación del motor físico PhysX de Nvidia, etc.

Unity ofrece tres lenguajes de scripting, Boo, Javascript y C#. El más usado de los tres es C# ya que es el único orientado a objetos. El scripting está construido sobre Mono, la implementación de código abierto del CLR de Microsoft.

Todos estos sistemas siguen la misma filosofía que el motor, *"crear una vez, implementar en todas partes"*. Unity permite que el mismo proyecto pueda ser implementado en todas las plataformas disponibles haciendo que la mayor parte del proyecto sea independiente de la plataforma de destino. Normalmente Unity es usado junto con software de modelado como 3ds max, Maya, Zbrush o Blender, y con editores gráficos como Gimp o Photoshop para la creación de los recursos gráficos.

Por otra parte Unity Technologies ofrece una serie de servicios que complementan el desarrollo de un videojuego. Algunos de estos servicios son la Unity Cloud Build (una nube donde se actualizan las builds en las que estás trabajando y se distribuyen al equipo de desarrollo), Unity Analitics (servicio analítico de datos de mercado de tu proyecto) o Unity Ads (servicio que da al desarrollador la opción de poner publicidad en el videojuego).

El más usado de todos estos servicios es la famosa Asset Store. Se lanzó en 2010 y ofrece una tienda de recursos donde hay multitud de categorías como scripts, modelos, texturas, plugins, animaciones... Es un lugar perfecto para empezar a buscar recursos para tu prototipo. Cualquier persona o equipo puede poner recursos a la venta en el Asset Store, además ofrece a los artistas un lugar donde vender sus trabajos y darse a conocer.

Todos estos servicios están construidos sobre una comunidad muy activa de usuarios en los foros oficiales de Unity y sobretodo en Unity Answers, un servicio web muy parecido a la famosa web Stack Overflow.

La red social creada alrededor de Unity realiza una conferencia anual sobre el motor y sobre los proyectos desarrollados en él llamada Unite. En esta conferencia suelen haber ponentes de muchos perfiles, equipos de desarrollo explicando las arquitecturas de sus videojuegos, anuncios de futuras características de Unity, experiencias de desarrolladores autónomos...

Otra parte muy importante del modelo de negocio de Unity, y de cualquier framework de desarrollo en general, es la política de enseñanza y aprendizaje que tienen. En su página web hay multitud de tutoriales muy sencillos de todos los subsistemas del motor acompañados con un video. Además ofrecen cursos de aprendizaje en vivo a través de YouTube gratuitos muy interesantes que cubren diferentes aspectos del motor. Todo este conjunto de tutoriales y videos está respaldado por una gran referencia de la API del motor, traducida a varios idiomas y con pequeños ejemplos en cada sección.

En conclusión, Unity es un motor joven con mucho potencial, resuelve el problema de la fragmentación y portabilidad a muchos dispositivos móviles y cuenta con un potente sistema de renderizado que es necesario para el desarrollo del videojuego.

2 4 Realidad aumentada

Todo videojuego busca hacerse un hueco en el sector entre un perfil concreto de jugador. En el mercado existen muchos títulos que compiten con los mismos perfiles pero que buscan diferenciarse unos de otros. Con esta idea en mente hay que buscar algún tipo de mecánica de juego o tecnología que permita añadirle valor al videojuego.

La elección de la realidad aumentada viene motivada porque no existen muchos títulos que la utilicen y ofrece una interacción interesante entre el jugador, el mundo real y el mundo virtual. Además se encuentra en un estado de vida temprano aunque existen varios frameworks de desarrollo interesantes.

La **realidad aumentada** es una tecnología que consiste en combinar el mundo real captado por un dispositivo de captura digital de video y un mundo virtual diseñado para ofrecer nueva funcionalidad de manera interactiva en tiempo real [7]. El término fue acuñado en 1992 por Tom Caudell y David Mizell.

Esta combinación de espacios diferentes se hace a partir del reconocimiento y procesamiento de patrones que se encuentran en el mundo real, ya sean bidimensionales o tridimensionales. Con esta información se superpone el entorno virtual sobre el entorno real.

En 1994 Paul Milgram y Fumio Kishino definieron el continuo realidad-virtualidad, un espacio en el que se puede representar cualquier entorno real o virtual.

Figura 7. Continuo Realidad-Virtualidad.

Es una tecnología que junto con la realidad virtual promete cambiar la forma en la que interactuamos con el mundo pero desde sus inicios ha tenido un gran problema, le ha faltado una estandarización para su desarrollo en las pasadas y actuales tecnologías hardware.

Al ver el potencial de esta tecnología muchas empresas han creado de kits de herramientas de desarrollo para sus propias plataformas. Algunos ejemplos son FLARToolkit de Adobe Flash o Vuforia de Qualcomm [9].

Con el fin de estandarizar y hacer que se expandiera esta tecnología, se fundó el grupo AugmentedReality.org, el cual en 2009 creó el logo de la realidad aumentada.

Figura 8. Augmented Reality Logo.

En la actualidad la realidad aumentada se utiliza en muchos sectores, entre los cuales se encuentran el entretenimiento, dispositivos de navegación, proyectos educativos, etc. Con la llegada masiva de los nuevos dispositivos móviles la realidad aumentada ha encontrado un mercado en el que extenderse y llegar a más gente.

Sin embargo actualmente esta tecnología tiene sus problemas. Tiene un coste computacional elevado haciendo que la experiencia en algunos dispositivos no sea fluida. Debido a esto la realidad aumentada es muy dependiente del futuro de los algoritmos de reconocimiento de formas y de la potencia de procesamiento de las nuevas tecnologías hardware.

El número de empresas que ofrecen soluciones de realidad aumentada es pequeño pero está en crecimiento. Para la realización del proyecto se han probado varias soluciones que ofrecían cierto grado de integración con Unity, la mayoría eran gratuitas y únicamente compatibles con Android. Finalmente se decidió usar Vuforia [9].

Este plugin está orientado a dispositivos móviles y soporta desarrollo nativo en Android e iOS. Además ofrece unas librerías para integrar ambos sistemas en Unity, haciendo que el mismo código en C# sirva para las 2 plataformas.

2.5 Tipo de juego para el dispositivo móvil

El juego diseñado pertenece al género de cartas coleccionables y rol fantasía por turnos. La idea de inicio fue diseñar el modo multijugador en el que dos jugadores se enfrentan entre sí. Un juego normal transcurre de la siguiente manera.

Los dos jugadores eligen un mazo para jugar. Este mazo consta de un número de unidades fijo (en la última versión del juego 3) y en futuras actualizaciones un grupo de hechizos. Cada jugador puede invocar una unidad en su turno, pudiendo usar una habilidad de esa unidad por turno. Las habilidades provocan diferentes efectos en el estado del juego, infligir daño, sanar a otra unidad, incapacitar durante un número de turnos, etc.

El terreno de juego se va llenando progresivamente de unidades hasta alcanzar el límite. Mientras tanto las unidades se atacan entre sí, usando habilidades físicas o mágicas que hacen que la pantalla se llene de efectos visuales muy vistosos para los jugadores.

El juego termina cuando un jugador ha matado al número de total de unidades del oponente. Este método de juego hace que las partidas sean cortas y que la curva de decisiones que los jugadores pueden tomar se haga mayor conforme avanza la partida pero se estabilice en un momento dado, dando así situaciones en las que los combos entre unidades hacen que se decida la partida.

En futuras actualizaciones se puede implementar el modo offline en el que el jugador podrá enfrentarse a una inteligencia artificial en un modo de juego de aventura típico, además de añadir un tutorial básico. En este modo se podrán ganar cartas nuevas que únicamente se pueden conseguir realizando la aventura. La forma de jugar de este modo será similar al modo multijugador, con la diferencia de que el jugador debe de colocar los marcadores de la IA para poder interactuar con ellos.

Con este diseño de las reglas y la posibilidad de hacer ampliaciones sobre él mediante cartas o aventuras offline nuevas hacen el modelo de negocio sea viable. En los últimos años gracias a la expansión de los dispositivos móviles se han puesto de moda los juegos casuales cuyas partidas suelen durar poco tiempo y los cuales se pueden jugar desde cualquier parte.

3 Análisis de requisitos

Para llevar a cabo el análisis de requisitos se definirán una serie de conceptos claves mediante un **glosario** que ayudarán a la lectura de futuros apartados de análisis, diseño e implementación. Después se describirán los **objetivos** que harán que el proyecto tenga viabilidad y le den valor frente a otros títulos del mercado. Para finalizar se presentarán un conjunto de **casos de uso** que debe de cumplir el proyecto para crear una buena interacción usuario-sistema.

3.1 Glosario

Concepto	Definición
Cliente	Persona que no ha iniciado sesión en el sistema.
Usuario	Persona que ha iniciado sesión en el sistema con éxito y está navegando por los menús o interactuando con el sistema sin jugar.
Jugador	Persona que está jugando contra otro jugador.
Turno	Periodo de tiempo en el que un jugador realiza acciones que repercuten en el estado de la partida.
Unidad	Agente básico con el que interactúa el jugador para realizar acciones.
Acción de juego (Game Action)	Acción básica de juego.
Habilidad	Acción concreta que pueden realizar las unidades.
Habilidad pasiva (Passive)	Tipo de habilidad que se activa cuando ocurre un evento.
Habilidad alternada (Toggle)	Tipo de habilidad pasiva que el jugador decide cuando activar o desactivar.
Habilidad activa (Active)	Tipo de habilidad que el jugador decide cuando usar.
Habilidad objetivo (Target)	Tipo de habilidad activa que requiere seleccionar un objetivo.
Animación	Proceso y periodo de tiempo desde que el jugador realiza una acción hasta que esta se completa.

3 2 Objetivos

Los objetivos que debe de cumplir el proyecto son **multiplataforma**, **multijugador** y **realidad aumentada**.

Multiplataforma.

La naturaleza del videojuego hace que sea un requisito fundamental que la pantalla sea grande. Este requisito descarta de primeras cualquier smartphone o dispositivo de tamaño similar (como consolas portátiles) ya que los jugadores deben de poder ver el tablero de juego completo con sus dispositivos.

Por tanto, la plataforma más adecuada son las tabletas. Un factor importante a la hora de elegir el tipo de tableta es la calidad de las cámaras de video. En el proyecto se hará uso de la cámara trasera, con la ventaja de que tienen mucha más calidad que las cámaras frontales. El problema surge en las tabletas de gama media, sus cámaras tienen una calidad un poco baja para el uso interactivo de la realidad aumentada.

A pesar de este inconveniente, si analizamos el mercado podemos ver como las tabletas de gama alta tienen un calidad más que suficiente para el videojuego. Modelos como el Google Nexus 9, la Samsung Galaxy Tab 4 o el iPad Air 2 nos indican la tendencia que el mercado está tomando, tabletas con calidades de grabación de video HD de 1080p o 720p.

	Cámara Trasera	Resolución	Tamaño	Ancho x Alto x Grosor
Google Nexus 9	Fotos 8 Mpx Video 1080p	2048x1536 281ppp	8,9''	22,8x 15,4x 0,8 cm
Samsung Galaxy Tab 4	Fotos 3 Mpx Video 720p HD	1280x800 149ppp	10,1''	24,3x 17,6x 0,8 cm
iPad Air 2	Fotos 8 Mpx Video 1080p HD	2048x1536 264ppp	9,7''	24x 17x 0,9 cm

Figura 9. Comparativa tabletas gama alta.

Además de los diferentes sistemas operativos que tengan las tabletas, también hay una gran variedad de modelos con diferentes tamaños y resoluciones de pantalla. Para crear interfaces de usuario que se adapten a los diferentes tipos de pantallas usaré el API de diseño de interfaces de Unity que resuelve muchos de los problemas de escalado y posicionamiento en múltiples resoluciones.

Otro sistema importante dentro del videojuego es el sistema de toques. Cada sistema operativo implementa su propia forma de interactuar con la pantalla táctil. Para implementar dentro del videojuego este subsistema usaré el trazador de rayos de Unity que me permite trazar rayos desde la pantalla a un punto específico de la escena.

Multijugador.

Otro objetivo muy importante es que el videojuego sea multijugador. La interacción entre jugadores es un factor muy importante que añade multitud de experiencias al juego. Además la naturaleza de los juegos de cartas hace que sean más divertidos cuando se juegan contra otros jugadores.

Para desarrollar este sistema se han barajado varias alternativas como el clásico modelo cliente-servidor o el P2P. Los pros y contras de estas alternativas se analizarán con detalle en el apartado de diseño e implementación.

Realidad aumentada.

El requisito y objetivo que más valor le va a dar al videojuego frente a otros títulos del mercado es el uso de la realidad aumentada. Para su integración en el proyecto se han buscado multitud de plugins. La inmensa mayoría de ellos solo soportaban una plataforma, lo que hacía más costoso el desarrollo del proyecto.

El plugin elegido es Vuforia, está desarrollado por Qualcomm y permite con prácticamente la misma API desarrollar para Android e iOs. Además tiene una versión integrada en Unity.

Por contra, uno de los límites que tiene Vuforia es que no puede hacer un reconocimiento simultáneo de dos o más marcadores idénticos. Por tanto en una misma partida no pueden aparecer dos cartas iguales al mismo tiempo. Este inconveniente fue decisivo en el diseño del videojuego ya que no se puede diseñar un juego de cartas coleccionables clásico.

Modelo de negocio.

Ahora que el mercado se ha globalizado y han aparecido varios modelos de negocio que no son el clásico de compra y juega es imprescindible plantear el modelo de negocio que seguirá el videojuego desde los inicios del desarrollo. Esta decisión repercutirá sobre futuras decisiones de diseño.

Analizando el mercado se puede ver como muchos juegos de Android e iOS han tenido éxito gracias a la accesibilidad de ellos a través de sus precios. Por ello el modelo de negocio elegido es F2P con microcompras.

Esto hace que el juego limitado (algo más grande que una simple demo) esté accesible para todo el mundo y que el videojuego se sostenga gracias a un pequeño grupo de usuarios que compren cartas o extensiones.

Otro punto a tener en cuenta en el modelo de negocio de este videojuego en concreto es cómo el jugador puede obtener las cartas físicas (cartas con marcadores que se reconocerán en la aplicación de realidad aumentada). La idea principal es permitir su descarga gratuita a través de la página web oficial del producto, dando consejos de impresión y mostrando la galería total de cartas. Sin embargo dentro del juego solo podrán ser usadas las cartas que tengas en posesión virtual en la cuenta.

3 Casos de uso

Los casos de uso que se presentarán a continuación son una representación a alto nivel de cómo interactúan los usuarios con el sistema y del flujo de información entre ellos ya que una de las características principales del videojuego es su modo multijugador.

Los casos de uso que se analizan son por ejemplo inicio de sesión en el sistema, la búsqueda de usuarios o una acción básica de juego entre ellos. Estos casos muestran de forma general cómo debe de interactuar del usuario con el sistema para que la experiencia interactiva sea la correcta y se logren características importantes como la seguridad o una interactuación fluida con los diferentes menús.

Para su análisis, por cada uno se mostrará una tabla con una descripción general del caso de uso en cuestión, los actores que participan en él, las condiciones que se deben de cumplir para que pueda llevarse a cabo y si es necesario unos comentarios adicionales. Después de mostrar el diagrama del caso de uso se pasará a hacer una breve explicación del mismo.

El caso de uso de inicio de sesión representa la forma en la que interactúa un cliente normal con el sistema por primera vez. En él se realizan algunas pruebas de seguridad y el cliente pasa a ser un usuario.

El caso de uso de búsqueda de usuarios aborda el clásico problema de encontrar a 2 usuarios entre sí intercambiando sus datos públicos.

En los últimos 2 casos de uso, desafiar usuario y realizar acción, interactúan los 2 usuarios entre sí y con el sistema. Este intercambio de información pasando por el servidor es necesario ya que no suele ser buena idea implementar toda la seguridad y la lógica de juego únicamente en el cliente ya que sería más fácil hacer trampas y hackear el sistema. A continuación se analizarán todos estos casos con más detalle.

Nombre	Inicio de Sesión.
Descripción	Un cliente sin identificar intenta iniciar sesión en el sistema, si tiene éxito se le promociona a usuario.
Actor	Cliente.
Precondición	Para poder iniciar sesión debe de haberse registrado correctamente en el sistema mediante un subsistema de registro.
Comentarios	

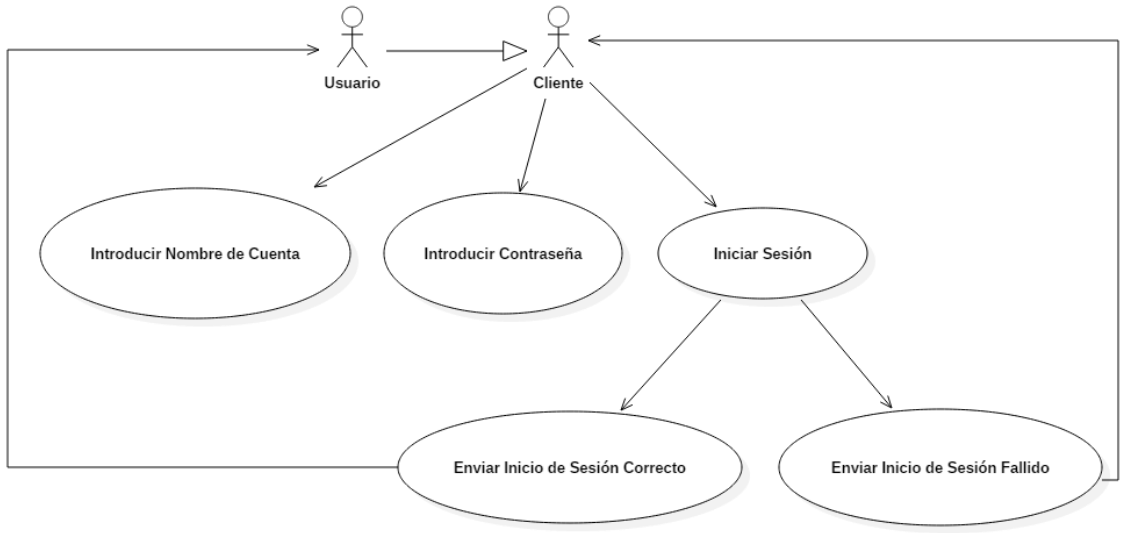

Figura 10. Caso de uso inicio de sesión.

Para que un cliente pueda usar los servicios del sistema o jugar debe de iniciar sesión en el sistema previamente (la parte de registro no está implementada todavía). Si el inicio de sesión se realiza con éxito el cliente promociona a usuario obteniendo un subconjunto de los servicios del sistema.

Nombre	Búsqueda de Usuarios.
Descripción	Un usuario le pide al sistema una lista de usuarios que cumplan una determinada condición.
Actor	Usuario.
Precondición	Caso de Uso Inicio de Sesión.
Comentarios	La condición de búsqueda viene dada por una futura decisión de diseño, recuperar usuarios que estén en la misma subred, usuarios que estén buscando otros usuarios, etc.

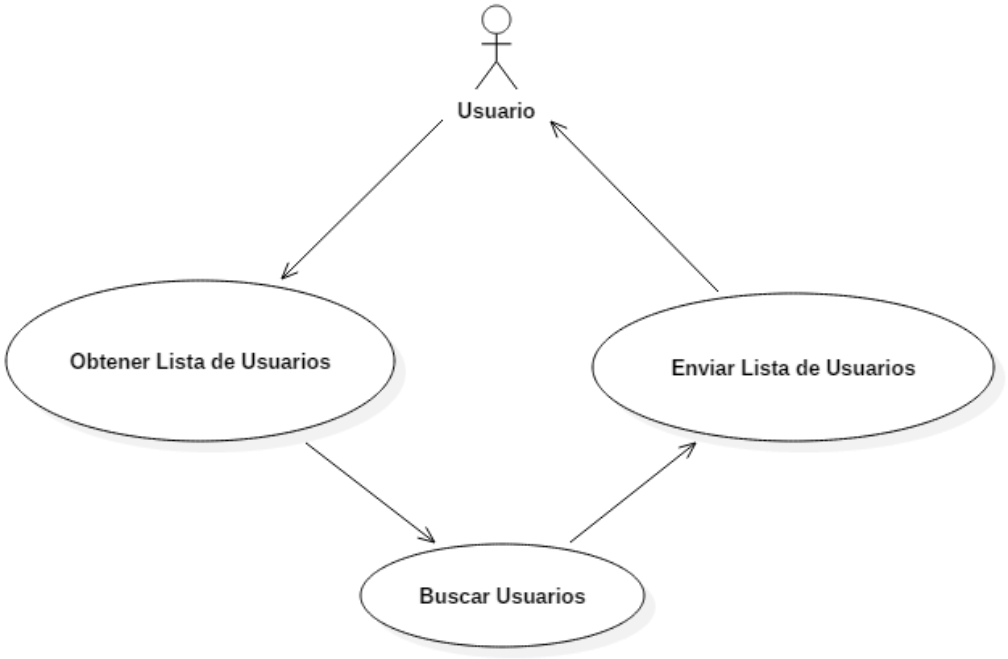

Figura 11. Caso de uso búsqueda de usuarios

Para que un usuario pueda realizar una búsqueda de usuarios debe de introducir los parámetros de búsqueda, algunos ejemplos son usuarios con un nivel parecido al suyo, usuarios que se encuentren en la misma red, etc.

Nombre	Desafiar Usuario.
Descripción	Un usuario desafía a otro a un duelo. Si se acepta el desafío los 2 promocionan a jugadores.
Actor	Usuario.
Precondición	Caso de Uso Búsqueda de Usuarios.
Comentarios	

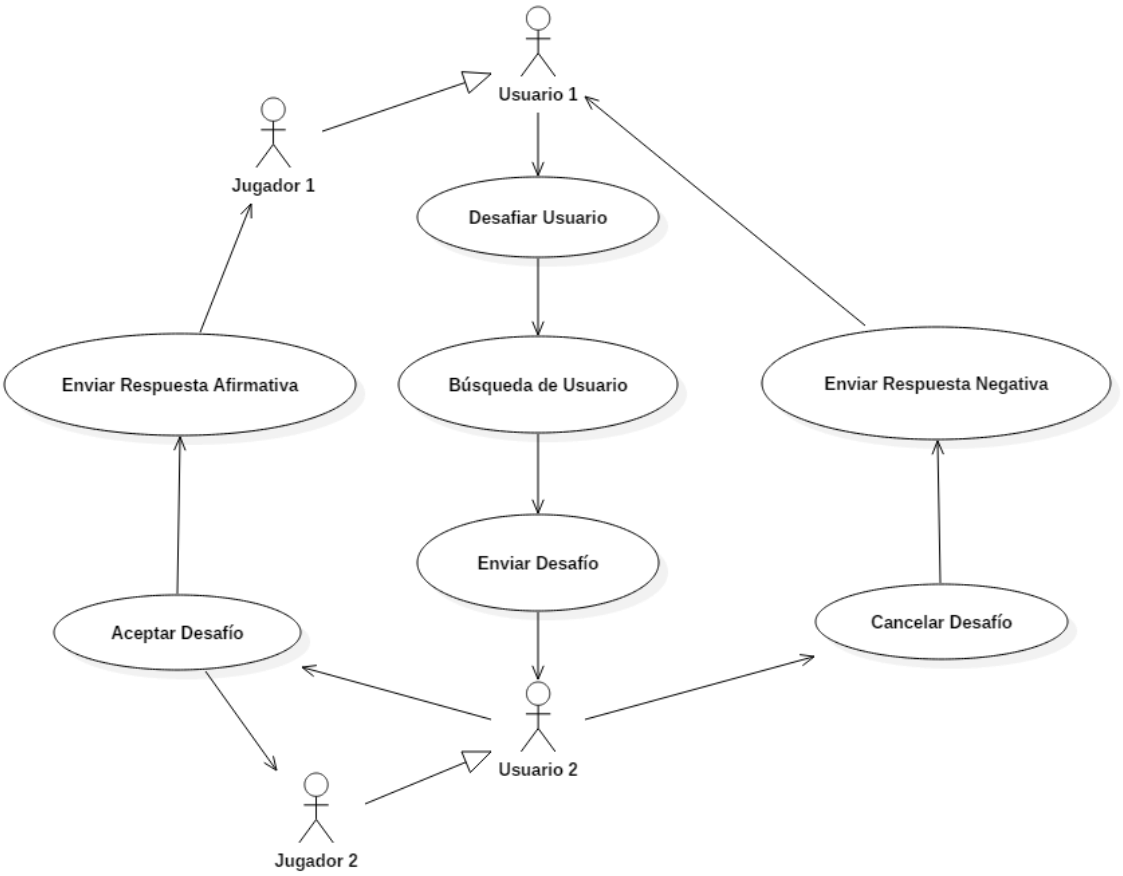

Figura 12. Caso de uso desafiar usuario.

Una vez el usuario ha buscado adversarios puede desafiar a uno de ellos. Para desafiarlo le envía un mensaje y espera su respuesta, si ha pasado un cierto periodo de tiempo sin respuesta, esta se resuelve de forma negativa para ambos. Si el adversario contesta positivamente ambos usuarios promocionan a jugadores.

Nombre	Realizar Acción
Descripción	Realiza una acción que alterará el estado de una partida.
Actor	Jugador.
Precondición	Caso de Uso Desafiar Usuario.
Comentarios	

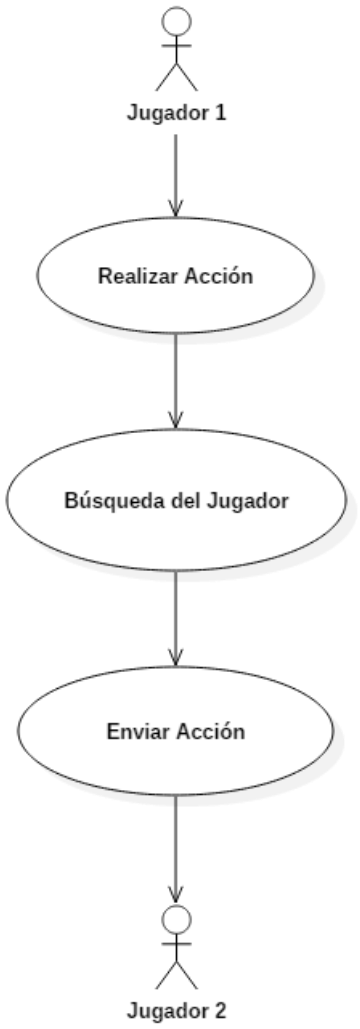

Figura 13. Caso de uso realizar acción.

El envío de una acción de juego es sencillo ya que los 2 jugadores están registrados en el sistema como jugando un versus y gestionando sus conexiones. Si una acción es incorrecta el cliente de juego se encarga de avisar al jugador de que no puede realizar esa acción y no la procesa. Si es correcta se manda a través del servidor al adversario.

4 Diseño e implementación

En este apartado se describirá el proceso de diseño e implementación dividiéndolo en los siguientes apartados, **desarrollo con Unity**, **mecánicas de juego**, **interfaz gráfica, realidad aumentada** y **comunicación**.

4.1 Desarrollo con Unity

Para desarrollar con Unity hace falta conocer cómo funcionan los motores de videojuegos modernos en general.

Todo el sistema gira en torno al concepto de objeto de juego, en Unity la clase *GameObject*. Los gameobject tienen una lista de componentes que los definen, algunos ejemplos son *Transform* (posición, rotación y escala del objeto), *Collider* (componente que se encarga de detectar y gestionar las colisiones entre objetos), *RigidBody* (permite al objeto interactuar con el motor físico), etc.

El motor funciona iterando sobre un **Game Loop** [3] que itera en cada frame para realizar los cálculos y el renderizado. Este patrón de diseño es el más usado en este tipo de sistemas cuya prioridad es ejecutar el programa en tiempo real sin perder rendimiento.

Otro patrón de mucha importancia en el motor es **Update Method** [2,3]. Consiste en que el bucle principal ejecuta el método seleccionado en cada iteración para actualizar el estado del objeto. En Unity existen 3 métodos que se actualizan periódicamente (aunque es posible implementar los tuyos propios con otros periodos), el método Update (se actualiza en cada frame), LateUpdate (al acabar cada frame) y FixedUpdate (se ejecuta a la frecuencia del motor físico).

Para el uso de estos métodos es necesario crear una clase que derive de la superclase MonoBehaviour. Esta clase proporciona por herencia estos métodos además de otros, algunos ejemplos son el método Awake (se ejecuta cuando el objeto es creado), el método Start (se ejecuta la primera vez que se activa el objeto) o algunos eventos del sistema como *on destroy* o los encargados de gestionar las colisiones o interactuar con el motor físico.

Para repartir la carga del sistema y mejorar el rendimiento del videojuego, Unity da la posibilidad de gestionar la periodicidad con la que una función se ejecuta a través de las corutinas. Su ejecución puede hacerse cada cierto tiempo, una vez por frame o al final del frame, tal como lo hacen los métodos de update.

El bucle de juego y los estados por los que pasa el sistema en cada frame pueden verse en la siguiente máquina de estados que representa el ciclo de vida de la clase MonoBehaviour.

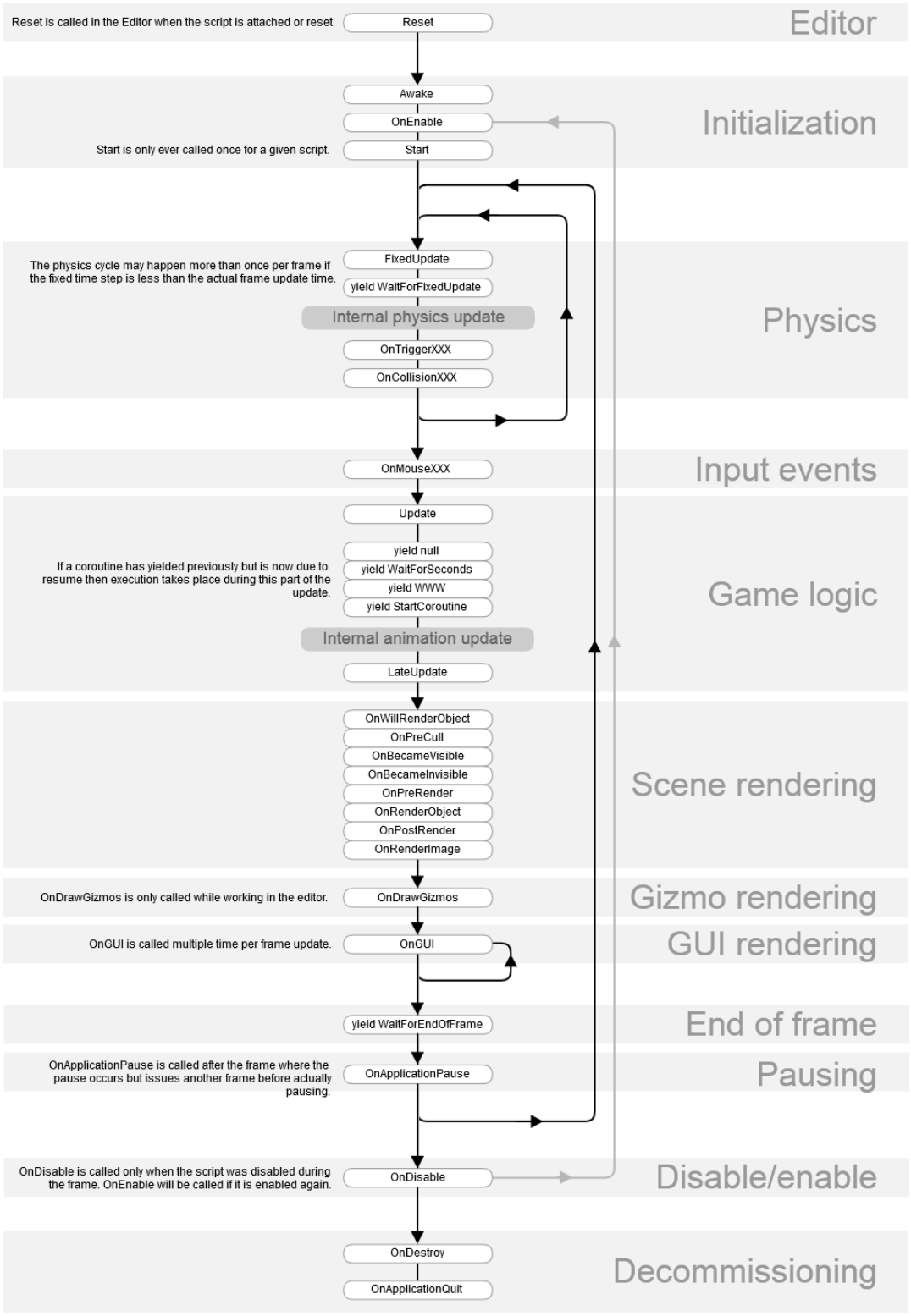

Figura 14. Máquina de estados de MonoBehaviour.

4 2 Mecánicas de juego

Para implementar las mecánicas de juego hace falta definir el sistema de clases que tiene el videojuego. El sistema gira en torno a la clase *Card* que representa la funcionalidad de la carta física, de esta clase deriva la clase *Unit* (agente básico de juego) y está abierta a la implementación de otros tipos de cartas como objetos o hechizos.

La relación entre *Unit* y *UnitController* no es más que la separación que hay entre el estado estático (inalterable en la partida) y el estado dinámico que controla la unidad a lo largo de la partida.

Además las unidades tienen de una a cuatro habilidades, representadas en la clase *Ability*. Esta clase es tan solo un contenedor de datos y deriva del tipo *ScriptableObject* que es una de las formas que tiene Unity de hacer que un objeto sea persistente.

La funcionalidad de las habilidades está implementada en la clase que representa el concepto de acción, del cual hablaré a continuación.

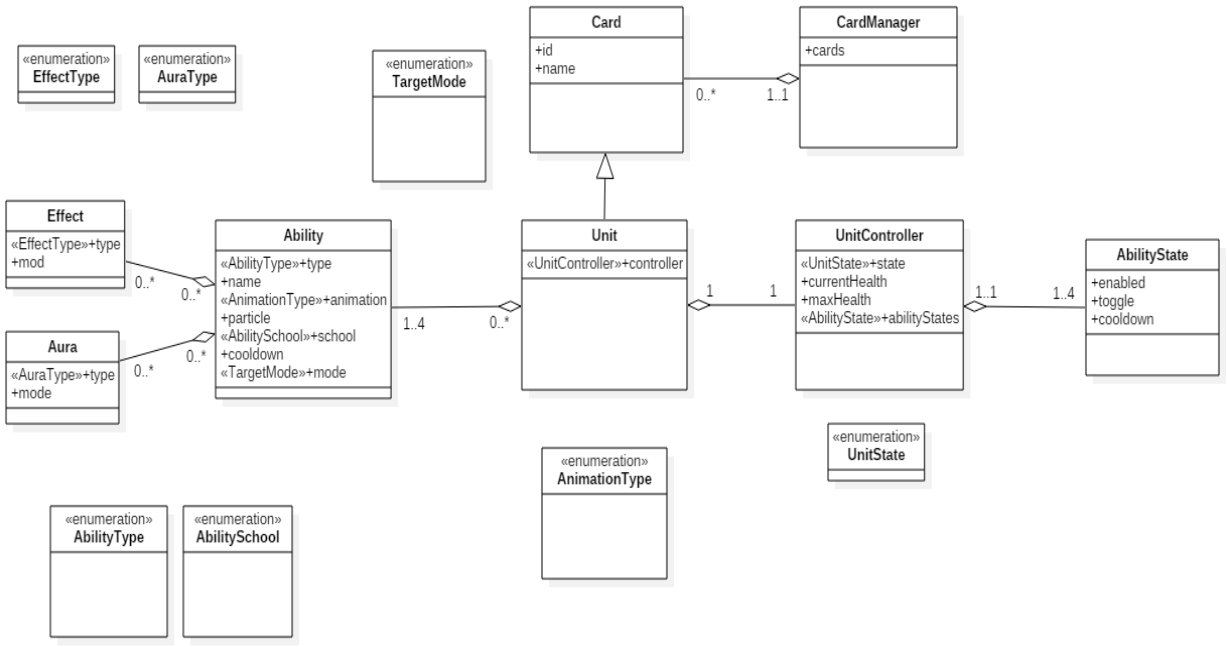

Figura 15. Diagrama de clases del sistema.

Toda la implementación de las mecánicas gira en torno al concepto de acción. El jugador puede realizar una serie de acciones que tienen unos parámetros comunes y un conjunto de efectos muy variados. Este concepto está implementado en código como la clase abstracta *GameAction*.

Esta forma de encapsular una acción o una llamada a un método es la definición del patrón de diseño **Command** [2,3].

El diagrama de clases del núcleo de las mecánicas de juego es el siguiente.

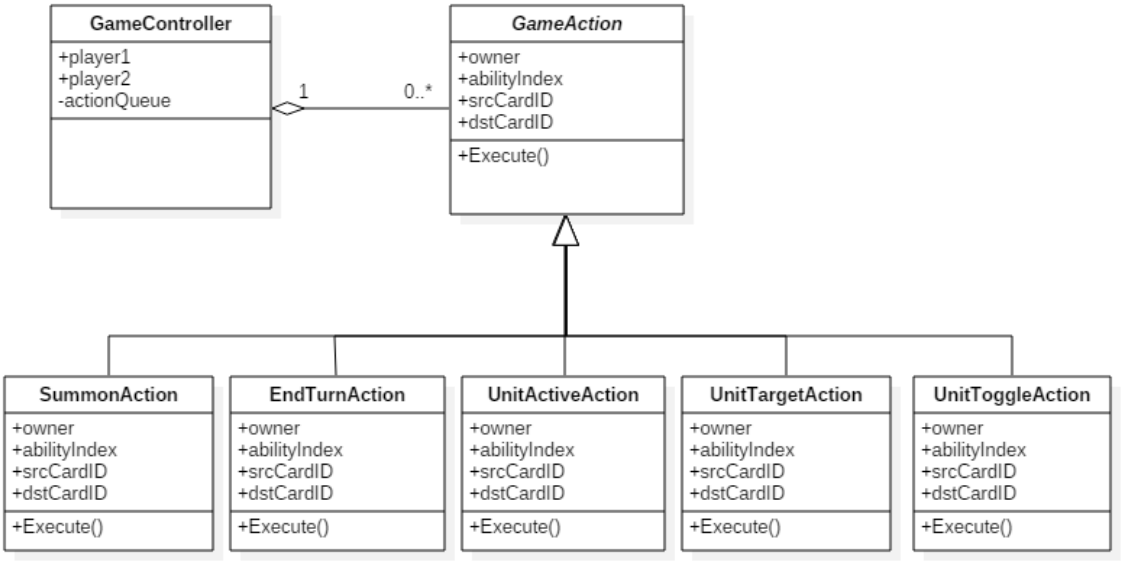

Figura 16. Diagrama de clases mecánicas de juego.

En el diagrama se puede observar como de la clase *GameAction* derivan todas las acciones que un jugador puede realizar. Todas las acciones tienen 4 parámetros en común, el propietario de la acción, el índice o identificador de la habilidad que se realiza y los identificadores de las cartas fuente y destino (en caso de que hubiera destino).

En lo único que se diferencian las acciones es en la implementación del método *Execute*. Este método ejecuta la acción buscando los jugadores y unidades implicadas, actualizando el estado del juego y lanzando la animación correspondiente.

En la ejecución de una acción la unidad fuente realiza una animación e instancia un sistema de partículas con un tiempo de vida corto. Este proceso ocupa un intervalo de tiempo mucho mayor que el tiempo que se necesita para aplicar los efectos de la acción sobre el estado de juego.

Para evitar que dos animaciones o más se solapen he implementado una cola de acciones. Esta cola se encuentra en la clase *GameController*, la cual se encarga de que una acción se ejecute completamente antes de que otra pueda lanzarse.

El conjunto de animaciones que una unidad puede realizar está definido por una máquina de estados implementada sobre el sistema de animación de Unity llamado Mecanim.

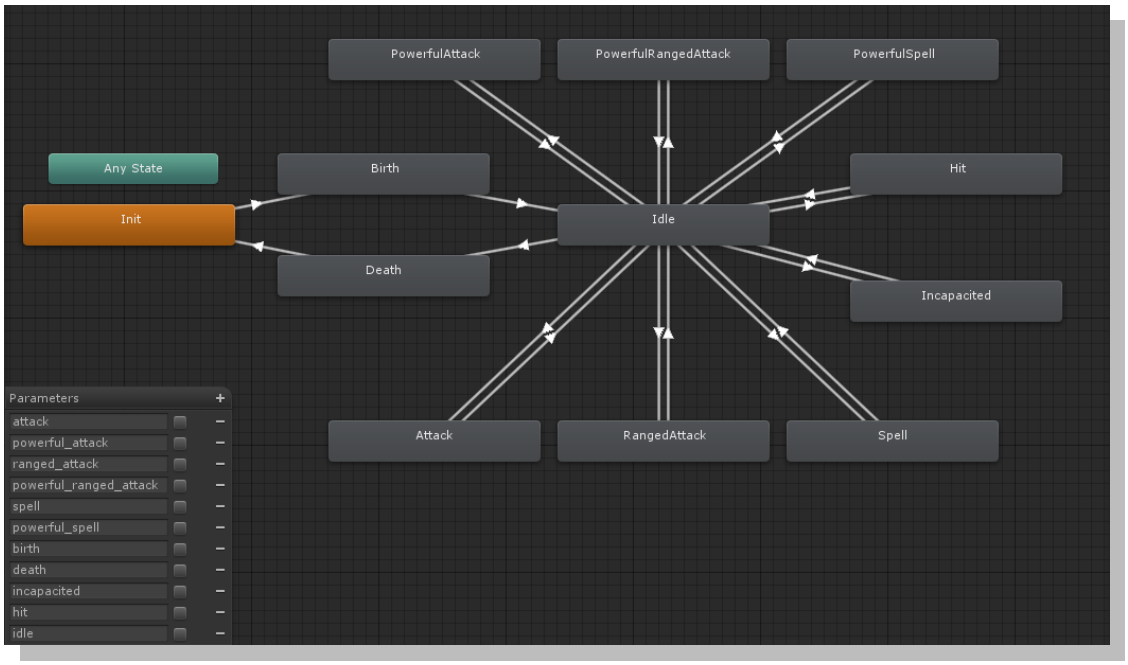

Figura 17. Máquina de estados de animaciones.

Cada vez que una animación es lanzada se activa un disparador, si el estado actual tiene una transición con ese disparador se transita al estado destino, reproduciendo la animación.

Sobre este sistema se encuentra la clase *AnimationController* que se encarga de abstraer el funcionamiento de Mecanim y ofrecer su interfaz a la clase *UnitController*. Esta arquitectura permite controlar a la clase *UnitController* mediante una clase que gestione las entradas externas al sistema hechas por el jugador (*InputController*) o mediante un sistema de IA que se encargue de proporcionar las entradas al controlador de unidad.

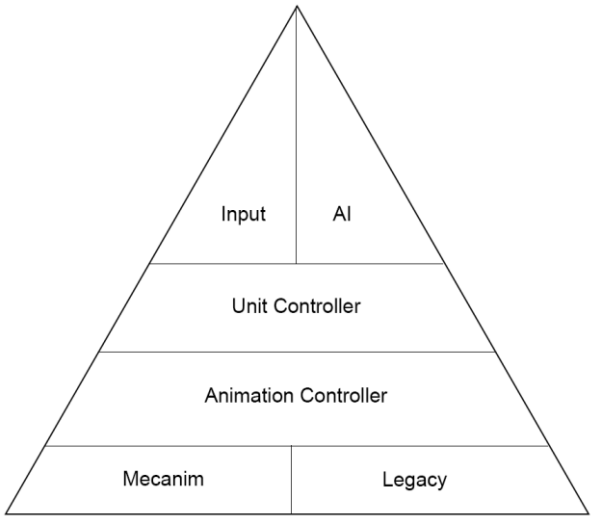

Figura 18. Jerarquía de controladores.

4 3 Interfaz gráfica

Al tratarse de un juego por turnos el diseño de la interfaz gráfica toma un papel más relevante que en otros géneros. Por esto decisiones como establecer el número de unidades simultáneas en el juego o las habilidades que tiene cada unidad es muy importante.

Las clases *TouchController* y *UIController* conforman el sistema de entrada e interacción del usuario con el sistema. La primera se encarga de ofrecer una interfaz de toques al resto de clases y la segunda transforma estos toques en acciones de interfaz si el toque se ha hecho sobre un elemento gráfico o widget.

El diseño inicial de la interfaz gráfica de juego es el siguiente.

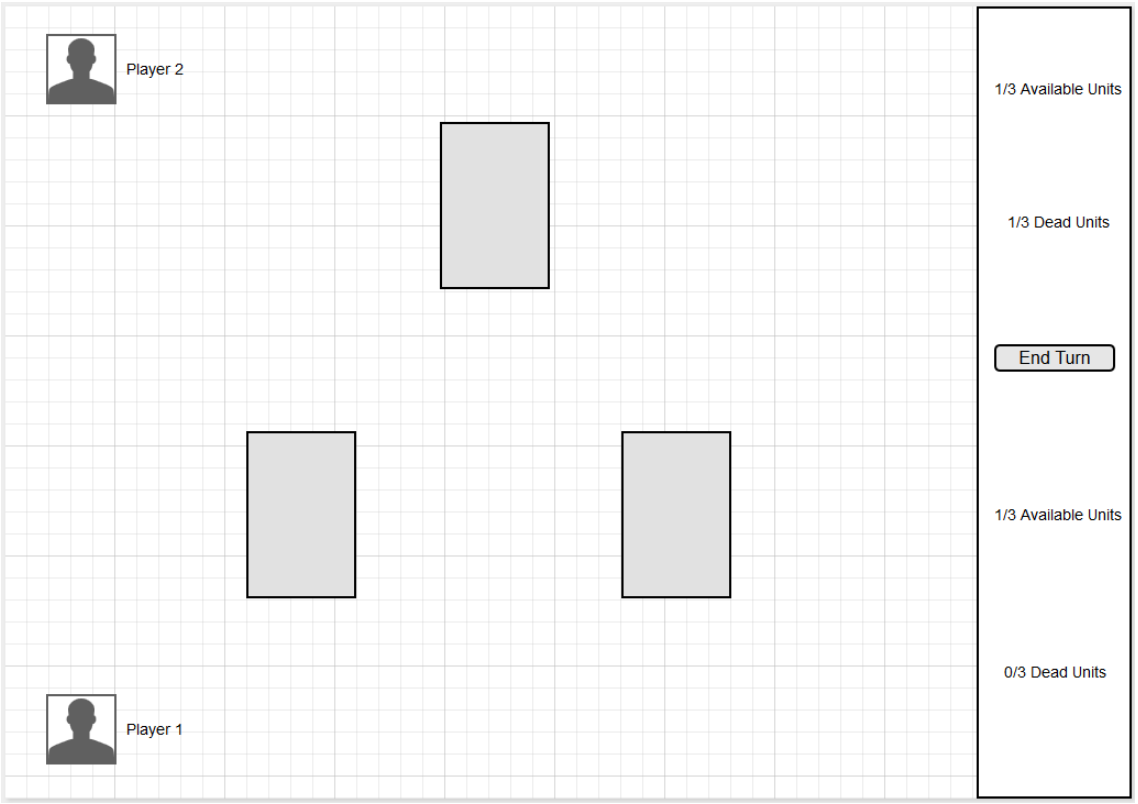

Figura 19. Interfaz gráfica de juego.

Las cuadrículas representan el video capturado por la cámara de video y los rectángulos grises las cartas físicas donde se proyectaran los modelos 3D mediante la realidad aumentada.

Con este diseño podemos ver que la mayor parte de la interfaz está destinada a la captura de video y realidad aumentada para centrar el foco de la partida en el mundo real y 3D.

El jugador dueño del dispositivo tiene su tablero y estado de la partida en la parte inferior mientras que el oponente lo tiene en la parte superior, sus avatares y nombres se pueden ver en estas zonas. El panel lateral nos muestra la información importante de la partida así como el botón de acabar el turno. Toda esta interfaz gráfica es un panel proyectado ortográficamente sobre el video capturado por la cámara.

Además de este panel, hay otros paneles pequeños que permiten al jugador interactuar con las unidades del tablero de juego. El panel de habilidad aparece cuando un jugador selecciona una unidad, muestra las habilidades de la unidad seleccionada así como su estado y los puntos de vida actuales. Este panel es básico para la interacción del jugador en la partida. La acción de invocar unidades o de marcar como objetivo de una habilidad se hace con paneles similares a este. Estos paneles se crean en la escena y se posicionan en el centro de la unidad seleccionada proyectándose ortográficamente sobre la cámara.

Para la implementación de estos paneles ha sido muy importante el uso de la abstracción que hace Unity de pantallas táctiles. La interfaz que ofrece es la clase *Touch*, la cual tiene la lista de dedos que están tocando la pantalla en sus determinadas posiciones, además identifica cada dedo con una id para poder llevar un control sobre deslizamientos y otros efectos.

Una vez la posición del dedo sobre la pantalla es conocida, el siguiente paso lógico es trazar un rayo desde la pantalla a la escena tridimensional. Este rayo puede colisionar tanto sobre objetos 3D como 2D. Un punto fuerte de este sistema es que el coste computacional es despreciable ya que solo se traza el rayo cuando el usuario toca la pantalla. Una vez se ha detectado la colisión en un objeto determinado, se realizan las operaciones y cómputos necesarios para el tratamiento del evento.

```
28    private void LateUpdate()
29    {
30        if( !EventSystem.current.IsPointerOverGameObject() )
31        {
32            #if UNITY_EDITOR || UNITY_STANDALONE
33            if( Input.GetMouseButtonUp(0) )
34            {
35                ray = arCamera.ScreenPointToRay(Input.mousePosition);
36                CastRay();
37            }
38            #endif
39
40            #if UNITY_ANDROID || UNITY_IPHONE
41            if( Input.touchCount > 0 )
42            {
43                touch = Input.GetTouch(0);
44                if( touch.phase != TouchPhase.Ended && touch.phase != TouchPhase.Canceled )
45                {
46                    ray = arCamera.ScreenPointToRay(touch.position);
47                    CastRay();
48                }
49            }
50            #endif
51        }
52    }
```

Figura 20. Núcleo del sistema de Raycast.

Este es el núcleo de la clase que controla los toques. Dependiendo de la plataforma destino el rayo se tomará desde un periférico como un ratón o desde la pantalla táctil. Con estas abstracciones he conseguido que todo el sistema use la misma interfaz, creando una capa independiente de la plataforma en la que se ejecuta el videojuego.

Otra parte importante de la interfaz son los menús por los que atraviesa el cliente hasta que se convierte en jugador. Esta navegación debe de ser sencilla y un objetivo clave es que el usuario sepa cómo moverse por los menús de una forma intuitiva.

La navegación entre menús está dividida en tres paneles principales.

El primer panel es el menú de inicio de sesión. Se trata de un menú típico en el que el cliente puede cambiar las opciones del juego, salir de la aplicación o iniciar sesión recibiendo información a través de ventanas de diálogo emergentes.

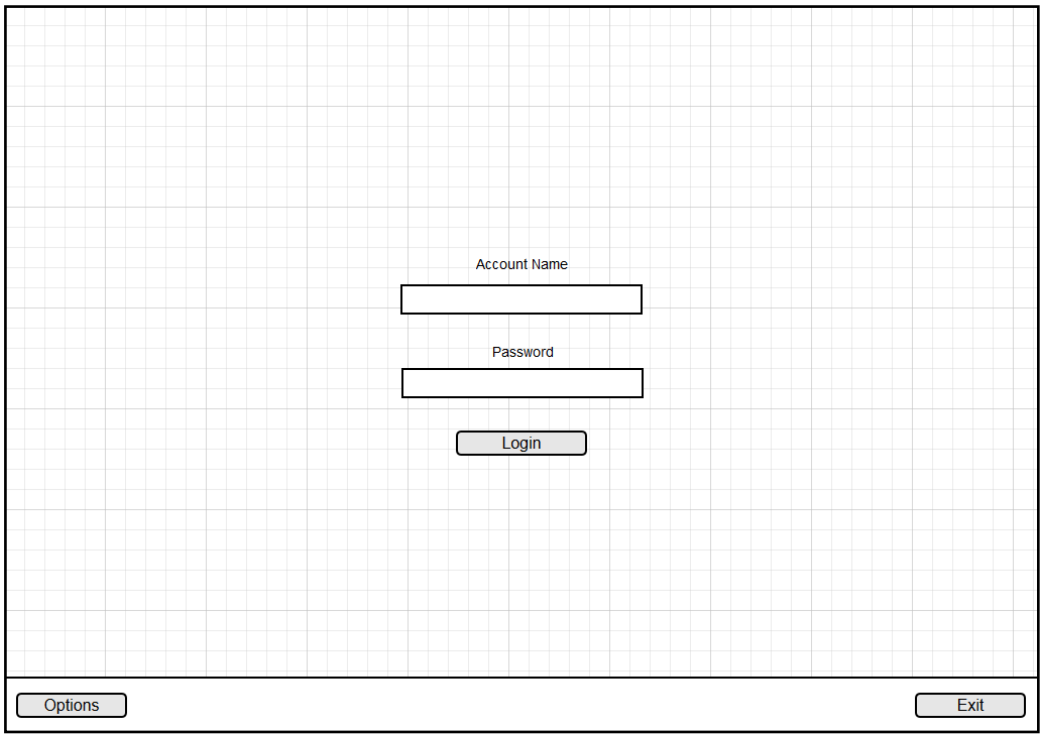

Figura 21. Menú de inicio de sesión.

El siguiente panel es el menú principal en el que el usuario puede elegir el modo de juego, aventura o duelo, o ir a la tienda a comprar expansiones o cartas. El panel de la tienda no está implementado todavía pero su diseño será parecido al de cualquier tienda virtual, el usuario podrá acceder a la compra de cartas individuales, packs de rebajas o expansiones completas. Además el menú tiene las opciones de navegación previas de los otros paneles.

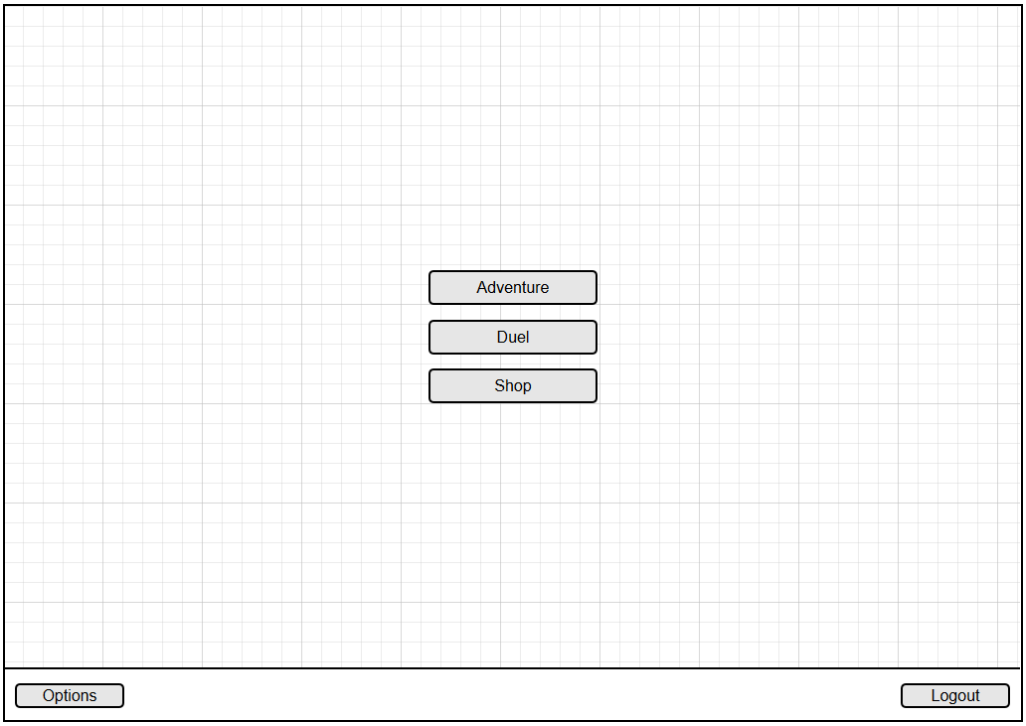

Figura 22. Menú principal.

El último panel es el menú de búsqueda donde el usuario busca oponentes a los que enfrentarse en duelo. Con el botón de refresco se actualiza la lista de candidatos y en el borde inferior se encuentra el mismo panel de navegación que tienen el resto de menús.

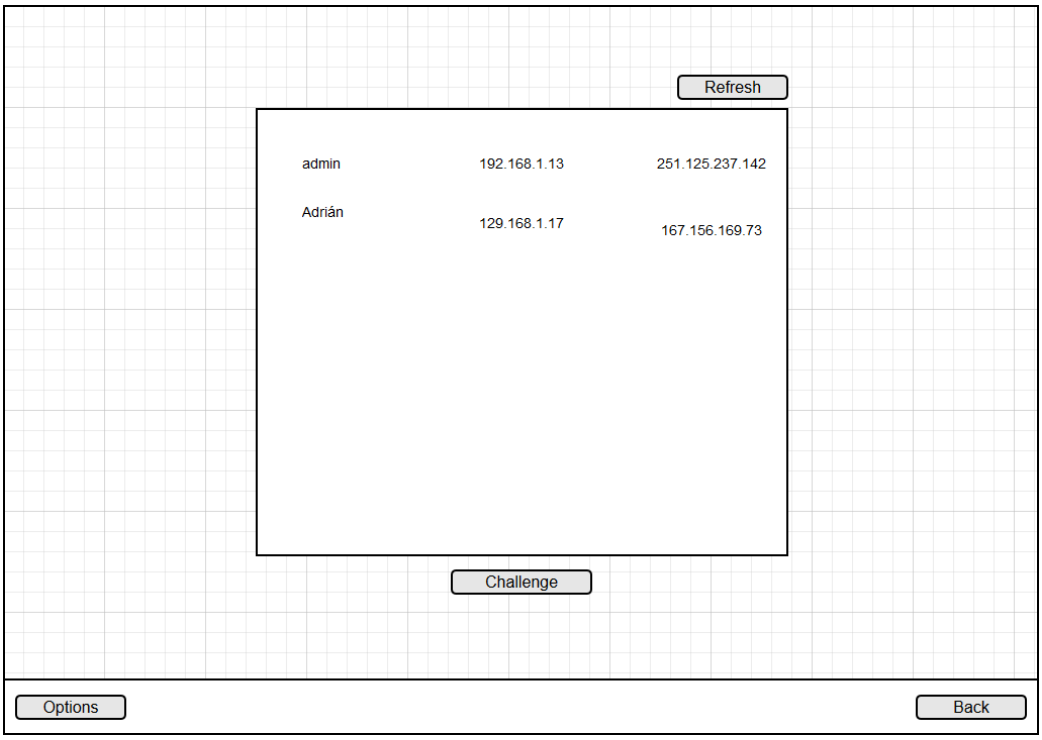

Figura 23. Menú de búsqueda.

Para implementar la interfaz gráfica he usado el patrón de diseño **State** [2,3] cuya definición es *"permite a un objeto alterar su comportamiento cuando su estado interno cambia"*.

Este patrón es perfecto para diseñar interfaces y su funcionamiento está basado en las máquinas de estado finitas (FSM, *finite state machines*) [18].

El diagrama de clases de la interfaz gráfica es el siguiente (la aridad de las clases de este diagrama es de 1 a 1).

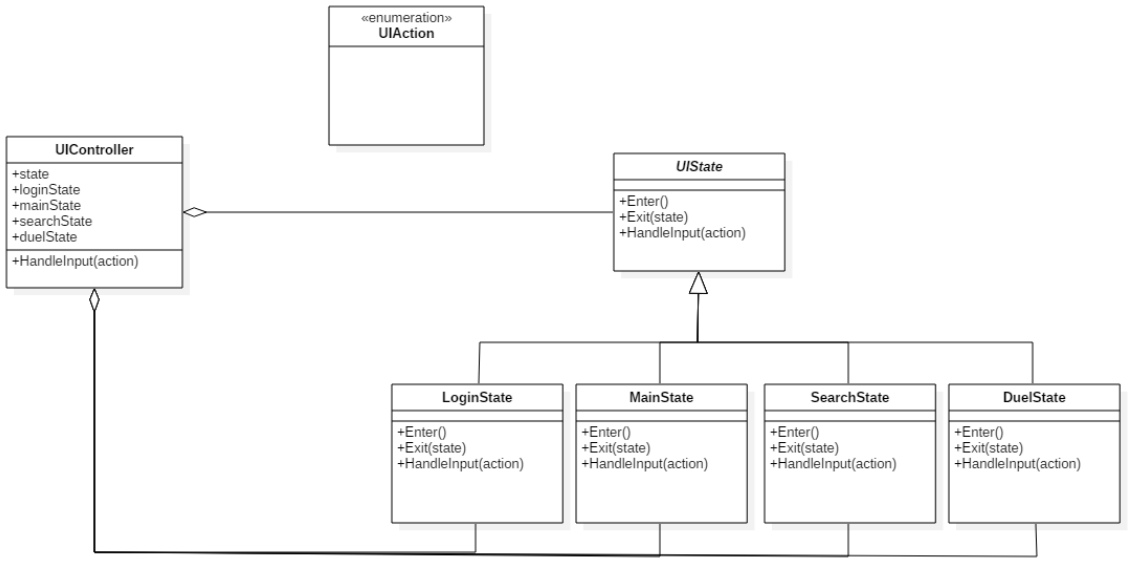

Figura 24. Diagrama de clases interfaz gráfica

La clase *UIController* tiene una variable del tipo abstracto *UIState* y una variable por cada clase que deriva de esta. El método *HandleInput* delega en el tipo de clase que contenga la variable *state*. Todas estas clases implementan los métodos abstractos de la clase de la que derivan.

El método *Enter* permite inicializar el estado al que se ha transitado y realizar una animación (en el caso concreto del videojuego se realiza una animación donde el menú invisible va cambiando su valor de transparencia de 0 a 1 gradualmente).

El método *Exit* realiza el tránsito del estado actual a un estado final. Este método realiza una animación de desaparición del menú actual, y al finalizar esta, una animación de aparición del menú destino.

El método *HandleInput* ejecuta las acciones que el usuario realiza, inhibiendo las acciones que no son válidas en el menú actual. Por ejemplo un usuario no puede cancelar una ventana de diálogo si esta aún no ha aparecido.

4 Realidad aumentada

La implementación de la realidad aumentada se ha realizado con el plugin Vuforia de Qualcomm [9].

Este framework ofrece un nivel de abstracción sobre los sistemas operativos Android e iOS y trabaja como socio de Unity Technologies. La web de Vuforia tiene tutoriales, el foro y una API muy detallada para ayudar a los desarrolladores con sus proyectos.

El sistema a alto nivel de Vuforia se puede ver en la siguiente figura.

Figura 25. Diagrama a alto nivel de Vuforia.

El sistema de captura de patrones y formas usa como elemento básico la clase *Trackable*, de la cual derivan todas las clases que quieran implementar un método de detección. La clase que he usado para la detección de patrones es del tipo *frame marker*. Esta clase ofrece un conjunto de patrones que el sistema tiene registrados y que hace que su detección sea rápida. Una limitación que tiene es que no puede detectar 2 marcadores iguales simultáneamente.

Cada clase del videojuego que tiene la característica de ser reconocida y tiene una forma visible en el mundo de la escena, tiene una instancia del tipo *frame marker* que gestiona mediante eventos como "*se ha detectado un marcador"* o "*se ha perdido un marcador"*.

Para escuchar estos eventos hace falta subscribirse previamente a ellos, por ello hago una subscripción masiva al inicio del videojuego, cuando el jugador está esperando a que se inicie la aplicación. Con esto consigo que los turnos de juego sean más ágiles.

Cuando un evento de reconocimiento es escuchado se activa el objeto reconocido, activando sus scripts de comportamiento y sus componentes importantes como el *collider* (el componente que detecta si hay colisiones con el *gameobject*). Después el control pasa a manos del sistema de mecánicas del juego, el cual decide si el objeto reconocido debe mostrarse, en qué punto de la máquina de estados de las animaciones se encuentra o si debe de mostrar algún sistema de partículas concreto.

Cuando un evento de pérdida de marcador es escuchado se desactivan los componentes que gestionan el estado visual del *gameobject*. De esta manera el objeto sigue despierto a la espera de peticiones aunque el usuario no pueda interactuar con él, las animaciones siguen su ejecución aunque el objeto no esté visible. Esta escucha silenciosa es un método para seguir actualizando el estado del objeto aunque no pueda recibir ninguna entrada del jugador.

Con este sistema se puede realizar una comunicación en tiempo real muy rápida y fluida, aunque como he mencionado en apartados anteriores el factor más importante que influye en la interactividad del sistema es la calidad de la cámara de video del dispositivo.

Por otra parte Vuforia ofrece otros servicios relacionados con la realidad aumentada. Uno de los más famosos es el reconocimiento de formas online en la nube. Este sistema permite gestionar una base de datos de marcadores online. Desde tu aplicación puedes conectarte a ella mediante tu registro en Vuforia y acceder a todos los marcadores subidos a la nube.

Vuforia ofrece este acceso a la gestión de la base de datos de marcadores a través de la web. Allí puedes crear conjuntos de marcadores y enlazarlos con tu aplicación. Este servicio es de pago aunque tiene una versión gratuita con un límite de reconocimientos pequeño.

Uno de los problemas que tiene Vuforia es su modelo de negocio. Hace menos de un año publicar aplicaciones que usaran Vuforia era completamente gratuito con el debido uso legal del framework. Ahora, coincidiendo con el requisito de iOS de publicar también las aplicaciones en 64 bits, el precio por app publicada con Vuforia es de 500$ y el uso de la nube requiere un coste adicional.

Este cambio ha ido justo en sentido contrario al modelo de Unity. Con la versión 5 de Unity todas las características del motor se volvieron completamente gratuitas, con el requisito de que si tu aplicación supera un umbral de beneficios debes comprar la versión profesional. Esta decisión sigue la línea de democratizar el desarrollo de videojuegos haciéndolo más accesible para equipos pequeños, idea que choca con el modelo de negocio de Vuforia.

.

4 5 Comunicación

Con el sistema de juego diseñado el siguiente paso es implementar la forma en que los jugadores se comunican y los mensajes de juego se envían. Al tener encapsuladas las acciones de juego en la clase *GameAction* la comunicación es tan simple como enviar y ejecutar estas acciones de forma ordenada al oponente.

Ante estos requisitos en la capa de transporte se ha elegido como protocolo *TCP*, capa sobre la cual estará implementada la aplicación. El requisito de ejecutar en orden las acciones, ya sean de un jugador o de otro, ya está resuelto gracias a la cola de acciones del *GameController* y a que el protocolo TCP garantiza el envío ordenado de mensajes.

Con el envío de mensajes claro hay que buscar una forma con la que los clientes se encuentren y se puedan comunicar. Para ello se han probado 2 alternativas.

La **primera alternativa** fue realizar toda la transferencia de mensajes a través de un servidor que esté a la escucha de peticiones. Esta alternativa sería crítica si se tratase de un juego en tiempo real donde hay muchas variables que deben estar actualizadas cada segundo (posición y rotación del jugador, de los enemigos, estado del juego, etc.). Sin embargo al tratarse de un juego por turnos donde la media de envío de mensajes por segundo es muy baja esta opción es bastante interesante.

La **segunda alternativa** es establecer el encuentro de 2 clientes mediante un servidor pero realizar el envío de mensajes entre clientes (o más comúnmente llamado P2P, *peer-to-peer*). Esta técnica haría que la validación de mensajes se hiciera exclusivamente en la parte cliente.

Sin embargo esta alternativa tiene un problema bastante importante, los sistemas P2P tienen muchas dificultades para atravesar la traducción de direcciones de red (NAT, network address translation) y en el contexto de los videojuegos dejar la seguridad únicamente en manos del software de los clientes suele hacer que estos sistemas sean fáciles de hackear y hacer trampas.

Para resolver el problema de atravesar los NAT usé una técnica llamada *hole punching* que describe Bryan Ford en uno de sus artículos llamado *"Peer-to-Peer Communication Across Network Address Translators"* [6].

Esta técnica consiste en, dadas las conexiones de los 2 clientes con el servidor, usar estas mismas conexiones para compartir los puntos finales de los 2 clientes (los pares IP, puerto) y conectar los clientes mediante esta conexión y otras auxiliares.

Teniendo en cuenta las características del juego, flujo de datos entre clientes bajo, número de jugadores por partida fijo y tamaño del mensaje pequeño, se ha decidido implementar la primera alternativa ya que el servidor tiene la suficiente potencia de cómputo.

El siguiente paso es elegir las tecnologías para implementar el sistema distribuido.

La parte cliente está implementada usando C# sobre Mono y la parte servidor C# sobre el CLR de Microsoft. Con esta elección se consigue que la interfaz de red que se proporciona al cliente sea la misma que se proporciona al servidor, el lenguaje C# y más concretamente la implementación de sockets sobre TCP.

Para realizar una comunicación correcta entre clientes y servidor se ha implementado el concepto de mensaje mediante la clase *NetworkMessage*. Esta clase se encuentra en una biblioteca de enlace dinámico (DLL, *dynamic-link library*) cuya finalidad es hacer que el cliente y servidor trabajen con las mismas clases y que el proceso de *marshalling* se haga correctamente en ambos extremos. Además en este DLL se encuentra la clase estática que almacena la información de puertos e IPs necesarios.

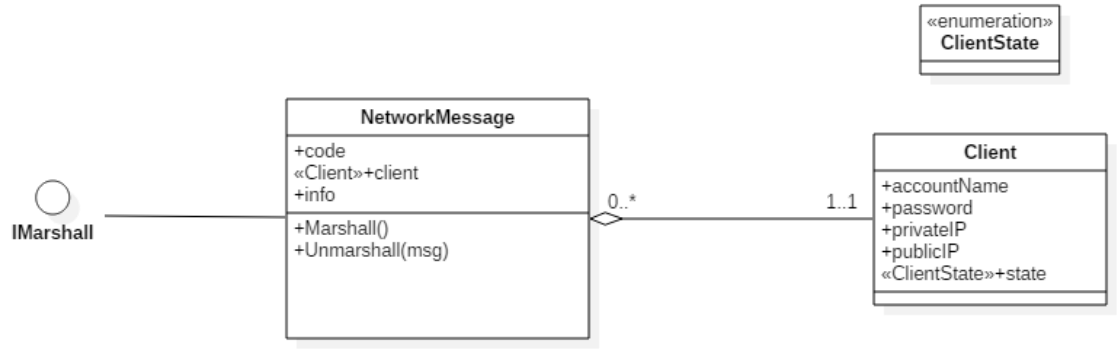

Figura 26. Diagrama de clases mensaje de red.

Todo mensaje de red está identificado por el cliente que lo envía, si este es erróneo se descarta el mensaje. El código del mensaje indica el tipo de servicio que se está solicitando, y se realizará si todos los parámetros del mensaje son correctos. El campo info es un array de bytes que contiene la información serializada necesaria para la realización del servicio.

Esta clase implementa la interfaz *IMarshall* que contiene los métodos necesarios para realizar el Marshalling. Gracias a que este proceso se encuentra en una biblioteca común al cliente y servidor ambos son capaces de gestionar los mensajes correctamente.

A continuación se explicará cómo está implementado el flujo de mensajes entre cliente y servidor detallando la arquitectura.

La tecnología usada en ambas partes, cliente y servidor, es la interfaz de sockets asíncronos de C#.

Cuando un cliente quiere iniciar sesión en el sistema abre un socket predeterminado que una vez validado el servicio, deja abierto escuchando mensajes del servidor. Para la petición de otros servicios una vez iniciada la sesión se usa otro socket. Este método hace que usando 2 sockets diferentes se consiga una comunicación bidireccional entre cliente y servidor. Además con el socket de sesión abierto el servidor puede saber en todo momento si el cliente se ha desconectado de manera natural o si ha habido algún error con la conexión, enviando los avisos necesarios a otros clientes y realizando la desconexión del mismo. Esto implica que el servidor mantiene una conexión de sesión abierta en todo momento por cada cliente conectado.

Además el núcleo de Unity no es *thread-safe*, por tanto fue necesario diseñar un mecanismo de sincronización entre los hilos que gestionan la comunicación y el hilo principal de Unity que ejecuta el bucle principal.

```
1  using UnityEngine;
2  using System.Collections.Generic;
3
4   public delegate void Task();
5
6  public class TaskScheduler : Singleton<TaskScheduler>
7  {
8      private static readonly int MAX_TASKS = 100;
9
10     private Queue<Task> _taskQueue = new Queue<Task>();
11
12     private void Update()
13     {
14         lock( _taskQueue )
15         {
16             if( _taskQueue.Count > 0 )
17             {
18                 _taskQueue.Dequeue()();
19             }
20         }
21     }
22
23     public void Schedule( Task task )
24     {
25         lock( _taskQueue )
26         {
27             if( _taskQueue.Count < MAX_TASKS )
28             {
29                 _taskQueue.Enqueue(task);
30             }
31         }
32     }
33 }
```

Figura 27. Task Scheduler.

Este mecanismo se basa en tener una cola de métodos anónimos que es consultada en cada frame. Cuando se recibe un mensaje y se requiere ejecutar un método anónimo para alertar del mensaje se guarda el método en esta cola. En cada frame se consulta la cola y se ejecuta el método anónimo. Esto hace que el tiempo de espera máximo entre que se recibe el mensaje y se ejecuta sea el tiempo total de un frame. Además una restricción que tiene impuesta es que sólo puede ejecutar un método por frame para no sobrecargar al sistema y perder rendimiento en la fase de renderizado del frame.

Este protocolo es un mecanismo *ad hoc* de comunicación entre procesos (IPC, *inter-process comunication*). Diseñar un sistema distribuido de esta forma tiene muchas limitaciones.

Una de ellas es que el nivel de dependencia entre la aplicación y el sistema de comunicación es muy grande. Por ello la usabilidad de este nivel con otra aplicación diferente es muy baja, es incluso difícil portar una versión del producto a otra. Para sistemas cuyos requerimientos varían mucho a lo largo de la vida del proyecto usar solo este nivel de distribución es insuficiente para alcanzar objetivos como usabilidad, escalabilidad, calidad de servicio (QoS, *quality of service*), etc.

Por todas estas razones, si el proyecto varía mucho a lo largo de su vida (como es el caso de la mayoría de productos software) la mejor opción para diseñar el sistema distribuido es crear un middleware que haga de puente entre el sistema operativo y el nivel de aplicación.

El sistema *ad hoc* diseñado es válido para un prototipo donde la finalidad sea profundizar en las mecánicas de juego, mientras que pensando en el futuro del producto y quizá de la empresa es interesante pensar en el desarrollo de un middleware como producto en sí mismo que sea un soporte para futuros proyectos.

Dadas las limitaciones del sistema de comunicación del videojuego al final del proyecto se ha empezado el diseño de un middleware de comunicación que tuviera 2 características principales, que permitiera abstraer la localización de los objetos remotos permitiendo al cliente interactuar con ellos de igual manera que lo hace localmente, y que permitiera implementar varios protocolos de comunicación para elegir el más adecuado en la aplicación que usara ese middleware.

Como arquitectura distribuida se ha usado el lenguaje de patrones de diseño **Broker** [4]. Este diseño permite aplicar un nivel de abstracción de localidad para los objetos remotos, además tiene un patrón muy importante dentro del lenguaje, el **Proxy** [2, 3, 4], que permite abstraer la serialización de los objetos para realizar el marshalling/unmarshalling de forma que el cliente no tenga necesidad de saber la implementación ni la arquitectura que está usando.

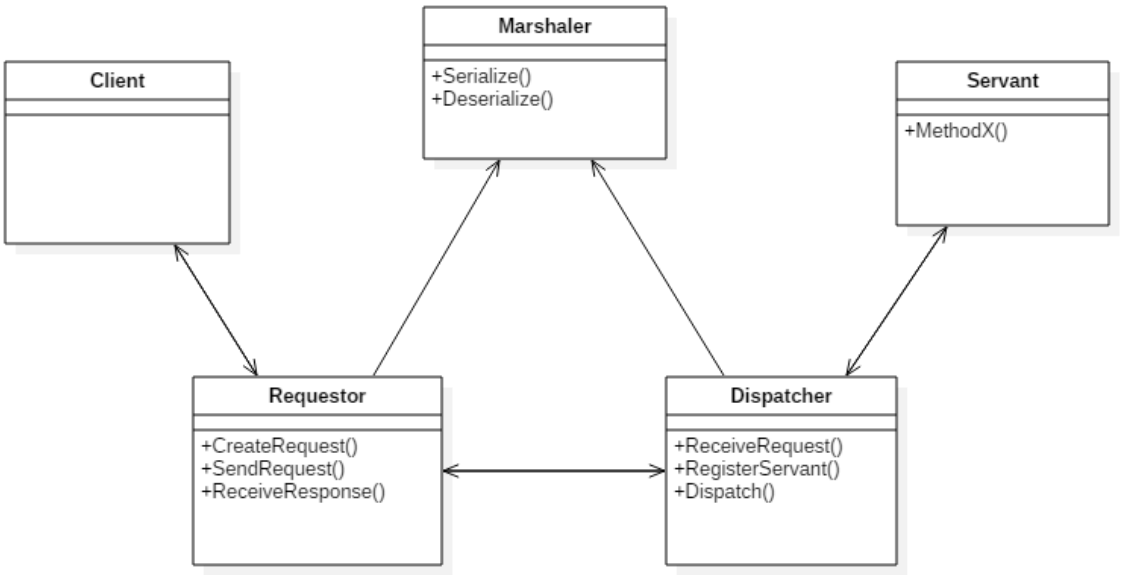

Figura 28. Broker pattern.

El sistema funciona de la siguiente manera. El paso previo a la comunicación es registrar el *Servant* del cual queremos hacer uso en el broker. El módulo del broker está principalmente compuesto por los elementos *Requestor* y *Dispatcher*.

El cliente obtiene del *Requestor* el proxy del objeto del cual quiere invocar el método. Una vez adquiere el proxy invoca al método de igual manera que si lo hiciera en un objeto local ya que el proxy implementa la misma interfaz que el objeto destino. Este objeto está representado por el rol de *Servant* en el diagrama. Esta invocación internamente lo que hace es serializar y enviar el mensaje a través de la red usando un protocolo de comunicación. El proxy es el encargado de este rol, *Marshaler*.

Una vez el *Dispatcher* recibe la petición, la despacha enviando al skeleton (la interfaz del *Servant*) los parámetros para la invocación del método. Una vez se completa la invocación el resultado de ella atraviesa las mismas etapas en sentido contrario. Finalmente el cliente recibe la respuesta.

El sistema distribuido estará implementado sobre C# de igual manera que lo está el sistema de comunicación usado en el videojuego. Para la fase de registro del objeto y de adquisición del proxy se crea un servidor de nombres cuya funcionalidad es almacenar los pares (nombre, referencia al objeto). Este servidor no es más que una clase que funciona de la misma forma que cualquier *Servant* en el broker.

Uno de los objetivos es abstraer el protocolo de comunicación usado por el sistema. La idea es implementar 3 protocolos, TCP, UDP confiable (RUDP) y HTTP.

La implementación de TCP es la normal en estos sistemas ya que se busca que el mensaje llegue siempre al destino y en orden, pero para sistemas cliente-servidor donde se intercambian mensaje-respuesta con mucha frecuencia o sistemas en tiempo real donde es importante el tiempo de respuesta, TCP no cumple los objetivos.

Para ello es capital implementar una capa segura sobre UDP, que permita asegurar la recepción de mensajes y su orden sin tener que establecer conexiones como hace TCP.

La idea de implementar HTTP está motivada en que hay sistemas de red cerrados que solo tienen abierto el puerto 80 y puede ser interesante tener un protocolo que pudiera comunicarse con esos sistemas.

Este diseño de sistema distribuido hace que sea reutilizable en otros proyectos en los que se trabaje con C# usando el CLR o Mono, haciéndolo muy interesante para ahorrar tiempo y costes en futuros proyectos.

5 Test

El testeo del videojuego se divide en 2 partes, medidas de los tiempos de respuesta del sistema de comunicación y análisis de la portabilidad de la interfaz gráfica en diferentes pantallas de distinta resolución.

Comunicación.

El testeo de la parte de comunicación entre cliente y servidor se divide en 3 fases, en la primera se recogen datos sobre el tiempo de respuesta en una red local, en la segunda separando el cliente y servidor en 2 redes diferentes y en la tercera separando al servidor y los 2 clientes en 3 redes diferentes. Todos los experimentos se han realizado utilizando comunicación sin cables.

Estas medidas están tomadas desde que el usuario realiza una acción hasta que el bucle principal de Unity recibe la respuesta, por tanto los datos tienen una pequeña sobrecarga de tiempo de como máximo la duración de un frame.

Además para mejorar el tiempo de ida y vuelta de los mensajes TCP se ha desactivado el algoritmo de Nagle [19] para mandar lo más rápido posible los mensajes sin esperar a que el buffer tenga que alcanzar un cierto tamaño.

Estos son los resultados del primer experimento, el cliente y el servidor se encuentran en la misma LAN. Este experimento se ha realizado usando wifi.

Nº	Tiempo de respuesta (ms)
1	121
2	142
3	123
4	122
5	125
6	134
7	133
8	132
9	141
10	147
11	141
12	134
13	136
14	130
15	143
16	122
17	142
18	133
19	122
20	119

Este es un escenario ficticio ya que el servidor estará en una red diferente de los clientes. Aun así es notable ver como los tiempos de respuesta son muy pequeños y aceptables para un videojuego por turnos, con una media de 132 ms.

El otro experimento de comunicación se ha realizado separando al cliente y servidor en 2 redes diferentes, además este es el mensaje más costoso de enviar ya que es el primero. Este tiene como finalidad mostrar una cota superior del tiempo de respuesta y se ha realizado usando 3G.

Nº	Tiempo de respuesta (ms)
1	990
2	1610
3	333
4	1453
5	1205
6	200
7	1754
8	1360
9	1600
10	340
11	1489
12	2320
13	1387
14	1429
15	2065
16	2553
17	1618
18	1605
19	1127
20	1335

Estos 2 experimentos muestran el caso mejor y el caso peor que el sistema tiene en las respuestas a los inicios de sesión. Este primer mensaje establece la conexión y verifica al usuario, las operaciones más costosas que realiza el servidor. Para escenarios en los que los 2 jugadores estén conectados y verificados los tiempos de respuesta son inferiores ya que el tamaño de los mensajes es menor y se reutilizan las conexiones abiertas por ambos.

En el siguiente experimento puede verse que con un tráfico de mensajes normal se consiguen tiempos de respuesta aceptables ya que los 2 jugadores están conectados y verificados. El experimento se ha realizado con el servidor en una red, un jugador en otra red usando wifi y otro jugador en una tercera red usando 3G.

Nº	Tiempo de respuesta (ms)
1	256
2	211
3	227
4	218
5	207
6	203
7	205
8	223
9	244
10	204
11	203
12	207
13	204
14	206
15	203
16	205
17	204
18	205
19	200
20	206

Estos experimentos nos demuestran que los tiempos de respuesta son aceptables si el flujo de mensajes no se incrementa en futuras versiones. Se trata de un juego por turnos en el que no se intercambia mucha información, a diferencia de los juegos en tiempo real.

A pesar de ello sería interesante cambiar el protocolo de comunicación usado o implementar uno nuevo. Pensando en un valor a largo plazo para futuros proyectos habría que implementar un protocolo fiable sobre UDP que permitiera personalizar el envío de mensajes para garantizar un envío cien por cien seguro o una tasa para videojuegos que se comuniquen en tiempo real haciendo un alto uso del ancho de banda de la red.

De esta manera se mejorarían los tiempos de respuesta para el tráfico de mensajes que continuamente actualizan el estado del sistema y se garantizaría la fiabilidad de envío y recepción para aquellos que sea capital que lleguen a su destino, haciendo más lentos sus tiempos de envío y respuesta.

Otra opción sería en vez de implementar un protocolo propio, integrar uno de terceros en el sistema. Uno de los más interesantes es UDT [21], un protocolo basado en UDP, fiable, altamente configurable y que está diseñado para redes de alta velocidad o sistemas distribuidos en tiempo real.

Interfaz gráfica.

Para testear la escalabilidad de la interfaz en varios tipos de pantallas con resoluciones diferentes se ha analizado el caso de ejecutar la aplicación en un PC con una pantalla de resolución 1920x1080 y en una tableta de 2048x1536. Estas son las capturas realizadas.

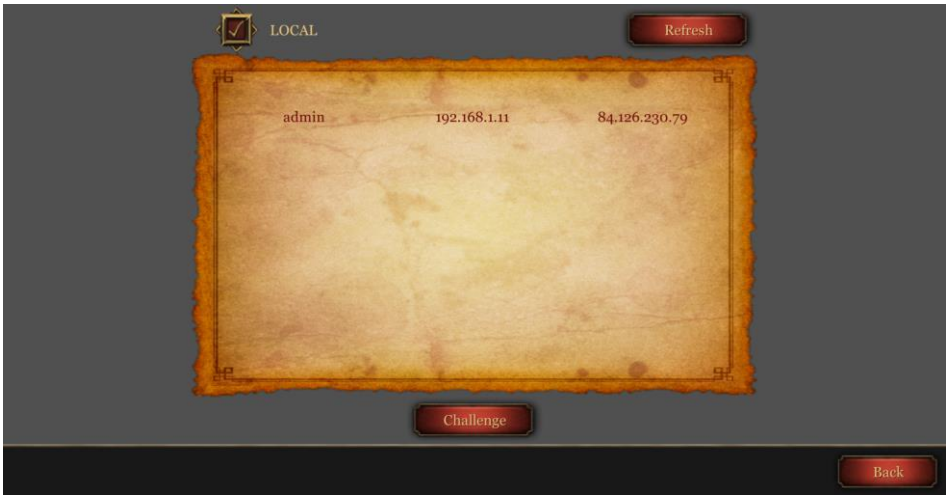

Figura 29. Captura en PC.

Figura 30. Captura en tableta.

En ambas capturas se puede observar como hay elementos que se escalan dependiendo del tamaño de la pantalla mientras que otros poseen unas texturas más complejas que basan su tamaño en hacer porciones de unas zonas de ellas. Este es el caso de los botones, las esquinas que forman el rectángulo del botón son simples texturas mientras que las zonas que se escalan son repeticiones de una misma porción o slice.

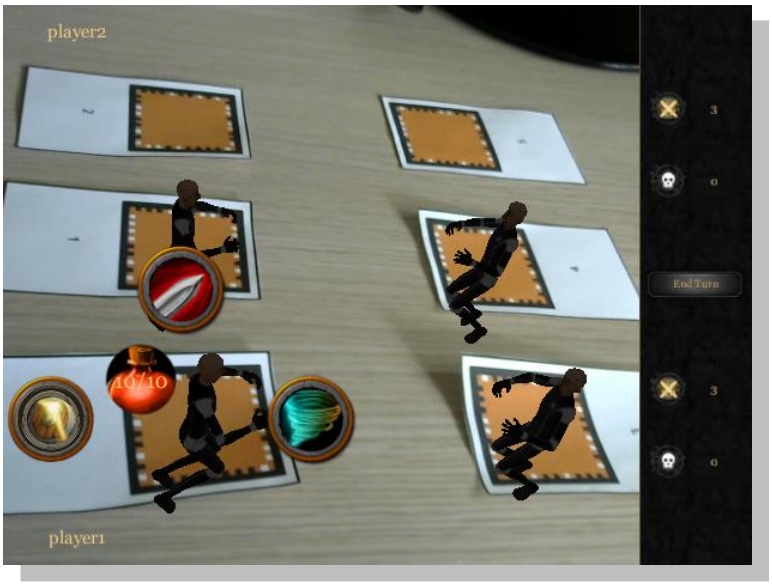

Figura 31. Captura de juego.

Otro ejemplo puede verse en el fondo del menú lateral. Este fondo no es más que una textura de tamaño pequeño con un pequeño ruido añadido. Esta textura se duplica las veces que sea necesaria para adaptarse a las dimensiones del menú, en este caso un 10% del tamaño de la pantalla.

Con estos resultados puede afirmarse que el sistema de diseño de interfaces gráficas que ofrece Unity cumple con los requisitos del proyecto y que permitirá realizar diseños más complejos.

6 Conclusiones

Se ha desarrollado un videojuego de cartas coleccionables y rol fantasía por turnos que cumple con los objetivos de ser multiplataforma, multijugador y añadirle valor al producto mediante la realidad aumentada.

La elección de los dispositivos en los que se ejecutaría el videojuego hizo aparecer el problema de la fragmentación. Para resolver este problema se eligió el motor de videojuegos Unity, esta elección ha cumplido los objetivos esperados de sencillez y portabilidad.

Por otra parte se seleccionó como plugin de realidad aumentada Vuforia. Una de sus más importantes características era su API común para Android e iOS. Este objetivo es muy importante para el desarrollo ya que permitirá abrir el mercado final del producto a un mayor número de dispositivos. Sin embargo el futuro cambio de modelo de negocio de Vuforia hace que para los siguientes proyectos sea muy importante buscar otras soluciones de realidad aumentada que cumplan con los objetivos iniciales para no tener que depender de una única solución.

Para resolver la parte de comunicación se ha optado por implementar una solución distribuida basada en un *broker* que realice la abstracción de la capa de red de la aplicación. Además se ha propuesto implementar o integrar un protocolo fiable sobre UDP, que resuelva los problemas de ancho de banda que el sistema pueda tener.

En conclusión, se trata de un videojuego con potencial, pertenece a un género que no está masificado, el uso de la tecnología de realidad aumentada ofrece un valor añadido frente a otros títulos similares, lo que es ventajoso a la hora de atraer nuevos usuarios, y es viable expandir el contenido del juego con futuros parches para alargar la vida de este y aumentar los beneficios.

7 Bibliografía

[1] Game Engine Architecture.
Jason Gregory.

[2] Design Patterns: Elements of Reusable Object-Oriented Software.
Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.

[3] Game Programming Patterns.
Robert Nystrom.

[4] Pattern-Oriented Software Architecture Volume 4.
Frank Buschmann, Kevin Henney, Douglas C. Schmidt.

[5] *Computer Networks.*
Andrew S. Tanenbaum, David J. Wetherall.

[6] Peer-to-Peer Communication Across Network Address Translators.
Bryan Ford.

[7] The History of Mobile Augmented Reality
Institute for Computer Graphics and Vision Graz University of Technology, Austria.
Clemens Arth, Lukas Gruber, Raphael Grasset, Tobias Langlotz, Alessandro Mulloni, Dieter Schmalstieg, Daniel Wagner

[8] Unity.
http://unity3d.com/unity

[9] Vuforia.
https://www.qualcomm.com/products/vuforia

[10] AEVI.
Asociación Española de Videojuegos.
http://www.aevi.org.es/

[11] DEV.
Desarrollo Español de Videojuegos.
http://www.dev.org.es/es/la-asociacion

[12] Pay to Play, Wikipedia
https://es.wikipedia.org/wiki/Pay_to_play

[13] Free to Play, Wikipedia
https://es.wikipedia.org/wiki/Free_to_play

[14] Modding, Wikipedia.
https://es.wikipedia.org/wiki/Modding

[15] Peer to Peer, Wikipedia.
https://es.wikipedia.org/wiki/Peer-to-peer

[16] Inter-Process Communication, Wikipedia.
https://es.wikipedia.org/wiki/Comunicaci%C3%B3n_entre_procesos

[17] Localización, Wikipedia.
https://es.wikipedia.org/wiki/Localizaci%C3%B3n_de_idiomas

[18] Máquinas de Estados, Wikipedia.
https://es.wikipedia.org/wiki/M%C3%A1quina_de_estados

[19] Algoritmo de Nagle, Wikipedia.
https://es.wikipedia.org/wiki/Algoritmo_de_Nagle

[20] Fragmentación de dispositivos, Wikipedia.
https://en.wikipedia.org/wiki/Fragmentation_%28programming%29

[21] UDT, UDP-based Data Transfer.
http://udt.sourceforge.net/