SALTOUR

Soluciones tecnológicas para una mejora de la experiencia turística

Anexo V: Manual del programador

VNiVERSiDAD D SALAMANCA

Proyecto de Fin de Máster en Ingeniería Informática

Tutores:

Ángel Luis Sánchez Lázaro

Ana Belén Gil González

Alumno:

Miguel Cabezas Puerto

Manual del programador

1. Manual del programador

Este anexo explicará de manera general cómo se ha implementado el sistema en una aplicación Android. Esto conlleva la explicación de las adaptaciones de todos los modelos descritos en anteriores anexos (elicitación de requisitos, análisis y diseño) con el fin de conseguir un nivel de abstracción menor para así poder obtener un sistema tangible. Estasección estará compuesta por:

- ◊ Implementación de datos: detallará cómo se han almacenado y tratado los datos requeridos en los requisitos y desarrollados en el resto de los modelos.
- ◊ Implementación de la interfaz: este apartado, en el que se indicará cómo se ha desarrollado la implementación de las diferentes interfaces de usuario de la aplicación, así como las transiciones entre estas.
- ◊ Código fuente: este apartado detallará la implementación en código fuente de los patrones y modelos mencionado en anexos anteriores.

1.1 Implementación de datos

En las anteriores fases se describieron modelos relacionales para el almacenamiento de los datos, sin embargo, llegados a la implementación surgieron dos contratiempos que obligaron a transformar este modelo de datos a uno no relacional tratando siempre de aproximar lo máximo posible a su equivalente si se siguiera un modelo relacional.

Estos factores fueron, por una parte, principalmente la no existencia de servidores gratuitos con bases de datos relacionales a las que poder consultar en remoto (uno de los requisitos dado el carácter cercano a tiempo real de la aplicación). Por otra parte, aunque menos relevante, la necesidad recuperar datos de forma más rápida en las lecturas ya que estas se presuponen más abundantes que las operaciones de lectura.

Se eligió un modelo no relacional debido a que la plataforma Firebase a través de su herramienta Firestore ofrecía, sin necesidad de implementar un servidor propio (fuera del alcance de este proyecto), una base de datos basada en documentos y clave-valor que poder utilizar de forma gratuita e integrada con el SDK de Android.

1.1.1 Firestore: modelo no relacional

A continuación, se detalla cómo finalmente han quedado distribuidos los datos obedeciendo una estructura de base de datos documental con clave-valor.

Autenticación de usuarios

Ilustración 1 Autenticación usuarios

Como se puede observar la base de datos para la autenticación es en realidad relacional al disponer de diferentes columnas (identificador, proveedor de autenticación (correo y contraseña o Google), fecha de creación del usuario, última fecha de acceso y un identificador único calculado automáticamente para cada usuario)

Retos

Se trata de una base de datos documental. En ella hay una colección, en este caso para retos, compuesta por múltiples documentos cada uno de ellos asociado a un reto. Este documento a su vez está formado por parejas clave-valor para cada uno de sus atributos. Este esquema puede observarse en la ilustración siguiente.

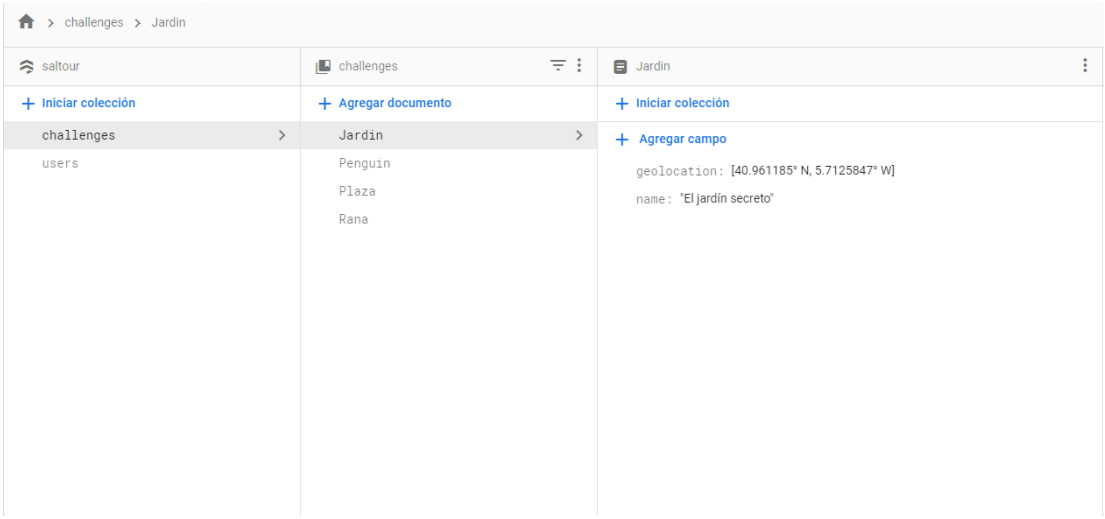


Ilustración 2 Retos

Usuarios

De nuevo se trata de una base de datos documental. En ella existe una colección conformada por varios documentos, cada uno de ellos asociado a un usuario. Dentro de cada usuario, en parejas clave-valor se pueden observar los diferentes atributos almacenados.

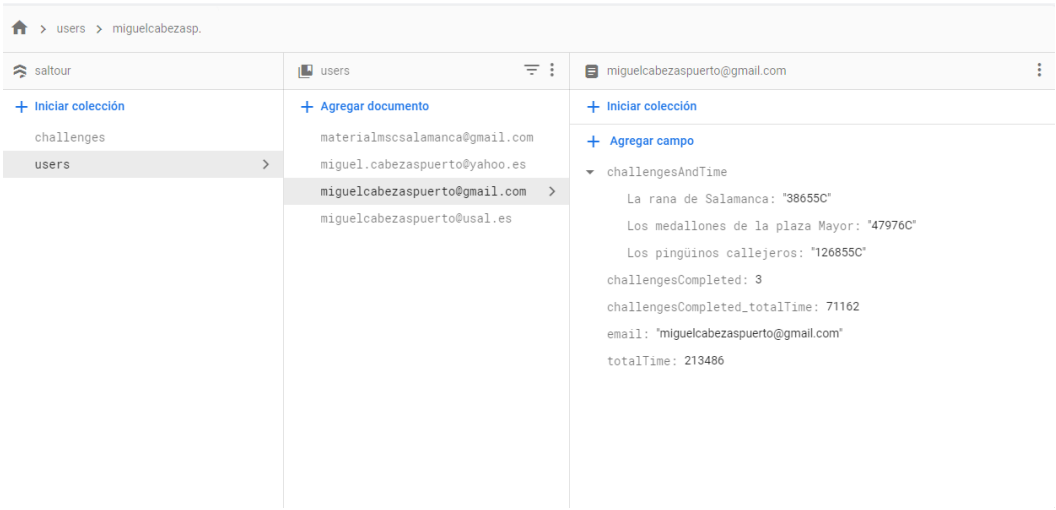


Ilustración 3 Usuarios-retos

Imágenes

A mayores de los datos se ha empleado el contendor de archivos que ofrece Firebase a través de su herramienta Firestorage para el almacenamiento de las imágenes de perfil que tenga el usuario en cada momento para que así estas sean accesibles y visibles entre todos los usuarios de la aplicación.

Este almacenamiento consiste en un contenedor accesible a través de una ruta (con los permisos adecuados) a modo de disco “en red”. Dentro de él se pueden crear a su vez subdirectorios o directamente archivos también accesibles a través de una ruta.

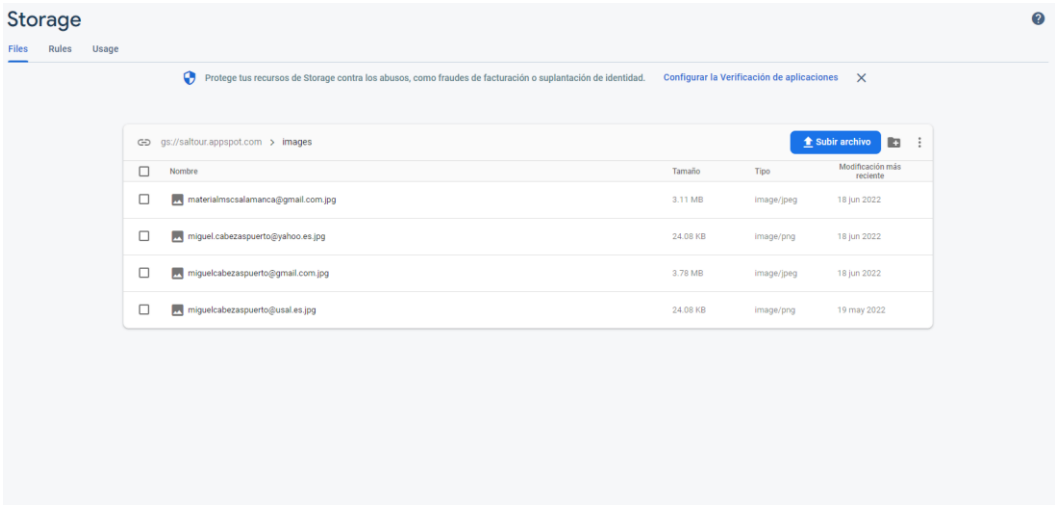


Ilustración 4 Repositorio imágenes

1.2 Implementación de la interfaz

La implementación de la interfaz de la aplicación ha sido un proceso también iterativo, ya que se ha ido aprovechando el aumento de conocimientos de programación en Androidy las opiniones de gente para mejorar la interfaz de usuario.

En los requisitos se recogía que la interfaz fuera sencilla de utilizar. Es por ello que, comose menciona

en el anexo V Diseño, se siguió el modelo de otras aplicaciones Android estándares para así poder reducir la curva de aprendizajedel usuario.

También se ha implementado la internacionalización del contenido estático acorde al idioma del dispositivo que corre la aplicación, además de otros elementos e iconos gráficos que permiten conocer al usuario más rápidamente las funcionalidades de las distintas opciones que se ofrecen.

Es importante destacar que, en Android, todos los aspectos visuales se implementan en archivos XML, que luego serán importados por actividades que controlan las vistas y les aportan contenidos. Existen diferentes maneras de implementar las vistas, bien sea colocando los objetos de manera absoluta en la pantalla o siguiendo alguna disposición especial. Debido a que Android se ejecuta en dispositivos diversos (tablets, teléfonos...), el diseño de la interfaz gráfica se complica, ya que los elementos se pueden mostrar de una manera diferente. Por ello, la disposición de los elementos se ha implementado, en lamedida de lo posible, independientemente del tamaño o la resolución de pantalla. No obstante, debido al tiempo disponible, se ha limitado la orientación de la pantalla en la aplicación a la posición vertical quedando como una ampliación de sencilla implementación el desarrollo de las vistas en posición horizontal.

1.2.2 Disposición de los elementos.

Basándose en el mockup especificado en el anexo V *Diseño* y en las opiniones de gente a lo largo del proceso de implementación, se ha ido modificando la interfaz parafinalmente quedar constituida por los elementos que se detallan en el anexo VII Manualdel usuario

1.2.3 Transiciones entre pantallas

Este apartado se puede consultar en el anexo VII Manual del usuario

1.3 Código fuente

1.3.1 Introducción

Este apartado pretende facilitar la comprensión del código fuente del proyecto desarrollado bajo el patrón MVC. Es por ello que se explica cada una de las clases pertenecientes a cada paquete a través de la elaboración de una documentación del código siguiendo el formato *javadoc*, para mayor detalle se recomienda acudir a la carpetaadjunta con dicho nombre.

Cabe destacar que los paquetes son independientes, por lo que si se desea añadir uno nuevo basta con unirlo en los controladores. Por el contrario, si se desea eliminar algún
paquete sólo sería necesario realizar pequeños cambios sin llegar a comprometer el resto de la funcionalidad.

Para una mejor comprensión de las clases detalladas en el *javadoc* se debe tener en cuenta que en la programación Android se divide la aplicación en módulos independientes que solo realicen una tarea concreta. Con esto llegamos a la conclusión de que las aplicaciones Android no tienen un punto de entrada y otro de salida, se pueden definir todos los que se necesiten.
Para ello Android proporciona cuatro tipos de componentes básicos:

- ◊ Activity: son las encargadas de mostrar la interfaz de usuario e interactuar con él. Responden a los eventos generados por el usuario (pulsar botones etc). Heredan de la clase Activity. El aspecto de la actividad se aplica pasando un objeto View (Encargado de dibujar una parte rectangular en la pantalla, pueden contener más objetos View, además todos los componentes de la interfaz (botones, imágenes, etc) heredan de View) al método Activity.setContentView(), que es el método encargado de dibujar la pantalla. Normalmente las vistas ocupan toda la pantalla, pero se pueden configurar para que se muestren como flotantes. Estas vistas se definirán mediante ficheros XML. Las actividades también pueden llamar a componentes que se mostrarán sobre su View (como diálogos o menús).
- ◊ Servicios: no tienen interfaz visual y se ejecutan en segundo plano, se encargan de realizar tareas que deben continuar ejecutándose cuando la aplicación no está en primer plano. Todos los servicios extienden de la clase Service.
- ◊ Broadcast Receiver: reciben un mensaje y reaccionan ante él, extienden de la clase BroadcastReceiver, no tienen interfaz de usuario, pero pueden lanzar Actividades como respuesta a un evento.
- ◊ Content Provider: ponen un grupo de datos a disposición de distintas aplicaciones, extienden de la clase ContentProvider para implementar los métodos de la interfaz, pero para acceder a esta interfaz se ha de usar una clase llamada ContentResolver. Se explicarán posteriormente.

Por otra parte, cabe destacar que toda actividad o Activity en Android tiene un ciclo de vida. Este lo componen las siguientes fases o estados:

- ◊ Activa (Running): La actividad está encima de la pila, lo que quiere decir que es visible y tiene el foco.
- ◊ Visible (Paused): La actividad es visible pero no tiene el foco. Se alcanza este estado cuando pasa a activa otra actividad con alguna parte transparente o que no ocupa toda la pantalla. Cuando una actividad está tapada por completo, pasa a estar parada.
- ◊ Parada (Stopped): Cuando la actividad no es visible. El programador debe

guardar el estado de la interfaz de usuario, preferencias, etc.

◊ Destruida (Destroyed): Cuando la actividad termina al invocarse el método finish(), o es matada por el sistema.

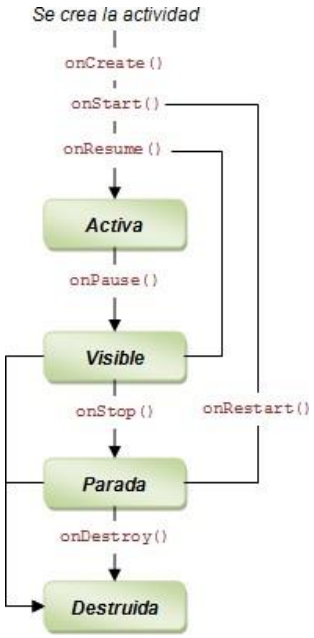


Ilustración 5 Estados actividad Android

Por ello es importante aclarar la funcionalidad de los métodos inherentes a este tipo de clases antes de explicarlo en próximos apartados relacionados con la Vista.

◊ onCreate(Bundle): Se llama en la creación de la actividad. Se utiliza para realizar todo tipo de inicializaciones, como la creación de la interfaz de usuario o la inicialización de estructuras de datos. Puede recibir información de estado de la actividad (en una instancia de la clase Bundle), por si se reanuda desde una actividad que ha sido destruida y vuelta a crear.

◊ onStart(): Nos indica que la actividad está a punto de ser mostrada al usuario.

◊ onResume(): Se llama cuando la actividad va a comenzar a interactuar con el usuario. Es un buen lugar para lanzar las animaciones y la música.

◊ onPause(): Indica que la actividad está a punto de ser lanzada a segundo plano, normalmente porque otra actividad es lanzada.

◊ onStop(): La actividad ya no va a ser visible para el usuario.

◊ onRestart(): Indica que la actividad va a volver a ser representada después de haber pasado por onStop().

◊ onDestroy(): Se llama antes de que la actividad sea totalmente destruida.

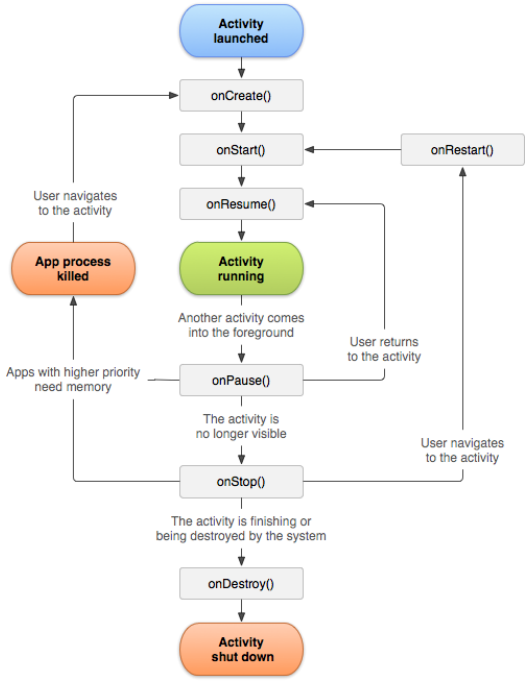


Ilustración 6 Métodos y estados actividades Android

Por último, es importante resaltar que la gestión de estilos, idiomas, tamaños, formatos, etc. Se realiza, en Android, a través de ficheros XML, cada uno asociado con un recurso. Para su creación se debe crear como el tipo de recurso que represente. Cada uno de los tipos de recursos tiene asociada una carpeta, estas son:

Carpeta identificador	**Descripción**
res/drawable/ R.drawable	Ficheros en bitmap (.png, .jpg o .gif). Ficheros PNG en formato Nine-patch (.9.png). Ficheros XML con descriptores gráficos
res/mipmap/ R.mipmap	Ficheros en bitmap (.png, .jpg o .gif). Estos gráficos no son rescaldados para adaptarlos a la densidad gráfica del dispositivo, sino que se buscará en las subcarpetas el gráfico con la densidad más parecida y se utilizará directamente.
res/layout/ R.layout	Ficheros XML con los Layouts usados en la aplicación.
res/menu/ R.menu	Ficheros XML con la definición de menús. Podemos asignar una actividad o

	una vista.

res/anim/ R.anim	Fichero XML que permiten definir una animaciones Tween también conocidas como animaciones de vista.
res/animator R.animator	Ficheros XML que permiten modificar las propiedades de un objeto a lo largo del tiempo
res/xml/ R.xml	Otros ficheros XML, como los ficheros de preferencias
res/raw/ R.raw	Ficheros que se encuentran en formato binario. Por ejemplo ficheros de audio o vídeo.
res/values/	Ficheros XML que definen un determinado valor para definir un color, estilo, cadena de caracteres, etc. Se describen en la siguiente tabla.

Dentro de la carpeta *values* podemos encontrar:

Fichero por defecto **identificador**	**Descripción**
strings.xml R.string	Identifica cadenas de caracteres. Asociando cada fichero strings.xml a un idioma se consigue un contenido de las cadenas recogidas en este fichero en diferentes idiomas (traduciendo cada identificador de cada al idioma deseado en su correspondiente strings.xml) *<string name="saludo">¡Hola Mundo!</string>* *<string name="saludo">Hello World!</string>* *<string name="saludo">Salut à tous!!</string>*

colors.xml R.color	Un color definido en formato ARGB (alfa, rojo, verde y azul). Los valores se indican en hexadecimal en uno de los siguientes

	formatos: #RGB, #ARGB, #RRGGBB ó #AARRGGBB *<color name="verde_opaco">#0f0</color>* *<color name="red_translucido">#80ff0000</color>*
dimensions.xml R.dimen	Un número seguido de una unidad de medida. px - pixeles, mm - milímetros, in – pulgadas, pt – puntos (=1/72 pulgadas), dp – píxeles independientes de la densidad (=1/160 pulgadas), sp – igual que dp pero cambia según las preferencias de tamaño de fuente. *<dimen name="alto">2.2mm</dimen>* *<dimen name="tamano_fuente">16sp</dimen>*
styles.xml R.style	Definen una serie de atributos que pueden ser aplicados a una vista o a una actividad. *<style name="TextoGrande" parent="@style/Text">* *<item name="android:textSize">20pt</item>* *<item name="android:textColor">#000080</item>* *</style>*
R.int	Define un valor entero. *<integer name="max_asteroides">5</integer>*
R.bool	Define un valor booleano. *<bool name="misiles_ilimitados">true</bool>*
R.id	Define un recurso de id único. La forma habitual de asignar id a los recursos es utilizando el atributo id="@+id/nombre". Aunque en algunos casos puede ser interesante

	disponer de id previamente creado, para que los elementos así nombrados tengan una determinada función. *<item type="id" name="button_ok"/>* *<item type="id" name="dialog_exit"/>*
R.array	Una serie ordenada de elementos. Pueden ser de strings, de enteros o de recursos (TypedArray)

1.4 Implementación de la realidad aumentada

1.4.1 Instalación de Unity-Vuforia

Para comenzar, es necesario tener instalado Unity, debido a la asignatura de Animación Digital cursada tanto en el Grado en Ingeniería Informática como la misma cursada durante el primer cuatrimestre del máster, este paso ya se había realizado.

A continuación, se debe descargar el SDK de Vuforia para Unity. Una vez descargado y abierto un Proyecto en Unity, se debe importar como nuevo paquete. Para ello, en MacOS, en la pestaña superior Assets, se selecciona Import Package y una vez ahí, Custom Package. Con ello se seleccionará el paquete SDK de Vuforia descargado y se importará al proyecto.

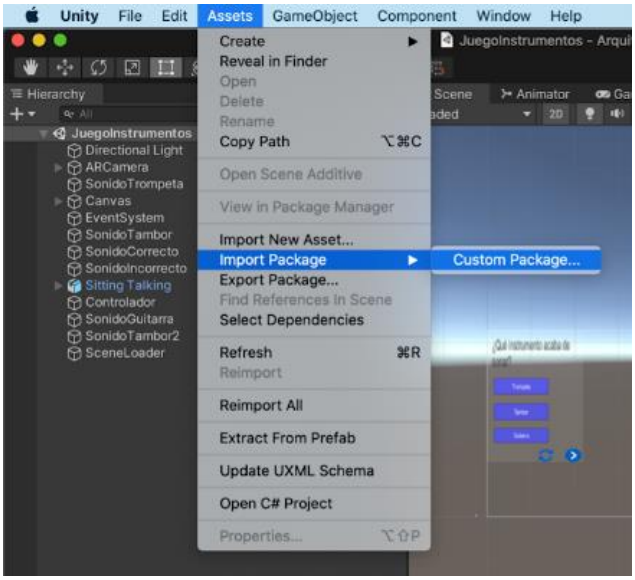


Ilustración 7 Importar SDK Vuforia

1.4.2 Portal de desarrolladores de Vuforia

Licencia

Es necesario disponer de una licencia, gratuita, para poder desarrollar realidad aumentada con Vuforia. Para ello se accede a su web, se registra una nueva cuenta, se hace login. Una vez dentro, se

selecciona la pestaña Develop, License Manager y se crea la licencia asignándole un nombre y confirmando como se puede ver en la siguiente imagen. Cabe destacar que se ha reutilizado la licencia ya creada para la asignatura Paradigmas Avanzados de la Interacción Persona Ordenador, de ahí que su nombre permanezca como PAIPO.

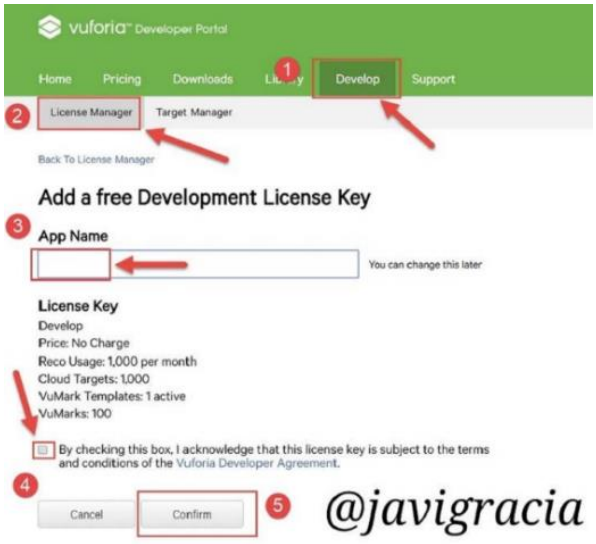
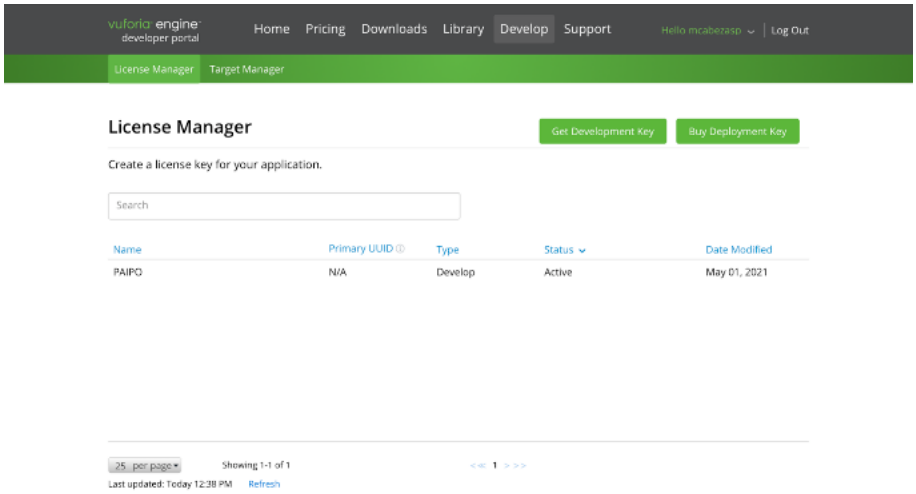


Ilustración 8 Añadir licencia gratuita Vuforia

Una vez creada, se puede acceder a ella para consultar la licencia requerida en futuros pasos.



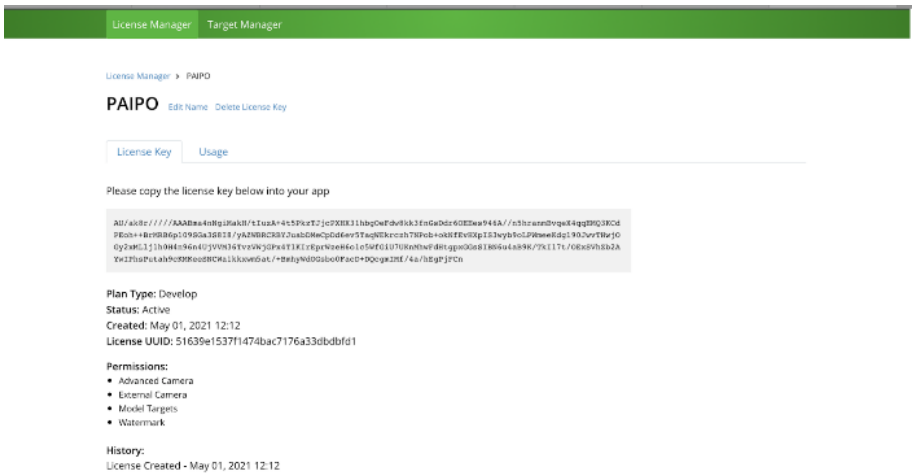
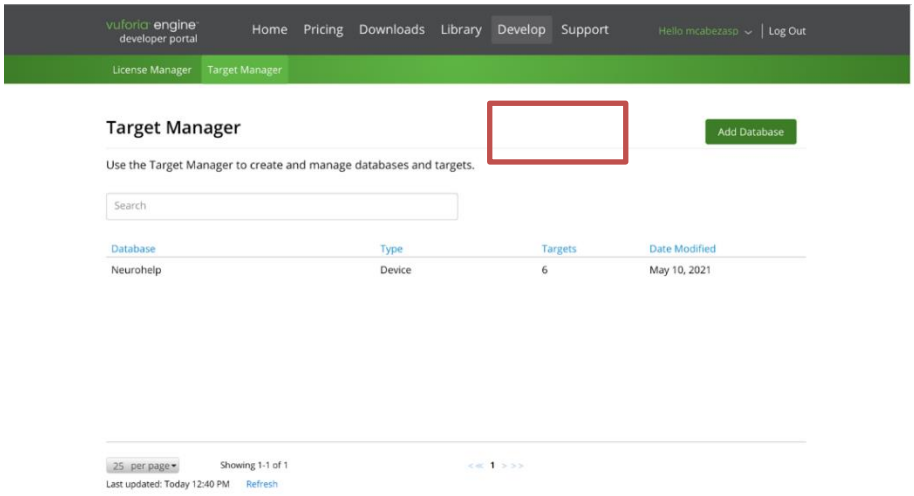


Ilustración 9 Gestor de licencias Vuforia

Base de datos

Para crear una base de datos de objetos, que posteriormente se reconocerán mediante realidad aumentada, se debe pulsar en la pestaña *Target Manager* del portal de desarrolladores de Vuforia. Una vez ahí se pulsa sobre el botón *Add Database* para crear una nueva base de datos. Se solicita el nombre y el tipo, en este caso, para dispositivo.



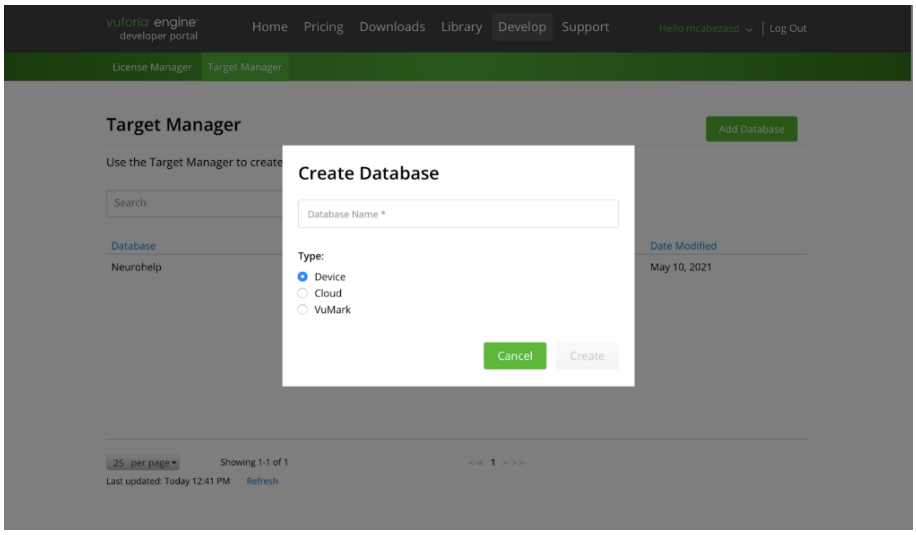


Ilustración 10 Crear base de datos Vuforia

Una vez creada, aparecerá en pantalla la posibilidad de introducir elementos en ella, tal y como se detalla en el siguiente punto.

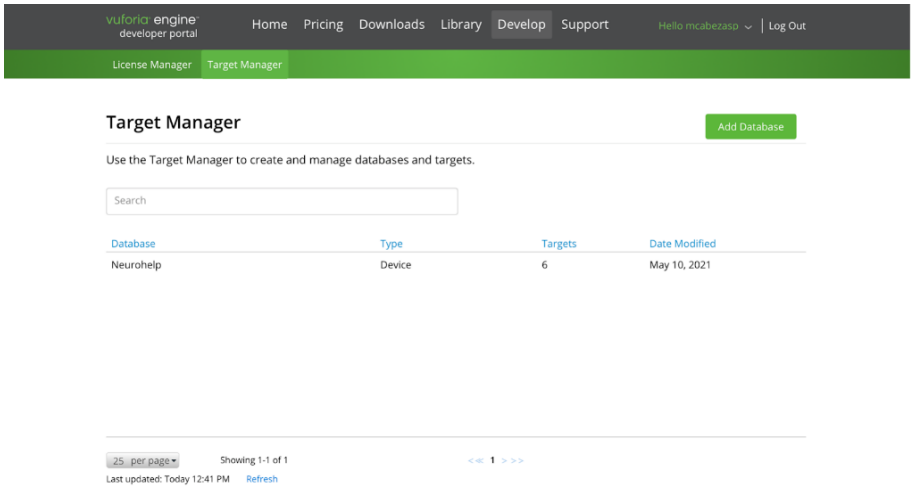


Ilustración 11 Gestor base de datos Vuforia

Agregación de elementos

Para crear un nuevo elemento reconocible por cámara basta con entrar en la base de datos creada y pulsar sobre *Add Target.* Se solicitará un nombre para la plantilla, una imagen, en el caso de seleccionar *Single Image* como patrón de reconocimiento/marcación como es este caso, y una anchura.

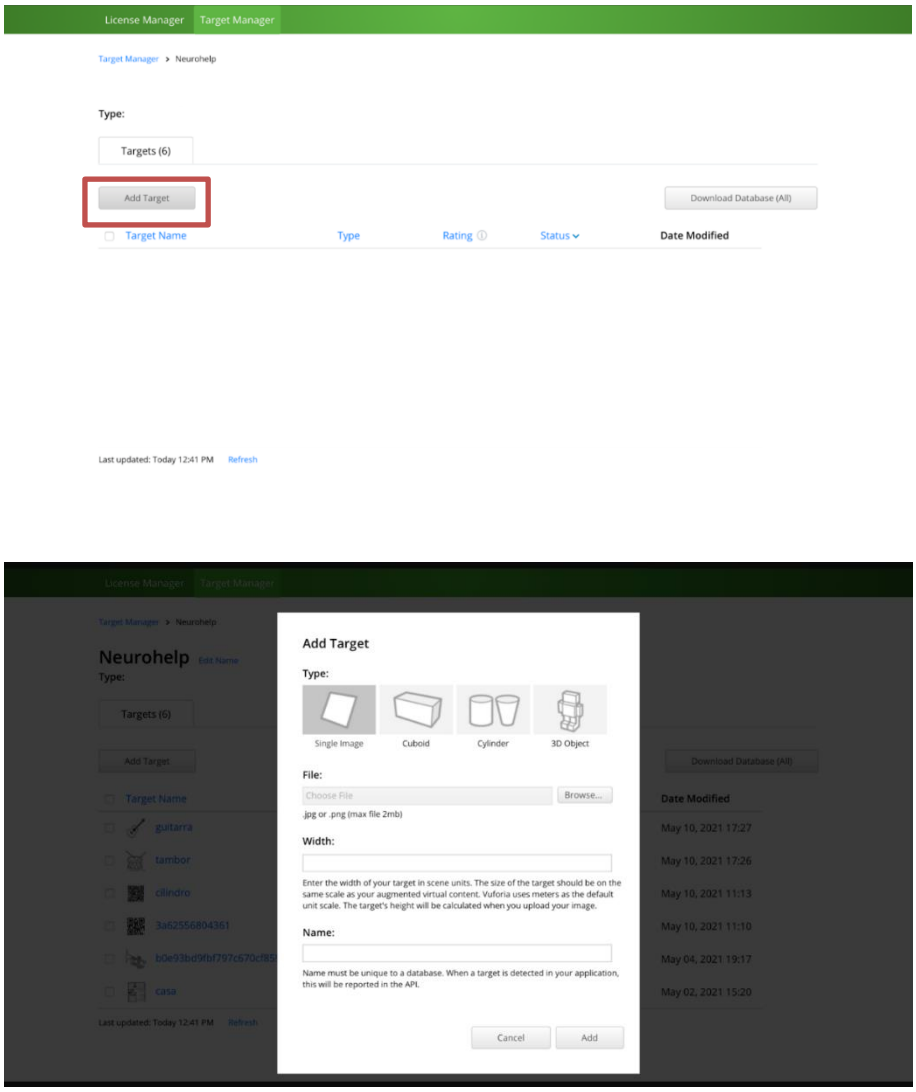


Ilustración 12 Añadir target Vuforia

Una vez relleno, se muestra la plantilla creada junto con su nivel de reconocimiento marcado con estrellas, de forma que, a mayor número de estrellas, más sencillo será su reconocimiento por parte de la cámara.

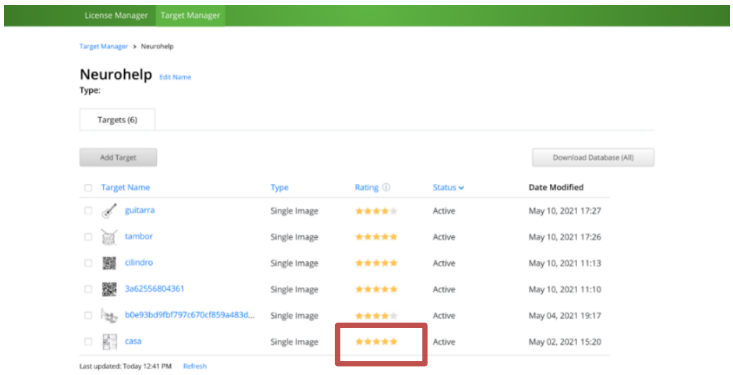


Ilustración 13 Calidad otorgada Vuforia

Descarga de base de datos

Una vez dispuestos todos los marcadores en la base de datos, es necesario descargarla para poder usarla posteriormente en el proyecto. Para ello basta con seleccionar *Download Database(All)* y comenzará la descarga de un paquete Unity que deberá importarse al proyecto tal y como se explica a continuación.

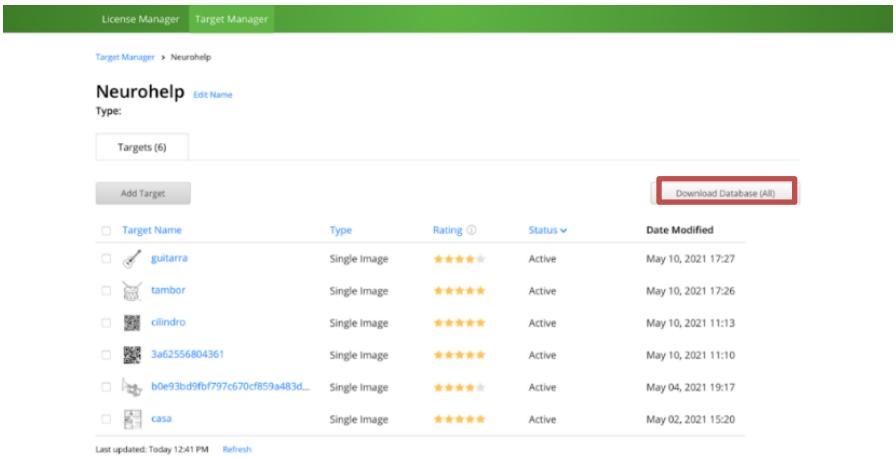


Ilustración 14 Descargar base de datos Vuforia

1.4.3 Unity-Vuforia

Importar base de datos

Para poder utilizar la base de datos anteriormente creada, se debe importar al proyecto Unity, para ello, en el Sistema operativo MacOS, se debe pulsar la pestaña superior *Assets*, seleccionar *Import Package* y una vez ahí, *Custom Package.* A continuación, se elige el paquete previamente descargado.

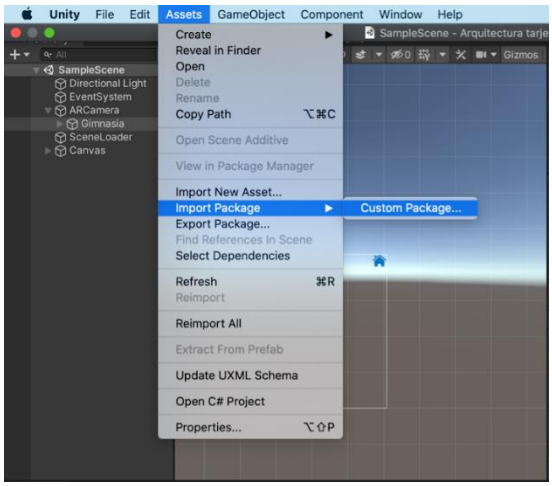


Ilustración 15 Importar base de datos

Introducción al manejo de realidad aumentada

ARCamera

El primer elemento que se debe incorporar para el reconocimiento de los marcadores anteriormente creados es la cámara de realidad aumentada. Este componente lanzará la cámara del dispositivo una vez arranque la escena e inspeccionará la existencia de los marcadores que se le indiquen, como se explicará en el siguiente punto. Para añadir este elemento, basta con pulsar el botón derecho sobre la zona de objetos de la escena de Unity, seleccionar la opción Vuforia y dentro de ella *ARCamera.*

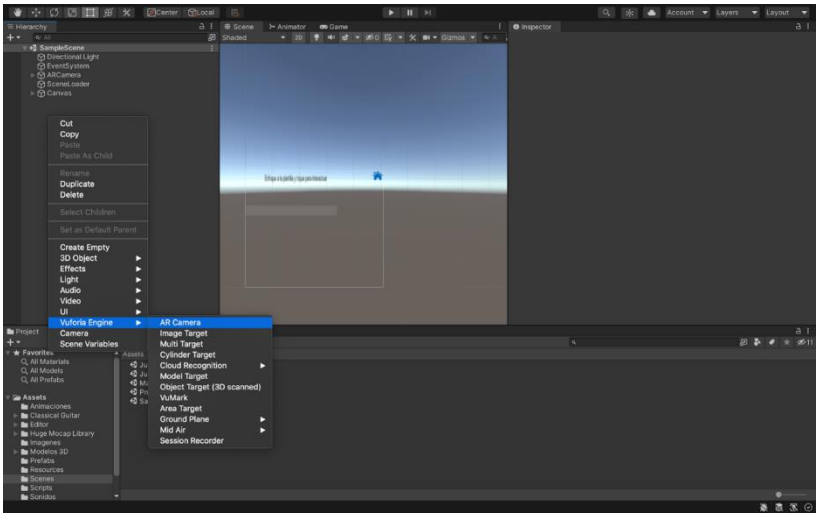


Ilustración 16 Crear ARCamera

Una vez situada en la escena, se accede a su menú de propiedades mediante el inspector situado en la parte derecha de la pantalla. En él, para modificar aspectos relacionados con la realidad aumentada, como la licencia para el uso de Vuforia o el número de objetos que puede detectar a la vez, se debe pulsar en la opción *Open Vuforia Engine Configuration.*

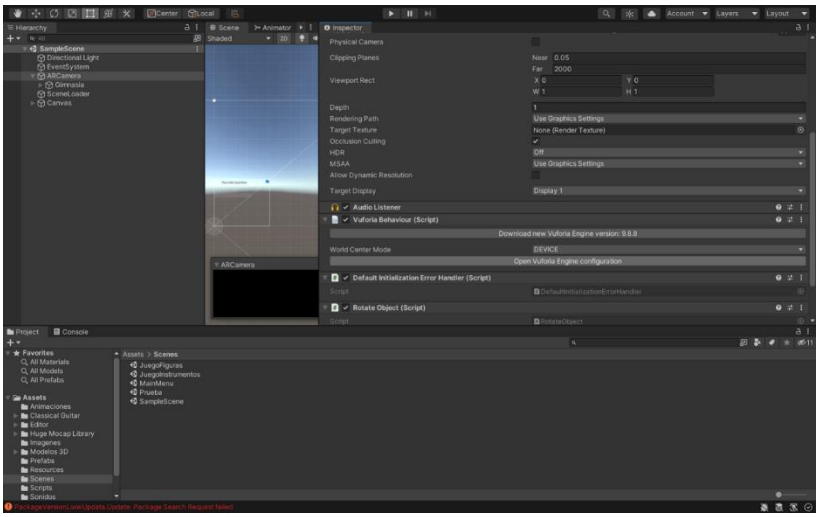


Ilustración 17 Inspector ARCamera

Una vez dentro, se introduce en el cuadro de texto *App Licence Key* la licencia obtenida previamente.

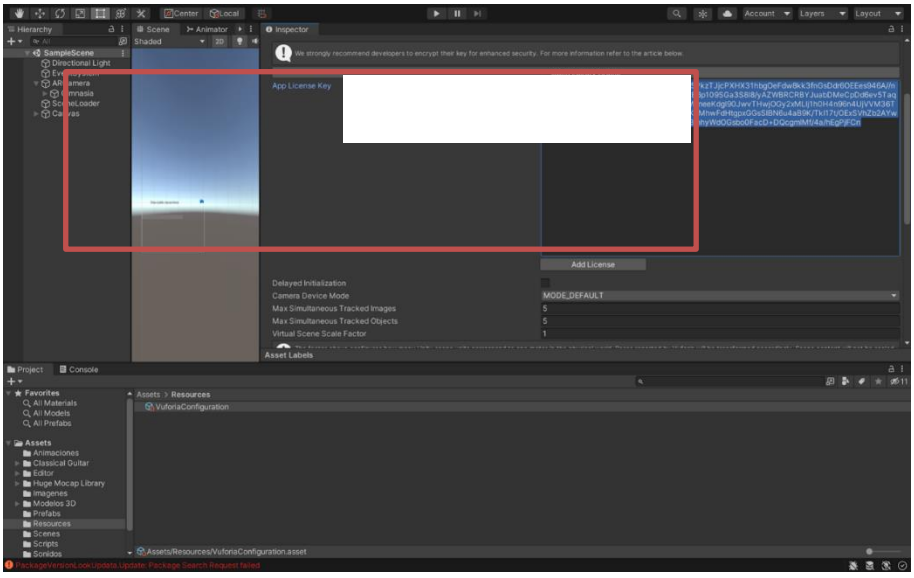


Ilustración 18 License key

Image Target

Una vez creada en escena la cámara de realidad aumentada, se le debe indicar qué plantillas o marcadores debe reconocer. Para ello se anidan en su interior los objetos *Image Target* que se deseen. Estos contendrán la imagen que se debe reconocer y de qué base de datos procede. Para el desarrollo de todas las escenas explicadas con detalle en los siguientes apartados, se ha empleado esta técnica.

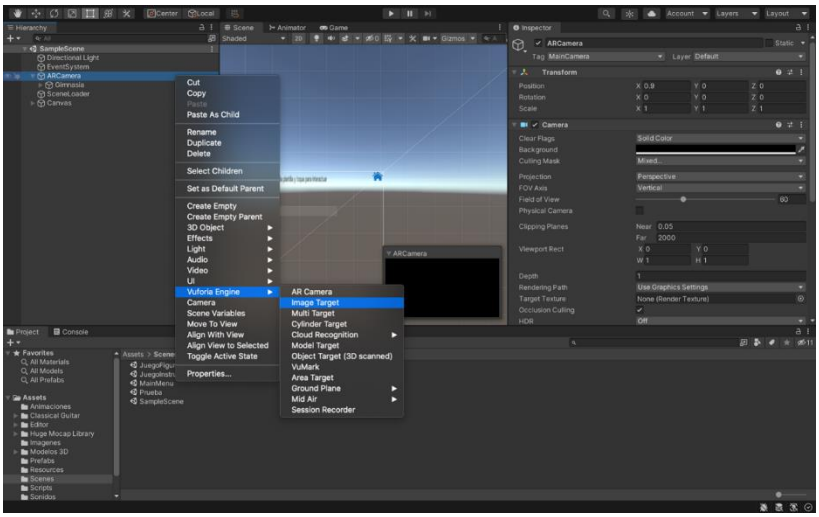


Ilustración 19 Image target

Una vez situado el objeto en la escena, mediante el inspector se indica de dónde y cuál marcador corresponde al objeto.

1.4.4 Unity

Animación e interacción

En anteriores apartados se ha explicado cómo asociar un marcador a la cámara y que ésta lo reconozca, sin embargo, resulta necesario vincular un objeto o modelo 3D de Unity al marcador para que, al ser detectado, se represente. Para ello basta con anidar el modelo 3D deseado al marcador, de igual forma que ocurría entre marcador y cámara.

Para este caso, el modelo 3D representado, así como su animación, se ha descargado de la página Mixamo e importado al proyecto mediante el método *drag and drop*, es decir, arrastrando y soltando en el directorio de carpetas del proyecto.

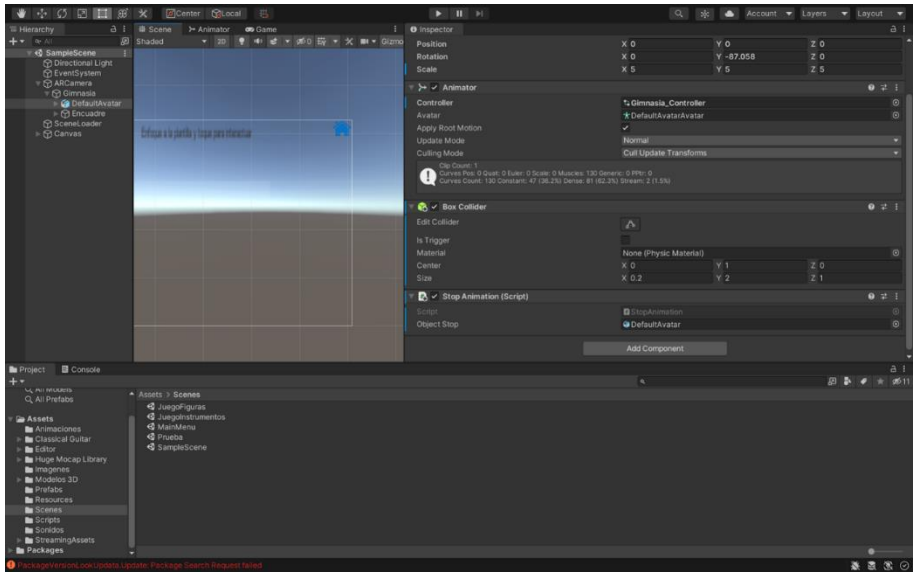


Fig. 1. Importar modelos 3D

Una vez situado correctamente en la escena, mediante la técnica de ensayo y error, se le añade un element *Box Collider* con el objetivo de poder interaccionar si se pulsa sobre él. Este componente actúa a modo de estructura exterior que detecta pulsaciones, colisiones con otros objetos, etc.

Una vez incorporado el elemento, se añade un *script* al modelo para que, al pulsar sobre el personaje, la animación se detenga. En él mediante el método *OnMouseDown,* se detecta la posición sobre el componente *Box Collider* y, en caso de estar la animación activa, se para. De lo contrario, se reanuda la animación.

Además, en este script se controlará la emisión de sonidos. Esto se consigue introduciendo en la escena un objeto *Audio Source* contenedor del clip de audio. Una vez detectado la pulsación sobre la realidad aumentada se tiene que poder pausar y reanudar el sonido. Esta pausa y reanudación se incluye en la detección del evento *onMouseDown* anteriormente comentado. A mayores, para indicar de una forma visual las indicaciones de la realidad aumentada, se ha incluido un vídeo textual que se pausa y reanuda también con la detección de pulsaciones sobre el modelo 3D de la realidad aumentada.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Video;

public class StopAnimation : MonoBehaviour
{
    public GameObject objectStop;


    public GameObject quad;

    VideoPlayer videoPlayer;

    bool animacionActiva = false;

    public void paraSi(){
        if(animacionActiva == true){
            animacionActiva = false;
            objectStop.GetComponent<Animator>().enabled = false;
            videoPlayer.Pause();
            Debug.Log("[StopAnimation-paraSi()]-Animated character stopped successfully");
        }
        else{
            animacionActiva = true;
            objectStop.GetComponent<Animator>().enabled = true;
            videoPlayer.Play();
            Debug.Log("[StopAnimation-paraSi()]-Animated character (re)started successfully");
        }
    }
    // Start is called before the first frame update
    void Start()
    {
        Debug.Log("[StopAnimation-Start()]-Animated motion started successfully");
        videoPlayer = quad.GetComponent<VideoPlayer>();
        objectStop.GetComponent<Animator>().enabled = false;
    }

    // Update is called once per frame
    //void Update()
    /*{
        Debug.Log("update");
    }*/

    public void OnMouseDown()
    {
        Debug.Log("[StopAnimation-onMouseDown()]-Interaction with AR animated character detected");
        paraSi();
    }

```

Ilustración 20 Detección de eventos Unity