UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño e implementación de una app de Realidad Aumentada como apoyo al motor de Realidad Virtual de la UPV

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Marcos Haba Claramunt
Tutor: M. Carmen Juan Lizandra
Dir. Experimental: Ferrando Serrano Capena

Curso 2019-2020

Resum

La Realitat Augmentada ja és una tecnologia present en el nostre dia a dia. Hi ha aplicacions en la indústria, la salut, l'educació i l'entreteniment. S'investiga i es desenvolupa contínuament mitjançant l'addició de noves característiques i l'augment del seu rendiment. Telèfons mòbils, cascs i ulleres són proves del progrés i adaptació d'aquesta tecnologia a diferents dispositius. A més, com veurem, té un passat no tan recent com podria semblar.

Aquest treball pretén dur a terme una prova de concepte, per a la Universitat Politècnica de València, utilitzant la Realitat Augmentada per a telèfons mòbils Android. L'objectiu de la mateixa és dotar a la universitat d'una eina complementària al seu motor de Realitat Virtual que es pugui utilitzar a discreció de l'usuari.

Per a això desenvoluparem una aplicació mòbil, en el motor de desenvolupament de videojocs Unreal Engine, i un complement al motor de realitat virtual de la universitat, que ho comunique amb el telèfon mòbil, per aconseguir la descàrrega directa dels models 3D d'aquest motor al nostre dispositiu.

Paraules clau: Realitat Augmentada, Realitat Augmentada Mòbil, Unreal Engine

Resumen

La Realidad Aumentada ya es una tecnología presente en nuestro día a día. Existen aplicaciones en la industria, la salud, la educación y el entretenimiento. Se investiga y desarrolla continuamente añadiendo nuevas características y aumentado su rendimiento. Los teléfonos móviles, los cascos y las gafas son pruebas del progreso y la adaptación de esta tecnología a diferentes dispositivos. Además, como veremos, tiene un pasado no tan reciente como podría parecer.

El presente trabajo trata de realizar una prueba de concepto, para la Universitat Politècnica de València, usando la Realidad Aumentada para teléfonos móviles Android. El propósito de la misma es el de dotar a la universidad de una herramienta complementaria a su motor de Realidad Virtual que se pueda usar a discreción del usuario.

Para ello desarrollaremos una aplicación móvil, en el motor de desarrollo de videojuegos Unreal Engine, y un complemento al motor de Realidad Virtual de la universidad, que comunique el mismo con el teléfono móvil, para lograr la descarga directa de los modelos 3D de este motor a nuestro dispositivo.

Palabras clave: Realidad Aumentada, Realidad Aumentada Móvil, Unreal Engine

Abstract

Augmented Reality is already a technology present in our day to day. There are applications in industry, health, education and entertainment. It is continuously researched and developed by adding new features and increasing their performance. Mobile phones, helmets and glasses are tests of the progress and adaptation of this technology to different devices. Besides, as we will see, it has got a past not as recent as it might seem.

This work seeks to carry out a proof of concept, for the Universitat Politècnica de València, using the Augmented Reality for Android mobile phones. Its purpose is to provide the university with a tool complementary to its Virtual Reality engine that can be used at the user's discretion.

To do this we will develop a mobile application, in the Unreal Engine video game development engine, and a complement to the virtual reality engine of the university, which communicates it with the mobile phone, to achieve direct download of the 3D models of this engine to our device.

Key words: Augmented Reality, Mobile Augmented Reality, Unreal Engine

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

El presente trabajo trata sobre el desarrollo de una prueba de concepto de RA (*Realidad Aumentada*) para móviles Android mediante el motor de videojuegos Unreal Engine[1]. La aplicación resultante tendría como función suplir la necesidad de los usuarios del motor de RV (*Realidad Virtual*) de la UPV para tener un medio en el que probar sus diseños antes de acceder a él. De este modo se reducirían los tiempos de espera, gestiones por parte del profesorado y errores fácilmente solucionables de modelado 3D.

En este apartado se expondrá la motivación detrás de este proyecto, los objetivos a cumplir para llevarlo a buen término y la estructura de esta memoria.

1.1 Motivación

El trabajo realizado como técnico de apoyo a servicios, en el ASIC (*Área de Sistemas de Información*) en la UPV, me brindó la oportunidad de conocer a Fernando Serrano Capena, especialista técnico multimedia, que me mostró su trabajo con el motor de VR Bg2 Engine[2] y la herramienta para trabajar con él, llamada Bg2 Composer. Esta utilidad se usa en el Máster Universitario en Ingeniería del Diseño, más concretamente en la asignatura de "Visualizaciones Avanzadas. Realidad Virtual Aplicada al Diseño de Productos", impartida por María Begoña Saiz Mauleón, para mostrar elementos 3D diseñados por alumnos en un entorno virtual y trabajar sobre ello.

Fernando habló conmigo sobre el interés que tenía en aumentar la funcionalidad de la herramienta y en la utilidad que se le podía dar. Con ello, hizo hincapié en los principales problemas que tenía. Por un lado, existía la necesidad de disponer de un espacio reservable en el que, además, los alumnos deben compartir por turnos unas gafas de VR. Por otro lado, la cantidad de errores que surgían, al no disponer de un medio en el que poder corregirlos previamente a utilizar las gafas de RV, hacía que estos tiempos de uso fuesen más altos, retrasando en una reacción en cadena a todos los usuarios.

Además, debido a la pandemia del COVID-19, y el aumento del teletrabajo, hace que ahora sea especialmente interesante desarrollar una alternativa al trabajo tradicional *in situ*.

Nos encontramos ante una tecnología en constante evolución y con un rápido crecimiento, pasando; por ejemplo, de una facturación en el mercado estadounidense de AR y VR de menos de 12 millones de dólares en 2019 a una proyección de más de 400

[1] https://www.unrealengine.com
[2] https://www.bg2engine.org

hacia 2025[3] en el mercado de las app de inmersión colaborativa y de videoconferencias únicamente.

Finalmente, según Michael Inouye, *Principal Analyst* de ABI Research «*While no one knows for sure how much the COVID-19 pandemic will impact us, there is strong evidence to support there will be long-lasting effects [...]. Some companies, like Facebook, have already amended work from home policies that extend beyond the current pandemic and provide early glimpses into what a new normal might look in the workplace*»[3]. Debemos sumar a esto no solo el valor *per se* de esta industria, sino también la influencia que tendrá sobre la próxima Industria 4.0 [1] [2].

1.2 Objetivos

El principal objetivo a cumplir en este Trabajo Fin de Grado es desarrollar una prueba de concepto, previa a una posible aplicación, de RA para dispositivos móviles Android que complemente a la herramienta Bg2 Composer. Ésta debe ser capaz de mostrar modelos 3D desarrollados por alumnos o personal de la UPV. Con ello se pretende dotar al usuario de una forma de identificar sus errores de diseño en el modelo 3D y, por lo tanto, ser capaz de corregirlos en un *software* externo antes de utilizar las gafas de RV. Para alcanzar este objetivo principal se han establecido los siguientes subobjetivos:

- Desarrollar un *plugin*[4] para Bg2 Composer que funcione como servidor local dentro de la misma red del teléfono móvil. Este complemento se encargará de suministrar al dispositivo móvil los elementos que compondrán lo que desea visualizar el usuario en RA.
- Construir en Unreal Engine un *plugin* que comunique con el servidor local, descargue un primer elemento índice y lo almacene en el teléfono móvil.
- Implementar en este complemento el *plugin* Bg2UnrealTools[5], incluido dentro de las herramientas de desarrollo de Bg2 Engine. De forma que se descargarán el resto de elementos indicados en el índice y pasarán a componer el entorno 3D. Esto dará como resultado la experiencia de RA que estamos buscando.

1.3 Estructura de la memoria

La presente memoria se compone de 6 capítulos y la bibliografía. El documento se estructura como sigue:

En el primer capítulo tratamos de introducir el trabajo, la motivación por el que se hace, los objetivos a cumplir y esta misma explicación de la estructura de la memoria.

En el segundo capítulo expondremos el estado del arte. Aquí nos detendremos en la historia de la RA, la evolución y la situación de la industria RA. Además, veremos aplicaciones que comienzan a usarse en este sector.

En el tercero analizaremos los requisitos del trabajo a realizar así como las diferentes soluciones posibles para llevarlo a cabo. Por último, se desarrollará en profundidad la solución propuesta.

[3] https://prn.to/2WT3YP9
[4] https://es.wikipedia.org/wiki/Complemento_(inform%C3%A1tica)
[5] https://github.com/ferserc1/bg2-unreal-tools

Seguiremos, en el cuarto capítulo, con las decisiones a tomar sobre cómo se ha de realizar y diseñar la solución del trabajo. Además, hablaremos de la arquitectura del sistema a seguir.

En el quinto capítulo pasaremos a analizar los resultados de este trabajo.

A continuación, nos dedicaremos a pormenorizar las conclusiones obtenidas en el desarrollo de este proyecto, lo que hemos aprendido y la relación del trabajo con nuestros estudios. Además, nos detendremos en los posibles trabajos futuros, ampliaciones y añadidos, que hemos encontrado interesantes.

Por último, contamos con un apéndice, en la parte final, para entender y replicar la instalación de este trabajo.

CAPÍTULO 2
Estado del arte

En este capítulo se detalla el significado de la RA, la evolución histórica que ha llevado, la situación de esta tecnología y aplicaciones que comienzan a utilizarse en sectores como la medicina, la educación y el entretenimiento.

2.1 Entendiendo la Realidad Aumentada

Hay muchas interpretaciones sobre qué es esta tecnología. Sin embargo, hemos optado por restringir esta definición a lo más apropiado a nuestro interés. Por lo tanto, y basándonos en la definición de Azuma [3], podemos decir que la RA:

Permite al usuario ver el mundo real con elementos virtuales superpuestos, o compuestos, sobre este mundo real. Por lo tanto, la RA complementa a la realidad, en lugar de reemplazarla. Idealmente, al usuario debería parecerle que los objectos reales y virtuales deberían coexisistir en el mismo espacio. Entonces, la diferencia de la RA y la RV estriba en que mientras la primera complementa a la realidad, permitiendo al usuario visualizar el mundo real junto con el virtual, la segunda sumerge al usuario únicamente en un entorno virtual.

Es interesante señalar que, aunque no lo cubriremos aquí, la RA no se aplica únicamente al sentido de la vista; se puede aplicar también al resto de los sentidos.

2.2 Evolución de la Realidad Aumentada

El desarrollo de las tecnologías de RV y RA han sido prácticamente indistinguibles desde sus inicios. No fue hasta finales de la década de 1980 cuando se habló del término de *Realidad Virtual*, atribuido a Jaron Lanier en una conferencia anual de ACM[1] [4]. Algo similar ocurrió en 1992, donde se cree que Tom P. Caudell acuñó el término que nos atañe [5], en la HICSS[2]. Esto nos permite entrever que no había una distinción clara entre ellas previamente a estas puntualizaciones.

[1] https://www.acm.org/
[2] https://hicss.hawaii.edu/

Pero, evidentemente, se trabajó y se imaginó con estas tecnologías con anterioridad. Ya en 1901 Frank L. Baum, conocido por ser el autor de «El maravilloso mago de Oz» (adaptado al cine como «El mago de Oz»), hablaba de unas gafas que, mientras las llevabas puestas, a cada persona que te encuentrobas se le marcaba en la frente con una letra indicativa de su carácter (sea bueno, malo, etcétera).

En 1935 Stanley G. Weinbaum publicó un relato corto de ciencia ficción llamado «*Pygmalion's Spectacles*» [3], considerado el primer registro literario de lo que es ahora la RV, en el que se describían unas gafas que hacían al portador sentir un mundo con olor, gusto y tacto ficticios.

Fue en 1960 cuando Morton L. Heilig patentó el «*Stereoscopic-television apparatus for individual use*» [4] (figura 2.1). Heilig no sólo construyó un aparato que mostraba una visión ficticia al portador, sino que también le proporcionaba audio y olores.

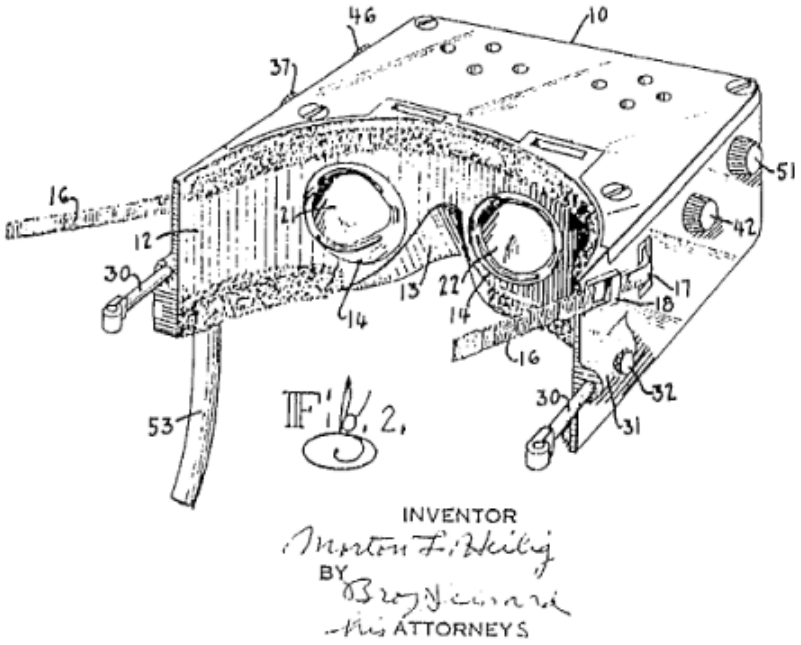


Figura 2.1: Aparato individual de Morton L. Heilig

En el año 1962, patentó el «*Sensorama Simulator*»[5] (figura 2.2), en el que según el propio Heilig era una invención relacionada con una especie de aparato simulador. Más concretamente, era un aparato que estimulaba los sentidos (la vista, el olfato y el oído) del usuario para vivir una experiencia irreal de forma realista.

[3] https://es.wikipedia.org/wiki/Las_gafas_de_Pigmali%C3%B3n
[4] https://patentimages.storage.googleapis.com/81/df/f1/f6cc2106f8c7ab/US2955156.pdf
[5] https://patentimages.storage.googleapis.com/90/34/2f/24615bb97ad68e/US3050870.pdf

Con este aparato simulador, Heilig, ya hablaba de la idea de instruir a los usuarios a través de este tipo de máquinas, evitando así los riesgos que implica el aprendizaje en según qué trabajos o para adaptarse de forma rápida a los cambios. Ponía al ejército, a la industria y a las instituciones educativas como potenciales demandantes de este tipo de recurso.

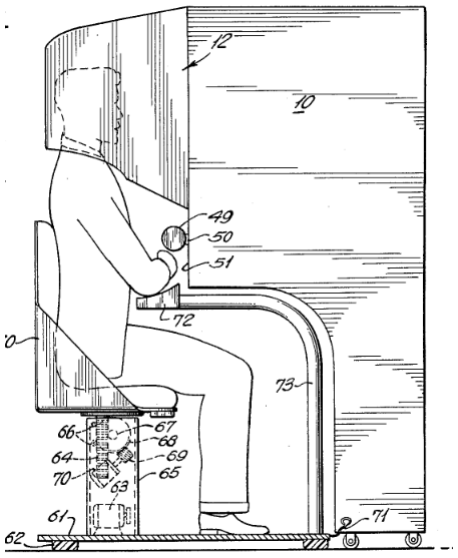
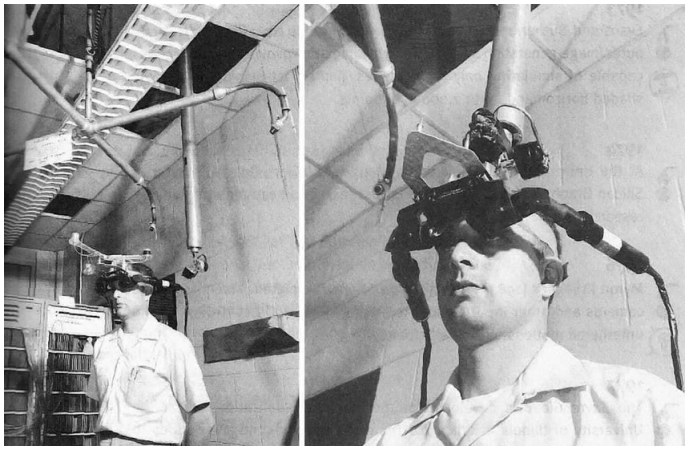


Figura 2.2: Sensorama de Morton L. Heilig

En el año 1968 Ivan Sutherland desarrolló lo que sería el primer prototipo de RA [6] (figura 2.3). Incorporaba un sensor de posicionamiento de la cabeza. Este podía ser mecánico o ultrasónico y ambos servían para estabilizar los sencillos gráficos que se proyectaban en el mundo real. Este fue, además, el primer HDM (*Head-mounted display* [Casco de visualización]) reconocido, un casco con unas lentes conectado a un sensor de movimiento que representaba gráficos en 3D.

(a) Visor de RA



(b) Sensor mecánico y ultrasónico

Figura 2.3: Prototipo de Sutherland

Myron Krueger presentó en 1975 «*Videoplace*»[6] [7] (figura 2.4). La tecnología que estaba detrás de esto fue denominada como Realidad Artificial. *Videoplace* consitía, según sus creadores, en un entorno gráfico computerizado en el que el participante ve su imagen en vivo proyectada sobre una pantalla de vídeo. Podía estar sola en esta pantalla, o podía haber imágenes de otras personas en diferentes lugares de la misma. Adicionalmente, se podían añadir objetos y criaturas virtuales que interactuaban con la imagen del usuario.

Videoplace fue la culminación de varios proyectos[7], Glowflow, Metaplay y Psychic space, de Realidad Artificial en los que participó Krueger.

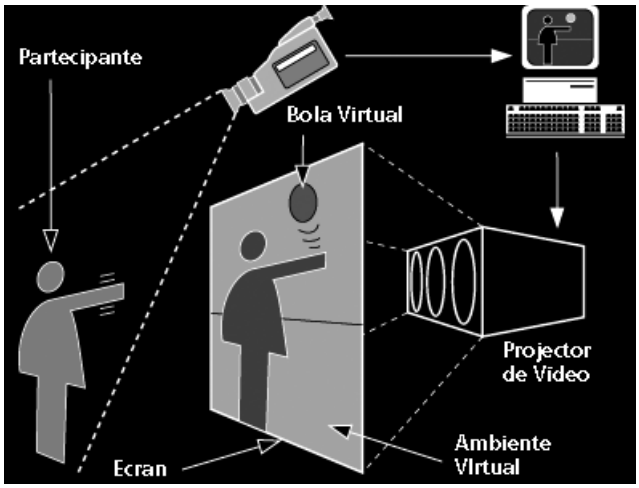


Figura 2.4: *Videoplace*

Al principio de este apartado hemos hablado de Lanier y Caudell. Ellos no solo lograron aportar el nombre que usamos ahora para las tecnologías de RV y RA, respectivamente. Jaron Lanier fundó VPL, compañía que desarrolló y vendió productos de RV.

En cambio, Caudell, en la HICCS de la que hemos hablado previamente [5], basó su trabajo (figura 2.5) en el concepto de proporcionar un modo de ver la RV, a través de unas gafas, al trabajador de una fábrica de Boeing para hacer su trabajo más fácil y eficiente. Además, este dispositivo sería usado para aumentar el campo visual del trabajador con ayudas e información dinámicas respecto al entorno.

Presentó este aparato como un HUDset (*Head-up display*, HUD [Visualización frontal]), esencialmente un HMD que presenta una interfaz de ayuda que está presente allá donde mire el usuario. Caudell vió este aparato como una iteración más del trabajo de Sutherland [6] y Robinett et al. [8].

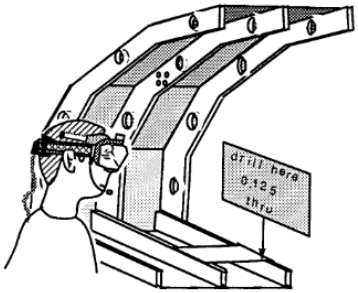


Figura 2.5: *HUDset* de Caudell

[6] https://youtu.be/d4DUIeXSEpk
[7] https://en.wikipedia.org/wiki/Myron_W._Krueger

Sin embargo, la RA no era barata de producir; muchos dispositivos necesitaban apoyo por parte de computadores externos, como el *HUDset* de Caudell ya en 1992. Por lo tanto, esto no se tradujo en que esta tecnología fuese llevada al gran público.

Steve Feiner et al. [9] empezaron a trabajar en 1996 en uno de los primeros sistemas móviles de RA, el MARS[8] (*Mobile Augmented Reality Systems*) (figura 2.6). Este proyecto hacía hincapié en que fuese viable comercialmente, sacrificando mayormente la precisión, y centrado en el diseño de la interfaz; usando para ello el hardware disponible en el mercado. En 1997 crearon un prototipo que proporcionaba al usuario información a su alrededor, convirtiéndo este dispositivo en una *máquina turística* de su universidad. Se usaba una combinación de GPS y sensores de orientación para saber su posición y, la información, se presentaba en combinación con un casco con lentes, que era rastreado, y una tableta.

(a) Equipo de la *máquina turística*

(b) Visión de la *máquina turística*

Figura 2.6: Proyecto MARS de Feiner et al.

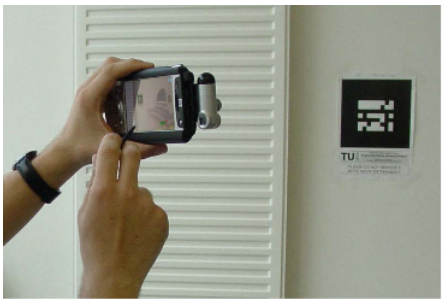
Kato y Billinghurst desarrollaron ARToolKit[9] [10] en 1999, un SDK que se desarrollaría en los años venideros hasta 2015, cuando se vendió. Sin embargo, en 2017 ARToolworks creó artoolkitX[10] para continuar dando soporte al *software* ya desarrollado de ARToolKit y su comunidad.

Thomas et al., en su desarrollo de ARQuake[11] [11], dijeron que ARToolKit era un SDK de seguimiento de visión por computador que podía usarse para calcular la posición y la orientación de una cámara, en tiempo real, en relación con unos marcadores físicos. Entre sus características también incluía el uso, para una única cámara, del seguimiento de posicionamiento y orientación, marcadores de referencia que podían ser cuadrados negros sencillos, *software* de *pattern matching* que permitía usar cualquier patrón de marcadores, calibración para aplicaciones de vídeo y suficiente optimización para usarse en aplicaciones de RA en tiempo real.

[8] http://monet.cs.columbia.edu/projects/mars/
[9] http://www.hitl.washington.edu/artoolkit/
[10] http://www.artoolkitx.org/
[11] https://youtu.be/No7QF0MwSjg

Como hemos dicho, esta herramienta se utilizó en multitud de nuevos proyectos, como ARQuake [11] (primer juego al aire libre, del 2000) o el primer sistema de RA, desarrollado en el 2003 por Wagner y Schmalstieg [12], completamente independiente y autosuficiente, usando para ello una PDA.

(a) Equipo de ARQuake



(b) Prototipo de Wagner et al.

Figura 2.7: ARQuake y prototipo de Wagner et al.

Fue en esta época, a finales de la década de los 90, cuando esta tecnología tomó impulso y cobró más relevancia. Se fundaron nuevas conferencias dedicadas a la RA, como ISMAR (*International Symposium on Mixed and Augmented Reality*)[12], y se hizo más fácil el desarrollo de aplicaciones. Además, numerosos investigadores comenzaron a aportar estudios y problemas relacionados con la RA, como Azuma [3] en 1997 o junto Baillot et al. [13] en el 2001 o, más reciente, el artículo de 2010 de Carmigniani et al. [15].

2.2.1. Situación actual

La aparición de los llamados *smartphones* (teléfonos inteligentes), y su gran popularidad en detrimento de los teléfonos móviles anteriores[13] a finales de la primera década (y principios de la segunda) del 2000, ha supuesto lo que ya se estaba buscando en los trabajos de Wagner y Schmalstieg [12] y Feiner [9]; un dispositivo potente, portátil, autosuficiente e independiente. La vida cambió con estos nuevos móviles y la RA se ha beneficiado claramente de esta aparición.

No solo eso, sino que, investigadores de RA siguieron trabajando y sacaron nuevos *kits* de herramientas más allá de ARToolKit, del que ya hemos hablado anteriormente. ARCore[14] y ARKit[15], ArUco [16] basado en OpenCV, Vuforia[17] y Wikitude[18], han marcado el sentido del desarrollo de RA en la última década hacia una precisión y realismo que llevaba intentando conseguirse desde los inicios de esta tecnología.

La RA ha demostrado ser capaz de beneficiarse de la tecnología que le ha ido proporcionando el mercado y, con la potencia de los dispositivos móviles, se ha usado para desarrollar aplicaciones que la han hecho más conocida al gran público. Por ejemplo, el caso de *Pokémon Go* [33], creado en 2016, ha seguido sumando año a año más beneficios y usuarios[19].

[12] https://www.ismar.net/
[13] https://bit.ly/3jAgMnm
[14] https://developers.google.com/ar
[15] https://developer.apple.com/augmented-reality/arkit/
[16] https://www.uco.es/investiga/grupos/ava/node/26
[17] https://developer.vuforia.com/
[18] https://www.wikitude.com/
[19] https://www.businessofapps.com/data/pokemon-go-statistics/

Ya hemos mostrado el crecimiento previsto a consecuencia de la situación actual, en el apartado de "Motivación"(capítulo 1), y su implicación en la industria 4.0. Pero además, se estima que crezca desde 8.810 millones de dólares estadounidenses en 2019 a 12.620 en 2020[20], un incremento del 43,19 % en el mercado global. Esta misma agencia postula que en el año 2023 el mercado habrá crecido en un porcentaje del 66,19 %, llegando a unos 57.920 millones de dólares, debido a la crecida de demanda del sector de videojuegos, entrenamiento y educación, y *marketing*.

Según Reportlinker[20], las compañías envueltas en la creación de *software* de RA están principlamente enfrascadas en el diseño, en el desarrollo y en la investigación del mismo. Hasta el momento se ajustaba la RA a un propósito específico, ahora hemos llegado al punto en el que se debe elaborar un sistema aplicable a un conjunto más amplio de situaciones [17]; desarrollar aplicaciones más amplias y adaptables.

El *software* de RA está empezando a introducirse en el sector de la salud [18] [19]. Esta tecnología, por ejemplo, puede beneficiar a las personas con visión limitada, o prácticamente con una discapacidad visual completa, en la realización de tareas sencillas sin más ayuda. También, hay estudios [20] que indican que puede ser útil para tratar fobias o para ayudar a niños con diabetes [21]. Sin embargo, es un campo a explorar aún, pese al prometedor futuro que se aguarda en su utilización.

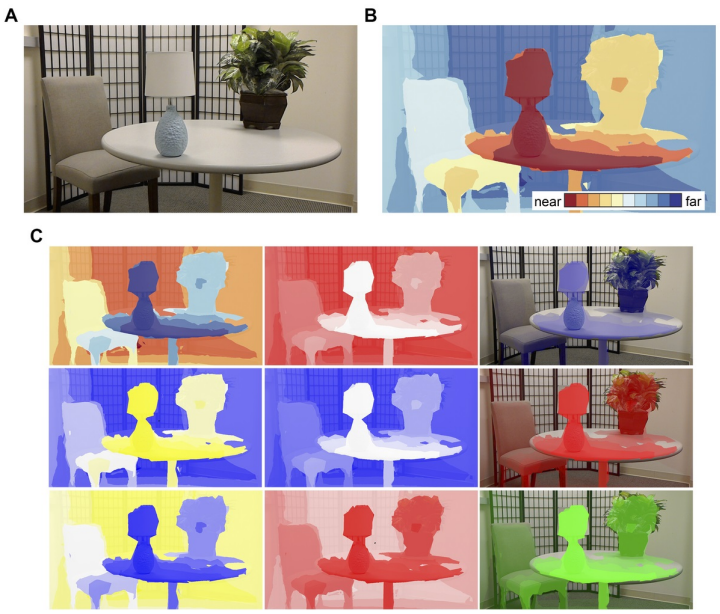


Figura 2.8: Ayuda para reconocer formas y distancias

La industria del entretenimiento ha salido muy beneficiada con la RA [33] [34]. Como hemos señalado antes, *Pokémon Go* ha supuesto un antes y un después en el sector del videojuego. Además, juegos como este benefician a la salud de sus jugadores, dejando entrever que hay beneficios indirectos con la salud. Pero no sólo tenemos este caso, la RA se puede aplicar a juegos con propósitos educativos, como *Dragon Tale* [35], un juego para aprender los kanjis japoneses.

[20] https://bit.ly/2D51bv8

En la educación también podemos encontrar nuevos intentos de introducir esta tecnología [22] [23] [24] [25] [28] [29] [30] [31] [32]. Hay nuevas ideas para apoyar la formación de una forma más práctica y visual en colegios, institutos y universidades, o en museos, zoos, etcétera. Aplicando información gráfica, explicaciones y modelos en 3D, entre otras técnicas, es posible conseguir una participación mayor de los usuarios, observación activa, trabajo en equipo y autodidactaismo. Incluso, se investigan los resultados de usar juegos educacionales tradicionales y móviles [26] o los efectos que producen en ellos el peso y el tamaño de estos dispositivos [27].

Figura 2.9: RA en la educación

El comercial también busca adaptar la RA a su campo. Soluciones [36] [37] [38] que investigan el impacto que puede tener en el *marketing* y su aspecto diferenciador en la marca que use esta tecnología, haciendo que el consumidor se decante por los productos actuales, facilitando la toma de decisión antes de la compra, e inspirar confianza para futuras compras, o también para crear una comunidad de usuarios y que exista una interacción entre ellos y la marca.

Figura 2.10: Anuncio usando RA

De igual forma, la industria aeroespacial comienza a entender los beneficios de la RA [42] [43]. La precisión necesaria en la fabricación y ensamblaje de piezas de los transbordadores espaciales son aquí especialmente importantes y, por lo tanto, se intenta integrar en este proceso como una ayuda para reducir estos errores, que pueden ser tan costosos, y resolverlos de forma sencilla.

Incluso el sector militar tiene interés en la utilización de esta tecnología. Se está buscando crear un sistema [39] [40] de RA que facilite al ejercito y, más concrétamente, al soldado de toda la información posible; desde un mapa del terreno al estado de la batalla. También para prevenir muertes en el campo de batalla debido a fallos de los médicos de combate [41] debidos a la inexperiencia o al estrés.

Por lo tanto, podemos decir que estamos ante una tecnología ya asentada y que, además, se encuentra en una situación de crecimiento de ventas, uso y desarrollo. Se utiliza por cada vez más compañías para, además, aportar un factor de diferenciación respecto a sus rivales. Es un hecho que multinacionales como Google, Microsoft o el grupo Alibaba, aumentan su interés en la RA y se espera que, por ejemplo, en el caso de este último adquiriendo InfinityAR[21] en 2019, aporten más beneficios tanto a ellos mismos como a la industria.

Sin embargo, como ya advertía Azuma en el año 2001 [13], los problemas de seguridad asociados a la privacidad hacen que su tasa de adopción sea baja. Esto, junto a la complejidad del diseño de las aplicaciones de RA, se espera que lastre el crecimiento del que hemos hablado con anterioridad.

Pero, opinamos que aunque se respalde con estudios[22] el aumento de la preocupación de los usuarios por su privacidad, vemos que no ha alcanzado un consenso claro en el debate actual. Es un hecho bien sabido, por ejemplo, el uso que ha hecho Facebook con los datos[23] y su incremento, al parecer, imparable de usuarios[24]. Instagram (adquirido por Facebook en 2012) usa AR Spark para crear filtros y efectos de RA en las publicaciones de los usuarios. En 2019 se supo de más fallos de privacidad[25], eso sí, de cuentas públicas (abiertas a todo usuario que quiera ver su contenido). Por lo que, ¿hasta qué punto es consciente la sociedad de esto? ¿y de qué forma puede afectar este hecho si el uso de la RA, en este tipo de aplicaciones[26], aumenta pese a estos fallos y filtraciones?

Vemos con escepticismo las advertencias que se hacían con anterioridad, los estudios publicados sobre el tema y la conciencia de los usuarios sobre su privacidad. Aunque, de todas formas, pensamos que es cuestión de tiempo que se llegue a un verdadero debate público y se informe de forma más profunda sobre el uso de nuestros datos. Suponemos que, en un futuro próximo, estos hechos formarán parte de nuestra vida e interiorizaremos la protección de nuestra identidad en la red, como nos hemos adaptado a tecnologías anteriores.

Con esta breve opinión queremos indicar que la privacidad no sólo incumbe, como hemos intentado transmitir, a la RA, sino al conjunto de esa sociedad virtual que se ha ido generando en internet.

2.3 Motores de Videojuegos

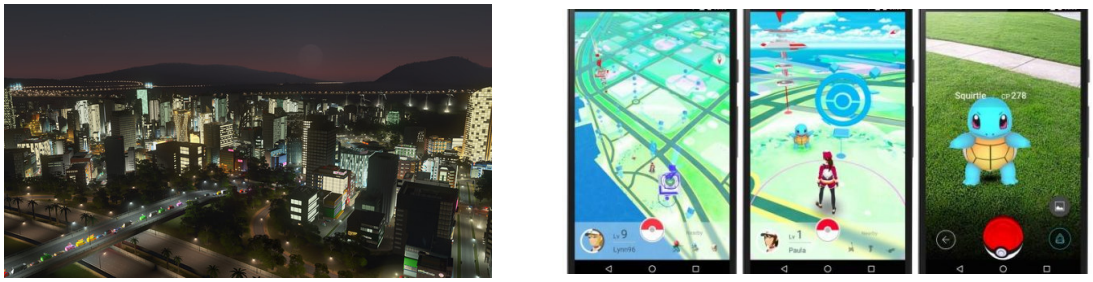
Los videojuegos han sido uno de los impulsores de los gráficos y los gráficos 3D. No cubriremos su evolución histórica, pero sí que es cierto que tanto la RA como la RV se benefician de estos avances. Unity, lanzado en 2005, y Unreal Engine, en 1998, son dos de los motores que se usan para el desarrollo de videojuegos y, al igual que estos, para la creación de aplicaciones de RA entre otras muchas cosas.

Unity puede implementar, a través de su *Package Manager* (un contendor de *plug-ins*, etcétera), AR Foundation; una solución que ofrece funcionalidad básica de ARCore y ARKit en una API común en su entorno. Si bien, el desarrollador puede ampliar la funcionalidad de este complemento si lo ve necesario. Es posible añadir el SDK con las herramientas necesarias para su funcionamiento (en este caso ejemplos de uso, etc), de ARCore y ARKit.

[21] https://www.crunchbase.com/organization/infinity-augmented-reality
[22] https://www.kaspersky.com/blog/my-precious-data-report-one/14093/
[23] https://www.bbc.com/mundo/noticias-49093124
[24] https://bit.ly/3g0yzSv
[25] https://tcrn.ch/2WUFXHs
[26] https://bit.ly/2OV54FY

Este es un motor que ofrece [44] la posibilidad de crear videojuegos y aplicaciones en 2D y 3D, usando principalmente el lenguaje de programación C# para su desarrollo. Su principal característica es su facilidad de uso, respecto a Unreal Engine (por ejemplo), y su curva de aprendizaje. Esta facilidad ha permitido que se adopte en otras industrias fuera del videojuego; como el cine, la automoción, la arquitectura o la ingeniería. Además, no es asunto baladí, Unity ha ido creciendo[27] hasta haber sido utilizado para crear el 54 % de los mil mejores videojuegos móviles y una media, entre juegos de PC, consola y móvil, del 50 %.

Su modelo de plan de licencias[28] ofrece al usuario varias opciones. Su uso puede ser gratuito, si eres un estudiante en una institución acreditada o si los ingresos son inferiores a 100.000 dólares estadounidenses en los últimos 12 meses. Las restricciones irán incrementando por encima de estos ingresos, haciendo que pase a ser una herramienta de pago. Sin embargo, actualmente, Unity no permite pasar de un plan a otro en un mismo proyecto si, por ejemplo, nuestros ingresos aumentan; haciendo que debamos crear el mismo proyecto en ese nuevo plan partiendo desde 0. Tampoco puede una empresa tener licencias de distinto tipo entre sus empleados.



(a) Videojuego Cities Skylines

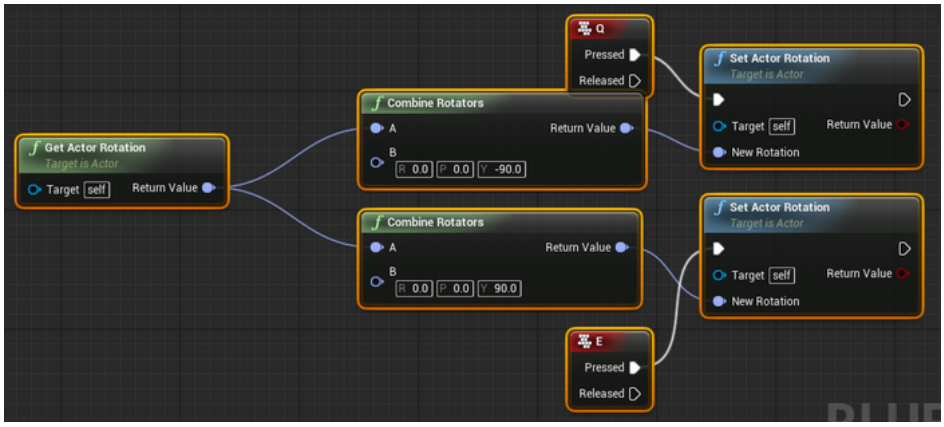
(b) Videojuego Pokémon Go

Figura 2.11: Ejemplos de Unity

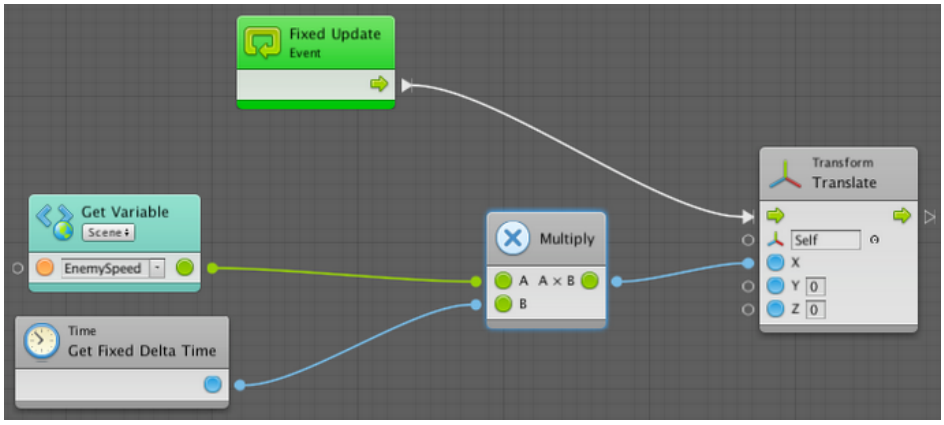
Unreal Engine (UE), al igual que Unity, proporciona al desarrollador funcionalidad básica al proyecto que desee desarrollar. Si este considera necesaria una funcionalidad más avanzada, puede implementar en el proyecto el complemento de ARCore o ARKit que proporciona Unreal Engine en su *Plugin Browser*.

UE [45] también permite crear videojuegos y otras aplicaciones en 2D y 3D. Aunque requiere una curva de aprendizaje mayor, en este momento existen multitud de manuales y tutoriales; haciendo que no sea un factor tan importante. Además, usa el lenguaje de programación C++ para desarrollar sus aplicaciones, pero también proporciona una herramienta de programación visual llamada *Blueprint*. Esta herramienta permite a los usuarios probar nuevas funcionalidades de forma fácil y sencilla. Las personas, y equipos, que no tienen conocimientos de programación también pueden beneficiarse de esta característica para evitar escribir código. UE tiene una amplia variedad de recursos para hacer que la programación no sea algo tan vital como sí lo es en Unity (*Aunque Unity hizo pública la adquisición de Bolt, una herramienta similar a los blueprints de UE, en mayo de 2020*[29]).

[27] `https://unity.com/our-company` *Revisado en julio 2020*
[28] `https://store.unity.com/` *Revisado en julio 2020*
[29] `https://bit.ly/3gv2nWL` *Revisado en agosto 2020*



(a) *Blueprint* de UE



(b) Bolt de Unity

Figura 2.12: Ejemplos de programación visual

En este momento UE también se usa fuera del desarrollo de videojuegos de la misma forma que Unity, llegando a conseguir resultados muy realistas. Ganó en 2014 el record Guinness al motor de videojuegos más exitoso y sigue creciendo su uso y beneficios, siempre más centrado en el PC y las consolas que Unity, que crece más en el sector móvil.

UE usa un modelo de licencias[30] diferente al de Unity. Su uso es gratuito, para videojuegos, hasta tener unos beneficios superiores a los 3.000 dólares estadounidenses; al llegar a esta cantidad el 5 % de los beneficios obtenidos serán compartidos con Epic Games. En cambio, para el desarrollo de aplicaciones que no sean videojuegos es totalmente gratuito y, si es necesario, es posible pagar 1.500 dólares al año por puesto de trabajo para obtener ayuda y cursos personalizados.

(a) Escena de modelo de arquitectura

(b) Uso de RA

Figura 2.13: Ejemplos de Unreal Engine

[30] https://www.unrealengine.com/en-US/get-now/games *Revisado en julio 2020*

CAPÍTULO 3
Análisis del problema

En este apartado identificaremos los requisitos necesarios para llevar a cabo el trabajo, veremos las soluciones posibles y cuál ha sido nuestra solución, junto con la serie de pasos (simplificados) para llegar a ella.

3.1 Especificación

Lo que pretendemos conseguir con esta aplicación móvil es que el usuario pueda utilizarla como complemento a un programa instalado en un ordenador. Entonces, la aplicación, descargará de un servidor; situado en la red local del dispositivo móvil y alojado en el ordenador, los modelos que el usuario desea visualizar en RA. El móvil mostrará estos modelos visuales situándolos en el mundo real. A continuación identificaremos los principales requerimientos necesarios para cumplir con esto, así como los casos de uso y los bocetos (*mock-ups*) de la interfaz de usuario.

3.1.1. Requisitos

Requisitos funcionales

- El usuario inicia y apaga el servidor local del equipo de escritorio cuando lo requiera. Cuando lo inicie visualizará la IP que debe escribir en la aplicación para poder conectarse a él.
- El usuario es capaz de escribir la IP del servidor en la aplicación antes de iniciar la descarga.
- El usuario inicia la descarga de contenido cuando lo solicite desde el dispositivo móvil.
- Cuando se inicie la descarga se descargará primero la escena y se analizará su contenido, un archivo JSON, y se extraerá de él los demás elementos a descargar a continuación.
- El usuario visualiza el modelo a escala real a través de la pantalla del dispositivo móvil y de la cámara, fundiéndolo con el mundo real.

Requisitos no funcionales

- La aplicación debe poder ejecutarse en móviles con poca capacidad de cómputo, aunque deben soportar ARCore o ARKit.
- La interfaz de usuario debe ser intuitiva y la mínima para no entorpecer la visión. Además, mientras se descargan los componentes del servidor la pantalla se bloqueará hasta que finalice.
- La IU de Composer se bloqueará impidiendo al usuario interactuar, una vez inicie el servidor, con ella, con la excepción de la opción de apagarlo.

3.1.2. Modelado conceptual

Casos de uso

Una vez identificados los requisitos pasaremos a capturar los requisitos funcionales mediante un diagrama de casos de uso siguiendo la notación UML.

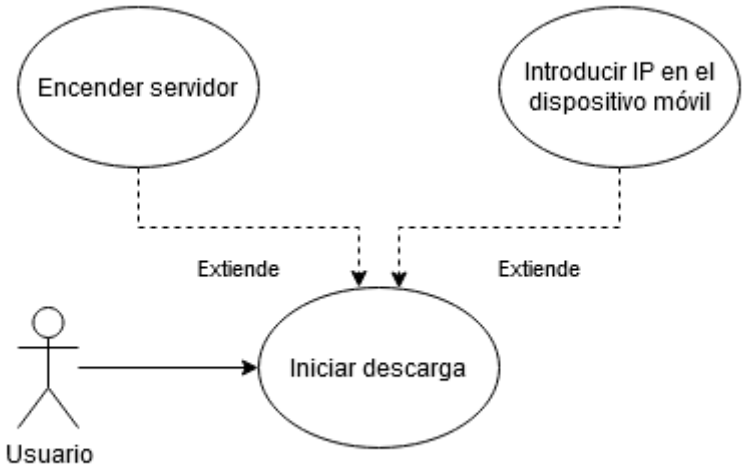


Figura 3.1: Diagrama de casos de uso

Podemos identificar un único actor: el usuario, que tendrá el caso de uso de iniciar la descarga. El resto de operaciones serán transparentes al usuario y manejadas automáticamente por el sistema. A continuación lo especificaremos.

Caso de uso	Descargar escena.
Actor	Usuario
Descripción	El usuario puede solicitar el inicio de la descarga para ser almacenada en el dispositivo móvil.
Pre-condición	- El teléfono móvil no tiene almacenados ya los archivos a descargar. - El servidor del Composer debe estar iniciado antes de comenzar la descarga.
Flujo general	- El usuario inicia el servidor en el Composer. - El usuario usa la interfaz de usuario de la aplicación móvil para introducir la IP que indica el Composer.
Flujo alternativo	El usuario descarga de nuevo los elementos, sin cambiar la escena en el Composer.
Post-condición	El usuario es capaz de ver en la pantalla de su dispositivo, a través de la cámara, la escena que ha descargado en RA junto al mundo real.

Tabla 3.1: Caso de uso Descargar escena

3.1.3. Bocetos o *mock-ups*

A continuación, se mostrarán los bocetos que hemos utilizado para trabajar en las diferentes pantallas de nuestro trabajo.

Aplicación móvil

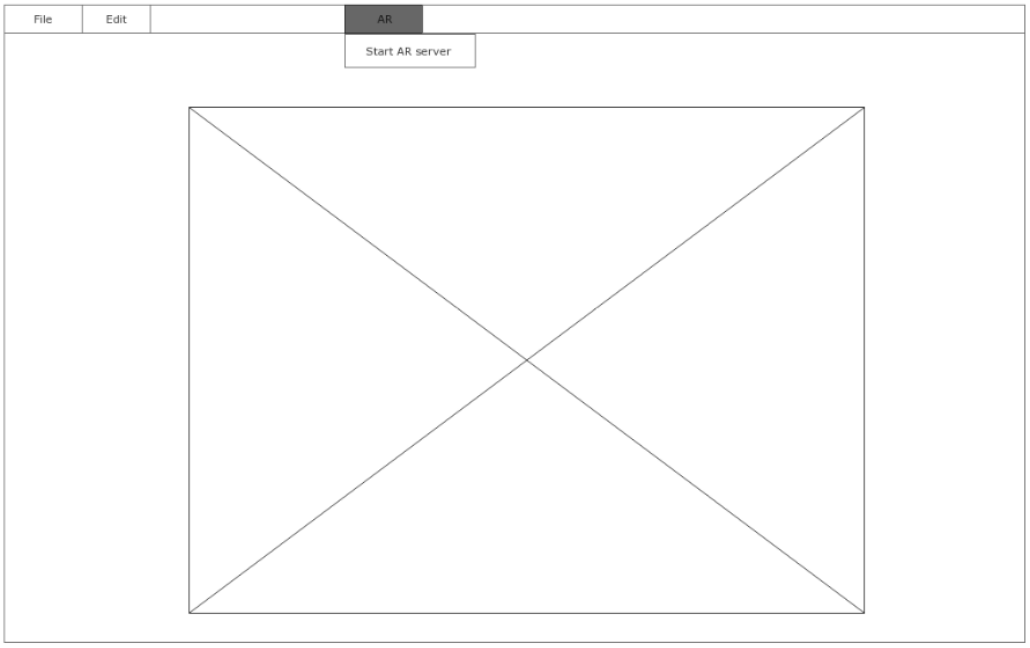
(a) Pantalla de descarga **(b)** Pantalla de espera

Figura 3.2: Bocetos de la descarga

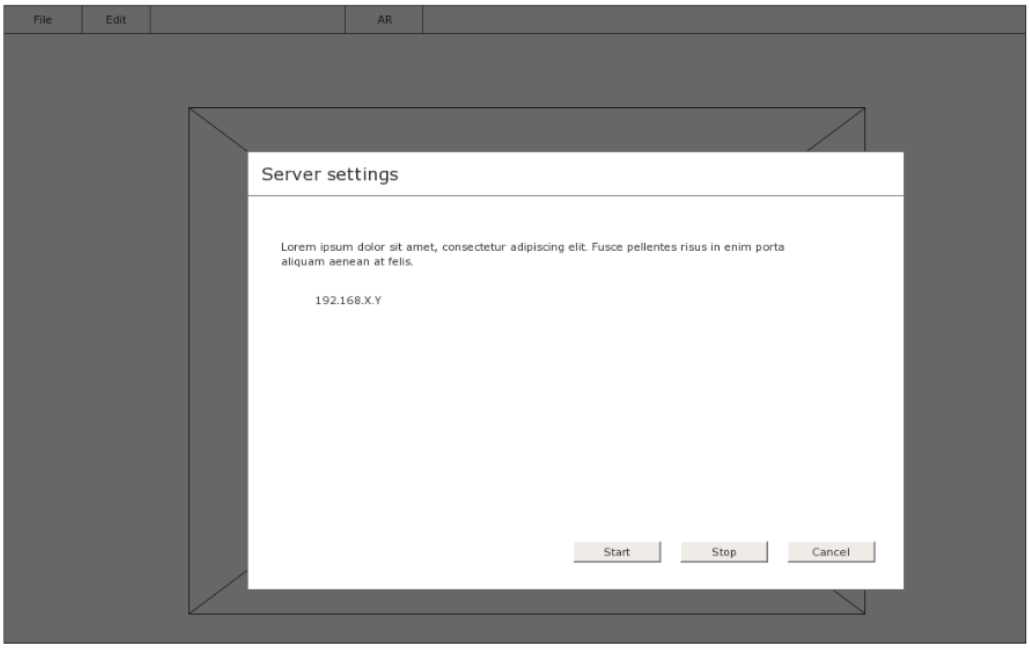
La pantalla mostrada en la figura 3.2.a aparecerá si el usuario abre la aplicación sin que se estuviese ejecutando ya. En esta, se le pedirá, mediante una entrada de texto, la IP del servidor iniciado en el Composer. La pantalla de la figura 3.2.b se dará cuando se haya iniciado la descarga, en ella se le indicará al usuario que se están descargando los elementos que componen la escena.

La pantalla principal, la de visualización en RA a través de la cámara no la mostramos debido a que no hemos añadido nada. Simplemente el usuario pulsará para iniciar la descarga y se mostrará la escena.

Plugin Composer



(a) Nueva opción en barra de herramientas



(b) Ventana de configuración

Figura 3.3: Bocetos del Composer

En la figura 3.3.a añadimos una nueva opción a la barra de herramientas que abrirá un submenú para iniciar nuestro servidor. Una vez clicada en esta opción aparecerá una ventana. Aquí daremos al usuario las direcciones IP que puede usar para introducir en la aplicación. Además, añadiremos las opciones de «*Start*» y «*Stop*», para iniciar y parar el servidor, y «*Cancel*» para cerrar la ventana.

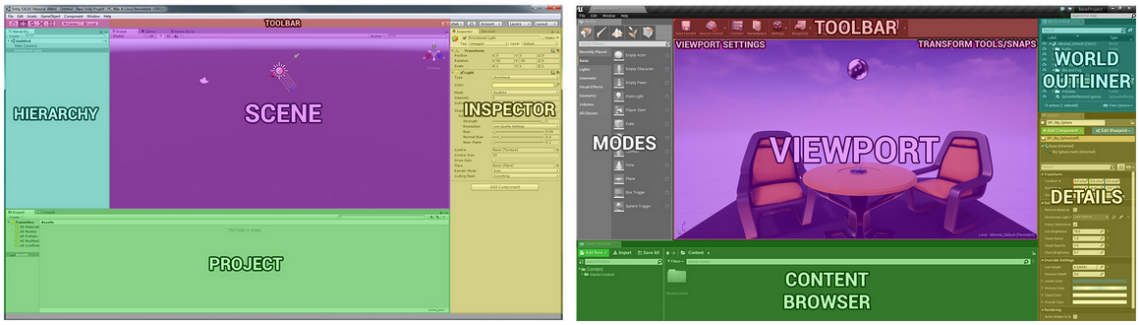
3.2 Identificación y análisis de soluciones posibles

La escala de un trabajo de fin de grado ha hecho que las diferentes opciones para realizarlo sean reducidas, siempre teniendo en cuenta que la solución final debe ser real y usable. Por lo tanto, disponemos de varias posibilidades que hemos estado barajando para los diferentes puntos que componen el trabajo.

3.2.1. Motor para el desarrollo

Hemos hablado anteriormente de las características generales de Unity y Unreal Engine, pero es importante ver qué más nos ofrecen para este proyecto. Lo que buscamos en este punto es un motor que nos proporcione las herramientas necesarias para desarrollar nuestro proyecto. Hemos barajado el uso de estos motores debido a que son los que cuentan con una mayor comunidad y prestaciones para nuestro trabajo. Y esto es lo que hacen, nos ofrecen un entorno; del que ya hemos hablado anteriormente de forma general y hablaremos de nuevo un poco más adelante, para construir una aplicación de RA para dispositivos móviles e implementar en ella un cliente a conectar en un servidor local.

Estos motores tienen, evidentemente, la misma finalidad; el desarrollo de videojuegos y aplicaciones. Debido a ello, las diferencias, en principio, no son muy relevantes. Sin ir más lejos, la ventana de edición (esta, junto el entorno de desarrollo, es la ventana en la que pasaremos más tiempo trabajando) es muy parecida.



(a) Editor de Unity **(b)** Editor de Unreal Engine

Figura 3.4: Editores de Unity y UE

Esta ventana es la principal, aquí es donde crearemos y añadiremos los componentes de nuestra aplicación a las escenas. Una escena (*scene* en inglés) es el archivo donde están contenidos los objetos de la aplicación. Se usan para crear menús y las partes que visualizará el usuario. Una aplicación tendrá, normalmente, más de una escena.

Más allá de la escena, los términos que usa cada uno de los motores suele ser diferente y puede tener uno o varios equivalentes[1]. Por ejemplo, en Unity llamamos *GameObject* a

[1] https://docs.unrealengine.com/en-US/GettingStarted/FromUnity/index.html

un objeto contendedor de *Components*, que le dotarán de la funcionalidad necesaria, que representa personajes, al jugador, el escenario, etc. En cambio, en UE este concepto se divide en varios; *Actor*, *Pawn* y *Character*. El *Actor* se deriva de la clase *Object* y sirve como base para *Pawn* y *Character*, que formarán una cadena; cada uno derivará del anterior. De esta forma *Actor* es un objeto que se puede colocar en la escena que, *a priori*, no tiene ninguna funcionalidad activa dentro de la escena. Si queremos añadirle funcionalidad será posible, siempre se pueden añadir controladores, al igual que en Unity, e interactuar a través del código con él. Aunque, quizá sería mejor añadir en este caso un *Pawn*. Esta será la clase base de todos los *Actors* que pueden ser controlados por el usuario (el jugador) o la IA (o el programa en sí). De esta forma se añade una capa de complejidad al objeto, de la misma forma que ocurre con *Character*, que servirá mejor para ser usado por el jugador ya que implementa un esqueleto y la funcionalidad base para ser animado.

Más específicamente, buscamos sistemas que nos abstraigan de la creación de nuestros propios protocolos de comunicación HTTP. Este es un problema que encontramos en Unity. Aquí se disponía de UnityWebRequest, que era un reemplazo de su anterior objeto WWW. UnityWebRequest consistía en dos capas: una API de bajo nivel (LLAPI) y otra de alto nivel (HLAPI), que envuelve a la primera. Sin embargo[2], desde la versión 2018.4 (LTS) HLAPI iba a dejar de ser implementado en el motor e iba a dejar de recibir mantenimiento pasados dos años. Lo mismo ocurrió con LLAPI desde la versión 2019.4 (LTS). Esto ha hecho que los desarroladores hayan pasado a depender, si lo estiman, de usar directamente *sockets* para poder implementar servicios multijugador o web en sus aplicaciones. También pueden empezar a usar el nuevo sistema[3], aunque aún está desarrollándose y muchas de sus funcionalidades básicas no están cubiertas aún, tampoco hay mucha información al respecto sobre cómo usarlo.

En cambio, UE sí que dispone de una API HTTP mantenida y también otra para usar *sockets*. Además, está disponible tanto para usarse mediante *blueprints* o con código C++. También, el hecho de no estar produciéndose un cambio como en Unity hace que haya más soporte hacia la comunidad y tutoriales o información que contemplan la implementación de esta funcionalidad en los proyectos. Un ejemplo [57] de ello lo tenemos en la editorial Pakt, que en 2019 sacó una nueva edición de su familia de recursos de UE donde contemplaba ayudas y ejemplos para trabajar con la API de HTTP.

También, como hemos dicho previamente ambos disponen de un administrador de paquetes, *Package Manager* y *Plugin Browser*. Además, los dos ofrecen la funcionalidad que hemos llamado *plugin* (que en Unity se denomina "paquete de *assets*", algo así como un paquete de contenido). Esto permite que nuestro proyecto sea fácilmente ampliable, reutilizable y compartible, aspecto interesante hoy en día y que, sin duda, es útil que lo sea para la UPV.

Sin embargo, aunque Unreal Engine tiene una mayor curva de aprendizaje y Unity es más intuitivo, es destacable que el modelo de licencias del primero es más atractivo para una organización grande, como lo es nuestra universidad, que no desarrolle videojuegos sino aplicaciones educativas o de un ámbito diferente. Además, los *blueprints* de UE permiten a usuarios, ajenos al ámbito de la programación, una forma de añadir contenido al proyecto, o a proyectos posteriores, sin necesidad de escribir código alguno. También, como hemos señalado en el capítulo anterior, Unity adquirió recientemente Bolt (previamente a su compra era un *plugin* disponible en la tienda de Unity que debía comprar el usuario si quería hacer uso de él). Sin embargo, esta compra vino con la promesa de una versión nueva (Bolt 2), que cambiaría la arquitectura y el funcionamiento de este *plugin*;

[2] https://bit.ly/3g4BUz7
[3] https://github.com/Unity-Technologies/multiplayer *Revisado en julio 2020*

haciendo que el cambio de una versión a otra sea algo traumático y sin saber aún hasta dónde funcionará el Bolt que está disponible en estos momentos.

Los dos motores pueden usar el entorno de desarrollo Visual Studio, que está preparado para trabajar con ellos; dispone de complementos de C# y C++, o Visual Studio Code. También hay otras alternativas como Rider o cualquier editor de texto, si luego es compilado en el editor.

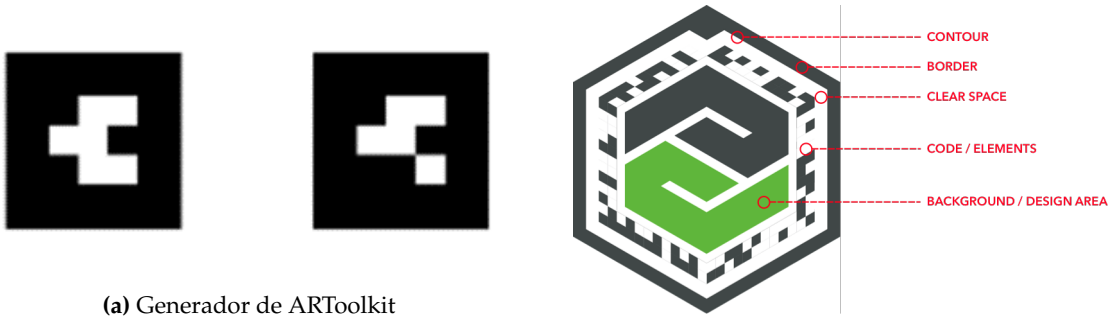
Finalmente, ambos disponen de los SDK de ARCore y ARKit, por lo que el desarrollo de RA para dispositivos móviles no debería ser un problema en ninguno de los dos casos. Además, cuentan con comunidades lo suficientemente activas y numerosos tutoriales en plataformas gratuitas como YouTube, o más profesionales como Pluralsight, como para que el desarrollo en uno suponga una gran diferencia respecto al otro.

3.2.2. Parametrización y reconocimiento del espacio

Existen diversos métodos para implementar la RA [46]. En todos ellos se usa lo que se denomina como *target*, que básicamente es la forma que tiene el *software* de reconocer y, después, añadir la RA al entorno que se está visualizando. Esto puede ir desde imágenes sencillas hasta mapeo y localización simultáneos (técnica SLAM) [50] [51], o geolocalización. Sin embargo, algunos de ellos no los cubriremos debido a que no son lo suficientemente interesantes para nuestra solución, como por ejemplo el reconocimiento de texto.

Markers, imágenes y reconocimiento de figuras u objetos

El *target* más sencillo es el *marker*, o marcador. Este tipo puede ir desde un marcador con un borde ancho, como el *Hiro* visto en la figura 2.13 b, a uno más complejo como códigos QR o códigos generados con patrones comunes. Respecto a estos últimos, ARToolkit proporciona un generador de marcadores y Vuforia incluye un sistema llamado VuMark. La idea básica de estos es el respeto por los bordes, y los elementos comunes, mateniendo así un patrón y una parte modificable.



(a) Generador de ARToolkit

(b) VuMark

Figura 3.5: Generadores de marcadores

El reconocimiento de imágenes evita la creación de marcadores, hecho que puede hacer la aplicación más independiente. Pero, la base de su funcionamiento es la misma; la búsqueda de caracterísitcas como bordes, patrones irregulares y buenos contrastes. Además, este reconocimiento de imágenes se basa en la técnica de rastreo de características naturales (NFT *Natural Feature Tracking*) [47] [48]. Para utilizar este método basta con fotografiar, o incorporar imágenes externas, que funcionarán como marcadores en nuestra aplicación. Primero se analizará y se mapeará, tras ello se guardará y se usará para de-

tectar imágenes, que coincidan con las que ya haya analizado de igual forma que los marcadores.

De igual forma que los dos anteriores se pueden usar figuras sencillas, como conos o cilindros, que tengan patrones y superficies suficientemente detalladas y grandes como para ser reconocibles; podría ser una lata de refresco. La detección de objetos más complejos, como juguetes con el suficiente contraste, funcionaría de la misma manera.

SLAM

A diferencia del punto anterior, esta técnica no precisa de reconocer marcadores, imágenes u objetos ya registrados para su funcionamiento. No necesita que generemos o guardemos nuestros propios objetos para su uso posterior, puede funcionar en un entorno desconocido para el dispositivo. Mediante la creación de anclas o puntos de referencia, en el entorno, podrá calcular distancias y posiciones de nuevos puntos, siempre siguiendo una distribución probabilística sobre estas posiciones.

SLAM [49] [50] se apoya fuertemente en los sensores (básicamente la cámara, el acelerómetro, el giroscópio y el sensor de luz) de nuestro dispositivo, extrayendo de ellos información de nuestro entorno y mapeandolo en tiempo real. ARCore y ARKit implementan este método.

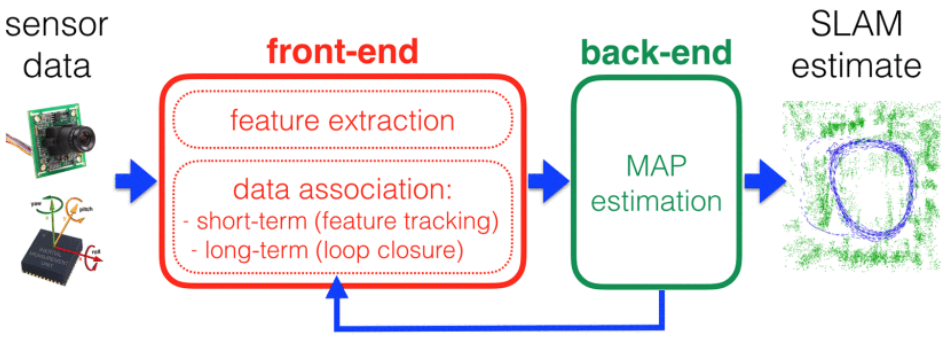


Figura 3.6: Funcionamiento de SLAM

Un sistema SLAM tendrá en su arquitectura dos componentes principales, el frontal (*front end*) y el trasero (*back end*). El frontal extraerá la información que le proporciona los sensores y la abstraerá, y asociará, en puntos o anclas usables para la estimación, mientras que el trasero inferirá los datos, proporcionados por el primer componente, relacionándolos entre las diferentes actualizaciones de los sensores, en el posterior mapeo y proporcionará información de sus resultados al componente frontal para verificarlos.

Esencialmente, aquí SLAM analizará cada *frame* proporcionado por la cámara junto con la información de los sensores para detectar los puntos clave en su escaneo. Después los almacenará junto a la referencia de estos mapeos, lo que se usará para detectar *frames* futuros y redifinir los anteriores.

GPS

Las soluciones de RA basadas en GPS o en ubicación son usadas también mediante los sensores del dispositivo móvil, evidentemente la geolocalización es el sensor predominante. No es tan preciso como las alternativas mencionadas anteriormente, sobre todo en interiores, pero sí que es una opción viable en la industria. Además, no depende de las condiciones meteorológicas o de luz.

La capacidad de crear aplicaciones basadas en esta tecnología no debe ser despreciadas, prueba de ello es el éxito de *Pokemon Go*, comentado previamente.

3.2.3. Sistema de Realidad Aumentada

Cascos y gafas de RA

Cascos como las HoloLens de Microsoft son aparatos que funcionan de forma independiente, no necesitan móviles u ordenadores para mejorar sus capacidades. Disponen de un visor en el que se muestran modelos y hologramas, y lo hace de tal forma que estos cascos llegan a clasificarse como de realidad mixta (RM). Según Milgram et al. [58], esta tecnología combina el mundo real y el virtual para producir nuevos entornos y visualizaciones, donde los objetos físicos y virtuales coexisten e interactuan en tiempo real. Sin embargo, aunque estos dispositivos ofrecen al usuario la libertad de usar ambas manos, la fidelidad de su representación de los objetos de RA en el mundo e independencia de otros dispositivos, suelen ser más caros que los demás. Sin ir más lejos, el modelo más reciente de HoloLens, las HoloLens 2, cuestan 3.500 dólares estadounidenses. Esto hace que, junto a sus capacidades, sean puramente focalizadas al ámbito empresarial.

Los cascos podrían entrar en la categoría de gafas de RA, pero vemos una distinción en la potencia y en la autonomía. De todas formas, vemos que ambas posibilidades adolecen de los mismos problemas esenciales [59]. El contenido virtual de la pantalla y la interacción del usuario con este se convierte en una tarea difícil y fatigosa debido a que la pantalla no es táctil. Adicionalmente, comparándose siempre con los teléfonos móviles, tienen una pantalla de visualización de RA más pequeña que la de estos, peor poder de cómputo y una autonomía menor.

Dispositivos móviles

Android e iOS son las opciones[4] relevantes hoy en día (ver figura 3.7), representando entre los dos ténicamente la totalidad de las ventas.

Se ha ido señalando lo similares que son en su funcionamiento a la hora de tratar la RA. De hecho, la SDK de Google ARCore soporta iOS, mientras que su contraparte; ARKit, no hace lo mismo con Android. Aunque, esto no es tan importante, ya que los propios motores proporcionan estas herramientas indistintamente del sistema operativo en el que se vaya a usar la aplicación.

[4] https://gs.statcounter.com/os-market-share/mobile/worldwide *Revisado en agosto 2020*

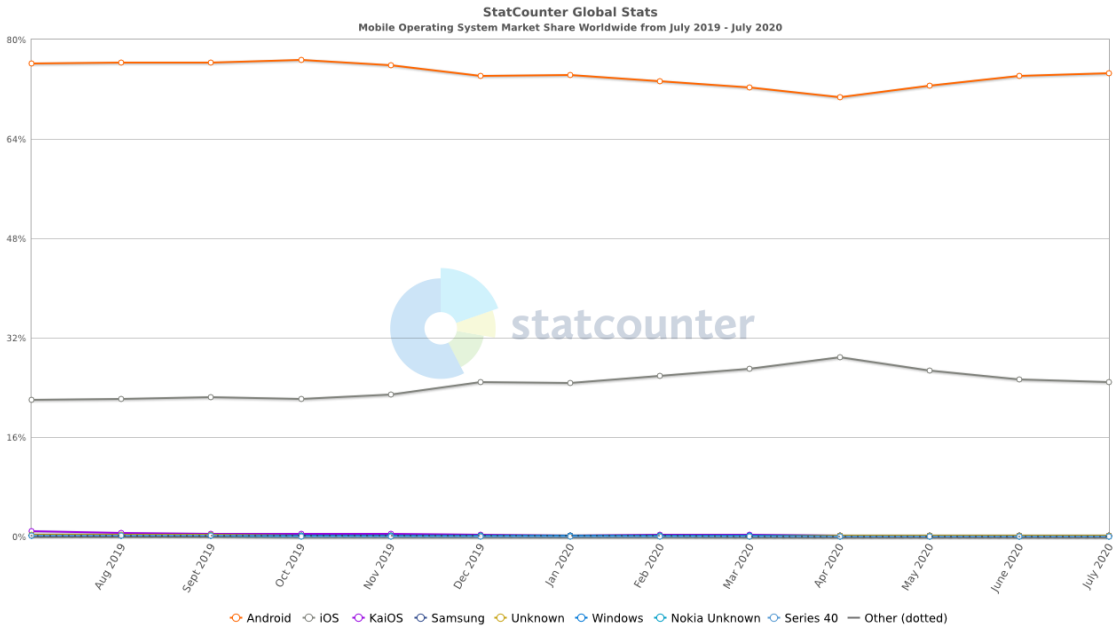


Figura 3.7: Cuota de mercado de los sistemas operativos móviles

Esto nos lleva a tratar la elección del sistema como una opción en la que hay que tener en más en cuenta el público que hará uso de nuestra solución. Además, debido a las herramientas que nos facilitan Unity o UE, nuestra aplicación puede desarrollarse en cada uno de los diferentes sistemas tratados aquí; tanto para gafas o cascos de RA o dispositivos móviles Android o iOS.

3.3 Solución propuesta

Como conclusión de este capítulo mostraremos las distintas elecciones de las herramientas para llevar a cabo el proyecto. Por lo que, nuestra elección es la de desarrollar una aplicación de RA usando SLAM para dispositivos móviles Android mediante Unreal Engine.

3.3.1. Unreal Engine

Hemos optado por usar Unreal Engine sobre todo debido a su sistema de licencias, ya que este es un proyecto con vistas a ser ampliado y, debido a esto, es interesante para la universidad el hecho de no pagar por hacer una aplicación para dispositivos móviles. Recordemos que Unity ofrece un plan de pagos en el que no importa qué se desarrolle ni cuántos desarrolladores formen el equipo que va a llevar a término el proyecto, en Unity se paga por el puesto del desarrollador a no ser que se use la licencia personal (que es gratuita pero no sirve para una institución como es la UPV).

Más allá de esto, hemos señalado anteriormente la necesidad de implementar en nuestra aplicación un cliente que debe comunicarse con un servidor que esté en la red local. Esto hace que la elección de UE sea más atrativa. Aunque Unity disponga de APIs para lograr crear un cliente, estas no tienen una vida útil de más de unos años y su funcionalidad será reemplazada por la de una nueva de la que aún no se sabe lo suficiente. Sin embargo, sí que se podía recurrir a usar *sockets*, pero pensamos que es una dificultad añadida que no es necesaria. Por lo tanto, pensamos que UE es la opción más segura en este ámbito para un desarrollo continuado.

Respecto a la programación visual[5] también creemos que UE es una mejor opción, ya que Bolt va a tener una duración indeterminada; pero con una fecha relativamente próxima en la que pasará a estar obsoleto. Si bien, ambas opciones son complejamente y visualmente parecidas, por lo que decidir entre una u otra, con este criterio, no inclinaría la balanza en favor de cada uno de sus respectivos motores.

Además, UE usa el lenguaje C++ y Unity C#; con sus respectivos *frameworks*. Es un hecho que C++ es un lenguaje de más bajo nivel que C# y que, también, es más flexible y tiene un rendimiento mayor que el otro. En consecuencia, esto hace que objetivamente C++ sea un lenguaje más difícil de leer y de aprender, pero hace que nuestra aplicación sea más ligera; hecho importante para la limitada memoria de la que puede disponer un teléfono móvil, y rápida.

Finalmente, otra de las razones por las que elegir Unreal Engine es el hecho de que Fernando Serrano Capena, nuestro director experimental, desarrolló Bg2UnrealTools (nombrado en el punto 1.2) como un *plugin* de UE. Haciendo que, de usar Unity, tuviese que hacer su contraparte para este motor para que pudiésemos llevar a cabo de la misma forma este trabajo. Por lo tanto, quizá esta sea una de las razones con más peso para su elección.

3.3.2. SLAM

Hemos tomado la decisión de usar la tecnología SLAM, aunque cuenta con puntos débiles. Aquí, la información que aporta cada sensor se actualizará de forma constante con nuevas medidas. Esto hace que cada vez que esto ocurra todos los puntos creados como vínculos con el mundo real serán influenciados, cambiando de posición o cambiando su relación con otros puntos, teniendo un coste computacional alto. Además, cada medida de los sensores puede contener errores, que se podrán ir acumulando con cada actualización hasta hacer que el mapeo se torne inconsistente [52].

Aunque ese problema se ha ido resolviendo alineando los escaneos [52] basándose en una red, o conjunto, de restricciones derivadas de la combinación de pares de escaneos y otros sensores. También con estrategias [51] basadas en algoritmos genéticos, que han ido haciendo con el tiempo que SLAM sea más óptimo y usable, y técnicas [53] que ya aplican SLAM para dispositivos que usan RA y no solo robots; como era su propósito original.

Por lo tanto, esta tecnología sigue progresando [54] pero, la característica de escanear en tiempo real; sin necesidad de almacenamiento previo de los componentes a reconocer es clave para el tipo de aplicaciones que nos concierne. Hace que sea independiente y autónoma, sin necesidad de nada más que el dispositivo móvil, que ya contiene los sensores necesarios y la cámara. Los usuarios no necesitan tener nada más que eso y creemos que precisamente eso es lo más interesante, y lo que hace de SLAM la opción que hemos elegido.

Como prueba de su potencia y viabilidad Apple[6] [7], Google[8] y Microsoft[9] usan esta técnica en sus dispositivos. De una u otra manera la industria tecnológica adapta[10] [11] esta

[5] *Ver ejemplos de la figura 2.12*
[6] `https://developer.apple.com/augmented-reality/arkit/` *Revisado en agosto 2020*
[7] `https://developer.apple.com/videos/play/wwdc2018/610/` *Revisado en agosto 2020*
[8] `https://bit.ly/31Vq8lE` *Revisado en agosto 2020*
[9] `https://bit.ly/3hyzOsX` *Revisado en agosto 2020*
[10] `https://patents.google.com/patent/US20170336511A1/en` *Revisado en agosto 2020*
[11] `https://patents.google.com/patent/US8711206B2/en` *Revisado en agosto 2020*

técnica para dotar a sus dispositivos de las posibilidades que ofrece SLAM. Además, ya hay recursos y conferencias [55] [56] hablando de esta funcionalidad y lo que ofrece.

3.3.3. Dispositivo móvil Android

Habiendo mostrado los porqués de las elecciones anteriores hablaremos de nuestra preferencia por los teléfonos móviles Android.

Como hemos hablado en el punto 3.2.3, los cascos y las gafas ofrecen muchas posibilidades, pero su precio hace que sea virtualmente imposible para el usuario medio adquirir uno de estos sistemas; al menos de momento. De hecho, si tenemos en cuenta esto, nos encontraríamos con prácticamente el mismo problema que tenemos con las gafas de RV que ofrece y dispone, en nuestro caso, la UPV para su uso; un limitado acceso al mismo debido a su precio.

Por lo que nos quedamos con los dispositivos móviles. Aquí nos encontramos con Android e iOS, el primero es software libre y de código abierto, mientras que iOS es un sistema cerrado. *A priori* esto último no supondría un problema, pero para desarrollar aplicaciones para iOS se necesita compilar en una máquina macOS, entre otras cosas. Para lograrlo, si no se dispone de un ordenador Mac, es posible usar uno mediante una conexión SSH desde Windows a uno ya conocido o adquirir un servicio *cloud* que disponga de Macs para su uso.

El hecho de no disponer de uno hace que esto sea una dificultad que creemos innecesaria. Además, a diferencia de iOS, encontramos que existen más usuarios de Android. Aún así, desarrollar esta aplicación para iOS podría ser una posibilidad para futuras ampliaciones, si se dan, para nuestro trabajo; ya que es innegable que, si bien hay más usuarios de Android también los hay de iOS que podrían estar interesados en disponer de nuestra aplicación.

3.4 Fases de desarrollo

Para desarrollar esta solución elaboraremos un servidor local sencillo del que descargar un archivo de ejemplo, para hacer pruebas sencillas. Además, para realizar estas primeras pruebas implementaremos en UE un cliente simplificado que descargue este archivo.

Una vez comprobado el funcionamiento de este ejemplo ampliaremos la funcionalidad en ambas partes. En el Composer haremos una escena de ejemplo y la guardaremos. Aquí nos encontraremos con el archivo con la extensión .vitscnj, que será nuestra escena, y los archivos .vwglb serán nuestros modelos 3D, planos, etc. Los demás archivos serán materiales e imágenes. Lo que queremos lograr en esta fase es descargar nuestra escena y el resto de objetos que la componen. Esto debe seguir siendo manejado por nuestro servidor.

Cuando lo hayamos logrado haremos que estos elementos aparezcan en la pantalla de nuestro dispositivo móvil como parte de nuestra experiencia de RA en un punto fijo o predeterminado por nosotros. Además, trasladaremos nuestro servidor a un *plugin* para el Composer. De esta manera añadiremos la funcionalidad que perseguimos a la aplicación donde se crean las escenas y dejaremos de usar algo externo (nuestro servidor de pruebas).

Como punto final, cuando hayamos conseguido este conjunto de características y funcionalidades, pasaremos a dotar a nuestra aplicación, y a las funcionalidades añadidas al Composer, de una interfaz más accesible y atractiva.

Más allá de esto, pasaríamos a trabajar en los puntos de la sección 6.2 «Trabajos futuros» del capítulo 6. Empezando por dotar al usuario de una interfaz con *feedback*, que le retroalimente con errores, o con la capacidad de colocar la escena virtual en la superficie que desee.

CAPÍTULO 4
Diseño de la solución

En este capítulo presentaremos los subsistemas en los que hemos dividido nuestra solución y una explicación detallada de estos componentes y su función global.

4.1 Arquitectura y diseño del sistema

Para comprender el funcionamiento de nuestro sistema debemos hablar a varios niveles; de qué es una arquitectura cliente-servidor, el funcionamiento del sistema, etcétera.

4.1.1. Sistema cliente-servidor

Pasaremos ahora a explicar cómo funciona nuestro modelo cliente-servidor.

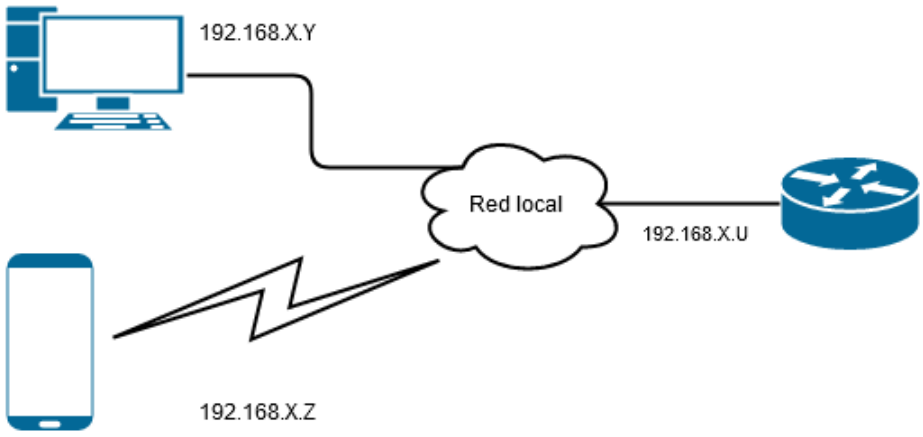


Figura 4.1: Diagrama de red

El sistema se comunica a través de una red local, con una serie de IPs que nosotros como desarrolladores no tenemos porqué conocer. El usuario, que usará un ordenador en la red, finalizará su trabajo en el composer y guardará la escena. Si desea ver lo que ha hecho, antes de acceder a las gafas de RV que le proporciona la universidad, podrá usar nuestra aplicación en su dispositivo móvil; siempre suponiendo que soporta[1] la RA en su teléfono. Para ello, como ya hemos explicado, iniciará el servidor en el Composer y este le dará un IP que deberá introducir en la aplicación del móvil. Como resultado el servidor

[1] https://developers.google.com/ar/discover/supported-devices *Revisado en agosto 2020*

alojado en el ordenador mandará al cliente la escena y los elementos que la componen. Para entender cómo se da esto hablaremos de qué es la arquitectura cliente-servidor.

Un acercamiento temprano a este sistema nació alrededor de la década de 1960, en estas fechas se buscaba optimizar el rendimiento de los ordenadores; máquinas que podían ocupar una habitación entera. Por esta época, los científicos, matemáticos e ingenieros empezaron a construir ARPANET[2], la base de lo que hoy conocemos como Internet, usando los términos de *serving host* y *using host*. Aunque estos no significan exactamente lo mismo que lo que hoy llamamos servidor y cliente, respectivamente, conceptualmente sí lo son. *Serving host* y *using host* hacían referencia a ordenadores en su conjunto y, en cambios, servidor y cliente son programas que funcionan en los ordenadores.

En este modelo[3] existen dos partes, una en las que las tareas se reparten entre los proveedores de servicios o recursos, los servidores, y los demandantes de los cuales, los llamados clientes. Por lo tanto, un cliente realiza peticiones al programa servidor, que le dará la respuesta que está buscando. Aún así esta idea puede aplicarse, por ejemplo, a programas que se ejecutan en una sola máquina, por lo que podríamos tener dos programas diferentes; un cliente y un servidor, que se comunicarán de la misma forma que si estuviesen en ordenadores separados.

Esta arquitectura es de tal importancia y está tan extendida que asume la mayoría de los servicios de Internet. Tanto para visitar un sitio web, donde nuestro navegador sería el cliente y el servidor la máquina, bases de datos y usos que componen el sitio visitado, como para jugar a un juego en línea estamos usando este sistema.

En resumen, para nuestro proyecto usamos esta arquitectura en una red local, esto es conectándonos en; por ejemplo, el wifi de nuestro propio hogar o de la universidad. Por tanto, el programa Composer situado en una máquina con IP 192.168.X.Y alojará nuestro servidor y el cliente, con una IP 192.168.X.Z, será nuestra aplicación situada en el dispositivo móvil.

No es baladí la estructura que siguen nuestras IPs, como podemos comprobar todas las direcciones comparten los tres primeros octetos; el último de ellos es «X» debido a que nosotros no lo conocemos (puede cambiar dependiendo de la empresa con la que tengamos contratado Internet), esto quiere decir que comparten la misma red. Como vemos el que cambia será el último, este en nuestro caso identificará a cada una de las máquinas para que puedan comunicarse entre sí. Aunque, dependiendo del usuario y la red donde se situa, la estructura de cada dirección puede cambiar y no coincidir con lo que mostramos.

[2] https://www.rfc-editor.org/rfc/pdfrfc/rfc5.txt.pdf
[3] https://es.wikipedia.org/wiki/Cliente-servidor

4.1.2. Funcionamiento del sistema

Detallaremos ahora el funcionamiento de nuestro sistema.

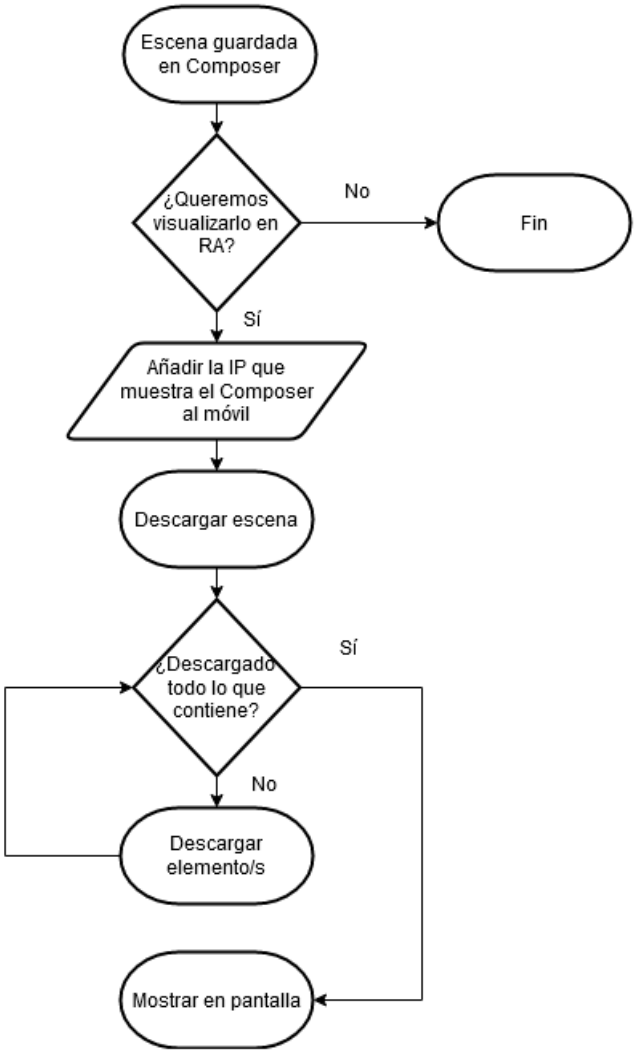


Figura 4.2: Diagrama de flujo del sistema

Ya hemos hablado anteriormente sobre el funcionamiento y la estructura de esto: nuestro trabajo se compone de dos grandes bloques, la aplicación para dispositivos móviles y Composer (el *plugin* que hemos desarrollado para él).

Evidentemente, en ambos bloques nos servimos de trabajos pasados, no partimos de 0. Composer fue desarrollado por Fernando y lo que hacemos en él es añadirle un complemento, construido sobre su base. Lo mismo ocurre con la aplicación para móviles, está hecha en Unreal; con todas las herramientas que proporciona como motor, y utiliza el *plugin* Bg2UnrealTools.

Bloque dispositivo móvil

En la parte de la interfaz hemos optado por un diseño sencillo, creemos que para esta versión es algo de menor importancia mientras funcione y sea usable. Por ello, además, para facilitarnos su implementación nos hemos servido de un blueprint de tipo *widget*.

(a) Jerarquía del menú **(b)** Interfaz del diseñador

Figura 4.3: Construcción de la interfaz

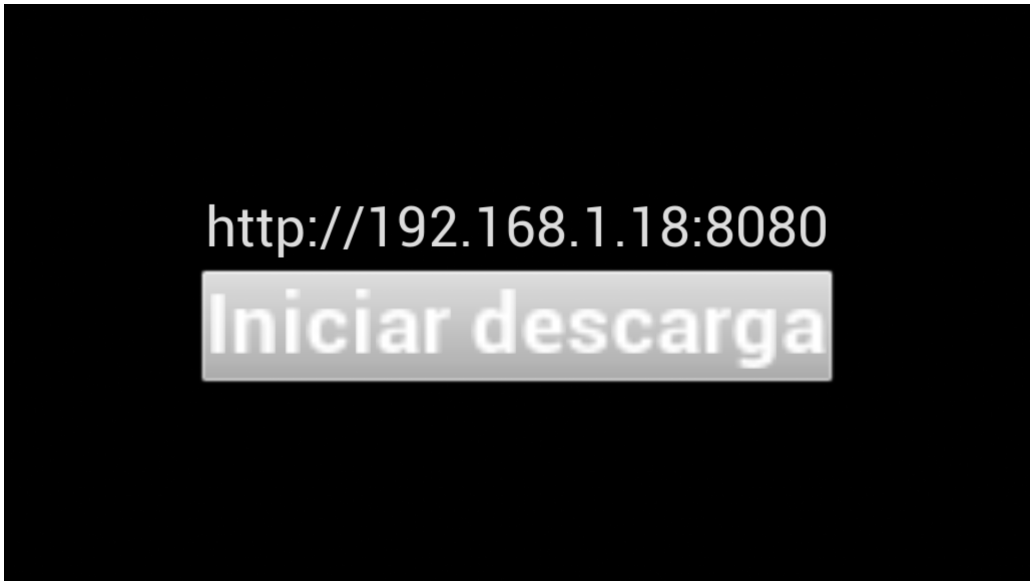


Figura 4.4: Interfaz de la aplicación móvil

La figura 4.3.a nos muestra la jerarquía del diseñador del menú. Se puede apreciar que los elementos del menú comparten la nomenclatura de otros sistemas. Además, el diseñador de interfaces (figura 4.3.b) que nos proporciona UE es sencillo. Se puede construir mediante *drag and drop* con elementos básicos. No hacía falta, en esta versión inicial, una interfaz más compleja.

Podemos ver en la figura 4.4 el resultado de esto. Aquí, nos encontramos con una IP prefijada que el usuario puede editar y un botón. Al clicar aquí se emitirá un evento que capturamos mediante blueprints, como se muestra en la figura 4.5.

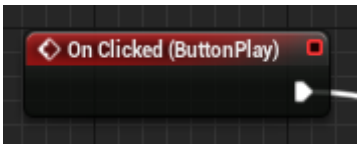


Figura 4.5: Evento *OnClicked*

Como hemos visto, en la figura 4.3.a, llamamos a nuestro botón «ButtonPlay» y este nombre coincide con el que aparece en la figura 4.5. Lógicamente, esto indica el manejo del evento que emite al clicar en este botón por parte de ese nodo.

La funcionalidad importante que sigue a este punto será la de la figura 4.6.

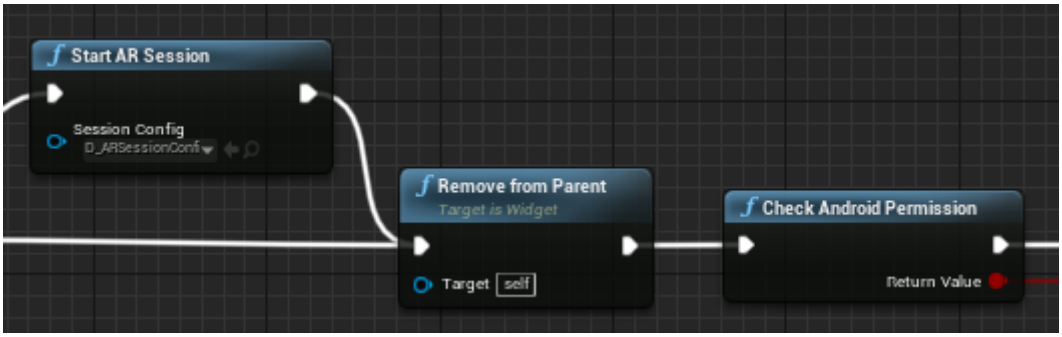


Figura 4.6: Sesión de RA y permisos de Android

Aquí comenzamos una sesión de RA, un nodo que incorpora UE. Tras esto, quitamos de la pantalla el menú con el nodo de *Remove from Parent* y comprobamos los permisos de escritura de Android (figura 4.7) con una función personalizada.

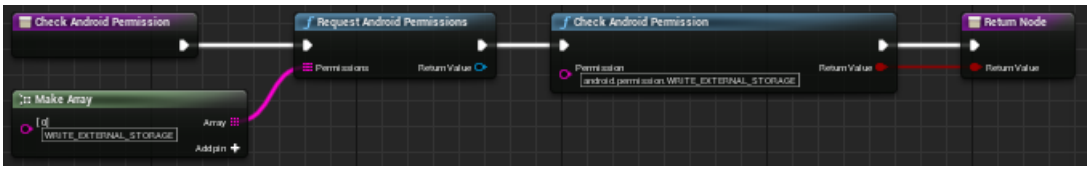


Figura 4.7: Adquisición de permisos en Android

Hecho esto, como se ve en la siguiente imagen, llamaremos a nuestra función en C++ *Download* para comenzar la descarga. En el campo «URL» de esta funcion estamos usando el contenido del campo de texto de *Address text input*, visto en la figura 4.3.a. Esta funcionalidad la veremos más adelante.

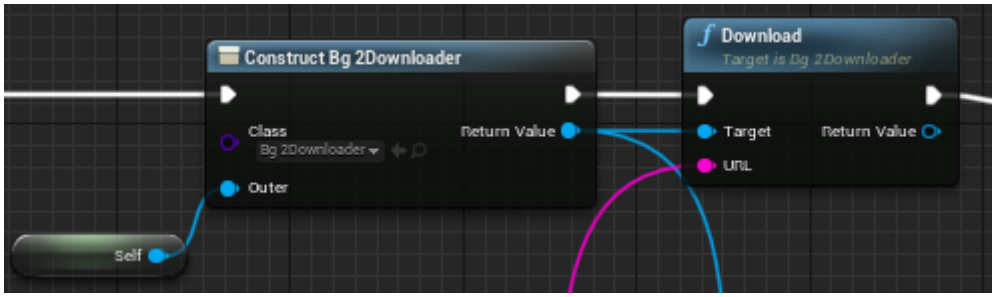


Figura 4.8: Inicio de la descarga

Los puntos que vienen tras la descarga será la espera de su finalización mediante un evento. Cuando este evento se dispare, indicando que ha terminado, cargaremos la escena en pantalla. El nodo *Load Scene* es una función creada en el *plugin* de Bg2UnrealTools de Fernando. De esta forma, el nodo cargará la escena en el modelo «BP_Spawn_Scene»; que hemos creado para esto, y lo sustituirá. El campo «Target» será este modelo a reemplazar y el «Path» vendrá del código de nuestro *plugin*, siendo la ruta en la que está situada la escena que queremos llevar a la pantalla.

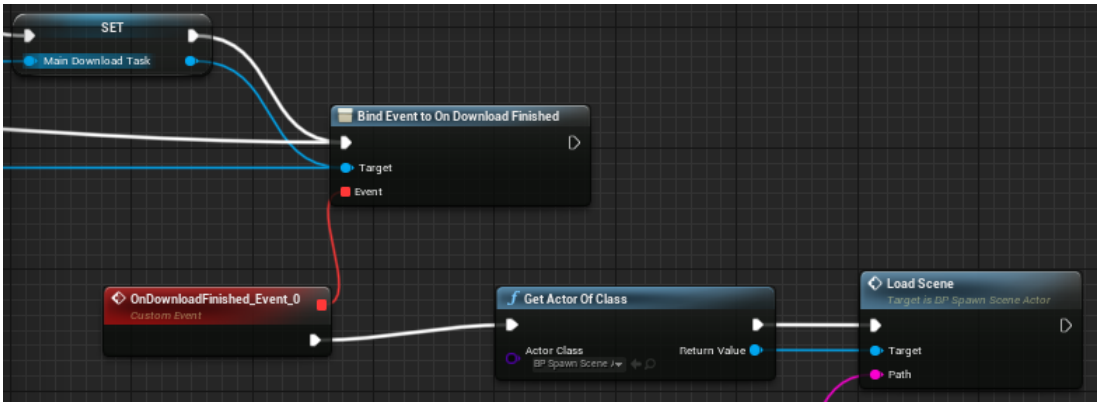


Figura 4.9: Cargar escena

Echemos ahora un vistazo a la funcionalidad escrita en C++ de nuestro *plugin*. La figura 4.10 muestra la funcionalidad del nodo visto en la figura 4.8.

```
DECLARE_DYNAMIC_MULTICAST_DELEGATE(FDownloadFinishedDelegate);
```

(a) Declaración evento

```
UPROPERTY(BlueprintAssignable, Category = "Bg2Downloader")
FDownloadFinishedDelegate OnDownloadFinished;
```

(b) Variable «OnDownloadFinished»

```
UBg2Downloader* UBg2Downloader::Download(FString URL) {
    Start(URL, [&]() {
        // Se han descargado todos los recursos, lanzamos un evento
        //para informar de que se ha terminado la descarga
        this->OnDownloadFinished.Broadcast();
    });

    return this;
}
```

(c) Función «Download»

Figura 4.10: Función «Download»

Para programar en Unreal Engine se usa C++. Lógicamente, UE proporciona su propio *framework* y API para desarrollar código de una forma más sencilla (los blueprints también hacen uso de esto). Esto quiere decir que aunque ofrece una capa de modificación por encima de C++, UE sigue compartiendo sus usos. Como ejemplo de esto último, la programación de clases aquí se hace en dos ficheros diferentes: los ficheros cabecera (*header* de extensión .h) y los ficheros de implementación (.cpp). Los primeros sirven para declarar las clases, funciones y variables que usaremos en nuestro programa. Mientras que los .cpp son la implementación de esta funcionalidad que hemos definido en el archivo anterior. De esta forma una clase dispondrá de un fichero .h y uno .cpp y, a la hora de acceder a las funciones de otras clases deberemos incluir su cabecera en nuestro fichero .h o en el .cpp.

Con ello, la figura 4.8.a y la 4.8.b se situarán en la cabecera, el fichero .h, de la clase «Bg2Downloader» de nuestro *plugin*, y la 4.8.c será la implementación de esta definición. En la figura 4.8.b, además, podemos ver que estamos haciendo accesible a los blueprints. Esto será lo que posteriormente usaremos para manejar el evento. Como vemos, esta función también llama al método «Start» de la figura 4.11.

```
void UBg2Downloader::Start(FString URL, std::function<void ()> onComplete) {
    mOnComplete = onComplete;

    SetActualURL(URL);

    const FRegexPattern containsFilePattern(TEXT(".*[^/]/{1}([a-zA-Z0-9\\-_\\%]+\\.[a-zA-Z0-9\\-_]+)$"));
    FString ActualUrl = GetActualURL();
    FRegexMatcher fileMatcher(containsFilePattern, ActualUrl);

    if (!fileMatcher.FindNext()) {
        SetBaseURL(URL);
        URL += mSceneName;
        SetActualURL(URL);
    }

    // Create the IHttpRequest object from FHttpModule singleton interface.
    TSharedRef<IHttpRequest> request = FHttpModule::Get().CreateRequest();
    request->OnProcessRequestComplete().BindUObject(this, &UBg2Downloader::HandleRequest);
    request->SetURL(URL);
    request->SetVerb(TEXT("GET"));

    //  Start Processing the request.
    request->ProcessRequest();

    //  Prevent the object to be deleted during garbage collection.
    AddToRoot();
}
```

Figura 4.11: Función «Start»

En esta imagen vemos una expresión regular que filtrará la dirección URL que se está usando en esta descarga. Si esta dirección no hace referencia a algo concreto, sino que es una dirección sin más; como podría ser `https://www.google.com/`, entonces esto quiere decir que lo que deseamos descargar es la escena. De forma que añadiremos a esta dirección el nombre de la escena, que hemos definido como «/ExampleScene.vitscnj».

Una vez establecida la URL que queremos, usaremos las herramientas que nos proporciona UE para crear una *request* y proceder a la descarga. Para aclarar este punto, la comunicación entre cliente y servidor http se basa fundamentalmente en *requests* y *responses*. Las primeras serán la forma que tiene el cliente de notificar al servidor que quiere hacer una operación (las más importantes son GET y POST), en este caso desea un recurso; por lo que usa el método GET junto a la URL, que contiene el nombre de este recurso a solicitar. En cambio, las *responses* son las respuestas del servidor a estas solicitudes del cliente.

En el método «HandleRequest», figura 4.12.a, pasamos a comprobar si hemos hecho la *request* correctamente y hemos recibido una *response* válida. Hecho esto pasaremos a crear el directorio dentro de nuestro dispositivo y, más adelante, el archivo con su contenido.

```
if (bSuccess && Response.IsValid() && Response->GetContentLength() > 0) {

    IPlatformFile& PlatformFile = FPlatformFileManager::Get().GetPlatformFile();
    IFileManager* FileManager = &IFileManager::Get();

    //  Create save directory path
    FString savePath = FPaths::ProjectSavedDir();

    FString fileSavePath = savePath;
    fileSavePath += FGenericPlatformHttp::UrlDecode(FPaths::GetCleanFilename(mURL));



    if (!PlatformFile.DirectoryExists(*savePath) || !FileManager->DirectoryExists(*savePath)) {
        //  Create directory
        PlatformFile.CreateDirectoryTree(*savePath);
    }

    //  Create the file
    IFileHandle* fileHandler = PlatformFile.OpenWrite(*fileSavePath);
    if (fileHandler) {
        //  Write the new file from the response
        fileHandler->Write(Response->GetContent().GetData(), Response->GetContentLength());
        fileHandler->Flush(true);
        //  Close and finish rhe operation
        delete fileHandler;
    }

    DoLoadResources(fileSavePath, mResources);
}
```

(a) «HandleRequest»

```
//  Obtain the scene's objects to download.
if (mURL.Contains(mSceneName)) {
    // End of main scene file download
    ScenePath = Path;
    UBg2DownloadParser::SceneParser(Path, Result);
}
else {
    // End of resource file download
    mOnComplete();
    return true;
}

// Main scene download: check how many resource files
mNumResources = Result.Num();
mDownloadedResources = 0;
for (int32 i = 0; i < Result.Num(); ++i)
{
    FString URL = GetBaseURL() + "/" + FGenericPlatformHttp::UrlEncode(*Result[i]);

    UBg2Downloader* DownloadTask = NewObject<UBg2Downloader>();
    DownloadTask->Start(URL, [&]() {
        mDownloadedResources++;
        if (mDownloadedResources == mNumResources) {
            mOnComplete();
        }
    });
}
return true;
```

(b) «DoLoadResources»

Figura 4.12: Funciones «HandleRequest» y «DoLoadResources»

Completados estos pasos llamaremos a la función «DoLoadResources» (figura 4.12.b). Aquí analizaremos el contenido de la escena, con un método del *plugin* Bg2UnrealTools de Fernando, y extraeremos de aquí los recursos a descargar. Los concatenaremos a nuevas URL y repetiremos para cada uno la misma operación de «Start». Además, aquí podemos ver cómo comprobamos los recursos que hay y cuando paramos. Cuando hayamos completado todas las descargas llamaremos a «mOnComplete()», que avisará de este fin y emitirá el evento a los blueprints de la figura 4.9.

Este es el funcionamiento esencial de la descarga de los recursos.

Bloque *plugin* del Composer

Comenzamos a trabajar en esta parte desde el inicio de las pruebas creando un pequeño servidor local, en el que se gestionaba la descarga de un archivo .txt. Más tarde, como vemos en las figuras 4.13 y 4.14, pasamos a construir uno más complejo.

```
console.log(`Server running and listening at:\n`);
console.log("Interfaces:\n");
for (let key in networkInterfaces) {
    console.log(`  ${key}:`);
    networkInterfaces[key].forEach((interface) => {
        console.log(`      ${interface.address}:${port}`);
    });
}

const server = http.createServer((req, res) => {
    var q = url.parse(req.url, true);
    var filepath = basePath + q.pathname;
    filepath = path.normalize(filepath);

    if (!filepath.includes(".")) {
        filename = defaultSceneName;
        filepath += filename;
    }
    else {
        filename = path.basename(filepath);
    }

    doReadFile(filename, filepath, res);

}).listen(port);
```

Figura 4.13: Servidor de pruebas (parte 1)

Como vemos en el bucle, al iniciar el programa, se imprimirán en la consola las interfaces de red de las que dispone nuestra máquina. Esta es una idea que luego llevaremos al plugin del Composer para informar al usuario en la ventana, eliminando las direcciones IPv6.

Una vez analizado si el usuario desea descargar la escena o un recurso normal, se pasará a leer el fichero de la escena y se añadirán para ser descargados los recursos que se encuentren en el directorio en el que se ha guardado la escena creada en Composer.

En esta parte debemos tener en cuenta la codificación del archivo escena para ser usado correctamente en los siguientes puntos. Además, buscamos en el directorio de la escena nuevos recursos y, si los hay, los añade a ella; ya sabemos que la escena es un archivo JSON. De esta forma la prepararemos para que, en la aplicación, sea analizado el archivo y se puedan descargar los demás recursos.

```
function doReadFile(filename, filepath, res) {
    filename = decodeURIComponent(filename);
    filepath = decodeURIComponent(filepath);
    let encoding = filename == defaultSceneName ? "utf8" : "";
    fs.readFile(filepath, encoding, function (err, data) {
        if (err) {
            console.log(`Error, file not found. filename: ${filename} \n`);

            res.writeHead(404, { 'Content-Type': 'text/plain' });
            return res.end("404 Not Found");
        }
        else {
            console.log(`Requested filename: ${filename} \n`);
            res.statusCode = 200;
            res.setHeader('Content-disposition', 'attachment; filename=' + filename);
            res.setHeader('Content-Type', 'text/plain');

            if (filename == defaultSceneName) {
                let sceneData = JSON.parse(data);
                sceneData.resources = [];
                fs.readdirSync(basePath).forEach((resource) => {
                    if (resource != defaultSceneName) {
                        console.log(`Adding external resource: ${resource}`);
                        sceneData.resources.push(resource);
                    }
                });

                data = JSON.stringify(sceneData, "", "\t");
            }
            console.log("------------------ Done ------------------ ");
            console.log("------------------------------------------ \n");
        }
        doEnd(data, res);
    });
}
```

Figura 4.14: Servidor de pruebas (parte 2)

Esta versión simplificada seguía teniendo errores, por ejemplo nos percatamos de que si guardábamos diferentes escenas en el directorio que usábamos para hacer pruebas, se añadían todos los recursos que las componían al fichero de la escena actual para ser posteriormente descargados. Son evidentes los efectos negativos que acarreaba este fallo.

En la versión final a utilizar en la aplicación Composer, nuestro complemento al completo, nos encontramos con más ficheros, como vemos en la figura 4.15. Tanto para definir la interfaz de la ventana que hemos creado en el plugin, como para añadir una opción en la barra de herramientas o suplir de la funcionalidad al *plugin* junto al servidor local.

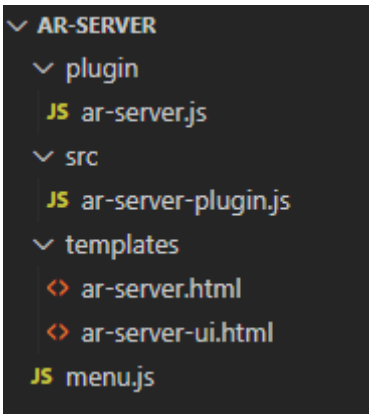


Figura 4.15: Jerarquía de los ficheros del complemento del Composer

El funcionamiento es el mismo, pero con algunas correciones y, lógicamente, añadiendo la funcionalidad que hemos indicado en el anterior párrafo. En la figura 4.16 mostramos el ejemplo de la correción del fallo que detectamos anteriormente.

```
// Borrar archivos del directorio temporal
fs.readdirSync(this.tempBasePath).forEach((resource) => {
    fs.unlink(path.join(this.tempBasePath,resource));
});

// Guardar la escena en directorio temporal
app.render.Scene.Get().selectionManager.removeGizmos();
bg.base.Writer.Write(this.tempScenePath, app.render.Scene.Get().sceneRoot)
```

Figura 4.16: Borrado de archivos temporales en el directorio y guardado de los nuevos

CAPÍTULO 5
Resultados

Estamos ante un trabajo que tiene como objetivo ser una prueba de concepto. Se ha buscado estudiar y comprobar la validez del desarrollo de una aplicación real de RA, para ser usada en la universidad como una herramienta complementaria a su motor de Realidad Virtual y que se pueda usar a discreción del usuario.

Las pruebas que hemos realizado se han llevado a cabo con un móvil LG Nexus 5X. Hay que tener en cuenta, como hemos dicho previamente, que no todos los móviles Android soportan ARCore. Teniendo esto en mente procedamos a ver los resultados obtenidos.

(a) Esfera

(b) Esfera

Figura 5.1: Capturas de pantalla

Los modelos 3D suelen estar representados por mallas poligonales[1]. Por ejemplo, una esfera como la de la figura 5.1.a, tendrá más polígonos que un cubo; esto se traduce en que su representación será más costosa computacionalmente. Lo mismo ocurre con la silla de la figura 5.1.b, ya que es una figura más compleja que una esfera. Podemos ver la diferencia en su representación, y lo que significa tener más polígonos o menos, en la figura 5.2. Hablamos de una limitación que tienen las aplicaciones de este tipo, tanto en teléfonos móviles como en los ordenadores más potentes.

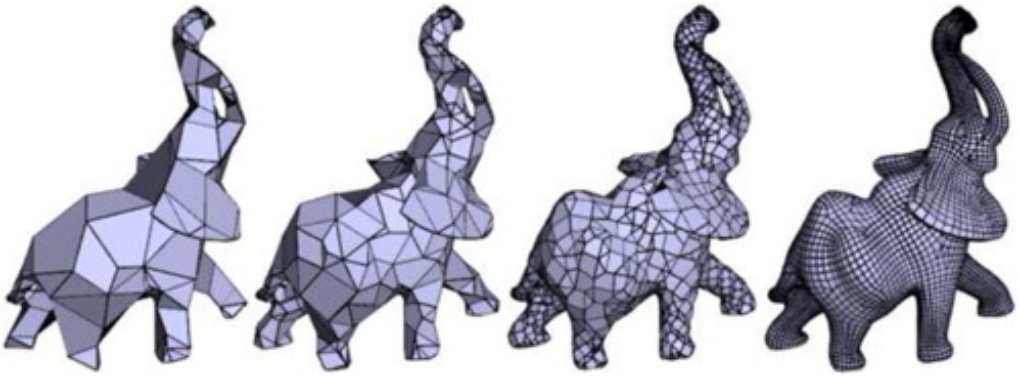


Figura 5.2: Mallas poligonales

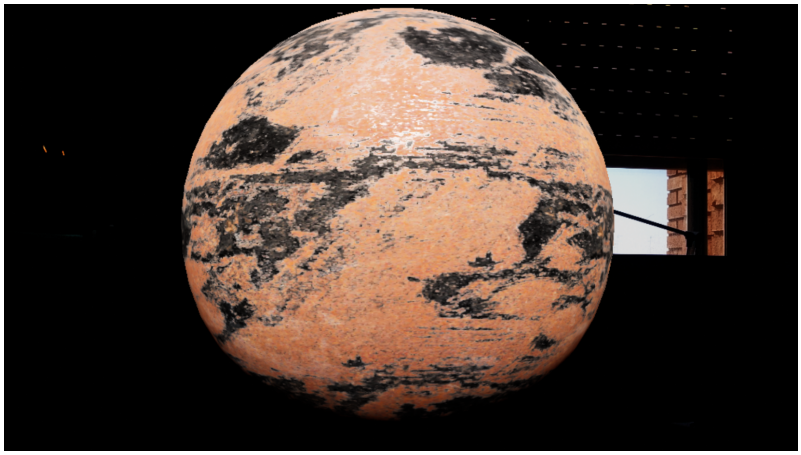
Figura 5.3: Busto

Sin embargo, también tenemos que tener en cuenta que, para el uso que se requiere, el dispositivo más óptimo para llevar a cabo esta tarea, como hemos dicho previamente, es un teléfono móvil. Es ligero, más barato que el resto de opciones y está a disposición de todo usuario. Además, aunque no lo cubrimos en este trabajo, los avances técnicos en los dipositivos móviles se incrementan año a año; lo que ofrece al usuario móviles más baratos y potentes que sus predecesores.

Por tanto, nuestro móvil representaba el busto en tiempo real, aunque se notaba la diferencia de rendimiento a la hora de cargarlo, una vez hecho funcionaba correctamente. Lo mismo ocurriría con, por ejemplo, el primer elefante (en nuestro caso la esfera de la figura 5.1.a) de la figura 5.2 y el último (el busto de la figura 5.3).

[1] https://es.wikipedia.org/wiki/Malla_poligonal

Hemos hablado de la técnica SLAM y su inclusión en este trabajo. El resultado al haber usado esto hace que dependamos de la calidad de imagen de la cámara, la luz y los sensores, sobre todo para captar puntos de referencia. En nuestra aplicación puede llegar a ser un problema trabajar en entornos a contraluz o mal iluminados, donde haya poco contraste.



(a) Esfera

(b) Silla

Figura 5.4: Capturas de pantalla

Como vemos en las figuras 5.4.a y 5.4.b la aplicación adolece de los mismos problemas que puede tener una cámara de un teléfono móvil cualquiera. El visor de la cámara, *per se*, capta el mundo real oscurecido respecto a como lo vemos nosotros. No suele ser más que un problema fotográfico, pero en nuestro caso afecta a la funcionalidad.

El uso de la cámara, con fotos o vídeos normales que podamos tomar desde nuestros móviles, y el uso que le damos aquí no tiene los mismos resultados; el mundo real se capta de forma diferente al virtual. Esto da lugar a no sólo este curioso efecto visual, que resalta la disparidad entre ambos mundos, sino a errores y desplazamientos extraños a la hora de emplazar correctamente nuestra escena en el entorno.

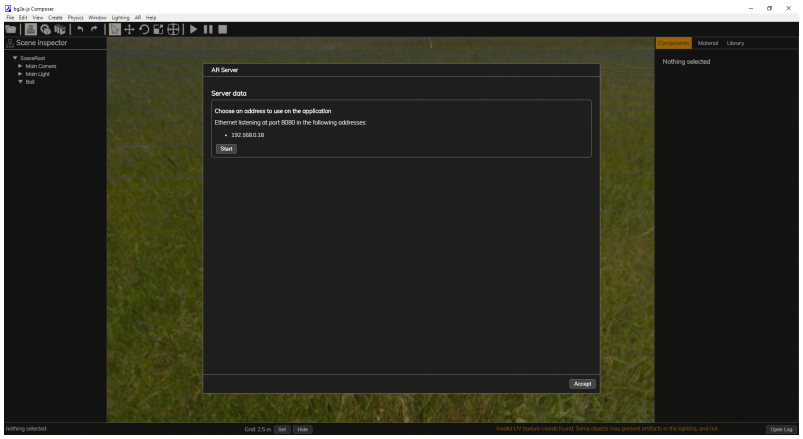
Sin embargo, esto también ocurriría al utilizar *markers*, las condiciones de luz afectan a estos tipos de RA. Además, esta aplicación está pensada para usarse en entornos cerrados: hogares, oficinas o la propia universidad. Por lo que, la luz realmente no es un problema que vaya a afectarnos.

Aún así esto no supone un impedimento para lograr la representación que se busca. En este trabajo se busca representar los modelos que se introducen en el Composer a escala real para corregir errores. Cosa que, de por sí, se consigue sin ningún tipo de problema y que nosotros mismos al realizar las capturas que se ven en este capítulo hemos hecho.

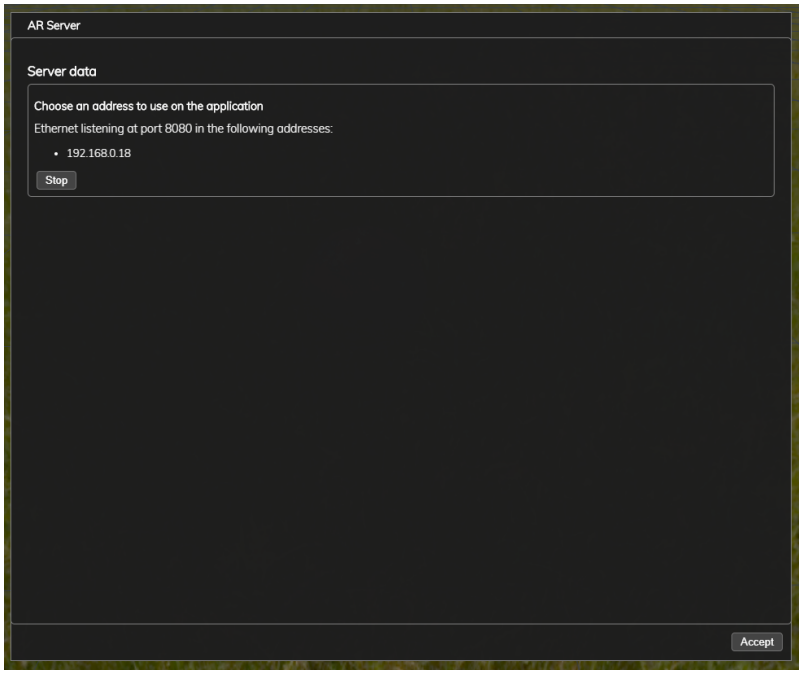
Así mismo, comprobamos que el *plugin* del Composer funciona correctamente. El proceso es sencillo, como vemos a la hora de iniciar el servidor (figuras 5.5.a, 5.5.b, 5.5.c) para representar la esfera de las figuras 5.1.a y 5.4.a. Aunque, en la parte de la aplicación móvil, no existe un *feedback* que informe al usuario del estado de la aplicación. Por lo que, en caso de usar una escena más compleja; como el busto de la figura 5.3, la aplicación parecerá que está bloqueada aunque realmente esté descargando los recursos de la escena. Cosa que no se aprecia con la esfera de la figura 5.1.a, por ejemplo.

Por tanto, pensamos que los resultados son buenos y son los esperados. Como hemos dicho, esta es una prueba de concepto que se debería refinar hasta llegar a una aplicación real y, en el caso de continuar con su desarrollo, esta aplicación ofrece una base robusta, fiable y fácilmente escalable para ampliar su funcionalidad.

(a) Composer



(b) Composer



(c) Composer

Figura 5.5: Capturas de pantalla

CAPÍTULO 6
Conclusiones

El origen de este proyecto fue el de explorar la opción de suministrar a la UPV de una herramienta básica de RA, que sirviese de complemento para los alumnos y profesores que hacían uso del sistema de RV. Prueba de ello son los objetivos, requisitos funcionales y no funcionales que destacamos y que hemos cumplido. Sin embargo, siempre se tuvo en mente el alcance que podía tener un TFG y se intentó adaptar estas necesidades tanto a las limitaciones impuestas por los conocimientos adquiridos en el grado, de lo que hablaremos en el siguiente apartado, y el tiempo del que disponíamos para realizarlo. También se ha realizado con la perspectiva de ser una prueba de concepto y de dotar nueva funcionalidad más adelante.

Ha habido dificultades durante el desarrollo del trabajo. Entre ellas nos hemos encontrado con la dificultad de aprender a trabajar con Unreal Engine y a usar C++. Por supuesto, UE nos ofrece un *framework* de ayuda de C++ que facilita trabajar con él, pero esto mismo también es una dificultad. Nos hemos apoyado en la documentación oficial de UE, en tutoriales y libros de aprendizaje; mucho tiempo de nuestro trabajo ha sido de construir una base de conocimientos para trabajar con UE y entender el funcionamiento de una aplicación de RA.

Más allá de la dificultad impuesta por no tener formación específica al respecto, ha sido un problema depurar una aplicación hecha en UE para Android. Sí que se ha depurado evidentemente para Windows, el sistema operativo en el que se ha desarrollado el proyecto, pero no es lo mismo hacerlo para Android. Aquí nos encontramos con la necesidad de utilizar Android Studio para encontrar errores y ser informados de ello, lo que requería tiempo de compilación y depuramiento, con funciones específicas para este sistema operativo que antes no habíamos contemplado para lograr su visualización. Ejemplo de esto último es que Android Studio no capta las variables de tipo FString propias de UE.

Como resultado, hemos adquirido un rico conocimiento del contexto actual de la RA y su funcionamiento. Sabemos que la industria encargada de desarrollar e investigar sobre ello está muy activa, y surgen nuevas técnicas y metodologías, mejoras de anteriores, constantemente. Pensamos que es ahora donde se está explorando el potencial de esta tecnología y el trabajo detrás de este proyecto sitúa a su autor en una buena posición de partida para desarrollar su carrera profesional en este ámbito, ya que tiene un especial interés al respecto.

En suma, estas dificultades habrían sido más sencillas de haber desarrollado este proyecto en situaciones normales. La pandemia ha sido un añadido a esta complejidad que hemos intentando paliar con más esfuerzo y autoaprendizaje, cosa que nos ha motivado y nos ha hecho ver nuestra capacidad de sintetizar nuevos conceptos con la gran base que nos ha proporcionado nuestra universidad. Además, el apoyo por parte de nuestra

tutora, M. Carmen, y de nuestro director experimental, Fernando, ha sido irreprochable. Sin su ayuda la calidad de nuestro proyecto habría sido más reducida.

6.1 Relación del trabajo desarrollado con los estudios cursados

La realización de este trabajo ha seguido la metodología y los conocimientos que hemos ido adquiriendo a lo largo del grado. Por ello, debemos destacar algunas de las asignaturas que más nos han servido de guía, o de base, para elaborar correctamente el mismo.

Por un lado, los conocimientos de programación en conjunto de todas las asignaturas han sido fundamentales para llevar a cabo el trabajo; sin ellos habría sido imposible, ya que nos brindan de las competencias necesarias para la programación. Ejemplo de esto serían Programación y Estructuras de Datos y Algoritmos. Por otro lado, más específicamente, Redes de computadores, Tecnologías de sistemas de información en la red, Diseño y Configuración de Redes de Área Local y Desarrollo Web fueron las asignaturas principales para elaborar un pequeño servidor de prueba, con el *framework* de JavaScript Node.js, y la adaptación de este a un *plugin* para el Composer. Además, estas últimas también nos ayudaron a comprender el sistema que queríamos elaborar de cliente-servidor y la comprensión conceptual de lo que hacíamos.

Asignaturas, como Ingeniería del Software, han ayudado a comprender la estructuración de un proyecto como este y el ciclo de vida del desarrollo de *software*.

Interfaces Persona Computador y Desarrollo Centrado en el Usuario nos brindaron con ayudas y guías para crear interfaces de usuario intuitivas.

Integración de Aplicaciones nos hizo comprender que las aplicaciones no son del todo autónomas y que pueden usar JSON, como aquí, para intercambiar con otros procesos información y generar un sistema más complejo.

Matemática Discreta y Teoría de Autómatas y Lenguajes Formales nos ayudaron a orientarnos en el uso de expresiones formales; clave en nuestro aprendizaje, que usamos aquí.

6.2 Trabajos futuros

Conforme hemos ido desarrollando el proyecto nos han ido surgiendo posibles mejoras y ampliaciones que habría sido interesante implementar, pero debido al alcance del mismo eran inabarcables en este corto espacio de tiempo. Algunas de estas características adicionales serían:

- RA en superficies. Fue una de las primeras ideas en intentar implementarse, pero hacia el final de este proyecto se convirtió en un trabajo a realizar más adelante. SLAM es un sistema muy dependiente de la calidad de imagen de la cámara del dispositivo y la luz que capta, lo que complicó no solo el desarrollo de esta característica, sino la viabilidad de esta prueba de concepto. Aún así, se deberían realizar pruebas con dispositivos móviles diferentes para asegurarnos de ello, pero estaba fuera de nuestro alcance.

- Sistema de guardado y carga de escenas. Una de las mejoras más importantes, desde nuestro punto de vista. Este sistema consistiría en dotar a nuestra aplicación de la capacidad de guardar diferentes escenas, en este momento capaz de guardar únicamente una, y cargarlas para su visualización. De esta forma el usuario podría guardar diferentes versiones de un mismo trabajo o, incluso, diferentes trabajos o modelos propios que quiera probar a ver a escala real.
- iOS. Hemos hablado anteriormente de la posibilidad de hacer este trabajo para los dispositivos que usan iOS, sabemos que hay usuarios que no podrán servirse de esta aplicación. Esta mejora, si no hace llegar a todo el conjunto de usuarios potenciales, que no usan Android, nuestra aplicación; al menos la haría llegar a prácticamente todos los posibles.
- Mejora de la interfaz. Hemos implementado una interfaz sencilla y pensamos que sería interesante mejorar la experiencia del usuario tanto en nuestro complemento del Composer, como en la aplicación de teléfono móvil. Pensamos en el interés de ofrecer al usuario maneras de reconocer que la aplicación móvil funciona correctamente. Por ejemplo, añadir superficies reconocidas por nuestro sistema como idóneas para situar la escena que quiere visualizar el usuario con ayudas visuales, coloreando la zona reconocida, añadiendo pantallas de espera o información de carga, etcétera.

Además, debido a la formación que hemos recibido sabemos que sería imperativo analizar cuáles son las necesidades de los usuarios futuros y el contexto de uso de nuestra aplicación. Para ello sería conveniente realizar una investigación cualitativa; recolección de datos mediante cuestionarios, la observación de su forma de trabajar y estudiar los usos que van a darle a la aplicación; qué quieren representar virtualmente. A través de estos datos, definiríamos personas que representen a nuestros usuarios y diseñaríamos la solución entorno a ellos; suponemos desde ya que el objetivo de esta aplicación puede ser bastante uniforme, podrían ser dos grandes grupos: alumnos y profesores. Una vez hecho lo anterior intentaríamos representar el uso de nuestra aplicación, por estas personas, a través de escenarios. De esta forma podríamos adaptar esa futura aplicación a las necesidades reales de su público objetivo, no el nuestro como desarrollador.

Bibliografía

[1] Bortolini, M., Ferrari, E., Gamberi, M., Pilati, F. y Faccio, M. (2017). Assembly system design in the Industry 4.0 era: a general framework. *IFAC-PapersOnLine*, Vol. 50, Issue 1, pp. 5700-5705. https://doi.org/10.1016/j.ifacol.2017.08.1121

[2] Masood, T. y Egger, J. (2019). Augmented reality in support of Industry 4.0—Implementation challenges and success factors. *Robotics and Computer-Integrated Manufacturing*, Vol. 58, pp. 181-195. https://doi.org/10.1016/j.rcim.2019.02.003

[3] Azuma, R. (1997). A survey of augmented reality. *Presence: Teleoperators & Virtual Environments*, Vol. 6, no. 5, pp. 355-385. https://doi.org/10.1162/pres.1997.6.4.355

[4] Conn, C., Lannier, J., Minsky, M. y Druin, A. (1989). Virtual environments and interactivity: windows to the future. *SIGGRAPH89: 16th Annual ACM Conf on Computer Graphics and Interactive Techniques*, pp. 7-18. https://doi.org/10.1145/77276.77278

[5] Caudell, T. y Mizell, D. (1992). Augmented reality: an application of heads-up display technology to manual manufacturing processes, *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, Vol. 2, pp. 659-669. https://doi.org/10.1109/HICSS.1992.183317

[6] Sutherland, I. (1968). A Head-Mounted Three Dimensional Display, *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, pp. 757-764. https://doi.org/10.1145/1476589.1476686

[7] Krueger, M., Gionfriddo, T. y Hinrichsen, K. (1985). VIDEOPLACE—an Artificial Reality, *CHI '85: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 35-40. https://doi.org/10.1145/317456.317463

[8] Robinnet, W. y Rolland, J. (1993) 5 - A Computational Model for the Stereoscopic Optics of a Head-Mounted Display, *Virtual Reality Systems*, pp. 51-75. https://doi.org/10.1016/B978-0-12-227748-1.50013-5

[9] Feiner, S. MacIntyre, B., Höllerer, T., y Webster, A. (1997) A touring machine: Prototyping 3D mobile augmented reality systems for exploring the urban environment, *Personal and ubiquitous computing*, Vol. 1, no. 4, pp. 208-217. https://doi.org/10.1007/bf01682023

[10] Kato, H. y Billinghurst, M. (1999) Marker Tracking and HMD Calibrationfor a Video-based Augmented Reality Conferencing System, *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99)*, pp. 85-94. https://doi.org/10.1109/IWAR.1999.803809

[11] Thomas, B., Close, B., Donoghue, J., Squires, J., De Bondi, P., Morris, M. y Piekarski, W. (2000). ARQuake: an outdoor/indoor augmented reality first person application. *Digest of Papers. Fourth International Symposium on Wearable Computers*, pp. 139-146. https://doi.org/10.1109/ISWC.2000.888480

[12] Wagner, D. y Schmalstieg, D. (2003) First steps towards handheld augmented reality, *Seventh IEEE International Symposium on Wearable Computers, 2003. Proceedings.*, pp. 127-135. https://doi.org/10.1109/ISWC.2003.1241402

[13] Azuma, R., Baillot, Y., Behringer, R., Feiner, S., Julier, S. y MacIntyre, B. (2001) Recent advances in augmented reality, *IEEE Computer Graphics and Applications*, Vol. 21, no. 6, pp. 34-47. https://doi.org/10.1109/38.963459

[14] Papagiannakis, G., Singh, G. y Magnenat-Thalmann, N. (2008). A survey of mobile and wireless technologies for augmented reality systems. *Computer Animation and Virtual Worlds*, Vol. 19, no. 1, pp. 3-22. https://doi.org/10.1002/cav.221

[15] Carmigniani, J., Furht, B., Anisetti, M., Ceravolo, P., Damiani, E. y Ivkovic, M. (2010) Augmented reality technologies, systems and applications. *Multimedia Tools and Applications*, Vol. 51, no. 1, pp. 341-377. https://doi.org/10.1007/s11042-010-0660-6

[16] Felt, A., Egelman, S. y Wagner, D. (2012) I've got 99 problems, but vibration ain't one. *Proceedings of the second ACM workshop on security and privacy in smartphones and mobile devices*, pp. 33-44. https://doi.org/10.1145/2381934.2381943

[17] Grubert, J., Langlotz, T., Zollmann, S., y Regenbrecht, H. (2017) Towards Pervasive Augmented Reality: Context-Awareness in Augmented Reality. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 23, Issue 6, pp. 1706-1724. https://doi.org/10.1109/tvcg.2016.2543720

[18] Kinateder, M., Gualtieri, J., Dunn, M., Jarosz, W. (2018) Using an Augmented Reality Device as a Distance-based Vision Aid—Promise and Limitations. *Optometry and Vision Science*, Vol. 95, Issue 9, pp. 727-737 https://doi.org/10.1097/opx.0000000000001232

[19] Katz, B., Kammoun, S., Parseihian, G., Gutierrez, O., Brilhault, A. (2012) NAVIG: augmented reality guidance system for the visually impaired. Virtual Reality. *Virtual Reality*, Vol. 16, pp. 253-269. https://doi.org/10.1007/s10055-012-0213-6

[20] Juan, M. C y Calatrava, J. (2011) An Augmented Reality System for the Treatment of Phobia to Small Animals Viewed Via an Optical See-Through HMD: Comparison With a Similar System Viewed Via a Video See-Through HMD. *International Journal of Human–Computer Interaction*, Vol. 27, Issue 5, pp. 436-449. https://doi.org/10.1080/10447318.2011.552059

[21] Calle-Bustos, A. M., Juan, M. C., Garcia-Garcia, I. y Abad, F. (2017) An augmented reality game to support therapeutic education for children with diabetes. *PLoS ONE*, Vol. 12, no. 9. https://doi.org/10.1371/journal.pone.0184645

[22] Dunleavy, M., Dede, C. (2014) Augmented reality teaching and learning. *Handbook of research on educational communications and technology*, pp. 735-745. https://doi.org/10.1007/978-1-4614-3185-5_59

[23] Arvanitis, T.N., Petrou, A., Knight, J.F. et al. (2009) Human factors and qualitative pedagogical evaluation of a mobile augmented reality system for science education used by learners with physical disabilities. *Personal and Ubiquitous Computing*, Vol. 13, pp. 243-250. https://doi.org/10.1007/s00779-007-0187-7

[24] Akçayır, M., Akçayır, G., Pektaş, H., Ocak, M. (2016) Augmented reality in science laboratories: The effects of augmented reality on university students' laboratory skills and attitudes toward science laboratories. *Computers in Human Behavior*, Vol. 57, pp. 334-342. https://doi.org/10.1016/j.chb.2015.12.054

[25] Morillo, P., García-García, I., Orduña, J. M. y Juan, M. C (2019) Comparative study of AR versus video tutorials for minor maintenance operations. *Multimedia Tools and Applications volume*, Vol. 79, no. 11-12, pp. 7073-7100. https://doi.org/10.1007/s11042-019-08437-9

[26] Furió, D., Gónzalez-Gancedo, S., Juan, M. C., Seguí, I., Rando, N. (2013) Evaluation of learning outcomes using an educational iPhone game vs. traditional game. *Computers & Education*, Vol. 64, pp. 1-23. https://doi.org/10.1016/j.compedu.2012.12.001

[27] Furió, D., Gónzalez-Gancedo, S., Juan, M. C., Seguí, I., Costa, M. (2013) The effects of the size and weight of a Mobile device on an educational game. *Computers & Education*, Vol. 64, pp. 24-41. https://doi.org/10.1016/j.compedu.2012.12.015

[28] Juan, M. C., Furió, D., Alem, L., Ashworth, P., Cano, J. (2011) ARGreenet and BasicGreenet: Two mobile games for learning how to recycle. *Conference: International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG 2011)*, Paper Code H61, pp. 25-32.

[29] Juan, M. C., Carrizo, M., Giménez, M., Abad, F. (2011). Using an augmented reality game to find matching pairs. *International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG 2011)*, Paper Code: I13, pp. 59-66.

[30] Juan, M. C., Toffetti, G., Abad, F., Cano, J. (2010) Tangible cubes used as the user interface in an augmented reality game for edutainment. *The 10th IEEE International Conference on Advanced Learning Technologies (ICALT 2010)*, pp. 599-603. https://doi.org/10.1109/ICALT.2010.170

[31] Juan, M. C., Llop, E., Abad, F., Lluch, J. (2010) Learning words using augmented reality. *The 10th IEEE International Conference on Advanced Learning Technologies (ICALT 2010)*, pp. 422-426. https://doi.org/10.1109/ICALT.2010.123

[32] Juan, M. C., Beatrice, F., Cano, J. (2008) An augmented reality system for learning the interior of the human body. *The 8th IEEE International Conference on Advanced Learning Technologies (ICALT 2008)*, pp. 186-188. https://doi.org/10.1109/ICALT.2008.121

[33] Zsila, Á., Orosz, G., Bőthe, B., Tóth-Király, I., et al. (2018) An empirical study on the motivations underlying augmented reality games: The case of Pokémon Go during and after Pokémon fever. *Personality and Individual Differences*, Vol. 133, pp. 56-66. https://doi.org/10.1016/j.paid.2017.06.024

[34] LeBlanc, A., y Chaput, J-P. (2016) Pokémon Go: A game changer for the physical inactivity crisis? *Preventive Medicine*, Vol. 101, pp. 235-237. https://doi.org/10.1016/j.ypmed.2016.11.012

[35] Plecher, D., Eichorn, C., Kindl, J, Kreisig, S., et al. (2018) Dragon Tale - A Serious Game for Learning Japanese Kanji. *CHI PLAY '18 Extended Abstracts: Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play Companion Extended Abstracts*, pp. 577-583. https://doi.org/10.1145/3270316.3271536

[36] Scholz, J. y Smith, A. (2016) Augmented reality: Designing immersive experiences that maximize consumer engagement. *Business Horizons*, Vol. 59, Issue 2, pp. 149-161. https://doi.org/10.1016/j.bushor.2015.10.003

[37] Abou El-Seoud, S. y Taj-Eddin, I. (2019) An Android Augmented Reality Application for Retail Fashion Shopping. *International journal of interactive mobile technologies*, Vol. 13, no. 1, pp. 4-19. https://doi.org/10.3991/ijim.v13i01.9898

[38] Dacko, S. (2017) Enabling smart retail settings via mobile augmented reality shopping apps. *Technological forecasting & social change*, Vol. 124, pp. 243-256. https://doi.org/10.1016/j.techfore.2016.09.032

[39] You, X., Zhang, WW., Ma, M., Deng, C. y Yang, J. (2018) Survey on Urban Warfare Augmented Reality. *ISPRS INTERNATIONAL JOURNAL OF GEO-INFORMATION*, Vol. 7, no. 2, p. 46. https://doi.org/10.3390/ijgi7020046

[40] Hicks, J.D, Flanagan, R.A, Petrov, P.V, y Stoyen, A.D. (2002) Eyekon: augmented reality for battlefield soldiers *27th Annual NASA Goddard/IEEE Software Engineering Workshop*, pp. 156-163. https://doi.org/10.1109/SEW.2002.1199462

[41] Wilson, KL., Doswell, JT., Fashola, OS., Debeatham, W. et al. (2013) Using Augmented Reality as a Clinical Support Tool to Assist Combat Medics in the Treatment of Tension Pneumothoraces. *MILITARY MEDICINE*, Vol. 178, Issue 9, pp. 981-985. https://doi.org/10.7205/MILMED-D-13-00074

[42] Chen, H., Sun, G., Chen, C., y Wan, B. (2019) Augmented Reality Based Assembly Guidance for Spacecraft Conductive Network. *IOP Conference Series. Earth and Environmental Science*, Vol. 252, Issue 4. http://dx.doi.org/10.1088/1755-1315/252/4/042076

[43] Chen, H., Sun, G., Chen, C., y Wan, B. (2019) Augmented Reality Based Visualization Method for Spacecraft Cable Assembly Process. *IOP Conference Series. Earth and Environmental Science*, Vol. 612, Issue 3. http://dx.doi.org/10.1088/1757-899X/612/3/032184

[44] Foxman, M. (2019) United We Stand: Platforms, Tools and Innovation With the Unity Game Engine. *Social Media + Society*, Vol. 5, no. 5. https://doi.org/10.1177/2056305119880177

[45] Lee, J. (2016) Learning unreal engine game development a step-by-step guide that paves the way for developing fantastic games with Unreal Engine 4. *Packt Publishing Ltd.*, 1st ed.

[46] Linowes, J., y Babilinski, K. (2017) Augmented reality for developers: build practical augmented reality applications with Unity, ARCore, ARKit, and Vuforia. *Packt Publishing Ltd.*

[47] Neumann, U., y You, S. (1999) Natural feature tracking for augmented reality. *IEEE Transactions on Multimedia*, Vol. 1, no. 1, pp. 53-64. https://doi.org/10.1109/6046.748171

[48] Fei, T., Xiao-Hui, L., Zhi-Ying, H.y Guo-Liang, H. (2008) A Registration Method Based on Nature Feature with KLT Tracking Algorithm for Wearable Computers. *2008 International Conference on Cyberworlds*, pp. 416-421. https://doi.org/10.1109/CW.2008.29

[49] Cadena, C., Carlone, L., Carrillo, H., Latif, et al. (2016) Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age. *IEEE Transactions on Robotics*, Vol. 32, no. 6, pp. 1309-1332. https://doi.org/10.1109/TRO.2016.2624754

[50] Klein, G and Murray, D. (2007) Parallel Tracking and Mapping for Small AR Workspaces. *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 225-234. https://doi.org/10.1109/ISMAR.2007.4538852

[51] Mur-Artal, R., Montiel, J. M. M., y Tardos, J. D. (2015) ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE transactions on robotics*, Vol. 31, no. 5, pp. 1147-1163. https://doi.org/10.1109/tro.2015.2463671

[52] Lu, F., y Milios, E. (1997) Globally Consistent Range Scan Alignment for Environment Mapping. *Autonomous Robots*, Vol. 4, pp. 333-349. https://doi.org/10.1023/A:1008854305733

[53] Davison, A., Reid, I., Molton, N., y Stasse, O. (2007) MonoSLAM: real-time single camera SLAM. *IEEE transactions on pattern analysis and machine intelligence*, Vol. 29, pp. 1052-1067. https://doi.org/10.1109/TPAMI.2007.1049

[54] Schönberger, J. L., Pollefeys, M., Geiger, A., y Sattler, T. (2017) Semantic visual localization. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6896-6906.

[55] Lanham, M. (2018) Learn ARCore - fundamentals of Google ARCore. *1st ed. Packt Publishing*

[56] Buerli, M., y Misslinger, S. (2017) Introducing ARKit-augmented reality for iOS. *Apple Worldwide Developers Conference (WWDC 2017)*.

[57] Doran, J. P., Sherif, W., y Whittle, S. (2019) Unreal Engine 4.x Scripting with C++ Cookbook - Second Edition. *2nd ed. Packt Publishing*, pp. 535-542.

[58] Milgram, P., y Kishino, F. (1994) A taxonomy of mixed reality visual displays. *IEICE Transactions on Information and Systems*, Vol. 77, no 12, pp. 1321-1329.

[59] Lee, L., y Hui, P. (2018) Interaction Methods for Smart Glasses: A Survey. *IEEE Access*, Vol. 6, pp. 28712-28732. https://doi.org/10.1109/ACCESS.2018.2831081

APÉNDICE A
Configuración e inicialización del sistema

Para usar este sistema primero debemos descargar el *software* Bg2 Composer[1]. Actualmente está disponible la versión 3.1, de marzo de 2020, por lo que detallaremos el uso que se hará con esta versión específicamente.

Cuando lo hayamos hecho dispondremos, en el directorio donde se localice la descarga, de un archivo comprimido tipo .zip, que se podrá descomprimir con el programa WinRAR o 7-Zip, por ejemplo. Realizado este punto accederemos a la carpeta composer y buscaremos el ejecutable «composer», como se muestra en la figura A.1. Lo ejecutamos.

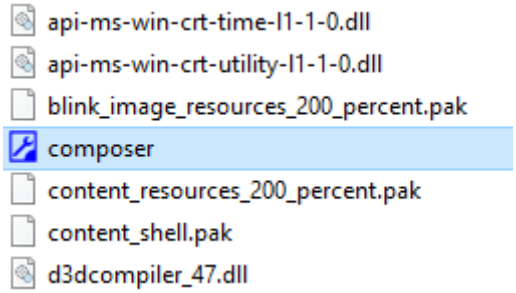
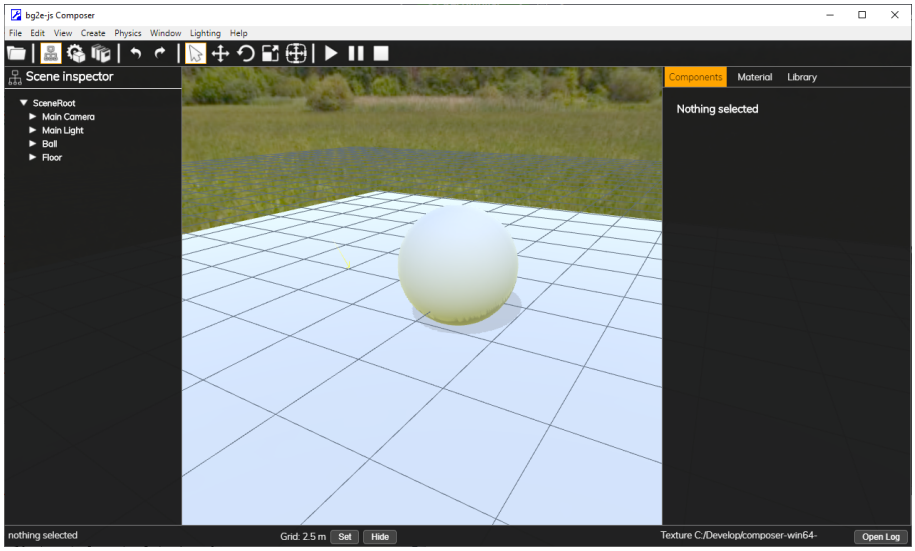


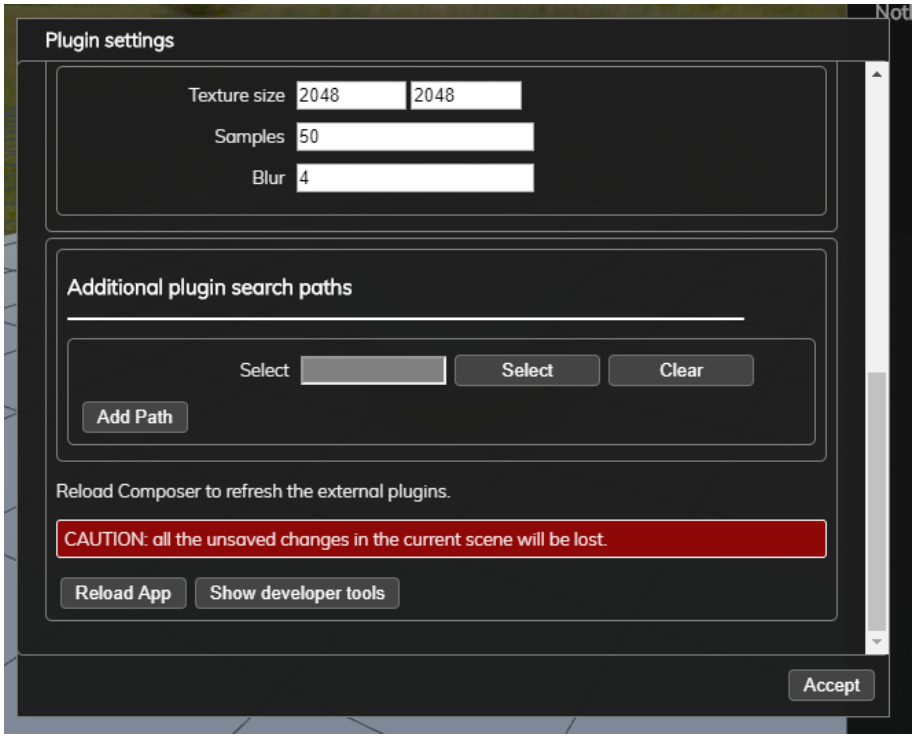
Figura A.1: Archivo composer

[1] https://www.bg2engine.org/#/downloads *Revisado en agosto 2020*

Cuando se haya abierto la aplicación nos aparecerá la pantalla del editor (figura A.2.a):



(a) Pantalla edición del Composer



(b) Ventana configuración de *plugins*

Figura A.2: Editor de la aplicación Composer

Para añadir nuestro *plugin* debemos acceder a File > Plugin Settings. Nos aparecerá una ventana como la de la figura A.2.b, en la que debemos incluir la ruta del complemento que queremos añadir.

En el directorio de nuestro trabajo se encuentra la carpeta «composer-plugins». Esta contendrá el complemento que debemos seleccionar desde la ventana de la aplicación. Por lo que, seleccionamos esta ruta, mediante el buscador, y clicamos en «Add Path».

Debería quedarnos algo como lo siguiente:

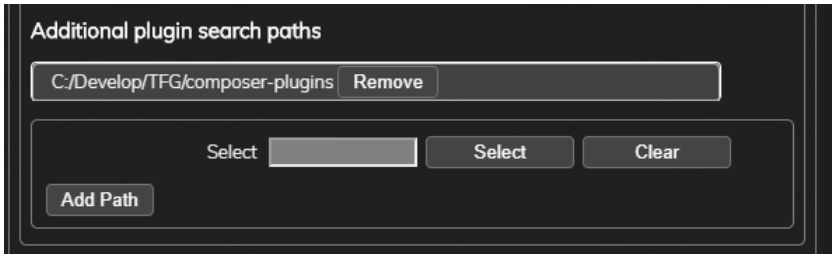


Figura A.3: Añadiendo la ruta del *plugin* al Composer

Una vez añadida la ruta recargaremos la aplicación, como vemos en la figura A.2.b, clicando en «Reload App» y le damos a aceptar al diálogo emergente al hacerlo. Si hemos finalizado el proceso correctamente veremos en la barra de herramientas que se ha añadido una nueva opción llamada «AR».

Para utilizar la funcionalidad que hemos añadido debemos acceder a esta opción y clicar en la opción del submenú que se nos muestra, lo que abrirá una ventana como la de la figura A.4:

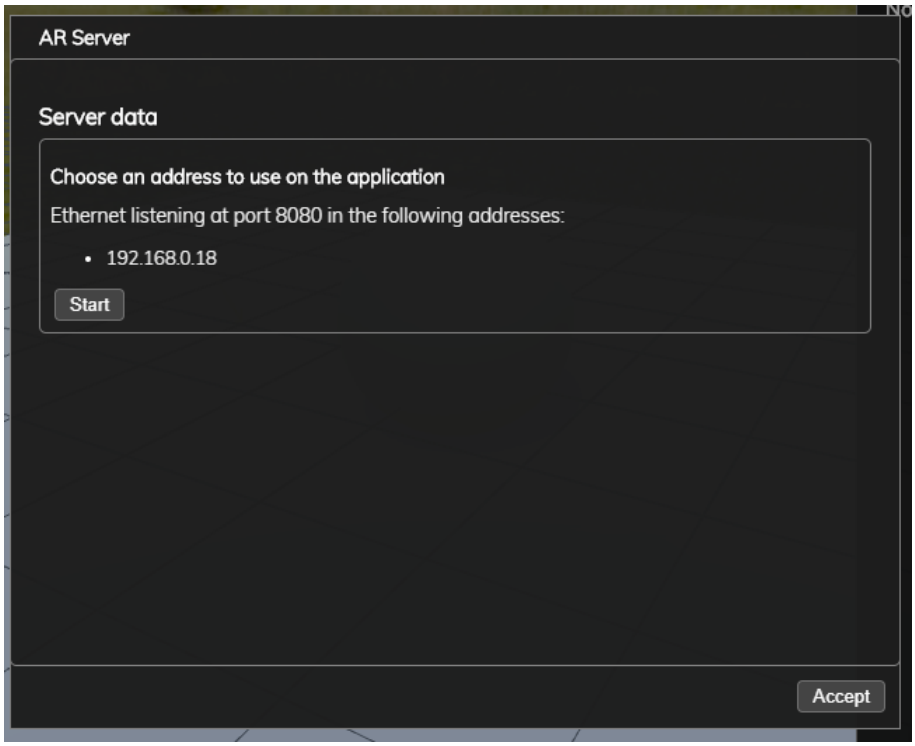


Figura A.4: Ventana inicio servidor del Composer

Aquí se nos mostrarán las opciones que podemos escoger para utilizar en la aplicación de nuestro dispositivo móvil. En nuestro caso deberemos elegir la única que aparece, la dirección 192.168.0.18. Teniendo esto claro clicaremos en «Start» y podremos realizar la descarga de esta escena en el móvil, si está conectado a la misma red que nuestro ordenador.

Para instalar nuestra aplicación del dispositivo móvil tendremos que, primero, asegurarnos de que es compatible con ARCore[2], sino no podremos probar la aplicación. Lo segundo sería ponerlo en modo depuración. Esto es diferente en cada dispositivo, pero es fácil de hacer; basta con una búsqueda por internet y seguir unos sencillos pasos que nos guiarán para hacerlo.

Una vez configurado iremos a la página web del repositiorio (https://github.com/mhaba/TFG), y desde aquí podemos descargarnos todo el proyecto o accedemos al directorio Build, donde se situa un archivo .bat. Haremos esto último.

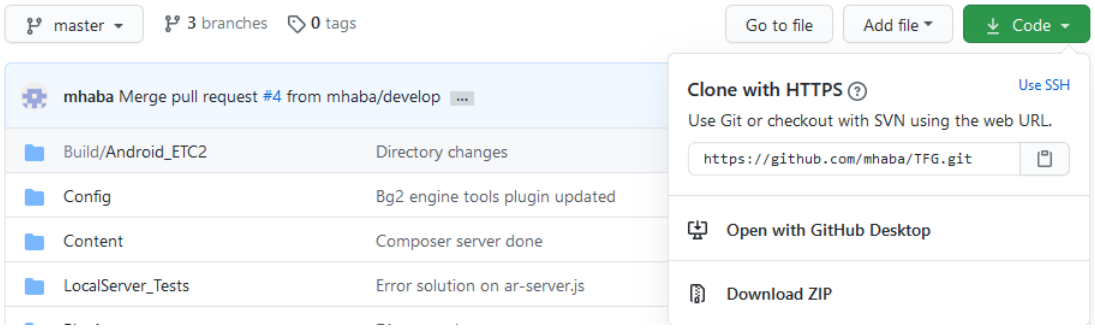


Figura A.5: Descarga de repositorio

Iremos al directorio donde hayamos guardado la descarga y accederemos a su carpeta, como vemos en la figura A.6. Aquí haremos doble clic en el primer recurso, «Install_TFG-armv7» mientras tenemos conectado nuestro teléfono al ordenador, lo que procederá a instalar la aplicación en nuestro dispositivo.

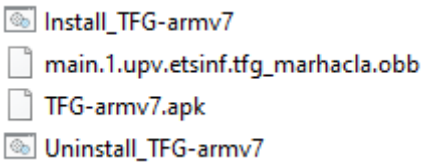


Figura A.6: Directorio descargado

Una vez instalada en nuestro teléfono la buscaremos, es una aplicación llamada «TFG»; figura A.7, la iniciaremos y nos aparecerá en la pantalla lo mismo que en la figura 4.4 del capítulo de Diseño de la solución. Si pulsamos sobre «Iniciar descarga», habiendo iniciado el servidor, se nos descargará la escena que tengamos en el Composer (en este caso, si no hemos modificado nada de la escena, se descargaría lo mismo que aparece en la figura A.2.a, ya que es la escena de ejemplo).

[2] https://developers.google.com/ar/discover/supported-devices

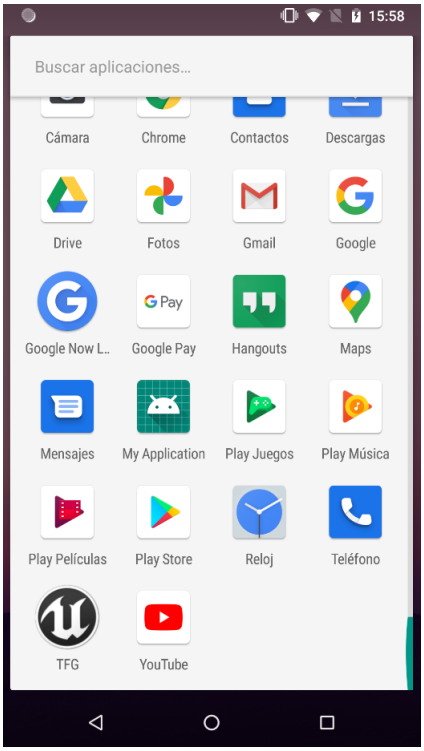


Figura A.7: Icono y búsqueda de la aplicación

A partir de aquí se pedirán permisos para usar la cámara del dispositivo y, si accedemos a que se use, se cargará la escena en RA y se mostrará en la pantalla junto al mundo real.