



TRABAJO DE FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Realidad aumentada para la monitorización de servicios de TI en movilidad

Estudiante: Isaac Cores López
Dirección: Ángel Gómez García
José Carlos Dafonte Vázquez

A Coruña, 7 de septiembre de 2020.

A mi familia y amigos.

Agradecimientos

A mis padres, gracias a su esfuerzo he podido llegar hasta aquí.

A mis amigos por su apoyo y compañía.

A mis tutores Ángel Gómez García y José Carlos Dafonte Vázquez por guiarme y aconsejarme durante el desarrollo de este proyecto.

Resumen

La monitorización de sistemas dentro de un centro de procesamiento de datos (CPD) proporciona al personal técnico una gran cantidad de información en tiempo real sobre el estado de los activos (máquinas físicas y virtuales) y de los servicios bajo supervisión dentro de cada activo.

En el presente Trabajo de Fin de Grado (TFG) se lleva a cabo el desarrollo de una aplicación para dispositivos móviles centrada en la utilización de técnicas de realidad aumentada (RA) para integrar y mejorar las funcionalidades de las herramientas de monitorización. Así se proporcionará al usuario un sistema de acceso más rápido e intuitivo a la información de los equipos a los que está observando o que estén situados físicamente cerca. También, se mejorará la representación de la información de una forma más visual, simplificada, esquemática e inteligible.

Se comenzará con un estudio sobre las herramientas de desarrollo de RA disponibles en el mercado. Se hará una comparativa de las funcionalidades de cada una de ellas, mostrando sus ventajas y desventajas. De entre estas herramientas, ARCore fue finalmente la tecnología utilizada.

La aplicación desarrollada detecta marcadores (tanto imágenes como códigos QR) asociados a los equipos del CPD y que referencian la visualización mediante RA obteniendo la información sobre el estado de ese equipo concreto mediante la conexión con el software de monitorización Zabbix. Si el equipo es un rack, la aplicación muestra un resumen del estado de todos sus componentes. Si el equipo es un componente de rack, advierte la presencia de problemas, indica su disponibilidad y proporciona datos sobre sus recursos (CPU, memoria, almacenamiento y red).

Esta información se visualiza mediante diferentes estilos como interfaces con texto (tanto integradas como independientes a la RA), gráficos vectoriales y modelos 3D representativos. Además, se utiliza la interacción con estos elementos para permitir el acceso a distintos niveles de información.

Abstract

System monitoring in a data center provides technical staff with a large amount of real-time information about the status of assets (physical and virtual machines) and services under supervision within each asset.

This end-of-degree project develops an application for mobile devices focused on the use of augmented reality (AR) techniques to integrate and improve the functionality of monitoring

tools. Thus the user will be provided with a faster and more intuitive access to information about the devices he is observing or that is physically located nearby. Also, the representation of information will be improved in a more visual, simplified, schematic and intelligible way.

It will start with a study of the AR development tools available on the market. A comparison of the functionalities of each of them will be made, showing their advantages and disadvantages. Among these tools, ARCore was finally the technology used.

The developed application detects markers (both images and QR codes) associated with the data center's devices and that reference the AR display obtaining the information about the status of that particular device by connecting to the monitoring software Zabbix. If the device is a rack, the application displays a summary of the status of all its components. If the device is a rack component, it notices the presence of problems, indicates its availability and provides data about its resources (CPU, memory, storage and network).

This information is displayed using different styles such as text interfaces (both integrated and independent of the AR), vector graphics and representative 3D models. In addition, interaction with these elements is used to allow access to different levels of information.

Palabras clave:

- realidad aumentada
- monitorización
- AR.js
- ARCore
- Unity
- Zabbix
- dispositivo móvil
- Android
- ARMo

Keywords:

- augmented reality
- monitoring
- AR.js
- ARCore
- Unity
- Zabbix
- mobile device
- Android
- ARMo

Índice general

1	Introducción	1
1.1	Objetivos y alcances	2
1.2	Descripción de la memoria	3
2	Estado del arte	5
2.1	Introducción	5
2.2	Historia	5
2.3	Aplicaciones	10
2.3.1	Aplicaciones con funcionalidad similar	10
2.3.2	Aplicaciones con funcionalidad relacionada	12
3	Herramientas de realidad aumentada	15
3.1	ARCore	15
3.2	ARKit	16
3.3	Wikitude	17
3.4	Vuforia	18
3.5	AR.js	19
3.6	Comparativa	22
4	Desarrollo	23
4.1	Tecnologías empleadas	23
4.1.1	ARCore	23
4.1.2	Unity	23
4.1.3	Visual Studio	28
4.1.4	C#	28
4.1.5	Zabbix	28
4.1.6	VM VirtualBox	30
4.1.7	GitHub	30

4.2	Metodología	30
4.2.1	Scrum	31
4.3	Planificación	33
4.3.1	Sprint 0: Establecimiento de las bases del proyecto y búsqueda de tecnología de realidad aumentada	35
4.3.2	Sprint 1: Primeros pasos con AR.js	36
4.3.3	Sprint 2: Creación de un primer prototipo de interfaz	37
4.3.4	Sprint 3: Implementación de la lectura de códigos QR	39
4.3.5	Sprint 4: Primeros pasos con ARCore	40
4.3.6	Sprint 5: Creación de un primer prototipo de interfaz	49
4.3.7	Sprint 6: Experimentando con los diferentes estilos de representación de la información	52
4.3.8	Sprint 7: Simulación de la conexión entre la aplicación y el software de monitorización Zabbix	54
4.3.9	Sprint 8: Conexión entre la aplicación y Zabbix sobre un escenario simulado compuesto por máquinas virtuales	55
4.4	Recursos y estimación de costes	61
4.4.1	Descripción de los recursos	61
4.4.2	Estimación de costes	62
4.5	Análisis	63
4.6	Diseño	63
4.6.1	Diseño gráfico	69
4.7	Implementación	74
4.8	Pruebas	75
4.8.1	Pruebas unitarias	75
4.8.2	Pruebas de integración	75
4.8.3	Pruebas de sistema	76
4.8.4	Pruebas de aceptación y validación	76
5	Conclusiones	77
6	Líneas futuras	79
A	Guía de usuario	83
	Lista de acrónimos	9
	Glosario	11

ÍNDICE GENERAL

Bibliografía

13

Índice de figuras

2.1	Máquina Sensorama.	6
2.2	Dispositivo Espada de Damocles (Sword of Damocles).	7
2.3	Diseño del sistema Videoplace.	7
2.4	Diseño del Virtual fixture.	8
2.5	Videojuego de realidad aumentada Pokémon GO.	9
2.6	Aplicación OpenManage Mobile.	11
2.7	Proyecto Stefanini AR.	12
2.8	Aplicación EcoStruxure Augmented Operator Advisor.	13
2.9	Aplicación Senko AR.	14
2.10	Aplicación Dell EMC AR Assistant.	14
3.1	Ejemplo sobre ARCore.	16
3.2	Ejemplo sobre ARKit.	17
3.3	Ejemplo sobre Wikitude.	18
3.4	Ejemplo sobre Vuforia.	19
3.5	Ejemplo sencillo de código AR.js.	20
3.6	Marcador Hiro.	20
3.7	Ejemplo de imagen transformada en un Pattern Marker de AR.js.	21
3.8	Ejemplo de Barcode Marker de AR.js.	21
4.1	Ventana Project de Unity.	26
4.2	Ventana Scene de Unity.	26
4.3	Ventana Hierarchy de Unity.	26
4.4	Inspector de Unity.	27
4.5	Ventana Toolbar de Unity.	28
4.6	Ventana Game de Unity.	28
4.7	Diagrama de la integración de la aplicación de RA con Zabbix.	29
4.8	Diagrama de Gantt con las tareas del proyecto.	34

4.9	Aplicación AR.js para probar la interacción con el usuario.	37
4.10	Prototipo de interfaz desarrollado con AR.js.	38
4.11	Combinación de un marcador AR.js con un código QR.	40
4.12	Captura del proyecto Augmented Faces de ARCore.	41
4.13	Captura del proyecto Augmented Image de ARCore.	42
4.14	Captura del proyecto HelloAR de ARCore.	42
4.15	Fragmento del script AugmentedImageExampleController.cs.	45
4.16	Fragmento del script AugmentedImageVisualizer.cs.	46
4.17	Subventana con información sobre una base de datos de imágenes.	48
4.18	Prototipo de interfaz cuando se detecta una sola imagen.	50
4.19	Prototipo de interfaz cuando se detectan dos imágenes de forma simultánea. .	51
4.20	Interacción del usuario con el prototipo de interfaz mientras se detectan dos imágenes simultáneamente.	52
4.21	Detalle del archivo XML para simular la conexión de la aplicación con Zabbix.	55
4.22	Detalle de la interfaz web de Zabbix.	57
4.23	Diagrama del funcionamiento de Zabbix.	57
4.24	Detalle de un gráfico de Zabbix que representa el porcentaje de uso de un recurso.	58
4.25	Visualización mediante RA de la información de los equipos de un CPD. . .	60
4.26	Interfaz con información detallada de los equipos de un CPD.	60
4.27	Diagrama de flujo entre las escenas de la aplicación.	66
4.28	Detalle del diagrama de clases de la escena AR	67
4.29	Diagrama de clases del CPD.	68
4.30	Modelos 3D de los elementos del CPD.	69
4.31	Gráficos vectoriales que representan los niveles de severidad de un problema.	69
4.32	Gráficos vectoriales utilizados para representar la disponibilidad de los hosts. .	70
4.33	Interfaz de la escena AR cuando no se detecta ninguna imagen.	71
4.34	Interfaz de la escena AR cuando se detecta una sola imagen.	72
4.35	Interfaz de la escena AR cuando se detectan dos imágenes simultáneamente. .	73
4.36	Interfaz de la escena Information	73
4.37	Detalle de un objeto Canvas con sus Components e hijos.	74
4.38	Representación del área Canvas en la ventana Scene.	75
A.1	Ventana de inicio.	2
A.2	Pantalla de carga.	2
A.3	Recuadro para ajustar la imagen a escanear.	3
A.4	Detección de una imagen asociada a un rack.	4
A.5	Detección de una imagen asociada a un componente de rack.	4

ÍNDICE DE FIGURAS

A.6 Niveles de severidad	5
A.7 Estados de la conexión de los componentes de rack.	5
A.8 Pestaña asociada a un recurso.	5
A.9 Ventana con información detallada sobre los activos del CPD.	6
A.10 Detección de dos imágenes de forma simultánea.	7

Índice de tablas

3.1	Comparación entre herramientas de desarrollo de realidad aumentada.	22
4.1	Estimación del coste de los recursos materiales del proyecto.	62
4.2	Estimación del coste de los recursos humanos del proyecto.	63

Capítulo 1

Introducción

La monitorización se encarga de la supervisión del estado y funcionamiento de componentes en diferentes ámbitos de la informática como por ejemplo los servicios, las aplicaciones, la administración de redes y las bases de datos. El proceso de monitorización supone un papel esencial dentro de un sistema debido a la gran importancia de sus funcionalidades, entre las que se encuentran la optimización de los recursos disponibles y la prevención, detección y búsqueda de fallos. Este papel es indispensable dentro un centro de procesamiento de datos (CPD), lugar donde la monitorización proporciona una gran cantidad de información en tiempo real sobre el estado de los activos (máquinas físicas y virtuales) y de los servicios bajo supervisión dentro de cada activo.

Inicialmente, los CPDs solo disponían de herramientas de monitorización en forma de aplicación de escritorio, que se manejaba a través de una interfaz o comandos de terminal, instalada en un equipo no portable. Esto obligaba al personal técnico a desplazarse continuamente al ordenador durante la inspección de una máquina para consultar el estado de esta. La llegada de los smartphones aportó nuevas formas de usar las herramientas de monitorización, actualmente estas herramientas se pueden utilizar en cualquier lugar del CPD a través de una aplicación móvil o web consiguiendo reducir el tiempo de respuesta de los técnicos ante incidentes y logrando que su trabajo se vuelva más ágil, flexible y eficiente. Y si se combinan las prestaciones de los dispositivos móviles con las capacidades de la realidad aumentada (RA) se puede lograr un paso más en la mejora de la monitorización de los CPDs.

En los últimos años, la realidad aumentada ha llegado a un punto álgido gracias a los avances tecnológicos alcanzados por los dispositivos móviles como los smartphones, tablets y smartglasses. El uso de la RA se ha llevado a diferentes campos como el entretenimiento, la educación, la medicina y la industria aprovechando el espectro de posibilidades que ofrece en el manejo de la información. Esta gran cantidad de posibilidades se debe, en parte, a que la RA tiene la capacidad de integrar los datos en el mundo real a través de múltiples técnicas como la detección de marcadores, que pueden consistir en imágenes o códigos de barras, la detección

de superficies, el reconocimiento de objetos y el uso de ubicaciones. La realidad aumentada también ofrece muchas formas de representar la información mediante interfaces con texto, etiquetas, gráficos vectoriales, modelos 2D, modelos 3D, sonidos, vídeos, animaciones, etc. El presente Trabajo de Fin de Grado surge con la idea de utilizar estas cualidades de la RA para mejorar el proceso de monitorización de los CPDs.

1.1 Objetivos y alcances

El objetivo de este proyecto es el desarrollo de una aplicación para dispositivos móviles centrada en la utilización de técnicas de realidad aumentada para integrar y mejorar las funcionalidades de las herramientas de monitorización de servicios de TI. Así se proporcionará al usuario un sistema de acceso más rápido e intuitivo a la información de los equipos a los que está observando o que estén situados físicamente cerca. También, se facilitará el proceso de localización de los activos dentro del CPD y se mejorará la representación de la información de una forma más visual, simplificada, esquemática e inteligible. El objetivo principal se compone de los siguientes objetivos concretos:

- Realizar un estudio sobre las herramientas RA disponibles en el mercado en el cual se llevará a cabo una comparativa de las funcionalidades de cada una de ellas, mostrando sus ventajas y desventajas. El propósito del estudio es elegir la herramienta más adecuada para cumplir con las necesidades del proyecto.
- Conseguir un acceso más inmediato e intuitivo a la información mediante la detección de marcadores, que pueden consistir en una imagen o un código de barras, asociados a los equipos del CPD y que referencian la visualización mediante RA. Al detectar estos marcadores se llamará a una aplicación de monitorización para obtener la información sobre el estado de ese equipo concreto.
- Estudiar y experimentar con los diferentes estilos para visualizar la información que ofrece la RA, como por ejemplo los anteriormente mencionados gráficos vectoriales, gráficos 2D y 3D. Además de utilizar la interacción con estos elementos para acceder a distintos niveles de información.
- Posibilitar el acceso y la explotación de la información proporcionada por las herramientas de monitorización de sistemas, visualizando los indicadores más significativos de los activos y permitiendo el acceso directo a un mayor detalle.

1.2 Descripción de la memoria

En esta sección se realiza una breve descripción de cada uno de los capítulos que componen la memoria:

- **Capítulo 1: Introducción.** En este capítulo se lleva a cabo una presentación del proyecto, de sus motivaciones y objetivos. Además, se incluye una breve explicación de la estructura de la memoria.
- **Capítulo 2: Estado del arte.** Este capítulo está dedicado a explicar el contexto histórico y tecnológico de la realidad aumentada y expone las principales aplicaciones de la misma temática del proyecto.
- **Capítulo 3: Herramientas de realidad aumentada.** En este capítulo se analizan las herramientas de desarrollo de realidad aumentada más relevantes de la actualidad.
- **Capítulo 4: Desarrollo.** Este capítulo describe el proceso de desarrollo seguido en la creación de la aplicación incluyendo una explicación de las tecnologías empleadas y la metodología aplicada. Dentro de esta metodología se incluirá la descripción de la planificación efectuada y de las fases del proceso software: análisis, diseño, implementación y pruebas.
- **Capítulo 5: Conclusiones.** En este capítulo se analiza la aplicación creada y se compara el trabajo realizado con los objetivos iniciales.
- **Capítulo 6: Líneas futuras.** En este capítulo se plantean funcionalidades y modificaciones que en un futuro puedan ser implementadas en la aplicación.

Capítulo 2

Estado del arte

En este capítulo se realiza una introducción a la realidad aumentada, se describe parte de su historia y se exponen aplicaciones de la misma temática que este Trabajo de Fin de Grado con el objetivo de establecer un contexto tanto histórico como tecnológico.

2.1 Introducción

La realidad aumentada es una tecnología que permite la visualización del mundo real combinando elementos reales con elementos virtuales para mejorar la experiencia del usuario [1]. Estos elementos virtuales suelen ser gráficos 2D o 3D, pero también pueden consistir en texto, imágenes, audio, vídeo, etc. Las principales cualidades que caracterizan a la realidad aumentada son:

- La información está integrada en el mundo real.
- Es interactiva.
- Es un sistema de tiempo real.
- Permite la movilidad del usuario.
- A diferencia de la realidad virtual, la realidad aumentada complementa el mundo real en vez de remplazarlo.

2.2 Historia

La historia de la realidad aumentada está ligada a la realidad virtual e incluye una serie de hitos que han influido de forma importante en su evolución. A continuación se describen algunos de estos hitos [2][3]:



Figura 2.1: Máquina Sensorama.

- 1957-1962: el artista cinematográfico Morton Heilig da los primeros pasos de la realidad aumentada con Sensorama (Figura 2.1), comenzando su desarrollo en 1957 y patentándolo en 1962. Sensorama consiste en una máquina que emplea imágenes, sonidos, vibraciones y olores para simular un paseo en motocicleta por Nueva York.
- 1968: el científico Ivan Sutherland y su estudiante Bob Sproull crean el considerado primer visor de realidad aumentada y virtual o HMD (Head Mounted Display), que fue apodado como Espada de Damocles (Sword of Damocles). Este dispositivo (Figura 2.2) está formado por un gran casco, colgado del techo, que posee unas gafas por las cuales se ven unos gráficos sencillos que cambian de ángulo con el movimiento del usuario.
- 1975: Myron Krueger crea Videoplace (Figura 2.3). En Videoplace hay una cámara que graba al usuario mientras su silueta es proyectada sobre una pantalla que tiene justo delante. También son proyectados objetos virtuales que reaccionan a los movimientos del usuario.
- 1990: el investigador Tom Caudell (Boeing) crea el término "realidad aumentada".

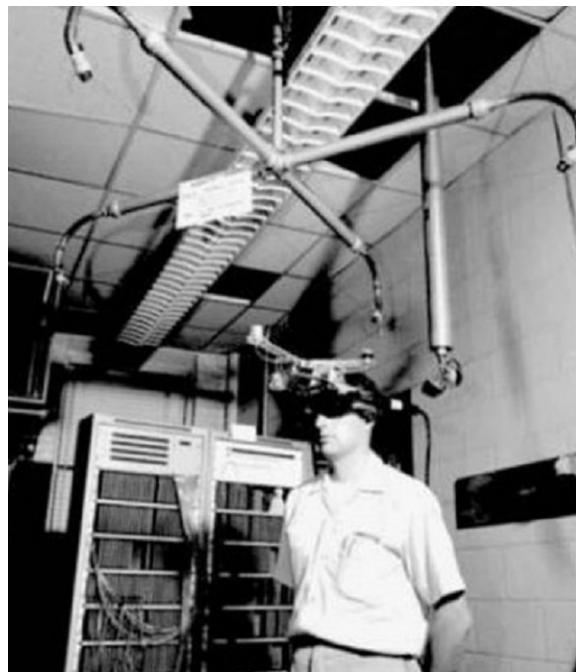


Figura 2.2: Dispositivo Espada de Damocles (Sword of Damocles).

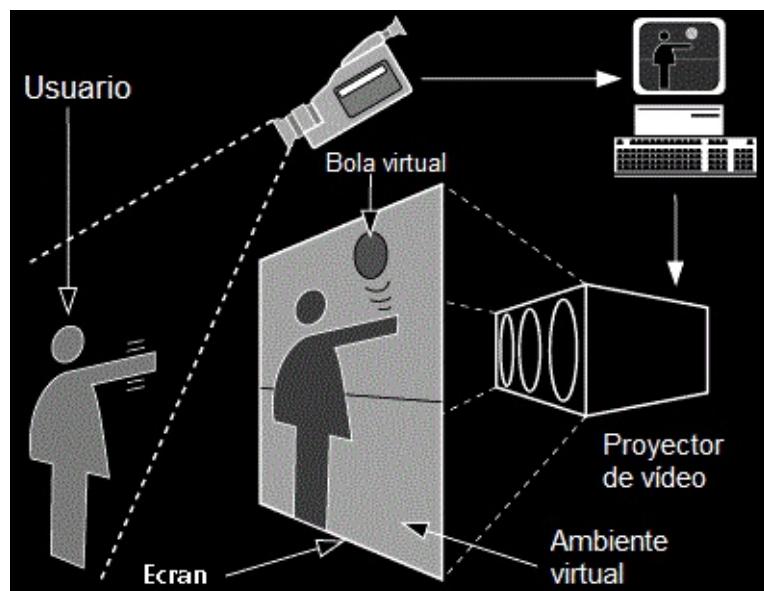


Figura 2.3: Diseño del sistema Videoplace.

- 1992: Louis Rosenberg desarrolla Virtual fixture (Figura 2.4) para la Fuerza Aérea de los Estados Unidos. Virtual fixture, considerado el primer sistema de realidad aumentada inmersiva, permite realizar tareas en un entorno remoto (remote environment) desde el llamado espacio del operador (operator space):
 - Entorno remoto (remote environment). Como se puede observar en la imagen de la Figura 2.4a, consta de un brazo robótico, una cámara y el objeto físico a manipular.
 - Espacio del operador (operator space). Como indica la imagen de la Figura 2.4b, el usuario se coloca unos binoculares que muestran la grabación de la cámara en tiempo real y también se equipa un exoesqueleto para controlar los movimientos del brazo robótico. En el vídeo se suelen incluir objetos virtuales para facilitarle las tareas al usuario, como por ejemplo una regla para aumentar la precisión.

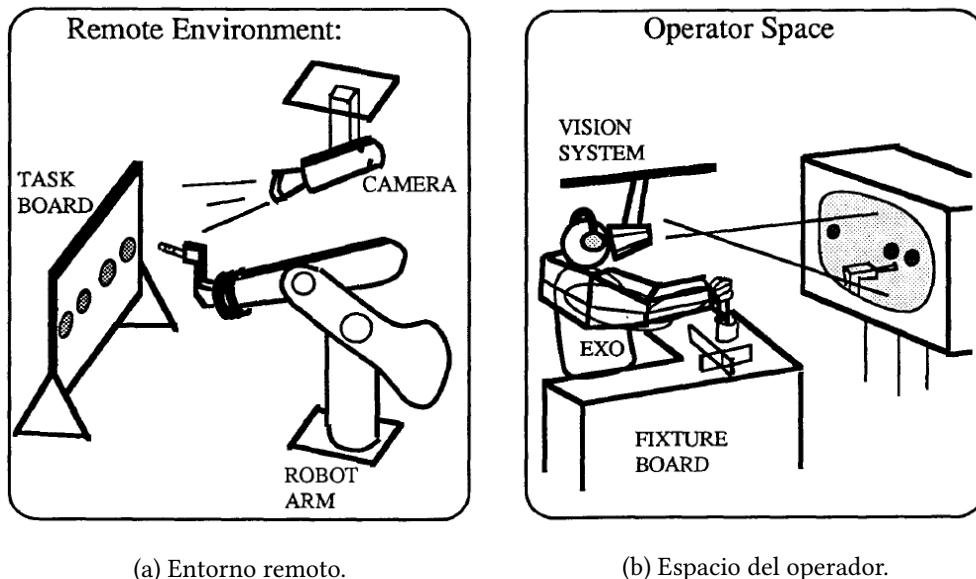


Figura 2.4: Diseño del Virtual fixture.

- 1994: Julie Martin combina las artes escénicas con la realidad aumentada en "Dancing in Cyberspace" donde bailarines y acróbatas bailan dentro y alrededor de objetos virtuales proyectados sobre el escenario.
- 1998: Sportvision utiliza el sistema "1st and Ten Line" durante un partido de la NFL mostrando la primera línea amarilla virtual sobre el campo de juego.
- 1999: la nave NASA X-38 utiliza el software LandForm para proyectar mapas virtuales sobre imagen real.

CAPÍTULO 2. ESTADO DEL ARTE

- 2000: Hirokazu Kato crea la librería de código abierto ARToolKit para desarrollar aplicaciones de realidad aumentada.
- 2000: Bruce Thomas presenta el primer juego de realidad aumentada al aire libre AR-Quake, una versión del juego de escritorio Quake.
- 2003-2005: comienzan a difundirse aplicaciones de realidad aumentada para móviles.
- 2009: ARToolkit es implementado para su funcionamiento en navegadores web.
- 2012: Google anuncia las Google Glass. Las Google Glass consisten en smartglasses compatibles con la realidad aumentada.
- 2013: la realidad aumentada empieza a ser utilizada por los fabricantes de automóviles para la creación de manuales. Un ejemplo es la aplicación MARTA de Volkswagen.
- 2016: la empresa Niantic lanza el videojuego de realidad aumentada para móviles Pokémon GO (Figura 2.5¹), convirtiéndose en uno de los videojuegos para iOS y Android más descargados.



Figura 2.5: Videojuego de realidad aumentada Pokémon GO.

¹Fuente de la imagen: <https://pokemongolive.com/img/homepage/vid-still.jpg>

2.3 Aplicaciones

En este apartado se hace una distinción entre aplicaciones con una funcionalidad similar a la planteada en este Trabajo de Fin de Grado y aplicaciones de realidad aumentada con una funcionalidad relacionada. Como se mencionó en la introducción de la memoria, este TFG tiene como finalidad utilizar la realidad aumentada para mejorar el proceso de monitorización de los CPDs.

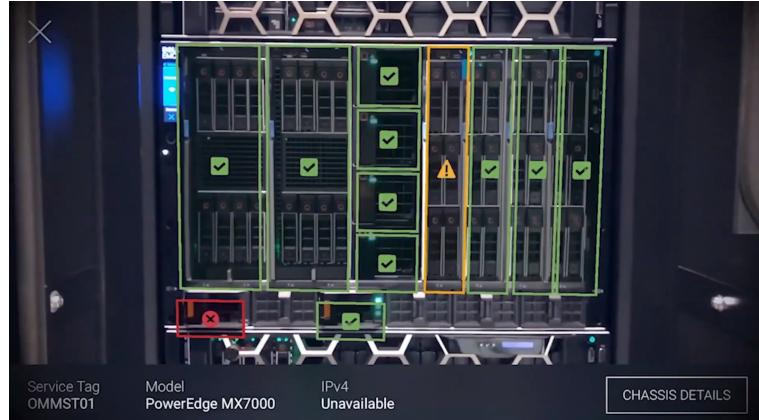
2.3.1 Aplicaciones con funcionalidad similar

Una vez realizada una búsqueda sobre aplicaciones de realidad aumentada que traten de proporcionar una funcionalidad similar a la de este TFG, se puede concluir que hay un gran interés en dar este uso a la realidad aumentada pero las empresas o bien destinan su software al uso interno o bien todavía están en fase de desarrollo. Se llegó a esta conclusión porque solo se encontró una aplicación accesible al público general, **OpenManage Mobile**. El resto de trabajos consisten en demostraciones y artículos que explican las ventajas que ofrece la realidad aumentada en la monitorización de CPDs. A continuación se incluye un resumen del resultado de la búsqueda:

- **OpenManage Mobile** [4] (Figura 2.6²), es una aplicación propietaria de Dell, disponible en iOS y Android, para monitorizar componentes de CPD como los servidores Dell EMC PowerEdge y los chasis MX7000 (Figura 2.6). La realidad aumentada fue incluida en las últimas versiones de la aplicación y por el momento solo funciona con los chasis MX7000. Algunas de sus características son:
 - Proceso de detección. En vez de detectar imágenes o códigos QR, detecta la superficie del chasis. A continuación, coloca un ancla (anchor) sobre esta superficie. El ancla consiste en una estructura que contiene datos sobre la localización de un objeto físico en el mundo real. Por último, se utiliza la información del ancla junto con la información de los sensores para calcular las posiciones de los equipos del chasis.
 - Interfaz y funcionamiento. Hay una ventana inicial con la información general del chasis (Figura 2.6a). Encima de cada dispositivo hardware se coloca un rectángulo y un ícono con diferentes colores según el estado: verde si todo está correcto, amarillo si hay algún warning y rojo si el estado es crítico. Al hacer clic sobre uno de los modelos 3D se muestra una ventana con información sobre el equipo asociado al modelo 3D (Figura 2.6b): nombre del equipo, tipo de hardware, encendido/apagado, warnings, errores y logs.

²Fuente de las capturas: <https://www.youtube.com/watch?v=I5QUW2WUxcQ>

- La versión de Android fue desarrollada con ARCore y la versión iOS con ARKit.



(a) Detalle de la interfaz de la aplicación para la ventana de inicio.



(b) Detalle de la interfaz de la aplicación para mostrar la información de un equipo.

Figura 2.6: Aplicación OpenManage Mobile.

- Nokia **MIKA** (Multi-purpose Intuitive Knowledge Assistant o Asistente de conocimiento intuitivo multiuso) [5] es un asistente virtual destinado a facilitar el trabajo y mejorar la eficiencia de los ingenieros y técnicos de comunicaciones de Nokia. Dos ejemplos de sus funciones son la creación de diagnósticos y la búsqueda del origen de los fallos de red. Inicialmente esta aplicación no usaba realidad aumentada, pero Nokia presentó en el Mobile World Congress 2018 una demostración [6] en la cual un operario utiliza la aplicación sobre un rack para ver el estado de sus equipos.
- La multinacional Stefanini publicó un vídeo demostración de su proyecto **Stefanini AR** donde explican las ventajas y posibilidades de la realidad aumentada en la monito-

rización de los CPDs. En la captura de la Figura 2.7³ se puede observar que el programa muestra un modelo 3D de un rack con un servidor resaltado en rojo porque está causando problemas. Se ve al servidor rojo moviéndose a otra posición del rack indicando así la forma de solucionar el problema.



Figura 2.7: Proyecto Stefanini AR.

2.3.2 Aplicaciones con funcionalidad relacionada

En este apartado se indican las principales aplicaciones que tienen una funcionalidad relacionada con la de este TFG. Es decir, no utilizan la realidad aumentada para monitorizar CPDs pero la usan en tareas relacionadas como por ejemplo el mantenimiento de componentes hardware o la monitorización en sectores como la minería o la alimentación:

- **EcoStruxure Augmented Operator Advisor** (Figura 2.8⁴) es una aplicación comercial de monitorización desarrollada por la multinacional francesa Schneider Electric. Está enfocada a la minería, control de aguas, alimentación e infraestructura pero ofrece la posibilidad de adaptarla a cualquier sector industrial. Algunas de sus características son:
 - Indicar fallos y facilitar su búsqueda. Por ejemplo, si hay un fallo en un armario eléctrico la aplicación indicará dónde está el componente y mostrará un modelo 3D del armario para que el usuario no tenga que abrirlo.
 - Añadir información en forma de notas ligadas a las máquinas para mejorar la comunicación entre técnicos.
- **GE Data Center AR** [7] es una aplicación para iOS desarrollada por Resolution 3D LLC que enseña al usuario, mediante realidad aumentada, los componentes de un CPD y sus características.

³Fuente de la captura: <https://www.youtube.com/watch?v=2Jz6A-zjdgo>

⁴Fuente de la captura: <https://www.youtube.com/watch?v=usWhNIKeio8>

- Senko, una empresa dedicada a la venta de componentes de fibra óptica, ofrece la app **Senko AR** (Figura 2.9⁵) para aprender sobre sus productos mediante modelos 3D y vídeos.
- **Dell EMC AR Assistant** ayuda a los usuarios con el mantenimiento del hardware de Dell. Por ejemplo, en la captura de la Figura 2.10⁶ se puede observar que la aplicación coloca un modelo 3D sobre una pieza para indicar al usuario que debe quitarla. Además, explica la forma de retirar la pieza mediante una animación del modelo 3D.



Figura 2.8: Aplicación EcoStruxure Augmented Operator Advisor.

⁵Fuente de la captura: <https://www.senko.com/ar/>

⁶Fuente de la captura: <https://www.youtube.com/watch?v=RFO6H7AKB8o>



Figura 2.9: Aplicación Senko AR.



Figura 2.10: Aplicación Dell EMC AR Assistant.

Capítulo 3

Herramientas de realidad aumentada

EN este capítulo se analizan las herramientas de desarrollo de realidad aumentada más relevantes de la actualidad. El resultado de este análisis es resumido en la Tabla 3.1 en forma de comparativa entre las principales ventajas y desventajas de cada herramienta. El propósito de los análisis presentados en este capítulo es seleccionar el software que mejor se ajuste a las necesidades del proyecto.

3.1 ARCore

ARCore [8] fue lanzado a principios de 2018 por Google con el objetivo de sustituir a Project Tango, su anterior tecnología de realidad aumentada. Es una de las plataformas de realidad aumentada más recientes, lo que conlleva una mayor cantidad de problemas con respecto a las alternativas que llevan más tiempo en el mercado. De estos problemas, los principales son la escasez de documentación y la abundancia de bugs. Aun así, actualmente ARCore es la mejor opción entre las herramientas de código abierto.

Las plataformas de destino de ARCore conforman una lista limitada de dispositivos iOS y Android. Se ofrecen distintas versiones del SDK para desarrollar en Android Studio, Unity y Unreal Engine, lo cual permite la compatibilidad con los lenguajes de programación Java, C# y C++. A continuación se incluye un resumen de las características de ARCore que serán explicadas en profundidad en el Capítulo 4:

- Puede mostrar gráficos 2D, 3D y vídeos. En la captura de la Figura 3.1¹ se puede observar un ejemplo de una aplicación que muestra y reproduce un vídeo sobre una imagen situada en una pared.

¹Fuente de la imagen: <https://www.youtube.com/watch?v=sX1aOB1wGw>

- Puede detectar cualquier tipo de imagen tanto de forma individual como de forma múltiple.
- Puede detectar superficies.
- Puede detectar caras.
- Las imágenes deben estar situadas en superficies planas.

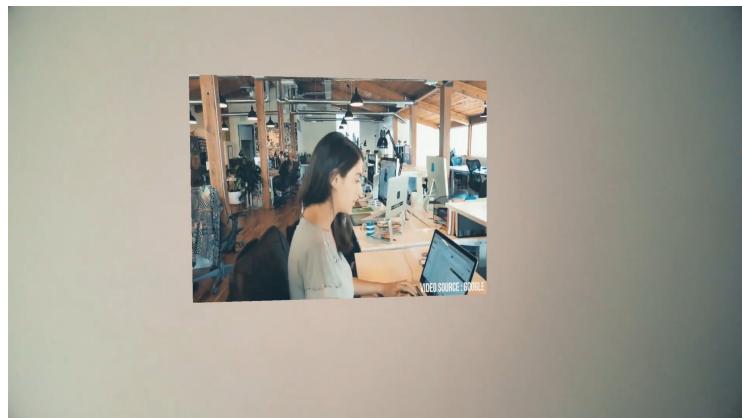


Figura 3.1: Ejemplo sobre ARCore.

3.2 ARKit

ARKit [9] es la plataforma de realidad aumentada de Apple, su lanzamiento sucedió en Junio de 2017. Las distintas versiones de ARKit dependen de la licencia de desarrollador de Apple adquirida, la publicación de aplicaciones requiere la posesión de una de las versiones de pago. ARKit solo es compatible con dispositivos iOS y con respecto al desarrollo, es compatible con el entorno de desarrollo integrado (IDE) Xcode o con los motores de videojuegos Unity y Unreal Engine. Sus principales lenguajes de programación son C#, C++ y Objective-C. Entre las características de ARKit se encuentran:

- Puede mostrar gráficos 2D, 3D y vídeos.
- Puede detectar cualquier tipo de imagen tanto de forma individual como de forma múltiple.
- Puede detectar superficies. En la captura de la Figura 3.2² se puede observar la ejecución de **IKEA Place**, una aplicación creada con ARKit que utiliza la detección de superficies para colocar modelos 3D de muebles.

²Fuente de la imagen: <https://www.youtube.com/watch?v=hRxjeflu9Sg>

- Puede detectar caras, hasta tres simultáneamente.
- Puede detectar objetos.
- Las imágenes deben estar situadas en superficies planas.



Figura 3.2: Ejemplo sobre ARKit.

3.3 Wikitude

La plataforma Wikitude [10] fue lanzada en 2008 por la compañía austriaca Wikitude GmbH, siendo pionera en llevar la realidad aumentada a los smartphones, tablets y smart-glasses. Wikitude es una tecnología de pago que ofrece una versión gratuita con limitaciones como la de no permitir la publicación de aplicaciones. Es compatible con dispositivos iOS y Android. Wikitude ofrece su SDK en forma de APIs y plugins para una gran variedad de plataformas como Unity, Flutter, Xamarin, Titanium y Xcode. Wikitude permite el desarrollo en los lenguajes JavaScript, Java, C#, C++ y Objective-C. Algunas de sus propiedades son:

- Puede mostrar gráficos 2D, 3D y vídeos.
- Puede detectar cualquier tipo de imagen tanto de forma individual como de forma múltiple.
- Puede detectar superficies.
- Puede detectar caras.
- Puede detectar uno o varios objetos. En la captura de la Figura 3.3³ se puede observar un ejemplo en el que se detectan tres objetos.

³Fuente de la imagen: <https://www.youtube.com/watch?v=iak9zBhh0R4>

- Las imágenes pueden estar situadas tanto en superficies planas como en superficies cilíndricas, como por ejemplo una botella o una lata.

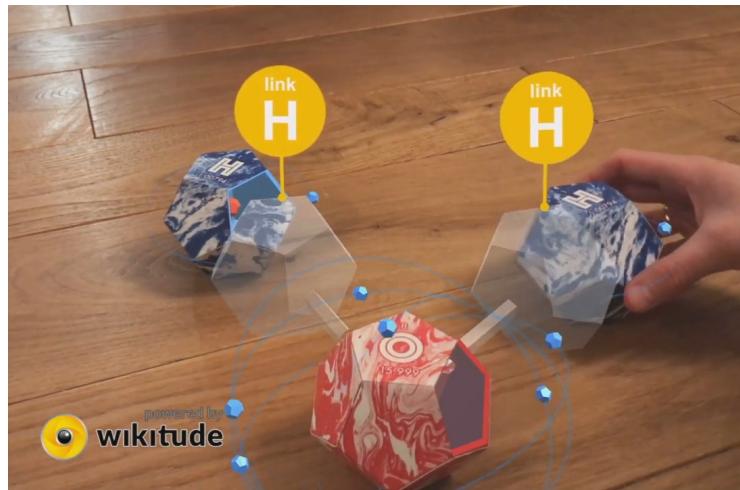


Figura 3.3: Ejemplo sobre Wikitude.

3.4 Vuforia

Vuforia [11] es una plataforma de realidad aumentada creada por la compañía Qualcomm y posteriormente adquirida por PTC. Vuforia posee varias versiones de pago y una gratuita, teniendo esta última versión el gran inconveniente de no permitir al desarrollador la publicación de sus aplicaciones. Vuforia es compatible con una amplia variedad de dispositivos iOS y Android. Ofrece SDKs para el motor de videojuegos Unity y para los IDEs Android Studio y Xcode. Los principales lenguajes de programación usados son C#, Java y C++. Entre sus cualidades destacan:

- Puede mostrar gráficos 2D, 3D y vídeos.
- Puede detectar cualquier tipo de imagen tanto de forma individual como de forma múltiple.
- Puede detectar superficies.
- Puede detectar caras.
- Puede detectar objetos.
- Puede escanear VuMarks, que son una evolución de los códigos QR creada por el equipo de Vuforia.

- Las imágenes pueden estar situadas tanto en superficies planas como en superficies cilíndricas. En la captura de la Figura 3.4⁴ se puede observar un ejemplo en el que se detecta una imagen con forma cilíndrica y se muestra una esfera 3D orbitando a su alrededor.



Figura 3.4: Ejemplo sobre Vuforia.

3.5 AR.js

AR.js [12] es una librería JavaScript de código abierto para el desarrollo de realidad aumentada en web. A diferencia del resto de herramientas que pertenecen a compañías comerciales, detrás de AR.js se encuentra un proyecto de GitHub mantenido por una pequeña comunidad de usuarios. Internamente, el funcionamiento de AR.js se basa en el uso de A-Frame, un framework web para desarrollar experiencias de realidad virtual que a su vez utiliza la librería three.js [13] para la gestión de gráficos, y en el uso de Jsartoolkit5, la versión JavaScript de ARToolKit. En comparación con el resto de herramientas, AR.js tiene una menor potencia gráfica, muchas menos funcionalidades y una menor capacidad de detección. Sin embargo, posee las siguientes ventajas:

- Usabilidad. AR.js tiene una leve curva de aprendizaje y su programación es sencilla. En la Figura 3.5 se puede observar un fragmento de código funcional, en el cual se realizan la importaciones necesarias, se crea la escena de realidad aumentada, se especifica el marcador Hiro (Figura 3.6) como objetivo a detectar y por último se establece una caja amarilla como el modelo 3D a mostrar.
- Portabilidad. Las aplicaciones de AR.js solo necesitan un navegador que soporte WebGL

⁴Fuente de la imagen: <https://gamedevelopment.tutsplus.com/es/tutorials/vuforia-tips-and-tricks-on-unity--cms-28744>

y WebRTC, por lo que no requiere instalación y prácticamente se puede ejecutar en cualquier dispositivo.

- Desarrollo ágil. Gracias a que utiliza código interpretado.

```

1 <!doctype HTML>
2 <html>
3 <!-- Se importa las librerías de a-frame -->
4 <script src="https://aframe.io/releases/1.0.0/aframe.min.js"></script>
5 <script src="https://raw.githack.com/jeromeetienne/AR.js/2.1.4/aframe/build/aframe-ar.js"></script>
6 <!-- Cuerpo del documento html -->
7 <body style='margin: 0px; overflow: hidden;'>
8   <!-- Se crea una escena a-frame -->
9   <a-scene embedded arjs>
10    <!-- Se utilizará el marcador por defecto, que es el marcador "hiro" -->
11    <a-marker preset="hiro">
12      <!-- Cuando se detecte el marcador se mostrará un modelo 3D de una caja amarilla -->
13      <a-box position='0 0.5 0' material='color: yellow;'></a-box>
14    </a-marker>
15    <!-- Se añade la cámara de realidad aumentada -->
16    <a-entity camera></a-entity>
17  </a-scene>
18 </body>
19 </html>
```

Figura 3.5: Ejemplo sencillo de código AR.js.



Figura 3.6: Marcador Hiro.

Como se mencionó anteriormente, la capacidad de detección de AR.js tiene limitaciones. Solamente puede detectar determinadas imágenes llamadas marcadores, del inglés "markers", que se dividen en dos tipos:

- Pattern Markers. Consisten en marcadores personalizados que poseen un aspecto similar al marcador Hiro. Son creados a partir de imágenes escogidas por el usuario, como por ejemplo la de la Figura 3.7a, que mediante un software se transforman y dan como resultado una nueva imagen como la de la Figura 3.7b. Las imágenes escogidas por el usuario están limitadas por estas condiciones:

- Debe tener una forma cuadrada, es decir, la altura y la anchura deben ser iguales.

- Solo puede contener los colores negro y gris claro.
- Su contenido debe ser sencillo como una letra, un número o un símbolo.
- Barcode Markers. Consisten en códigos de barras de dos dimensiones que representan un número. La cantidad de Barcode Markers existentes es limitada y su complejidad es menor a la de los códigos QR. En la imagen de la Figura 3.8 se puede observar un ejemplo de Barcode Marker de tamaño 3x3 que representa el número 5.

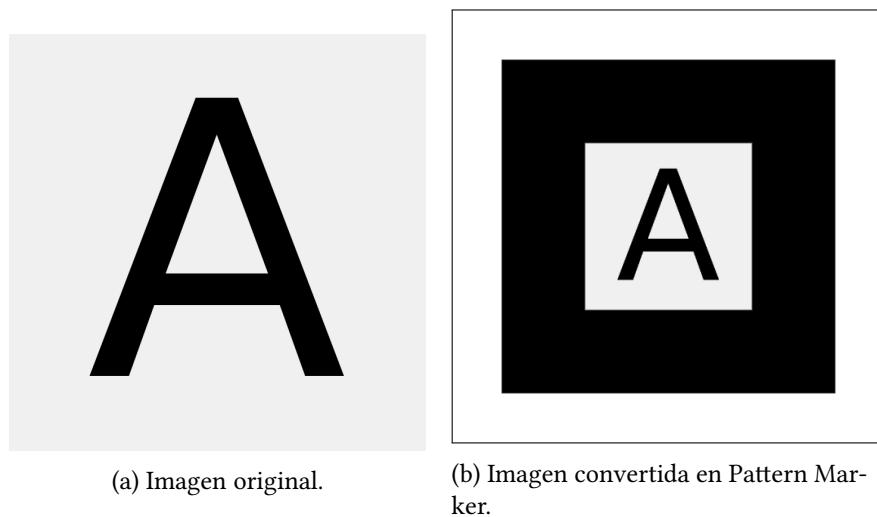


Figura 3.7: Ejemplo de imagen transformada en un Pattern Marker de AR.js.



Figura 3.8: Ejemplo de Barcode Marker de AR.js.

3.6 Comparativa

En esta sección se incluye una tabla en la que se comparan las principales prestaciones de las herramientas de desarrollo de realidad aumentada analizadas en el presente capítulo. Entre estas prestaciones se encuentran el tipo de software (gratuito o de pago), el almacenamiento de las imágenes (local o remoto), plataformas de desarrollo, plataformas de destino, desarrollo compatible con Unity y algunas de las cualidades de la capacidad de detección (tipos de imágenes soportadas, múltiple detección de imágenes, detección de caras, superficies y objetos).

	ARCore	ARKit	Wikitude	Vuforia	AR.js
Precio	Gratis	(1)	Versión gratuita y de pago (2)	Versión gratuita y de pago (2)	Gratis
Almacenamiento de la imágenes	Local	Local	Local o remoto (3)	Local y remoto (4)	Local o remoto (3)
Plataforma de desarrollo	Windows, macOS, GNU/Linux	Windows, macOS	Windows, macOS, GNU/Linux	Windows, macOS, GNU/Linux	Windows, macOS, GNU/Linux
Plataforma de destino	Android, iOS	iOS	Android, iOS	Android, iOS	Android, iOS
Desarrollo en Unity	Si	Si	Si	Si	No
Detecta múltiples imágenes a la vez	Si	Si	Si	Si	No
Tipos de imágenes	Cualquier tipo	Cualquier tipo	Cualquier tipo	Cualquier tipo	Limitado a determinados patrones y códigos de barras
Detección de superficies	Si	Si	Si	Si	No
Detección de caras	Si	Si	Si	Si	No
Detección de objetos	No	Si	Si	Si	No

- (1) Depende del tipo de licencia de desarrollador de Apple. Para poder publicar apps es necesario adquirir una de las licencias de pago.
- (2) Para poder publicar apps es necesario adquirir una de las versiones de pago.
- (3) Permite escoger entre almacenar las imágenes localmente o remotamente.
- (4) Primero se deben subir las imágenes a la web de Vuforia y después descargarlas para almacenarlas en local.

Tabla 3.1: Comparación entre herramientas de desarrollo de realidad aumentada.

Capítulo 4

Desarrollo

En este capítulo se describe el proceso de desarrollo seguido en la creación de la aplicación implementada en este TFG. Así se expondrán las tecnologías empleadas y la metodología aplicada. Dentro de esta metodología se incluirá la descripción de la planificación efectuada y de las fases del proceso software: análisis, diseño, implementación y pruebas.

4.1 Tecnologías empleadas

Esta sección se centra en presentar las tecnologías utilizadas en el proyecto además de exponer su función y los motivos de su elección frente a otras opciones.

4.1.1 ARCore

De entre las herramientas de realidad aumentada comentadas en el capítulo anterior, ARCore [14] ha sido la empleada para la implementación de la aplicación del presente proyecto. Entre los motivos de esta elección se encuentran el ser un software gratuito, el tener un almacenamiento local de las imágenes, la diversa cantidad de posibilidades brindadas por la compatibilidad con los motores Unity y Unreal Engine, unido a las múltiples pruebas realizadas con esta y otras tecnologías como se describirá durante este capítulo.

4.1.2 Unity

De las diferentes versiones de ARCore se ha escogido la versión para Unity [15][16] debido a tener experiencia previa con este motor de videojuegos, a que posee una gran cantidad de documentación, a que tiene una gran comunidad de desarrolladores detrás y a su *Asset Store*, una plataforma en línea para la adquisición y publicación de elementos de videojuego tanto gratuitos como de pago. Un motor de videojuegos es un software que proporciona un conjunto de prestaciones para hacer el desarrollo de videojuegos más sencillo, rápido y eficiente. Estas prestaciones suelen estar conformadas por un motor gráfico, un motor de físicas, un motor

de colisiones, un gestor de audio, animaciones, inteligencia artificial, scripting, etc. Unity fue creado en Dinamarca por Joachim Ante, Nicholas Francis y David Helgason y su primera versión fue lanzada en 2005. Además de ofrecer versiones de pago, Unity tiene una versión gratuita llamada Unity Personal destinada a usuarios individuales, aficionados y empresas pequeñas. Una de las cualidades de Unity más destacables es la capacidad de crear aplicaciones para un gran número de plataformas de diferentes tipos:

- Consolas de videojuegos como la PS4 y la Xbox One.
- Dispositivos móviles para sistemas operativos iOS y Android.
- Sistemas operativos como Microsoft Windows, Apple MacOS y GNU/Linux.
- Navegadores web compatibles con WebGL.
- El dispositivo Apple TV.

Unity utiliza el lenguaje de programación C#, aunque anteriormente también permitía el uso de Boo y UnityScript, una versión propia de JavaScript. Unity ofrece una extensa API [17] para la codificación de videojuegos y, de entre sus clases, la más relevante es *MonoBehaviour*. *MonoBehaviour* es la clase base de la que deriva cada script de Unity y sus métodos son los encargados de la ejecución de la aplicación, siendo los siguientes algunos de los más importantes:

- *Start()*. Se ejecuta en el frame en el que el script es habilitado y antes de que se ejecute cualquier método *Update()* por primera vez.
- *Update()*. Se ejecuta en cada frame.
- *FixedUpdate()*. Similar a *Update()*, pero se utiliza en el caso de que el objeto sea afectado por las físicas.
- *Awake()*. Se ejecuta cuando se carga la instancia del script.

Terminología

La documentación de Unity maneja un conjunto de términos, de los cuales algunos son propios y otros son variantes de conceptos ya existentes, para representar distintos aspectos que componen su estructura. De entre estos términos destacan:

- **Escena**. Las escenas son los contenedores de los elementos del videojuego. En ellas se colocan los objetos, luces, escenarios, etc. Las escenas funcionan como las unidades o

partes en las que se divide un videojuego. Un ejemplo de división podría estar conformado por una escena para el menú principal, otra para un nivel jugable y otra para el menú de fin de la partida.

- **Asset.** Representación de cualquier ítem que se pueda utilizar en un proyecto de Unity. Puede ser tanto interno o externo a Unity y puede ser cualquier tipo de archivo como un modelo 3D, un audio, una imagen, etc.
- **GameObject.** Objetos que representan entidades internas del juego como personajes, accesorios, luces, cámaras, escenarios y efectos especiales. Es el concepto más importante de Unity. No pueden hacer nada por sí mismos. Sirven como contenedores de Components, mediante los cuales obtienen propiedades y funcionalidades.
- **Component.** Elemento que se asocia a los GameObjects para otorgarles propiedades como por ejemplo posición, rotación y escala mediante un *Transform* o funcionalidades mediante un script.
- **Prefab.** GameObject configurado, es decir, con sus propios Components (incluyendo los valores de sus campos) y sus propios GameObject hijos. Incrementa la reutilización gracias a permitir instanciar objetos con las mismas características y gracias a posibilitar la creación de variantes de un mismo objeto.

Interfaz de Unity

La interfaz de Unity está compuesta por varias ventanas que ofrecen diferentes funcionalidades. Estas funcionalidades hacen posible la modificación de algunos aspectos de la aplicación sin la necesidad de que el usuario implemente código, de este modo se incrementa la agilidad y se reduce la complejidad del trabajo. Entre las ventanas más importantes se encuentran [18]:

- **Project Window** (Figura 4.1). Esta ventana permite acceder a la estructura jerárquica de las carpetas que componen el proyecto.
- **Scene View** (Figura 4.2). Esta ventana permite seleccionar y posicionar los GameObjects de la escena dentro del sistema de coordenadas del mundo.
- **Hierarchy Window.** Esta ventana gestiona la jerarquía de la escena, es decir, la relación padre-hijo entre los GameObjects de la escena. Los GameObjects hijos realizan los cambios de su posición, rotación y escala con respecto a su padre en lugar de hacerlo con respecto al sistema de coordenadas del mundo. En la captura de la Figura 4.3 se puede apreciar que los GameObjects *Armature* y *Lowpoly_Notebook* son hijos de *NoteBook*.

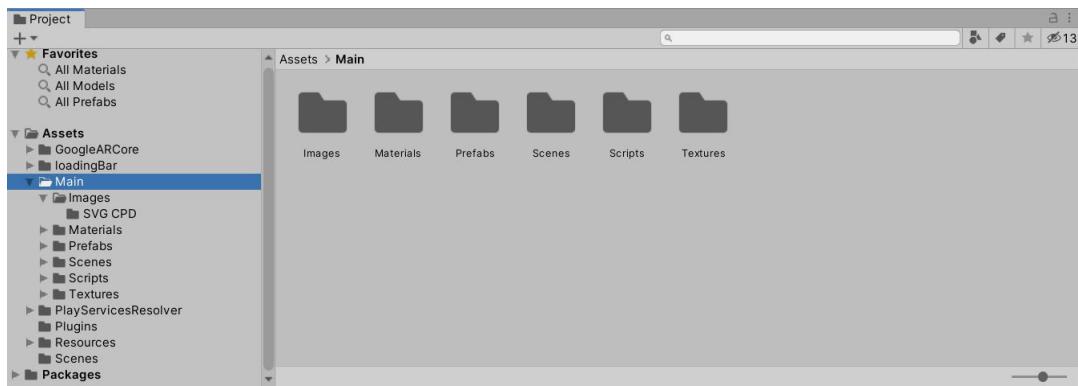


Figura 4.1: Ventana Project de Unity.

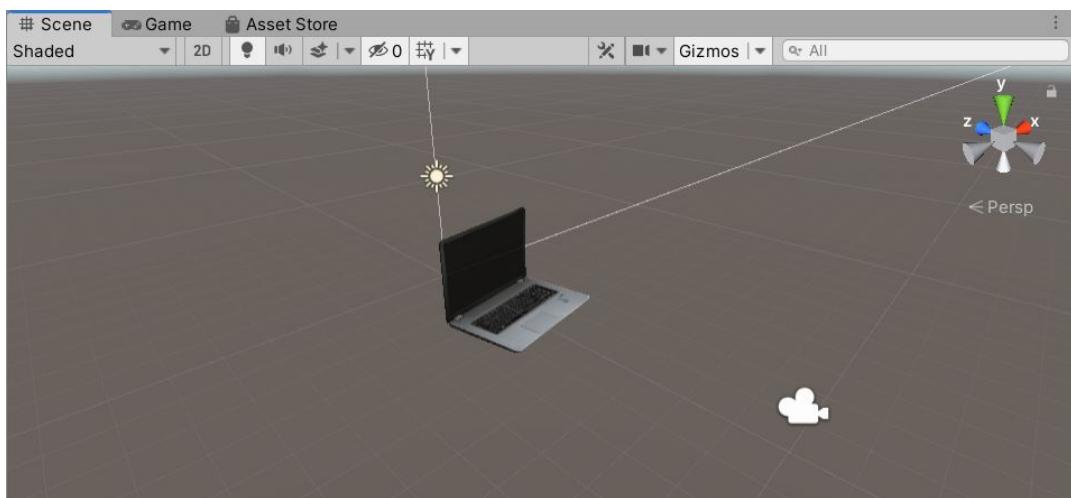


Figura 4.2: Ventana Scene de Unity.



Figura 4.3: Ventana Hierarchy de Unity.

- **Inspector Window.** Esta ventana permite ver y modificar información detallada sobre el GameObject seleccionado sin tener que entrar al código. En esta información se encuentran los Components asignados al GameObject con sus propiedades. En la captura de la Figura 4.4 el GameObject *NoteBook* tiene dos Components, *Transform* (cuyas propiedades son *Position*, *Rotation* y *Scale*) y *Animator* (cuyas propiedades son *Controller*, *Avatar*, *Apply Root Motion*, *Update Mode* y *Culling Mode*). También cabe señalar que la ventana Inspector permite inicializar los atributos públicos de los scripts sin la necesidad de implementar código.
- **Toolbar** (Figura 4.5). Esta ventana es una barra de herramientas. Entre las herramientas que posee se encuentran las que editan las transformadas de los GameObjects, las que cambian la visualización de la escena y las que controlan la ejecución en la ventana Game.
- **Game View** (Figura 4.6). Esta ventana muestra el juego renderizado, es decir, es una representación del juego ya finalizado. Es necesario la presencia de al menos una cámara en la escena para poder tener una visual. También permite ejecutar el juego dentro de Unity para poder probarlo sin la necesidad de instalarlo en el dispositivo de destino.



Figura 4.4: Inspector de Unity.



Figura 4.5: Ventana Toolbar de Unity.



Figura 4.6: Ventana Game de Unity.

4.1.3 Visual Studio

Visual Studio [19] es el entorno de desarrollo integrado (IDE) usado en Unity. Es propiedad de Microsoft y está disponible en Windows y macOS. Además de ofrecer versiones de pago, también ofrece una versión gratuita. Presenta una gran variedad de funciones, siendo las principales el editor de código, el depurador, las herramientas de pruebas y el control de versiones. Permite desarrollar tanto aplicaciones de escritorio, como de web y de dispositivos móviles. A pesar de que Unity utilice el lenguaje C#, Visual Studio también es compatible con otros muchos lenguajes como C++, F#, Visual Basic, PHP, HTML, CSS, JavaScript y Python.

4.1.4 C#

C# [20][21] es un lenguaje de programación multiparadigma y de propósito general desarrollado por Microsoft que forma parte de la plataforma .NET. Entre los paradigmas a los que pertenece se encuentran el estructurado, el imperativo, el orientado a objetos y el dirigido por eventos.

4.1.5 Zabbix

Zabbix [22][23] es un herramienta gratuita y de código abierto para la monitorización de infraestructuras TI tales como redes, servidores, aplicaciones, servicios en la nube, máquinas vitales, etc. Fue creado por Alexei Vladishev y su primera versión fue lanzada en 2001. Es

compatible con una variedad de plataformas entre las que se incluyen Windows, GNU/Linux, Unix y Solaris. Se ha optado por Zabbix por ser la tecnología utilizada en el CPD del CITIC¹, en el que encuentran los servidores del grupo de investigación. El objetivo inicial de este proyecto era realizar una prueba real sobre estos servidores, pero no se pudo utilizar y se optó por un escenario simulado compuesto por máquinas virtuales. La finalidad de Zabbix en el proyecto es monitorizar las máquinas y proporcionar datos de su estado a la aplicación de realidad aumentada. La integración de la aplicación de RA con Zabbix es representada en el diagrama de la Figura 4.7. El servidor Zabbix consiste en un equipo que monitoriza los elementos del CPD (en el caso de un escenario real). La aplicación del proyecto detecta marcadores (códigos QR o imágenes) asociados a los elementos del CPD y obtiene información sobre los mismos a través de peticiones a la API de Zabbix, que puede proporcionar respuestas tanto en formato JSON como en XML. Finalmente, la aplicación interpreta las respuestas y visualiza esta información mediante RA.

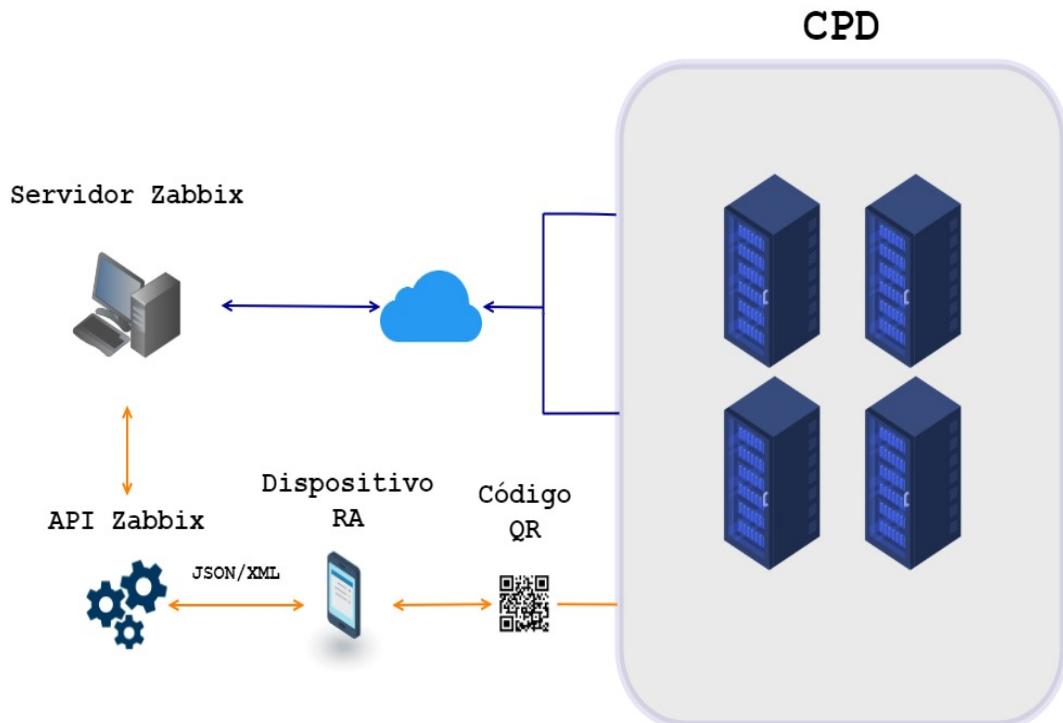


Figura 4.7: Diagrama de la integración de la aplicación de RA con Zabbix.

¹Centro de Investigación en Tecnologías de la Información y las Comunicaciones

4.1.6 VM VirtualBox

VM VirtualBox [24], comúnmente llamado VirtualBox, es un software propiedad de Oracle Corporation para la virtualización de las arquitecturas x86 y AMD64/Intel64. Es gratuito, de código abierto y compatible con Windows, macOS, GNU/Linux y Solaris. VirtualBox tiene la capacidad de virtualizar un amplio conjunto de sistemas operativos, como por ejemplo los propietarios de Microsoft Windows NT 4.0, 2000, XP, Server 2003, Vista, 7, 8, y 10, así como también GNU/Linux, Solaris y OpenSolaris, entre otros. Se ha elegido VirtualBox por su gran número de funcionalidades, por su gran rendimiento y porque ya se posee experiencia con este programa. Su propósito en este proyecto es crear un escenario de pruebas formado por un conjunto de máquinas virtuales que simulan un CPD.

4.1.7 GitHub

Para el control de versiones del proyecto se ha utilizado la plataforma GitHub [25], sistema propietario de Microsoft construido alrededor de la herramienta Git. Los motivos de la elección de GitHub son su versión gratuita y la familiaridad poseída con este software.

4.2 Metodología

En esta sección se lleva a cabo una breve explicación de la metodología aplicada en el proyecto además de exponer las razones que justifican su elección. Durante el proceso de selección de una metodología se debe tener en cuenta la siguiente clasificación [26]:

- Metodologías tradicionales. Fue el primer tipo de metodologías creado. Se caracterizan por tener como base unos requisitos iniciales a partir de los cuales se elabora un plan formado por recursos y tiempos adecuados a dichos requisitos, lo que se denomina gestión predictiva. Este tipo de metodología requiere saber todo lo que quiere el cliente desde el comienzo, lo cual no siempre es posible. No es adecuada para entornos volátiles, los cambios en los requisitos provocan grandes modificaciones y costes que aumentan a medida que se avanza en el proyecto. Otros rasgos de las metodologías tradicionales son la elaboración exhaustiva de documentación y fundamentarse en los procesos. Ejemplos de metodologías tradicionales son la metodología en cascada, RUP (Rational Unified Process) y MSF (Microsoft Solution Framework).
- Metodologías ágiles. Nacieron como respuesta a los problemas presentes en las metodologías tradicionales y sus principios fueron plasmados en el denominado *Manifiesto Ágil* [27], cuyas ideas principales se resumen en:

- Se priorizan los individuos y las interacciones entre ellos frente a las herramientas y los procesos empleados. Las personas son consideradas el factor de éxito del proyecto y se otorga más importancia a la composición del equipo que a la construcción del entorno.
- Se priorizan las respuestas a cambios frente al seguimiento exhaustivo de un plan. La respuesta a cambios es un aspecto esencial y se necesita una planificación flexible.
- Se prioriza la colaboración con el cliente frente a la negociación de los contratos. Se busca conseguir la mayor interacción entre el cliente y el equipo de desarrollo.
- Se prioriza el funcionamiento del producto software frente a la documentación exhaustiva. Se sigue la idea de solamente elaborar la documentación necesaria y siempre de una forma breve y enfocada a los aspectos más fundamentales.

Ejemplos de metodologías ágiles son XP (Extreme Programming), Scrum, Iconix y AUP (Agil Unified Process).

El entorno de trabajo del presente proyecto es cambiante y está cargado de incertidumbre, debido en gran medida a que los límites de las capacidades de las herramientas de realidad aumentada no son claros y no están bien definidos de antemano. Estas condiciones requieren de una metodología adaptable, por lo cual se ha optado por las metodologías ágiles, en concreto Scrum.

4.2.1 Scrum

El término Scrum [28] fue usado por primera vez en el año 1986 por Hirotaka Takeuchi e Ikujiro Nonaka en el artículo "The New New Product Development Game", fijándose en la forma de gestionar los proyectos de algunas empresas tecnológicas que conseguían desarrollar productos de buena calidad con menor tiempo y coste en comparación a empresas que seguían modelos tradicionales. En el artículo se utilizó la palabra Scrum, que literalmente significa melé, como metáfora entre la forma de trabajar de estas empresas y la formación de rugby para resaltar la importancia del trabajo en equipo. En Scrum [29] las necesidades del cliente son ordenadas por prioridad en una lista llamada *Product Backlog*, esta lista es la única fuente que puede provocar cambios en el producto. En Scrum las iteraciones son la base del desarrollo y reciben el nombre de Sprints. El Sprint consiste en un periodo de tiempo de máximo un mes de duración durante el cual se realizan un conjunto de tareas del *Product Backlog*, este conjunto es definido al comienzo del Sprint y se le denomina *Sprint Backlog*. Los objetivos cumplidos en un Sprint sumados al valor de los incrementos de todos los Sprints anteriores conforman el *Increment*. El comienzo de un Sprint se sitúa inmediatamente después de la terminación del

anterior Sprint. Dentro del proceso Scrum existen tres roles comprometidos directamente con el proyecto:

- **Product Owner.** Es el responsable de gestionar el *Product Backlog*, es decir, es el encargado de reflejar, de expresar con claridad y de ordenar las ideas del cliente.
- **Scrum Master.** Es el encargado de promocionar el uso de Scrum. También comprueba que funcione con éxito y se asegura que se cumplen con sus pautas.
- **Development Team.** Consiste en un pequeño equipo de desarrollo auto-organizado responsable de convertir las tareas del *Product Backlog* en incrementos de producto y entregarlos al final de cada Sprint.

Durante el desarrollo de los Sprints se llevan a cabo distintas reuniones de los miembros del proyecto:

- **Planificación del Sprint (Sprint Planning).** Esta reunión se lleva a cabo al inicio de cada Sprint y en ella se establecen los objetivos del Sprint y la forma de alcanzarlos. Tiene una duración máxima de ocho horas.
- **Scrum diario (Daily Scrum).** Esta reunión es realizada diariamente por el *Development Team* y en ella se comentan los avances conseguidos desde la última reunión, se elabora un plan a seguir en el siguiente trabajo y se comentan los problemas surgidos para solucionarlos. Tiene una duración máxima de quince minutos.
- **Revisión del Sprint (Sprint Review).** Esta reunión se lleva a cabo al final de cada Sprint. Se realiza una inspección del producto desarrollado hasta el momento, se presentan los avances alcanzados en el Sprint además de comentar las funcionalidades que no se consiguieron implementar y los problemas surgidos. Tiene una duración máxima de cuatro horas.
- **Retrospectiva del Sprint (Sprint Retrospective).** Esta reunión ocurre después de la Revisión del Sprint y antes de la siguiente Planificación del Sprint. Su finalidad es elaborar una inspección del propio equipo para encontrar puntos débiles y crear un plan de mejoras que serán efectuadas en el próximo Sprint. Tiene una duración máxima de tres horas.

Adaptación de Scrum al proyecto

Las condiciones que atañen al presente proyecto requieren de una adaptación de las características de Scrum. Con respecto a los roles involucrados dentro del proceso Scrum, se procede con la siguiente asignación:

- **Scrum Master.** Este rol se asigna al alumno por ser el principal responsable de asegurar el cumplimiento y supervisión de Scrum.
- **Development Team.** Aunque Scrum requiere de un equipo de 5 a 7 personas, en este caso el equipo solo puede estar integrado por el alumno.
- **Product Owner.** Este rol se asigna a los dos tutores del proyecto porque son los supervisores y orientadores del desarrollo del mismo.

En relación a las propiedades de los Sprints, se establece como duración, de forma aproximada, la máxima permitida, que es de un mes, debido a que Scrum está planteado para compañías que disponen de una mayor cantidad de recursos de los que se disponen y también se tiene en cuenta el tiempo de aprendizaje y de experimentación que se necesita con las herramientas utilizadas. Se llevan a cabo los siguientes ajustes en las reuniones:

- **Planificación del Sprint.** La duración de esta reunión varía en cada Sprint según el tiempo de trabajo estimado para su desarrollo.
- **Scrum diario.** Se omite esta reunión por las limitaciones de tiempo además de no necesitar una comunicación dentro del *Development Team* ya que está compuesto por una sola persona.
- **Revisión del Sprint.** Se establece que dure entre una hora y una hora y media.
- **Retrospectiva del Sprint.** Su duración será inferior a una hora y media y dependerá del transcurso de cada Sprint.

4.3 Planificación

En esta sección se describe la planificación seguida en el proyecto en la cual se ha aplicado la metodología Scrum, que se basa en iteraciones o Sprints donde se cumplen un conjunto de objetivos o tareas. Esta descripción incluye una explicación de cada uno de los Sprints, una exposición de los recursos del proyecto y una estimación de los costes producidos por su desarrollo. En la Figura 4.8 se puede observar un detalle del diagrama de Gantt que muestra los Sprints junto con sus objetivos.

Durante el transcurso del proyecto, la compaginación de su desarrollo con la realización de exámenes y la aparición de los siguientes imprevistos provocaron una demora de tiempo: se planteó que era necesario descartar la tecnología AR.js a causa de las limitaciones de sus capacidades, por lo que se contactó con los creadores de la herramienta y, después de esperar por su respuesta, confirmaron estas limitaciones, así que para poder realizar el cambio de tecnología a ARCore se tuvo que adquirir un dispositivo compatible con la nueva herramienta.

4.3. Planificación

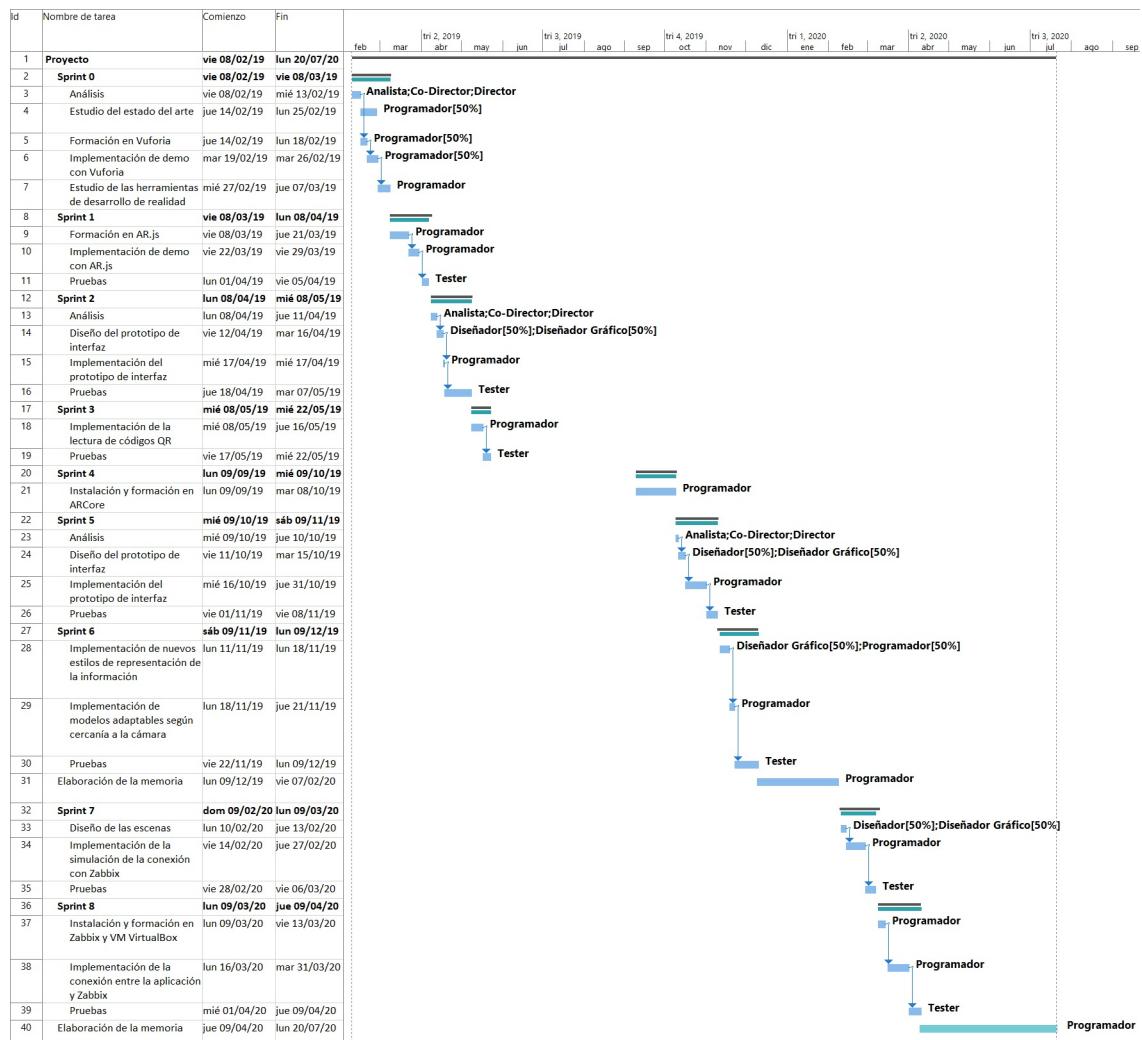


Figura 4.8: Diagrama de Gantt con las tareas del proyecto.

4.3.1 Sprint 0: Establecimiento de las bases del proyecto y búsqueda de tecnología de realidad aumentada

Fecha: 08/02/19 - 08/03/19

El proyecto comenzó con una reunión con los directores en donde se establecieron las pautas que conforman el objetivo principal del TFG, mejorar las capacidades de monitorización de los equipos de un CPD mediante técnicas de realidad aumentada. Estas pautas consisten en:

- Acceso a la información de una forma más inmediata e intuitiva. Esto se consigue con la detección de marcadores, que pueden ser tanto una imagen como un código de barras, asociados a los componentes del CPD. De esta forma, el usuario puede acceder a la información de un equipo desplazándose físicamente a su posición.
- Estudio de los diferentes estilos para representar la información con la finalidad de mejorar la comprensión de los datos. Como por ejemplo las interfaces con texto, gráficos vectoriales y modelos 2D y 3D.
- Análisis de las distintas herramientas de desarrollo de realidad aumentada para seleccionar aquella que más se ajuste a las necesidades del proyecto. Aunque se verá en las siguientes iteraciones, este análisis ha sido descrito en el Capítulo 3.
- Formación del entorno de trabajo complementario a la herramienta de realidad aumentada, del que forma parte Unity, Zabbix, VirtualBox, etc. Esta información puede ser consultada en la Sección 4.1 del presente capítulo.

No se poseía experiencia con la realidad aumentada, salvo unos conceptos teóricos aprendidos en la asignatura "Contornos Inmersivos, Interactivos y de Entretenimiento", así que se decidió crear una demo para realizar una primera toma de contacto con la creación de este tipo de aplicaciones.

La intención era utilizar una tecnología de realidad aumentada compatible con Unity por sus funcionalidades y porque ya se estaba familiarizado con este motor de videojuegos. Se escogió Vuforia por la gran cantidad de documentación que hay sobre el uso conjunto de estas dos herramientas. Siguiendo la documentación oficial, se preparó el entorno de Unity para instalar el plugin de Vuforia. Después de aprender los conceptos básicos de esta herramienta de realidad aumentada se creó una demostración en la cual se detectaba un código QR para a continuación mostrar un modelo 3D sobre la imagen. Durante el proceso se observó que Vuforia no permite un almacenamiento estrictamente local de imágenes, lo que limita el control de recursos por parte del usuario. También se observó que la versión gratuita de Vuforia tiene restringida la publicación de aplicaciones y las versiones de pago tienen un alto precio. Estas dos características nos llevaron a descartar la herramienta.

Por lo tanto, se procedió con la búsqueda de una nueva herramienta. Del resto de alternativas a Vuforia, las más completas son ARKit, Wikitude y ARCore, pero ARKit está dirigido a desarrolladores de Apple, Wikitude es esencialmente un software de pago de alto precio y no se disponía de un dispositivo compatible con ARCore. La búsqueda se centró en el software gratuito y se observó una situación que ocurría con frecuencia, los proyectos encargados del mantenimiento del software carecían de actividad desde hace tiempo. Esta situación también se daba con ARToolKit, pero se le dio una oportunidad por ser una de las herramientas de realidad aumentada de código abierto más importantes por nivel de uso y funcionalidades. En el proceso de instalación surgieron errores que no estaban contemplados en el proyecto GitHub de ARToolKit, se logró resolverlos a excepción de uno, el plugin de ARToolKit empleaba clases obsoletas de la API de Unity y necesitaba de una actualización por parte de sus creadores. Se mostraron los avances realizados a los *Product Owners* y en vista de los problemas encontrados, se optó finalmente por probar la librería AR.js como alternativa para tratar de desarrollar la aplicación.

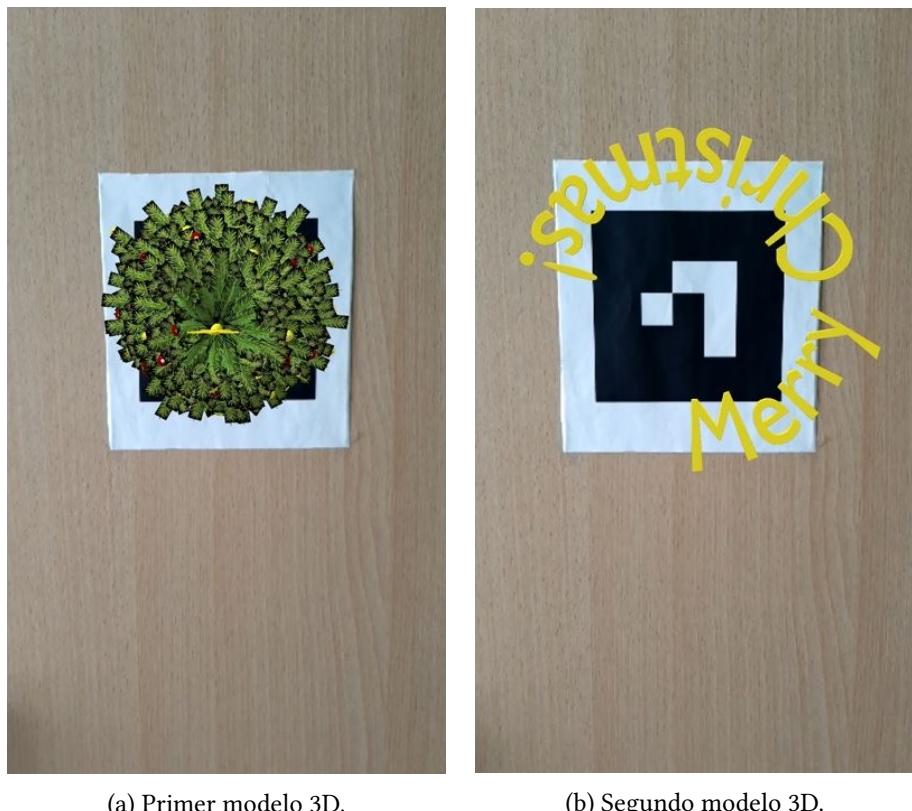
4.3.2 Sprint 1: Primeros pasos con AR.js

Fecha: 08/03/19 - 08/04/19

Antes de comenzar con el desarrollo de la aplicación, fue necesario dedicar tiempo al aprendizaje con AR.js porque era la primera vez que se usaba esta librería y además se necesitaba profundizar en sus capacidades para conocer todo lo que podía aportar al proyecto. Para poder desarrollar con AR.js solo se precisó de un editor de código (en este caso se utilizó inicialmente Sublime Text) y un navegador web actualizado (en este caso se utilizó Mozilla Firefox). Los primeros pasos del aprendizaje consistieron en la lectura de documentación oficial y la ejecución y estudio de un código sencillo. De entre los conocimientos sobre AR.js adquiridos en este proceso de aprendizaje destacan los siguientes (los dos primeros fueron explicados detalladamente en la Sección 3.5 del Capítulo 3-Herramientas de realidad aumentada):

- La estructura de sus programas además de la dinámica que sigue su ejecución.
- Los diferentes tipos de imágenes, llamados marcadores (markers), que soporta. Se aprendió acerca de sus características y limitaciones además del proceso para crearlos.
- La forma de especificar el modelo 3D a mostrar y la forma de importarlos.
- El método para asociar un modelo a un marcador.
- La manera de gestionar los marcadores, que pueden ser almacenados tanto en local como en remoto.

- La gestión de eventos y la interacción con el usuario. Se utilizaron los conocimientos adquiridos hasta el momento para desarrollar un sencillo ejemplo (Figura 4.9) en el cual el usuario podía pulsar un modelo 3D para cambiarlo por otro distinto. El propósito de este ejemplo era estudiar la forma de realizar la transición entre las ventanas de una interfaz gráfica.



(a) Primer modelo 3D.

(b) Segundo modelo 3D.

Figura 4.9: Aplicación AR.js para probar la interacción con el usuario.

4.3.3 Sprint 2: Creación de un primer prototipo de interfaz

Fecha: 08/04/19 - 08/05/19

Una vez terminado el proceso de aprendizaje y como la documentación disponible de AR.js es bastante escasa y no se detallan los límites de sus capacidades gráficas, el siguiente paso consistió en experimentar con la herramienta para saber que interfaces gráficas puede crear.

Se partió de la idea de desarrollar un prototipo de interfaz compuesto por dos ventanas que permitan la navegación entre ellas. El resultado obtenido puede ser observado en las capturas de la Figura 4.10. Las ventanas contienen información sobre los ítems de distintos recursos de una máquina. Cada ventana está formada por un panel con el nombre del recurso, un botón

para realizar la transición a la otra ventana y varios campos con los nombres y los valores de los ítems. La primera ventana (Figura 4.10a) representa la memoria de la máquina y los ítems que muestra son el porcentaje de uso, la cantidad de memoria disponible y la memoria total. La segunda ventana (Figura 4.10b) representa el procesador de la máquina y sus ítems son el porcentaje de uso, la velocidad y el número de procesos.

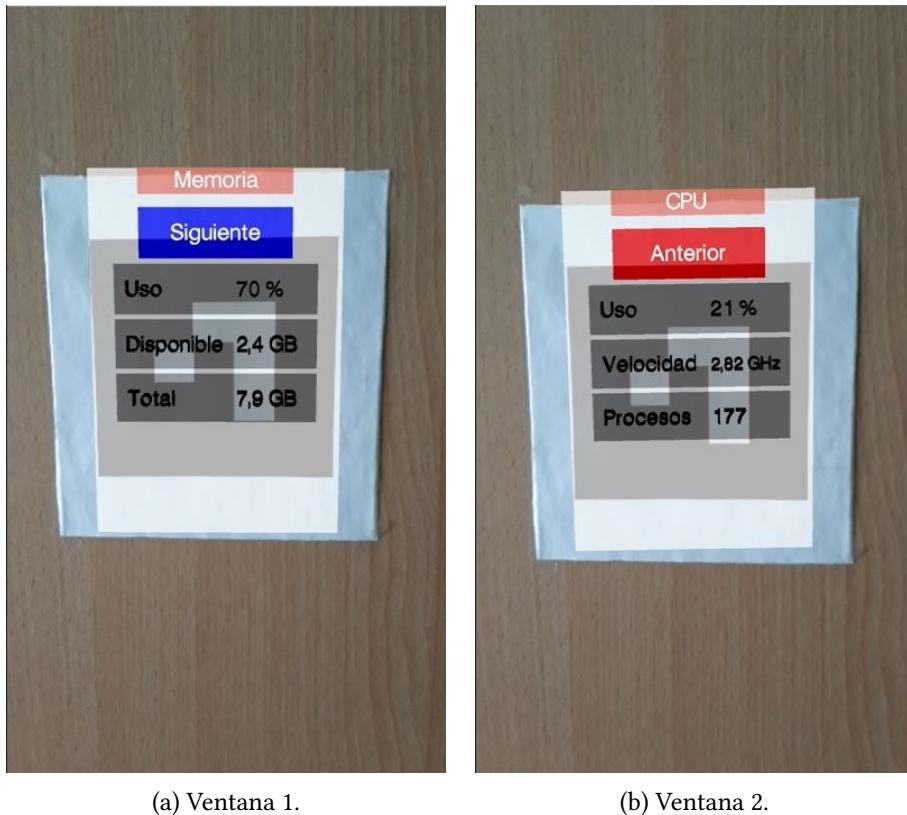


Figura 4.10: Prototipo de interfaz desarrollado con AR.js.

Dentro del proceso de desarrollo de la interfaz se encontraron una serie de dificultades. En el repositorio GitHub oficial de AR.js no existe información sobre cómo implementar interfaces, pero dentro de la comunidad AR.js se encontró una librería, denominada aframe-gui [30], que facilitaba la creación de las mismas. Utilizando esta librería, se implementó la interfaz pero se comprobó que presentaba un error (bug) con el refresco de la imagen, se veían interferencias y la pulsación (clic) sobre los elementos no funcionaba. En el repositorio oficial de aframe-gui no había constancia de estos errores, por lo que fue necesario realizar una serie de pruebas hasta encontrar la solución. Para resolver la falta de detección de las pulsaciones del usuario fue necesario utilizar elementos `<a-entity>` configurados para tener el aspecto y el comportamiento de un botón. Los elementos `<a-entity>` consisten en objetos que se posicionan en la escena de realidad aumentada y obtienen apariencia, comportamiento y funcionalidad

mediante la asignación de componentes. Con la finalidad de encontrar la causa del error del refresco de la imagen, se llevaron a cabo las siguientes pruebas:

- Se realizó una versión gráficamente más sencilla de la interfaz para comprobar si la carga gráfica era el origen del problema. Sin embargo, el error seguía presente en esta nueva versión, por lo que la idea tuvo que ser descartada.
- Se experimentó con el almacenamiento de los recursos (imágenes y librerías), pero se presentaba la misma situación tanto con un almacenamiento local como con uno remoto.
- Se planteó la bajada repentina de los fotogramas por segundos como posible causa del error. Para comprobarlo se agregó un contador que medía la tasa de fotogramas de la aplicación, pero no se observó ninguna anomalía.
- Se comenzó a investigar la forma de sacar el mayor beneficio posible al uso de la memoria caché del navegador con el objetivo de mejorar el rendimiento de la aplicación. Sin embargo, no se logró configurar el uso de la memoria caché del navegador por parte de la aplicación.

Las anteriores pruebas no tuvieron éxito, por lo cual se optó por aplicar la misma solución utilizada en el error de las pulsaciones. De este modo, se implementó una interfaz utilizando solamente elementos `<a-entity>`, sin emplear la librería aframe-gui. Esta nueva versión de la interfaz (Figura 4.10) ya no presentaba interferencias en el refresco de la imagen. Cabe indicar que también se realizaron distintas interfaces de prueba que combinaban elementos `<a-entity>` con elementos de la librería aframe-gui para conseguir el mismo resultado con un código de menor tamaño, pero todas estas interfaces de prueba presentaban el error de las interferencias.

4.3.4 Sprint 3: Implementación de la lectura de códigos QR

Fecha: 08/05/19 - 22/05/19

En esta iteración se estableció como objetivo implementar en la aplicación AR.js la funcionalidad de escanear códigos QR con el propósito de utilizar la información almacenada en estas imágenes para acceder a los datos sobre los activos del CPD de pruebas. De este modo, también se conseguiría una mejora en la capacidad de detección de la herramienta AR.js que, como se explicó en la Sección 3.5 del Capítulo 3-Herramientas de realidad aumentada, posee ciertas limitaciones en el tipo y en la cantidad de imágenes que puede detectar.

Se comenzó con la búsqueda de una librería JavaScript que permitiese la lectura de códigos QR. Se realizaron pruebas con las librerías qr-scanner, jsQR e instascan y se optó por utilizar esta última porque las otras dos presentaban errores en el uso de la cámara. Para realizar la integración de instascan con AR.js se siguió un tutorial, recomendado en el repositorio GitHub

oficial de AR.js, en el que se indica cómo combinar los códigos QR con los marcadores AR.js. Esta combinación consiste en posicionar el marcador AR.js en la zona central del código QR, como se puede observar en la imagen de la Figura 4.11.

Con el nuevo marcador, se buscaba que la ejecución de la aplicación siguiera una nueva dinámica que consistía en: primero escanear el código QR para obtener información sobre el activo del CPD y a continuación detectar el marcador AR.js para mostrar esta información dentro de la interfaz. Sin embargo, se presentó un nuevo problema: como en AR.js el tamaño de los modelos 3D dependen del tamaño real del marcador, el tamaño de la interfaz se vio bastante reducido debido a que el tamaño del marcador AR.js también lo estaba. Para solucionar el problema se intentó buscar la forma de calcular el tamaño real del marcador en tiempo de ejecución y utilizar esta medida para ajustar el tamaño de la interfaz a un valor aceptable. Como en la documentación de AR.js no figuraba información al respecto, se procedió a contactar con los creadores de la herramienta a través del chat oficial de dudas y respondieron que no era posible calcular el tamaño real del marcador. Esta situación provocó que no se pudiera continuar el desarrollo con AR.js.

En la reunión con los *Product Owners*, ante la imposibilidad de seguir con el proyecto con la herramienta AR.js, se decidió cambiar nuevamente de tecnología pasando a ARCore y por lo tanto adquiriendo un dispositivo compatible con esta nueva tecnología, concretamente una tablet del modelo Samsung Galaxy Tab S4.

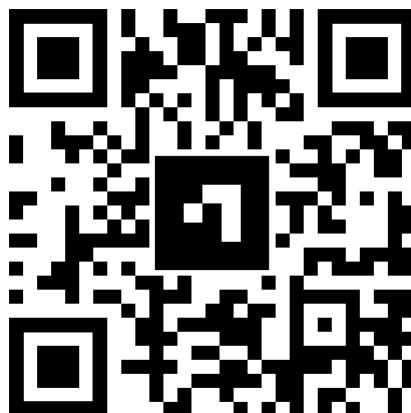


Figura 4.11: Combinación de un marcador AR.js con un código QR.

4.3.5 Sprint 4: Primeros pasos con ARCore

Fecha: 09/09/19 - 09/10/19

El nuevo cambio de AR.js por ARCore brindó la oportunidad de ampliar la exploración de las capacidades de las tecnologías de realidad aumentada disponibles. En esta iteración se

llevó a cabo un proceso de aprendizaje sobre la herramienta ARCore. Este aprendizaje abarcó la preparación del entorno de trabajo, la estructura y funcionamiento del paquete ARCore para Unity y las características principales de la herramienta de realidad aumentada como su capacidad de detección de imágenes y su forma de gestionarlas. Además, se implementaron dos funcionalidades básicas de la aplicación: la asignación única entre las imágenes y los modelos 3D y la ocultación del modelo al parar de detectar la imagen.

Para el primer paso, se tomó como base la guía oficial de inicio rápido (quickstart) de desarrollo para Android en Unity. El quickstart explica cómo realizar la instalación, cómo preparar el entorno de desarrollo y cómo preparar el dispositivo de pruebas. Una vez terminada la instalación, en la ventana Project de Unity se puede observar que los archivos de ARCore se encuentran en la carpeta *Assets/GoogleARCore/*. Se proporcionan tres ejemplos de proyectos:

- *Assets/GoogleARCore/Examples/AugmentedFaces/*: su funcionalidad consiste en detectar caras y colocar modelos 3D sobre ellas (Figura 4.12²).



Figura 4.12: Captura del proyecto Augmented Faces de ARCore.

- *Assets/GoogleARCore/Examples/AugmentedImage/*: su funcionalidad consiste en detectar una serie de imágenes y colocar sobre ellas cuatro modelos 3D de las cuatro esquinas de un marco de cuadro (Figura 4.13³).

²Fuente de la captura: <https://www.youtube.com/watch?v=81g1xKj0Tbs>

³Fuente de la captura: <https://codelabs.developers.google.com/codelabs/augimg-intro/index.html>



Figura 4.13: Captura del proyecto Augmented Image de ARCore.

- *Assets/GoogleARCore/Examples/HelloAR/*: su funcionalidad consiste en detectar superficies planas en el mundo real y colocar sobre ellas modelos 3D del icono de Android (Figura 4.14⁴).

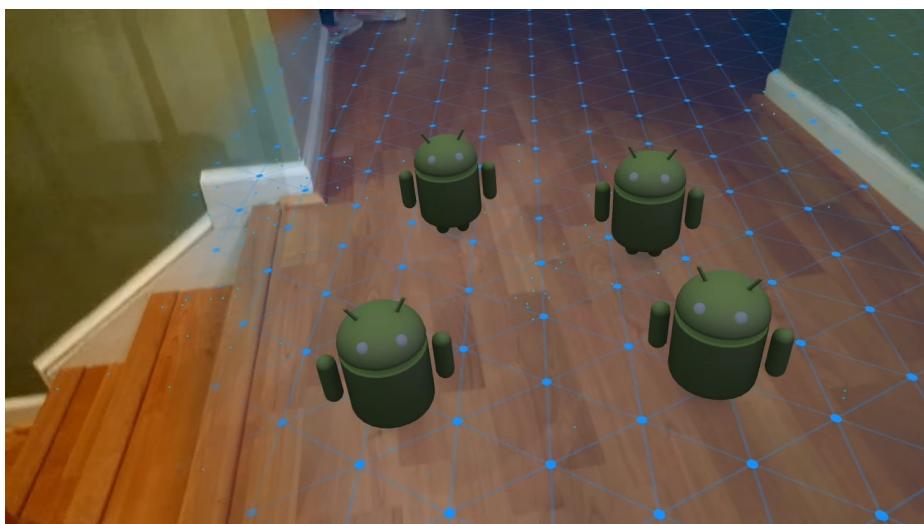


Figura 4.14: Captura del proyecto HelloAR de ARCore.

Como en este proyecto se busca que la aplicación detecte imágenes para acceder y mostrar la información de los activos de un CPD, el desarrollo continuó con la modificación del proyecto AugmentedImage.

⁴Fuente de la captura: <https://www.youtube.com/watch?v=5i0iT1h6JQg>

Estructura y funcionamiento de Augmented Image

En la introducción del presente Sprint se indicó que la funcionalidad de Augmented Image consiste en detectar una serie de imágenes y colocar sobre ellas un modelo 3D. Esta funcionalidad presenta unos comportamientos no deseados, muestra el mismo modelo 3D en todas las imágenes y permanece activado cuando la cámara ha perdido de vista al objetivo. Se busca utilizar distintos modelos para distintas imágenes además de desactivarlos cuando se termine la detección. Antes de proceder con la implementación de estas modificaciones, fue necesario realizar un aprendizaje sobre las características del proyecto Augmented Image como su estructura de directorios y su gestión y especificación de imágenes. En relación a la estructura de directorios, se presenta la siguiente distribución:

- */Configurations*. Contiene el archivo de configuración `AugmentedImagesSessionConfig`, aquí se especifica qué base de datos de imágenes utilizar.
- */Images*. Donde se almacenan las imágenes y sus bases de datos.
- */Prefabs*, */Materials* y */Textures*. Donde se almacenan los prefabs, sus materiales y las texturas de los modelos 3D.
- */Scenes*. Contiene la escena de la imagen de la Figura 4.13.
- */Scripts*. Contiene los dos scripts encargados de la funcionalidad de la aplicación.
 - `AugmentedImageExampleController.cs`: es el script principal, la ejecución de la aplicación se centra en su método `Update()`. Se encarga de inicializar las variables más importantes, de comprobar qué imágenes se están detectando y cuales no y de mostrar un mensaje cuando no se detecta ninguna imagen. En la Figura 4.15 se incluye un fragmento de este código. Como se puede observar, `AugmentedImageExampleController` es una clase hija de `MonoBehaviour` y hereda el método `Update()` de ella. `AugmentedImageExampleController` contiene los siguientes atributos:
 - * *AugmentedImageVisualizerPrefab*. Es el Prefab formado por los modelos 3D de los marcos de cuadro.
 - * *FitToScanOverlay*. Es un elemento de interfaz con el mensaje "Fit the image you're scanning", en castellano "Ajusta la imagen que estás escaneando". Será mostrado cuando no se detecte ninguna imagen.
 - * *m_Visualizers*. Es un diccionario cuyas claves son los índices de las imágenes en la base de datos y cuyos valores son los Prefabs a mostrar. Se van añadiendo elementos al diccionario a medida que se van detectando las imágenes.

- * *m_TempAugmentedImages*. Es una lista que almacena el estado de las imágenes, es decir, si se están detectando o no. Este atributo es actualizado en cada frame.

La implementación del método *Update()* de la clase *AugmentedImageExampleController* tiene el siguiente funcionamiento:

1. Se comprueba si el usuario presionó el botón "atrás" y en el caso afirmativo se termina la ejecución de la aplicación.
 2. Se obtienen los estados en los que se encuentran las imágenes en el frame actual y se almacenan en el atributo *m_TempAugmentedImages*.
 3. Se recorre *m_TempAugmentedImages* comprobando el estado de cada imagen. Cuando una imagen es detectada por primera vez, se calcula su posición en el mundo real gracias a la clase *Anchor* y se crea una instancia del atributo *AugmentedImageVisualizerPrefab*. Cuando se termina la ejecución de la aplicación, se libera la memoria utilizada.
 4. Se activa o se desactiva el atributo *FitToScanOverlay*.
- AugmentedImageVisualizer.cs: este script se asocia al Prefab que contiene los modelos de los marcos de cuadro (Figura 4.13) y se encarga de activarlo cuando se detecta alguna imagen. En la Figura 4.16 se incluye un fragmento de este código. Al igual que la clase *AugmentedImageExampleController*, la clase *AugmentedImageVisualizer* también es hija de *MonoBehaviour* e implementa el método *Update()*. *AugmentedImageVisualizer* contiene los siguientes atributos:

- * *Image*. Es un objeto de la clase *AugmentedImage*. *AugmentedImage* contiene información sobre la imagen real, como su posición y tamaño.
- * *FrameLowerLeft*, *FrameLowerRight*, *FrameUpperLeft* y *FrameUpperRight*. Son los modelos 3D de los marcos de cuadro.

La implementación del método *Update()* de la clase *AugmentedImageVisualizer* sigue la siguiente ejecución: primero comprueba si la imagen no ha sido detectada y en el caso afirmativo desactiva los modelos y se salta a la siguiente iteración del método *Update()*. En el caso contrario activa los modelos y los posiciona en las esquinas de la imagen real.

CAPÍTULO 4. DESARROLLO

```
1 // Se define la clase AugmentedImageExampleController.
2 public class AugmentedImageExampleController : MonoBehaviour
3 {
4     // El Prefab que se coloca sobre las imágenes.
5     public AugmentedImageVisualizer AugmentedImageVisualizerPrefab;
6
7     // Elemento de interfaz con el mensaje "Fit the image you're scanning".
8     // Se muestra si no se detecta ninguna imagen.
9     public GameObject FitToScanOverlay;
10
11    // Diccionario con los pares
12    // <"índice de la imagen en la BBDD", "Prefab a mostrar">.
13    private Dictionary<int, AugmentedImageVisualizer> m_Visualizers
14        = new Dictionary<int, AugmentedImageVisualizer>();
15
16    // Lista auxiliar para almacenar las imágenes actualizadas en cada frame.
17    private List<AugmentedImage> m_TempAugmentedImages = new List<AugmentedImage>();
18
19    // El método Update de Unity se ejecuta cada frame.
20    public void Update()
21    {
22        // Sale de la aplicación si se ha pulsado el botón "atrás".
23        if (Input.GetKeyDown(KeyCode.Escape))
24        {
25            Application.Quit();
26        }
27
28        // Obtiene las imágenes actualizadas en este frame.
29        Session.GetTrackables<AugmentedImage>(
30            m_TempAugmentedImages, TrackableQueryFilter Updated);
31
32        // Este bucle recorre la lista con todas las imágenes
33        // y crea un visualizer y un anchor a las imágenes que
34        // son detectadas por primera vez.
35        foreach (var image in m_TempAugmentedImages)
36        {
37            AugmentedImageVisualizer visualizer = null;
38            m_Visualizers.TryGetValue(image.DatabaseIndex, out visualizer);
39            // Comprueba si la imagen se está detectando y todavía no tiene un visualizer.
40            if (image.TrackingState == TrackingState.Tracking && visualizer == null)
41            {
42                // Crea un anchor para que ARCore siga trackeando esta imagen.
43                // El anchor contiene la posición de la imagen en el mundo real.
44                Anchor anchor = image.CreateAnchor(image.CenterPose);
45                visualizer = (AugmentedImageVisualizer) Instantiate(
46                    AugmentedImageVisualizerPrefab, anchor.transform);
47                visualizer.Image = image;
48                m_Visualizers.Add(image.DatabaseIndex, visualizer);
49            }
50            // Si la ejecución termina, elimina las variables asociadas a las imágenes.
51            else if (image.TrackingState == TrackingState.Stopped && visualizer != null)
52            {
53                m_Visualizers.Remove(image.DatabaseIndex);
54                GameObject.Destroy(visualizer.gameObject);
55            }
56        }
57
58        // Se oculta el mensaje "Fit the image you're scanning"
59        // si se detecta alguna imagen.
60        foreach (var visualizer in m_Visualizers.Values)
61        {
62            if (visualizer.Image.TrackingState == TrackingState.Tracking)
63            {
64                FitToScanOverlay.SetActive(false);
65                return;
66            }
67        }
68
69        // Se muestra el mensaje "Fit the image you're scanning"
70        // si no se detecta ninguna imagen.
71        FitToScanOverlay.SetActive(true);
72    }
73}
```

Figura 4.15: Fragmento del script AugmentedImageExampleController.cs.

```

1 // Se define la clase AugmentedImageVisualizer.
2 public class AugmentedImageVisualizer : MonoBehaviour
3 {
4     // Clase con información de la imagen real.
5     public AugmentedImage Image;
6
7     // Los 4 modelos 3D de los marcos de cuadro.
8     // Esquina inferior izquierda.
9     public GameObject FrameLowerLeft;
10    // Esquina inferior derecha.
11    public GameObject FrameLowerRight;
12    // Esquina superior izquierda.
13    public GameObject FrameUpperLeft;
14    // Esquina superior derecha.
15    public GameObject FrameUpperRight;
16
17    // El método Update de Unity se ejecuta cada frame.
18    public void Update()
19    {
20        // Si la imagen no se está detectando, se desactivan los modelos.
21        if (Image == null || Image.TrackingState != TrackingState.Tracking)
22        {
23            FrameLowerLeft.SetActive(false);
24            FrameLowerRight.SetActive(false);
25            FrameUpperLeft.SetActive(false);
26            FrameUpperRight.SetActive(false);
27            return;
28        }
29
30        // Se colocan los modelos en las esquinas de la imagen real.
31        float halfWidth = Image.ExtentX / 2;
32        float halfHeight = Image.ExtentZ / 2;
33        FrameLowerLeft.transform.localPosition =
34            (halfWidth * Vector3.left) + (halfHeight * Vector3.back);
35        FrameLowerRight.transform.localPosition =
36            (halfWidth * Vector3.right) + (halfHeight * Vector3.back);
37        FrameUpperLeft.transform.localPosition =
38            (halfWidth * Vector3.left) + (halfHeight * Vector3.forward);
39        FrameUpperRight.transform.localPosition =
40            (halfWidth * Vector3.right) + (halfHeight * Vector3.forward);
41
42        // Si la imagen se está detectando, se activan los modelos.
43        FrameLowerLeft.SetActive(true);
44        FrameLowerRight.SetActive(true);
45        FrameUpperLeft.SetActive(true);
46        FrameUpperRight.SetActive(true);
47    }
48 }
```

Figura 4.16: Fragmento del script AugmentedImageVisualizer.cs.

Gestión de las imágenes

Dentro de la gestión de imágenes del proyecto Augmented Image de ARCore se agrupan los distintos factores a tener en cuenta en la selección de imágenes y durante el tiempo de ejecución. Los conocimientos presentados en este apartado sirvieron para profundizar en las características y limitaciones de ARCore, rasgos que repercuten directamente en las capacidades de sus aplicaciones. En relación a la selección de imágenes, ARCore define los siguientes principios:

- Tienen que estar en formato PNG o JPEG.
- Deber tener una resolución mínima de 300 x 300 píxeles.
- Mayor resolución no implica mayor rendimiento.
- Evitar imágenes con pocos puntos característicos.

- Evitar imágenes con patrones repetitivos. Si tenemos una base de datos muy grande, ARCore tendrá dificultades para diferenciar unas imágenes de otras.
- Como se puede observar en la imagen de la Figura 4.17, al crear una base de datos, ARCore asigna a cada imagen una puntuación de calidad sobre 100 según lo buenas que son para ser detectadas y es recomendable usar aquellas que tengan al menos un valor de 75.

También, es importante tener en cuenta que durante la ejecución se deben respetar las siguientes condiciones, que afectan a la imagen real y al uso de la cámara, para lograr un correcto funcionamiento:

- La imagen real debe llenar al menos el 25% del marco de la cámara la primera vez que es detectada.
- La imagen real debe situarse sobre una superficie plana y procurar que no tenga arrugas o colocarla sobre una superficie curva como por ejemplo una botella.
- Evitar que la imagen real esté tapada parcialmente.
- Evitar que la imagen esté ensombrecida.
- Intentar que la luz no incida directamente sobre la imagen por los reflejos que ocasiona.
- La cámara debe situarse de forma perpendicular a la imagen.
- La cámara no se debe mover demasiado.

Especificación de imágenes

Los pasos a seguir para especificar las imágenes que detectará la aplicación comienzan con la creación de una base de datos que se almacenará, junto con las imágenes que la componen, en el directorio *Assets/GoogleARCore/Examples/AugmentedImage/Images/*.

El primer paso consiste en importar las imágenes al proyecto de Unity. Una forma de hacerlo es copiar las imágenes en la ruta *Assets/GoogleARCore/Examples/AugmentedImage/Images/*. A continuación, se crea la base de datos mediante el siguiente procedimiento:

1. Seleccionar las imágenes deseadas. El orden de las imágenes es muy importante porque se emplea para asignar los modelos 3D a las imágenes.
2. Hacer clic derecho y elegir *Create > Google ARCore > AugmentedImageDatabase*. Esto mostrará una subventana, como la que se puede observar en la imagen de la Figura 4.17, que contiene todas las imágenes con su puntuación de calidad correspondiente.

Una vez creada la base de datos, será necesario modificar la configuración de ARCore para que la pueda utilizar. Esto se consigue realizando los siguientes pasos:

1. Abrir el archivo AugmentedImagesSessionConfig almacenado en *Assets/GoogleARCore/Examples/AugmentedImage/Configurations*.
2. Colocar el archivo con la base de datos en el apartado *Augmented Image Database*.

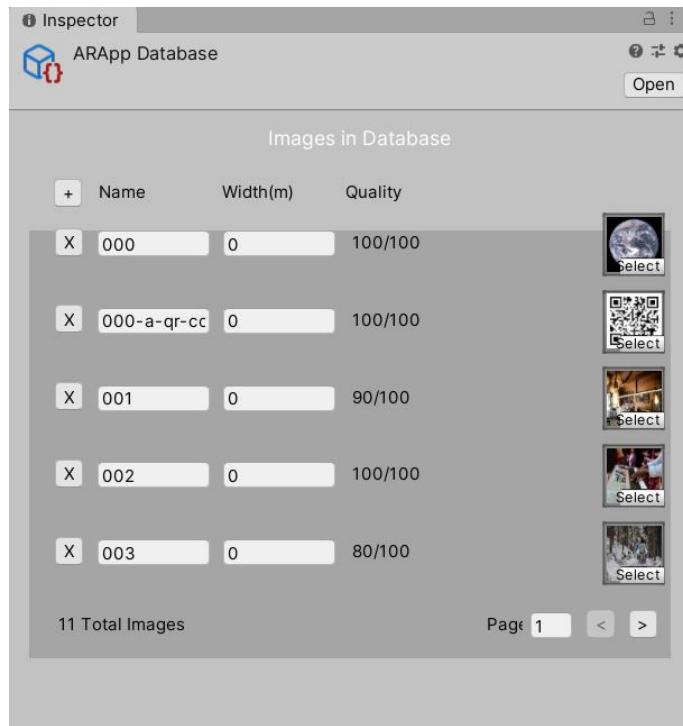


Figura 4.17: Subventana con información sobre una base de datos de imágenes.

Asignar un modelo a cada imagen

La primera modificación en el comportamiento del proyecto Augmented Image en ser implementada fue la asignación de diferentes modelos a diferentes imágenes. Primero, se tuvo que tener en cuenta que en Unity los GameObjects se suelen situar dentro de la escena para ser utilizados, pero esta dinámica cambia en ARCore. En ARCore se crea un Prefab del GameObject y este Prefab es utilizado desde la ventana de Project. Se procedió a crear los Prefabs deseados y posteriormente se asignó el script *AugmentedImageVisualizer.cs* a cada uno de ellos y se realizaron las siguientes modificaciones:

- En el script *AugmentedImageVisualizer.cs* (Figura 4.16) se sustituyeron los atributos de tipo GameObject por una colección de GameObject para poder almacenar todos los

Prefabs. Se tuvo en cuenta el orden de los Prefabs dentro de la colección porque, como se ha explicado anteriormente, es importante ya que el Prefab almacenado en la posición i de la colección se asocia a la imagen almacenada en la posición i de la base de datos.

- En el script AugmentedImageExampleController.cs (Figura 4.15) se convirtió el atributo *AugmentedImageVisualizerPrefab* en una colección con el objetivo de permitir la gestión de múltiples modelos.

Ocultar el modelo al parar de detectar la imagen

La siguiente modificación del comportamiento del proyecto Augmented Image fue desactivar el modelo cuando la imagen se pierde de vista. De entre las posibles soluciones aportadas por la comunidad de ARCore se optó por comprobar el atributo *AugmentedImage.TrackingMethod* de cada imagen. Este atributo puede tomar los siguientes valores:

- *FullTracking*. Cuando la imagen está siendo detectada.
- *LastKnownPose*. Cuando la imagen ha dejado de ser detectada y su seguimiento se realiza basándose en su última posición conocida.
- *NotTracking*. Cuando la imagen no está siendo detectada.

Por lo tanto, en la aplicación se implementó que el estado del modelo se modifique según el valor del atributo *AugmentedImage.TrackingMethod*. El modelo se activa cuando su valor es *FullTracking* y se desactiva cuando es *LastKnownPose* o *NotTracking*. Con este nuevo comportamiento se buscaba mejorar la eficiencia de la aplicación ahorrando al dispositivo el trabajo de procesar gráficos que posiblemente el usuario ya no necesite.

4.3.6 Sprint 5: Creación de un primer prototipo de interfaz

Fecha: 09/10/19 - 09/11/19

En esta iteración se ha implementado un primer prototipo de la interfaz del proyecto cuyo diseño se divide en tres partes, cada una de las cuales se muestra en un estado distinto de la ejecución de la aplicación:

- Cuando no se está detectando ninguna imagen se muestra el mensaje "Fit the image you're scanning", en castellano "Ajusta la imagen que estás escaneando". Este comportamiento ya estaba implementado en el proyecto Augmented Image y se decidió no modificarlo porque es útil para indicar al usuario cómo debe orientar la cámara del dispositivo respecto a la imagen real, lo que facilita la detección de la misma.

- Cuando se está detectando una sola imagen se muestra una ventana (Figura 4.18) compuesta por los siguientes elementos:
 - Un panel con el nombre de la máquina situado en la parte superior de la pantalla.
 - Una serie de botones situados en la parte inferior de la pantalla que permiten la navegación entre las distintas pestañas de la ventana.
 - Cuatro modelos 3D situados en las esquinas de la imagen real. Aunque en esta iteración se han utilizado modelos de prueba, la idea era utilizar un modelo de un rack, dos gráficas circulares que representasen el porcentaje de uso de CPU y de RAM y un modelo que representase la temperatura de la máquina. La activación y la posición de estos modelos varía según la pestaña que está activa.
 - Cuatro pestañas que poseen un panel con su nombre. Cada una de ellas se relaciona con los cuatro modelos 3D de la siguiente forma:
 - * La primera pestaña es la pestaña de inicio y muestra los cuatro modelos.
 - * La segunda, tercera y cuarta pestaña representan la CPU, la RAM y la temperatura respectivamente. Cuando se muestran estas pestañas, su modelo 3D correspondiente es desplazado al centro de la imagen real (Figura 4.18b), con la finalidad de mejorar la visualización del mismo y obtener más información, y los modelos restantes son desactivados.

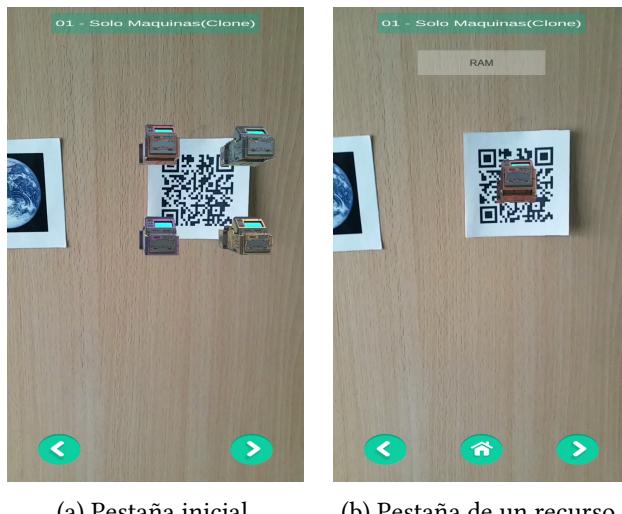


Figura 4.18: Prototipo de interfaz cuando se detecta una sola imagen.

- Cuando se están detectando dos imágenes simultáneamente se muestra una ventana con un texto que pide al usuario que pulse sobre un modelo para obtener más información (Figura 4.19).



Figura 4.19: Prototipo de interfaz cuando se detectan dos imágenes de forma simultánea.

Para conseguir implementar el prototipo de interfaz se han tenido que realizar una serie de tareas no relacionadas de forma directa, entre las que se encuentran:

- Adaptar el proyecto Augmented Image para poder asignar varios modelos a una misma imagen.
- Se buscaba aprovechar lo máximo posible la capacidad de interacción de la realidad aumentada. Por ello, además de utilizar los botones de la interfaz para navegar por las pestañas, se implementó la funcionalidad de poder navegar por la interfaz pulsando sobre los modelos 3D. Por ejemplo, si el usuario pulsa sobre el modelo 3D que representa la RAM de la máquina se mostrará la pestaña asociada a este recurso. Para conseguir esto fue necesario asignar *Colliders* (tipo de componente de Unity que define la forma de un GameObject para los propósitos de colisiones físicas) a los modelos de la aplicación e implementar un script encargado de comprobar las pulsaciones del usuario sobre los objetos identificando la imagen y la pestaña/recurso a las que pertenecen.
- El desplazamiento de los modelos al centro de la imagen real se implementó usando las corutinas de Unity para que se realizara de forma progresiva y no de forma instantánea. De este modo se consiguió un movimiento más estético y natural.
- Para aprovechar las cualidades de la detección de ARCore, además de ampliar los conocimientos sobre la herramienta, en la aplicación se ha implementado la capacidad de gestionar la detección simultánea de dos imágenes. De este modo, el usuario podrá utilizar la aplicación de la siguiente manera:

1. Detecta dos imágenes simultáneamente.
2. Pulsa sobre los modelos de la imagen asociada a la máquina que desea monitorizar.
3. Puede acceder a la información del estado de la anterior máquina a través de una interfaz como la de la Figura 4.20 sin que la detección de la otra imagen interfiera en la interacción.



Figura 4.20: Interacción del usuario con el prototipo de interfaz mientras se detectan dos imágenes simultáneamente.

4.3.7 Sprint 6: Experimentando con los diferentes estilos de representación de la información

Fecha: 09/11/19 - 09/12/19

Una vez alcanzado el objetivo de la iteración anterior de implementar un primer prototipo de interfaz, se estableció como meta para el presente Sprint la mejora de la parte visual de la aplicación. Este proceso de mejora comprende diferentes tareas como la experimentación con nuevos tipos de gráficos (gráficos vectoriales), la búsqueda de nuevos sprites para los elementos de la interfaz y la búsqueda de nuevos modelos para la escena de realidad aumentada. Durante el desarrollo de la iteración se encontró un error en la aplicación con la activación y el desplazamiento de los modelos 3D, los modelos no respondían de la forma esperada a la interacción del usuario. Más adelante se explicará el proceso seguido y las pruebas realizadas para solucionar este error.

Hasta el comienzo de esta iteración, solo se habían utilizado gráficos del formato mapa de bits, es decir, gráficos compuestos por matrices de píxeles. Se planteó experimentar con el otro

tipo de imágenes digitales, los gráficos vectoriales, que a diferencia de los mapas de bits tienen la ventaja de no perder calidad con los cambios de escala, cualidad útil para la adaptabilidad de la aplicación a dispositivos de diferentes resoluciones. Se aprendió a importar y usar los gráficos vectoriales en Unity, cuyo formato compatible es el SVG (Scalable Vector Graphics). Se comprobó que también eran compatibles con la herramienta ARCore y que se les podía asignar un *Collider*, por lo tanto se puede detectar las pulsaciones del usuario sobre estos gráficos. Con este proceso de experimentación se llegó a la conclusión de que los gráficos vectoriales eran una buena opción a utilizar en la aplicación.

Una vez comprobado la compatibilidad de los gráficos vectoriales con la aplicación, se procedió con la mejora de la interfaz. Se probaron diferentes estilos creados con iconos gratuitos obtenidos de diferentes fuentes y se utilizaron tanto sprites como gráficos vectoriales.

En relación a los modelos 3D de la escena de realidad aumentada, se propuso utilizar gráficas circulares para representar el porcentaje de uso de los recursos de la máquina (uso de CPU, uso de memoria, uso de disco, etc). Se buscó en la *Asset Store* de Unity, pero como no se encontró ningún modelo gratuito se decidió buscar modelos desarrollados con la herramienta de modelado Blender. Una vez encontrada la gráfica, se aprendió a importar modelos Blender a Unity y se implementó un script encargado de rellenarla. Cabe señalar, que el uso de estas gráficas causó un error en el refresco de la imagen que fue arreglado configurando las propiedades de dichos modelos.

También surgió la idea de una nueva forma de interacción entre el usuario y los modelos 3D. En la situación de que el usuario esté detectando dos imágenes simultáneamente, se aumentaría el tamaño de los modelos de la imagen más cercana al dispositivo y se reduciría el tamaño de los modelos de la imagen más alejada con el objetivo de otorgar una mayor retroalimentación (feedback) a las acciones del usuario. Para implementar esta funcionalidad se tomó como guía algunas ideas planteadas por la comunidad ARCore. El API de ARCore permite obtener las coordenadas de la posición en el mundo real de la cámara y de la imagen. Estas dos coordenadas se proporcionan en formato de vector y calculando la distancia euclídea entre estos vectores se consiguió obtener la distancia real entre la cámara y la imagen. A continuación, se iba a proceder a calcular la imagen más cercana a la cámara pero se encontró un error que requirió más tiempo del estimado por lo que no se pudo finalizar la implementación de la funcionalidad.

Fue necesario invertir bastante tiempo en la solución del error porque para depurar la aplicación, ARCore solo disponía de la exposición de mensajes del sistema Android a través de la consola y se necesitaba compilar e instalar la aplicación en el dispositivo de pruebas (no fue posible utilizar la ventana Game de Unity para depurar). Antes de conseguir encontrar la causa del problema, se volvió a comprobar que los siguientes ámbitos de la aplicación funcionaban correctamente:

- La detección de las pulsaciones del usuario y el reconocimiento del objeto pulsado.
- El lanzamiento y la detención de las corrutinas encargadas del desplazamiento de los modelos.
- La ejecución de las zonas del código esperadas.

Finalmente, la causa del problema se encontró al comprobar todos los objetos activos en la escena durante la ejecución de la aplicación. Nos percatamos de que se instanciaban múltiples copias no deseadas de los modelos, lo que provocaba situaciones como por ejemplo cuando se desactivaba un modelo, sus copias permanecían activas dando la impresión de que no se producía ningún cambio en la escena. Esta creación múltiple de instancias de un mismo modelo se debía a la forma de gestionar los modelos que venía implementada en el proyecto Augmented Image, como se explicó en el [Sprint 4: Primeros pasos con ARCore](#). Esta gestión consistía en almacenar Prefabs de los modelos en la ventana Project e instanciarlos durante la ejecución de la aplicación. Para solucionarlo, se colocaron todos los modelos en la escena, permaneciendo desactivados hasta que su imagen correspondiente fuese detectada. Además, se implementó un script encargado de su almacenamiento y activación.

4.3.8 Sprint 7: Simulación de la conexión entre la aplicación y el software de monitorización Zabbix

Fecha: 09/02/20 - 09/03/20

Uno de los objetivos del proyecto era utilizar el software de monitorización Zabbix para obtener los datos del estado de los activos de un CPD, pero como en esta fase no era necesario disponer de acceso al Zabbix real se marcó como objetivo realizar una simulación de la conexión de la aplicación con Zabbix mediante la lectura de un archivo XML de prueba, creado por el *Development Team* con la intención de tener una estructura similar a los archivos XML proporcionados por el software de monitorización. Aunque Zabbix también proporciona archivos JSON, se optó por XML por la familiaridad poseída con este lenguaje de marcas.

En el archivo XML de pruebas (Figura 4.21) se definía al CPD como un conjunto de racks, donde cada rack poseía un número de identificación, una descripción general, un conjunto de recursos con sus respectivos ítems y una serie componentes pertenecientes al rack, cada uno de los cuales con sus propios recursos.

Para completar la simulación se desarrollaron tres nuevas escenas, incluyendo sus correspondientes componentes como scripts e interfaces, para implementar funcionalidades de la aplicación no relacionadas con la realidad aumentada:

- **Home.** Consiste en una escena inicial que se muestra al comienzo de la ejecución de la aplicación y que se encarga del inicio de sesión del usuario.

- **Loading.** Consiste en una escena intermedia que siempre se ejecuta en la transición de dos escenas y muestra una animación con el mensaje "Loading" mientras los recursos de la siguiente escena se carga de forma asíncrona, para que la aplicación siga informando al usuario de que se está procesando la escena.
- **Information.** Como en un escenario real donde la conexión con Zabbix proporcionaría una gran cantidad de información, se desarrolló esta escena para complementar a la escena de realidad aumentada en la visualización de estos datos. De este modo, el usuario en la escena de realidad aumentada podía acceder a una información resumida de los elementos del CPD y en esta nueva escena se le proporcionaría información más detallada.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <cpd>
3   <racks>
4     <rack name="Rack">
5       <rackid>118659</rackid>
6       <generaldescription>
7         Armario rack de servidor NetShelter SX de APC 12 U, 600 mm
8         x 900 mm.
9       </generaldescription>
10      <ram>
11        <processes>
12          <process name="Chrome">
13            <usagepercentage>20</usagepercentage>
14          </process>
15        </processes>
16      </ram>
17      <cpu>
18        <processes>
19          <process name="MatLab">
20            <usagepercentage>20</usagepercentage>
21          </process>
22          <process name="VirtualBox">
23            <usagepercentage>10</usagepercentage>
24          </process>
```

Figura 4.21: Detalle del archivo XML para simular la conexión de la aplicación con Zabbix.

4.3.9 Sprint 8: Conexión entre la aplicación y Zabbix sobre un escenario simulado compuesto por máquinas virtuales

Fecha: 09/03/20 - 09/04/20

En esta iteración se implementó la conexión entre la aplicación del proyecto con el API REST de la herramienta de monitorización Zabbix. El objetivo inicial del proyecto era realizar

una prueba real sobre los servidores del CPD del CITIC, pero como no se pudo utilizar el escenario real, la aplicación desarrollada se diseñó para un caso general consistente en un escenario simulado compuesto por máquinas virtuales creadas con el software VirtualBox, de tal modo que la adaptación de la aplicación al entorno real pudiese ser implementada de forma simple. Para alcanzar el objetivo de la iteración se realizaron las siguientes tareas:

- Se instaló Zabbix Appliance [31], una máquina virtual para el software VirtualBox pre-configurada como un servidor Zabbix. Este elemento se encarga de monitorizar y de obtener los datos del estado de los agentes Zabbix. Conectándose al servidor mediante un navegador se puede acceder a la interfaz web de Zabbix (Figura 4.22). El funcionamiento de Zabbix se ha reflejado en el diagrama de la Figura 4.23, este consiste en:
 1. El equipo a monitorizar con un agente Zabbix instalado y configurado para enviar datos al servidor Zabbix. Los agentes pueden ser instalados en multitud de plataformas como Windows, GNU/Linux, Solaris y Unix. El agente no se emplea si el equipo es un componente hardware como routers, sensores de temperatura, impresoras, etc.
 2. El servidor Zabbix, encargado de recolectar datos de los agentes. Este elemento ofrece una interfaz web donde se deben registrar los equipos a monitorizar.
 3. Una vez que el agente ha sido registrado en la interfaz web, este recibe el nombre de *Host*.
 4. Cada *Host* está compuesto por *Items*, estructura que se encarga de recopilar los datos del equipo. En la figura aparecen los *Items* CPU y disco, pero pueden monitorizarse multitud de *Items* tanto relativos a dispositivos hardware como a procesos software.
 5. Los *Keys* son parámetros utilizados por los *Items* que permiten especificar el tipo de información que se solicita al agente Zabbix. En la figura hay dos *Items* y cada uno posee *Keys* diferentes: el *Key* de la parte superior es "Key uso CPU" y sirve para solicitar información sobre el "Porcentaje de uso del CPU del Host" y el *Key* inferior es "Key espacio disco" y tiene como función solicitar información sobre el "Espacio total del disco del Host".
 6. Los *Triggers* son módulos que evalúan los datos recolectados por los *Items* con condiciones definidas por el usuario. Permiten establecer un umbral en el cual los datos son considerados aceptables y si se supera ese umbral, los *Triggers* emitirán un evento o alerta. Por ejemplo, se podría configurar un *Trigger* que enviase una alerta cuando el porcentaje de uso de CPU alcanzase el 100%.
 7. Las alertas lanzadas por los *Triggers* son mostradas de forma gráfica en la interfaz web.

CAPÍTULO 4. DESARROLLO

8. Zabbix también permite enviar las alertas a través de correo electrónico o SMS.

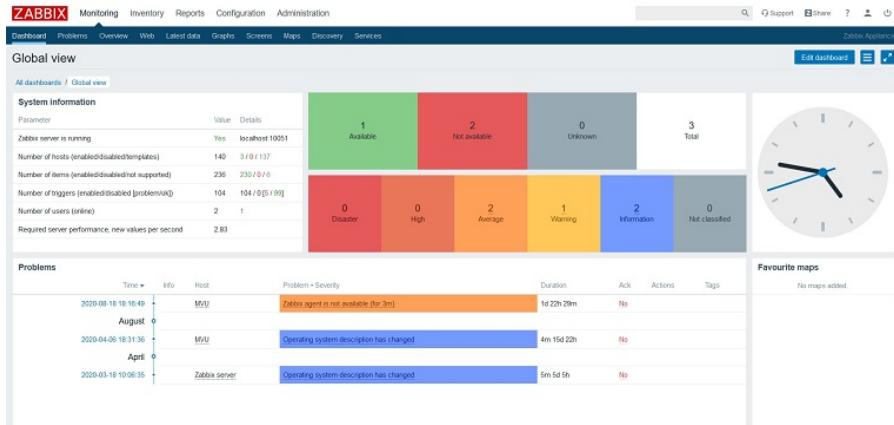


Figura 4.22: Detalle de la interfaz web de Zabbix.

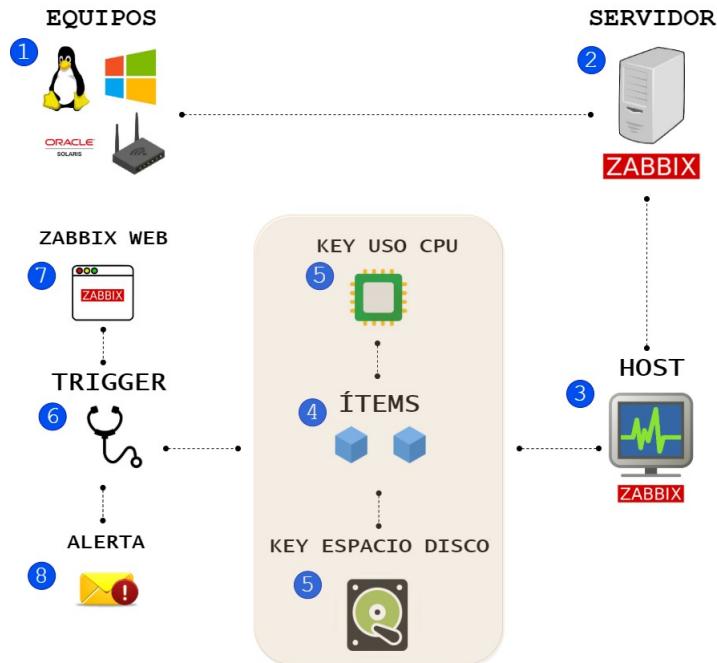


Figura 4.23: Diagrama del funcionamiento de Zabbix.

- Mediante VirtualBox se creó una máquina virtual Ubuntu, se le instaló y configuró un agente VirtualBox para poder ser monitorizada por el servidor Zabbix. Esta máquina simularía un equipo de un rack del CPD real.

- Después de configurar la conexión entre el servidor y el agente, se comprobó que funcionase correctamente y que el servidor recibía información del agente.
- Se aprendió a utilizar la interfaz web de Zabbix incluyendo los datos que es capaz de obtener de los agentes y las funcionalidades que ofrece, entre las que se encuentran: monitorizar tanto en formato texto como en formato gráfico (Figura 4.24), alerta y gestión de problemas, tareas de configuración como el registro de agentes y la agregación de *Items* y permitir la personalización del usuario como por ejemplo modificar la interfaz gráfica, seleccionar la información a obtener y establecer elementos (*Hosts* y gráficas) como favoritos. También se creó un grupo de *Hosts*/máquinas en el cual incluir a la máquina virtual Ubuntu para simular un rack (el grupo) y un equipo (la máquina virtual) contenido dentro de este.

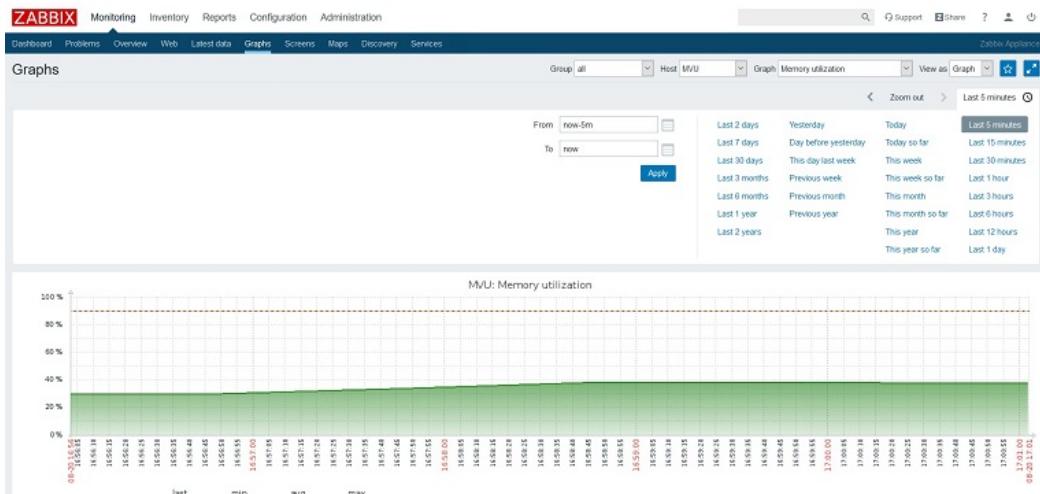


Figura 4.24: Detalle de un gráfico de Zabbix que representa el porcentaje de uso de un recurso.

- Se preparó el servidor Zabbix para que pudiese responder a las peticiones de la API de Zabbix.
- Se estudió cómo son y cómo se realizan las peticiones a la API de Zabbix y qué información ofrece sobre los agentes.
- Para elaborar y lanzar las peticiones a la API de Zabbix de una forma más sencilla para el programador se siguieron las recomendaciones de la documentación oficial de Zabbix [32] y se utilizó una librería para el lenguaje C# desarrollada por la comunidad, en concreto la librería Zabbix.NET [33]. Para poder instalar Zabbix.NET fue necesario instalar el framework Newtonsoft.Json [34].
- Aunque la API de Zabbix puede proporcionar respuestas tanto en formato XML como

JSON, Zabbix.NET solamente admite respuestas en JSON, por lo cual se tuvo que realizar una adaptación en la aplicación de XML a JSON y se aprendió a elaborar y a lanzar las peticiones a la API de Zabbix.

- En la escena **Home** se agregó un nuevo campo de entrada para la dirección IP del servidor Zabbix y el inicio de sesión del usuario se modificó para ser realizado con la API de Zabbix.
- Se modificaron las escenas de realidad aumentada e **Information** para manejar los nuevos datos proporcionados por la API de Zabbix.

Para la realización de las pruebas en esta iteración se empleó una amplia variedad de situaciones para asegurar el correcto funcionamiento de la aplicación en distintas circunstancias que pueden producirse en el entorno real. Estas situaciones se diseñaron para abarcar los distintos estados de accesibilidad de las máquinas y los distintos niveles de severidad de los problemas manejados por Zabbix.

De los avances conseguidos en este Sprint, cabe destacar las siguientes funcionalidades implementadas en la aplicación (en el apartado [Diseño gráfico](#) y en el Apéndice [Guía de usuario](#) se describen con un mayor detalle): conexión con el software de monitorización Zabbix, inicio de sesión para controlar el acceso a la aplicación, visualización mediante RA de la información más significativa de los racks (la cantidad de sus componentes accesibles, no accesibles y con estado desconocido, como se puede observar en la Figura 4.25a), visualización de la información más significativa de los componentes de los racks (la presencia de problemas, su estado de accesibilidad, el porcentaje de uso de los recursos CPU, RAM y disco, como se puede observar en la Figura 4.25b) y mejora e incremento de la interfaz que contiene información más detallada sobre estos equipos, que en el caso de un rack (Figura 4.26a) muestra una descripción general del mismo, una clasificación de sus componentes según su disponibilidad y una lista con los componentes que presentan problemas y en el caso de un componente de rack (Figura 4.26b) muestra una lista con los problemas del componente junto con información general del elemento e información sobre el estado de la RAM, de la CPU, del disco y de la red.

Finalmente, la aplicación fue presentada a los *Product Owners* para su revisión. Después de realizar las adaptaciones necesarias, se consideró que cumplía ampliamente con los requisitos establecidos inicialmente y por lo tanto el producto obtenido es la versión final del proyecto. En esta iteración también se estableció un nombre para identificar a la aplicación: *ARMo* (Augmented Reality Monitoring, en castellano "Monitorización de Realidad Aumentada").

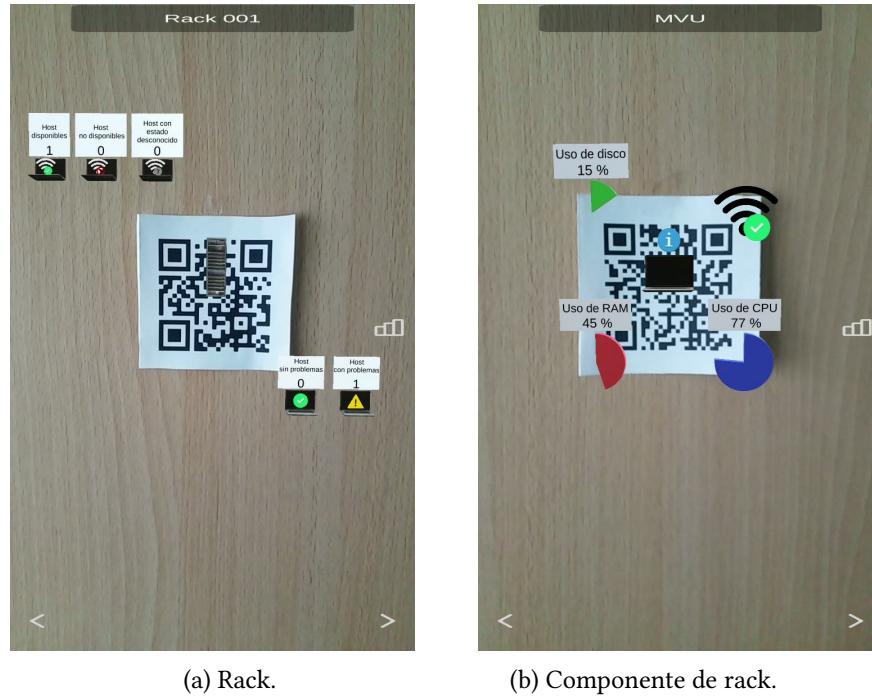


Figura 4.25: Visualización mediante RA de la información de los equipos de un CPD.

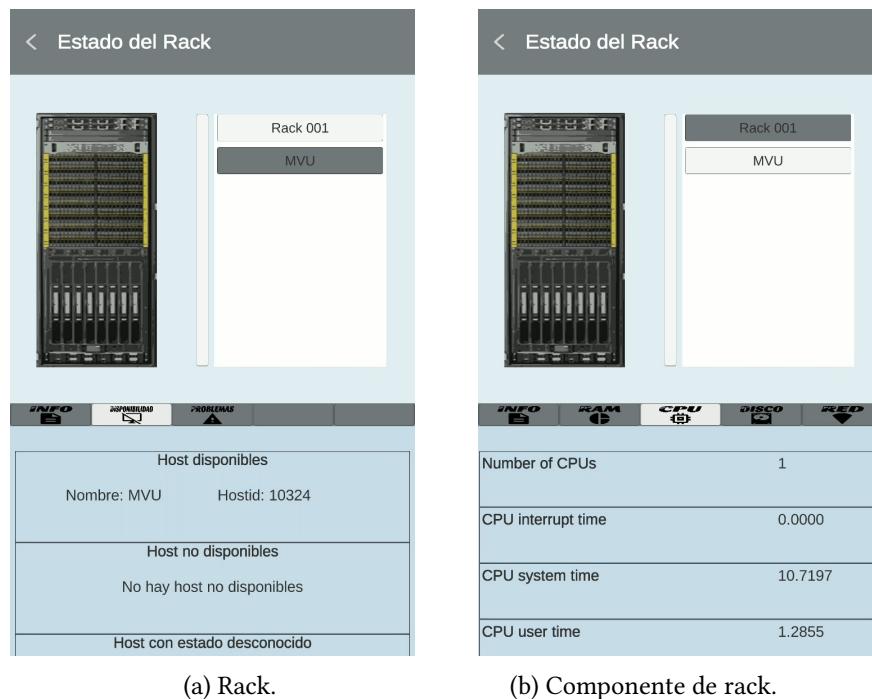


Figura 4.26: Interfaz con información detallada de los equipos de un CPD.

4.4 Recursos y estimación de costes

En esta sección se realiza una descripción de los recursos, tanto materiales como humanos, del proyecto y una estimación de los costes producidos por su desarrollo.

4.4.1 Descripción de los recursos

Los recursos se clasifican en recursos humanos y recursos materiales. Los recursos humanos del proyecto están conformados por los roles Analista, Diseñador, Diseñador Gráfico, Programador y Tester que han sido desempeñados por el alumno. También se disponen de los roles Director y Co-Director, correspondientes a los directores del TFG. Con respecto a los recursos materiales, se necesitó de un ordenador portátil cuyas principales prestaciones son:

- Modelo: MSI GV62-7RD.
- Sistema operativo: Windows 10 Home de 64 bits.
- Procesador: Intel(R) Core(TM) i7-7700HQ (2.80GHz).
- Disco duro: HDD 1TB.
- Memoria RAM: 8G.
- Tarjeta gráfica: NVIDIA GeForce GTX 1050.

Además, se necesitó adquirir un dispositivo móvil compatible con la herramienta de realidad aumentada ARCore para la realización de las pruebas y cuyas principales características son:

- Modelo: Samsung Galaxy Tab S4.
- Sistema operativo: Android.
- Procesador: Qualcomm Snapdragon 835 Octa Core (4 x 2.35 Ghz + 4 x 1.9 Ghz).
- Memoria interna: 64 GB.
- Memoria RAM: 4GB.

En cuanto al software utilizado, se han empleado las siguientes herramientas:

- AR.js.
- ARCore.

- Unity.
- Visual Studio.
- Zabbix.
- VM VirtualBox.
- GitHub.
- MagicDraw.

4.4.2 Estimación de costes

Debido a que todo el software utilizado en el proyecto o bien es gratuito o bien se ha empleado una licencia gratuita, los costes materiales son producidos por el ordenador portátil y la tablet teniendo en cuenta su precio de compra, como se indica en la Tabla 4.1, estimando un total de 1400.00€.

Recursos materiales	Coste Total
Ordenador portátil MSI GV62-7RD	900.00€
Tablet Samsung Galaxy Tab S4	500.00€
AR.js	0.00€
ARCore	0.00€
Unity (Licencia personal)	0.00€
Visual Studio	0.00€
Zabbix	0.00€
VM VirtualBox	0.00€
GitHub (Licencia gratuita)	0.00€
MagicDraw (Licencia de estudiante)	0.00€

Tabla 4.1: Estimación del coste de los recursos materiales del proyecto.

En la estimación de los costes de los recursos humanos se han asignado los salarios respetando los mínimos establecidos en el BOE [35] del *Convenio colectivo del sector de empresas de ingeniería y oficinas de estudios técnicos* y para calcular el coste total de cada recurso se ha usado la fórmula *coste/hora * número de horas dedicadas al proyecto* (dentro de una jornada laboral de 4 horas diarias y de 5 días a la semana). El resultado se ha plasmado en la Tabla 4.2 dando un coste total de 26825.00€.

Por lo tanto, sumando el coste de los recursos materiales con el coste de los recursos humanos, el coste total del proyecto se estima en 28225.00€. Su duración es de 300 días que ascienden a 377 días teniendo en cuenta los retrasos debidos a imprevistos sufridos durante su desarrollo.

Recursos humanos	Coste/Hora	Horas Totales	Coste Total
Analista	30.00€/h	40h	1200.00€
Diseñador	25.00€/h	20h	500.00€
Diseñador Gráfico	15.00€/h	31h	465.00€
Programador	20.00€/h	899h	17980.00€
Tester	20.00€/h	214h	4280.00€
Director	30.00€/h	40h	1200.00€
Co-Director	30.00€/h	40h	1200.00€

Tabla 4.2: Estimación del coste de los recursos humanos del proyecto.

4.5 Análisis

Una vez concluida la explicación de la planificación efectuada en el desarrollo del proyecto se procede a describir las fases del proceso software llevadas a cabo en el mismo comenzando por la fase de análisis. Después de analizar las herramientas de realidad aumentada más relevantes de la actualidad y de las pruebas realizadas se pudieron ampliar los conocimientos sobre sus capacidades permitiendo, de este modo, definir de forma exacta los requisitos que se buscaban alcanzar en el proyecto. Finalmente, los requisitos fueron los siguientes:

- Sistema de inicio de sesión para solo permitir el acceso a los usuarios autorizados.
- Detección de marcadores que pueden consistir en una imagen o un código QR.
- Permitir la visualización de gráficos 2D y 3D sobre los marcadores.
- Permitir la interacción del usuario con los modelos.
- Representar la accesibilidad (estado de la conexión) de un equipo del CPD mediante el uso de gráficos para simplificar y mejorar la comprensión de la información.
- Representar la presencia o ausencia de problemas en un equipo mediante el uso de gráficos que indiquen su nivel de severidad.
- Representar el porcentaje de uso de un recurso del equipo mediante gráficas circulares.
- Acceso al historial de los problemas no resueltos presentes en los equipos.

4.6 Diseño

En esta sección se exponen las decisiones tomadas durante la fase de diseño del desarrollo de la aplicación. La aplicación se dividió en cuatro "escenas", término utilizado en Unity y en

otros motores de videojuegos para referirse a las unidades o partes en las que se divide la aplicación para separarla según distintas funcionalidades o según distintas fases del videojuego. En cada escena se han colocado sus correspondientes objetos, luces, escenarios, scripts, etc. En la Figura 4.27⁵ se puede observar un diagrama de flujo que muestra la transición entre las cuatro escenas. Estas consisten en:

- **Home.** Esta escena es la primera en ser ejecutada y en ella el usuario puede iniciar sesión enviando sus datos al servidor Zabbix. Una vez que el usuario ha iniciado sesión, se procede a ejecutar la escena intermedia **Loading** mientras se carga la escena **AR**.
- **Loading.** Esta escena consiste en una pantalla de carga que siempre se ejecuta durante la transición entre el resto de escenas hasta que se cargan los recursos (modelos, scripts, imágenes, luces, etc.) de la siguiente escena de forma asíncrona. La función de **Loading** es proporcionar retroalimentación al usuario indicándole que se está cargando la siguiente escena.
- **AR (Augmented Reality).** Esta escena tiene gran importancia porque es la encargada de la parte de realidad aumentada de la aplicación y es donde se utiliza la herramienta ARCore. En ella, se muestran los indicadores más significativos de los activos del CPD y se le concede al usuario acceso a información más detallada sobre el activo que está consultando cambiando la aplicación a la escena **Information** (ejecutando previamente la escena **Loading**). Entre las clases más importantes de **AR** se encuentran las siguientes (Figura 4.28):
 - **AugmentedImageExampleController.** Clase de ARCore encargada de comprobar el estado de las imágenes, si se están detectando o no, para indicárselo a las clases responsables de la activación/desactivación de los modelos. Además, se encarga de gestionar las variables necesarias para el correcto funcionamiento de la realidad aumentada. Aunque en esta clase se realizaron modificaciones para ajustarse a las necesidades de la aplicación, en el [Sprint 4: Primeros pasos con ARCore](#) se incluye una explicación más detallada de su funcionamiento general.
 - **CheckingMultitracking.** Clase auxiliar de **AugmentedImageExampleController** para calcular la cantidad de imágenes detectadas simultáneamente.
 - **AugmentedImageVisualizer.** Clase de ARCore que contiene información necesaria para la activación/desactivación y el posicionamiento sobre la imagen real de los GameObjects de la escena.
 - **ModelManager.** Clase encargada de gestionar el almacenamiento, la activación, la posición y el desplazamiento de los GameObjects utilizados de la escena.

⁵Diagrama desarrollado con la herramienta MagicDraw bajo la licencia de estudiante

- **ClickManager.** Clase encargada de detectar las pulsaciones del usuario y de reconocer el objeto pulsado para efectuar la navegación por la interfaz.
- **GUIManager.** Clase encargada de modificar la interfaz gráfica según el estado de la ejecución y la interacción con el usuario. Estos estados, que son modificados por las clases **AugmentedImageExampleController** y **ClickManager**, consisten en:
 - * *NoTracking*. Cuando no se está detectando ninguna imagen.
 - * *SingleTracking*. Cuando se está detectando una sola imagen.
 - * *NormalMultiTracking*. Cuando se están detectando dos imágenes simultáneamente pero el usuario no ha interactuado con ninguno de los modelos.
 - * *SpecialMultitracking*. Cuando se están detectando dos imágenes simultáneamente y el usuario está interactuando con los modelos de una de ellas.
- **Information.** Esta escena consiste en una interfaz con información detallada sobre los activos del CPD. El usuario puede volver a la escena **AR** (ejecutando previamente la escena **Loading**) para volver a utilizar la parte de realidad aumentada de la aplicación.

La conexión con la API de Zabbix se ha implementado mediante la clase **ZabbixImport**. **ZabbixImport** se ha establecido como una clase estática para poder ser utilizada en todas las escenas de la aplicación. Se encarga de preparar y lanzar las peticiones a la API de Zabbix además de deserializar las respuestas JSON en las siguientes clases que representan la estructura del CPD (Figura 4.29):

- **DataCenter.** Representa al propio CPD.
- **Rack.** Representa un rack del CPD. Cada rack se identifica por su *groupid*, posee un nombre (*name*) y una breve descripción (*description*) de sus características hardware.
- **Host.** Representa un componente o elemento de un rack y cada uno es identificado por su *hostid*. De entre sus atributos, se utiliza *available* para representar el estado de la accesibilidad de la máquina. **Host** utiliza la clase **ProblemComparer** que implementa la interfaz *IComparer* para ordenar su lista de problemas según su nivel de severidad.
- **Resource.** Representa un tipo de recurso de un host y está compuesto por un conjunto de ítems, representados con la clase **Item**, cada uno con su identificador (*itemid*), su nombre (*name*) y su valor (*value*). Por ejemplo, algunos de los ítems que componen el recurso *CPU* son *Number of CPUs*, *CPU interrupt time*, *CPU system time* y *CPU user item*.

- **Problem.** Representa un problema de un host. Se identifican por su *eventid* y su nivel de severidad se indica con el atributo *severity*. De menor a mayor, los niveles de severidad de Zabbix son *not classified* (no clasificado), *information* (información), *warning* (advertencia), *average* (promedio), *high* (alto) y *disaster* (desastre).

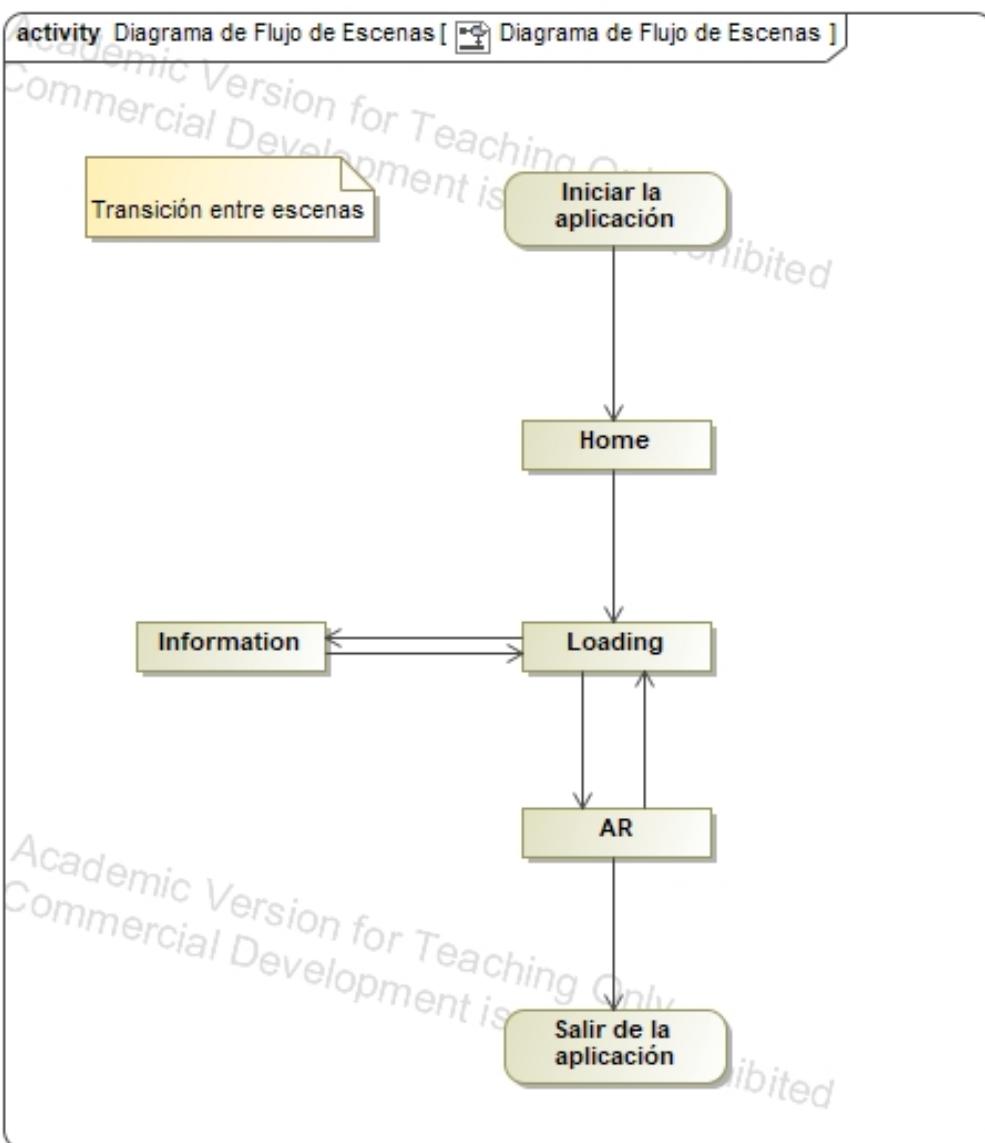


Figura 4.27: Diagrama de flujo entre las escenas de la aplicación.

CAPÍTULO 4. DESARROLLO

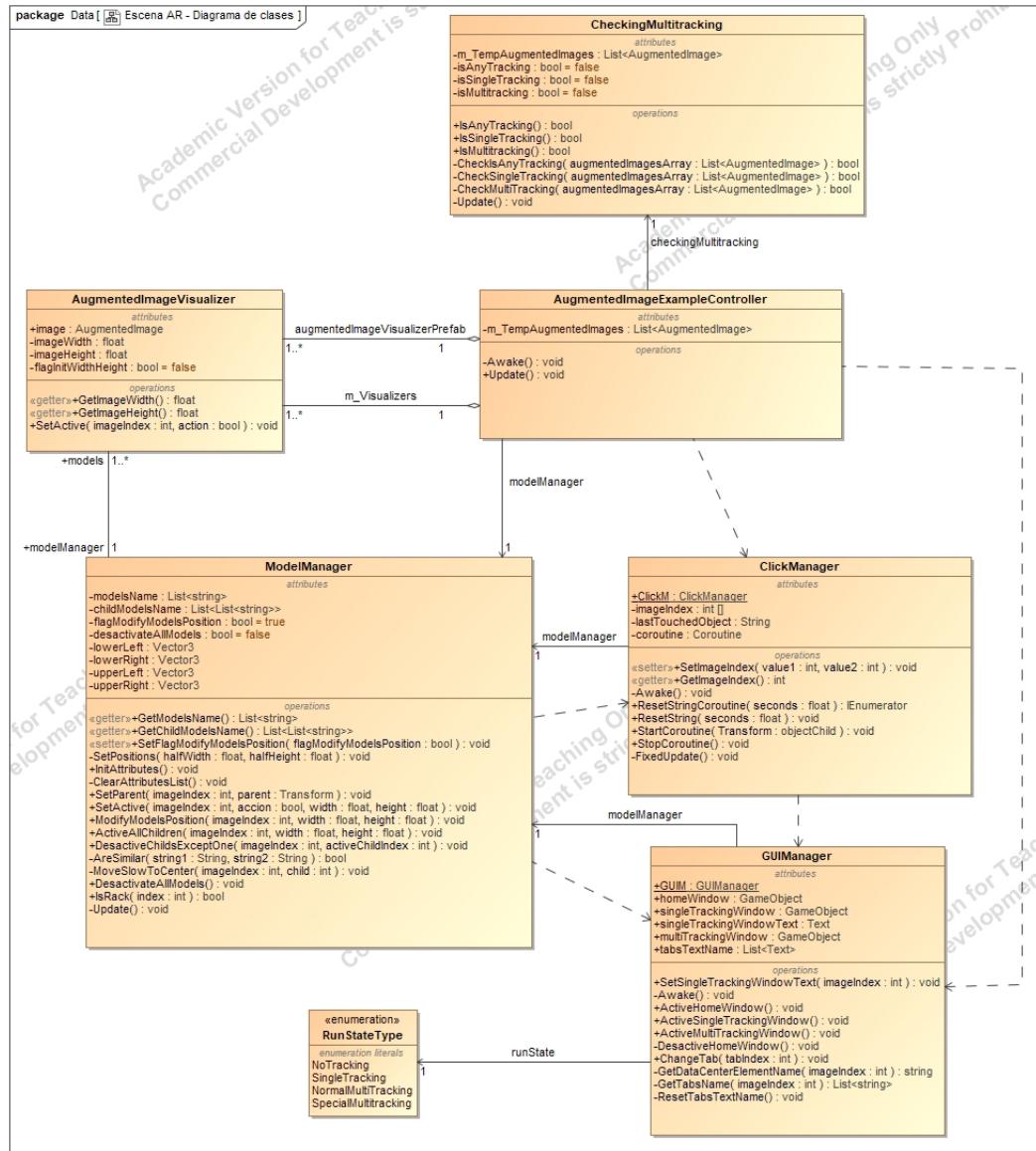


Figura 4.28: Detalle del diagrama de clases de la escena AR.

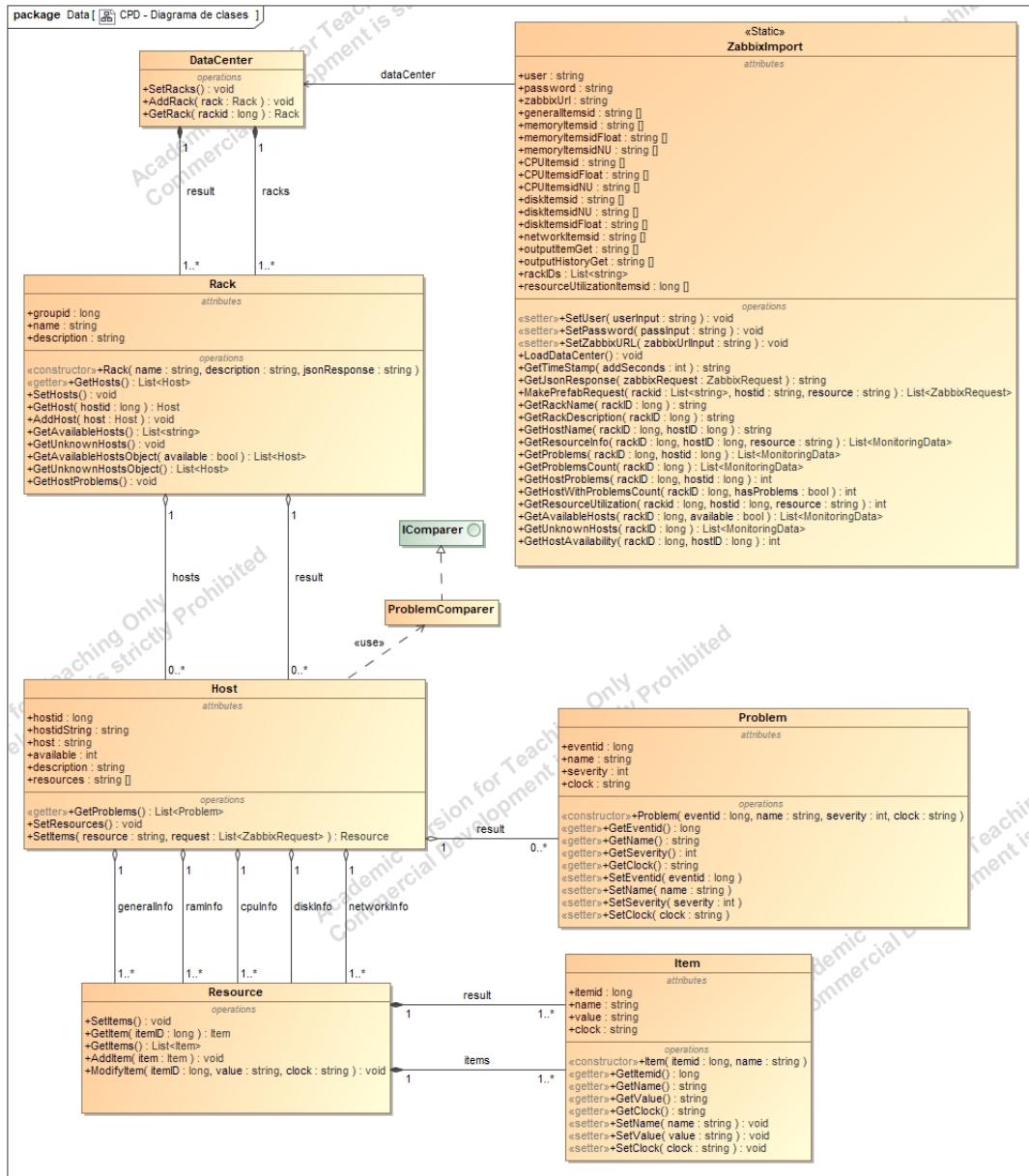


Figura 4.29: Diagrama de clases del CPD.

4.6.1 Diseño gráfico

En una aplicación de realidad aumentada el diseño gráfico adquiere bastante importancia porque la forma de representar la información es un factor determinante para que el usuario pueda asimilar correctamente los datos que se le proporcionan y para mejorar su experiencia con las distintas funcionalidades del software. Por ello, en esta sección se describen las formas de representar la información, mediante interfaces gráficas y modelos que reaccionan a las interacciones con el usuario, utilizadas en las escenas de la aplicación:

- El aspecto gráfico de la escena **Home** consiste en una interfaz con tres campos de entrada (dirección IP del servidor Zabbix, nombre y contraseña del usuario) y un botón para enviar los datos al servidor e iniciar sesión. El aspecto gráfico de la escena **Loading** consiste en una animación con el mensaje "Loading". En la [Guía de usuario](#) del Apéndice A se incluyen las capturas de la Figura A.1 y Figura A.2 de las escenas **Home** y **Loading** respectivamente.



(a) Rack.

(b) Componente de rack.

Figura 4.30: Modelos 3D de los elementos del CPD.



Figura 4.31: Gráficos vectoriales que representan los niveles de severidad de un problema.



(a) Estado desconocido. (b) Estado disponible. (c) Estado no disponible.

Figura 4.32: Gráficos vectoriales utilizados para representar la disponibilidad de los hosts.

- **AR.** En esta escena se combinan elementos de interfaz anclados a la pantalla del dispositivo con modelos 2D (gráficos vectoriales) y 3D posicionados en el mundo real (sobre las imágenes). Los modelos 3D se han utilizado para representar a los racks (Figura 4.30a) y a los componentes de los racks (Figura 4.30b). Estos modelos se posicionan en el centro de sus respectivas imágenes para indicarle al usuario a qué tipo de elemento de CPD está asociada la imagen. El modelo del componente del rack se ha combinado con gráficos vectoriales para indicar la severidad de los problemas que tiene. En la imagen de la (Figura 4.31) se pueden observar los diferentes niveles de severidad ordenados de menor a mayor: el host no tiene problemas, *not classified*, *information*, *warning*, *average*, *high* y *disaster* (para estos tres últimos niveles se ha utilizado el mismo gráfico para simplificar la información presentada al usuario). También se han utilizado gráficos vectoriales para representar el estado de la disponibilidad del host que puede ser: estado desconocido (Figura 4.32a), disponible (Figura 4.32b) y no disponible (Figura 4.32c). Con respecto a la interfaz, esta varía según el estado de la ejecución modificando también la posición y activación de los modelos:

1. Cuando no se está detectando ninguna imagen (Figura 4.33) se muestra el mensaje "Fit the image you're scanning", en castellano "Ajusta la imagen que estás escaneando", junto con cuatro sprites con la forma de las esquinas de una imagen para indicar al usuario cómo debe orientar la cámara del dispositivo con respecto a la imagen real para facilitar la detección de la misma.
2. Cuando se está detectando una sola imagen (Figura 4.34) la interfaz mostrada está formada por los siguientes elementos:
 - Un panel con el nombre del elemento de CPD situado en la parte superior de la pantalla.
 - Un conjunto de pestañas cada una de las cuales se relaciona con los modelos situados sobre la imagen modificando su estado de activación y su posición. Por ejemplo, cuando se activa la pestaña de un recurso el modelo asociado

a ella se desplaza al centro de la imagen para mejorar su visualización y el resto de modelos son desactivados como se puede observar en la captura de la (Figura 4.34b). El número de pestañas varía según si la imagen está asociada a un rack o a un elemento de un rack, pero en ambos casos hay una pestaña inicial (Figura 4.34a) que muestra todos los modelos:

- * En el caso de un rack hay tres pestañas: la inicial, una que indica la cantidad de hosts del rack con estado desconocido, disponible, no disponible y otra pestaña que indica la cantidad de hosts del rack con problemas o que no tienen problemas.
- * En el caso del elemento de un rack hay cinco pestañas: la inicial, disco, memoria, procesador y estado de la red.
- Un subpanel con el nombre de la pestaña. La pestaña inicial (Figura 4.34a) no posee este elemento.
- Una serie de botones situados en la parte inferior de la pantalla que permiten una navegación por las pestañas de forma alternativa a las pulsaciones sobre los modelos.
- Un botón situado en la parte derecha de la pantalla para llevar al usuario a la escena **Information** y mostrarle más información sobre el recurso que está consultando.



Figura 4.33: Interfaz de la escena **AR** cuando no se detecta ninguna imagen.



(a) Pestaña inicial.

(b) Pestaña de un recurso.

Figura 4.34: Interfaz de la escena AR cuando se detecta una sola imagen.

3. Cuando se están detectando dos imágenes simultáneamente se muestra una ventana (Figura 4.35a) con un mensaje que indica al usuario que pulse sobre los modelos de una de las imágenes. Una vez que el usuario ha realizado la pulsación, podrá navegar mediante la interfaz del punto 2 por los modelos de esa imagen como se puede observar en la captura de la Figura 4.35b.

- **Information.** En esta escena se muestra una interfaz (Figura 4.36) compuesta por:

- Un panel con el nombre de la ventana.
- Un gráfico vectorial del rack.
- Una lista de botones con el nombre del rack y sus componentes. Al pulsar sobre estos botones se mostrará la información del rack o componente correspondiente en las pestañas situadas debajo de la lista de botones.
- Un conjunto de pestañas en la parte inferior de la pantalla cuya cantidad de pestañas y tipo de información mostrado presentan dos variantes según el botón pulsado:
 - * Si el botón está asociado a un rack se presentan tres pestañas (Figura 4.36a): una con una descripción general del rack, una que clasifica sus componentes según su disponibilidad y una que indica cuáles de sus componentes tienen problemas.
 - * Si el botón está asociado a un componente de un rack se presentan cinco pestañas (Figura 4.36b): una con la lista de problemas del componente y con

CAPÍTULO 4. DESARROLLO

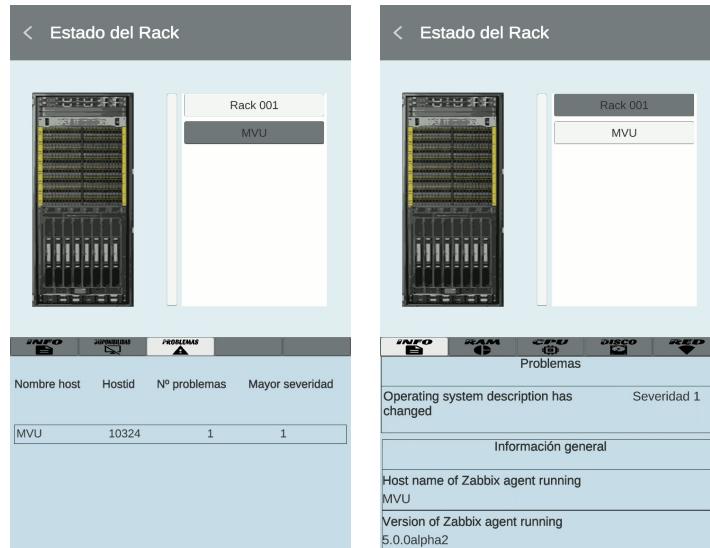
información general del mismo, una dedicada a la memoria, una al procesador, una al disco y una a la red.



(a) Instante previo a la interacción del usuario.

(b) Instante posterior a la interacción del usuario.

Figura 4.35: Interfaz de la escena AR cuando se detectan dos imágenes simultáneamente.



(a) Detalle con la información de un rack.

(b) Detalle con la información de un componente de un rack.

Figura 4.36: Interfaz de la escena **Information**.

4.7 Implementación

En la implementación de la lógica de las escenas se han utilizado los GameObjects de Unity y se les han otorgado propiedades y funcionalidades mediante la asignación de Components, entre los que destacan:

- *Transform*. Para modificar la posición, rotación y escala del GameObject.
- *Rigidbody*. Para permitir que el objeto actúe bajo el control de la física.
- *Collider*. Para definir la forma del GameObject para los propósitos de colisiones físicas y de este modo detectar las pulsaciones del usuario sobre los objetos.
- *Script*. Para otorgar comportamiento al objeto. Los scripts asignados a GameObjects deben derivar de la clase *MonoBehaviour*, de la cual heredan métodos como *Start()*, *Update()*, *FixedUpdate()* y *Awake()* como se mencionó cuando se explicaron las características principales de Unity en la Sección 4.1-Tecnologías empleadas del presente Capítulo.

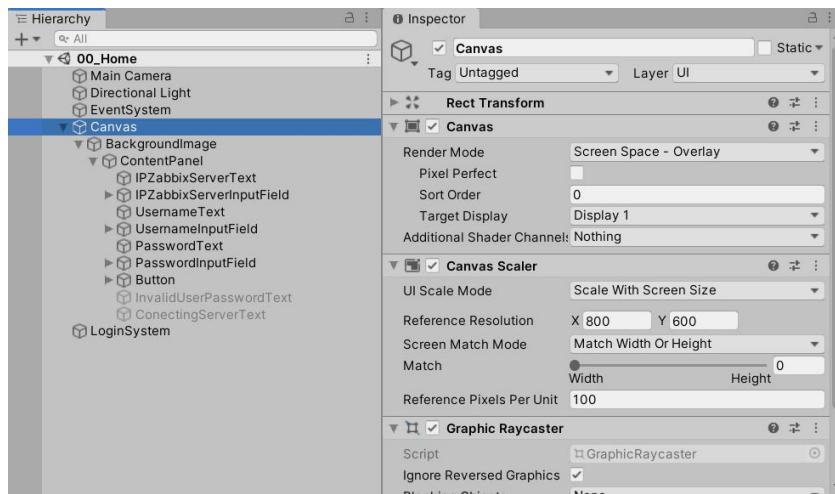


Figura 4.37: Detalle de un objeto Canvas con sus Components e hijos.

Para desarrollar las interfaces gráficas de las escenas se ha empleado el área Canvas de Unity [36]. Este área consiste en un GameObject que se sitúa en la escena y que tiene asignado el Component homónimo Canvas (Figura 4.37). Todos los elementos de interfaz (paneles, botones, campos, barras de scroll, etc.) tuvieron que ser incluidos, mediante la ventana Hierarchy, en las escenas como hijos de Canvas. Gracias a que el área Canvas es mostrado como un rectángulo en la ventana Scene (Figura 4.38) se pudieron posicionar los elementos de interfaz sin recurrir a la ventana Game y se utilizó el Inspector para modificar las propiedades de estos.

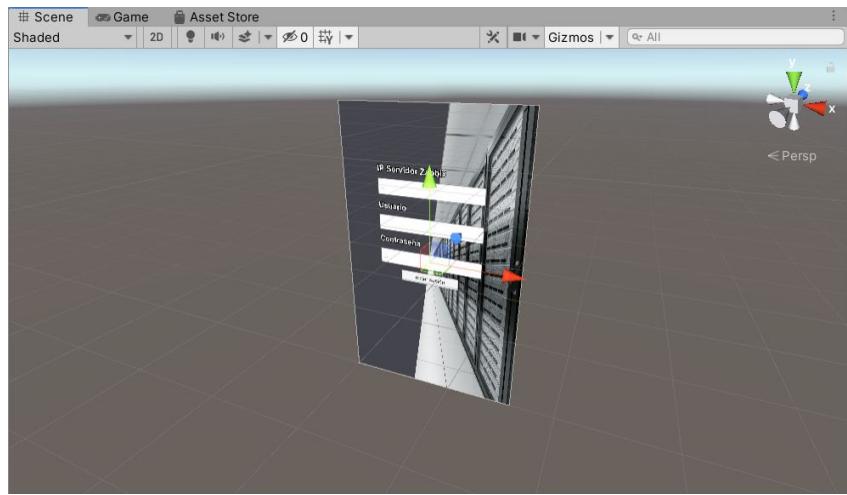


Figura 4.38: Representación del área Canvas en la ventana Scene.

4.8 Pruebas

A continuación, se describen los distintos tipos de pruebas realizadas al final de cada iteración de la metodología Scrum para comprobar el correcto funcionamiento del producto desarrollado. Durante la mayor parte del desarrollo de la aplicación hasta que se estableció la conexión con la API de Zabbix, se han empleado *mockups* en la realización de las pruebas.

4.8.1 Pruebas unitarias

Las pruebas unitarias se realizan sobre componentes individuales (unidades) o módulos de un software con el objetivo de validar que su funcionamiento cumpla con el comportamiento previsto y con el objetivo de comprobar que funcionen correctamente como primer paso antes de comprobar que estos módulos funcionen correctamente en conjunto formando subsistemas. Por lo tanto, se comprobó cada componente de cada GameObject del proyecto, incluyendo todos los métodos de los scripts, de forma individual y se arreglaron los errores encontrados para evitar que estos módulos provocasen nuevos errores en módulos dependientes.

4.8.2 Pruebas de integración

Las pruebas de integración se realizan sobre las unidades combinadas para comprobar que funcionen correctamente en conjunto y detectar fallos en la interacción entre las mismas. Una vez finalizadas las pruebas unitarias, en cada escena se comprobó el funcionamiento de todos sus componentes en conjunto además de comprobar la interacción entre ellos.

4.8.3 Pruebas de sistema

Las pruebas de sistema se realizan sobre el software completo con todos sus componentes integrados para comprobar que se han cumplido con los requisitos establecidos. De este modo, se estuvo ejecutando la aplicación probando las distintas combinaciones posibles y probando la transición entre escenas.

4.8.4 Pruebas de aceptación y validación

Las pruebas de aceptación y validación se realizan sobre el producto finalizado para comprobar si cumple con las expectativas del cliente y si es aceptable para su entrega. Fueron llevadas a cabo por los directores del TFG comprobando que se cumplían con los objetivos iniciales del proyecto y comunicando las adaptaciones que tuvieron que ser aplicadas sobre el mismo.

Capítulo 5

Conclusiones

FINALIZADO el presente Trabajo de Fin de Grado, podemos concluir que el objetivo de este proyecto se ha cumplido satisfactoriamente, es decir, se ha desarrollado una aplicación, que ha sido nombrada como *ARMo*, para dispositivos móviles centrada en la utilización de técnicas de realidad aumentada para integrar y mejorar las funcionalidades de la herramienta de monitorización de servicios de TI con Zabbix. De este modo, *ARMo* proporciona al usuario un acceso más rápido, directo e intuitivo a la información de los activos del CPD (máquinas físicas y virtuales) a los que está observando o que estén situados físicamente cerca mediante la detección de marcadores, que pueden consistir en imágenes o códigos QR.

La aplicación es fácilmente configurable y se integra con un sistema de monitorización Zabbix para obtener la información de los activos bajo supervisión y utiliza un mecanismo de autenticación de usuarios para restringir el acceso a la aplicación solo al personal autorizado y por consiguiente a los datos del CPD.

Entre sus características destaca que la aplicación explota las capacidades de la RA y del motor de videojuegos Unity para ofrecer nuevos estilos de representar la información más visuales, simplificados, esquemáticos e inteligibles: aprovecha el espacio real distribuyendo los modelos sobre distintas zonas de las imágenes, integra las interfaces gráficas con la realidad aumentada para complementar los datos que se proporcionan al usuario, muestra los indicadores más importantes de los equipos mediante modelos 3D de gráficas circulares para representar el uso de recursos (RAM, CPU y disco), combina gráficos 3D de los componentes de los racks con gráficos vectoriales para indicar la disponibilidad del equipo y la presencia de problemas, muestra un resumen del estado de todos los componentes de un rack (número de componentes disponibles y de no disponibles o en estado desconocido, y número de componentes con algún tipo de alerta) para reducir el tiempo de reacción del usuario ante incidentes.

También se consigue aprovechar la interacción con los modelos para ofrecer una forma más natural y directa de navegar por las interfaces gráficas mediante la pulsación sobre los

mismos, permitiendo el acceso a un mayor detalle sobre los indicadores mostrados con RA agregando nuevas interfaces gráficas independientes de esta tecnología.

Durante el desarrollo de este proyecto también se ha realizado un estudio sobre las herramientas RA disponibles en el mercado actual y se ha elaborado una comparativa de las funcionalidades de cada una de ellas. De las conclusiones obtenidas del resultado anterior cabe señalar que se ha apreciado que las grandes corporaciones apuestan por la RA, lo que se puede considerar como muy positivo y que puede ayudar a la mejora futura de esta tecnología en cualquier ámbito de aplicación (ejemplo Google, Apple o incluso Microsoft con sus dispositivos de realidad mixta). En cuanto al estado actual del mercado de herramientas de desarrollo, este está más enfocado a empresas que a desarrolladores particulares. Las herramientas más utilizadas requieren de licencia y su coste general es elevado, como en el caso de Vuforia y Wikitude. En el caso de las herramientas gratuitas, se dio con frecuencia la situación de que los proyectos encargados de este tipo de software carecían de actividad reciente.

De entre el software gratuito, AR.js no presentaba este problema y parecía una opción factible para desarrollar la aplicación de este proyecto, pero con la experiencia obtenida con el uso de la herramienta se llegó a la conclusión de que está pensada para otro tipo aplicaciones mucho más simples, que solo muestran información en pantalla o con un nivel de interacción reducido.

Consideramos a la herramienta ARCore como la opción gratuita más recomendable por sus prestaciones, sin embargo, la tecnología disponible actualmente presenta ciertas limitaciones que dificultan el desarrollo de este tipo de aplicaciones lo que advierte que aún está en proceso de maduración y todavía no se ha consolidado. Por ejemplo, requiere la adquisición de un dispositivo hardware específico que lo soporte, en la API de ARCore se echan en falta algunos métodos importantes que durante el desarrollo del proyecto tuvieron que ser implementados por el *Development Team* como indicar cuándo se está detectando múltiples imágenes simultáneamente e identificar cuales son, la documentación oficial necesitaría una ampliación que incluya soluciones a bugs comunes y una mayor cantidad de tutoriales que actualmente no proporciona y que tuvieron que ser elaborados por la comunidad de desarrolladores. Además, se echa en falta que ofrezcan herramientas para agilizar el proceso de depuración porque ha sido bastante rígido y ha requerido de una gran inversión de tiempo.

Como conclusión final, consideramos que la aplicación ARMo supone un aporte de valor a la herramienta ARCore ya que sirve como demostración de sus capacidades y posibilidades de desarrollo.

Capítulo 6

Líneas futuras

UNA vez terminado el desarrollo del proyecto y cumplidos con los objetivos establecidos, se plantean nuevas funcionalidades y modificaciones que en un futuro puedan ser implementadas en la aplicación *ARMo*:

- Realizar los ajustes y adaptaciones de configuración necesarios para la prueba sobre un escenario real, como es el caso de los servidores del CPD gestionados por el CITIC. *ARMo* ha sido diseñada para un caso general para que pueda ser adaptada al escenario real de forma sencilla.
- Mejora de aspectos gráficos de la aplicación facilitando la parametrización de la interfaz (multiidioma, personalización de colores e iconos, añadir nuevos modelos de objetos 2D/3D, etc.).
- Añadir funcionalidades para ayudar al usuario a orientarse por el CPD, mediante la visualización por RA de información guiada de la ruta a seguir hasta el armario/activo que se está buscando y utilizando técnicas de geolocalización en interiores y planos del entorno.

Apéndices

Apéndice A

Guía de usuario

En el presente apéndice se incluye una breve guía de uso de la aplicación *ARMo* (*Augmented Reality Monitoring*).



GUÍA DE USUARIO
ARMo (Augmented Reality Monitoring)
Versión 1.0

Al iniciar la aplicación *ARMo* se muestra una ventana de inicio como la de la Figura A.1 con tres campos de entrada donde debe introducir la dirección IP del servidor Zabbix, su nombre de usuario y su contraseña para iniciar sesión y poder acceder a las funcionalidades de la aplicación.



Figura A.1: Ventana de inicio.

Cuando haya iniciado sesión se mostrará una pantalla de carga como la de la Figura A.2 y si es la primera vez que utiliza *ARMo*, se le solicitará permiso para utilizar la cámara. En cambio, si no se ha podido iniciar sesión, se mostrará un mensaje de error indicando que los datos introducidos no son correctos o que no se ha podido establecer conexión con el servidor Zabbix.

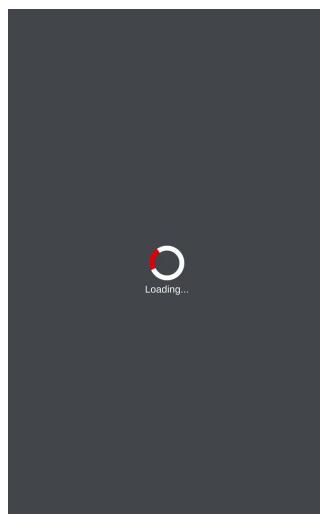


Figura A.2: Pantalla de carga.

Después de conceder el permiso de uso de la cámara, esta se activará y se mostrará una plantilla o recuadro (Figura A.3) en el cual debe ajustar la imagen que desea escanear.



Figura A.3: Recuadro para ajustar la imagen a escanear.

Si esta imagen está asociada a un rack, en la parte superior de la pantalla se mostrará el nombre de este elemento y sobre la imagen se posicionarán los siguientes tres modelos 3D como se puede observar en la Figura A.4:

- En el centro un modelo del rack.
- En la esquina superior izquierda se indica cantidad de componentes del rack disponibles, no disponibles y con estado desconocido.
- En la esquina inferior derecha se indica la cantidad de componentes del rack que presentan o no presentan problemas.

Si la imagen se corresponde a un componente de rack, en la parte superior de la pantalla se mostrará el nombre del componente y sobre la imagen se posicionarán cinco modelos 3D, como se puede observar sobre la Figura A.5, que representan los siguientes recursos:

- En el centro un modelo del componente del rack con un ícono que indica si el componente tiene problemas y de tenerlos, muestra el nivel de severidad más alto. El nivel de severidad es un indicativo de la gravedad de un problema. Ordenados de menor a mayor (y de izquierda a derecha en la imagen de la Figura A.6), estos consisten en: no hay problemas, *not classified*, *information*, *warning*, *average*, *high* y *disaster* (estos tres últimos niveles son considerados críticos, por lo cual son representados por el mismo símbolo).
- En la esquina superior izquierda una gráfica circular que representa el porcentaje de uso del disco.

-
- En la esquina superior derecha un modelo que indica el estado de la conexión (o accesibilidad) del componente que puede ser: estado desconocido (Figura A.7a), disponible (Figura A.7b) y no disponible (Figura A.7c).
 - En la esquina inferior izquierda una gráfica circular que representa el porcentaje de uso de la RAM.
 - En la esquina inferior derecha una gráfica circular que representa el porcentaje de uso de CPU.



Figura A.4: Detección de una imagen asociada a un rack.



Figura A.5: Detección de una imagen asociada a un componente de rack.



Figura A.6: Niveles de severidad.



(a) Estado desconocido. (b) Estado disponible. (c) Estado no disponible.

Figura A.7: Estados de la conexión de los componentes de rack.

Tanto si la imagen detectada está asociada a un rack como a un componente de rack, aparecerán una serie de botones en la parte inferior de la pantalla que le permitirán navegar entre los modelos de los recursos mostrándolos en el centro de la imagen y ocultando el resto para mejorar su visualización (Figura A.8). También puede pulsar sobre uno de los modelos para acceder a su pestaña correspondiente desde la pestaña inicial.



Figura A.8: Pestaña asociada a un recurso.

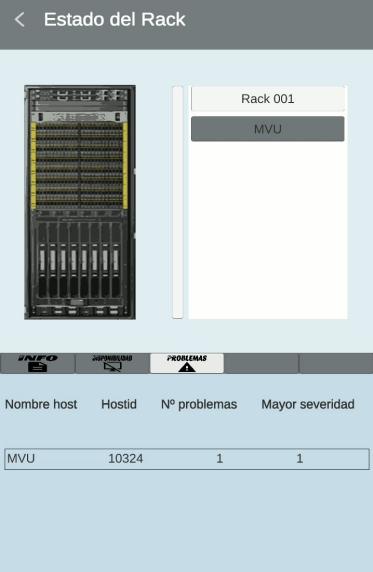
En el lateral derecho de la pantalla se encuentra un botón que le mostrará información más detallada sobre el recurso consultado. La nueva ventana (Figura A.9) tiene una lista de

botones para acceder a la información del rack o a la de sus componentes. Esta información se mostrará en las pestañas de la zona inferior de la pantalla. La información del rack se muestra en tres pestañas (Figura A.9a):

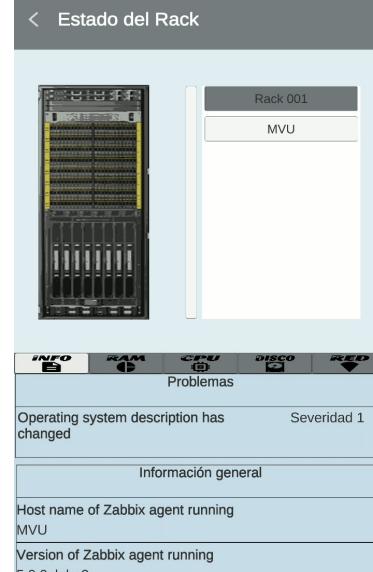
1. Contiene una descripción general del rack.
2. Contiene una clasificación de sus componentes según su disponibilidad
3. Contiene una lista con los componentes que presentan problemas.

Y la información de los componentes del rack se muestra en cinco pestañas (Figura A.9b):

1. Contiene una lista con los problemas del componente y con información general del elemento.
2. Contiene información sobre el estado de la memoria.
3. Contiene información sobre el estado del procesador.
4. Contiene información sobre el estado del disco.
5. Contiene información sobre el estado de la red.



(a) Rack.



(b) Componente de rack.

Figura A.9: Ventana con información detallada sobre los activos del CPD.

En la parte superior izquierda de la pantalla hay un botón que le permite volver a utilizar las funciones de realidad aumentada de *ARMo* para acceder a la información de otros racks.

APÉNDICE A. GUÍA DE USUARIO

Otra de las funcionalidades que ofrece *ARMo* es la detección de dos imágenes de forma simultánea (Figura A.10a). Al usar esta función, se le pedirá que pulse sobre los modelos de una de las imágenes y podrá acceder a su información a la vez que se muestran los modelos de la otra imagen (Figura A.10b).



(a) Instante previo a la interacción del usuario con los modelos.
(b) Interacción del usuario con los modelos de una imagen.

Figura A.10: Detección de dos imágenes de forma simultánea.

Lista de acrónimos

RA *Realidad Aumentada.*

CPD *Centro de Procesamiento de Datos.*

IDE *Integrated Development Environment.*

API *Application Programming Interface.*

SDK *Software Development Kit.*

XML *eXtensible Markup Language.*

JSON *JavaScript Object Notation.*

Glosario

Asset *Representación de cualquier ítem que se pueda utilizar en un proyecto de Unity. Puede ser tanto interno o externo a Unity y puede ser cualquier tipo de archivo como un modelo 3D, un audio, una imagen, etc.*

GameObject *Objeto de Unity que representa una entidad interna del juego como personajes, accesorios, luces, cámaras, escenarios y efectos especiales. No pueden hacer nada por sí mismos. Sirve como contenedor de Components, mediante los cuales obtiene propiedades y funcionalidades.*

Component *Elemento que se asocia a los GameObjects para otorgarles propiedades como por ejemplo posición, rotación y escala mediante un Transform o funcionalidades mediante un script.*

Prefab *GameObject configurado, es decir, con sus propios Components (incluyendo los valores de sus campos) y sus propios GameObject hijos. Incrementa la reutilización gracias a permitir instanciar objetos con las mismas características y gracias a posibilitar la creación de variantes de un mismo objeto.*

Transform *Tipo de Component encargado de la posición, rotación y escala de los GameObjects.*

Rigidbody *Tipo de Component que permite a los GameObjects actuar bajo el control de la física.*

Collider *Tipo de Component que define la forma de los GameObjects para los propósitos de colisiones físicas como por ejemplo la detección de las pulsaciones del usuario.*

Bibliografía

- [1] R. T. Azuma, “A survey of augmented reality,” *Presence: Teleoperators & Virtual Environments*, vol. 6, no. 4, pp. 355–385, 1997.
- [2] C. Arth, R. Grasset, L. Gruber, T. Langlotz, A. Mulloni, and D. Wagner, “The history of mobile augmented reality,” *arXiv preprint arXiv:1505.01319*, 2015.
- [3] Augment, “Infographic: The history of augmented reality - augment news,” 2016. [En línea]. Disponible en: <https://www.augment.com/blog/infographic-lengthy-history-augmented-reality/>
- [4] “Soporte para dell emc openmanage mobile | dell us.” [En línea]. Disponible en: <https://www.dell.com/support/article/us/en/04/sln310980/soporte-para-dell-emc-openmanage-mobile?lang=es>
- [5] “Multi-purpose intuitive knowledge assistant (mika) | nokia networks.” [En línea]. Disponible en: <https://www.nokia.com/networks/services/digital-assistant-as-a-service/>
- [6] “Intel® network builders – nokia* demo: Data center management powered by mikka at mwc 2018.” [En línea]. Disponible en: <https://networkbuilders.intel.com/social-hub/video/nokia-demonstration-of-data-center-management-powered-by-mikka-at-mwc-2018>
- [7] “Ge data center ar en app store.” [En línea]. Disponible en: <https://apps.apple.com/es/app/ge-data-center-ar/id1226007262>
- [8] “Build new augmented reality experiences that seamlessly blend the digital and physical worlds.” [En línea]. Disponible en: <https://developers.google.com/ar>
- [9] “Augmented reality - apple developer.” [En línea]. Disponible en: <https://developer.apple.com/augmented-reality/>

- [10] “Wikitude augmented reality: the world’s leading cross-platform ar sdk.” [En línea]. Disponible en: <https://www.wikitude.com/>
- [11] “Vuforia developer portal |.” [En línea]. Disponible en: <https://developer.vuforia.com/>
- [12] “Ar.js/readme.md at master · jeromeetienne/ar.js · github.” [En línea]. Disponible en: <https://github.com/jeromeetienne/AR.js/blob/master/README.md>
- [13] “three.js – javascript 3d library.” [En línea]. Disponible en: <https://threejs.org/>
- [14] M. Lanham, *Learn ARCore-Fundamentals of Google ARCore: Learn to build augmented reality apps for Android, Unity, and the web with Google ARCore 1.0.* Packt Publishing Ltd, 2018.
- [15] “Unity - manual: Unity user manual (2019.3).” [En línea]. Disponible en: <https://docs.unity3d.com/Manual/index.html>
- [16] M. Lidon, *Unity 3D.* Marcombo, 2019.
- [17] “Unity - scripting api.” [En línea]. Disponible en: <https://docs.unity3d.com/ScriptReference/index.html>
- [18] “Unity - manual: Unity’s interface.” [En línea]. Disponible en: <https://docs.unity3d.com/Manual/UsingTheEditor.html>
- [19] “Ide de visual studio, editor de código, azure devops y app center - visual studio.” [En línea]. Disponible en: <https://visualstudio.microsoft.com/es/>
- [20] “C# docs - get started, tutorials, reference. | microsoft docs.” [En línea]. Disponible en: <https://docs.microsoft.com/en-us/dotnet/csharp/>
- [21] S. Putier, *C# 7 y Visual Studio 2017 Los fundamentos del lenguaje.* Ediciones Eni, 2018.
- [22] “Zabbix :: The enterprise-class open source network monitoring solution.” [En línea]. Disponible en: <https://www.zabbix.com/>
- [23] A. Dalle Vacche, *Mastering Zabbix.* Packt Publishing Ltd, 2015.
- [24] “Oracle vm virtualbox.” [En línea]. Disponible en: <https://www.virtualbox.org/>
- [25] “The world’s leading software development platform · github.” [En línea]. Disponible en: <https://github.com/>
- [26] R. G. Figueroa, C. J. Solís, and A. A. Cabrera, “Metodologías tradicionales vs. metodologías ágiles,” *Universidad Técnica Particular de Loja, Escuela de Ciencias de la Computación,* 2008.

BIBLIOGRAFÍA

- [27] “Manifesto for agile software development.” [En línea]. Disponible en: <http://agilemanifesto.org/>
- [28] “Scrum, what’s in a name? - dzone agile.” [En línea]. Disponible en: <https://dzone.com/articles/scrum-whats-in-a-name>
- [29] K. Schwaber and J. Sutherland, “The scrum guide,” *Scrum Alliance*, vol. 21, p. 19, 2011.
- [30] “Github - rdub80/aframe-gui: Graphical user interface component framework for a-frame vr.” [En línea]. Disponible en: <https://github.com/rdub80/aframe-gui>
- [31] “Zabbix appliance [zabbix documentation 5.0].” [En línea]. Disponible en: <https://www.zabbix.com/documentation/current/manual/appliance>
- [32] “Zabbix documentation [zabbix documentation 5.0].” [En línea]. Disponible en: <https://www.zabbix.com/documentation/current/start>
- [33] “Github - vidlec/zabbix.net: Zabbix api library for .net writen in c#.” [En línea]. Disponible en: <https://github.com/Vidlec/Zabbix.NET>
- [34] “Nuget gallery | newtonsoft.json 12.0.3.” [En línea]. Disponible en: <https://www.nuget.org/packages/Newtonsoft.Json>
- [35] A. E. B. O. del Estado, *Resolución de 7 de octubre de 2019, de la Dirección General de Trabajo, por la que se registra y publica el XIX Convenio colectivo del sector de empresas de ingeniería y oficinas de estudios técnicos.* BOE, 2019. [En línea]. Disponible en: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2019-14977
- [36] “Unity - manual: Canvas.” [En línea]. Disponible en: <https://docs.unity3d.com/2020.1/Documentation/Manual/UICanvas.html>

Bibliografía