uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Informática

2017-2018

Trabajo Fin de Grado

"Sensorización en realidad aumentada"

Margarita Varela Mato

Tutor

José Antonio Iglesias Martínez

Leganés, Julio de 2018

ABSTRACT

Introduction

This document presents the End of Degree Work done by this autor at the Carlos III University of Madrid.

The basic premise of this Project consists in the creation of an application where an augmented reality model of terminal T4 of the Adolfo Suárez Madrid – Barajas Airport is shown, with temperatura and humidity sensor that provides information in real time during the execution.

Throughout the document, the realization of this Project is described, from its planning and Budget, through the analysis of the environment and the theoretical framework, to its analysis and codification.

The implementation of this work tries to demonstrate how the synergy between technologies, what may seem disparate, is complementary and can be used with a clear benefit. In this case, the information is not useful, but it is an example of how this use can be transferred to the real field.

Motivation

The augmented reality is an emerging technology, which over the years will be increasingly immersed in our daily lives. The possibilities that it presents are endless since it allows complement the perception and interaction with the real world by the user, allowing him to be in a real environment, but with additional information generated by computer.

This technology is part of the Tango Project developed by a division of Google called ATAP (Advanced Technology and Projects) and whose first version was released on June 5, 2014.

Tango uses computer vision to allow mobile devices to detect their relative position in the world around them without using GPS or other external signals, allowing developers to create experiences that include interior navigation, 3D scanning, measurements of physical space, recognition of the environment and positioning of virtual windows on the real world.

Its characteristics provide great realism in terms of placing virtual objects in reality taking into account the conditions of the environment in which it is located.

Another technology that is used in the realization of this project consists of a humidity and temperature sensor connected to a Raspberry Pi device, in such a way that it is possible to obtain fairly accurate data of the environmental conditions belonging to the location in which it is located.

What motivates this work is to discover how new forms of augmented reality work, how objects are created in three dimensions visible through the camera of a mobile phone and discover how to endow them with realism, creating an experience for the user that is not usual in the present. The fact of working with a relatively unknown technology at the moment (at the user level, above all) means knowing new possibilities and applications for its use that other technologies already known and most popular do not offer.

Another of the main motivations has been the agreement between the concepts of augmented reality and sensorization, the union of two technologies a priori, disparate, but which together complement each other very well and provide a small idea of the capabilities of combined systems of this type.

The fact of using sensorization has been caused by the desire to learn more about the possibilities offered by these systems, a concern that arose long ago, but for which there was no greater purpose to browse and build a domestic system that uses this type of technologies, so that including it in this project not only enriches its objectives, but also responds to personal purposes.

Taken together, this application could serve as a starting point for more complex applications or that have similar functions, since most of the tools offered by the Unity rendering engine have been put to use, and, therefore, with certain Changes could be reused for similar applications or that use some of their functionalities.

Goals

The purpose of this project is to develop an augmented reality application that simulates the operation of Terminal 4 of the Adolfo Suárez-Madrid Barajas Airport using temperature and humidity information obtained in real time and provided through a sensor, and non-real information about flights, generated for that purpose in a partially random manner. For this purpose, the capabilities offered by the Google Tango project, the DHT22 temperature and humidity sensor and its connection to the Raspberry Pi 2 Model B are used.

Therefore, this system will have two main parts:

- Augmented reality: a reproduction of the airport is created in three dimensions as true as possible to reality, but always taking into account the space we have, since the application is used through a mobile phone and must be perfectly visible and manageable by the user.
- Sensorización: Sensorisation: the humidity and temperature sensor will give information about the conditions in which the location is located (ideally it would be at the airport to use that information in the application properly, as it is not the case, simply will show the information).

The objectives that are sought throughout this work are the following:

- Study the different development options that are available: modeling and texturing tools in 3 dimensions, types of sensors that can provide the information sought here, acquire notions of the connections between sensor and computer, rendering programs for the creation of the graphic environment and the programming of the operation, etc.
- Know in detail the development environment of augmented reality technologies.
- Build the application following design and usability criteria.
- Establish a work plan and a budget for the realization of it.
- Document all the work done.

Estate of the art

There are two relevant aspects in this project: on the one hand, there is augmented reality, a key part of this work and that encompasses both the design made in three dimensions and everything that affects visualization, and, on the other, the information coming from of the sensors and obtaining the data.

The design of any structure or element in 3D requires a series of processes to achieve a final product. It is recommended that these phases begin with a manual design (either in paper or electronic format) in two dimensions of what you want to transfer to the modeling programs.

Once the sketch is clear, we move on to the modeling phase where it will be shaped as if it were clay. Each program has a multitude of modeling tools. There are those that allow to modify the structure based on brushes with certain shapes or those that provide other types of modifications, less artistic, but more precise. The choice of one or the other will depend fundamentally on the objectives sought and the skill of the modeler.

Once the finished modeling is completed, the texturing phase continues. It is so called because it is to translate a texture on the three-dimensional figure, and this can be flat (for example, a blue ball) or can have infinite nuances. The texturing process is what gives realism to a figure in three dimensions, since it is possible to choose different materials (wood, vinyl, tile, block, earth, brick, cement ...), opacity characteristics, how it behaves with light (reflects, absorbs, refracts ...), relief, etc.

In large part of the modeling (and to a greater extent in the case of more complex ones such as humans, animals and all kinds of organic beings) it is necessary to analyze and export the mesh in two dimensions, before mapping a texture. This process is called UV mapping and consists of repositioning the vertices and faces that form the mesh in such a way that the texture looks uniform, without visual cuts. Any image editing program can be used to perform this step.

The next stage of 3D design is to provide movement (if required) to the created object. You can distinguish between two types of animation: linear movements and complex animations. While for the first one we need an initial and a final point, in such a way that the program generates all the intermediate frames, for complex animations a more complex process must be done.

Rendering is the last part of the process, in which the necessary light and environment conditions are added so that the scene fulfills the objectives set. This part is essential in areas such as interior design since it is necessary to make reproductions of very realistic scenarios.

The current trend in the area of 3D design is increasingly focused on reality, to build virtual universes that are exactly the same to the real world, so it is popularizing all types of software that allows moving drawings or objects to the three dimensions. There are already certain devices that allow you to scan objects or people and introduce them in a virtual world [14], as well as softwares that materialize the photos we make of an object in a virtual mesh with the texture of our photographs [15].

The term augmented reality was coined by Tom Caudellen 1992 [16], but from then on several definitions have been happening, of which one of them perfectly limits the scope of this emerging technology. It was provided by Ronald Azuma in 1997 [17] and states that augmented reality responds to three main characteristics:

- Combine real and virtual elements.
- It is interactive in real time.
- It is registered in three dimensions.

Although virtual reality and augmented reality are often conceived as a technology within the same theoretical framework, there are profound differences between the two:

- The virtual reality moves the user to a world created expressly, where he is oblivious to everything that happens in reality. On the other hand, augmented reality consists of the enrichment of the real world, of the inclusion of virtual elements in the real context, so that it does not suppose an absolute immersion for the user.
- Although the devices that allow access to these two worlds are similar, the characteristics differ. In the case of the virtual, they usually use glasses or helmets that completely isolate the person (such as Oculus Rift or HTC Vive) and that require an external machine (in this case a computer capable of executing this type of applications) while that in the augmented we enjoy a full vision and audio of the outside world (Hololens or Daqri) and are usually autonomous devices (mobile, tablet or glasses).
- Both are focused on different areas, while virtual reality is generally limited to the world of entertainment, the augmented reality has a greater field of application since it does not reverse the dangers of being in a virtual world alien to reality

The Tango project, developed in 2014 by a division of Google called Advanced Technology and Projects (ATAP), uses computer vision to allow mobile devices that use your system to detect their position in relation to the world around them without using the GPS or any other external signal. This type of technology allows the development of applications that include interior navigation, 3D mapping, measurement of physical spaces and recognition of rooms, among others.

In December 2017 Google announced the closure of the Tango platform, so although there are still developers who create content for this augmented reality format, Google will not provide more support for it. The main reason has been the hardware requirements to use this

technology since it is only available through two smartphones with very specific characteristics and with prices similar to the mobile phones of medium - high range. These include the triple camera to detect depth and perform positioning, a large storage for applications and a high processing capacity to get these applications run smoothly.

All this has contributed to Google prefer to continue with the development of augmented reality technologies with an application that has much more in common the developments carried out by its biggest competitor. This is the case of ARCore [24], the new kit that Google expects to continue working with developers interested in augmented reality and who will replace Tango. Its characteristics are similar to those of its predecessor, but with the advantage of not requiring specific hardware but that can be integrated into applications for any conventional Smartphone. Using the standard sensors of mobile devices (gyroscopes, accelerometers, laser focus), ARCore is able to recognize objects and distances in three dimensions using the integrated camera.

A sensor is a device that has the characteristic of being sensitive to a certain magnitude, so that by varying this magnitude in the environment a measurement corresponding to that variation in these devices is obtained.

There are many existing examples of the use of sensors in combination with the *Raspberry Pi* device since this mini computer is capable of working autonomously without more requirements than a power source and it is not difficult to establish a connection with other devices.

Analysis

For the realization of this project, decisions have been made about which platforms to use, what programming language, what kind of sensors, etc. And it is in this section where those decisions are specified.

The fact of working with augmented reality greatly restricts the possibilities to develop content, since there are few platforms that support it or that give access to the development package. Some of these are: Vuforia, ARToolKit, ARKit or Tango.

The choice to develop this application on the Tango platform lies mainly in the one that wanted to use a powerful platform but that had not been used until then by the author of this end-of-degree project. Thus, and having worked with other virtual reality platforms (Oculus Rift, HTC Vive) and augmented (Microsoft Hololens), we wanted to see the possibilities of this new technology.

On the other hand, it had quickly ruled out platforms such as ARKit for the economic component or Vuforia because of the limitations of the free version of the license.

Finally, it is important to note that the augmented reality provided by Tango differs from the others in the sense of depth since the fact of running it on devices that have three cameras ex profeso (a RGB of 16 MPX, one of depth and another with a fisheye lens) for this type of technology, provides a different experience to that of augmented reality in a conventional Smartphone.

A three-dimensional modeling software is a tool that allows you to create and manipulate 3D graphics by computer. There are several options in this area, more or less specialized and with different characteristics): ZBrush, 3DMAX, Autodesk Maya and Blender.

Finally, for the realization of this TFG, the 3D modeling software called Maya has been selected. The main reason for this selection is that, as the author of this project, she had experience working with this program, in addition to the free license for being a student and a 280-hour course that was previously held. Regardless of this, and having worked with two of the other softwares (Blender and Zbrush) has been considered that the first does not have the tools or can not make them intuitive for any programmer to use, so try to model something in it is complicated. In the case of Zbrush is a software that is much more focused on a purely organic modeling and with processes similar to manual craftsmanship, in which a figure with different brushes and tools is being shaped, not suitable for the structural modeling that it was needed.

One of the fundamental pillars of the development of an application such as the one presented in this end-of-degree project is the coding of its operation, establishing a series of classes that interact with each other and with the objects of the scene, to ensure that the desired activity is carried out, for which it was necessary to choose an appropriate program.

For the development of an application for Android we have different options: Android Studio, Unreal and Unity.

Both Unity and Unreal had a Project Tango SDK to start developing the application, however, Unity was chosen mainly because it is software whose tools are more thoroughly known since it has been worked for almost a year and a half in a way professional with him. Android Studio was ruled out because working at the graphic level and with objects in three dimensions is more complex than in any of the other two, besides that the export in Unity (for the platform that is) is very simple.

It is also very important that there is a large Internet community that develops on this platform and that means that most problems or errors that arise, will have some response in the forums or documentation, which will always decrease the time spent solving them.

The communication between the mobile application and the sensors connected to the mini computer was essential to show the temperature and humidity information of the fictitious airport.

There are several Raspberry models that we explain below: Raspberry Pi 1 Model A, Raspberry Pi 1 Model B and B, Raspberry Pi 2 Model B, Raspberry Pi 3 Model B and Raspberry Pi Zero models. The latter were not considered because they have a lower power and are much smaller. At the time of devising the end-of-degree project, there were already two of these devices with which to work: the Raspberry Pi 1 model B and the Raspberry Pi 2 model B.

This way, once it was verified that the two devices served to the implementation of the sensors and the communication with the application, it was decided to use the most recent model (2-B) because it provided more connection options and at first it was unknown if only one or several sensors would be used, so the option more complete.

The fact of wanting to show real-time information about conditions that you do not have access to priori (such as temperature and humidity) implied having to acquire a device that could access that information and make it visible in some way. This is where the use of sensors capable of determining what environmental conditions are available and how to communicate it through the Raspberry Pi to reach the application is necessary.

Regarding the choice of sensors, it was clear from the first moment that the choice would be limited to two main models, as a consequence of three initial requirements: that they be available quickly (through Amazon's online sale), that they were simple to install and that there was some tutorial or guide to do it.

This last requirement has been the one that tipped the scales for these two models: DHT11 or DHT22. In this case the DHT22 sensor was selected because it provided a wider range of measurements, as well as more reliable and with a lower error.

Design

Regarding the visual design, we wanted to maintain a certain coherence with the colors used, as well as the types of buttons, shading and typography so that these details do not break the thread of the execution.

The modeling of terminal four has been carried out in accordance with the photos obtained from the Google Earth application that allow a very precise approach to the structures of the same. In addition, several websites of photographs have been consulted to verify certain details of both modeling and texturing. It could not be 100% realistic because the extension of the terminal is very large compared to the space that can be had to execute the application, but everything else has tried to be faithful to the real model (in shapes, textures and colors):

For the initial screen we wanted a simple background related to the theme of the application, which already delimited a little the possibilities. Finally, a blue background with clouds and the image of one of the buildings in terminal T4 were chosen to give a clue about the content of the application.

The same goes for the choice of the name, given that the content is somewhat holographic, it has opted for the simplicity of a name that is easy to remember and descriptive, such as HoloT4. The distribution of buttons on all screens responds to the criterion of greater visibility and the user can use them without interfering too much with the view of the terminal.

As for the information of the sensors, it is shown through a screen designed as if it were a monitor with a type of letter very similar to that of any electronic screen. The fact that the numbers are red breaks with the similarity of colors used, but considered that it was necessary for it to produce greater attention by the user.

Regarding the technical part, this section will explain the operation of the application, but to expand the technical details you can go to Annexes I, II, III and IV where these concepts will be detailed in detail.

When compiling the application from Unity we obtain an executable for Android with the extension .apk, which once clicked will install it on our device.

Unity allows the creation of various three-dimensional scenarios and the jump between them during execution so the initial screen was conceived as a single scene, apart from the rest of the content to not have a saturation of content and the thread of execution was fluid.

The composition of this scene consists of a background image composed of the terminal 4 building of the Madrid-Barajas airport, a blue sky, a series of white clouds and the name of the application, which are included as a single image inside of a panel created as a base. The other two main elements of the scene would be the buttons: Start and Exit, each of them configured in a similar way, but which performs different actions.

To configure the second scene it was important to hide or make visible certain sets of elements, so that the thread of the execution made sense. This resulted in the creation of three screens, each with specific elements and different functions, and happening during execution.

On the first of these screens, the camera of the device is opened with a printed message on it that urges the user to locate a place where he wants to place the airport (a table, the floor, a flat place) to touch the screen and place the three-dimensional model. Once this touch is made, two actions take place: on the one hand, the change to another screen (moment in which the message screen becomes hidden and the next one becomes visible) and on the other, the instantiation of the object in three dimensions, always with the same predefined rotation, so that the user will always see it at this point of execution from the same angle.

Once the terminal is placed, a series of buttons are shown in the upper left that will allow the user to make changes in the position of the 3D object: move, rotate and scale. In this way the user can adapt the three-dimensional object to the space available to view it correctly.

At the bottom of the screen there are two more buttons, Set and Home, which with the icon of a house symbolizes the return to the home screen. This option is useful because it allows you to restart the application and place the terminal in a different zone. If, on the other hand, the user likes the placement of the three-dimensional object, the fix button would be clicked, which in turn performs three functions: generating the information of eight departure flights and eight arrival flights, setting the terminal at the point where the user has adjusted it and change to the next screen.

The flight generation system randomly chooses between more than twenty destinations and flight codes, using the time at which the application is running so that all flights take off or land in the nearby minutes.

In the upper part there are two buttons, Departures and Arrivals, in which the user can see those flights that have been previously generated, as happens on the screens of an airport. As the seconds pass, you can see how the planes begin to move, to execute the takeoff or landing and parking in the terminal. Another option that this application raises is to know what information corresponds to each flight, and that can be known by touching the screen directly above each plane.

Future works

From the project we have proposed, a more complex application could be developed that would provide a better and more useful service with the use of real data and its continuous updating. On the other hand, we could improve usability by introducing a quick-use manual on the operation of the application that the user could access if needed.

These technical changes can be summarized in the following points:

- Change the way to generate flights so that real-time data is obtained through the official FlightStats API. This new functionality would offer real flight numbers and you could see a simulation of what actually happens at the airport, also having the advantage of knowing the status of each flight instantly. This part is very interesting but it has not been included in the current project due to economic reasons since the use of this API implies the payment of a fee.
- Following this line of improvement, landing aircraft could self-generate as flights are scheduled on airport screens, so that there is always an aircraft available to land. In the case of departing planes, we could change the status of the planes that land after a certain time, changing their arrival flight information to be programmed with a new departure flight (as in reality).
- In addition, it would be interesting to collect the real information provided by the sensors of the Adolfo Suárez-Madrid Barajas airport runway to integrate it into the application itself.
- Another option would be to increase the number and type of sensors, including some in the aircraft to provide us with altitude, speed, etc. information. And you could even perform an analysis of that data to include a history of the routes and that the user can see relative information on what altitude a certain route is made, for example.
- When this project was proposed, one of the ideas was that it would not only work as an augmented reality application but could be used in some way within the airport itself. Many people go to receive or dismiss their family or friends at airports and wait for the plane to take off, updating the information on mobile devices or attending the screens to better understand the status of the flights in question. The holographic part of this project could eventually be used as a virtual model that was present in the terminal, and that anyone could approach to see in a way that would not require having the device, if not with each other's own (which would imply a development with another platform that was not Tango's).

The premise of the realization of this project was to create a model in augmented reality, in the airport itself, so that users could see it through some device, so an option to be made to improve this work and guide it towards this topic, It would be to migrate it from platform.

One of the options is to have it in virtual reality and to use some glasses in an airport stand so that users know the tool. And once this phase is over, it could be adapted to a more advanced platform that does not require specific hardware (such as ARCore, Tango's successor) and make the application known.

So, once the users discovered it from the airport, they could download the application and interact with it anywhere without having to be present at the airport.

Also, if the idea prospered, the same approach could also be carried out for other airports or even choose the terminal you want within a drop-down in the same application.

Conclusions

Returning to the principle of this TFG, you can verify that the initial objectives have been met, and that the operation of the application you wanted to create is correct. The most complex part of this work has been to relate all the systems that intervene in the operation so that the flow of execution is succeeded in the right way: that the animations of the planes are executed when necessary, that they alternate randomly between the flights of departures and arrivals, that the departure and landing of each flight is executed without obstructing any other, that the data of the departure and arrival flights are generated, etc.

It has not been easy to structure all the generated content without forgetting important aspects, which from the point of view of the developer are implicit but it is important that they are present in the documentation. The fact of having to document all the steps taken and thoroughly explain the process also helps to analyze it and to know it more deeply, so this part has done nothing but improve the abilities of this author as far as those competences are concerned.

In spite of the great amount of time dedicated, the balance of this work has been satisfactory, mainly for all the learning realized on technologies of which the author of this work had not worked (as the use of sensors connected to a Raspberry Pi and his communication with a mobile application) and the consolidation of others that he knew and knew how they worked, but he had not developed as widely (a mobile application in Unity using augmented reality).

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

1. INTRODUCCIÓN

1.1Motivación

La realidad aumentada es una tecnología emergente, que con el paso de los años estará cada vez más inmersa en nuestra vida diaria. Las posibilidades que presenta son infinitas ya que permite complementar la percepción e interacción con el mundo real por parte del usuario, permitiéndole estar en un entorno real, pero con información adicional generada por ordenador.

Dentro de esta tecnología se enmarca el Proyecto Tango desarrollado por una división de *Google* denominada *ATAP* (*Advanced Technology and Projects*) y cuya primera versión vio la luz el 5 de junio de 2014.

Tango usa la visión por computador para permitir a dispositivos móviles detectar su posición relativa en el mundo que les rodea sin utilizar *GPS* u otro tipo de señales externas, por lo que permite a los desarrolladores crear experiencias que incluyan navegación interior, escaneado 3D, realizar mediciones del espacio físico, reconocimiento del entorno y posicionamiento de ventanas virtuales sobre el mundo real.

Este sistema incorpora tres tipos de funcionalidades:

- Rastreo de movimiento: se utiliza la visualización del entorno y la información proveniente del acelerómetro y del giroscopio, lo que permite rastrear el movimiento del dispositivo en el espacio.
- Reconocimiento del espacio: almacena los datos del entorno en un mapa que puede enriquecerse con metadatos como notas, instrucciones o puntos de interés y que eventualmente puede compartirse con otros dispositivos *Tango* o utilizarse más tarde.
- Percepción de la profundidad: detecta distancias, tamaños y superficies en el entorno.

Ilustración 1: Características principales del proyecto Tango

Estas tres características unidas proporcionan un gran realismo en cuanto a emplazar objetos virtuales en la realidad teniendo en cuenta las condiciones del medio en el que se encuentra.

Otra de las tecnologías que se usa en la realización de este proyecto consiste en un sensor de humedad y temperatura conectado a un dispositivo *Raspberry Pi*, de tal forma que se pueda obtener datos bastante precisos de las condiciones ambientales pertenecientes a ubicación en la que esté el sensor.

Lo que hace motivador a este trabajo es descubrir cómo funcionan las nuevas formas de realidad aumentada, cómo se crean objetos en tres dimensiones visibles a través de la cámara de un móvil y descubrir como dotarlos de realismo creando una experiencia para el usuario nada habitual en la actualidad. El hecho de trabajar con una tecnología relativamente poco conocida por el momento (a nivel usuario, sobre todo) implica conocer nuevas posibilidades y aplicaciones para su uso que otras tecnologías ya conocidas y más populares no ofrecen.

Otra de las motivaciones principales ha sido la concordancia entre los conceptos de realidad aumentada y sensorización, la unión de dos tecnologías a priori, dispares, pero que unidas se complementan muy bien y proporcionan una pequeña idea de las capacidades de sistemas combinados de este tipo.

El hecho de utilizar sensorización ha sido provocado por las ganas de aprender más sobre las posibilidades que ofrecen estos sistemas, una inquietud que surgió hace tiempo, pero para la que no existía un propósito mayor que curiosear y construir algún sistema doméstico que utilice este tipo de tecnologías, por lo que incluirlo en este proyecto no sólo enriquece los objetivos del mismo, sino que también responde a propósitos personales.

En conjunto, esta aplicación podría servir de punto de partida para aplicaciones más complejas o que tengan funciones similares, ya que se han puesto en uso la mayor parte de las herramientas que ofrece el motor de renderizado *Unity*, y, por lo tanto, con ciertos cambios podría reutilizarse para aplicaciones similares o que utilicen alguna de sus funcionalidades.

1.2 Objetivo

El propósito que persigue este proyecto es desarrollar una aplicación en realidad aumentada que simule el funcionamiento de la Terminal 4 del Aeropuerto Adolfo Suárez-Madrid Barajas utilizando información de temperatura y humedad obtenida en tiempo real y proporcionada a través de un sensor, e información no real sobre los vuelos, generada para ese fin de forma parcialmente aleatoria. Para este fin, se utilizan las capacidades que ofrece el proyecto Tango de *Google*, el sensor de temperatura y humedad *DHT22* y su conexión con la *Raspberry Pi* 2 Modelo B.

Por lo tanto, este sistema contará con dos partes principales:

- Realidad aumentada: se crea en tres dimensiones una reproducción del aeropuerto lo más fiel posible a la realidad, pero siempre teniendo en cuenta el espacio del que disponemos, ya que la aplicación se utiliza a través de un móvil y debe ser perfectamente visible y manejable por parte del usuario.
- Sensorización: el sensor de humedad y temperatura dará información sobre las condiciones en las que se encuentra el lugar donde esté ubicado (idealmente se tendría

en el aeropuerto para utilizar esa información en la aplicación de forma adecuada, como no es el caso, simplemente se mostrará la información).

Los objetivos que se buscan a lo largo de este trabajo son los siguientes:

- Estudiar las diferentes opciones de desarrollo de las que se disponen: herramientas de modelado y texturizado en 3 dimensiones, tipos de sensores que puedan proporcionar la información que aquí se busca, adquirir nociones de las conexiones entre sensor y ordenador, programas de renderizado para la creación del entorno gráfico y la programación del funcionamiento, etc.
- Conocer en detalle el entorno de desarrollo de las tecnologías de realidad aumentada.
- Construir la aplicación siguiendo criterios de diseño y usabilidad.
- Establecer un plan de trabajo y un presupuesto para la realización de la misma.
- Documentar todo el trabajo realizado.

1.3 Marco regulador

La aplicación desarrollada en este proyecto, HoloT4, está sujeta a la regulación de la *Ley Orgánica de Protección de Datos Personales 13/1999 de diciembre*, que regula el tratamiento de los datos de carácter personal, todos aquellos derechos que tienen los ciudadanos sobre ellos y las obligaciones de los terceros que los crean o los tratan [1].

Al utilizarse la cámara del dispositivo móvil durante la ejecución de la aplicación, se debe cumplir esta normativa, de lo que el usuario ya está informado ya que, al instalarla se pregunta si se desea permitir el acceso.

Dado que la naturaleza de la aplicación no requiere que los usuarios se registren para hacer uso de ninguna de sus las funcionalidades, no es necesario analizar el tratamiento de los datos.

La información de los vuelos proporcionada en la herramienta es ficticia, por lo que no está sujeta al cumplimiento de ninguna normativa. En el caso de avanzar en el desarrollo de la misma y pasar a utilizar información en tiempo real de vuelos de la terminal 4 del Aeropuerto Madrid Barajas - Adolfo Suárez, sería necesario contratar un servicio con *Flight Stats* [2] ya que son los que disponen de la misma.

1.4 Entorno socioeconómico

La idea de convertir la realidad aumentada en algo cotidiano, que se utilice diariamente a través de los dispositivos móviles está cada vez más cerca. Son muchas las aplicaciones existentes en móviles especializados (como los que engloban el uso de esta aplicación) pero, debido en parte a aplicaciones de mucho éxito como *Pokémon Go*, se ha popularizado el uso de esta nueva tecnología soportada en mayor o menor medida por cualquier Smartphone [3][4].

El 66% de la población mundial ya cuenta con un móvil, siendo España la que lidera el ranquin mundial con un 88% de usuarios únicos. Si sumamos a estas estadísticas el hecho de que las nuevas generaciones de jóvenes dedican cada vez más tiempo a la utilización de estos

dispositivos, nos encontramos con un entorno con mucho potencial en el que desarrollar aplicaciones [5]:

Ilustración 2: Usuarios móviles frente a usuarios de internet en el mundo

Está previsto que entre 2016 y 2022 las tendencias de crecimiento de consumo desde el Smartphone se distribuyan de la siguiente manera:

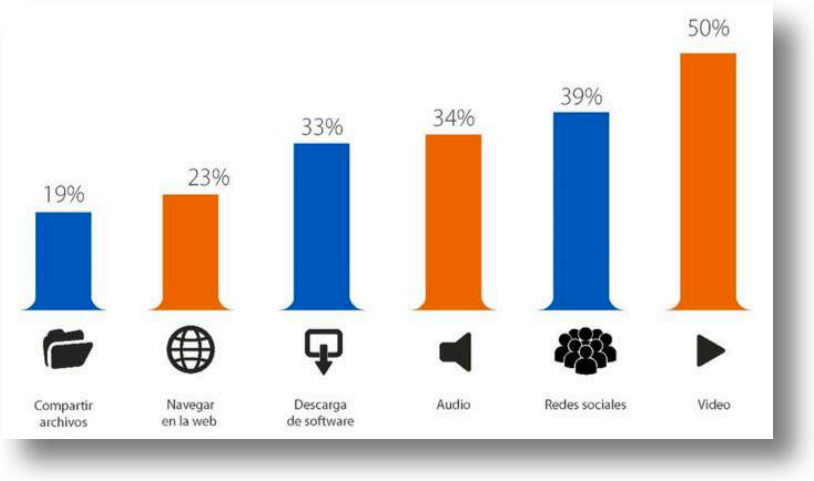

Ilustración 3: Gráfica de tendencias de crecimiento de consumo

Aunque el crecimiento es más acusado en otras áreas, la descarga de software tendrá un aumento significativo, un 33% nada desdeñable, que implica grandes oportunidades para el desarrollo en este campo.

Los españoles se gastan una media de 228 euros en cambiar de móvil (datos de 2017), aunque cada vez esa tendencia es mayor, ya que la cifra en el año 2016 estaba entorno a los 192 euros. Sin embargo, comparando las estadísticas de ambos años, podemos ver que el mayor

incremento se produce en la compra de dispositivos de gama alta, que se sitúa entre los 400 y 500 euros [6]:

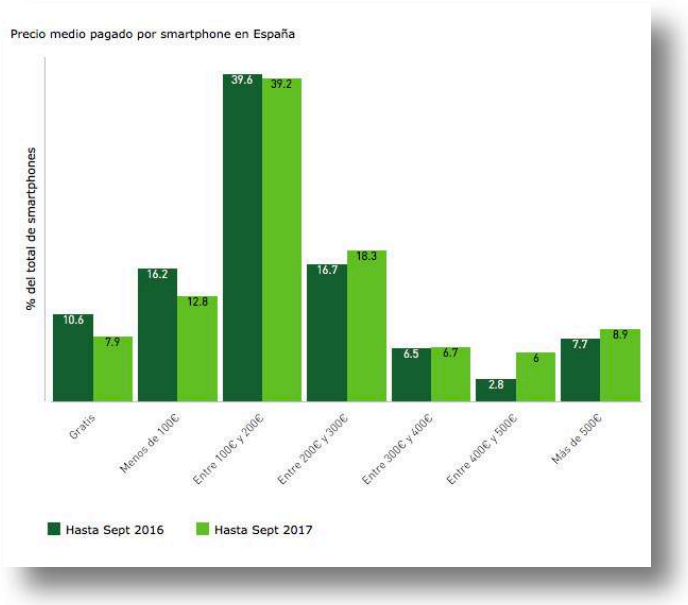

Ilustración 4: Gráfica precio medio pagado por Smartphone en España. Fuente: Kantar

El acceso a la aplicación *HoloT4* está limitado a dispositivos Android que posean el soporte para las librerías del proyecto Tango, lo cual se reduce a dos: *Lenovo Phab 2 Pro* y *Asus Zenphone AR*. Son dos *smartphones* que por su rango de precio se sitúan en la gama media-alta y por lo tanto siguiendo la tendencia mostrada por el gráfico anterior, están dentro de presupuesto que una parte importante de la población dedica a la compra de dispositivos móviles.

2. ESTADO DEL ARTE

En este apartado se van a abordar los aspectos más importantes de este proyecto. Por un lado, está la realidad aumentada, parte clave de este trabajo y que engloba tanto el diseño realizado en tres dimensiones como todo lo que afecta a la visualización, y, por el otro, la información proveniente de los sensores y la obtención de los datos.

2.1 Diseño en 3D

Se define como diseño 3D todo aquel conjunto de técnicas que permiten conseguir una representación tridimensional. Hoy en día no se concibe esta actividad sin estar ligada a un ordenador, pero antes de la dependencia de éstos, los profesionales utilizaban otras técnicas para conseguirlo, como por ejemplo planos técnicos y maquetas o modelos a escala.

Un primer acercamiento digital a este tipo de procedimientos lo realizó *Autocad* en 1997 [7], y a medida que pasaron los años han sido muchos los programas informáticos que han surgido para el diseño de entornos en tres dimensiones. El área se ha diversificado tanto que incluso hoy en día existe *software* muy específico en función del tipo de actividad para la que se realice el diseño, ya sea para arquitectura, diseño de interiores, videojuegos, películas animadas, etc.

El diseño de cualquier estructura o elemento en 3D requiere de una serie de procesos hasta conseguir un producto final. Es recomendable que estas fases comiencen con un diseño manual (ya sea en papel o formato electrónico) en dos dimensiones de aquello que se quiere trasladar a los programas de modelado. Este primer acercamiento al producto es muy importante ya que va a determinar las partes básicas que formarán la estructura y los detalles en los que se debe fijar el modelador, no obstante, es una parte más bien opcional y que los diseñadores expertos pueden no necesitar realizarla.

Una vez claro el esbozo, se pasa a la fase de modelado donde se le dará forma como si de barro se tratara. Cada programa tiene multitud de herramientas de modelado. Existen los que permiten modificar la estructura a base de pinceles con determinadas formas o los que proporcionan otro tipo de modificaciones, menos artísticas, pero más precisas. La elección de unos u otros dependerá fundamentalmente de los objetivos buscados y de la destreza del modelador. En esta etapa del diseño podemos presentar dos ejemplos totalmente opuestos, los programas *Zbrush* [8] y *Autodesk Maya* [9].

El primero se caracteriza por tener multitud de herramientas con los que dar forma de una manera manual, utilizando pinceles que actúan sobre la malla, sobre su tamaño, forma, posición, curvatura, etc. La utilización de diversos pinceles hace que se pueda pasar de un polígono inicial a algo tan complejo como un personaje de videojuegos. Es por ello que *Zbrush* está considerado como uno de los mejores softwares de diseño artístico para modelado [10].

Ilustración 5: Interfaz Zbrush

Autodesk Maya está más enfocado a modificaciones sobre vértices, aristas y caras de los polígonos de la malla que forma el objeto. Existen gran cantidad de posibilidades por lo que es relativamente sencillo conseguir figuras totalmente diferentes partiendo de un polígono básico (esfera, cilindro o cubo, por ejemplo).

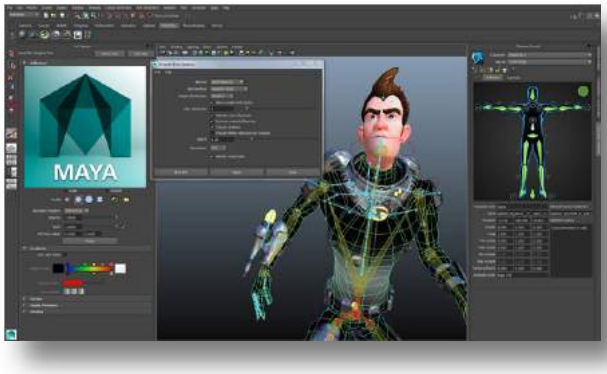

Ilustración 6: Interfaz Autodesk Maya

Una vez se tenga el modelado terminado se continúa con la fase de texturización. Se denomina así porque se trata de plasmar una textura sobre la figura tridimensional, y ésta puede ser plana (por ejemplo, una pelota de color azul) o puede tener infinidad de matices. El proceso de texturizado es lo que otorga realismo a una figura en tres dimensiones, ya que es posible escoger diferentes materiales (madera, vinilo, azulejo, bloque, tierra, ladrillo, cemento…), características de opacidad, cómo se comporta con la luz (la refleja, la absorbe, la refracta…), relieve, etc. [11].

En gran parte de los modelados (y en mayor medida en el caso de aquellos más complejos como humanos, animales y todo tipo de seres orgánicos) es necesario realizar un análisis y exportación de la malla en dos dimensiones, previo a mapear una textura. Este proceso se denominada mapeado UV y consiste en recolocar los vértices y caras que forman la malla de tal manera que

la textura se vea uniforme, sin cortes visuales, tal y como se muestra en la imagen a continuación:

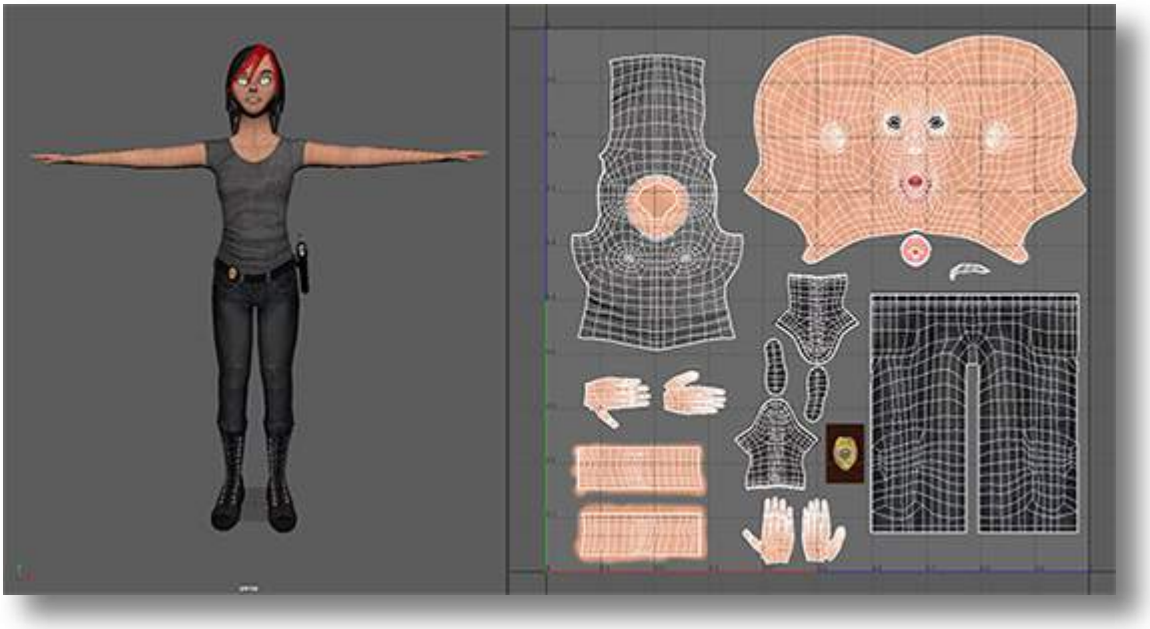

Ilustración 7: Desglose UV mapeo de textura en Autodesk Maya

Pintar una textura de piel y que resulte realista es muy complejo, y el hecho de que se pinte en dos dimensiones provoca cortes indeseados en los que destacan estas diferencias. Una zona en la que se puede observar el uso de mapeado de UV es en la cabeza del personaje (ilustración 7), ya que se han recolocado las caras de la malla para que la diferencia que pueda haber entre la pintura de las zonas cortadas quede oculta, es por ello que se realiza en la parte trasera de la misma.

Cualquier programa de edición de imágenes puede servir para realizar este paso.

La siguiente etapa del diseño en 3D consiste en dotar de movimiento (si se requiere) al objeto creado. Se puede distinguir entre dos tipos de animación:

- Movimientos lineales sencillos: son aquellos que requieren un fotograma inicial y otro final y es el propio programa el que genera todos los fotogramas intermedios. Uno de estas animaciones sería por ejemplo ejecutar el movimiento de una pelota desde un punto A hasta un punto B. Aquí podríamos englobar cualquier traslación, rotación y escalado que pueda sufrir el objeto en un intervalo de tiempo [12].
- Animaciones complejas: comprenden principalmente los movimientos de seres orgánicos, como por ejemplo el caminar de un personaje de videojuegos. En este tipo de animaciones el objeto 3D debe pasar por otra fase denominada *rigging* en la que se otorga una estructura similar a la de unos huesos humanos, a través de la cual el personaje ejecutará los movimientos de una forma controlada y más precisa. En estos

casos se anima cada una de las partes del cuerpo para cada instante, o también se puede generar una serie de fotogramas desde una posición inicial hasta otra final [13].

El renderizado es la última parte de proceso, en la que se añaden las condiciones de luz y entorno necesarios para que la escena cumpla los objetivos marcados. Esta parte es esencial en áreas como interiorismo ya que se necesita realizar reproducciones de escenarios muy realistas.

Ilustración 8: Render de estancia realista creada en un software de 3D

La tendencia actual en el área del diseño 3D está cada vez más enfocada a la realidad, a construir universos virtuales que sean exactamente iguales al mundo real, por lo que se está popularizando todo tipo de software que permita trasladar dibujos u objetos a las tres dimensiones. Existen ya determinados dispositivos que permiten escanear objetos o personas e introducirlos en un mundo virtual [14], así como softwares que materializan las fotos que hagamos a un objeto en una malla virtual con la textura de nuestras fotografías [15].

2.2 Realidad aumentada

El término realidad aumentada fue acuñado por Tom Caudellen 1992 [16], pero a partir de ahí se han ido sucediendo diversas definiciones, de las cuales una de ellas acota perfectamente el alcance de esta tecnología emergente. Fue proporcionada por Ronald Azuma en 1997 [17] y establece que la realidad aumentada responde a tres características principales:

- Combina elementos reales y virtuales.
- Es interactiva en tiempo real.
- Está registrada en tres dimensiones.

A pesar de que la realidad virtual y la realidad aumentada se conciben muchas veces como una tecnología comprendida dentro del mismo marco teórico, existen profundas diferencias entre ambas:

- La realidad virtual traslada al usuario a un mundo generado ex profeso, donde está ajeno a todo lo que ocurre en la realidad. Por el contrario, la realidad aumentada consiste en

el enriquecimiento del mundo de real, de la inclusión de elementos virtuales en el contexto real, por lo que no supone una inmersión absoluta para el usuario.

- A pesar de que los dispositivos que permiten acceder a estos dos mundos son similares, las características difieren. En el caso del virtual, acostumbran a ser gafas o cascos que aíslan por completo a la persona (tales como *Oculus Rift* o *HTC Vive*) y que requieren de una máquina externa (en este caso un ordenador capaz de ejecutar este tipo de aplicaciones) mientras que en de la aumentada gozamos de una plena visión y audio del mundo exterior (*Hololens* o *Daqri*) y por lo general son dispositivos autónomos (móvil, tablet o gafas).
- Ambas están enfocadas en áreas diferentes, mientras que la realidad virtual está limitada por lo general al mundo del entretenimiento, la realidad aumentada cada vez tiene un mayor campo de aplicación ya que no revierte los peligros de estar en un mundo virtual ajeno a la realidad.

Este último punto no resulta tan diferenciador como los dos anteriores ya que los dos ámbitos avanzan hacia una mayor incorporación en nuestra vida diaria, aunque sí es cierto que la realidad virtual tiene unas limitaciones que la realidad aumentada no posee [18].

Existen, además, aplicaciones de la realidad virtual más allá del mundo del entretenimiento, aunque sean por el momento minoritarias. Una de ellas está relacionada con la formación de personal cualificado para entornos peligrosos, ya que supone una gran ventaja poder enseñar a nuevos operarios aquellas labores que suponen un riesgo para la vida, pero sin trasladarse al entorno real, como por ejemplo en las subestaciones eléctricas. La mayor parte de las operaciones que aquí se realizan ponen en riesgo la vida de los operarios que las llevan a cabo, y más todavía si se trata de personal no cualificado o sin una amplia experiencia en este campo, por lo que utilizar esta herramienta implica un gran avance en sus condiciones de trabajo.

Enumeramos a continuación algunas de las aplicaciones de realidad aumentada:

- Videojuegos: *Super Mario Bros* y *Lemmings*, dos juegos que se hicieron populares en los años 90 tienen también su versión en realidad aumentada. En el caso de *Lemmings* [19], está disponible en *Microsoft Store* para *Hololens* y el de *Super Mario Bros* fue creado por Abhishek Singh y probado en Central Park, con una gran repercusión mediática [20].

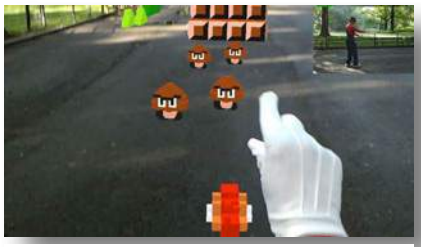

Ilustración 9: Videojuego Super Mario Bros para Hololens

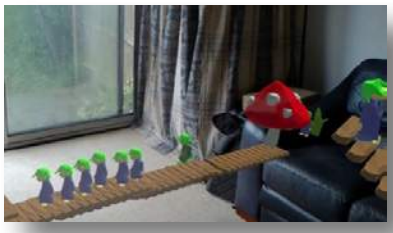

Ilustración 10: Videojuego Lemmings para Hololens

- *Global Positioning System (GPS)*: Una de las aplicaciones más utilizada hoy en día se centra en la navegación *GPS*, lo cual nos hace pensar que no resulta descabellado trasladarlo al ámbito de la realidad aumentada. Esta fue la idea que surgió desde diversas empresas para la creación de aplicaciones como *AR GPS CompassMap 3d* o *AR GPS Drive/WalkNavigation*.
- Industrial: Existen otros proyectos y dispositivos más enfocados al ámbito industrial, tal es el caso de las gafas y cascos *Daqri*. Estas poseen algunas características que las hacen idóneas para su funcionamiento en este tipo de entornos, como puedan ser: pueden usarse en entornos interiores y exteriores, poseen identificación por huella para detectar qué operario las está utilizando y tienen visión térmica que permite la detección de averías o de una posible zona peligrosa para el usuario.

Ilustración 11: Interfaz de las gafas de realidad aumentada Daqri

Una de las ventajas que se mostró durante su presentación en el *Consumer Electronics Show (CES)* de 2017 es la de cómo un operario sin conocimiento para solucionar una avería puede ser guiado por un experto desde las gafas con una video llamada en *streaming*.

Esta funcionalidad también la presentan las *Hololens* de *Microsoft*, cuyo *plugin* para Skype permite que los dos usuarios de la llamada (uno desde las gafas y otro desde cualquier otro ordenador o dispositivo móvil) puedan interactuar el uno en el espacio del otro.

Ilustración 12: Llamada por Skype para Hololens

2.3 Proyecto Tango

El proyecto *Tango*, desarrollado en 2014 por una división de *Google* denominada *Advanced Technology and Projects (ATAP)*, utiliza la visión por computador para permitir que los dispositivos móviles que utilicen su sistema detecten su posición en relación con el mundo que les rodea sin usar el GPS ni ninguna otra señal externa. Este tipo de tecnología permite el desarrollo de aplicaciones que incluyan navegación interior, mapeado 3D, medición de espacios físicos y reconocimiento de estancias, entre otras.

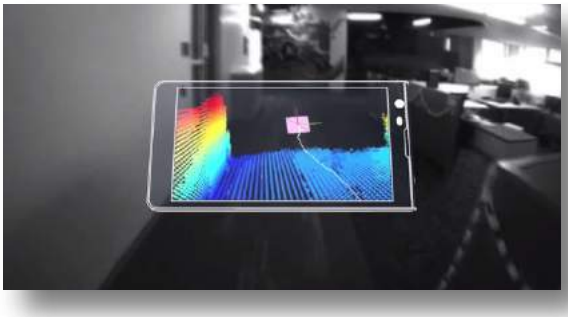

Ilustración 13: Rastreo utilizando Proyecto Tango

En enero de 2016 durante la celebración del CES [21], Google anunció una asociación con Lenovo para el lanzamiento de un *smartphone* que incluyera esta tecnología como forma de hacer que se popularizase y una gran cantidad de usuarios tuvieran acceso a ella [22]. Un año más tarde se llegó a un acuerdo con Asus para incluir esta misma tecnología en otro teléfono inteligente, y se anunció dicho lanzamiento en el CES de 2017 [23].

En diciembre de 2017 Google anuncia el cierre de la plataforma Tango, por lo que, aunque sigan existiendo desarrolladores que creen contenidos para este formato de realidad aumentada, Google no proporcionará más soporte para ello. El principal motivo ha sido los requisitos de hardware para utilizar esta tecnología ya que solo está disponible a través de dos *smartphones* con características muy específicas y con precios similares a los móviles de gama media – alta. Entre estas cabe destacar la triple cámara para detectar la profundidad y realizar el posicionamiento, un gran almacenamiento para las aplicaciones y una capacidad de procesamiento elevada para conseguir ejecutar estas aplicaciones de forma fluida.

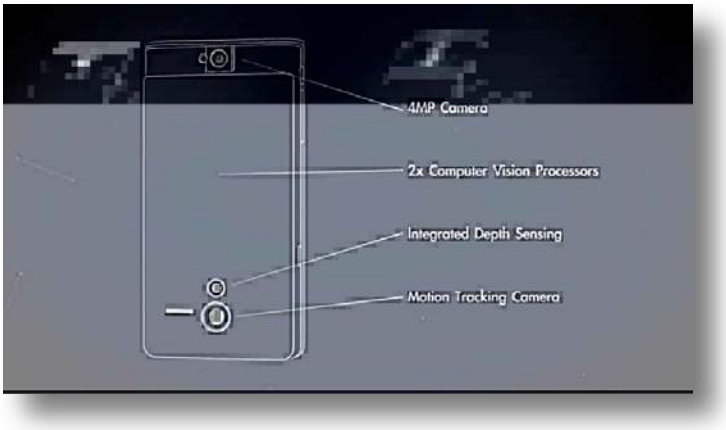

Ilustración 14: Características Hardware Tango

Todo ello ha contribuido a que *Google* prefiera continuar con el desarrollo de tecnologías de realidad aumentada con una aplicación que tiene mucho más en común los desarrollos llevados a cabo por su mayor competidor. Este es el caso de *ARCore* [24], el nuevo kit con el que Google espera que continúen trabajando los desarrolladores interesados en realidad aumentada y que sustituirá a *Tango*. Sus características se asemejan a las de su antecesor, pero con la ventaja de no requerir hardware específico si no que puede integrarse en aplicaciones para cualquier Smartphone convencional. Haciendo uso de los sensores estándar de los dispositivos móviles (giroscopios, acelerómetros, enfoque láser), *ARCore* es capaz de reconocer objetos y distancias en tres dimensiones utilizando la cámara integrada.

Ilustración 15: ARCore, sucesor de Tango

En cuanto a las aplicaciones más populares en los dispositivos que pueden utilizar Tango, podemos destacar tres principales:

- Medir objetos: *Measure* es una aplicación que permite medir objetos reales desde nuestro dispositivo.
- Escanear el entorno: *Scan* proporciona la posibilidad de escanear cualquier estancia de nuestro entorno para disponer de ella en tres dimensiones.
- Posicionar personajes: *Holo* permite colocar personajes animados precargados en la aplicación, en cualquier punto del espacio (encima de una mesa, en el suelo, en una silla, etc.).

A pesar del cierre prematuro de este proyecto y la cancelación del soporte por parte de *Google*, sigue existiendo una comunidad de desarrolladores [25] en la que se comparten proyectos y dudas relacionadas con *Tango*, lo cual siempre es un aliciente para que esta tecnología no muera del todo y pueda servir como germen de mejoras y avances para otros proyectos.

2.4 Sensores y Raspberry Pi

Un sensor es un dispositivo que tiene la característica de ser sensible a una determinada magnitud, de tal manera que variando esta magnitud en el entorno se obtiene una medición correspondiente a esa variación en estos dispositivos.

Las características técnicas de un sensor son muchas: rango de medida, precisión, desviación, correlación lineal, sensibilidad, resolución rapidez de respuesta, etc. Pero en este proyecto se tendrán en cuenta dos principales: el rango de medida y la precisión.

La tipología de sensores es tan amplia como lo son sus posibles usos en el mundo real. Existen sensores para medidas de posición línea y angular, de aceleración, presión, caudal, temperatura, sensores de presencia, de luz, acústicos, de proximidad, táctiles o de fuerza (entre otros), por lo que, en función de los objetivos a conseguir, se tiene una amplia gama para seleccionar los más adecuados.

Son muchos los ejemplos existentes de la utilización de sensores en combinación con el dispositivo *Raspberry Pi* [26] ya que este mini computador es capaz de trabajar de forma autónoma sin más requerimientos que una fuente de electricidad y no resulta complejo establecer una conexión con otros dispositivos.

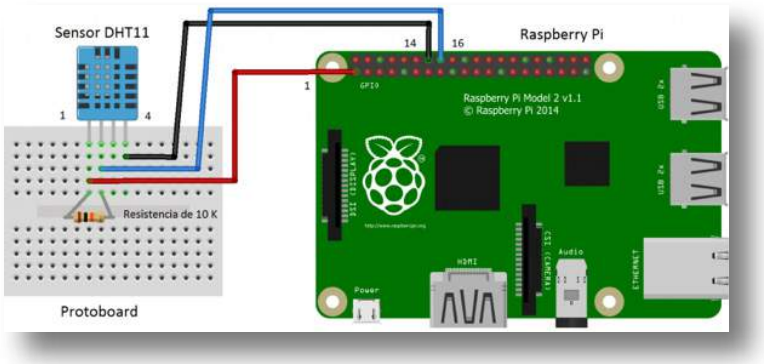

Ilustración 16: Esquema de montaje entre sensor y Raspberry Pi

Una de las formas de obtener la información procedente de los sensores es a través de la librería *Adafuit* [27].

Adafruit es una compañía de hardware de código abierto con sede en Nueva York y fundada por una estudiante del MIT, Limor Fried, en 2005 [28]. Diseña y vende componentes electrónicos de diversos tipos, así como accesorios y herramientas. Posee además gran cantidad de tutoriales,

vídeos y otros recursos para que cualquiera, incluso sin conocimientos avanzados sobre el tema, pueda iniciarse en este ámbito.

Uno de los mayores logros de la compañía ha sido desarrollar una librería capaz de establecer comunicación entre una amplia variedad de sensores y distintos dispositivos (entre ellos Raspberry pi). Estos recursos están abiertos al público para su utilización gratuita y cualquiera puede aprender a utilizarlos [29].

Una de las mayores ventajas de esta librería es que permite realizar un tratamiento y uso de la información compatible con distintos tipos de sensores por lo que proporciona una mayor integración y estandarización. Esto implica que se pueda sustituir o cambiar modelos de sensor con muy poco impacto en el resto del sistema, lo que ayuda a mitigar algunos de los riesgos y problemas de la disponibilidad del sensor y la reutilización del código.

3. ANÁLISIS

Este apartado se centra en el aspecto práctico del desarrollo de la aplicación: qué herramientas se han usado y las diferentes motivaciones para usarlas, la instalación del entorno de desarrollo, la creación de los objetos 3D, así como sus texturas y animaciones. Además, también se incluirán otras decisiones más enfocadas a la usabilidad y diseño de los elementos de la aplicación.

3.1 Realidad aumentada

En este caso se define plataforma como un sistema que proporciona un conjunto de herramientas para trabajar con realidad aumentada. Es importante disponer de una de ellas en este proyecto porque harán posible la definición de elementos tridimensionales en la realidad y la interacción del usuario con ellos.

El hecho de trabajar con realidad aumentada restringe mucho las posibilidades para desarrollar contenido, ya que existen pocas plataformas que lo soporten o que den acceso al paquete de desarrollo. Algunas de estas son:

- *Vuforia*: Es una herramienta muy potente y completa ya que permite desarrollar aplicaciones de realidad aumentada para cualquier soporte o dispositivo móvil (Android, IOS), y tiene versión gratuita cuyas funciones son limitadas y una versión de pago. Su funcionamiento se basa en el reconocimiento de etiquetas (ya sean imágenes, objetos o incluso textos en inglés) para mostrar algún contenido audiovisual como imágenes, vídeos o textos. Estas etiquetas pueden estar en una base de datos local o incluso en la nube (en cuyo caso se incluye una marca de agua en la versión gratuita). Otras de las ventajas que presenta *Vuforia* es el uso de botones virtuales, el escaneo de objetos reales para que se puedan reconocer en realidad aumentada, rastreo de objetivos y una opción que permite reconstruir un terreno en tiempo real de tal forma que se pueda crear un mapa del entorno geométrico en tres dimensione (llamada *Smart TerrainTM*).

Ilustración 17: Uso de marcas con Vuforia en un dispositivo tablet

- *ARToolKit*: Es una librería que ofrece características similares al reconocimiento de etiquetas de *Vuforia*. Utiliza el seguimiento de vídeo para calcular en tiempo real la

posición que tiene la cámara y la orientación relativa de ésta con respecto a esas etiquetas o marcadores físicos. Una vez que sitúa la cámara real es capaz de situar la cámara virtual en el mismo punto y sobreponer los modelos 3D en los marcadores. Una de sus principales ventajas es que se mantiene como un proyecto de código abierto con licencia *GNU General Public License.*

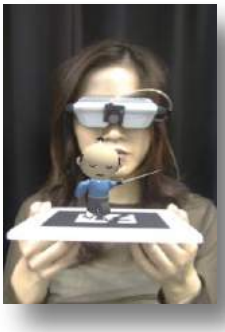

Ilustración 18: Uso de ARToolKit con unas gafas y una tablet

- *ARKit*: Es una característica disponible a partir de iOS 11.0 y que requiere un procesador A9 o superior, y al menos una cámara *iSight* de 8MP, pudiéndose desarrollar tanto para IPhone como para Android (a partir de los modelos *IPhone 6s* y *IPad Pro 2015*). Combina el seguimiento del movimiento del dispositivo, la captura de escenas de la cámara y el procesamiento avanzado de escenas para simplificar la tarea de crear una experiencia de realidad aumentada.

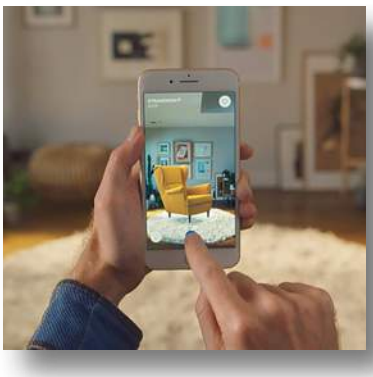

Ilustración 19: Uso de una aplicación desarrollada con ARKit

- *Tango*: Proporciona una amplia librería que permite desarrollar realidad aumentada apoyada en dos dispositivos compatibles que disponen de cámaras de profundidad, *RGB* y de objetivo de ojo de pez para componer en la realidad objetos virtuales. Permite además realizar un seguimiento óptimo del teléfono para recrear objetos virtuales dentro de nuestra realidad de una forma más fidedigna.

Ilustración 20: Uso de una aplicación desarrollada con Tango

A continuación, se muestra una tabla comparativa de las diversas plataformas:

	Vuforia	*ARToolKit*	*ARKit*	*Tango*
Necesita Hardware específico	Android	Cámara o similar	Iphone	Lenovo /Asus
Coste Hardware	-	0-100	> 800 €	> 400 €
Plugin para motores de renderizado	Sí	Sí	Sí	Sí
Licencia gratuita útil	No	Sí	Sí	Si

Tabla 1: Comparativa de las plataformas para 3D

La elección de desarrollar esta aplicación sobre la plataforma *Tango* radica principalmente en la que se quería utilizar una plataforma potente pero que no hubiera sido utilizada hasta entonces por la autora de este trabajo de fin de grado. Así, y habiendo trabajado con otras plataformas de realidad virtual (*Oculus Rift, HTC Vive*) y aumentada (*Microsoft Hololens*), se quería ver las posibilidades de esta nueva tecnología.

Por otro lado, había descartado rápidamente plataformas como el *ARKit* por el componente económico o *Vuforia* por las limitaciones de la versión gratuita de la licencia.

Por último, es importante destacar que la realidad aumentada proporcionada por *Tango* difiere de las demás en la sensación de profundidad ya que el hecho de ejecutarlo sobre dispositivos que tienen tres cámaras ex profeso (una *RGB* de 16 *MPX*, una de profundidad y otra con un

objetivo de ojo de pez) para este tipo de tecnologías, proporciona una experiencia diferente a la de la realidad aumentada en un Smartphone convencional.

3.2 Selección de software de modelado 3D

Un *software* de modelado tridimensional es una herramienta que permite crear y manipular gráficos 3D por ordenador. Al tener como premisa en este trabajo la creación de una aplicación de contenido en realidad aumentada, era vital seleccionar un software adecuado para la creación del contenido en tres dimensiones.

Existen diversas opciones en este ámbito, más o menos especializadas y con diferentes características):

- *ZBrush*: es un *software* de modelado que en sus inicios permitía crear pinturas digitales e insertar en ellas objetos 3D. Sin embargo, tras su uso por parte de diversos estudios cinematográficos, con las siguientes actualizaciones trataron de convertirse en un *software* de referencia para esculpir detallados modelos en tres dimensiones de la misma manera que se pinta en ellos. Hoy en día sigue siendo un *software* de referencia para la escultura y pintura digital.

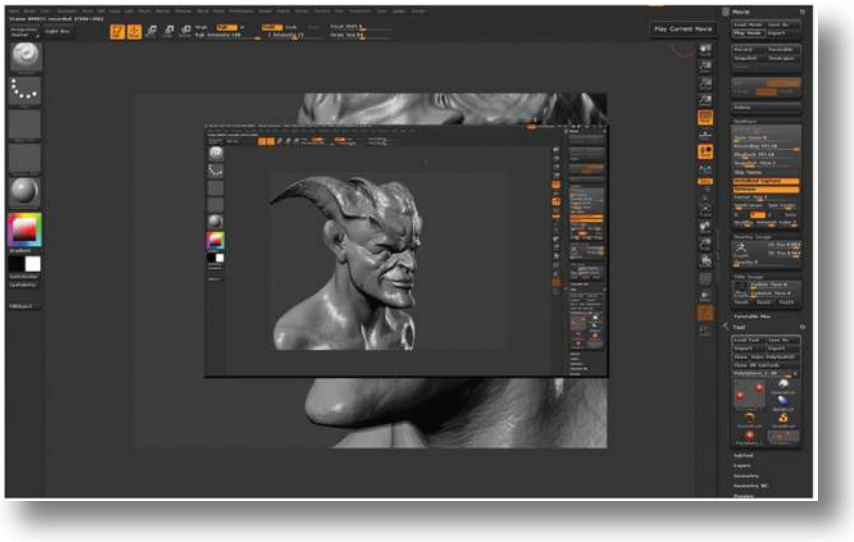

Ilustración 21: Interfaz ZBrush

- *3DMAX*: es uno de los programas de animación más utilizados, especialmente en los ámbitos del desarrollo de videojuegos y arquitectura. Se originó a mediados de los años 80 y sus características se han ido incrementando y mejorando con el paso de los años. Es uno de los mejores softwares del mercado debido a las numerosas posibilidades que presenta, desde el modelado hasta la animación, pasando por la texturización, iluminación y renderizado.

Ilustración 22: Interfaz Autodesk 3DMax

- *Maya*: *Autodesk Maya* es un programa informático dedicado al desarrollo de gráficos 3D, efectos especiales y animación. Tiene una larga trayectoria desde su creación en 1998 en la que poco a poco ha ido evolucionando como un software más potente y de referencia en la industria cinematográfica. Una de las mayores ventajas de *Maya* es que incluye un abanico de posibilidades muy amplio, ya que es posible modelar desde cero cualquier estructura o personaje gracias a las múltiples herramientas que proporciona, añadiéndole texturas, materiales, animaciones, filtros, luces, etc. Como desventaja tiene la de cualquier programa especializado, que es poco intuitivo para alguien que nunca lo ha utilizado y resulta difícil encontrar las herramientas.

Ilustración 23: Interfaz Autodesk Maya

- *Blender*: es un programa multiplataforma dedicado al modelado, renderizado, iluminación, animación y pintura digital. A diferencia de los demás softwares mencionados, *Blender* ofrece una característica nueva y es que permite el desarrollo de videojuegos al poseer un motor de juegos interno. Su principal ventaja es que es un *software* libre, lo que permite trabajar con él a cualquier nivel (ya sea personal o profesional) sin requerir una licencia. A pesar de su popularidad (en gran medida por ser un programa gratuito), es una herramienta compleja sobre todo si se carece de las principales nociones sobre utilizar un *software* 3D.

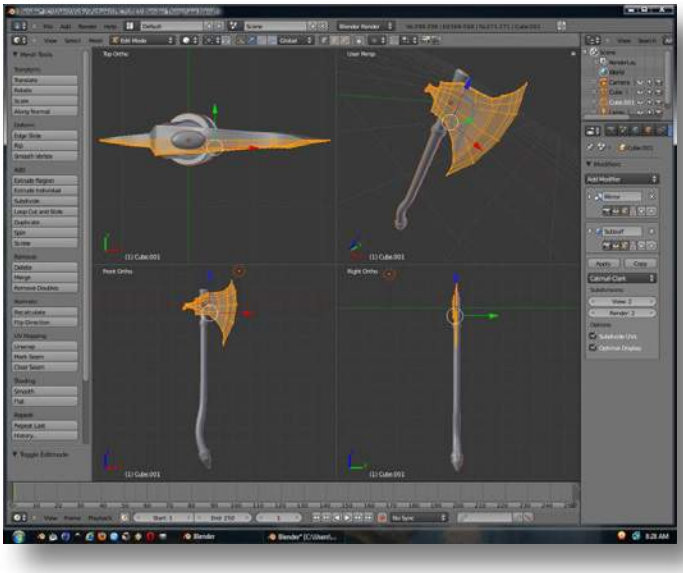

Ilustración 24: Interfaz Blender

La comparativa de los softwares anteriores se muestra a continuación:

	ZBrush	*3DMax*	*Maya*	*Blender*
Experiencia	Poca	Ninguna	Bastante	Poca
Licencia gratuita	No	Sí	Sí	Sí
Exportar en formatos compatibles	Sí	Sí	Sí	Sí
Herramientas adecuadas	No	Sí	Sí	Sí
Facilidad de uso	Sí	No	Sí	No

Tabla 2: Comparativa de los software de 3D

Para la realización de este TFG se ha seleccionado el software de modelado 3D denominado *Maya*. El principal motivo para esta selección de *Maya* es que, como autora de este proyecto,

contaba con experiencia trabajando con dicho programa, además de la licencia gratuita por ser estudiante y un curso de 280 horas que se realizó con anterioridad. Independientemente de esto, y habiéndose trabajado con dos de los otros softwares (*Blender* y *Zbrush*) se ha considerado que el primero no posee las herramientas o no consigue hacerlas intuitivas para que cualquier programador las utilice, por lo que intentar modelar algo en él resulta complicado. En el caso de *Zbrush* es un software que está mucho más enfocado a un modelado puramente orgánico y con procesos similares a la artesanía manual, en la que se va dando forma a una figura con diferentes pinceles y herramientas, no aptas para el modelado estructural que se necesitaba.

En cuanto a *3DMax* tiene características similares a *Maya* (ya que además pertenecen ambos a la empresa *Autodesk*) sin embargo pudiendo obtener los mismos resultados con los dos, se ha preferido utilizar aquel en el que se tiene mayor destreza.

3.3Selección de software de desarrollo

Uno de los pilares fundamentales del desarrollo de una aplicación como la que se presenta en este trabajo de fin de grado, es la codificación de su funcionamiento, establecer una serie de clases que interactúen entre ellas y con los objetos de la escena, para conseguir que se realice la actividad deseada, por lo cual era necesario escoger un programa adecuado.

Para la elaboración de una aplicación para Android tenemos diferentes opciones:

- *Android Studio*: es un entorno de desarrollo integrado para Android, siendo el IDE más utilizado. Tienen una gran ventaja y es que permite visualizar los cambios que realizamos en nuestra aplicación en tiempo real y comprobarlo en las diferentes resoluciones de pantallas de los distintos Smartphone o Tablet. Además, trae preinstaladas diversas plantillas y asistentes para aquellos elementos más comunes de la programación en Android.
- *Unreal*: es un motor de videojuegos creado en 1998, es gratuito (con ciertas limitaciones) desde 2015 y además de estar optimizado para consolas actuales, también soporta el desarrollo para dispositivos de realidad virtual, así como para Smartphone. Por todo eso y por sus múltiples características, se ha convertido en uno de los motores de renderizado más populares del momento. La principal ventaja de este software es que ofrece un control completo de su motor, de tal manera que los usuarios pueden modificarlo e implementar mejoras. Asimismo, ofrece una gran calidad y potencia en los aspectos de iluminación y de creación de materiales y shaders.
- *Unity*: es un motor de juego multiplataforma creado en 2005, que cuenta con una versión licenciada y de pago, y otra gratuita personal. Una de sus mayores ventajas es que es una herramienta muy versátil para el prototipado y presenta una curva de aprendizaje muy fácil, tanto por la estructura de su editor como por el uso de dos lenguajes sencillos como son C# y Javascript. Otra de sus ventajas es que ofrece múltiples plataformas de desarrollo y simplifica mucho la compilación de aplicaciones, ya que simplemente seleccionando aquella plataforma de destino que queremos e instalando los correspondientes archivos necesarios, podemos exportar para *iOS*,

Windows, Android, Oculus Rift, HTC Vive, etc. Además, cuenta con una gran cantidad de documentación y con una comunidad muy activa que responde a dudas y preguntas.

	Android Studio	*Unity*	*Unreal*
Experiencia	Poca	Bastante	Ninguna
Licencia gratuita	Sí	Sí	Sí
Simplicidad en la exportación	Sí	Sí	Sí
Facilidad de uso	Sí	No	No

Tabla 3: Comparativa de los softwares de desarrollo

Tanto *Unity* como *Unreal* contaban con un *SDK de Project Tango* para comenzar a desarrollar la aplicación, sin embargo, se optó por *Unity* principalmente porque es un *software* cuyas herramientas se conoce más a fondo ya que se ha trabajado durante casi un año y medio de forma profesional con él. Se descartó *Android Studio* porque trabajar a nivel gráfico y con objetos en tres dimensiones es más complejo que en cualquiera de las otras dos, además de que la exportación en *Unity* (sea para la plataforma que sea) resulta muy sencilla. También es muy relevante el hecho de que exista una gran comunidad en internet que desarrolle en esta plataforma y que implica que la mayor parte de problemas o errores que surjan, van a tener alguna respuesta en los foros o documentación, lo cual siempre va a disminuir el tiempo dedicado a solucionarlos.

3.4Selección de Raspberry Pi

Una de las premisas de este TFG era conseguir complementar el funcionamiento de la aplicación de realidad aumentada con información en tiempo real proveniente de sensores. Por lo que lo más compacto y fácil de transportar para establecer la comunicación entre los sensores y la aplicación era utilizar un pequeño ordenador denominado *Raspberry Pi*, un computador de placa única y de bajo coste que permite realizar múltiples proyectos utilizando todo tipo de sensores y que no requiere una gran inversión.

La comunicación entre la aplicación móvil y los sensores conectados al mini ordenador era indispensable para mostrar la información de temperatura y humedad del aeropuerto ficticio.

Existen diversos modelos de Raspberry que explicamos a continuación:

- *Raspberry Pi* 1 Modelo A: Surgió en el año 2012 y fue la primera versión. Carecía de entrada ethernet y necesitaba una fuente de alimentación de 5 voltios y 2 amperios (elemento que se mantuvo con los siguientes modelos). La conexión a internet requería de la compra de un adaptador *Wi-Fi* por *USB*, poseía salida de vídeo *HDMI* y vídeo *RCA* y el procesador era *SingleCore* a 700 MHz.

- *Raspberry Pi* 1 Modelo B y B+: Con respecto al modelo anterior aumentó la memoria *RAM* a 512 *MB*, se añadió el conector *ethernet* y un puerto *USB* más.
- *Raspberry Pi* 2 Modelo B: Lanzado en 2014 fue el primer modelo que cambió de procesador, pasa de un núcleo a cuatro y de 700 MHz a 900 *MHz* además de casi duplicar la memoria *RAM* y suprimir la conexión *RCA*.
- *Raspberry Pi* 3 Modelo B: En 2016 renueva procesador una vez más, en este caso sigue siendo de cuatro núcleos, pero pasa de 900 MHz a 1.2 *GHz*. La mayor novedad de este modelo fue la inclusión de *Wi-Fi* y *Bluetooth*.

Existen los modelos *Raspberry Pi Zero* que no fueron considerados debido a que tienen una menor potencia y son mucho más pequeños. Así, aunque para la envergadura de este proyecto habría sido suficiente, si el proyecto se ampliara, dichos modelos podrían no resultar adecuados.

	Modelo 1-A	*Modelos 1-B/B+*	*Modelo 2-B*	*Modelo 3-B*
Dispongo de ella	No	Sí	Sí	No
Suficientemente potente	Sí	Sí	Sí	Sí
Dispone de red	No	Sí	Sí	Sí
Gran cantidad de puertos	No	No	Sí	Sí

Tabla 4: Comparativa de los modelos de Raspberry Pi

En el momento de idear el trabajo de fin de grado ya se tenían dos de estos dispositivos con los que poder trabajar: la *Raspberry Pi* 1 modelo B y la *Raspberry Pi* 2 modelo B. De esta forma una vez comprobado que los dos dispositivos servían para la implementación de los sensores y la comunicación con la aplicación, se decidió utilizar el modelo más reciente (2-B) porque proporcionaba más opciones de conexión y en un principio se desconocía si sólo se utilizaría un sensor o varios, por lo que la opción más completa proporcionaba una mayor libertad en este caso.

3.5Selección de sensores

El hecho de querer mostrar información en tiempo real de condiciones a las que no se tiene acceso a priori (como la temperatura y la humedad) implicaba tener que adquirir un dispositivo que pudiera acceder a esa información y hacerla visible de alguna manera. Es aquí donde resulta necesario el uso de sensores capaces de determinar qué condiciones ambientales se tienen y cómo comunicarlo a través de la *Raspberry Pi* hasta llegar a la aplicación.

En cuanto a la elección de los sensores, se tuvo claro desde el primer momento que la elección se ceñiría a dos modelos principales, como consecuencia de tres requisitos iniciales: que estuvieran disponibles rápidamente (a través de la venta en internet de Amazon), que fueran sencillos de instalar y que existiera algún tutorial o guía para hacerlo.

Este último requisito ha sido el que ha inclinado la balanza por estos dos modelos:

- *DHT11*: La principal característica es su bajo coste. Es un buen sensor para lecturas de humedad de entre el 20% y el 80%, así como de temperaturas entre 0 y 50 grados con dos grados de precisión. Realiza el muestreo una vez por segundo usa 2.5mA mientas solicita datos.
- *DHT22*: También se considera un sensor de bajo coste, aunque menos que el anterior, la mayor diferencia entre ambos radica en la precisión de las lecturas principalmente. En este caso podemos obtener humedades entre el 0% y el 100% y temperaturas entre menos 40 y 80 grados con medio grado de precisión.

En este caso se seleccionó el sensor DHT22 porque proporcionaba un rango más amplio de mediciones, así como más fiables y con un error menor. Además, la diferencia en el coste entre ambos era prácticamente inexistente.

3.6Requisitos del sistema

Dentro de la fase de análisis de este proyecto, se encuentran también recogidos los requisitos del sistema. Éstos se definen para diseñar correctamente la aplicación, de manera que satisfaga todas las necesidades planteadas. Se pueden dividir en requisitos funcionales y no funcionales, dependiendo del área a la que afecten:

3.6.1 Requisitos Funcionales

Los requisitos funcionales son aquellos que recogen una funcionalidad concreta.

Código	RF-01
Descripción	La aplicación debe abrir la cámara del dispositivo una vez se inicie.

Tabla 5: Requisito Funcional del sistema RF-01

Código	RF-02
Descripción	La aplicación debe permitir colocar la terminal T4 tridimensional en cualquier punto del espacio que se vea a través de la cámara, siempre y cuando esté despejado y permita la horizontalidad.

Tabla 6: Requisito Funcional del sistema RF-02

Código	RF-03
Descripción	La aplicación debe permitir mover, rotar y trasladar al menos en un sentido, la terminal T4 para que el usuario la ajuste a su punto de vista.

Tabla 7: Requisito Funcional del sistema RF-03

Código	RF-04
Descripción	La aplicación debe permitir ver los vuelos de salida y llegada mediante la pulsación de los botones correspondientes (Salidas y Llegadas) en la última pantalla.

Tabla 8: Requisito Funcional del sistema RF-04

Código	RF-05
Descripción	La aplicación debe permitir la visualización de las animaciones correspondientes a los despegues y aterrizajes de los aviones.

Tabla 9: Requisito Funcional del sistema RF-05

Código	RF-06
Descripción	La aplicación debe permitir visualizar la información de cada vuelo cuando el usuario toca en la pantalla el cuerpo de cualquier avión, ya sea en movimiento o estacionado).

Tabla 10: Requisito Funcional del sistema RF-06

Código	RF-07
Descripción	La aplicación debe permitir visualizar la información que proporcionan los sensores en tiempo real, tanto de la temperatura como de la humedad.

Tabla 11: Requisito Funcional del sistema RF-07

Código	RF-08
Descripción	La aplicación debe permitir volver a la pantalla inicial desde cualquiera de las otras pantallas durante el hilo de ejecución, para restaurar el funcionamiento de la misma.

Tabla 12: Requisito Funcional del sistema RF-08

3.1.1 Requisitos no Funcionales

Los requisitos no funcionales hacen referencia a aquellas funcionalidades del sistema que afectan al ámbito técnico pero que no tienen por qué ser conocidos por el usuario.

Código	RNF-01
Descripción	La aplicación debe generar los vuelos, tanto de salida como de llegada, en los 30 minutos antes y después de la hora en la que se encuentra la aplicación ejecutándose.

Tabla 13: Requisito no Funcional del sistema RNF-01

Código	RNF-02
Descripción	El tiempo de respuesta al cargar la pantalla final donde se ejecutan las animaciones y se realiza la comunicación con el sensor debe ser el menor posible.

Tabla 14: Requisito no Funcional del sistema RNF-02

Código	RNF-03
Descripción	La aplicación debe mostrar como información de cada uno de los vuelos de salida lo siguiente: hora de salida, destino e identificador de vuelo. En el caso de los vuelos de llegada la información debe ser esta: hora de llegada, origen e identificador de vuelo.

Tabla 15: Requisito no Funcional del sistema RNF-03

Código	RNF-04
Descripción	La aplicación debe realizar una única comunicación con los sensores al cargar la pantalla final, ya que una continua consulta en cada fotograma consumirá demasiados recursos y ralentizará el flujo de ejecución de la aplicación.

Tabla 16: Requisito no Funcional del sistema RNF-04

3.7Casos de uso

En este apartado se describen los casos de uso de la aplicación HoloT4, que se detallan en el siguiente diagrama:

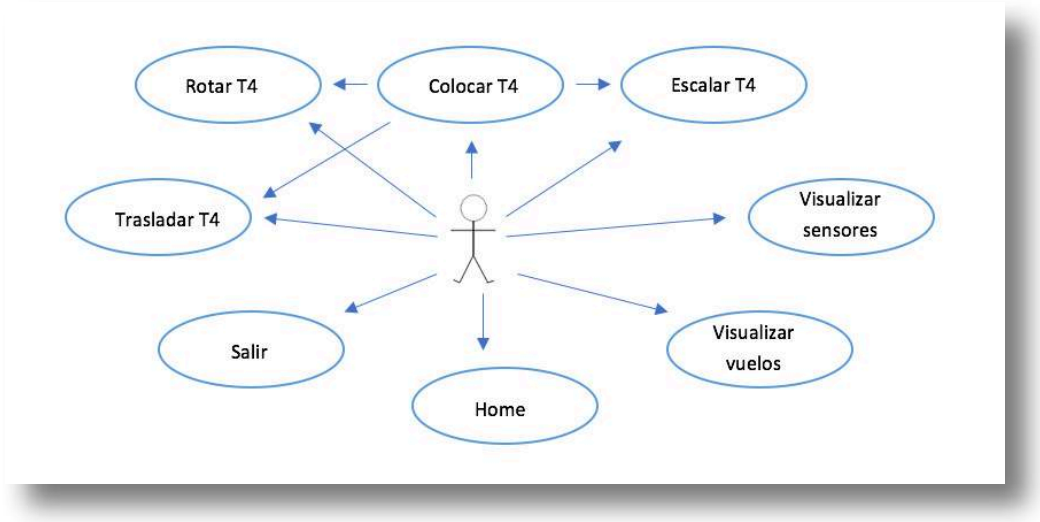

Ilustración 25: Diagrama de casos de uso

A continuación, se describen más detenidamente los casos de uso:

Código	CU-01
Actor	Usuario
Descripción	El usuario coloca la holografía de la terminal T4
Precondiciones	Haber iniciado la aplicación
Postcondiciones	Aparece la terminal y se cambia de pantalla

Tabla 17: Detalle del caso de uso CU-01

Código	CU-02
Actor	Usuario
Descripción	El usuario traslada la holografía de la terminal T4
Precondiciones	Haber colocado la terminal T4
Postcondiciones	La terminal se desplaza en el eje X

Tabla 18: Detalle del caso de uso CU-02

Código	CU-03
Actor	Usuario
Descripción	El usuario rota la holografía de la terminal T4
Precondiciones	Haber colocado la terminal T4
Postcondiciones	La terminal gira sobre el eje Y

Tabla 19: Detalle del caso de uso CU-03

Código	CU-04
Actor	Usuario
Descripción	El usuario escala la holografía de la terminal T4
Precondiciones	Haber colocado la terminal T4
Postcondiciones	La terminal aumenta o disminuye de tamaño en los tres ejes (X, Y, Z)

Tabla 20: Detalle del caso de uso CU-04

Código	CU-05
Actor	Usuario
Descripción	El usuario visualiza la información de temperatura y humedad
Precondiciones	Haber fijado la posición de la terminal T4
Postcondiciones	Ninguna

Tabla 21: Detalle del caso de uso CU-05

Código	CU-06
Actor	Usuario
Descripción	El usuario toca la pantalla en la posición donde se encuentra un avión
Precondiciones	Haber fijado la posición de la terminal T4
Postcondiciones	El usuario visualiza la información de vuelo de ese avión

Tabla 22: Detalle del caso de uso CU-06

Código	CU-07
Actor	Usuario
Descripción	El usuario presiona el botón táctil de salidas/llegadas
Precondiciones	Haber fijado la posición de la terminal T4
Postcondiciones	El usuario visualiza la información de los vuelos de salida/llegada

Tabla 23: Detalle del caso de uso CU-07

Código	CU-08
Actor	Usuario
Descripción	El usuario presiona el botón táctil de Home
Precondiciones	Haber iniciado la aplicación
Postcondiciones	El usuario regresa a la pantalla inicial de la aplicación

Tabla 24: Detalle del caso de uso CU-08

Código	CU-09
Actor	Usuario
Descripción	El usuario presiona el botón táctil Salir
Precondiciones	Ninguna
Postcondiciones	La aplicación se cierra

Tabla 25: Detalle del caso de uso CU-09

4. DISEÑO

4.1 Diseño Visual

A lo largo de todo el diseño de la aplicación se ha querido mantener cierta coherencia tanto con los colores usados, como con los tipos de botones, sombreados y tipografías para que estos detalles no rompieran el hilo conductor de la ejecución.

El modelado de la terminal cuatro se ha ido realizando acorde con las fotos obtenidas desde la aplicación *Google Earth* que permiten un acercamiento bastante preciso a las estructuras de la misma. Además, se han consultado diversas webs de fotografías para verificar determinados detalles tanto del modelado como del texturizado. No se ha podido ser 100% realista porque la extensión de la terminal es muy grande en comparación con el espacio que se puede tener para ejecutar la aplicación, pero en todo lo demás se ha intentado ser fiel al modelo real (en formas, texturas y colores):

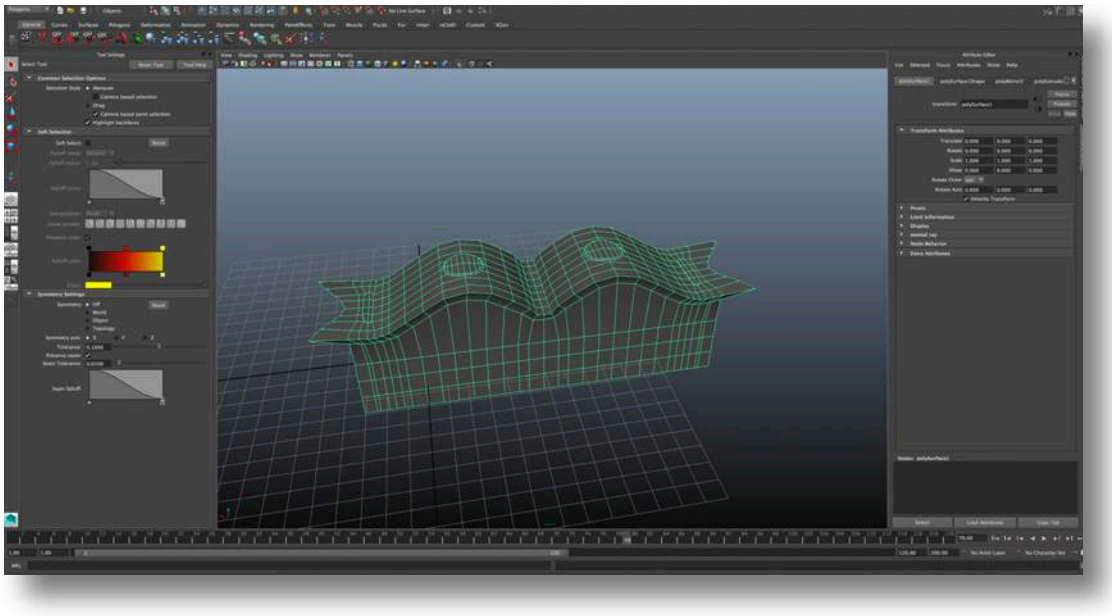

Ilustración 26: Módulo de la terminal T4 modelado en Maya

Para la pantalla inicial se quería un fondo sencillo relacionado con la temática de la aplicación, lo cual ya delimitaba un poco las posibilidades. Finalmente se escogió un fondo azul con nubes y la imagen de uno de los edificios de la terminal T4 para dar una pista acerca del contenido de la aplicación.

Lo mismo ocurre con la elección del nombre, dado que el contenido es algo holográfico se ha optado por la sencillez de un nombre fácil de recordar y descriptivo como es HoloT4. La distribución de los botones sobre la pantalla inicial responde al criterio de una mayor visibilidad ya que esos botones suponen las dos únicas formas iniciales de interactuar con la aplicación.

Ilustración 27: Pantalla inicial aplicación HoloT4

En la siguiente pantalla se abre la cámara del móvil y se muestra un mensaje superpuesto que indica al usuario lo que debe hacer. En un primer momento esta pantalla no existía, sin embargo, faltaba esa información y para el usuario no habría resultado nada intuitivo su funcionamiento por lo que se decidió añadirla.

Ilustración 28: Segunda pantalla de la aplicación HoloT4, donde se abre la cámara

En la siguiente pantalla se puede ver que se ha emplazado el aeropuerto 3D y aparecen ocho botones. Puede notarse que se ha cambiado la orientación del dispositivo para visualizar mejor el contenido.

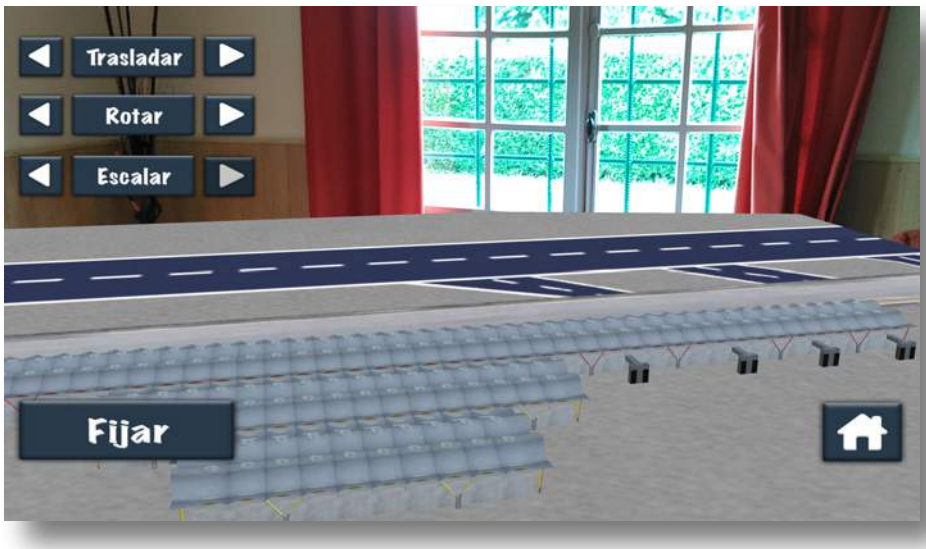

Ilustración 29: Tercera pantalla de la aplicación HoloT4

Los seis primeros, se han situado a la izquierda en la parte superior porque son un conjunto de botones que sólo tienen sentido si están juntos y dado su tamaño no era conveniente ponerlo en otra zona, ya que dificultaría la visibilidad de la terminal por parte del usuario. Estos seis botones permiten realizar unos cambios limitados sobre la posición del aeropuerto, por ejemplo, la traslación sólo se realiza sobre el eje X porque no tendría sentido añadir todas las opciones posibles, llenaríamos la pantalla de botones que al usuario tampoco le resultan excesivamente útiles. En el caso de la rotación, ésta se realiza sobre el eje Y porque rotarla en alguno de los otros ejes haría que no pudiéramos verla en su totalidad o incluso que se viera invertida, lo cual carece de sentido. En cuanto al escalado se ha realizado sobre los tres ejes simultáneamente para que no exista un alargamiento o acortamiento excesivo en uno de ellos y se distorsione la forma.

Las medidas tomadas para las tres acciones han sido con pequeños valores, para que haya un buen intervalo en el cual el usuario pueda ajustar el objeto en la realidad.

Ilustración 30: Imagen de un botón de la aplicación

En lo que respecta a los otros dos botones, ambos sirven para cambiar de pantalla, por lo que se ha creído conveniente situarlos ahí. El botón de retorno a la escena inicial tiene un icono de una casa para hacer alusión al típico botón Home de cualquier aplicación o juego, recurriendo a él cuando queramos reiniciar la ejecución o la posición la terminal.

El botón Fijar realiza varias acciones, una de ellas es cambiar de pantalla (ocultando la actual y haciendo visible la siguiente), otra de ellas es fijar la terminal de tal manera que ya no es posible modificar su posición y la última es lanzar toda la ejecución de los aviones, desde hacerlos visibles hasta hacer que se gestionen las colas que controlan el sistema e iniciar sus animaciones.

Una vez hemos accionado Fijar, podemos ver cómo aparecen los aviones en la terminal y en unos segundos alguno de los aviones (aleatoriamente de salida o llegada) comienza a ejecutar su animación. El hecho de que los aviones no aparecieran hasta que no se cambiaba de pantalla se hizo a propósito ya que resulta mucho más visual que coloquemos la estructura primero, y una vez vayamos a iniciar la ejecución, los aviones se coloquen y animen sobre esa estructura ya fijada.

Los aviones han sido el único elemento incluido que no ha sido modelado por la autora de este TFG, ha sido un recurso de internet cuya licencia gratuita permite su utilización [30]. Esto ha sido motivado principalmente por la funcionalidad y la rapidez que ofrecen este tipo de recursos, ya que el haber utilizado uno propio habría retrasado la evolución del proyecto en al menos unos días más.

Ilustración 31: Render del modelado de avión utilizado en la aplicación

En la pantalla final, tenemos una distribución similar a la escena anterior, dos botones a cada lado, tanto en la parte superior como en la parte inferior:

Ilustración 32: Pantalla final de la aplicación HoloT4

Los dos botones que representan las salidas y las llegadas de los aviones son interactuables, de tal manera que cada vez que los pulsamos podemos ver u ocultar la información relativa a los vuelos. La tipología de la información de los vuelos cambia con respecto a la que manteníamos hasta ahora, así como su color, esto es debido a que se intentaba conseguir una mayor similitud con las pantallas reales de los aeropuertos, donde la información de los vuelos aparece en amarillo:

Ilustración 33: Detalle de los vuelos de salida generados

En cuanto a la información de los sensores, ésta se muestra a través de una pantalla diseñada como si fuera un monitor con un tipo de letra muy similar a la de cualquier pantalla electrónica. El hecho de que los números sean de color rojo rompe con la similitud de colores utilizados, pero consideraba que era necesario para que produjese una mayor atención por parte del usuario.

Ilustración 34: Detalle de la temperatura y humedad mostrados por el sensor

Se mantiene al igual que en la pantalla anterior, el botón Home en el mismo lugar para poder reiniciar la aplicación en caso necesario.

4.2 Diseño Técnico

En este apartado se explicará el funcionamiento de la aplicación y cómo se ejecuta paso por paso. Para ampliar los detalles técnicos se puede acudir a los Anexos I, II, III y IV donde se detallarán en profundidad algunos de los apartados más importantes.

4.2.1 Instalación

Al compilar la aplicación desde Unity obtenemos un ejecutable para Android con la extensión .apk, que una vez clicado la instalará en nuestro dispositivo, informándonos que la aplicación no necesita ningún permiso especial para su ejecución.

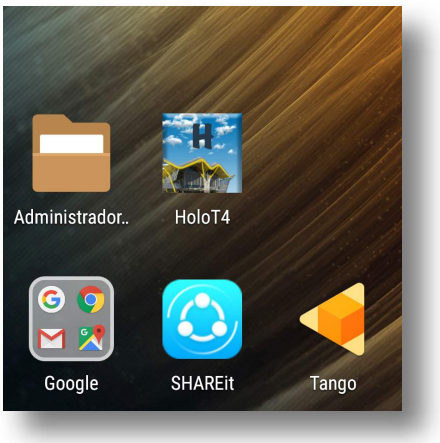

Ilustración 35: Icono de la aplicación HoloT4

4.2.2 Escena inicial

Unity permite la creación de diversos escenarios tridimensionales y el salto entre estos durante la ejecución por lo que la pantalla inicial se concibió como una sola escena, aparte del resto del contenido por diversos motivos. Uno de ellos responde a la idea de que no requerimos la existencia de una cámara que recoja la información del entorno, por lo tanto, aquí se usa la cámara virtual propia de cualquier proyecto de *Unity*, que simplemente muestra el contenido de una determinada área. Otro de los motivos para esta separación se basa en no tener una saturación de componentes interconectados en una misma escena, ya que eso complica la ejecución y dificulta localizar los problemas que puedan surgir.

La composición de esta escena consiste en una imagen de fondo compuesta por el edificio de la terminal 4 del aeropuerto de Madrid-Barajas, un cielo azul, una serie de nubes blancas y el nombre de la aplicación, que se incluyen como una sola imagen dentro de un panel creado como base.

Los otros dos elementos principales de la escena serían los botones: Iniciar y Salir, cada uno de ellos configurado de forma similar, pero que ejecuta acciones diferentes. *Unity* permite la creación de botones predefinidos cuyo funcionamiento va a responder al *script* que se le adjunte. En este caso, el botón Iniciar realizaría la acción de cambiar a la escena principal de la aplicación, y el botón Salir, que saldría de la ejecución.

Ilustración 36: Pantalla inicial

4.2.3 Escena principal

Para configurar esta segunda escena era importante ocultar o hacer visibles ciertos conjuntos de elementos, para que, el hilo conductor de la ejecución tuviera sentido. Esto dio lugar a la creación de tres pantallas, cada una con unos elementos concretos y funciones diferentes, y sucediéndose durante la ejecución.

Ilustración 37: Segunda pantalla

Al iniciarse esta escena se abre la cámara del dispositivo con un mensaje sobre impreso en ella que insta al usuario a que localice un lugar donde quiera emplazar el aeropuerto (una mesa, el suelo, un lugar plano) para que toque la pantalla y se coloque la maqueta tridimensional. Es importante destacar que, tal y como está codificado, el sistema buscará la identificación entre el punto que el usuario ha tocado en la pantalla con el mundo real, estableciendo un eje perpendicular a ese punto y fijando ahí el objeto 3D. Si no se colocara en el primer intento significaría que el sistema no ha encontrado un punto perpendicular claro por lo que se debe realizar varios intentos hasta conseguir emplazar la terminal 4.

En este momento de la ejecución se producen dos acciones, por un lado, el cambio a otra pantalla (momento en el que la pantalla mensaje pasa a estar oculta y se hace visible la siguiente) y por otro, la instanciación del objeto en tres dimensiones, siempre con la misma rotación predefinida, de forma que el usuario la verá siempre en este punto de la ejecución desde el mismo ángulo.

Ilustración 38: Tercera pantalla

Una vez colocada la terminal se muestran una serie de botones en la parte superior izquierda que permitirán al usuario realizar cambios en la posición del objeto 3D:

- Trasladar: sobre el plano X.
- Rotar: en sentido horario o anti horario sobre el eje Y.
- Escalar: proporcionalmente en los tres ejes (X, Y, Z).

De esta manera el usuario puede adaptar el objeto tridimensional al espacio del que dispone para visualizarlo correctamente. A través del código esto se controla con la función desplazarT4, que es llamada en cada fotograma de la ejecución de esta pantalla, de tal manera que, si se pulsa alguno de los botones, se ejecutan las acciones asociadas:

Los valores escogidos responden a las diferentes pruebas realizadas, ya que el ajuste debe ser lo suficientemente preciso como para que la terminal se coloque en la posición deseada por el usuario.

En la parte inferior de la pantalla se encuentran dos botones más, Fijar y Home, que con el icono de una casa simboliza el retorno a la pantalla de inicio. Esta opción es útil porque permite reiniciar la aplicación y colocar la terminal en otra zona distinta.

Si, por el contrario, al usuario le gusta la colocación que ha hecho del objeto tridimensional, se pulsaría el botón fijar, que realiza a su vez tres funciones: generar la información de ocho vuelos de salida y ocho vuelos de llegada, fijar la terminal en el punto en el que el usuario la ha ajustado y cambiar a la siguiente pantalla.

Ilustración 39: Cuarta pantalla

Es durante la ejecución de esta pantalla donde la aplicación reviste mayor complejidad.

El sistema de generación de vuelos escoge aleatoriamente entre más de veinte destinos y códigos de vuelo, utilizando la hora en la que se está ejecutando la aplicación para que todos los vuelos despeguen o aterricen en los minutos cercanos.

En la parte superior se observan dos botones, Salidas y Llegadas, en los que el usuario puede ver esos vuelos que se han generado anteriormente, tal y como ocurre en las pantallas de un aeropuerto (aunque omitiendo información irrelevante para el objetivo de esta aplicación, como por ejemplo la puerta de embarque o la compañía aérea que opera el vuelo).

A medida que pasan los segundos se puede observar como comienzan a moverse los aviones, para ejecutar el despegue o el aterrizaje y aparcamiento en la terminal.

Otra de las opciones que plantea esta aplicación es conocer qué información corresponde a cada vuelo, y eso puede saberse tocando la pantalla directamente encima de cada avión.

Ilustración 40: Detalle de la información que muestran los aviones al tocarlos

Con un toque sobre el cuerpo central de los mismos, aparece en la parte trasera la misma información que se tiene en las pantallas de salida y llegada: hora, destino u origen e identificador de vuelo. La diferencia en este caso es que se ha utilizado el color rojo para que sea más visible al estar superpuesta. Con un segundo toque sobre el cuerpo del avión, esta información desaparece.

Es en esta pantalla donde transcurre todo el resto de la ejecución, una vez terminado el despegue y aterrizaje de todos los aviones se detendría y habría que reiniciarla para poder observar nuevos vuelos.

En cuanto a la información proporcionada por los sensores, ésta se muestra en la parte inferior, en primer lugar, aparece la temperatura y debajo la humedad. La medición se realiza en el momento en el que se pasa a esta tercera pantalla, y no se actualiza a no ser que se reinicie la aplicación, ya que en la fase de pruebas se ha intentado una actualización cada pocos segundos, pero sobrecargaba demasiado al sistema y no se conseguía una ejecución fluida.

La forma en la que esta información es mostrada por pantalla se consigue accediendo a la Raspberry Pi a través de una conexión SSH (Secure Shell) [31] y se lanzan los comandos adecuados que utilizan la librería Adafruit para la medición de los sensores. Una vez estos devuelvan la información, el sistema la muestra por pantalla.

5. PRESUPUESTO

En este apartado se detallan los costes de materiales, personal y generales para el desarrollo de este proyecto. El coste de material a su vez se dividirá en el coste *hardware* y en el coste *software* y el coste global tendrá en cuenta todos los costes del proyecto.

5.1 Coste hardware

Concepto	Cantidad	Coste	Total
iMac 27” procesador 3.4GHz Intel Core i7	1	1.950 €	1.950 €
Memoria DDR3 16 GB 1333 MHz	4	20 €	80 €
SSD Samsung 850 EVO 500 GB	1	70 €	70 €
Ratón	1	89 €	89 €
Teclado	1	119 €	119 €
Lenovo Phab 2 Pro	1	424 €	424 €
Raspberry Pi	1	34 €	34 €
Carcasa Raspberry Pi	1	7 €	7 €
Sensor DHT22	1	9 €	9 €
Cables de conexión sensor	3	0 €	0 €
TOTAL			**2.782 €**

Tabla 26: Detalle del coste de hardware

5.2Coste software

Concepto	Cantidad	Coste	Total
Licencia Autodesk Maya 2015	1/mensual	248,05 €	248,05 €
Licencia Unity 3D Plus	4/mensual	21,42 €	85,68 €
Licencia Microsoft Office 2016	1	279 €	279 €
Almacenamiento Drive	1	0 €	0 €
TOTAL			**612,73 €**

Tabla 27: Detalle del coste de software

***Nota:** Se presupuesta con carácter general, aunque en este proyecto se hayan utilizado licencias gratuitas para estudiantes de *Autodesk Maya, Unity 3D y Microsoft Word 365*. En *Unity* se escoge la licencia Plus, si este proyecto se desarrollara en una compañía con unos ingresos superiores a 200.000 dólares, sería necesario obtener *Unity Pro*.

5.3Coste personal

En cuanto al coste de personal se muestra a continuación con sus correspondientes horas asignadas y el coste por hora determinado por convenio:

Concepto	Horas	Coste/Hora	Total
Jefe de Proyecto	120	25 €	3.000 €
Modelador 3D	60	10 €	600 €
Diseñador	40	15 €	600 €
Programador	290	9 €	2.610 €
Tester	40	9 €	360 €
TOTAL			**7.170 €**

Tabla 28: Detalle del coste personal

5.4Costes generales

Reflejamos como costes generales todos aquellos derivados de la actividad de este proyecto de forma proporcional al tiempo invertido en él:

Concepto	Total
Desplazamientos	310 €
Dietas	70 €
Material de Oficina	40 €
Conexión a Internet	160 €
Suministros (agua, luz)	225 €
TOTAL	**805€**

Tabla 29: Detalle de costes generales

5.5Coste global

A continuación, obtenemos el total del coste del proyecto descrito como coste global.

Costes Hardware	2.782 €
Costes Software	613 €
Costes Personal	7.170 €
Costes Generales	805 €
TOTAL	**11.370 €**

Tabla 30: Detalle del coste global

La realización del proyecto asciende por tanto a la cifra de once mil trescientos setenta euros.

6. PLANIFICACIÓN

La planificación que se ha seguido para la realización de este proyecto tenía una estructura inicial que se ha visto influenciada por las necesidades, tanto documentales como de desarrollo, que han ido surgiendo. A continuación, mostramos el esquema real con las diferentes fases del mismo y su avance en el tiempo.

6.1Detalle

Se han dedicado en total ochenta y cinco días, parte a tiempo completo y parte a tiempo parcial. La parte a tiempo completo se ha consumido alrededor de diez horas diarias, en la otra parte sobre cuatro, ya que se compaginó con el trabajo diario. Las fases que más tiempo han requerido han sido principalmente la elaboración de la documentación y la parte más compleja del funcionamiento de la aplicación, que sería la segunda pantalla, con todos los elementos que se interrelacionan entre sí (los aviones, sus paneles, las animaciones, el flujo de ejecución entre las pantallas de la misma escena, etc.).

Asimismo, las tareas que menor cantidad de tiempo han consumido son aquellas referentes a la idea inicial, su investigación, la instalación de los entornos de desarrollo, el modelado de los objetos 3D y la implementación de los sensores en la *Raspberry Pi*.

Existe un parón durante algunos meses motivado por la acumulación de trabajo y la preparación para la evaluación de alguna asignatura de la universidad.

6.2Tareas

Las tareas desarrolladas durante este proyecto y que están recogidas en el diagrama de Gant de este apartado son las siguientes:

- Elección de la idea: comprende la primera parte del proyecto, en la que se fueron sucediendo varias ideas y de entre las que se eligió la definitiva.
- Investigación, desarrollo y viabilidad de la idea: una vez seleccionada la idea, se ha investigado si era viable, qué partes podían desarrollarse y cuales era necesario descartar, si se podía realizar con información obtenida en tiempo real o no, qué software sería necesario utilizar para la realización de la aplicación, etc.
- Instalación entornos de desarrollo: aglutina todo el proceso de instalación de programas y entornos para poder llevar a cabo este proyecto en el equipo del que se dispone.
- Modelado 3D: en esta parte se lleva a cabo el modelado de todos los elementos 3D utilizados en la aplicación (terminal, pista de aterrizaje, aviones, paneles, etc.).
- Texturización modelos 3D: consiste en crear texturas para colocar sobre los objetos en tridimensionales.
- Memoria: en esta parte se documenta todo el trabajo realizado, se ido realizando por fases, a medida que se terminaba determinados apartados se completaba también la parte relacionada en la memoria.

- Creación elementos gráficos: consiste en la creación de los fondos, iconos, botones, paneles e imágenes que aparecen en la aplicación.
- Composición escena inicial: condensa la creación de la primera escena en Unity, utilizando elementos gráficos creados anteriormente y comenzando con el flujo de ejecución de la aplicación.
- Composición escena secundaria: en esta parte se desarrolla la parte más importante de la aplicación, se introducen los elementos en tres dimensiones y se controla el funcionamiento de todo el conjunto.
- Animaciones: comprende principalmente las animaciones de los aviones, como se generan, cuando se lanzan y en qué momento se terminan.
- Conexión con *Raspberry Pi*: en esta parte se trabaja con la *Rapsberry Pi* y el sensor de temperatura y humedad, asimismo se realizan las pruebas de comunicación necesarias con la aplicación.
- Revisión memoria final: consiste en realizar una revisión de toda la documentación creada para cambiar todos aquellos aspectos necesarios.
- Preparación presentación: aglutina la realización de las diapositivas para la presentación, así como el estudio y preparación de la misma.

A continuación, mostramos el diagrama de Gantt correspondiente, dividido en dos partes para poder visualizar mejor el tiempo de cada etapa:

Entre el 7 de enero y el 8 de marzo:

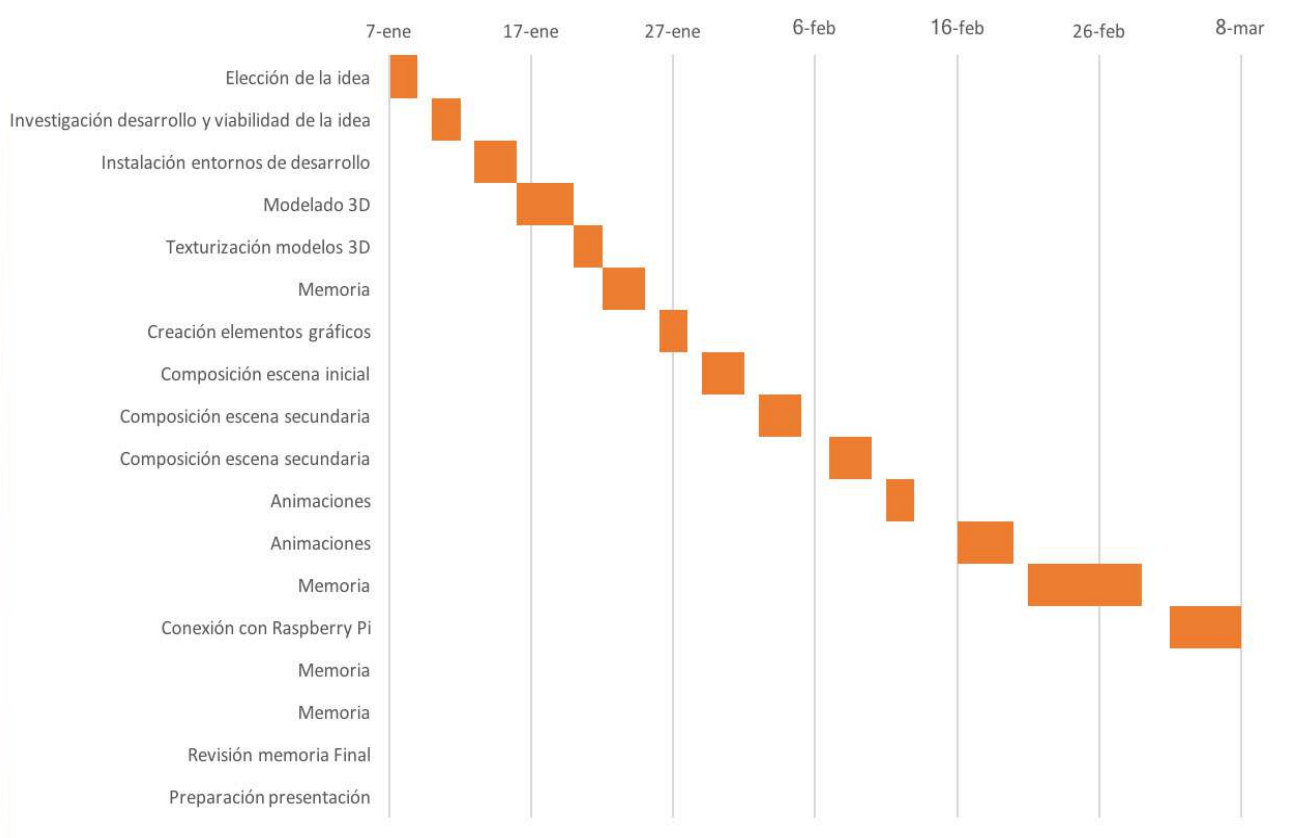

Ilustración 41: Detalle de la planificación entre enero y marzo

Entre el 18 de marzo y el 6 de junio:

Ilustración 42: Detalle de la planificación entre marzo y junio

7. CONCLUSIONES Y TRABAJOS FUTUROS

En este apartado se establecerán las conclusiones de este proyecto, así como se detallarán todas aquellas mejoras que se podrían aplicar a partir del trabajo ya realizado.

7.1Conclusiones

Retomando el principio de este TFG, se puede comprobar que se han cumplido los objetivos iniciales establecidos, y que el funcionamiento de la aplicación que se quería crear es correcto.

Las decisiones tomadas durante la realización de la misma, han resultado ser adecuadas ya que se han conseguido unos buenos tiempos de respuesta, fluidez en la ejecución, un diseño adaptado a las dos orientaciones (vertical y horizontal), que el usuario sea capaz de interactuar con la aplicación de forma sencilla e intuitiva y en definitiva, que resulte interesante.

La parte más compleja de este trabajo ha sido relacionar todos los sistemas que intervienen en el funcionamiento para que se suceda el flujo de ejecución de la manera adecuada: que las animaciones de los aviones se inicien cuando es necesario, que se alternen aleatoriamente entre los vuelos de salidas y llegadas, que se produzca la salida y aterrizaje de cada vuelo sin entorpecer a ningún otro, que se generen los datos de los vuelos de salida y llegada, etc.

Tampoco ha sido fácil estructurar todo el contenido generado sin olvidarse los aspectos decisivos, que desde el punto de vista del desarrollador están implícitos pero que es importante que estén presentes en la documentación. Sin embargo, el hecho de tener que documentar todos los pasos dados y explicar concienzudamente el proceso también ayuda a analizarlo y a conocerlo más profundamente, por lo que esta parte no ha hecho más que mejorar las capacidades de esta autora en lo que a esas competencias se refiere.

En cuanto al desarrollo técnico, este proyecto ha servido para reforzar y aumentar en gran medida los conocimientos de la autora sobre la programación en C# así como su integración en el motor de renderizado Unity 3D. Asimismo, ha afianzado otros sobre diseño y usabilidad, y mejorado competencias como la destreza en Photoshop. La parte que involucra a los sensores y su conexión con la Raspberry Pi ha sido muy ilustrativa, ya que ha servido no sólo para sacar adelante este trabajo, sino también para generar nuevas ideas y proyectos para desarrollar en el futuro.

A nivel personal también ha supuesto todo un reto, ya que hace meses no era probable que la autora de este trabajo fuera capaz de desarrollar una aplicación de esta envergadura, sin muchos de los conocimientos que ahora sí se tienen. Por otro lado, se ha tenido que compaginar muchas horas de trabajo semanales con la realización de este proyecto, por lo que no ha sido fácil llevarlo a cabo.

A pesar de la gran cantidad de tiempo dedicado, el balance de este trabajo ha sido satisfactorio, principalmente por todo el aprendizaje realizado sobre tecnologías de las que la autora de este

trabajo no había trabajado (como el uso de sensores conectados a una *Raspberry Pi* y su comunicación con una aplicación móvil) y el afianzamiento de otras que conocía y sabía cómo funcionaban, pero no había desarrollado tan ampliamente (una aplicación móvil en *Unity* utilizando realidad aumentada).

7.2Trabajos futuros

A partir del proyecto que se ha planteado podría desarrollarse una aplicación más compleja que proporcionara un servicio mejor y más útil con el uso de datos reales y su continua actualización. Por otro lado, podríamos mejorar la usabilidad introduciendo un manual de uso rápido sobre el funcionamiento de la aplicación al que pudiera acceder el usuario en caso de necesitarlo.

Se pueden resumir estos cambios técnicos en los siguientes puntos:

- Cambiar la forma de generar los vuelos para que se obtengan datos en tiempo real a través de la *API* oficial de *FlightStats*. Esta nueva funcionalidad ofrecería los números de vuelo reales y se podría ver una simulación de lo que ocurre en realidad en el aeropuerto, teniendo además como ventaja conocer el estado de cada vuelo al instante. Esta parte resulta muy interesante pero no se ha incluido en el proyecto actual debido a motivos económicos ya que el uso de esta *API* implica el pago de una cuota.
- Siguiendo esta línea de mejora, los aviones que aterrizan podrían autogenerarse a medida que se programen los vuelos en las pantallas del aeropuerto, de tal manera que siempre haya algún avión disponible para aterrizar. En el caso de los aviones de salida, podríamos cambiar de estado a los aviones que aterrizan pasado un determinado tiempo, cambiando su información de vuelo de llegada a ser programados con un nuevo vuelo de salida (como ocurre en la realidad).
- Además, sería interesante recoger la información real que proporcionan los sensores de la pista del aeropuerto Adolfo Suárez-Madrid Barajas para integrarla en la propia aplicación.
- Otra de las opciones sería aumentar el número y tipo de sensores, pudiendo incluir alguno en los propios aviones para que nos proporcione información de altitud, velocidad, etc. E incluso se podría realizar un análisis de esos datos para incluir un histórico de los trayectos y que el usuario pueda ver información relativa sobre a qué altitud se realiza determinada ruta, por ejemplo.
- Cuando se planteó este proyecto, una de las ideas era que no sólo funcionara como una aplicación de realidad aumentada al uso si no que se pudiera utilizar de alguna manera dentro del propio aeropuerto. Mucha gente acude a recibir o despedir a sus familiares o amigos a los aeropuertos y se quedan esperando a que el avión despegue, actualizando la información en los dispositivos móviles o atendiendo a las pantallas para conocer mejor cuál es el estado de los vuelos en cuestión. La parte holográfica de este proyecto podría eventualmente usarse como una maqueta virtual que estuviera presente en la terminal, y que cualquiera pudiera acercarse a ver de una manera que no requiriera tener el dispositivo, si no con el propio de cada uno (lo cual implicaría un desarrollo con otra plataforma que no fuera la de *Tango*).

La premisa de la realización de este proyecto fue crear una maqueta en realidad aumentada, en el propio aeropuerto, para que los usuarios pudieran verla a través de algún dispositivo, por lo que una opción a realizar para mejorar este trabajo y orientarlo hacia este tema, sería migrarlo de plataforma.

Una de las opciones es tenerlo en realidad virtual y utilizar unas gafas en un stand del aeropuerto para que los usuarios conozcan la herramienta. Y una vez pasada esta fase, podría adaptarse a otra plataforma más avanzada y que no requiera un hardware específico (como por ejemplo *ARCore*, sucesor de *Tango*) y dar a conocer la aplicación. Así, una vez que los usuarios lo descubrieran desde el propio aeropuerto, podrían descargarse la aplicación e interactuar con ella en cualquier sitio sin tener que estar presentes en el aeropuerto.

Asimismo, si la idea prosperase, también podría realizarse el mismo planteamiento para otros aeropuertos o incluso elegir la terminal que se quiera dentro de un desplegable en la misma aplicación.

8. BIBLIOGRAFÍA

Dentro de este capítulo incluiremos las referencias a la información consultada durante la realización de este proyecto.

[10]: https://books.google.es/books?hl=es&lr=&id=34Ng5ftjVtcC&oi=fnd&pg=PT13&dq=zbrush&ots=1mRQezzSxL&sig=OSp3EYhaCRKehcWlwkVQJUCYC1Q#v=onepage&q=zbrush&f=false

[11]: https://e-archivo.uc3m.es/bitstream/handle/10016/12936/modelado_fernandez_2011_pp.pdf?sequence=1&isAllowed=y

[15]: http://cvl-demos.cs.nott.ac.uk/vrn/

[16]: http://jeuazarru.com/wp-content/uploads/2014/10/RA2013.pdf

[17]: https://www.cs.unc.edu/~azuma/ARpresence.pdf

[18]: http://sedici.unlp.edu.ar/bitstream/handle/10915/18399/Documento_completo__.pdf?sequence=1

[26]: http://www.redalyc.org/html/784/78445977004/

Otros enlaces consultados:

[1]: http://noticias.juridicas.com/base_datos/Admin/lo15-1999.html

[2]: https://www.flightstats.com/v2/flight-tracker/search

[3]: https://elpais.com/tecnologia/2017/09/07/actualidad/1504785739_150459.html

[4]: http://www.itreseller.es/en-cifras/2017/12/el-mercado-de-realidad-virtual-y-aumentada-crecera-un-95-en-201

[5]: https://www.amic.media/media/files/file_352_1289.pdf

[6]: https://www.xatakamovil.com/mercado/lejos-de-la-gama-alta-los-espanoles-pagamos-228-euros-de-media-en-la-compra-de-un-smartphone

[7]: https://es.wikipedia.org/wiki/Historia_del_diseño_asistido_por_computadora

[8]: http://pixologic.com

[9]: https://www.autodesk.es/products/maya/overview

[12]: https://www.youtube.com/watch?v=VcpphObZAgg

[13]: https://www.youtube.com/watch?v=45vUXA14DU4

[14]: https://es.3dsystems.com/3d-scanners/sense-scanner

[19]: https://www.youtube.com/watch?v=dcfU8mIOm-4

[20]: https://www.youtube.com/watch?v=RjJ44klc8zk

[21]: https://www.ces.tech

[22]: https://www3.lenovo.com/es/es/tango/

[23]: https://www.asus.com/es/Phone/ZenFone-AR-ZS571KL/

[24]: https://developers.google.com/ar/discover/

[25]: https://plus.google.com/communities/114537896428695886568?hl=ES

[27]: https://www.adafruit.com

[28]: https://en.wikipedia.org/wiki/Adafruit_Industries

[29]: https://github.com/adafruit

[30]: https://www.cgtrader.com/free-3d-models/aircraft/commercial/template-boeing-787-9

[31]: https://www.ssh.com/ssh/

[32]: https://github.com/sshnet/SSH.NET

9. ANEXO I: ESCENA TIPO EN UNITY

A continuación, se detallarán los aspectos técnicos más importantes del funcionamiento de *Unity 3D* para la completa compresión de esta parte del proyecto.

9.1Conjuntos de herramientas

La interfaz de *Unity 3D* se divide en varios espacios que proporcionan las herramientas necesarias para trabajar en él. Existe la opción de personalizar su colocación o incluso cambiar algunas de esas herramientas por otras que resulten más prácticas para el desarrollador, pero a continuación se mostrarán en función de la colocación que trae el programa por defecto.

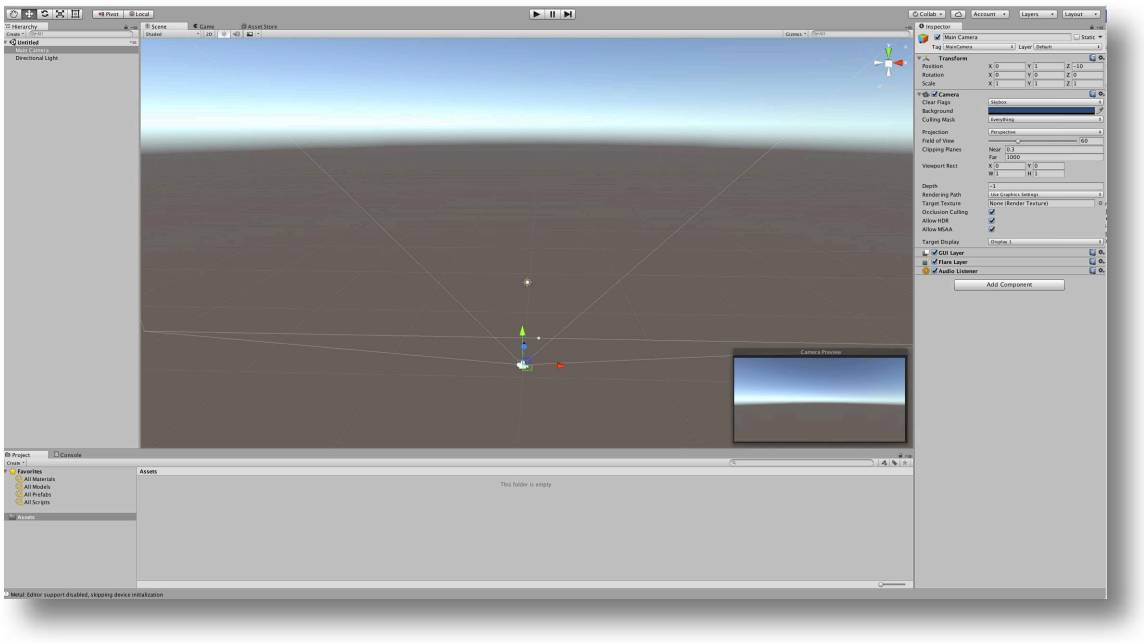

Ilustración 43: Interfaz de Unity 3D

Se detallan las partes principales:

- Jerarquía: se encuentra en la parte superior izquierda y es aquí donde se muestran todos los elementos que componen una escena.

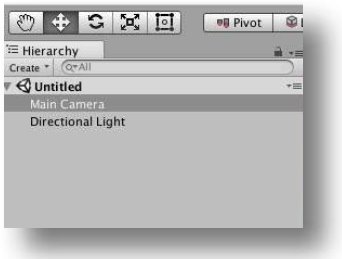

Ilustración 44: Jerarquía de un proyecto

- Proyecto: se pueden encontrar en la parte inferior izquierda, y se despliegan en la parte inferior central. Cada vez que se crea un proyecto, se crea una carpeta predefinida llamada *Assets*, donde el programador debe guardar los elementos pertenecientes a éste. Lo habitual es crear carpetas dentro de *Assets* para almacenar todos aquellos elementos susceptibles de ser usados en las diferentes escenas del proyecto (imágenes, objetos3D, texturas, *scripts*, etc.) y tenerlos así clasificados.

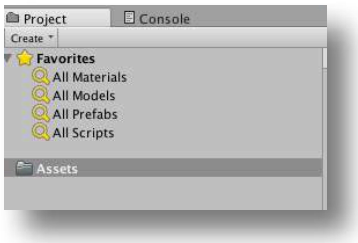

Ilustración 45: Carpetas del proyecto

- Inspector: se muestran en la zona superior derecha e indica la posición, estado y características del elemento que se haya marcado. Así mismo, ofrece la posibilidad de añadir propiedades a dichos elementos seleccionados, como por ejemplo materiales, *scripts*, botones, etc.

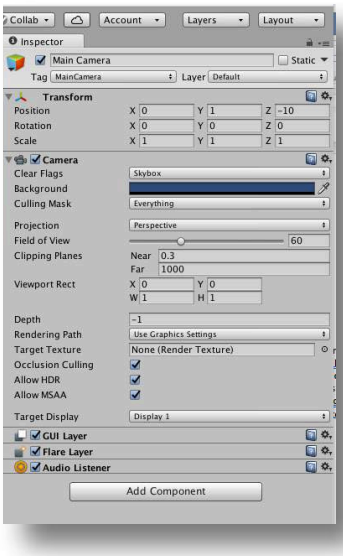

Ilustración 46: Inspector del elemento cámara, muestra sus características

- Consola: es una de las pestañas de la parte inferior y permite seguir la ejecución, mostrando mensajes de error o cualquier otro mensaje que se envíe a la pantalla.

Ilustración 47: Consola de Unity, donde se muestran los mensajes en la ejecución

- Escena: es la parte principal, se sitúa en el centro y es lugar donde se colocarán todos los elementos que se quiera mostrar en la escena.

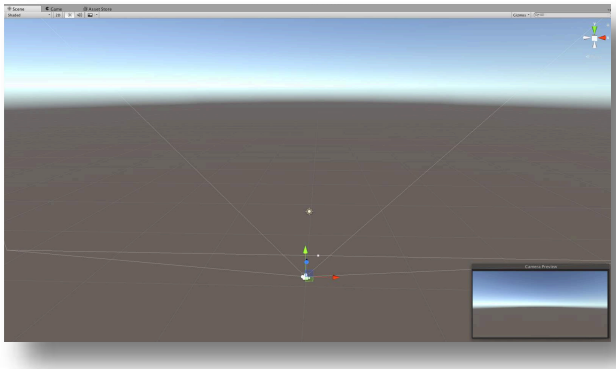

Ilustración 48: Escena de Unity, donde se sitúan todos los elementos a utilizar

- Ejecutar: se encuentra en la parte superior central y arranca o pausa la ejecución en función del botón que presionemos.

Ilustración 49: Botones de Unity para iniciar la ejecución, pausarla o pasar de escena

9.2 Elementos principales de una escena

En una escena de Unity aparecen por defecto dos elementos principales:

- *Directional Light*: consiste en una luz genérica para la escena, con las mismas propiedades a efectos de iluminación que la luz solar, es decir, se proyecta en todas las direcciones con la misma intensidad. Se incluye para que en proyectos simples el usuario no tenga que buscar cual es la luz más adecuada y pueda ver las escenas que crea de una forma sencilla.
- *Main camera*: es la cámara desde la cual se captará la escena una vez iniciemos la visualización. Todo el ambiente se creará en función de dónde esté situada y dependiendo de lo que se quiera mostrar.

10. ANEXO II: ESCENA INICIAL

A lo largo de este anexo se explicará con mayor detalle el funcionamiento interno de la escena inicial de la aplicación.

10.1 Elementos de la escena y sus características

Se muestra a continuación la escena inicial.

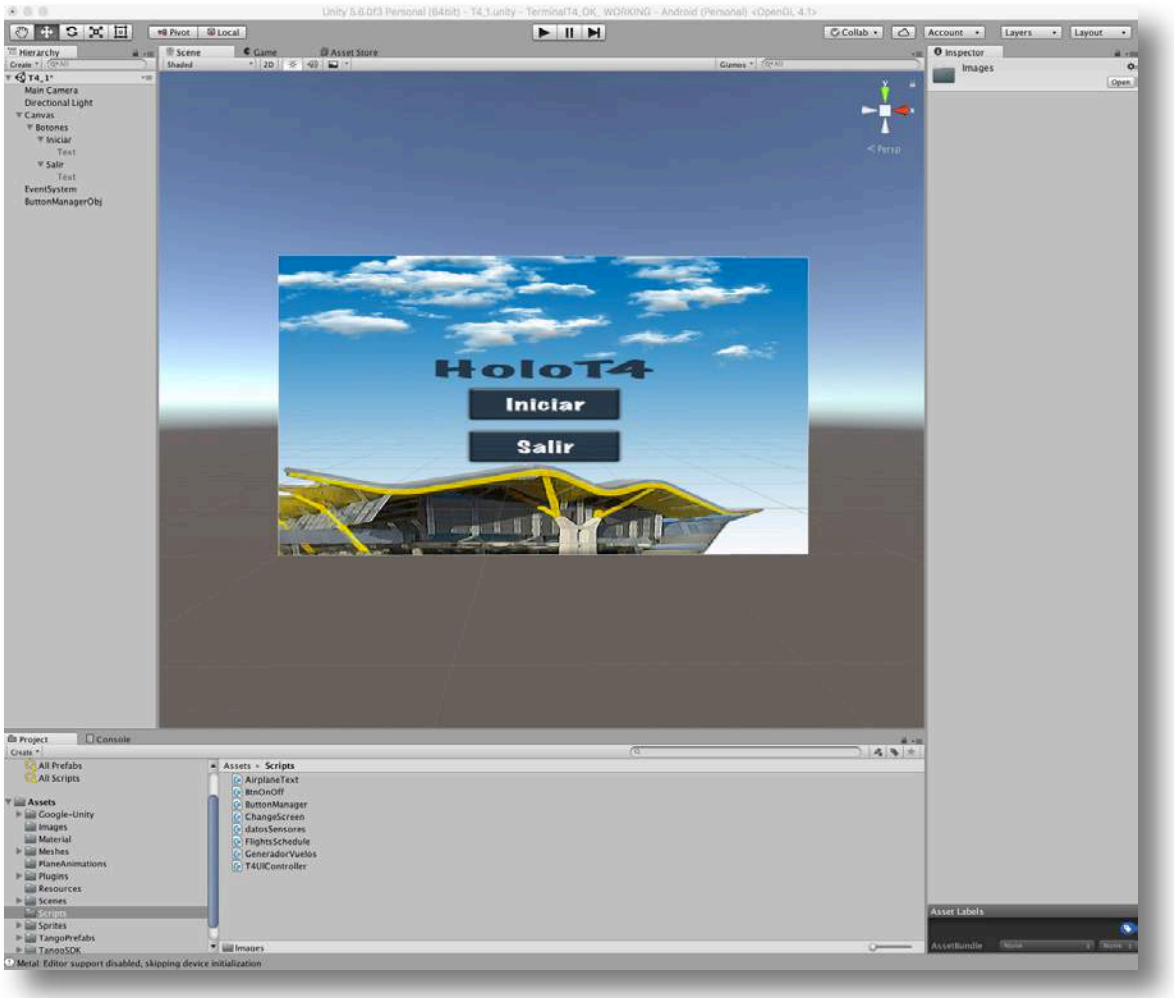

Ilustración 50: Interfaz de Unity en la creación de la escena inicial

Dentro de la jerarquía se tienen varios elementos predefinidos: *Main Camera* y *Directional Light*, cuyos parámetros sólo han sido modificados brevemente para ajustar la posición dentro de la escena. *EventSystem* es un componente que permite la interacción entre los objetos de la

escena, por lo que estará presente siempre y cuando se quiera condicionar el funcionamiento de alguna parte a través de código.

10.1.1 Canvas

Es un objeto predefinido de *Unity* que funciona como un panel en el que el desarrollador coloca aquellos elementos que quiere que se visualicen. Se utiliza tanto para objetos 3D como 2D, pero suele ser una buena herramienta para englobar carteles, fondos, letras y botones.

En este caso se ha creado dentro del *canvas* un objeto vacío (a modo de carpeta organizadora) denominado botones, que tiene dentro a su vez, los botones Iniciar y Salir.

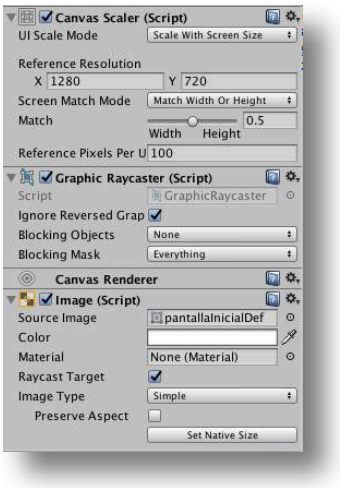

Ilustración 51: Inspector del objeto Canvas

10.1.2 ButtonManagerObj

Este objeto fue creado vacío para adjuntarle un script de código, de tal manera que cada vez que se necesite utilizar alguna de las partes del mismo, se pueda acceder a este objeto desde cualquier otra parte de la misma escena.

Ilustración 52: Características del objeto ButtonManagerObj

10.1.3 Botones

Ambos funcionan con el mismo planteamiento, al clicar se produce una acción. En el inspector se puede comprobar que sus características son prácticamente iguales:

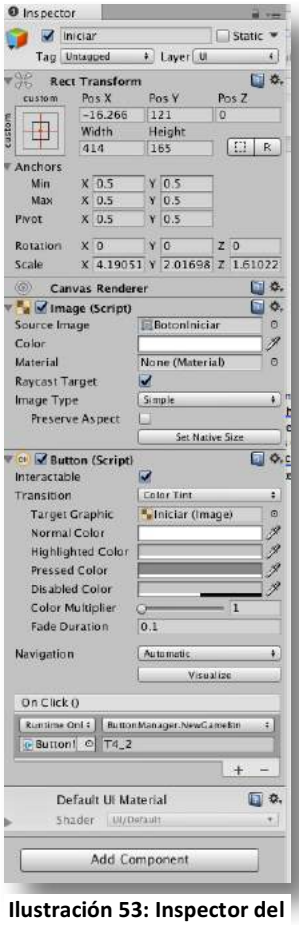

Ilustración 53: Inspector del botón Iniciar

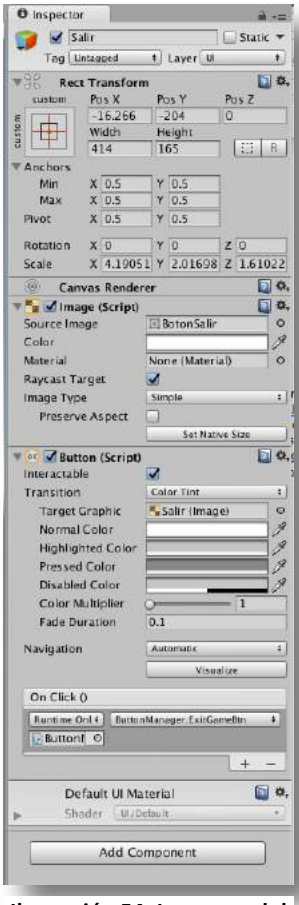

Ilustración 54: Inspector del botón Salir

Ambos tienen posiciones muy cercanas dentro de la escena, una imagen como fondo, un material predefinido y además un elemento *Button*. Éste se encarga de referenciar al objeto *ButtonManajerObj* y hacer uso de las funciones escritas en ese código:

```
using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;

public class ButtonManager : MonoBehaviour {


    // Se utiliza en la pantalla incial para que al pulsar Inicio cambie a la siguiente escena
    public void NewGameBtn (string newGameLevel) {
        SceneManager.LoadScene(newGameLevel);
    }

    // Se utiliza en la pantalla inicial para que al pulsar Salir salga del juego
    public void ExitGameBtn () {
        Application.Quit();
    }
}
```

Ilustración 55: Código que controla los botones Iniciar y Salir

11. ANEXO III: ESCENA PRINCIPAL

En este tercer anexo se explicará más en profundidad el funcionamiento de la escena principal de la aplicación. La estructura de la composición en *Unity 3D* es la siguiente:

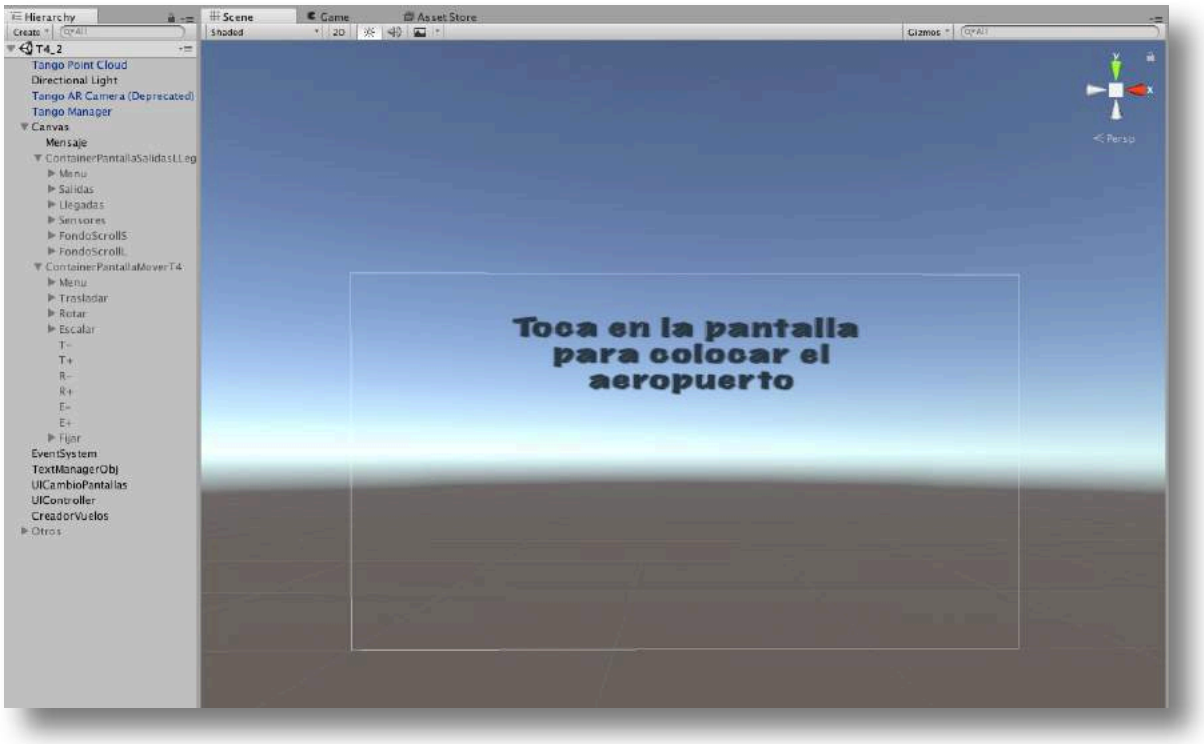

Ilustración 56: Interfaz de Unity en la creación de la escena principal

Es en esta parte donde cobra importancia la librería del proyecto Tango, ya que se utilizará para crear los objetos de realidad aumentada. Se usan tres elementos en esta escena con ese fin:

- *Tango Point Cloud*: se encarga de reconocer el posicionamiento del dispositivo en relación al resto del entorno y de calcular la profundidad de los objetos que capta a través de la cámara.
- *Tango Manager*: gestiona los eventos, permisos y notificaciones de errores del sistema.
- *Tango AR Camera*: sustituye a la cámara habitual de las escenas de *Unity* (*Main Camera*) y sus características unido al resto de componentes de la librería, proporciona profundidad y realismo a los objetos creados.

Se puede observar que dentro del objeto *Canvas* existen tres partes diferenciadas: Mensaje, Container pantalla salidas llegadas y Container pantalla mover T4.

La pantalla de Mensaje es la única visible al llegar a esta parte de la ejecución y su función es esperar (gracias a utilización del método *Update,* que se actualiza en cada fotograma) hasta que

el usuario toque en un punto de su entorno para emplazar la terminal T4. Se consigue controlar esto a través del script *ChangeScreen*:

```
public class ChangeScreen : MonoBehaviour {

    // Estas variables globales son publicas porque es necesario que en el inspector arrast
    // asi como el boton necesario para ello
    public GameObject contenedorPantallaAnterior;
    public GameObject contenedorPantallaNuevo;
    public GameObject mensaje;
    public Button yourButton;
    bool pantallaColocada = false;

    // En Update lo que hacemos es esperar por un click en la pantalla o en el boton fijar
    public void Update ()
    {
        // Si se toca el boton fijar se llama a la funcion pulsoBotonFijar
        Button btn = yourButton.GetComponent<Button>();
        btn.onClick.AddListener(pulsoBotonFijar);

        // Si tocamos la pantalla y la variable pantallaColocada esta a false, entramos
        if(Input.touchCount == 1 && pantallaColocada == false)
        {
            Touch t = Input.GetTouch(0);
            // Si el toque en la pantalla ha termiando, entramos
            if (t.phase == TouchPhase.Ended)
            {
                // Ocultamos el mensaje de colocar el aeropuerto y activamos el contentedor
                mensaje.SetActive(false);
                contenedorPantallaAnterior.SetActive(true);
                pantallaColocada = true;
            }
        }
    }
```

Ilustración 57: Código que controla el cambio entre las distintas pantallas

Con ese toque que el usuario debe dar en la pantalla ocurren diversas acciones.

Por un lado, tal y como se detalla en el código, se oculta la pantalla del mensaje y se activa la siguiente pantalla en la línea de ejecución. Para estos efectos se ha creado el objeto *UICambioPantallas*, que tiene como adjunto el script mencionado anteriormente, y cuyas variables globales se identifican con los elementos de la escena:

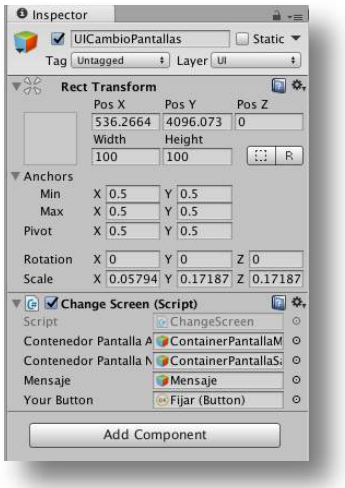

Ilustración 58: Características del objeto UICambioPantallas

Por otro lado, coincidiendo con el cambio de pantallas, también se coloca una instancia del objeto tridimensional de la terminal del aeropuerto. Esta parte se controla a través del objeto de la jerarquía *UIController*, cuyas características son las siguientes:

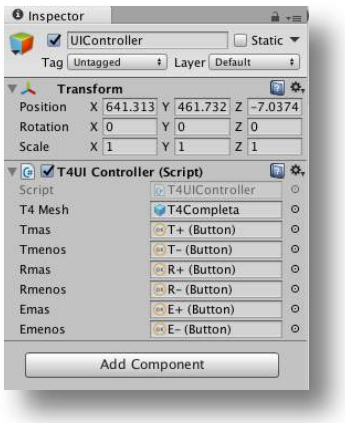

Ilustración 59: Inspector del objeto UIController

El elemento *T4Completa* es el objeto 3D de la terminal, que aparece en este momento de la ejecución:

```
public class T4UIController : MonoBehaviour
{
    public GameObject T4Mesh;
    private TangoPointCloud m_pointCloud;
    bool emplazado = false;
    GameObject instanciado; // NO RELLENAR EN EL INSPECTOR NUNCA
    public Button tmas;
    public Button tmenos;
    public Button rmas;
    public Button rmenos;
    public Button emas;
    public Button emenos;


    public void Start()
    {
        m_pointCloud = FindObjectOfType<TangoPointCloud>();

    }

    public void Update ()
    {
        if (Input.touchCount == 1)
        {
            // Se coloca la terminal
            Touch t = Input.GetTouch(0);
            // Booleano que reconoce si el objeto ya ha sido colocado
            if (t.phase == TouchPhase.Ended)
            {
                PlaceT4(t.position);

            }
        }

        desplazarT4 ();
    }
```

Ilustración 60: Código que controla los toques en la pantalla para emplazar la terminal

Si se produce un toque en la pantalla entonces se llama a la función *PlaceT4* utilizando el punto en el que se ha tocado. Además, se ha controlado que, si ésta ya ha sido colocada, no se pueda instanciar de nuevo.

La función *PlaceT4* funciona de la siguiente manera: detecta un toque en la pantalla, detecta cuando termina ese toque, analiza el punto que se ha tocado en el entorno real que observa a través de la cámara, si encuentra un punto en el que pueda trazar una perpendicular y que la terminal quede erguida sin apenas inclinación, entonces coloca el objeto.

```
public void PlaceT4(Vector2 touchPosition)
{
    // Se localiza un plano
    Camera cam = Camera.main;
    Vector3 planeCenter;
    Plane plane;
    if (!m_pointCloud.FindPlane(cam, touchPosition, out planeCenter, out plane))
    {
        Debug.Log("cannot find plane.");
        return;
    }

    // Se coloca en la superficie siempre frontalmente a la camara
    if (Vector3.Angle (plane.normal, Vector3.up) < 30.0f && emplazado == false)
    {
        Vector3 up = plane.normal;
        Vector3 right = Vector3.Cross (plane.normal, cam.transform.forward).normalized;
        Vector3 forward = Vector3.Cross (right, plane.normal).normalized;
        instanciado = Instantiate (T4Mesh, planeCenter, Quaternion.LookRotation (forward, up));
        emplazado = true;
    }


}
```

Ilustración 61: Código que controla la posición en la que se coloca la terminal

En este punto de la ejecución se nos ofrecen nuevos elementos con los que interactuar. Se puede ver la terminal colocada, botones en la parte superior que permiten rotar, trasladar y escalar el objeto y un botón en la parte inferior que dice “Fijar”.

Los botones de esta parte responden de diferente forma que los vistos anteriormente, en gran medida para tener menos componentes en los objetos de la escena y agilizar así la ejecución.

Desde la función *Update* de la ilustración 42 se puede observar una llamada a la función desplazarT4 que consiste en lo siguiente:

```
void desplazarT4 ()
{


    Button btn1 = tmas.GetComponent<Button>();
    Button btn2 = tmenos.GetComponent<Button>();
    Button btn3 = rmas.GetComponent<Button>();
    Button btn4 = rmenos.GetComponent<Button>();
    Button btn5 = emas.GetComponent<Button>();
    Button btn6 = emenos.GetComponent<Button>();

    btn1.onClick.AddListener (pulsoTrasladarMas);
    btn2.onClick.AddListener (pulsoTrasladarMenos);
    btn3.onClick.AddListener (pulsoRotarMas);
    btn4.onClick.AddListener (pulsoRotarMenos);
    btn5.onClick.AddListener (pulsoEscalarMas);
    btn6.onClick.AddListener (pulsoEscalarMenos);



}

public void pulsoTrasladarMas () {
    instanciado.transform.Translate (0.0001F,0.0F,0.0F);
}
void pulsoTrasladarMenos () {
    instanciado.transform.Translate (-0.0001F,0.0F,0.0F);
}
void pulsoRotarMas () {
    instanciado.transform.Rotate(new Vector3(0,0.01F,0));
}
void pulsoRotarMenos () {
    instanciado.transform.Rotate(new Vector3(0,-0.01F,0));
}
void pulsoEscalarMas () {
    instanciado.transform.localScale += new Vector3(0.0001F, 0.0001F, 0.0001F);
}
void pulsoEscalarMenos () {
    instanciado.transform.localScale -= new Vector3(0.0001F, 0.0001F, 0.0001F);
}
```

Ilustración 62: Código que controla cómo mover, trasladar y rotar la terminal

Se añaden componentes que están esperando en cada fotograma, el clic en el botón correspondiente, y si éste se produce se mueve, rota o escala el objeto 3D instanciado las unidades estipuladas en el código.
Una vez se tenga este colocado en la posición que el usuario desee, se pulsará en botón Fijar, que realiza a su vez dos acciones: hace visible la última de las pantallas y comienza el funcionamiento de la función *startSchedule.*

El funcionamiento de este script en su totalidad (*FlightSchedule*) resulta complejo por lo que se explicará paso por paso detallando su relación con el resto de elementos de la escena:

```
// Metodo que llamara a todos los anteriores una sola vez
public void StartSchedule () {
    leerFicheroSalidas ();
    leerFicheroLlegadas ();
    visibilidadAviones ();
    actualizarColas ();
    lanzarColas ();
}

// En la inicializacion se pone el contador a 10 segundos
void Start () {
    countdown = 15.0f;
}

// En esta funcion que se ejecuta en cada frame, se ira disminuyendo la cuenta atras
void Update () {
    countdown -= Time.deltaTime;

    if (countdown < 0.0f && pistaLibre == true){
        lanzarColas ();
        countdown = 15.0f;
    }
}
```

Ilustración 63: Código que controla toda la ejecución de la pantalla principal

La ejecución comienza llamando a dos funciones principales llamadas *leerFicheroSalidas* y *leerFicheroLlegadas*. Las tareas que realizan ambas son las mismas: generar una serie de vuelos aleatorios y plasmar esa información en determinadas zonas de la aplicación.

```
// Este metodo recoge la informacion de los vuelos de salida generada en el script GeneradorVuelos y deja la informacion
public void leerFicheroSalidas()
{
    // Se guarda en el array los datos de los vuelos creados en el script GeneradorVuelos y guardamos su longitud
    arraySalida = GameObject.Find("CreadorVuelos").GetComponent<GeneradorVuelos>().crearSalidas();
    avionesAbajo = arraySalida.Length;

    // Se copia la informacion de los vuelos a las pantallas de salidas y llegadas mostradas en la aplicacion
    for (int j=0; j<avionesAbajo; j++){
        TextFieldS.text += '\n' + arraySalida [j];
    }
}

// Funciona de la misma manera que el metodo anterior pero en este caso para los vuelos de llegada
public void leerFicheroLlegadas()
{
    arrayLlegada = GameObject.Find("CreadorVuelos").GetComponent<GeneradorVuelos>().crearLlegadas();
    avionesArriba = arrayLlegada.Length;

    for (int i=0; i<avionesArriba; i++){
        TextFieldL.text += '\n' + arrayLlegada [i];
    }
}
```

Ilustración 64: Código que controla la generación de los vuelos

Tal y como se muestra en el código, se accede al objeto *CreadorVuelos* de la jerarquía, cuyo *script* adjunto tiene una función llamada *crearSalidas* o *crearLlegadas*, pero que funcionan de la misma manera:

```
// En esta funcion se generan vuelos aleatorios utilizando unos parametros dados, como destinos, identificadores y la hora en la que iniciamos la aplicacion
public string [] crearSalidas(){

    // Se declaran variables como un array y la hora y minutos en la que se esta ejecutando
    string [] datosVuelosSalida = new string [8];
    int hora = System.DateTime.Now.Hour;
    int minutos = System.DateTime.Now.Minute;
    int diferencia = 60 - minutos;

    string [] destinos =  new string[] {"LONDRES", "PARIS", "NUEVA YORK", "ROMA", "BERLIN", "DUBLIN", "MANCHESTER", "ESCOCIA", "BARCELONA", "SEVILLA",
        "VALENCIA", "SANTIAGO DE COMPOSTELA", "VARSOVIA", "MILAN", "CAGLIARI", "BOSTON", "CARACAS", "MEXICO DF", "SAO PAULO", "ATENAS", "FLORENCIA", "FRANKFURT",
        "NANTES", "MARSELLA", "TOKYO", "LUXEMBURGO", "OPORTO", "BRUSELAS", "OSLO", "COPENHAGUE", "AMSTERDAM", "LUGA", "BRATISLAVA", "GINEBRA",
        "BUDAPEST", "VIENA", "HELSINKI", "MIAMI", "FILADELFIA", "LOS ANGELES", "BANGKOK", "NAIROBI", "SHANGHAY", "TORONTO", "MONTREAL", "PUNTA CANA",
        "CANCUN", "SAN JOSE", "LA HABANA", "MALÉ", "SIDNEY"};
    string[] identificadores = new string[] {"IB60", "IB12", "IB44", "IB30", "IB45", "IB34", "ES02", "ES64", "ES42",
        "ES37", "ES54", "ES55", "RY10", "RY66", "RY23", "RY43", "RY85", "RY47", "RY78", "RY76", "RY12", "RY64",
        "VU36", "VU36", "VU32", "VU19", "VU34", "VU32", "VU33", "FE39", "FE43", "FE45", "FE24", "FE31",
        "FE54", "FE65", "FE42", "FE31", "FE12", "FE51", "FE15", "FE21"};


    // Es necesario controlar en que fraccion de hora nos encontramos para generar los vuelos en los minutos cercanos
    for (int i = 0; i < 8; i++) {

        if (diferencia < 20 && diferencia > 10) {
            datosVuelosSalida [i] = " " + hora.ToString () + ":" + UnityEngine.Random.Range (minutos, diferencia).ToString () + "  " + RandomItem (destinos) + "   " + RandomItem (identificadores);

        }
        else if (diferencia < 10) {
            datosVuelosSalida [i] = " " + (hora+1).ToString () + ":" + UnityEngine.Random.Range (minutos, diferencia+20).ToString () + "  " + RandomItem (destinos) + "   " + RandomItem (identificadores);
        }
        else {
            datosVuelosSalida [i] = " " + hora.ToString () + ":" + UnityEngine.Random.Range (minutos, diferencia).ToString () + "  " + RandomItem (destinos) + "   " + RandomItem (identificadores);

        }

    }
    // La funcion devuelve un array con los datos de los vuelos: hora, destino e identificador para usarlos posteriormente
    return datosVuelosSalida;

}
```

Ilustración 65: Código que controla que genera la información de cada vuelo

Primero se calcula en qué hora y minuto se está, se selecciona aleatoriamente entre más de cincuenta destinos e identificadores de vuelo, y en función del minuto que sea, se crean aleatoriamente horas cercanas (entre veinte minutos antes y veinte minutos después, generalmente).
Se devuelven estos datos en un *array* y se utilizan de nuevo en la función *leerFicheroSalidas*, donde se calcula la cantidad de vuelos que hay (limitada a ocho de salida y ocho de llegada actualmente) y se copian a un elemento de texto de Unity para poder mostrarlos.

Estos datos serán visibles para el usuario en la nueva pantalla, clicando en el botón Salidas y Llegadas será posible ver u ocultar esta información.

La siguiente parte de la ejecución pasa por la llamada a *visibilidadAviones*:

```
// Este metodo hace visibles los aviones necesarios una vez le damos a fijar en la pantalla
public void visibilidadAviones()
{
    // Para que no se realicen los cambios sobre el prefab, lo que hacemos es llamar al metodo d
    T4Prefab = GameObject.Find("UIController").GetComponent<T4UIController> ().devolverInstancia

    // Se accede a los hijos 0 y 1 de la instancia de la T4 creada (que seran la flota de salida
    FlotaAvionesSalida = T4Prefab.transform.GetChild (0).gameObject;
    FlotaAvionesLlegada = T4Prefab.transform.GetChild (1).gameObject;

    // Se recorre toda la flota de los aviones de salida y los hacemos visibles, ademas de copia
    for (int contador = 0; contador < avionesAbajo; contador++) {

        FlotaAvionesSalida.transform.GetChild (contador).gameObject.SetActive (true);
        GameObject avion = FlotaAvionesSalida.transform.GetChild (contador).gameObject;
        TextMesh tm1 = avion.transform.GetChild(0).gameObject.GetComponent<TextMesh> ();
        tm1.text = arraySalida [contador];
    }

    // Se realiza la misma tarea para los aviones que estan arriba aunque no se hacen visibles t
    for (int contador2 = 0; contador2 < avionesArriba; contador2++) {

        GameObject avion2 = FlotaAvionesLlegada.transform.GetChild (contador2).gameObject;
        TextMesh tm2 = avion2.transform.GetChild(0).gameObject.GetComponent<TextMesh> ();
        tm2.text = arrayLlegada [contador2];
    }
}
```

Ilustración 66: Código que controla qué aviones son visibles y en qué momento

En esta parte del código se accede a propiedades de los objetos de la escena. Por ejemplo, para identificar qué aviones son de salida y cuales son de llegada, se identifican accediendo a ellos en la jerarquía del objeto 3D instanciado.

De esta manera, se hacen visibles todos los aviones que se han identificado como flota de salida, ya que son los que están en el aeropuerto esperando su turno para despegar. Sin embargo, los aviones pertenecientes a la flota de llegada no se harán visibles hasta que llegue su turno de aterrizar.

Es en los bucles de esta función donde se identifica a los aviones con cada uno de los vuelos creados con anterioridad, copiando su información dentro de las propiedades de cada objeto avión en la escena, para permitir que el usuario pueda reconocer a dónde se dirige o qué origen tiene cada uno de los aviones.

Durante la ejecución de la siguiente función se gestiona qué aviones comienzan la animación:

```
// Gestionar colas de pista y salida
public void actualizarColas (){

    // Se almacena la hora actual y los minutos
    int hora = System.DateTime.Now.Hour;
    int minutos = System.DateTime.Now.Minute;
    int minutosArriba = minutos + 10;
    int minutosAbajo = minutos - 10;

    // Se recorren los aviones de salida (8) y si el vuelo que contienen esta cerca de la hora actual
    for (int i = 0; i < avionesAbajo; i++){
        //cojo el texto de el avion en el que estamos iterando
        GameObject avion = FlotaAvionesSalida.transform.GetChild (i).gameObject;
        TextMesh tm1 = avion.transform.GetChild(0).gameObject.GetComponent<TextMesh> ();

        // Si contiene la hora en la que estamos y el minuto o 10 minutos antes o 10 minutos despues
        for(int j= minutosAbajo; j<minutosArriba; j++){
            if (tm1.text.Contains(hora.ToString()) == true) {
                if(tm1.text.Contains(j.ToString()) == true){
                    // Se mete en la cola de salida
                    colaSalida.Enqueue(FlotaAvionesSalida.transform.GetChild(i).gameObject);
                }
            }
        }
    }

    // Se realiza la misma operacion para los aviones que estan arriba
    for (int z = 0; z < avionesArriba; z++){
        //cojo el texto de el avion en el que estamos iterando
        GameObject avion2 = FlotaAvionesLlegada.transform.GetChild (z).gameObject;
        TextMesh tm2 = avion2.transform.GetChild(0).gameObject.GetComponent<TextMesh> ();


        // Si contiene la hora en la que estamos y el minuto o 10 minutos antes o 10 minutos despues
        for (int a = minutosAbajo; a < minutosArriba; a++) {
            if (tm2.text.Contains (hora.ToString ()) == true) {
                if (tm2.text.Contains (a.ToString ()) == true) {
                    // Se mete en la cola de pista
                    colaPista.Enqueue (FlotaAvionesLlegada.transform.GetChild(z).gameObject);
                }
            }
        }
    }
}
```

Ilustración 67: Código que controla las colas de aviones para aterrizar y despegar

Se utilizan colas para realizar esta gestión, de tal manera que si los datos del vuelo indican que debe aterrizar o despegar en los 10 minutos anteriores o posteriores a la hora en la que se está actualmente, entonces ese avión entra en la cola de despegue/aterrizaje.

Una vez todos los aviones estén en la cola en un determinado orden, el siguiente procedimiento es ejecutar la función lanzarColas:

```
// En este metodo se lanzan los elementos de las colas de salida y pista
public void lanzarColas (){

    // Estas variables indican el nombre del avion que se esta sacando de la cola y si la pista esta libre
    string nombreAnim;
    pistaLibre = true;

    // Se seleccionan aleatoriamente entre la cola de salida y la cola de pista
    int randomCola =  UnityEngine.Random.Range(0,2);

    // Si hay elementos todavia en la cola de salida y se sacado el numero random adecuado accedemos
    if(colaSalida.Count > 0 && randomCola == 0){
        // Si la pista esta libre se accede
        if (pistaLibre == true){
            // Se instancia el script de airplanetext y del avion que esta en la cola de salida
            AirplaneText apt1 = GameObject.Find (colaSalida.Peek ().name).GetComponent<AirplaneText> ();
            // Y se llama a la funcion ordenAnimar que lanzara la animacion de ese avion en concreto desde
            apt1.ordenAnimar ();
        }
        // Despues de realizar la llamada, se desencola ese avion
        colaSalida.Dequeue ();
    }

    // Se realizan las mismas operaciones que en el if, si el numero random es 1 se hace desde la cola de
    else if (colaPista.Count > 0 && randomCola == 1){
        // Si la pista esta libre
        if (pistaLibre == true){
            // Se consulta el gameobject de la cola
            GameObject go2 = colaPista.Peek ();
            // Lo hacemos visible
            go2.SetActive (true);
            // Se instancia su script y llamamos a ordenAnimar para lanzar la animacion
            AirplaneText apt2 = go2.GetComponent<AirplaneText> ();
            apt2.ordenAnimar ();
        }
        // Se desencola
        colaPista.Dequeue ();
    }

}
```

Ilustración 68: Código que lanza las animaciones pertinentes

En esta parte se selecciona aleatoriamente una cola (o bien la salida o bien la de llegada) y se pasa a la ejecución de los condicionales en función de esa selección. Ahí se estipula que si la pista está libre (es decir, no se está ejecutando la animación de ningún avión) entonces se puede desencolar el siguiente avión (además hacerlo visible si no se está en la cola de aterrizaje), acceder a su animación y ejecutarla.

La ejecución de las animaciones está recogida en el script *AirplaneText*:

```
// Este metodo es publico porque se accede desde otro script
// Cuando se saca un avion de la cola correspondiente, se llama a este metodo dentr
public void ordenAnimar (){

    // Se coge el nombre del gameobject avion (es el mismo que el de la animacion q
    nombreAnim = this.GetComponent<Animation> ().name;
    // Se instancia el script que controla las colas de los vuelos
    id1 = new FlightsSchedule();

    // Si la variable pista libre del script que controla las colas de los vuelos,
    if (id1.pistaLibre = true){
        this.GetComponent<Animation> ().Play (nombreAnim);
    }
    // Si se esta ejecutando la animacion pondremos la variable pista libre a false
    if (this.GetComponent<Animation> ().IsPlaying (nombreAnim) == true) {
        id1.pistaLibre = false;
    }

}
```

Ilustración 69: Código que controla si un avión determinado puede ejecutar su animación

Otra de las acciones que puede realizar el usuario en esta parte de la ejecución es comprobar la información de cada vuelo, para lo cual debe tocar en el cuerpo del avión, un toque para visualizar la información y dos para ocultarla. Esta parte está codificada en el script *AirplaneText*, adjunto a cada uno de los aviones y que se ejecuta de la siguiente manera:

```
// Este metodo coge el punto de la pantalla en el que se toca, y si coincide con el collider d
void infoTocarAvion (){

    // El gameObject panel es el objeto donde se guardara el Text Mesh con la informacion del
    panel = this.transform.GetChild (0).gameObject;
    // Se coge el nombre de este avion a traves del nombre de la animacion (porque es el mismo
    nombreObjeto = this.GetComponent<Animation> ().name;

    // Se almacena cada pulsacion sobre la pantalla
    Touch t = Input.GetTouch (0);
    // Se genera un raycast en esa posicion
    Ray raycast = Camera.current.ScreenPointToRay (t.position);
    RaycastHit raycastHit;

    // Si hay un toque en la pantalla
    if (Input.touchCount > 0) {
        // Si ese toque ha terminado y ademas el raycast choca con algo
        if (t.phase == TouchPhase.Ended && Physics.Raycast (raycast, out raycastHit) ){

            // Si el elemento con el que se ha chocado es el collider que tiene este avion y e
            if (raycastHit.collider.name == nombreObjeto && tocado == 0) {
                // Hace visible el panel con la informacion del vuelo
                panel.SetActive (true);
                // Aumenta la variable tocado
                tocado = 1;
            }
            // Si se ha tocado el collider de este avion por segunda vez, desaparece el panel
            else if (raycastHit.collider.name == nombreObjeto && tocado == 1){ // Se ha tocado
                // Desaparece el canvas y se vuelve a poner a cero tocado
                tocado = 0 ;
                panel.SetActive (false);
            }
        }
    }
}
```

Ilustración 70: Código que permite mostrar la información de vuelo de un avión

Existe también en esta pantalla un panel que muestra la información que devuelven los sensores, que se explica con más detenimiento en el Anexo IV.

Finalmente, si se desea reiniciar la aplicación, está presente el botón Home, que ejecuta el mismo código que el botón Inicio explicado con anterioridad, pero en lugar de cambiar a la escena principal, en este caso lo hace a la inversa, a la pantalla inicial.

12. ANEXO IV: SENSORES Y RASPBERRY PI

La manera de establecer una conexión entre los sensores y la Raspberry Pi pasaba por la utilización de la librería *Adafruit*. Fue necesario seguir las indicaciones de instalación que SE proporciona a través de la página del proyecto [29] e incluir en el trabajo realizado en Unity un *plugin* necesario [32] para realizar la conexión SSH desde la aplicación.

Una vez realizadas ambas acciones era posible acceder en remoto a la Raspberry Pi y ejecutar el comando que permitía realizar la medición a los sensores:

```
void Start () {
    accederSensor ();
}

void accederSensor ()
{
    // Se inicializa el texto
    text = this.GetComponent<UnityEngine.UI.Text> ();

    try
    {
        // Se realiza la conextion con los strings proporcionados antes, ademas del puerto 22
        var connectionInfo = new PasswordConnectionInfo(_host, 22, _username, _password);

        using (var client = new SshClient(connectionInfo ))
        {
            // Se conecta
            client.Connect();
            // Se ejecuta el script que realiza la lectura de la informacion de los sensores
            var command = client.RunCommand("sudo python /home/pi/Adafruit_Python_DHT/examples/AdafruitDHT.py 22 4");
            // Se recoge la informacion que nos proporciona
            parseo = command.Result.ToString();
            // Se corta el sring en funcion de los separadores
            arrayParseo = parseo.Split('=', ' ');
            //arrayParseo [4] += "hr";

            // Se copia en el texto que se mostrara la informacion que interesa (en las posiciones 1 y 4 estan respect
            text.text = arrayParseo [1] + "C" + '\n' + arrayParseo [4];
            client.Disconnect();
        }
    }
    catch(System.Exception e)
    {
        // Si no hay conexion se produce este error
        text.text = "Error\n" + e;

    }
}
```

Ilustración 71: Código que permite la comunicación de la aplicación con el sensor

Para realizar la conexión era necesario utilizar la dirección IP, el nombre del dispositivo, la contraseña y el puerto, información que se proporciona en las variables de *connectionInfo*.

Llegados a este punto, se ejecuta el comando que provoca que los sensores realicen una medición, guardándose el resultado en bruto y modificándolo de forma adecuada para mostrarlo por pantalla.

También se ha codificado que si no se establece conexión se lance un error y se notifique en la aplicación.

De cara a la exposición se ha estudiado detenidamente el uso de una dirección IP fija para la *Raspberry Pi*, de tal forma que estableciendo una red wifi móvil desde el Smartphone de la

creadora de este trabajo de fin de grado, se pudiera conectar la mini computadora y el móvil que ejecuta la aplicación, evitando así problemas derivados de la conexión con la red de la universidad y eliminando otras dependencias.

Para este fin ha sido necesario realizar modificaciones en tres archivos del sistema:

- *interfaces*: en este caso era necesario establecer una conexión wifi de forma automática, que accediera al archivo *wpa_supplicant.conf* para conocer las credenciales de la red a la que debía conectarse.

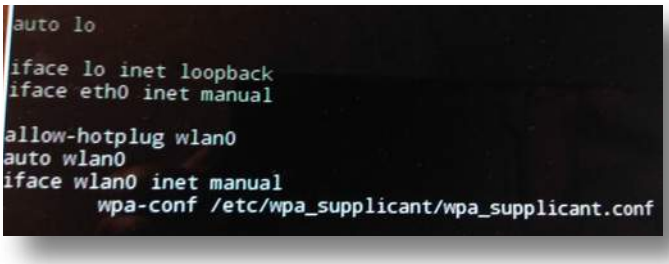

Ilustración 72: Código incluido en el archivo Intefaces

- *wpa_supplicant.conf*: aquí se ha estipulado que la red adecuada a la que conectarse recibe el nombre "TFGMarga" y cuya contraseña es "proyecto".

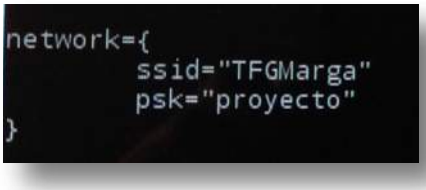

Ilustración 73: Código incluido en el archivo Supplicant

- *dhcpcd.conf*: en este archivo se establece la IP estática que desea otorgar al dispositivo.

Ilustración 74: Código incluido en el archivo Dhcpcd

13. MANUAL DE USUARIO

13.1 Introducción a la aplicación

A través de esta aplicación podremos ver en realidad aumentada un aeropuerto, colocarlo en cualquier espacio de nuestro entorno real y ver cómo se gestionan una serie de vuelos generados aleatoriamente. Podremos identificar los datos de vuelo de cada uno de los aviones, su destino y hora que tienen prevista para el despegue o aterrizaje.

La forma de interactuar es directamente con la pantalla táctil, sobre cualquier punto en el que identifiquemos a través de la cámara que queramos emplazar la terminal o directamente a través de los botones que se nos presentan.

13.2 Instalación

Se procede a la instalación de la aplicación a través de un ejecutable con la extensión *apk* que habremos guardado en descargas con anterioridad. Una vez toquemos el fichero se inicia la instalación y se nos pide confirmación para continuar con la operación:

13.3 Funcionamiento

Una vez realizada esta operación podemos ver como aparece el icono en nuestro escritorio:

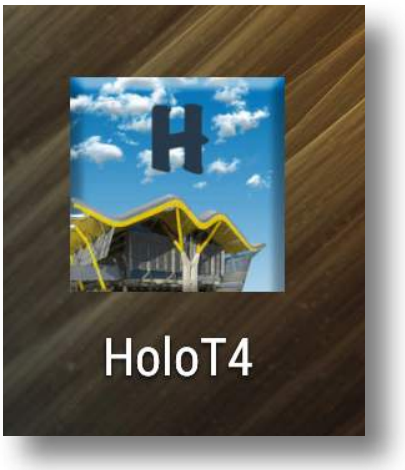

Ilustración 75: Icono de la aplicación

Lo pulsamos y abrimos la aplicación. En un primer momento solamente hay dos opciones posibles, iniciar la aplicación (botón superior) o salir de ella (botón inferior).

Ilustración 76: Pantalla inicial

Si pulsamos Iniciar, esto nos lleva a una segunda pantalla donde, tras 3-4 segundos se abre la cámara de nuestro dispositivo y aparece sobrepuesto un mensaje que dice: “Toca en la pantalla para colocar el aeropuerto”.

Ilustración 77: Segunda pantalla de la aplicación

Se debe buscar por lo tanto un lugar en el que emplazar la terminal, a ser posible un sitio plano (una mesa o superficie suficientemente grande, 2 x 2 metros sería suficiente) y cerca de la altura de nuestros ojos (ya que si lo colocamos en el suelo no podremos apreciar los detalles). Una vez pulsemos la pantalla nos aparece otra escena con diversos botones:

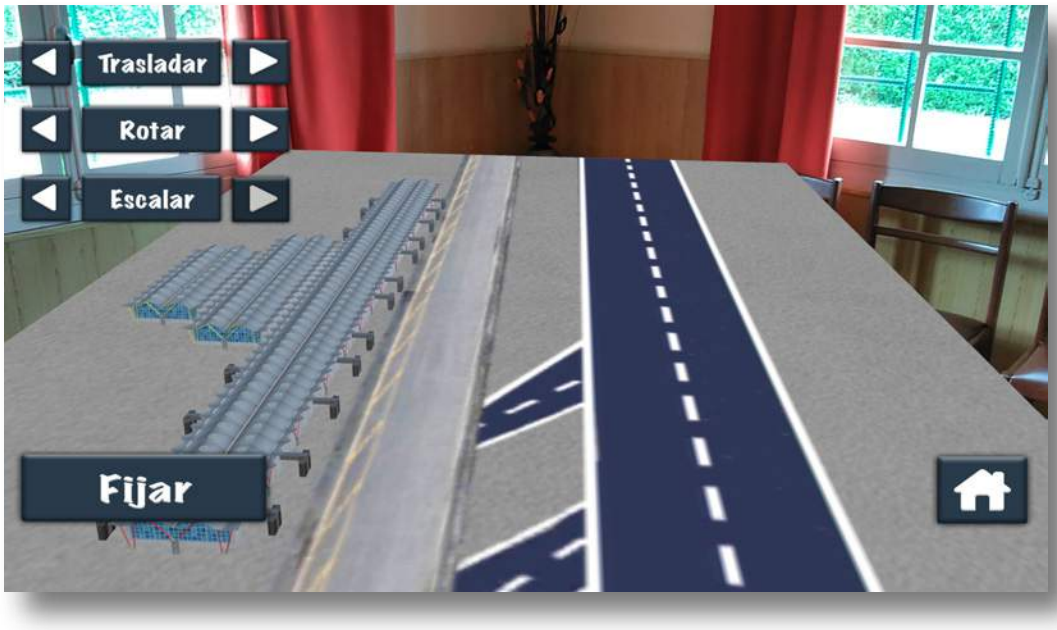

Ilustración 78: Tercera pantalla de la aplicación

Si se pulsan los botones que están a la izquierda y derecha de los paneles que ponen Trasladar, Rotar y Escalar podremos mover, rotar y modificar el tamaño de la terminal, respectivamente.

El botón que se encuentra en la parte inferior derecha de la pantalla, es el botón Home que nos llevará a la pantalla inicial para reiniciar la aplicación desde el principio.

Si clicamos sobre el botón inferior izquierdo, cuyo título dice Fijar, la posición de la terminal quedará fijada y comenzarán las animaciones de despegue y aterrizaje de los aviones.

En esta nueva pantalla veremos dos botones en la parte superior, que si los pulsamos mostrarán la información relativa a los vuelos programados cerca de la hora actual y que si volvemos a pulsar, se ocultarán.

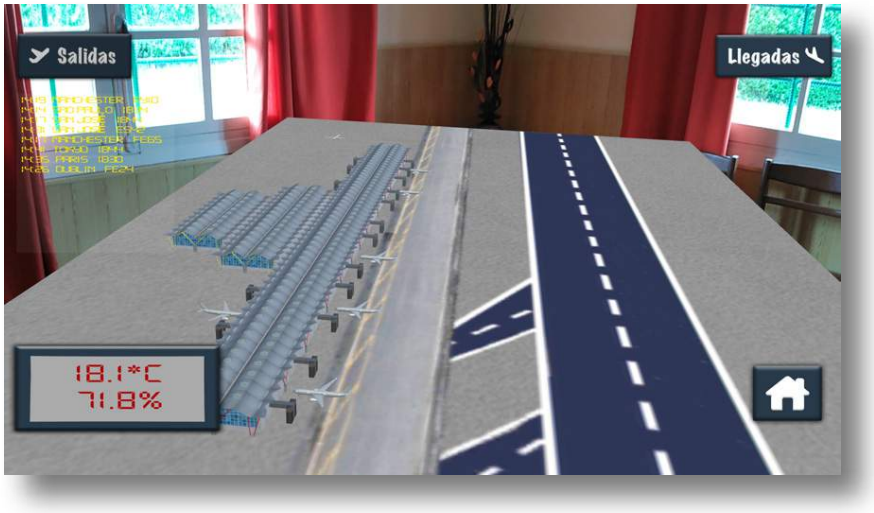

Ilustración 79: Pantalla final de la aplicación

En la parte inferior derecha está el botón *Home* tal y como aparecía en la pantalla anterior mientras que a la izquierda se nos muestra la información de temperatura y humedad procedente del sensor.

Ilustración 80: Detalle de la información de temperatura y humedad del sensor

Para salir de la aplicación es suficiente con pulsar el botón de menú de nuestro Smartphone o acudir a la pantalla de inicio de la aplicación y desde allí pulsar Salir.