Universidad Autónoma Metropolitana
Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería
Licenciatura en Ingeniería en Computación

Proyecto Tecnológico

# Aplicación turística para la identificación de edificios históricos en la CDMX, mediante el uso de realidad aumentada.

Tania Acosta Esparragoza
2132002738

Asesor
Dr. Leonardo Daniel Sánchez Martínez

Trimestre 2018 Invierno

6 de abril de 2018

# Declaratoria

Yo, Leonardo Daniel Sánchez Martínez, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Dr. Leonardo Daniel Sánchez Martínez

Yo, Tania Acosta Esparragoza, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Tania Acosta Esparragoza

# Tabla de contenido

# Índice de Figuras



# Índice de Tablas


## Resumen

VisitAR CDMX es una aplicación móvil desarrollada para beneficiar al sector turístico de la Ciudad de México. Desarrollada para Android, utiliza la tecnología proporcionada por Vuforia, que permite generar Realidad Aumentada basada en seguimiento.

A través del uso de ésta tecnología, reconoce tres edificios históricos emblemáticos para la cultura mexicana: el Palacio de Bellas Artes, la Basílica de Guadalupe y el Monumento a la Revolución. Al reconocerlos, proporciona la información esencial de cada uno, traducida al español, inglés e italiano.

Además, ésta aplicación permite conocer a qué distancia se encuentra el usuario con respecto a los edificios mencionados anteriormente; de ésta forma, si se encuentra cerca del lugar, podría generarle interés en visitarlo.

Utilizar Realidad Aumentada en éste proyecto permite que el usuario interactúe con su entorno, haciendo que su visita sea más entretenida, mientras conoce datos relevantes sobre la historia mexicana.

# 1. Introducción

La Ciudad de México es una zona turística atractiva en nuestro país, ya que abarca una gran cantidad de lugares históricos que son representativos en la cultura mexicana. Es común ver que los turistas deseen conocer datos relevantes sobre la edificación que están visitando; sin embargo, no siempre hay disponible un guía que los oriente, o existe una dificultad de comunicación por el idioma. Éste problema se puede resolver desarrollando una aplicación móvil que proporcione dichos datos.

Actualmente, las aplicaciones móviles son herramientas versátiles que nos permiten realizar una gran cantidad de tareas en nuestra vida diaria. No obstante, la gran mayoría de ellas sólo permiten una interacción entre el usuario y el dispositivo móvil. Con el avance de la tecnología ha aumentado el desarrollo de aplicaciones con nuevas funcionalidades, aprovechando las características de los dispositivos móviles.

La realidad aumentada es un claro ejemplo de ello, ya que ha sido tendencia en los últimos años al ser aplicable en diversos ámbitos. Consiste en capturar el entorno físico existente, agregando elementos virtuales como texto, audio, modelos 3D animados o imágenes, en tiempo real. En el caso de los dispositivos móviles, se requiere que la cámara visualice el entorno real en la pantalla, para que la aplicación sobreponga los elementos virtuales correspondientes.

En este trabajo se muestra el desarrollo de la aplicación turística VisitAR CDMX, la cual utiliza tecnologías de realidad aumentada para el reconocimiento de edificios históricos en la Ciudad de México, mejorando la experiencia del usuario, proporcionándole información relevante con respecto a su visita.

# 2. Antecedentes

Actualmente existen aplicaciones móviles con realidad aumentada que contribuyen al sector turístico del país. Sin embargo, tanto el funcionamiento como su finalidad son diferentes a las de éste proyecto.

Tal es el caso de *Zona ARquelógica* [1], una aplicación basada en geolocalización, que identifica puntos de interés dentro de la zona arqueológica de Teotihuacán. Si bien proporciona información acerca de puntos relevantes, está enfocada en una zona específica del país y no realiza reconocimiento de edificios.

Por otra parte, Microsoft cuenta con la aplicación *HERE City Lens* [2], que ofrece información sobre puntos de interés cercanos al usuario, a través de etiquetas basadas en geolocalización. Su panorama es más general, ya que abarca una gran variedad de sitios como restaurantes y hoteles, por lo que su objetivo no se centra en lugares históricos.

En Salamanca, España, se desarrolló una aplicación móvil [3] como un complemento al recorrido Scala Coeli, que permite subir a las torres de la Clerecía, patrimonio histórico y sede de la Universidad. Esta aplicación está basada en geolocalización y proporciona información sobre edificios cercanos en esa ciudad.

Si bien las aplicaciones anteriores tienen como objetivo brindar información sobre puntos relevantes, no utilizan realidad aumentada basada en seguimiento, ni se centran en los edificios históricos de la Ciudad de México, tal como VisitAR CDMX.

# 3. Justificación

La realidad aumentada se ha vuelto tendencia, ya que es una tecnología que se incorpora fácilmente a los dispositivos móviles y puede ser usada en un amplio rango de aplicaciones.

La sociedad actual está en constante evolución, por lo que es necesario implementar nuevas tecnologías en los dispositivos de uso cotidiano, colaborando con las tareas diarias mediante aplicaciones actualizadas.

De ésta forma, el uso de ésta aplicación apoya al sector turístico de la Ciudad de México, aprovechando las nuevas tecnologías disponibles.

VisitAR CDMX, al ser una aplicación móvil, permite que el turista puede conocer la información esencial sobre los edificios históricos que visite, de manera rápida.

Ésta aplicación presenta las siguientes ventajas:

- Facilidad de uso, ya que sólo es necesario posicionarse frente al edificio y enfocar la cámara hacia él, para realizar el reconocimiento de la imagen y así, presentar la información del mismo.

- Presenta la información del lugar en tres idiomas, haciéndola más entendible.

- El usuario no requiere saber el nombre del edificio para obtener información del mismo.

- Notifica al visitante si se encuentra cerca de algún edificio histórico reconocible, lo que podrá generar interés para visitarlo.

# 4. Objetivos

A continuación, se enlistan los objetivos a alcanzar durante el desarrollo de éste proyecto.

## 4.1 Objetivo General

Diseñar una aplicación turística para la identificación de edificios históricos en la CDMX, mediante el uso de realidad aumentada.

## 4.2 Objetivos Específicos

- Diseñar e implementar el módulo de realidad aumentada, para la identificación de cada edificio.
- Diseñar e implementar el módulo para administrar la información de los edificios.
- Diseñar e implementar el módulo de localización, para conocer la ubicación del usuario mediante GPS.
- Diseñar e implementar el módulo para el manejo de idiomas.

# 5. Marco teórico

Para el desarrollo de este proyecto, es indispensable conocer qué es y cómo funciona la Realidad Aumentada, por lo cual se detallará en ésta sección.

## 5.1 Realidad Aumentada

La Realidad Aumentada es una tecnología que permite interponer contenido virtual multimedia entre el mundo real y el usuario, creando la ilusión de que ambos universos, el real y el virtual, están combinados.

Existen diversas definiciones con respecto a qué es la Realidad Aumentada; Azuma [4] la define como un sistema con las siguientes características:

- Combina elementos reales y virtuales.
- Es interactiva en tiempo real.
- Se muestra en un espacio 3D.

La realidad aumentada se divide principalmente en dos técnicas, las cuales se describen a continuación:

### 5.1.1 Realidad Aumentada basada en seguimiento

Esta técnica hace uso de marcadores o imágenes para determinar la posición de la información a desplegar. El dispositivo, mediante un software de Realidad Aumentada, rastrea las imágenes en busca de patrones haciendo los cálculos necesarios para desplegar el contenido correspondiente.

### 5.1.2 Realidad Aumentada basada en geolocalización

Esta técnica utiliza las herramientas incorporadas a los dispositivos móviles, como son el GPS y el acelerómetro, para obtener las coordenadas geográficas del dispositivo. Posteriormente, despliega en pantalla los puntos de interés cercanos al dispositivo.

En éste proyecto se utilizó la técnica basada en seguimiento, ya que el reconocimiento de los edificios se realiza mediante imágenes.

# 6. Herramientas para el desarrollo

Las herramientas para desarrollo de este proyecto, así como sus características, se detallan en ésta sección.

## 6.1 Vuforia

Vuforia [5] es un kit de desarrollo de software (*SDK*) que permite la creación de aplicaciones con realidad aumentada. Utiliza la tecnología *Computer Vision* para reconocer y rastrear imágenes objetivo (*Image Targets*) y objetos 3D simples, entre otros, en tiempo real.

Esta capacidad de reconocimiento de imágenes permite a los desarrolladores ubicar y orientar objetos virtuales, en relación con las imágenes tomadas del mundo real cuando se ven a través de la cámara de un dispositivo móvil. Vuforia proporciona *APIs* para programación en lenguajes *C++, Java y C#*. De esta manera, el *SDK* es compatible para el desarrollo de aplicaciones tanto en *iOS* como en *Android*.

Vuforia proporciona dos tipos de base de datos para el reconocimiento de imágenes objetivo, las cuales son:

### 6.1.1 Vuforia *Device Databases*

Proporciona una base de datos local a la aplicación con Realidad Aumentada. La base de datos debe ser llenada antes de ser descargada para integrarla a la aplicación, por lo que, una vez compilada la aplicación, la base de datos no podrá ser modificada.

### 6.1.2 Vuforia *Cloud Recognition Service*

Este servicio [6] permite que la aplicación reconozca imágenes objetivo a través de una base de datos en la nube, dando la capacidad de actualizar las imágenes dinámicamente, de manera externa a la aplicación. Sin embargo, solo es posible asociar una base de datos por aplicación.

Para la realización del proyecto, se eligió el Servicio de *Cloud Recognition*, para almacenar las diferentes imágenes relacionadas a cada edificio. Al poder actualizar la base de manera externa, es posible agregar nuevas imágenes de cada edificio sin necesidad de recompilar la aplicación.

La implementación de esta base de datos se profundiza en la sección **7.6.1 Creación de la base de datos *Cloud Reco*.**

## 6.2 Unity 3D

Unity [7] es un motor de videojuegos multiplataforma utilizado principalmente para la creación de videojuegos 2D y 3D para computadoras, consolas y dispositivos móviles.

El lenguaje de programación utilizado es *C#*, a través de scripts basados en Mono, una implementación de código abierto para *.NET Framework*.

Está dirigido para las siguientes *APIs* gráficas: *Direct3D, OpenGL, OpenGL ES y WebGL*. Además, está disponible para las principales plataformas de videojuegos, como son *iOS, Android, Windows, PlayStation, Xbox One, Wii U*, entre otras.

A partir de la versión 2017.2, Vuforia se integra a Unity como el creador de AR por defecto.

# 7. Desarrollo del proyecto

En éste apartado se detalla el desarrollo de la aplicación VisitAR CDMX. Para referencias de los scripts desarrollados, ver la sección **12. Apéndice A: Código fuente.**

## 7.1 Configuración del proyecto en Unity

La aplicación está desarrollada para dispositivos móviles con sistema operativo Android, por lo que hay que seleccionar dicha plataforma en *File* > *Build Settings*…, como se muestra en la Figura 1.

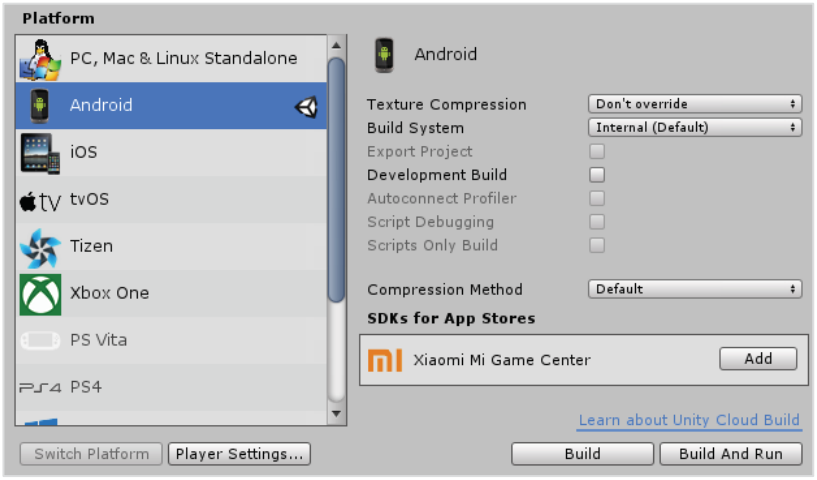


Figura 1. Selección de la plataforma Android.

Para utilizar el SDK de Vuforia, es necesario habilitarlo en *File* > *Build Settings*… > *Player Settings*… > *XR Settings*, como se muestra en la Figura 2.

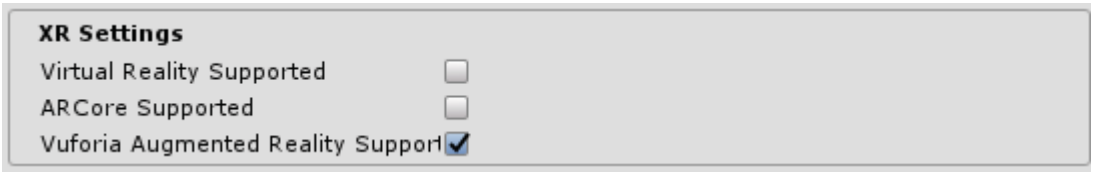


Figura 2. Vuforia habilitado en el proyecto de Unity.

Dentro de *Player Settings* > *Configuration* > *Scripting Runtime Version* debe seleccionarse también el Framework .NET versión 3.5, como se muestra en la Figura 3. De ésta forma, se podrá trabajar con los scripts en C#.

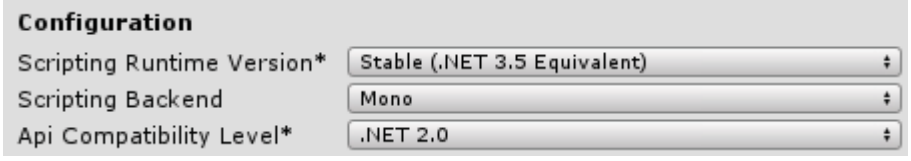


Figura 3. Versión de .NET Framework.

### 7.1.1 Vuforia *development key*

Para hacer uso de los servicios proporcionados por Vuforia, es necesario registrarse en www.developer.vuforia.com. El registro es gratuito y de ésta forma pueden utilizarse los servicios a nivel de desarrollo.

El siguiente paso es obtener una llave para asociar el servicio con la aplicación. En *License Manager*, seleccionamos la opción *Get Development Key,* como se muestra en la Figura 4.

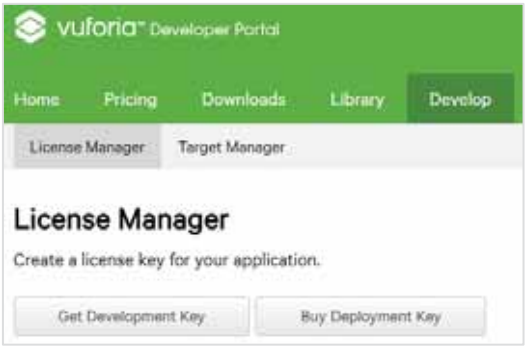


Figura 4. Licencia gratuita de desarrollo.

Posteriormente, introducimos un nombre para la misma y damos clic en *Confirm*. Se creará la licencia de desarrollo, tal como se muestra en la Figura 5.

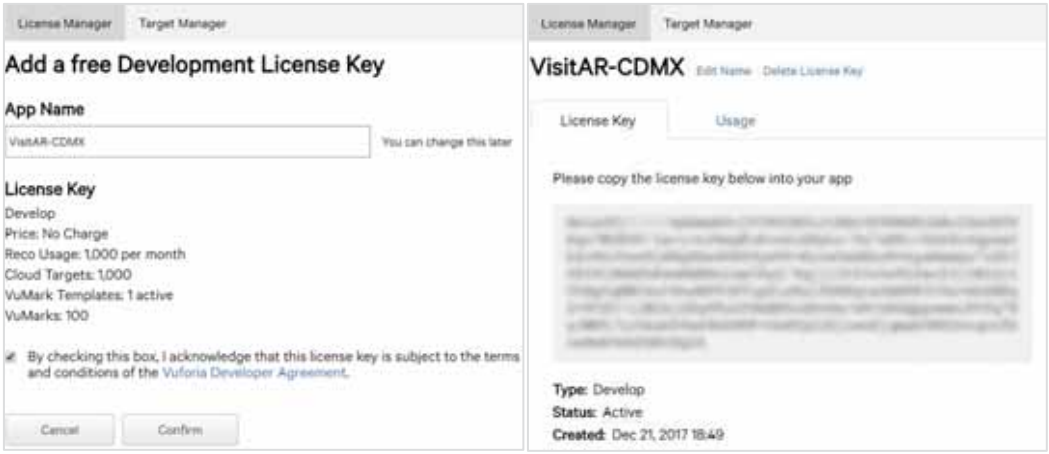


Figura 5. Creación de la licencia de Vuforia.

Una vez creada podremos copiar la licencia generada a nuestro proyecto en Unity, abriendo el menú *Window* > *Vuforia Configuration*, como se muestra en la Figura 6.

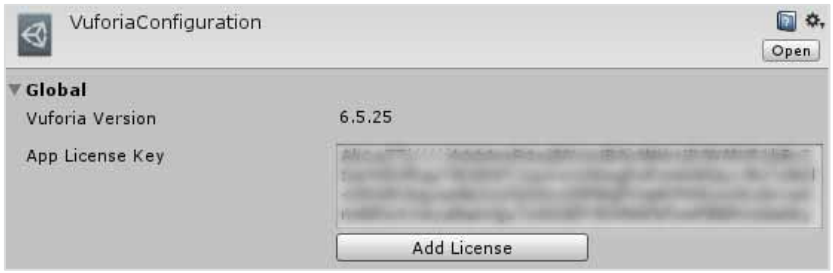

Figura 6. Licencia colocada en el proyecto de Unity.

De ésta forma, el proyecto está configurado para iniciar el desarrollo de la aplicación.

## 7.2 Escenas de la aplicación

La aplicación, desarrollada en Unity, se divide en cinco escenas, las cuales se muestran en el esquema de navegación de la Figura 7.

Así, las escenas Ubicación, Realidad Aumentada y Tutorial se acceden mediante botones en la escena Menú Principal, mientras que, la escena Información Edificio se accede únicamente cuando se reconoce un edificio en Realidad Aumentada.

Al presionar el botón Atrás del dispositivo móvil:

- Las escenas derivadas regresan al Menú Principal
- La escena Menú Principal cierra la aplicación

El script *KeyController.cs* describe éste comportamiento en las líneas 38-59.

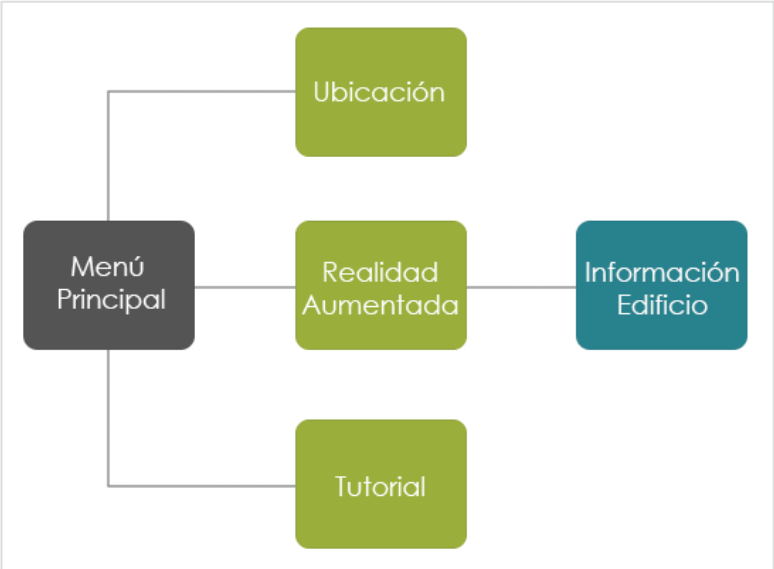

Figura 7. Esquema de navegación para las escenas de la aplicación.

De ésta forma, los módulos establecidos en los objetivos específicos se implementaron en las escenas correspondientes, como se muestra en la Tabla 1.

Tabla 1. Relación entre las escenas desarrolladas y los módulos especificados.

| Escena | Módulo |
|---|---|
| Ubicación | Módulo de Ubicación |
| Realidad Aumentada | Módulo de Realidad Aumentada |
| Información Edificio | Módulo de Administración de la Información<br>Módulo de Idiomas |

### 7.2.1 Navegación entre escenas

Unity asigna un número entero consecutivo a cada una de las escenas activas en el proyecto, como se muestra en la Figura 8. La escena cero es la que se mostrará al iniciar la aplicación.

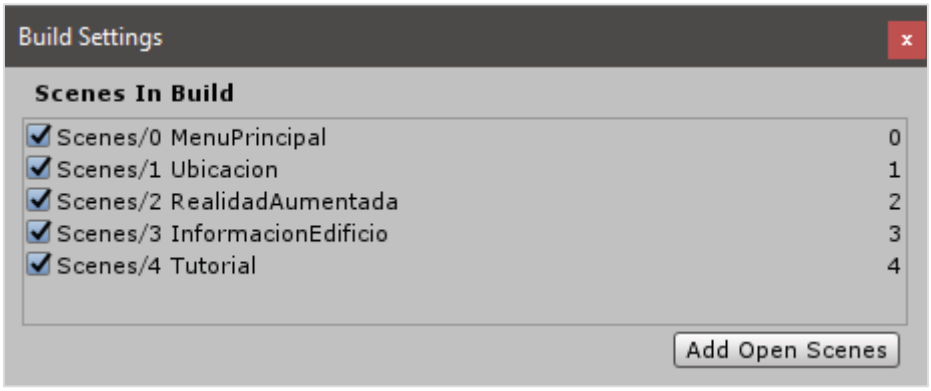


Figura 8. Escenas activas en el proyecto.

Cada escena tarda un tiempo variable en ser cargada y desplegada, dependiendo de la cantidad de objetos y scripts que deban ser cargados para la ejecución de la misma.

Por lo anterior, es necesario implementar la carga de cada escena de forma asíncrona, mediante la ejecución de una *coroutine*. Durante la carga de la escena, se muestra una pantalla de *loading*, tal como se muestra en la Figura 9.

El script *CargarEscena.cs* muestra cómo se maneja el comportamiento mencionado.


Figura 9. Pantalla de *loading* mostrada durante la carga de cada escena.

### 7.2.2 Canvas y pantallas responsivas

Para diseñar las vistas de la aplicación, es indispensable agregar un objeto Canvas a la escena (objeto padre), el cual contendrá todos los elementos gráficos que se mostrarán en la GUI (objetos hijo).

Cada objeto contenido en el canvas puede ser configurado para mantener o cambiar su posición y tamaño, en función del tamaño y orientación de la pantalla del dispositivo donde se ejecute la aplicación.

Esta configuración se realiza a través de las propiedades *Pivot* y *Anchors,* contenidas en *RecTransform*.

Las modificaciones de rotación, tamaño y escala ocurren alrededor del pivote, mientras que *anchors* ajusta la posición y escala del objeto en función del *RecTransform* de su padre, como se muestra en la Figura 10.

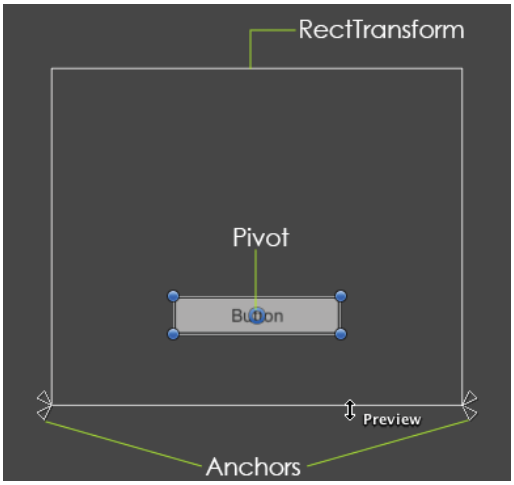


Figura 10. Propiedades de *RectTransform*.

De ésta manera, las vistas se ajustan a la resolución de la pantalla de manera responsiva.

## 7.3 Lectura de archivos

Los datos relacionados a cada edificio, tales como su información traducida y las coordenadas de su ubicación, se encuentran almacenados en archivos .txt dentro de la carpeta *Resources*, en los *Assets* del proyecto Unity.

La información está dividida en tres archivos, tal como se describe a continuación:

- **Edificio.txt**. Contiene el nombre del edificio, así como una clave que lo identifica.
- **InformacionEdificio.txt**. Contiene la información de cada edificio traducida en los tres lenguajes disponibles: español, inglés e italiano.
- **UbicacionEdificio.txt**. Contiene las coordenadas de latitud/longitud de cada edificio.

La información histórica de cada edificio fue consultada en las referencias [8]-[13], siendo traducida al español e italiano[1].

Los tres edificios disponibles, así como su clave se muestran en la Tabla 2.

Tabla 2. Edificios disponibles.

| Clave | Edificio |
|---|---|
| visitAR_ed1 | Basílica de Guadalupe |
| visitAR_ed2 | Palacio de Bellas Artes |
| visitAR_ed3 | Monumento a la Revolución |

La información de cada archivo es cargada y almacenada temporalmente en listas de objetos, tal como se muestra en la clase *CargarArchivos.cs.*

Los objetos corresponden a las clases Edificio, InformacionEdificio y UbicacionEdificio, pertenecientes al modelo del dominio, que pueden ser consultadas en la sección **12.2 Modelo del Dominio.**

Las escenas Ubicación e Información Edificio acceden a la información cargada utilizando los métodos definidos en la clase *ConsultaArchivo.cs*, los cuales se muestran en la
Tabla 3.

[1] Traducción realizada por Uriel Estrada Padilla.

Tabla 3. Métodos disponibles para el acceso a la información.

| Modificador y tipo | Método y descripción |
|---|---|
| `static Edificio` | `GetEdificio (String idEdificio)`<br>Devuelve un objeto de tipo Edificio, identificado a partir de la clave idEdificio. |
| `static List<InformacionEdificio>` | `GetInformacionEdificio (String idEdificio)`<br>Devuelve una lista de objetos InformacionEdificio. Cada uno almacena la información del edificio solicitado, traducida a cada idioma. |
| `static List<UbicacionEdificio>` | `GetUbicacionesEdificio ()`<br>Devuelve una lista de objetos UbicacionEdificio, de todos los edificios disponibles. Cada uno contiene las coordenadas de ubicación del edificio. |

## 7.4 Menú Principal

Es la escena principal de la aplicación, la cual muestra las instrucciones básicas para el uso de la misma, y proporciona el acceso al resto de las escenas, tal como se muestra en la Figura 11.

Figura 11. Pantalla Principal.

### 7.4.1 Inicialización del Servicio de Localización

La escena contiene la inicialización del servicio de localización, que desencadena la ejecución de los servicios para el cálculo de distancias y las notificaciones.

Nótese que, al cambiar de escena, todos los objetos contenidos en la última escena activa se destruyen. Esto genera un problema cuando se requiere que uno o más objetos se mantengan activos durante la ejecución de la aplicación, sin importar la escena activa.

Para solucionar lo anterior, es necesario que por cada objeto se cree una única instancia (aplicando el patrón *Singleton*), que se mantendrá activa a lo largo de todas las escenas, mediante la función `DontDestroyOnLoad(this)`.

Aplicando lo anterior, los tres objetos mostrados en la Figura 12, se mantendrán activos, permitiendo que el cálculo de distancias y el servicio de notificaciones actualicen su información constantemente.

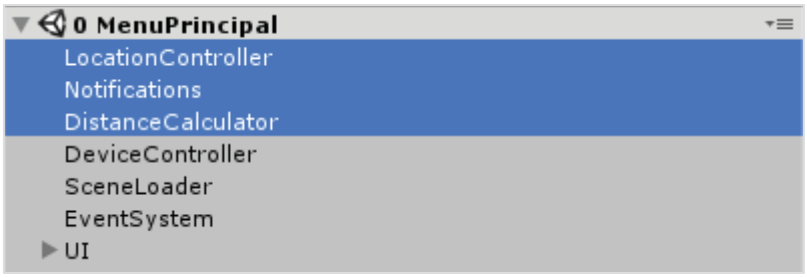


Figura 12. Objetos correspondientes al servicio de localización.

La implementación de los servicios mencionados se detalla en la sección **7.5 Ubicación.**

### 7.4.2 Imagen de fondo móvil con acelerómetro

Para un mejor efecto visual, se colocó una imagen de fondo móvil, que se desplaza sobre el eje x, tal como se muestra en el diagrama de la Figura 13. El desplazamiento se realiza al detectar movimiento a través del acelerómetro del dispositivo. Para acceder a la coordenada x del acelerómetro, se utiliza el valor contenido en la clase `Input.acceleration.x`.

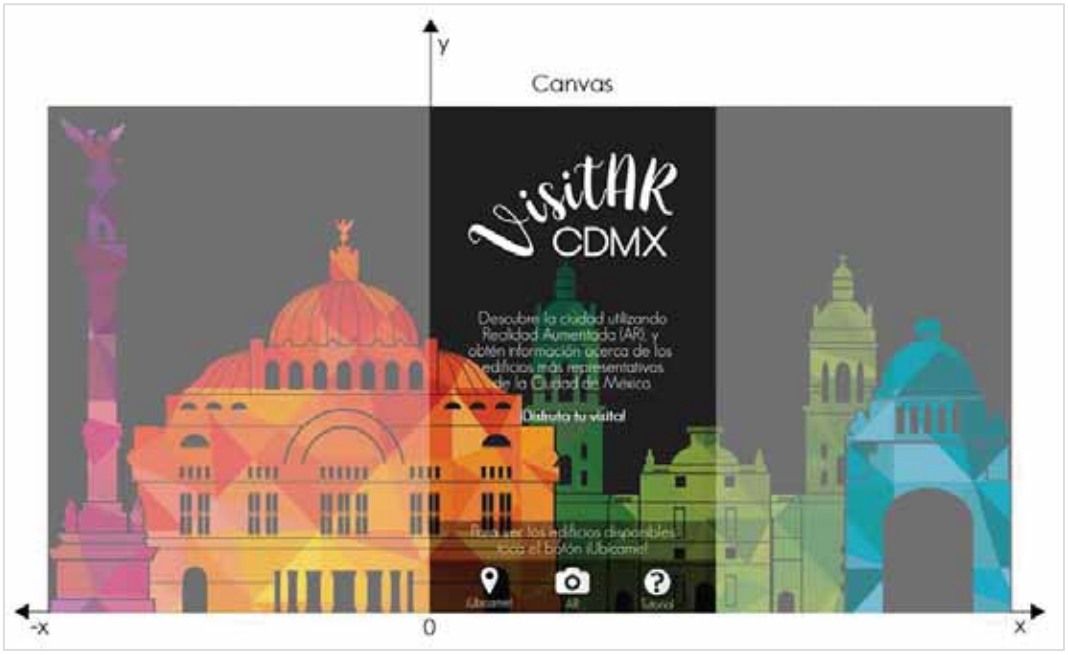


Figura 13. Diagrama ilustrativo del desplazamiento de la imagen.

Las escenas Menú Principal y Ubicación comparten dicho fondo. El script *MoverImagenFondo.cs* contiene los cálculos necesarios para realizar el desplazamiento en función del tamaño tanto del canvas como de la imagen.

## 7.5 Ubicación

La escena Ubicación muestra a qué distancia se encuentra el usuario de los edificios reconocibles por la aplicación.

### 7.5.1 Servicio de localización

El servicio de localización permite obtener la ubicación actual del dispositivo móvil, a través del GPS.

Este servicio se obtiene a través de la clase *`Input.location`*, la cual permite habilitar o deshabilitar el servicio, así como conocer el estado y coordenadas actuales, tal como se muestra en la Tabla 4.

Tabla 4. Descripción del servicio de localización.

| Instrucción | Descripción |
|---|---|
| `Input.location.Start(float p, float d)` | Inicializa el servicio de localización.<br><br>p - precisión del GPS en metros.<br>d - distancia en metros recorrida para realizar una actualización. |
| `Input.location.Stop()` | Detiene el servicio de localización. |
| `Input.location.status` | Devuelve el estado actual del servicio. |
| `Input.location.isEnabledByUser` | Es verdadero si el GPS está habilitado en el dispositivo. |
| `Input.location.lastData.latitude` | Devuelve el valor de la latitud. |
| `Input.location.lastData.longitude` | Devuelve el valor de la longitud. |

Algunas cuestiones a considerar:

- El servicio no inicia inmediatamente, por lo que es necesario implementar un mecanismo para el tiempo de espera.

- Es indispensable que el GPS esté encendido en el dispositivo, de lo contrario no se podrá inicializar el servicio.

- Además, es indispensable contar con conexión a internet, de lo contrario los datos:

- Podrían estar desactualizados, esto debido a que el GPS almacena los últimos valores detectados de forma temporal.

- Podrían no estar disponibles, deteniendo la actualización de los servicios dependientes de los mismos.

En la aplicación, el servicio inicializa con los parámetros por defecto, es decir, 10 metros de precisión, así como 10 metros de distancia recorrida para actualizar la información.

El tiempo de espera para el inicio del servicio se estableció en 50 segundos, tiempo suficiente para iniciar el servicio, contemplando que existe una conexión a internet activa.

El algoritmo para iniciar el servicio de localización puede consultarse en el script *ServicioLocalizacion.cs.*

### 7.5.2 Manejo de errores

La inicialización del servicio puede no ser exitosa, lo cual podría generar errores relacionados con estado del GPS, la conexión a internet o los permisos no otorgados en el sistema operativo. Éstos errores son manejados por la aplicación, como se muestra en la Tabla 5.

Tabla 5. Posibles errores generados al tratar de iniciar el servicio.

| Error | Relacionado con | Mensaje |
|---|---|---|
| GPS deshabilitado | GPS | Por favor, activa el GPS e inténtalo de nuevo. |
| Tiempo de espera agotado | Conexión a internet | Se agotó el tiempo de activación del servicio de localización. |
| Permiso denegado por el usuario | Sistema operativo Android | Se denegó el acceso al servicio de localización. |

Cualquiera de los errores anteriormente mencionados ocasiona que la escena de Ubicación no pueda ser accedida hasta que el error sea corregido, como se muestra en la Figura 14.


Figura 14. Ejemplos de errores generados.

El manejo de errores puede ser consultado en el script *ServicioLocalizacion.cs,* líneas 97 - 102 y en *LocationErrorHandler.cs.*

### 7.5.3 Cálculo de distancias

La aplicación cuenta con un servicio de cálculo de distancias, que devuelve la distancia en kilómetros entre el dispositivo y cada uno de los edificios reconocibles.

Este servicio depende directamente del servicio de localización y no se activará si la localización no está disponible.

Las coordenadas de latitud/longitud del dispositivo (origen) se obtienen a través del servicio de localización. Las coordenadas de latitud/longitud de cada edificio (destino) están almacenadas en el archivo *UbicacionEdificio.txt*, como se describe en la sección **7.3 Lectura de archivos**.

Para realizar el cálculo se utiliza la fórmula de Haversine [14], la cual permite calcular la distancia entre dos puntos, considerando la curvatura terrestre. El radio ecuatorial de la tierra es equivalente a 6367 km.

La fórmula de Haversine es la siguiente:

$$d = 2r \arcsin\left(\sqrt{sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + cos(\phi_1)\, cos(\phi_2)\, sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

Dónde:

$\phi_1 = latitud\ origen\ en\ radianes$
$\phi_2 = latitud\ destino\ en\ radianes$
$\lambda_1 = longitud\ origen\ en\ radianes$
$\lambda_2 = latitud\ destino\ en\ radianes$
$r = radio\ de\ la\ tierra\ en\ kilómetros$

En la escena Ubicación se muestran las distancias resultantes ordenadas de manera ascendente. Esto quiere decir que el primer edificio de la lista será el más cercano al usuario.

Para conocer la implementación detallada del cálculo de distancias, ver el script *CalcularDistancia.cs,* líneas 121 - 147.

### 7.5.4 Notificaciones

El servicio de notificaciones muestra una notificación en el dispositivo cuando se detecta una distancia menor a 1 kilómetro entre el usuario y algún edificio.

Este servicio depende directamente del servicio de cálculo de distancias y no se activará si la localización no está disponible.

Para la implementación de las notificaciones se hizo uso del *plugin* gratuito *Simple Android Notifications* [15], que permite crear notificaciones pre configuradas. La implementación para mostrar notificaciones se encuentra detallada en el script *Notificaciones.cs*

## 7.6 Realidad Aumentada

La escena Realidad Aumentada permite el reconocimiento de los edificios, a través de la cámara del dispositivo. A continuación, se detalla su implementación.

### 7.6.1 Creación de la base de datos *Cloud Reco*

Para crear la base de datos *Cloud Reco* se accede al portal *Developer de Vuforia*, y damos clic en la pestaña *Target Manager*. A continuación, damos clic en el botón *Add Database*, como se muestra en la Figura 15.

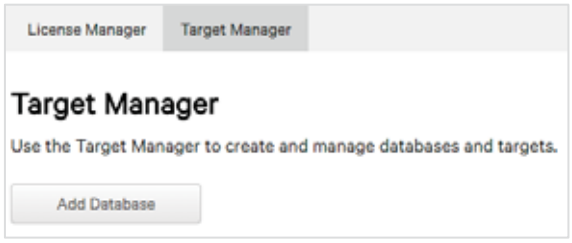


Figura 15. Añadir una nueva base de datos.

Se abrirá un recuadro donde se debe de introducir el nombre de la base de datos, y seleccionar la base de datos tipo *Cloud*. A continuación, seleccionamos la licencia creada en la sección **7.1.1 Vuforia *development key*** y damos clic en *Create*, como se muestra en la Figura 16.

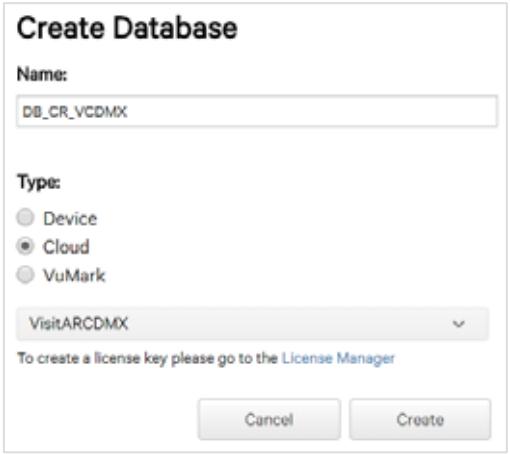


Figura 16. Creación de la base de datos.

Una vez creada, podremos acceder a las llaves necesarias para utilizar el servicio de *Cloud Reco*, mostradas en la Figura 17. Al igual que la licencia de Vuforia, es necesario copiar dichas llaves dentro de la aplicación.

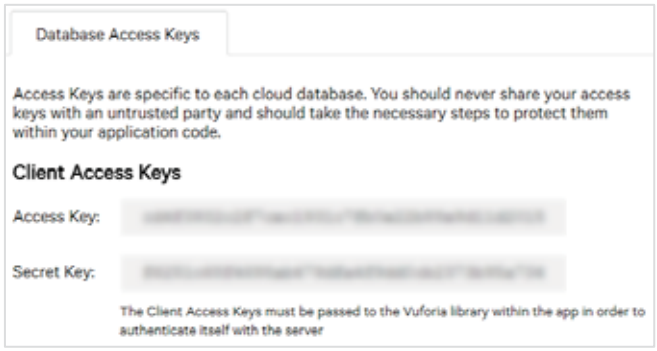


Figura 17. Llaves generadas para el servicio *Cloud Reco*.

Dentro del proyecto, seleccionamos el objeto *CloudRecognition,* y colocamos las llaves generadas dentro del componente *Cloud Reco Behaviour*, tal como se muestra en la Figura 18.

La descripción detallada con respecto los objetos que deben agregarse a la escena para implementar Realidad Aumentada, se encuentra en la sección **7.6.3 Configuración de la escena para el uso de Cloud Reco.**

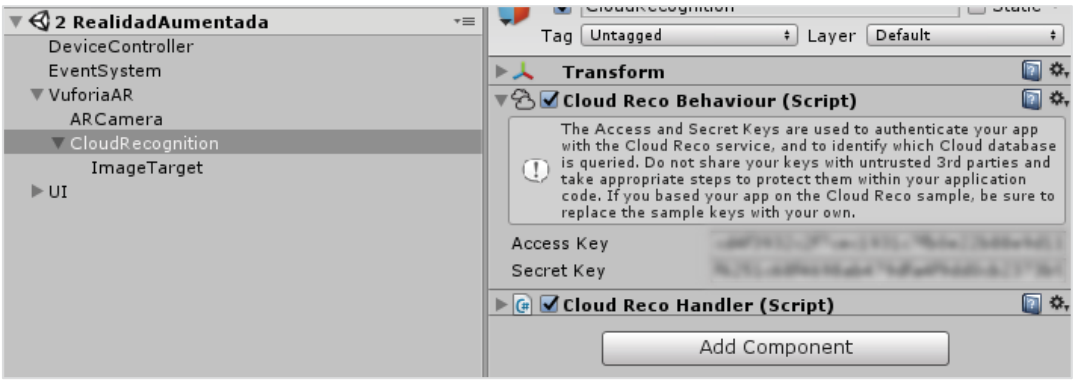


Figura 18. Llaves introducidas en el proyecto de Unity.

## 7.6.2 Image Target y Metadata

Una vez configurada la base de datos, es posible agregar las imágenes objetivo que serán reconocidas por Vuforia. Para esto, se selecciona la pestaña *Targets* y, a continuación, se da clic en *Add Target,* como se muestra en la Figura 19.

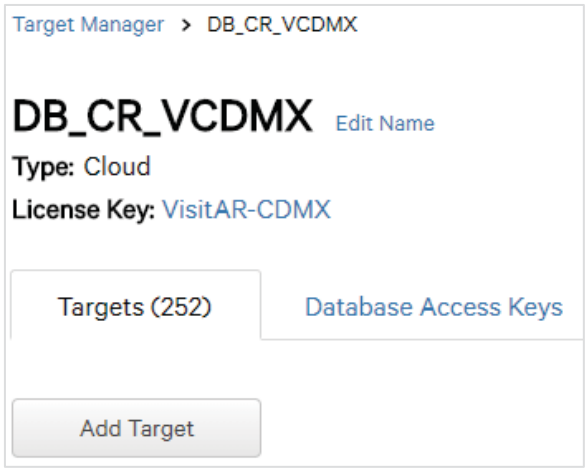


Figura 19. Agregar una nueva imagen objetivo.

Se despliega un recuadro donde se debe de seleccionar la imagen a cargar y definir las características de la imagen objetivo, como se muestra en la Figura 20.

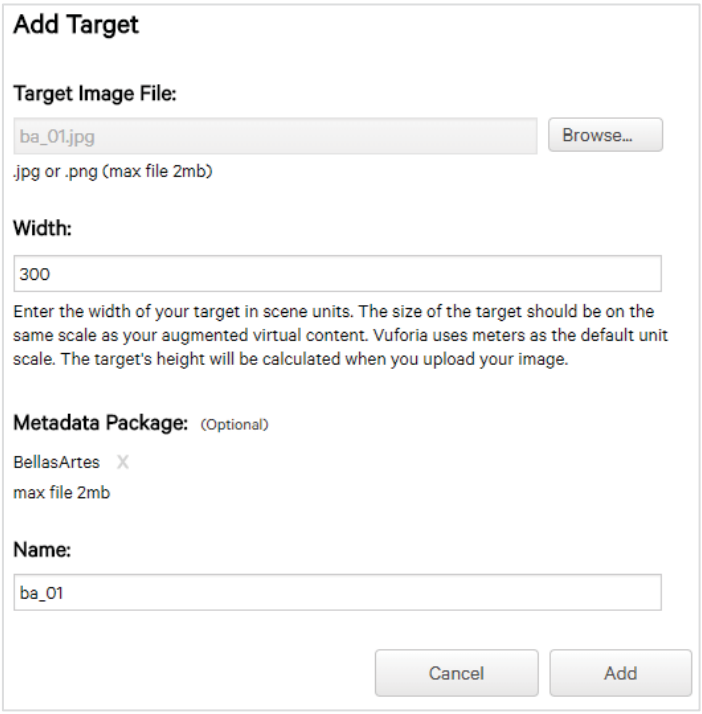


Figura 20. Creación de la imagen objetivo.

Las características a definir son las siguientes:

***Width***: Se refiere al ancho del objeto *ImageTarget* dentro del proyecto, el cual se estableció con un valor de 300. Al no desplegarse ningún contenido multimedia sobrepuesto al edificio, éste valor es irrelevante.

***Metadata Package***: Cada imagen puede tener relacionado un archivo de metadatos, el cual provee información sobre la imagen cargada.

En éste caso, a cada imagen se le asignó un archivo .txt que contiene la clave del edificio que le corresponde, como se describe en la Tabla 2.

Los nombres asignados para cada imagen, divididos por edificio, se muestra en la Tabla 6.

Tabla 6. Asignación de nombre de imagen conforme el edificio.

| Edificio | Nombre |
|---|---|
| **Basílica de Guadalupe** | bg_numeroImagen |
| **Palacio de Bellas Artes** | ba_numeroImagen |
| **Monumento a la Revolución** | mr_numeroImagen |

Al cargar y procesar la imagen, se muestra el *Rating* de la imagen, así como el número de *Recos* del mes, como se muestra en la Figura 21.

El *Rating* nos indica si la calidad de la imagen es buena para realizar reconocimiento sobre ella. Por otra parte, *Recos* muestra el número de veces que la imagen ha sido reconocida por la aplicación en el mes actual, limitado a 1000 reconocimientos al mes por imagen.


Figura 21. Imágenes agregadas a la base de datos

Vuforia, al procesar la imagen, asigna marcas que en conjunto forman un patrón, para luego ser reconocidas por la cámara del dispositivo. Estos patrones pueden visualizarse al abrir un *target*, seleccionando la opción *Show Features,* como se muestra en la Figura 22. La cantidad de marcas añadidas a la imagen está directamente relacionada al *Rating* de la imagen.

Figura 22. Patrón reconocible para el Palacio de Bellas Artes.

### 7.6.3 Configuración de la escena para el uso de *Cloud Reco*

La escena contiene los *game objects* proporcionados por Vuforia, necesarios para utilizar el servicio de *Cloud Reco,* como se puede ver en la Figura 23. Estos objetos ya están pre configurados para su uso, y su comportamiento puede modificarse mediante scripts.

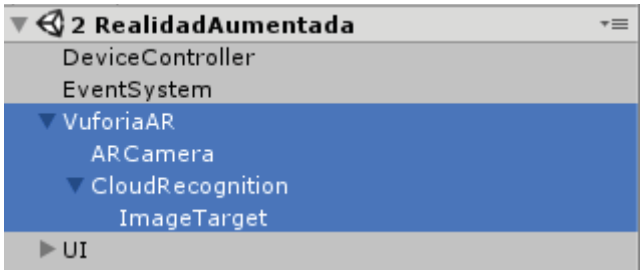


Figura 23. *Game objects* pertenecientes al SDK Vuforia.

La descripción de cada uno se muestra en la Tabla 7.

Tabla 7. Descripción de cada *game object*.

| ***Game object*** | **Descripción** |
|---|---|
| ARCamera | Es un tipo especial de cámara diseñada para aplicaciones que funcionan con realidad aumentada, ya sea para dispositivos móviles o lentes digitales. |
| CloudRecognition | Permite agregar las claves necesarias para el uso del servicio. Además, es el responsable de manejar el comportamiento de la aplicación una vez que se detecta una imagen. |
| ImageTarget | Representa las imágenes que el SDK Vuforia puede detectar y seguir. Una vez que la imagen es detectada, el SDK seguirá a la imagen siempre que esté al menos parcialmente en el campo de vista de la cámara. |

### 7.6.4 Reconocimiento de imágenes objetivo

Una vez abierta la escena, se desplegará la cámara del dispositivo para comenzar el reconocimiento. En éste momento se ejecuta el siguiente algoritmo para poder detectar una imagen objetivo y realizar una acción con la misma.

1. Se inicializa el servicio de *Cloud Reco*.
2. Se registra el objeto *CloudRecognition* en el Manejador de Eventos.
3. Vuforia busca patrones en el flujo captado por la cámara, comparándolo con los patrones existentes asociados a cada imagen en la base de datos.

    3.1. En caso de que ocurra algún error, se manejará conforme a lo descrito en la sección **7.6.5 Manejo de errores**.

4. Cuando se detecta un patrón existente:

    4.1. Se detiene el escaneo de la base de datos.
    4.2. Se elimina cualquier otro *target* que se haya detectado anteriormente.
    4.3. Se almacena la clave encontrada en los metadatos de la imagen reconocida.
    4.4. Se detiene la cámara del dispositivo.
    4.5. Se inicia la escena Información Edificio, enviándole la clave del edificio reconocido.

Para implementar este algoritmo, se utilizan los métodos provistos por la interface `ICloudRecoEventHandler`. En el script *CloudRecoHandler.cs* puede verse dicha implementación, en las líneas 29 - 88.

### 7.6.5 Manejo de errores

El servicio *Cloud Reco* provee una lista de errores conocidos que se identifican mediante una clave, como se puede ver en la Tabla 8.

Tabla 8. Errores que se pueden generar al usar *Cloud Reco*.

| Categoría | Clave | Significado |
|---|---|---|
| 0 | UPDATE_ERROR_AUTHORIZATION_FAILED | Licencias incorrectas. |
| 1 | UPDATE_ERROR_NO_NETWORK_CONNECTION | No hay conexión a internet. |
| 1 | UPDATE_ERROR_SERVICE_NOT_AVAILABLE | Servicio no disponible. |
| 2 | UPDATE_ERROR_BAD_FRAME_QUALITY | Baja calidad de fotograma. |
| 0 | UPDATE_ERROR_UPDATE_SDK | Versión antigua del SDK. |
| 0 | UPDATE_ERROR_TIMESTAMP_OUT_OF_RANGE | Hora del dispositivo desactualizada. |
| 1 | UPDATE_ERROR_REQUEST_TIMEOUT | Se agotó el tiempo de espera. |
| 0 | UPDATE_ERROR_PROJECT_SUSPENDED | Servicio suspendido. |

De ésta forma, los errores están clasificados en tres categorías, como se muestra en la Tabla 9.

Tabla 9. Categorías para los diferentes errores.

| Categoría | Significado |
|---|---|
| 0 | El error no puede ser corregido, por lo que al cerrar el mensaje la aplicación saldrá de Realidad Aumentada. |
| 1 | El error se debe a la falta de conexión a internet y se quitará automáticamente cuando se detecte conexión. |
| 2 | El error es momentáneo o no afecta la función de reconocimiento, por lo que puede cerrarse el mensaje y continuar la ejecución. |

El comportamiento en la interfaz gráfica será distinto para cada uno, como se muestra en los ejemplos de la Figura 24. Nótese que, al cerrar *Error de autorización*, la aplicación cierra la escena de Realidad Aumentada y regresa al Menú Principal, mientras que para *Calidad baja* se cierra la ventana, pero el reconocimiento sigue funcionando.

*Calidad baja* surge cuando la cámara está desenfocada por un periodo de tiempo, sin que el usuario se dé cuenta. Aun cuando el error puede cerrarse, es necesario volver a enfocar la cámara. Por ésta razón, se implementó el mecanismo de autoenfoque, mediante un toque en la pantalla. Tal comportamiento se describe en el script *TapController.cs*.

Por otra parte, el segundo error se resuelve al detectar conexión a internet, a través de un *request* exitoso a la página www.google.com.


Figura 24. Ejemplos de cada categoría.

El manejo de errores puede consultarse en las líneas 90-213 del script *CloudRecoHandler.cs*.

## 7.7 Información Edificio

Conforme al esquema de navegación mostrado en la Figura 7, la escena Información Edificio sólo puede ser accedida a través de la escena Realidad Aumentada. Aquí se muestra la información del edificio reconocido previamente, la cual está disponible en tres idiomas: español, inglés e italiano.

La información se accede a través de los metadatos contenidos en la imagen reconocida por *Cloud Reco*, ya que en ella se encuentra la clave que identifica al edificio.

Una vez obtenida la clave, se lee el archivo que contiene toda la información relacionada al edificio, tal como se describió en la sección **7.3 Lectura de archivos.**

La información puede desplegarse en cualquiera de los idiomas disponibles, seleccionando el idioma deseado de la lista desplegable, tal como se muestra en la Figura 25.

Figura 25. Ejemplo de la información traducida para el Palacio de Bellas Artes.

Las funcionalidades para la carga de la información, así como la selección del idioma pueden consultarse en los scripts *MostrarInformacion.cs* y *SeleccionarIdioma.cs.*

## 7.8 Tutorial

La escena Tutorial muestra una serie de imágenes que explican el funcionamiento de la aplicación. En ellas se describe el funcionamiento de la Realidad Aumentada, la Ubicación y el acceso a la Información del edificio reconocido, tal como se muestra en la Figura 26.

Las imágenes se cargan desde un arreglo de *sprites*, que avanzan al hacer *swipe* en la pantalla del dispositivo. Para ello, las imágenes están contenidas en un objeto *scrollView,* quien se encarga de detectar los toques en la pantalla.


Figura 26. Tutorial para el uso de la aplicación.

El funcionamiento del tutorial está contenido en el script *SwipeImageProgress.cs*.

# 8. Resultados

Para comprobar el funcionamiento correcto de la aplicación, se realizaron las siguientes pruebas tomando como referencia la Basílica de Guadalupe.

### Pruebas de Ubicación

Por medio de *Google Maps*, se calculó la distancia entre la ubicación del dispositivo y la Basílica de Guadalupe, para compararlo con el cálculo realizado por la aplicación a través de la fórmula de Haversine, como se muestra en la Figura 27.

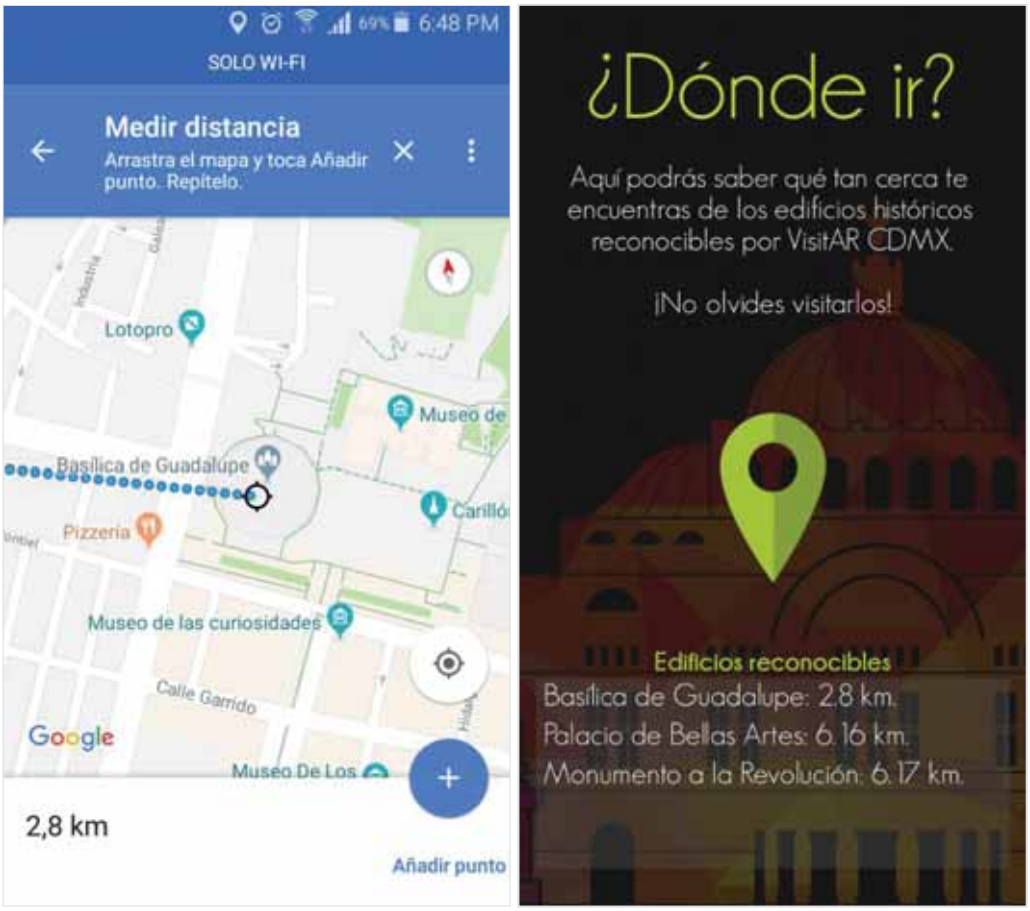


Figura 27. Comparativa entre la distancia calculada por Google Maps y VisitAR CDMX.

### Pruebas de Notificación

Conforme el usuario se acerca a la Basílica, la distancia calculada disminuye. Cuando el cálculo resultante es menor a un kilómetro, la aplicación lanza una notificación avisando que hay un edificio cercano, tal como se muestra en la Figura 28.


Figura 28. Notificación mostrada.

**Pruebas de Reconocimiento**

Una vez en el lugar, se probó el reconocimiento del edificio apuntando la cámara hacia el mismo, tal como se puede ver en la Figura 29.


Figura 29. Aplicación reconociendo la Basílica de Guadalupe.

Además, se realizaron pruebas para corroborar que los errores como la falta de internet o la calidad baja del fotograma son detectados correctamente, como se muestra en la Figura 30.


Figura 30. Errores detectados.

**Pruebas de Información Traducida**

Una vez reconocida la Basílica de Guadalupe, se muestra la información de la misma con la posibilidad de elegir entre los idiomas disponibles, como se muestra en la Figura 31.

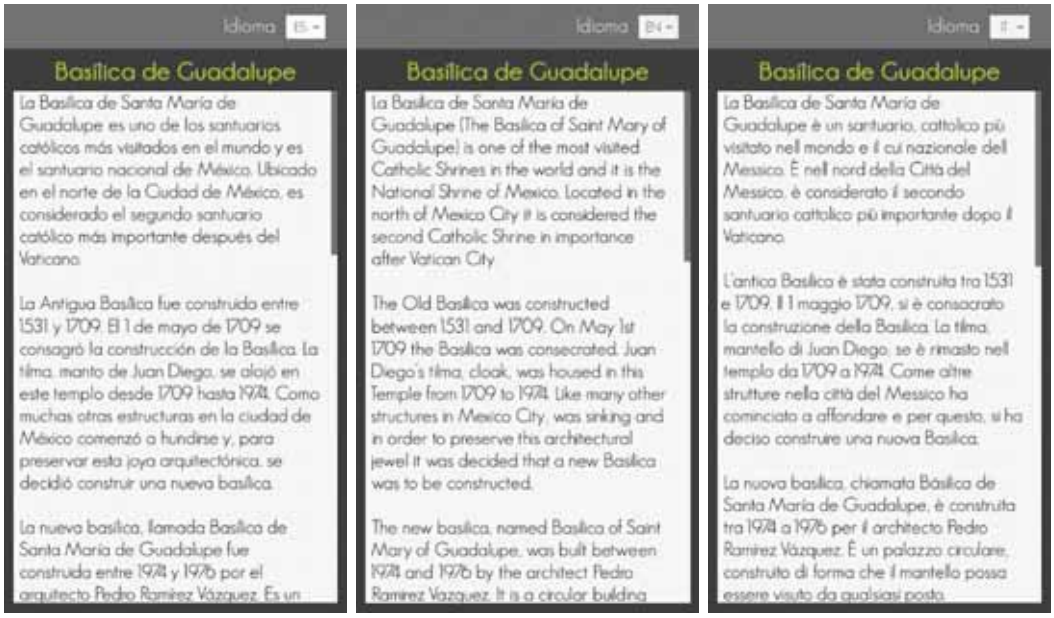


Figura 31. Información presentada en los tres idiomas.

# 9. Análisis de Resultados

Como parte del análisis, es importante mencionar lo siguiente:

El almacenamiento persistente para la aplicación se implementó con archivos, al no tener disponible un servidor de base de datos. Si bien es posible utilizar un servidor en la nube, implicaría costos no previstos en la planeación del proyecto.

Además, utilizando archivos de manera interna en la aplicación, el acceso se vuelve más rápido en comparación con la base de datos, ya que se ahorran los problemas que conlleva la disponibilidad de la red, así como las conexiones que sean necesarias.

En caso de que se requiera realizar una conexión a base de datos, es necesario utilizar el conector .NET para MySQL. Para ello, es necesario agregar y referenciar el archivo .dll en el proyecto, así como cambiar el .NET utilizado a la versión experimental 4.6 como se describió en la sección **7.1 Configuración del proyecto en Unity.**

Por otra parte, las notificaciones sólo son lanzadas cuando la aplicación está en primer plano. De otra forma, el servicio de GPS se detiene, lo que conlleva a que el cálculo de distancias no se actualice y, por lo tanto, no se notifica al usuario.

Las fotografías capturadas en días nublados funcionan mejor para realizar el reconocimiento. Esto sucede ya que, al no haber luz de sol, no se generan sombras sobre el edificio, por lo que Vuforia no las tomará en cuenta al momento de procesarlas para crear el patrón.

De ésta forma, al realizar el reconocimiento no importará si hay sombras sobre el edificio o no, ya que el patrón sólo contendrá las marcas que realmente corresponden al edificio.

Es necesario contemplar, además, que al ser edificios históricos podrían ser decorados en fechas conmemorativas o eventos especiales, lo cual obstruirá parcialmente el aspecto normal de los mismos.  En éste caso, puede que la aplicación no funcione como se espera.

# 10. Conclusiones

Es importante recalcar que la aplicación no realiza Realidad Aumentada, ya que no agrega elementos virtuales al entorno real. Sin embargo, esta tecnología resulta útil para realizar el reconocimiento de los edificios.

Esto nos muestra que el uso de la realidad aumentada basada en seguimiento puede servir como una herramienta capaz de escanear y reconocer elementos tridimensionales reales pertenecientes al entorno, lo cual expande el campo de aplicación de dicha tecnología.

Vuforia es una herramienta fácil de utilizar y cuenta con una amplia gama de funcionalidades para crear Realidad Aumentada, por lo que fue muy útil para el desarrollo del proyecto. Unity, en conjunto, proporciona todas las funcionalidades necesarias para el desarrollo de interfaces gráficas, aun cuando está orientado al desarrollo de videojuegos. A pesar de no haber desarrollado otras aplicaciones en lenguaje C# anteriormente, resultó sencillo realizar el proyecto al tener conocimientos previos sobre el paradigma de programación orientada a objetos.

VisitAR CDMX es una aplicación que puede fácilmente ser expandida a futuro, ya que las imágenes objetivo pueden cargarse de forma externa y, en ese caso, podría implementarse una base de datos en la nube que permita realizar las modificaciones necesarias a la información, sin necesidad de recompilar la aplicación.

Si bien no pudo desarrollarse el módulo de administración de la información tal como se había planteado, se utilizó otra alternativa que permitió el acceso a la información de manera más rápida.

VisitAR CDMX es capaz de reconocer y proporcionar información de al menos tres edificios, en tres idiomas disponibles. Además, puede detectar y notificar edificios históricos cercanos a la posición del usuario. De ésta forma, se concluye que los módulos planteados realizan las funcionalidades requeridas y, por tanto, el proyecto fue terminado con éxito.

# 11. Bibliografía

[1] U. A. Morales Enciso, "Geolocalización de puntos de interés en la zona arqueológica de Teotihuacán con Realidad Aumentada.", proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2013.

[2] Guiding Tech, "Understanding and Using Nokia City Lens on Nokia Lumia 920". https://www.guidingtech.com/20116/understanding-using-nokia-city-lens-lumia-920/

[3] Dicyt. "Visitas turísticas a Salamanca con realidad aumentada". http://www.dicyt.com/noticias/visitas-turisticas-a-salamanca-con-realidad-aumentada/

[4] Ronald T. Azuma, "A Survey of Augmented Reality", In Presence: Teleoperators and Virtual Environments 1997, pp. 355-385.

[5] Vuforia. "Vuforia is the leading AR platform. Here's why". https://www.vuforia.com/

[6] Vuforia Developer Library. "Vuforia Cloud Recognition Service". https://library.vuforia.com/content/vuforia-library/en/articles/Training/Cloud-Recognition-Guide.html

[7] Unity. "Unity 3D". https://unity3d.com/es

[8] Inside Mexico. "The Basilica of Our Lady of Guadalupe". https://www.inside-mexico.com/the-basilica-of-our-lady-of-guadalupe/

[9] Sacred Destinations. "Basilica of Our Lady of Guadalupe, Mexico City". http://www.sacred-destinations.com/mexico/mexico-city-basilica-guadalupe

[10] Monumento Revolución Mexicana. "The MRM". http://www.mrm.mx/eng/

[11] Mexconnect. "Mexico City's Revolution Monument". http://www.mexconnect.com/articles/3945-mexico-city-s-revolution-monument-monumento-a-la-revolucion

[12] World Monuments Fund. "Palace of Fine Arts". https://www.wmf.org/project/palace-fine-arts-palacio-de-bellas-artes

[13] Lonely planet. "Palacio de Bellas Artes". https://www.lonelyplanet.com/mexico/mexico-city/attractions/palacio-de-bellas-artes/a/poi-sig/1146839/361544

[14] Fórmula de Haversine. "¿Cómo calcular la distancia entre dos puntos geográficos?". https://www.genbetadev.com/cnet/como-calcular-la-distancia-entre-dos-puntos-geograficos-en-c-formula-de-haversine

[15] Notifications. "Simple Android Notifications Free - Asset Store Unity". https://assetstore.unity.com/packages/tools/integration/simple-android-notifications-free-68626

# 12. Apéndice A: Código fuente

Se anexan los scripts necesarios para el funcionamiento de la aplicación, en lenguaje C#. Los scripts se encuentran ordenados por el nombre de la escena que los implementa, o en Común si son compartidos por dos o más escenas.

## 12.1 Común

*KeyController.cs*

```
1.    using System.Collections;
2.    using System.Collections.Generic;
3.    using UnityEngine;
4.    using UnityEngine.SceneManagement;
5.
6.    public class KeyController : MonoBehaviour {
7.
8.        #region ElementosUI
9.        private Scene activeScene;
10.       public GameObject seccionLoadingScene;
11.       #endregion
12.
13.       #region MonoBehaviour
14.       //Define el apagado de la pantalla mediante el tiempo
          configurado en el dispositivo
15.       //Define la orientación de la interfaz, dependiendo de la
          escena activa
16.       void Start(){
17.
18.           Screen.sleepTimeout = SleepTimeout.SystemSetting;
19.
20.           activeScene = SceneManager.GetActiveScene ();
21.
22.           if (activeScene.buildIndex == 0 || activeScene.buildIndex
              == 1 || activeScene.buildIndex == 4) {
23.               Screen.orientation = ScreenOrientation.Portrait;
24.           } else {
25.               Screen.orientation = ScreenOrientation.AutoRotation;
26.           }
27.
28.           StartCoroutine (BotonAtras ());
29.       }
30.
31.       #endregion
32.
33.       #region Coroutines
34.
35.       //Define el comportamiento de la aplicación al tocar el botón
          <<Atrás>> del dispositivo
36.       //MenuPrincipal: La aplicación se cierra.
```

```
37.         //Otras escenas: Regresan a MenuPrincipal de manera asíncrona,
            mostrando una pantalla de carga
38.         IEnumerator BotonAtras(){
39.
40.             while (true) {
41.
42.                 if(Input.GetKeyDown (KeyCode.Escape)){
43.
44.                     switch (activeScene.buildIndex) {
45.
46.                     case 0:
47.                         Application.Quit ();
48.                         break;
49.
50.                     default:
51.                         SceneManager.LoadSceneAsync (0);
52.                         seccionLoadingScene.SetActive (true);
53.                         break;
54.                     }
55.                 }
56.
57.             yield return null;
58.             }
59.         }
60.         #endregion
61.     }
```

---

*CargarEscena.cs*

```
1.      using System.Collections;
2.      using UnityEngine;
3.      using UnityEngine.SceneManagement;
4.      using UnityEngine.UI;
5.
6.      public class CargarEscena : MonoBehaviour {
7.
8.          #region ElementosUI
9.          public GameObject seccionLoadingScreen;
10.         #endregion
11.
12.         #region BotonesCargarEscenas
13.
14.         //Carga la escena seleccionada de forma asíncrona, mostrando
            una pantalla de carga
15.         public void CargarEscenaSiguiente(int escena){
16.                 StartCoroutine (IniciarEscena (escena));
17.         }
18.
19.         IEnumerator IniciarEscena(int escena){
20.
21.             SceneManager.LoadSceneAsync (escena);
```

```
22.
23.             seccionLoadingScreen.SetActive (true);
24.
25.             yield return null;
26.         }
27.         #endregion
28.     }
```

---

*MoverImagenFondo.cs*

```
1.    using UnityEngine;
2.    using UnityEngine.UI;
3.
4.    public class MoverImagenFondo : MonoBehaviour {
5.
6.        #region ElementosUI
7.        public GameObject imagen;
8.        #endregion
9.
10.       #region Private
11.       private RectTransform rect;
12.       private float minOffset = 0;
13.       private float maxOffset = 0;
14.       #endregion
15.
16.       #region MonoBehaviour
17.       //Obtiene el tamaño de la imagen
18.       void Start(){
19.           rect = imagen.GetComponent<RectTransform> ();
20.           minOffset = rect.offsetMin.x;
21.           maxOffset = -rect.offsetMax.x;
22.       }
23.
24.
25.       void Update(){
26.
27.           //La imagen sigue dentro de los límites del canvas
28.           if (rect.offsetMin.x <= minOffset && rect.offsetMin.x >=
              maxOffset) {
29.               imagen.transform.Translate (Input.acceleration.x *
                  5.0f, 0f, 0f);
30.           }
31.               //La aceleración produce que la imagen salga de los
                  limites del canvas, para solucionarlo
32.               //se invierte el sentido del movimiento sobre el eje x,
                  dependiendo si se está en el límite
33.               //izquierdo o derecho del canvas.
34.           else {
35.               //Límite izquierdo
36.               if (rect.offsetMin.x > minOffset) {
```

```
                    imagen.transform.Translate (-0.02f, 0f, 0f);
                } else
                    //Límite derecho
                    if (rect.offsetMin.x < maxOffset) {
                    imagen.transform.Translate (0.02f, 0f, 0f);
                }
            }
        }

        #endregion
    }
```

---

*CargarArchivos.cs*

```
using System;
using System.IO;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class CargarArchivos
{
    #region CargarArchivos
    //Devuelve una lista de objectos Edificio, cada uno con el
    nombre y id de los edificios reconocibles
    public static List<Edificio> CargarEdificios(){
        List<Edificio> listaEdificio = new List<Edificio> ();

        String linea;
        Char[] delimitador = { '|' };

        try{
            StringReader reader = null;

            TextAsset data = (TextAsset)Resources.Load
            ("Edificio",typeof(TextAsset));

            reader = new StringReader (data.text);

            linea = reader.ReadLine ();

            while(linea != null){
                String[] subcadenas = linea.Split(delimitador);
                listaEdificio.Add(new
                Edificio(subcadenas[0],subcadenas[1]));

                linea = reader.ReadLine ();
            }
            reader.Close ();
        }
```

```
34.         catch(Exception e){
35.             Debug.Log("ManejoArchivos.CargarInformacionEdificios()
                "+e);
36.         }
37.         return listaEdificio;
38.     }
39.
40.     //Devuelve una lista de objetos InformacionEdificio, cada uno
        con la informacion traducida a cada idioma, de todos los
        edificios
41.     public static List<InformacionEdificio>
        CargarInformacionEdificio(){
42.
43.         List<InformacionEdificio> listaInformacion =
            newList<InformacionEdificio> ();
44.
45.         String linea;
46.         Char[] delimitador = { '|' };
47.
48.         try{
49.             StringReader reader = null;
50.
51.             TextAsset data = (TextAsset)Resources.Load
                ("InformacionEdificio",typeof(TextAsset));
52.
53.             reader = new StringReader (data.text);
54.
55.             linea = reader.ReadLine ();
56.
57.             while(linea!=null){
58.
59.                 String[] subcadenas = linea.Split(delimitador);
60.                 listaInformacion.Add(newInformacionEdificio(
                     subcadenas[1],subcadenas[2],subcadenas[3]));
61.
62.                 linea = reader.ReadLine();
63.             }
64.
65.             reader.Close();
66.
67.         }
68.         catch(Exception e){
69.             Debug.Log("ManejoArchivos.CargarInformacionEdificios()
                "+e);
70.         }
71.
72.         return listaInformacion;
73.     }
74.
75.     //Devuelve una lista con las ubicaciones de todos los edificios
76.     public static List<UbicacionEdificio>
        CargarUbicacionEdificios(){
77.
```

```
78.          List<UbicacionEdificio> listaUbicaciones = new
             List<UbicacionEdificio>();
79.
80.          String linea;
81.          Char[] delimitador = { '|' };
82.
83.          try{
84.              StringReader reader = null;
85.
86.              TextAsset data = (TextAsset)Resources.Load
                 ("UbicacionEdificio",typeof(TextAsset));
87.
88.              reader = new StringReader (data.text);
89.
90.              linea = reader.ReadLine ();
91.
92.
93.              while(linea!=null){
94.
95.                  String[] subcadenas = linea.Split(delimitador);
96.                  listaUbicaciones.Add(newUbicacionEdificio(
                     subcadenas[1],Double.Parse(subcadenas[2]),
                     Double.Parse(subcadenas[3])));
97.
98.                  linea = reader.ReadLine();
99.              }
100.
101.             reader.Close();
102.
103.         }
104.         catch(Exception e){
105.             Debug.Log("ManejoArchivos.CargarUbicacionEdificios()
                 "+e);
106.         }
107.
108.         return listaUbicaciones;
109.     }
110.     #endregion
111. }
```

---

*ConsultaArchivo.cs*

```
1.  using System;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.
5.  public class ConsultaArchivo
6.  {
7.      #region ConsultarInformacionCargada
8.      //Obtiene un objecto Edificio a partir de la id dada
```

```
9.      public static Edificio GetEdificio(String idEdificio){
10.
11.         Edificio edificio = null;
12.         List<Edificio> listaEdificio = CargarArchivos.CargarEdificios
();
13.
14.         for (int i = 0; i < listaEdificio.Count; i++) {
15.             if (listaEdificio [i].IdEdificio.CompareTo (idEdificio) ==
0) {
16.                 edificio = listaEdificio [i];
17.                 break;
18.             }
19.         }
20.
21.         return edificio;
22.     }
23.
24.     //Obtiene la lista de Información del edificio, traducida en los
        idiomas disponibles
25.     public static List<InformacionEdificio>
        GetInformacionEdificio(String idEdificio){
26.
27.         List<InformacionEdificio> listaInformacion
            =CargarArchivos.CargarInformacionEdificio ();
28.
29.         List<InformacionEdificio> listaInfoEdificio =
            newList<InformacionEdificio> ();
30.
31.         for (int i = 0; i < listaInformacion.Count; i++) {
32.
33.             if (listaInformacion [i].EdificioIdEdificio.CompareTo
                (idEdificio)== 0) {
34.                 listaInfoEdificio.Add (listaInformacion [i]);
35.             }
36.         }
37.
38.         return listaInfoEdificio;
39.     }
40.
41.     //Obtiene una lista con las Ubicaciones de todos los edificios
42.     public static List<UbicacionEdificio> GetUbicacionesEdificio(){
43.
44.         List<UbicacionEdificio> listaUbicaciones
            =CargarArchivos.CargarUbicacionEdificios ();
45.         return listaUbicaciones;
46.     }
47.
48.     #endregion
49. }
```

---

## 12.2 Modelo del Dominio

*DistanciaEdificio.cs*

```
1.  using System;
2.
3.  public class DistanciaEdificio
4.  {
5.      private String nombreEdificio;
6.      private double distancia;
7.
8.      public DistanciaEdificio (String nombreEdificio, double distancia)
9.      {
10.         this.nombreEdificio = nombreEdificio;
11.         this.distancia = distancia;
12.     }
13.
14.     public String NombreEdificio{
15.         get{
16.             return nombreEdificio;
17.         }
18.     }
19.
20.     public double Distancia{
21.         get{
22.             return distancia;
23.         }
24.     }
25.
26.     public override String ToString(){
27.         return nombreEdificio+": "+distancia+" km.\n";
28.     }
29. }
```

---

*Edificio.cs*

```
1.  using System;
2.
3.  public class Edificio
4.  {
5.      private String idEdificio;
6.      private String nombreEdificio;
7.
8.      public Edificio(){
9.
10.     }
11.
12.     public Edificio (String idEdificio, String nombreEdificio)
13.     {
14.         this.idEdificio = idEdificio;
```

```
15.         this.nombreEdificio = nombreEdificio;
16.     }
17.
18.     public String IdEdificio{
19.         get{
20.             return idEdificio;
21.             }
22.     }
23.
24.     public String NombreEdificio{
25.         get{
26.             return nombreEdificio;
27.         }
28.     }
29. }
```

---

*InformacionEdificio.cs*

```
1.  using System;
2.
3.  public class InformacionEdificio
4.  {
5.      private String edificioIdEdificio;
6.      private String idioma;
7.      private String informacion;
8.
9.      public InformacionEdificio (String edificioIdEdificio, String
        idioma,String informacion)
10.     {
11.         this.edificioIdEdificio = edificioIdEdificio;
12.         this.idioma = idioma;
13.         this.informacion = informacion;
14.     }
15.
16.     public String EdificioIdEdificio{
17.         get{
18.             return edificioIdEdificio;
19.         }
20.     }
21.
22.     public String Idioma{
23.         get{
24.             return idioma;
25.         }
26.     }
27.
28.     public String Informacion{
29.         get{
30.             return informacion;
31.         }
```

```
32.     }
33. }
```

---

*UbicacionEdificio.cs*

```
1.  using System;
2.
3.  public class UbicacionEdificio
4.  {
5.      private String idEdificio;
6.      private double latitud;
7.      private double longitud;
8.
9.      public UbicacionEdificio(String idEdificio, double latitud, double
        longitud){
10.         this.idEdificio = idEdificio;
11.         this.latitud = latitud;
12.         this.longitud = longitud;
13.     }
14.
15.     public String IdEdificio {
16.         get {
17.             return idEdificio;
18.         }
19.     }
20.
21.     public double Latitud{
22.         get{
23.             return latitud;
24.         }
25.     }
26.
27.     public double Longitud{
28.         get{
29.             return longitud;
30.         }
31.     }
32. }
```

---

## 12.3 Menú Principal

*CalcularDistancia.cs*

```
1.   using System.Collections.Generic;
2.   using UnityEngine;
3.   using UnityEngine.UI;
4.   using System;
```

```
5.  using System.Linq;
6.
7.  public class CalcularDistancia : MonoBehaviour
8.  {
9.      #region Static
10.     //Instancia un objeto estatico de la clase actual
11.     public static CalcularDistancia calcDistInstance;
12.     #endregion
13.
14.     #region Private
15.     private double latitudOrigen;
16.     private double longitudOrigen;
17.
18.     private List<UbicacionEdificio> listaUbicacionesEdificios;
19.     private List<Edificio> listaEdificios;
20.     public List<DistanciaEdificio> listaDistanciasEdificios;
21.     #endregion
22.
23.     #region MonoBehaviour
24.
25.     //Al iniciar la escena actual, instancia el objeto estático
        calcDistInstance y lo conserva durante
26.     //toda la ejecución. Al volver a la escena actual, elimina
        referencias duplicadas.
27.     void Awake(){
28.         DontDestroyOnLoad (this);
29.
30.         if (calcDistInstance == null) {
31.             calcDistInstance = this;
32.         } else {
33.             DestroyObject (gameObject);
34.         }
35.     }
36.
37.     //Inicializa las listas necesarias
38.     void Start(){
39.
40.         listaUbicacionesEdificios = new List<UbicacionEdificio> ();
41.         listaEdificios = new List<Edificio> ();
42.         listaDistanciasEdificios = new List<DistanciaEdificio> ();
43.
44.         ConsultarArchivo ();
45.
46.     }
47.
48.     void Update(){
49.
50.         //Si el servicio está activo, obtiene la ubicación del
            dispositivo y calcula la distancia con respecto a los
            edificios
51.         if (ServicioLocalizacion.locInstance.activo) {
52.
```

```
53.             latitudOrigen =
                ServicioLocalizacion.locInstance.latitud;
54.             longitudOrigen =
                ServicioLocalizacion.locInstance.longitud;
55.             MostrarInformacionLocalizacion ();
56.
57.         }
58.     }
59.
60.     #endregion
61.
62.     #region ArchivoUbicacionEdificio
63.     // Carga la lista de ubicaciones de todos los edificios.
64.     // Carga además, los nombres de cada edificio en el mismo orden
        de las ubicaciones.
65.     private void ConsultarArchivo(){
66.
67.         listaUbicacionesEdificios =
            ConsultaArchivo.GetUbicacionesEdificio ();
68.
69.         for (int i = 0; i < listaUbicacionesEdificios.Count; i++) {
70.
71.             String idEdificio = listaUbicacionesEdificios
                [i].IdEdificio;
72.             listaEdificios.Add
                (ConsultaArchivo.GetEdificio(idEdificio));
73.         }
74.     }
75.
76.     #endregion
77.
78.     #region MostrarDistancias
79.
80.     private void MostrarInformacionLocalizacion(){
81.
82.         Double distancia=0;
83.         listaDistanciasEdificios.Clear ();
84.
85.         //Calcula la distancia entre la ubicación del dispositivo y
            la ubicación del edificio
86.         for (int i = 0; i < listaUbicacionesEdificios.Count; i++) {
87.
88.             distancia = DistanciaEntreCoordenadas
                (latitudOrigen,longitudOrigen, listaUbicacionesEdificios
                [i].Latitud,listaUbicacionesEdificios [i].Longitud);
89.
90.             //Llena una lista auxiliar con el nombre del edificio y
                la distancia calculada
91.             listaDistanciasEdificios.Add(
                 newDistanciaEdificio(listaEdificios[i].
                 NombreEdificio,distancia));
92.         }
93.
```

```
94.          OrdenarEdificiosPorDistancias ();
95.
96.          LlenarTextDistancias ();
97.      }
98.
99.      //Ordena la lista auxiliar de Edificio-Distancia para mostrar
         las distancias de menor a mayor
100.     private void OrdenarEdificiosPorDistancias(){
101.         listaDistanciasEdificios = listaDistanciasEdificios.OrderBy
             (x =>x.Distancia).ToList();
102.     }
103.
104.     //Llena el texto txtDistancia en la Escena Ubicacion, con las
         distancias calculadas
105.     private void LlenarTextDistancias(){
106.
107.         String txtDistAux="";
108.
109.         for (int i = 0; i < listaDistanciasEdificios.Count; i++) {
110.             txtDistAux = txtDistAux + ""
                 +listaDistanciasEdificios[i].ToString();
111.         }
112.
113.         char[] c = new char[1];
114.         c[0] = '\n';
115.         txtDistAux = txtDistAux.TrimEnd (c);
116.
117.         MostrarDistancias.distancias = txtDistAux;
118.     }
119.     #endregion
120.
121.     #region CalculoDistancias
122.     //Realiza el calculo de la distancia entre la ubicación del
         edificio y la ubicación del dispositivo,
123.     //utilizando la formula de Haversine
124.     private double DistanciaEntreCoordenadas(double latOrigen,
         double lonOrigen, double latDestino, double lonDestino){
125.
126.         latOrigen = ConvertirRadianes(latOrigen);
127.         lonOrigen = ConvertirRadianes(lonOrigen);
128.         latDestino = ConvertirRadianes(latDestino);
129.         lonDestino = ConvertirRadianes(lonDestino);
130.
131.         const double r = 6367; //radio de la tierra en kilometros
132.
133.         var dlat = (latDestino - latOrigen) / 2;
134.         var dlon = (lonDestino - lonOrigen) / 2;
135.
136.         var q = Math.Pow (Math.Sin (dlat), 2) + Math.Cos (latOrigen)
             *Math.Cos (latDestino) * Math.Pow (Math.Sin (dlon), 2);
137.         var c = 2 * Math.Asin(Math.Sqrt(q));
138.         var d = r * c;
139.
```

```
140.         return Math.Round(d,2);
141.     }
142.
143.     //Convierte latitud o longitud en radianes
144.     private double ConvertirRadianes(double x){
145.         return (x * Math.PI) / 180;
146.     }
147.     #endregion
148.
149. }
```

---

*ServicioLocalizacion.cs*

```
1.    using System.Collections;
2.    using UnityEngine;
3.    using UnityEngine.UI;
4.    using System;
5.
6.    public class ServicioLocalizacion : MonoBehaviour {
7.
8.        #region Coordenadas
9.        public double latitud;
10.       public double longitud;
11.       #endregion
12.
13.       #region ServicioActivo
14.       public bool activo = false;
15.       #endregion
16.
17.       #region Static
18.       //Instancia un objeto estatico de la clase actual
19.       public static ServicioLocalizacion locInstance;
20.       #endregion
21.
22.       #region MonoBehaviour
23.       //Al iniciar la escena actual, instancia el objeto estático
          locDistance y lo conserva durante
24.       //toda la ejecución. Al volver a la escena actual, elimina
          referencias duplicadas.
25.       void Awake(){
26.           DontDestroyOnLoad (this);
27.
28.           if (locInstance == null) {
29.               locInstance = this;
30.           } else {
31.               DestroyObject (gameObject);
32.           }
33.       }
34.
35.       //Inicializa el servicio de localización
```

```
36.       void Start(){
37.           StartCoroutine (StartLocationService());
38.       }
39.       #endregion
40.
41.       #region Coroutine
42.       IEnumerator StartLocationService(){
43.
44.           while (true) {
45.
46.               //Si el GPS está deshabilitado en el dispositivo se
                  detiene el servicio de localización
47.               //Se desactiva la bandera
48.               if (!Input.location.isEnabledByUser) {
49.                   MostrarError ("Por favor, activa el GPS \ne
                      inténtalo de nuevo.");
50.                   Input.location.Stop ();
51.                   activo = false;
52.
53.               }
54.               //Si el GPS está habilitado
55.               else {
56.
57.                   //Si el servicio de localización no está activo
58.                   if (!activo) {
59.
60.                       //Inicializa el servicio de localización (10m
                          precisión, 10m para actualizar)
61.                       Input.location.Start (10f, 10f);
62.
63.                       //Espera 50s para inicializar el servicio
64.                       int count = 50;
65.
66.                     while (Input.location.status ==
                        LocationServiceStatus.Initializing && count > 0)
                        {
67.                           yield return new WaitForSeconds (1);
68.                           count--;
69.                       }
70.
71.                       //Si el servicio tarda más de 50s en
                          inicializar, muestra un error
72.                       if (count <= 0) {
73.                           MostrarError ("Se agotó el tiempo de
                              activación del servicio de Localización.");
74.                       }
75.                       //Si inicializa en el tiempo esperado se activa
                          la bandera
76.                       else{
77.                           activo = true;
78.                       }
79.
```

```
80.                         //Si el usuario no permite el acceso a la
                            ubicación, muestra un error
81.                         if (Input.location.status ==
                            LocationServiceStatus.Failed) {
82.                             MostrarError ("Se denegó el acceso al
                                Servicio de Localización.");
83.                         }
84.                     }
85.                     //Si el servicio de localización se encuentra
                        activo, obtiene las coordenadas actuales
86.                     else {
87.                         latitud =
                            (double)Input.location.lastData.latitude;
88.                         longitud =
                            (double)Input.location.lastData.longitude;
89.                     }
90.                 }
91.             yield return null;
92.             }
93.         }
94.
95.         #endregion
96.
97.         #region Error
98.         //Muestra el tipo de error generado en la Escena Ubicacion
99.         private void MostrarError (String mensajeError){
100.            LocationErrorHandler.mensajeError = mensajeError;
101.        }
102.        #endregion
103.    }
```

---

*Notificaciones.cs*

```
1.  using UnityEngine;
2.  using UnityEngine.UI;
3.  using System;
4.  using Assets.SimpleAndroidNotifications;
5.
6.  public class Notificaciones : MonoBehaviour {
7.
8.      #region Private
9.      private String ultimoEdificio = "";
10.     private String nuevoEdificio = "";
11.     #endregion
12.
13.     #region Static
14.     //Instancia un objeto estatico de la clase actual
15.     public static Notificaciones notificacionesInstance;
16.     #endregion
17.
```

```
18.     #region MonoBehaviour
19.     //Al iniciar la escena actual, instancia el objeto estático
        locDistance y lo conserva durante
20.     //toda la ejecución. Al volver a la escena actual, elimina
        referencias duplicadas.
21.     void Awake(){
22.         DontDestroyOnLoad (this);
23.
24.         if (notificacionesInstance == null) {
25.             notificacionesInstance = this;
26.         } else {
27.             DestroyObject (gameObject);
28.         }
29.     }
30.
31.     //Si el servicio está activo, calcula si algún edificio se
        encuentra en un radio de 1 km.
32.     void Update () {
33.         if (ServicioLocalizacion.locInstance.activo) {
34.             masCercano ();
35.         }
36.     }
37.     #endregion
38.
39.     #region EdificioCercano
40.     //Notifica cuando un edificio se encuentre dentro de un radio de
        1km.
41.     private void masCercano(){
42.
43.         for (int i = 0; i <
            CalcularDistancia.calcDistInstance.listaDistanciasEdificios.Co
            unt; i++) {
44.
45.             if
                (CalcularDistancia.calcDistInstance.listaDistanciasEdifici
                os [i].Distancia < 1.0) {
46.                 nuevoEdificio =
                    CalcularDistancia.calcDistInstance.listaDistanciasEdif
                    icios [i].NombreEdificio;
47.                     break;
48.                 }
49.             }
50.
51.         //Almacena cual fue el último edificio cercano a la posición
            del usuario
52.         //Cuando detecta un nuevo edificio, lanza una notificación con
            el nombre del edificio
53.         if (nuevoEdificio.CompareTo (ultimoEdificio) != 0) {
54.             ultimoEdificio = nuevoEdificio;
55.             NotificationManager.SendWithAppIcon (TimeSpan.FromSeconds
                (1), "¡Edificio Cercano!", "Estás cerca de
                "+ultimoEdificio, new Color (0.66f, 0.85f, 0.07f),
                NotificationIcon.Bell);
```

```
56.         }
57.     }
58.     #endregion
59. }
```

---

## 12.4 Ubicación

*LocationErrorHandler.cs*

```
1.    using System;
2.    using UnityEngine;
3.    using UnityEngine.UI;
4.
5.    public class LocationErrorHandler : MonoBehaviour {
6.
7.        #region ElementosUI
8.        public GameObject seccionError;
9.        public Text txtMensajeError;
10.       #endregion
11.
12.       #region static
13.       public static String mensajeError = "";
14.       #endregion
15.
16.       #region MonoBehaviour
17.
18.       void Update () {
19.
20.           //Si el servicio de localización está activo, no se muestra
              ningún error
21.           if (ServicioLocalizacion.locInstance.activo ||
              mensajeError.Equals("")) {
22.               txtMensajeError.text = "";
23.               mensajeError = "";
24.               seccionError.SetActive (false);
25.           }
26.           //En caso contrario, se obtiene el error desde
              ServicioLocalización y se muestra en pantalla
27.           else {
28.                   txtMensajeError.text = mensajeError;
29.                   seccionError.SetActive (true);
30.           }
31.       }
32.       #endregion
33.   }
```

---

*MostrarDistancias.cs*

```
1.  using UnityEngine;
2.  using UnityEngine.UI;
3.  using System;
4.
5.  public class MostrarDistancias : MonoBehaviour {
6.
7.      #region ElementosUI
8.      public Text txtDistancias;
9.      #endregion
10.
11.     #region Static
12.     public static String distancias;
13.     #endregion
14.
15.     #region MonoBehaviour
16.     void Update () {
17.
18.         //Si el servicio de localización está activo, muestra las
            distancias calculadas en CalcularDistancia
19.         if (ServicioLocalizacion.locInstance.activo) {
20.             txtDistancias.text = distancias;
21.         }
22.         //En caso contrario, muestra un mensaje de error
23.         else {
24.             txtDistancias.text = "No hay información disponible.";
25.         }
26.     }
27.     #endregion
28. }
```

---

## 12.5 Realidad Aumentada

*TapController.cs*

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.  using Vuforia;
5.
6.  public class TapController : MonoBehaviour {
7.
8.      #region UIElements
9.      public GameObject SeccionFocusCamera;
10.     #endregion
11.
12.     #region MonoBehaviour
13.
14.     //Define la cámara con autoenfoque
```

```
15.     void Start () {
16.         CameraDevice.Instance.SetFocusMode(
            CameraDevice.FocusMode.FOCUS_MODE_CONTINUOUSAUTO);
17.     }
18.
19.     //Se espera un toque en pantalla para autoenfocar la cámara
20.     //El toque solo funciona si la sección FocusCamera está activa
21.     void Update () {
22.         if (SeccionFocusCamera.activeInHierarchy) {
23.             Tap ();
24.         }
25.     }
26.     #endregion
27.
28.     #region Tap
29.     //Si se detecta un toque, se enfoca la cámara
30.     private void Tap(){
31.
32.         if (Input.GetMouseButtonUp (0)) {
33.             CameraDevice.Instance.SetFocusMode(
                 CameraDevice.FocusMode.FOCUS_MODE_TRIGGERAUTO);
34.             StartCoroutine (RestoreOriginalFocusMode ());
35.         }
36.     }
37.     #endregion
38.
39.     #region Coroutine
40.     //Despues de un segundo, se devuelve al modo autoenfoque
41.     private IEnumerator RestoreOriginalFocusMode()
42.     {
43.         yield return new WaitForSeconds(1.0f);
44.         CameraDevice.Instance.SetFocusMode(
            CameraDevice.FocusMode.FOCUS_MODE_CONTINUOUSAUTO);
45.     }
46.     #endregion
47. }
```

---

*CloudRecoHandler.cs*

```
1.  using UnityEngine;
2.  using Vuforia;
3.  using UnityEngine.SceneManagement;
4.  using UnityEngine.UI;
5.  using System.Collections;
6.
7.
8.  public class CloudRecoHandler : MonoBehaviour,
    ICloudRecoEventHandler
9.  {
10.     #region Private
```

```
11.     private CloudRecoBehaviour cloudReco;
12.     private ObjectTracker imageTracker;
13.     private int cerrar;
14.     private bool hayError;
15.     #endregion
16.
17.     #region ElementosUI
18.     public ImageTargetBehaviour imageTarget;
19.     public GameObject SeccionFocusCamera;
20.     public GameObject SeccionError;
21.     public GameObject SeccionLoading;
22.     public Text txtTipoError;
23.     public Text txtMensajeError;
24.     public Button btnCerrar;
25.     #endregion
26.
27.     #region MonoBehaviour
28.
29.     void Start(){
30.         hayError = false;
31.         cloudReco = GetComponent<CloudRecoBehaviour> ();
32.
33.         //Registro en el Manejador de Eventos
34.         if (cloudReco) {
35.             cloudReco.RegisterEventHandler (this);
36.         }
37.     }
38.
39.     #endregion
40.
41.     #region CloudReco
42.
43.     public void OnInitialized(){
44.
45.         //Obtiene ObjectTracker necesario para el acceso a
            TargetFinder
46.         imageTracker =
            TrackerManager.Instance.GetTracker<ObjectTracker> ();
47.     }
48.
49.     public void OnStateChanged(bool scanning){
50.
51.         //¿Vuforia está escaneando la base de datos?
52.         if (scanning) {
53.             //Elimina todas las imagenes detectadas anteriormente
54.             imageTracker.TargetFinder.ClearTrackables (false);
55.         }
56.     }
57.
58.     public void OnNewSearchResult(TargetFinder.TargetSearchResult
        targetSearchResult){
59.
60.         //Se detecta un nuevo target
```

```
61.
62.         //Detiene TargetFinder, no se escanea la base de datos
63.         cloudReco.CloudRecoEnabled = false;
64.
65.         //Elimina todas las imagenes detectadas anteriormente
66.         imageTracker.TargetFinder.ClearTrackables (false);
67.
68.         //Detiene la cámara de Vuforia cuando cambia de escena
69.         CameraDevice.Instance.Stop();
70.
71.         //almacena el id del Edificio reconocido y cambia de escena
72.         MostrarInformacion.idEdificio = targetSearchResult.MetaData;
73.         StartCoroutine( CargarEscenaSiguiente(3));
74.     }
75.
76.     #endregion
77.
78.     #region CargarEscenaInformacion
79.     IEnumerator CargarEscenaSiguiente(int escena){
80.
81.         SceneManager.LoadSceneAsync (escena);
82.
83.         SeccionLoading.SetActive (true);
84.
85.         yield return null;
86.     }
87.
88.     #endregion
89.
90.     #region Error
91.
92.     public void OnInitError(TargetFinder.InitState initError){
93.
94.         //Error al iniciar Cloud Reco
95.         switch (initError) {
96.
97.             case
                TargetFinder.InitState.INIT_ERROR_NO_NETWORK_CONNECTION:
98.                 MostrarError (1,"Conexión no disponible", "Por favor
                    revisa tu conexión a internet e inténtalo de
                    nuevo.");
99.                 break;
100.
101.            case
                TargetFinder.InitState.INIT_ERROR_SERVICE_NOT_AVAILABLE:
102.                MostrarError (1,"Servicio no disponible", "El
                    servicio Cloud Recognition no está disponible,
                    inténtalo más tarde.");
103.                break;
104.        }
105.    }
106.
107.    public void OnUpdateError(TargetFinder.UpdateState updateError){
```

```
108.
109.         //Error durante la ejecución de Cloud Reco
110.
111.         switch (updateError) {
112.
113.         case
             TargetFinder.UpdateState.UPDATE_ERROR_AUTHORIZATION_FAILED:
114.             MostrarError(0,"Error de autorización", "Las llaves
                 necesarias para el servicio han expirado o son
                 incorrectas.");
115.             break;
116.         case
            TargetFinder.UpdateState.UPDATE_ERROR_NO_NETWORK_CONNECTION:
117.             MostrarError(1,"Conexión no disponible", "Por favor
                 revisa tu conexión a internet e inténtalo de nuevo.");
118.             break;
119.         case
             TargetFinder.UpdateState.UPDATE_ERROR_PROJECT_SUSPENDED:
120.             MostrarError(0,"Error de autorización", "El servicio
                 Cloud Recognition ha sido suspendido.");
121.             break;
122.         case TargetFinder.UpdateState.UPDATE_ERROR_REQUEST_TIMEOUT:
123.             MostrarError (1,"Se agotó el tiempo", "Se agotó el
                 tiempo de espera, por favor revisa tu conexión a
                 internet.");
124.             break;
125.         case
             TargetFinder.UpdateState.UPDATE_ERROR_SERVICE_NOT_AVAILABLE:
126.             MostrarError(1,"Servicio no disponible", "El servicio
                 Cloud Recognition no está disponible, inténtalo más
                 tarde.");
127.             break;
128.         case
            TargetFinder.UpdateState.UPDATE_ERROR_TIMESTAMP_OUT_OF_RANGE:
129.             MostrarError(0,"Sincronización de reloj", "Por favor
                 actualiza la fecha y hora de tu dispositivo.");
130.             break;
131.         case TargetFinder.UpdateState.UPDATE_ERROR_UPDATE_SDK:
132.             MostrarError(0,"Versión antigua", "La aplicacion está
                 usando una Version antigua de Vuforia.");
133.             break;
134.         case
             TargetFinder.UpdateState.UPDATE_ERROR_BAD_FRAME_QUALITY:
135.             MostrarError(2,"Calidad baja", "Se ha detectado baja
                 calidad de fotograma.");
136.             break;
137.         }
138.     }
139.
140.     public void MostrarError(int errorTipo, string tipo, string
         mensaje){
141.
142.         //Si no hay error activo, entra
```

```
143.         //Si hay error actualmente, bloquea la entrada de uno nuevo
144.         if (!hayError) {
145.
146.             //Bloquea la entrada de otros errores mientras está
                 activo
147.             hayError = true;
148.
149.             SeccionFocusCamera.SetActive (false);
150.             txtTipoError.text = tipo;
151.             txtMensajeError.text = mensaje;
152.             SeccionError.SetActive (true);
153.             btnCerrar.interactable = true;
154.
155.             //Si el error es por internet
156.             if (errorTipo == 1) {
157.                     btnCerrar.interactable = false;
158.                     StartCoroutine (DetectarConexion ());
159.             }
160.             //Si es de otro tipo
161.             else{
162.                 cerrar = errorTipo;
163.             }
164.         }
165.     }
166.
167.     IEnumerator DetectarConexion(){
168.
169.         //Test de conexión a internet
170.         //Hace un request a google.com
171.         //Si la descarga es exitosa, significará que hay conexión a
             internet y quitará el error en automatico
172.         //Si hay error durante la descarga, el error se mantendrá
             hasta que haya conexión
173.         WWW www;
174.
175.         while (true) {
176.             www = new WWW ("http://www.google.com");
177.
178.             yield return www;
179.
180.             if (!string.IsNullOrEmpty (www.error)) {
181.                 yield return new WaitForSeconds (5);
182.             } else {
183.                 break;
184.             }
185.         }
186.
187.         //cuando detecta conexión, cierra la ventana de error y
             desbloquea la entrada
188.         SeccionError.SetActive (false);
189.         SeccionFocusCamera.SetActive(true);
190.         hayError = false;
191.     }
```

```
192.
193.     public void CerrarUIError(){
194.
195.         //Si el error no puede ser corregido, al tocar el botón
             cerrar se devuelve a la pantalla principal
196.         //No será posible utilizar AR
197.         if (cerrar == 0) {
198.             StartCoroutine (CargarEscenaSiguiente (0));
199.         }
200.         //Si el error no causa problemas, puede ser cerrado, se
             desbloquea la entrada
201.         else {
202.             SeccionError.SetActive (false);
203.             hayError = false;
204.             StartCoroutine (InitSeccionFocusCamera());
205.         }
206.     }
207.
208.     //Espera .2 segundos para no interferir con el autoenfoque (tap)
209.     IEnumerator InitSeccionFocusCamera(){
210.         yield return new WaitForSeconds (0.2f);
211.         SeccionFocusCamera.SetActive (true);
212.     }
213.     #endregion
214. }
```

---

## 12.6 Información Edificio

*MostrarInformacion.cs*

```
1.  using System;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.  using UnityEngine.UI;
5.
6.  public class MostrarInformacion : MonoBehaviour {
7.
8.      #region Constantes
9.      const String idiomaDefault = "ES";
10.     #endregion
11.
12.     #region ElementosUI
13.     public Text txtNombreEdificio;
14.     public Text txtInformacionEdificio;
15.     public Scrollbar scrollVertical;
16.     #endregion
17.
18.     #region Static
19.     //El idEdificio se llena en CloudRecoHandler, antes de cargar la
        escena Información
```

```
20.     public static String idEdificio = "";
21.     #endregion
22.
23.     #region ArchivoInformacionEdificio
24.     private List<InformacionEdificio> listaInformacion;
25.     private Edificio edificio;
26.     #endregion
27.
28.
29.     #region MonoBehaviour
30.     void Start () {
31.         //Se carga el nombre del edificio y la información en los
            idiomas disponibles (desde archivo)
32.         edificio = ConsultaArchivo.GetEdificio (idEdificio);
33.         listaInformacion = ConsultaArchivo.GetInformacionEdificio
            (idEdificio);
34.
35.         DesplegarInformacion ();
36.     }
37.     #endregion
38.
39.     #region Private
40.     //Despliega el nombre y la informacion (ES) del edificio en la GUI
41.     private void DesplegarInformacion(){
42.
43.         txtNombreEdificio.text = edificio.NombreEdificio;
44.         txtInformacionEdificio.text = CargarInfoIdiomas
            (idiomaDefault);
45.     }
46.
47.     //Selecciona la información del edificio en función del idioma
        seleccionado
48.     private String CargarInfoIdiomas(String idioma){
49.
50.         for (int i = 0; i < listaInformacion.Count; i++) {
51.
52.             if (listaInformacion [i].Idioma.CompareTo (idioma) == 0) {
53.
54.             String info = listaInformacion [i].Informacion;
55.
56.             //Agrega saltos de linea en el texto para mostrarlo
                correctamente en GUI.
57.             info = info.Replace ("<salto>", "\n");
58.
59.             return info;
60.             }
61.         }
62.         return "Idioma no disponible";
63.     }
64.     #endregion
65.
66.     #region DropDownListIdioma
```

```
67.     //Cambia la información desplegada en la GUI con base en el nuevo
        idioma seleccionado
68.     public void CambiarIdioma(int index){
69.         txtInformacionEdificio.text
            =CargarInfoIdiomas(SeleccionarIdioma.idiomas[index]);
70.         scrollVertical.value = 1;
71.     }
72.     #endregion
73. }
```

---

*SeleccionarIdioma.cs*

```
1.  using System.Collections.Generic;
2.  using UnityEngine;
3.  using UnityEngine.UI;
4.
5.  public class SeleccionarIdioma : MonoBehaviour {
6.
7.      #region Static
8.          public static List<string> idiomas = new List<string>() {"ES",
            "EN","IT"};
9.      #endregion
10.
11.     #region ElementosUI
12.         public Dropdown ddownListaIdiomas;
13.     #endregion
14.
15.     #region MonoBehaviour
16.         //Llena la lista con los idiomas ES, EN, IT
17.         void Start () {
18.             ddownListaIdiomas.AddOptions (idiomas);
19.         }
20.     #endregion
21. }
```

---

## 12.7 Tutorial

*SwipeImageProgress.cs*

```
1.  using UnityEngine;
2.  using UnityEngine.UI;
3.
4.  public class SwipeImageProgress : MonoBehaviour {
5.
6.      #region ElementosUI
7.      public Image imgProgreso;
8.      public Image imgTutorial;
```

```
9.      public Button btnSalir;
10.     public Sprite [] listaImgProgreso = new Sprite[5];
11.     public Sprite[] listaImgTutorial = new Sprite[6];
12.     #endregion
13.
14.     #region MonoBehaviour
15.     void Update () {
16.
17.         //Obtiene el valor actual [0,1] del scroll horizontal
18.         var script = this.GetComponent<Scrollbar> ();
19.         float valueScroll = script.value;
20.         btnSalir.interactable = false;
21.
22.         //Se divide el valor del scroll entre 6 imágenes del tutorial
23.         //se muestra la imagen actual y el progreso al inferior de la
            pantalla
24.         //en la última imagen muestra un botón para salir del tutorial
25.         if (valueScroll >= 0.0 && valueScroll < 0.2) {
26.             imgProgreso.sprite = listaImgProgreso [0];
27.             imgTutorial.sprite = listaImgTutorial [0];
28.         } else {
29.             if (valueScroll >= 0.2 && valueScroll < 0.4) {
30.                 imgProgreso.sprite = listaImgProgreso [1];
31.                 imgTutorial.sprite = listaImgTutorial [1];
32.             } else {
33.                 if (valueScroll >= 0.4 && valueScroll < 0.6) {
34.                     imgProgreso.sprite = listaImgProgreso [2];
35.                     imgTutorial.sprite = listaImgTutorial [2];
36.                 } else {
37.                     if (valueScroll >= 0.6 && valueScroll < 0.8) {
38.                         imgProgreso.sprite = listaImgProgreso [3];
39.                         imgTutorial.sprite = listaImgTutorial [3];
40.                     } else {
41.                         if (valueScroll >= 0.8 && valueScroll < 1.0) {
42.                             imgProgreso.sprite = listaImgProgreso [4];
43.                             imgTutorial.sprite = listaImgTutorial [4];
44.                         } else {
45.                             if (valueScroll <= 1.0) {
46.                                 btnSalir.interactable = true;
47.                                 imgTutorial.sprite = listaImgTutorial
                                    [5];
48.                             }
49.                         }
50.                     }
51.                 }
52.             }
53.         }
54.     }
55.     #endregion
56. }
```