Desarrollo de un Sistema de Navegación y Guía por Voz Sintética para una Aplicación de recorridos virtuales con realidad aumentada en el Centro de Innovación y Tecnología (CIT) de la Universidad del Valle de Guatemala

Diego Alberto Leiva Pérez

UVG | UNIVERSIDAD DEL VALLE DE GUATMEALA

UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

Desarrollo de un sistema de navegación y guía por voz sintética para una aplicación de recorridos virtuales con realidad aumentada en el Centro de Innovación y Tecnología (CIT) de la Universidad del Valle de Guatemala

Trabajo de Graduación en modalidad de Trabajo Profesional presentado por
Diego Alberto Leiva Pérez
Para optar al grado académico de Licenciado en
Ingeniería en Ciencia de la Computación y Tecnologías de la Información

Guatemala,
2025

Vo.Bo.

Firma: _____

Ing. Dulce María Chacón Muñoz

Firma: _____

Ing. Carlos Victor Hugo Alvarado Conde

Fecha de aprobación: Guatemala,18 de noviembre de 2025

Agradecimientos

El presente trabajo representa la culminación de un proceso académico y personal que no habría sido posible sin el apoyo de quienes me acompañaron a lo largo de este camino.

A mis padres, cuyo esfuerzo, confianza y dedicación hicieron posible mi formación universitaria y me brindaron la fortaleza necesaria para avanzar en cada etapa. A mi hermana, no solo por su colaboración en el diseño y por facilitar los recursos técnicos que hicieron posible este proyecto, sino también por su presencia constante y su apoyo incondicional, que fueron fundamentales a lo largo de este proceso.

Agradezco igualmente a mis amigos y compañeros de proyecto, Gustavo González, Marta Ramírez y Eduardo Ramírez, por su trabajo, compromiso y colaboración durante el desarrollo del proyecto. Extiendo además mi gratitud a Pablo Orellana, quien, aun sin formar parte del equipo, mostró siempre disposición para colaborar y poner a disposición sus propios medios de trabajo.

Agradezco al Ing. Carlos Alvarado, por su diligencia, compromiso y orientación durante el proceso de revisión, así como a la Ing. Dulce Chacón, por su acompañamiento, motivación y respaldo en las distintas fases del proyecto.

A todos ellos, mi más profundo agradecimiento por su confianza, generosidad y contribución al logro de este proyecto.

Índice

Lista de Figuras

Lista de Cuadros

Resumen

El presente trabajo desarrolla un sistema de navegación y guía por voz integrado en una aplicación móvil de tour virtual con realidad aumentada para el Centro de Innovación y Tecnología (CIT) de la Universidad del Valle de Guatemala. El problema abordado es la limitación logística y de personal para ofrecer recorridos personalizados a gran escala durante actividades institucionales, lo que demanda una solución tecnológica autónoma, escalable y coherente con la estrategia de transformación digital de la universidad.

La aplicación se implementó en Unity y se organizó en tres subsistemas principales: un sistema de tours y navegación basado en mallas de navegación (NavMesh) y en la integración de coordenadas de posicionamiento interior mediante sensores UWB; un sistema de interfaz de usuario construido con UI Toolkit bajo un enfoque arquitectónico tipo MVVM y data-driven para gestionar pantallas, estados del recorrido y contenido contextual; y un sistema de audio y voz que emplea modelos de síntesis de texto a voz Gemini 2.5 Pro TTS de Google, configurado mediante Prompts en lenguaje natural para generar narraciones más expresivas.

Como resultado se obtuvo una aplicación funcional que permite a los visitantes recorrer de forma autónoma las instalaciones del CIT, visualizando rutas en tiempo real, recibiendo información contextual de las áreas de interés y siendo asistidos por mensajes de voz sincronizados con su posición. Las pruebas internas demostraron la correcta integración entre navegación, interfaz y audio, aunque la validación completa con trilateración física se vio limitada por restricciones del SDK UWB utilizado, lo que motiva recomendaciones sobre infraestructura, tecnologías de posicionamiento y pruebas de usuario futuras.

Abstract

This work presents a navigation and voice-guidance system integrated into a mobile virtual tour application with augmented reality for the Innovation and Technology Center (CIT) at Universidad del Valle de Guatemala. The problem addressed is the logistical and staffing limitations for offering personalized guided tours at scale during institutional activities, which creates the need for an autonomous, scalable technological solution aligned with the university's digital transformation strategy.

The application was implemented in Unity and organized into three main subsystems: a tour and navigation system based on navigation meshes (NavMesh) and the integration of indoor positioning coordinates obtained from UWB sensors; a user interface system built with UI Toolkit, following a data-driven and MVVM-inspired architectural approach to manage screens, tour states, and contextual content; and an audio and voice system that uses Google's Gemini 2.5 Pro TTS models, configured through natural language prompts to generate more expressive narrations.

As a result, a functional application was obtained that allows visitors to autonomously explore the CIT facilities, visualizing routes in real time, receiving contextual information about points of interest, and being assisted by voice messages synchronized with their position. Internal tests demonstrated the correct integration of navigation, interface, and audio, although full validation with physical trilateration was limited by constraints of the UWB SDK used. These findings motivate recommendations regarding development infrastructure, positioning technologies, and future structured user testing.

1. Introducción

En los últimos años, la Universidad del Valle de Guatemala (UVG) experimentó un crecimiento sostenido en su comunidad estudiantil y en sus actividades de vinculación institucional, como ferias, visitas guiadas y eventos de bienvenida. Estas actividades requirieron una planificación logística considerable y la participación constante de personal especializado, lo que limitó la capacidad de ofrecer recorridos personalizados a un número amplio de visitantes. Ante esta situación, se identificó la necesidad de desarrollar una herramienta tecnológica que permitiera automatizar y escalar la experiencia de los recorridos por el campus, manteniendo la calidad informativa y la interacción personalizada.

En respuesta a esta necesidad, se desarrolló una aplicación móvil de tour virtual con realidad aumentada para el Centro de Innovación y Tecnología (CIT) de la UVG. Este sistema permitió que los visitantes recorrieran las instalaciones de forma autónoma, guiados por una combinación de navegación interior, interfaz visual e instrucciones de voz generadas dinámicamente. La aplicación integró tres componentes principales: un sistema de navegación basado en algoritmos de path-finding sobre mallas de navegación (NavMesh) en Unity, una interfaz modular desarrollada con UI Toolkit integrada mediante una arquitectura inspirada en el patrón MVVM, y un módulo de voz sintética que generó locuciones contextuales mediante servicios de text-to-speech.

Si bien en el planteamiento inicial del proyecto, reflejado en los objetivos, se contempló el uso de SSML (Speech Synthesis Markup Language) como tecnología base para la síntesis de voz, durante la ejecución se presentaron cambios en la plataforma de Google Cloud TTS y se dispuso de modelos más recientes basados en inteligencia artificial, como Gemini 2.5 Pro TTS. Este escenario motivó una decisión metodológica de migrar desde SSML hacia un enfoque de prompting en lenguaje natural, que permitió producir narraciones más naturales y reducir el esfuerzo de edición, manteniendo el propósito original de contar con una guía de voz sintética.

Durante el desarrollo, el proyecto amplió su alcance inicial. De una responsabilidad centrada únicamente en la guía por voz, evolucionó hacia la integración completa de los distintos módulos en Unity, incluyendo la gestión de niveles, áreas de interés, recorrido general y comunicación entre los sistemas nativos y el entorno de Unity. Este proceso permitió consolidar una arquitectura por capas que organizó el funcionamiento del sistema en torno a los tours, niveles y áreas, con separación clara de responsabilidades entre posicionamiento, navegación, interfaz y audio.

El documento presenta los fundamentos conceptuales, técnicos y metodológicos del proyecto, así como los resultados obtenidos durante la integración del sistema de navegación, posicionamiento UWB, guía por voz sintética y la interfaz modular de usuario. Se describen las decisiones de diseño, la arquitectura desarrollada, las pruebas realizadas, las limitaciones técnicas identificadas y las estrategias adoptadas para garantizar el funcionamiento coherente del sistema dentro de su entorno operativo.

2. Objetivos

2.1. General

Desarrollar un sistema de navegación y guía por voz para dirigir a los usuarios a lo largo de un recorrido predeterminado dentro de la aplicación de tour virtual con realidad aumentada, y proporcionarles información sobre los puntos de interés visitados, mediante el uso de algoritmos de path-finding en Unity y síntesis de voz con lenguaje SSML, en el Centro de Innovación y Tecnología (CIT) de la Universidad del Valle de Guatemala.

2.2. Específicos

1. Diseñar un sistema de navegación para calcular rutas entre puntos de interés y guiar al usuario de manera precisa durante un recorrido por las instalaciones del CIT, utilizando tecnologías de path-finding dentro de Unity.

2. Integrar la localización en tiempo real con el sistema de navegación para reflejar la ubicación física del usuario en su representación dentro del entorno virtual, mediante el uso de coordenadas proporcionadas por el módulo de trilateración basado en sensores UWB.

3. Desarrollar un sistema de generación de mensajes de voz sintética para producir audios con información relevante sobre los puntos de interés durante el recorrido, a través de tecnologías de síntesis de voz con soporte para lenguaje SSML.

4. Implementar la reproducción sincronizada de los mensajes de voz en función de la posición del usuario en el entorno virtual, para comunicar información contextual a lo largo del recorrido, mediante activadores espaciales que cargan los audios.

3. Antecedentes

El desarrollo de sistemas interactivos basados en realidad aumentada (AR) ha evolucionado significativamente en el ámbito académico y tecnológico, impulsando nuevas formas de navegación, visualización y comunicación dentro de entornos físicos. En este contexto, los recorridos virtuales guiados se han convertido en una herramienta de apoyo útil para la orientación de usuarios en espacios complejos, al combinar entornos tridimensionales con información contextual que replican el entorno real.

En los últimos años, la Universidad del Valle de Guatemala (UVG) ha promovido proyectos interdisciplinarios dentro del Centro de Innovación y Tecnología (CIT) orientada a la aplicación de tecnologías modernas en educación, innovación y divulgación científica. Uno de los resultados más relevantes de esta iniciativa fue el proyecto *Desarrollo de una aplicación para recorridos virtuales mediante el uso de realidad aumentada para visitas guiadas en el CIT de la UVG*, cuyo objetivo general consistió en crear una plataforma digital inmersiva que permitiera a los visitantes explorar las instalaciones universitarias mediante dispositivos móviles compatibles con AR.

El proyecto fue desarrollado por un equipo multidisciplinario de estudiantes, quienes trabajaron de manera coordinada para desarrollar las primeras iteraciones de la aplicación. De esta estructura surgieron dos productos principales, una aplicación desarrollada en Unity, basada en navegación con códigos QR, y una versión nativa para iOS que exploró el uso de sensores Ultra-Wideband (UWB) para posicionamiento en interiores. Ambas aplicaciones compartieron el mismo objetivo general, pero abordaron el problema desde diferentes aproximaciones técnicas.

El trabajo desarrollado por Mombiela (2024), titulado *Desarrollo de modelado 3D y algoritmo de path-finding en aplicación de recorridos virtuales con Unity*, abordó la problemática de la orientación dentro del CIT a partir de la creación de un entorno tridimensional de los pasillos del edificio. El autor utilizó Blender para el modelado de los niveles (con un error promedio en las mediciones del 4.87%) y Unity para la integración del entorno, incorporando un algoritmo de path-finding basado en NavMesh y pruebas del algoritmo A* para definir rutas de navegación.

Este módulo demostró la viabilidad técnica de un recorrido guiado en un entorno digital, aunque presentó limitaciones en la sincronización entre el espacio virtual y el entorno físico real. El propio autor identificó el problema de registro, es decir, la desalineación entre los objetos digitales y su correspondencia en el mundo real, como una de las principales restricciones para alcanzar una experiencia aumentada más precisa.

Sobre esta infraestructura, se desarrolló un módulo UI, elaborado por Hernández (2024). Su trabajo, *Desarrollo de UX/UI en Aplicación de Recorridos Virtuales con Unity,* tuvo como objetivo diseñar una experiencia gráfica coherente, adaptable e intuitiva, adaptable a distintos dispositivos móviles. El autor implementó la interfaz utilizando Unity UI Toolkit y UI Builder, estructurando pantallas jerárquicas para la navegación, la selección de rutas y la visualización de puntos de interés.

También integró recursos multimedia, con el fin de mejorar la presentación visual del recorrido. Su aporte metodológico radica en el enfoque centrado en el usuario, validado mediante pruebas de usabilidad que confirmaron la claridad e intuición de las pantallas.

La segunda aplicación surgió de la necesidad de sustituir los códigos QR por un sistema de localización automatizado que permitiera reconocer de una manera más exacta la posición del usuario dentro del edificio sin intervención manual. En esta fase se llevó a cabo una investigación y validación de sensores UWB como tecnología base para localización en interiores.

Gonzalez (2024), en su trabajo *Evaluación y selección de tecnologías para sistemas IPS para el CIT*, realizó una comparación entre Bluetooth Low Energy (BLE) y UWB, evaluando precisión, estabilidad y tolerancia a interferencias. Sus resultados demostraron que UWB alcanzó una precisión promedio de error del 2.42 %, mientras que BLE presentó errores superiores al 30 %, evidenciando la superioridad técnica del primero. Este análisis permitió recomendar formalmente la adopción de UWB como tecnología de referencia para las siguientes fases del proyecto.

En base a esos resultados, Santos (2024) desarrolló la versión nativa para iOS, documentada en su trabajo *Desarrollo de interfaz gráfica para aplicación de recorridos virtuales en la plataforma iOS*. En ella, el autor implementó la detección por proximidad utilizando ARKit, RealityKit y sensores UWB, desarrollando un prototipo funcional que permitía reconocer la cercanía de los usuarios a puntos de interés sin requerir códigos QR. A diferencia del enfoque anterior en Unity, esta versión no empleó mapas tridimensionales, sino un sistema de beacons de proximidad, que representó el primer acercamiento práctico al posicionamiento automatizado dentro del CIT.

En conjunto, estos trabajos constituyeron los cimientos conceptuales y experimentales para aplicación de recorridos guiados por el CIT. Los esfuerzos iniciales permitieron obtener una aplicación funcional basada en AR y una validación empírica del uso de UWB como alternativa tecnológica para la localización interior. No obstante, ambos productos se mantuvieron en una fase de prototipo, sin lograr la integración completa entre el modelo tridimensional, la interfaz de usuario y el sistema de posicionamiento.

4. Justificación

La Universidad del Valle de Guatemala impulsó de forma constante su proceso de transformación digital, integrando tecnologías emergentes en sus procesos académicos, administrativos y de comunicación institucional. En este contexto, las visitas guiadas presenciales realizadas durante eventos como Experiencia UVG o recorridos institucionales presentaron limitaciones logísticas, de personal y de cobertura. Estas visitas fueron esenciales para dar a conocer las instalaciones, los programas académicos y la filosofía institucional, pero implicaron una inversión significativa de tiempo y recursos humanos, lo que dificultó atender de manera simultánea a grandes grupos de visitantes.

El desarrollo de una aplicación de tour virtual con navegación interior y asistencia por voz respondió directamente a esta necesidad institucional. Esta solución permitió ampliar el alcance de las visitas guiadas sin depender de la disponibilidad constante de guías humanos, ofreciendo una experiencia autónoma, flexible e inclusiva. Los visitantes pudieron recorrer las instalaciones del CIT a su propio ritmo, con asistencia visual, auditiva y contextual en tiempo real. Además de modernizar la experiencia de visita, el sistema redujo los requerimientos operativos y mejoró la eficiencia en la gestión de los recorridos.

Desde el punto de vista técnico, el proyecto representó una integración avanzada de distintas tecnologías dentro de un mismo entorno. Se utilizó Unity como motor principal, algoritmos de path-finding sobre NavMesh para la navegación interior, comunicación con sensores UWB mediante plugins nativos en iOS y generación de voz dinámica a través de servicios en la nube. Este enfoque permitió aplicar principios de arquitectura de software, diseño orientado a datos y patrones de desarrollo que favorecieron la escalabilidad y modularidad del sistema.

En el plano académico, el proyecto fortaleció competencias en desarrollo de software, integración de hardware, comunicación entre entornos nativos, interacción humano computadora y diseño de experiencias inmersivas. Al mismo tiempo, aportó un valor tangible a la universidad al ofrecer una base replicable para futuros recorridos o aplicaciones institucionales. El trabajo demostró cómo un proyecto de desarrollo de software puede trascender su función técnica para convertirse en una solución estratégica de comunicación y experiencia, alineada con los objetivos de innovación y sostenibilidad tecnológica de la institución.

5. Marco Teórico

5.1. Arquitectura general de sistemas interactivos en Unity

En el desarrollo con Unity, la arquitectura de los sistemas se basa en principios de modularidad y composición, lo cual permite la construcción de sistema interactivos complejos apegándose a principios sólidos de ingeniería de software. A continuación, se describen los conceptos fundamentales para el desarrollo con Unity.

5.1.1. Paradigma basado en componentes

Unity adopta un patrón de Entidad-Componente para la construcción de comportamientos. En lugar de utilizar una jerarquía de herencia profunda, Unity favorece la composición, en donde cada entidad del juego es un `GameObject`. Por sí solo, un `GameObject` no hace nada; prácticamente es un contenedor al que se le adjuntan componentes como scripts, físicas, colisionadores, etc. Cada componente implementa una porción específica del comportamiento del objeto. De esa manera se sustituye la herencia de clases clásica por la composición de comportamientos. En lugar de definir una clase por cada tipo de objeto en un árbol de herencia rígido, como se tiende a hacer en la programación orientada a objetos se construyen objetos combinando componentes reutilizables. (Unity Technologies, 2025m)

Este paradigma resuelve problemas típicos de la herencia múltiple y promueve la flexibilidad. Al segmentar funcionalidades en componentes independientes y reutilizables, se evita la explosión combinatoria de subclases para cada caso particular de uso. Ya que se facilita la capacidad de cambiar o extender comportamientos simplemente agregando o quitando componentes, reduciendo la complejidad y acoplamiento. Unity garantiza que cada `GameObject` tenga siempre un componente de posición y orientación obligatorio, el resto de los componentes son opcionales según la funcionalidad deseada. (Song, 2024)

5.1.2. Ciclo de vida en scripts de Unity

Unity define un ciclo de vida de los scripts basado en callbacks que el motor invoca en un orden predefinido. Esta secuencia incluye etapas básicas como `Awake()`, `Start()`, `Update()` y `OnDestroy()`, donde cada una tiene un propósito distinto dentro del ciclo de ejecución. La arquitectura resultante es reactiva: en lugar de que el desarrollador llame manualmente a estas funciones, Unity las ejecuta automáticamente en momentos clave, y el código del desarrollador reacciona a dichos eventos. (Unity Technologies, 2025h)

`Awake()` se invoca una vez cuando el objeto y su script se inicializa, antes de cualquier otro método de inicio. Es llamado al cargar la escena o al instanciar un prefab, incluso si el `GameObject` está inactivo el callback se difiere hasta que el objeto se active.

Este se usa típicamente para inicializar variables internas o establecer referencias entre componentes, ya que Unity garantiza que todos los `Awake()` de la escena ocurren antes de ejecutar ningún `Start()`. (Unity Technologies, 2025q)

`Start()` se llama una vez justo antes de la primera actualización de fotograma en que el script esté habilitado. Es decir, tras completarse todos los `Awake()`, cada script activo ejecuta su `Start()` en el primer fotograma. Este callback se utiliza para lógica de inicialización que puede depender de que otros objetos ya hayan ejecutado `Awake()`. Unity asegura que, para los objetos presentes al iniciar la escena, todos los `Start()` ocurren antes del primer `Update()`. (Unity Technologies, 2025s)

`Update()` por su parte, es el método central de bucle de juego, invocado en cada fotograma para los scripts habilitados. Aquí se implementa la mayor parte de la lógica dinámica: lectura de entrada, movimientos, comprobaciones constantes, etc. `Update()` corre tan frecuentemente como la velocidad de fotogramas. Unity ofrece variantes como `FixedUpdate()` para física, a intervalos fijos de tiempo y `LateUpdate()` ejecutado tras todos los Updates, útil para seguir cámaras u ordenar dependencias adicionales, pero el patrón común es usar `Update()` para la mayoría de las actualizaciones por cuadro. (Unity Technologies, 2025t)

`OnDestroy()` se invoca cuando el objeto va a ser destruido o al descargar la escena. Es el último paso del ciclo de vida de ese objeto, útil para liberar recursos, cancelar suscripciones a eventos u otras tareas de limpieza. Por ejemplo, si un objeto se había registrado como observador en algún sistema global, en `OnDestroy()` debería darse de baja para evitar referencias colgantes. Unity garantiza que `OnDestroy()` se ejecuta después del último fotograma en que existió el objeto y antes de quitarlo de memoria. (Unity Technologies, 2025r)

Este ciclo de vida implica una inversión de control, donde el flujo principal de la aplicación viene dado por el motor de Unity y los scripts desarrollados reaccionan a dichos eventos. Esta arquitectura reactiva desacopla al programador del bucle principal del juego. En lugar de escribir un bucle principal, se tiene la certeza que Unity llamará a los callbacks en el momento apropiado. La ventaja es que cada componente se centra en su propia lógica cuando es notificado, facilitando el diseño orientado a componentes y eventos. De este modo, la secuencia de inicialización y actualización está estructurada por Unity, contribuyendo a la robustez y previsibilidad del comportamiento del sistema.

5.1.3. Programación orientada a eventos en C#

La Programación Orientada a Eventos (POA) es un mecanismo de diseño esencial para el desacoplamiento de sistemas. Se basa en el patrón Observador, donde un objeto emisor notifica a una lista de objetos dependientes sobre un cambio de estado, sin necesidad de conocer la identidad de los receptores. Unity y C# facilitan esto mediante un modelo dirigido por eventos empleando delegados y eventos. Un delegado en C# es esencialmente una referencia tipada a uno o varios métodos que comparten firma. Los delegados permiten invocar esos métodos de forma indirecta, habilitando el patrón Observador nativamente en el lenguaje. (Beyza, 2023; French, 2021)

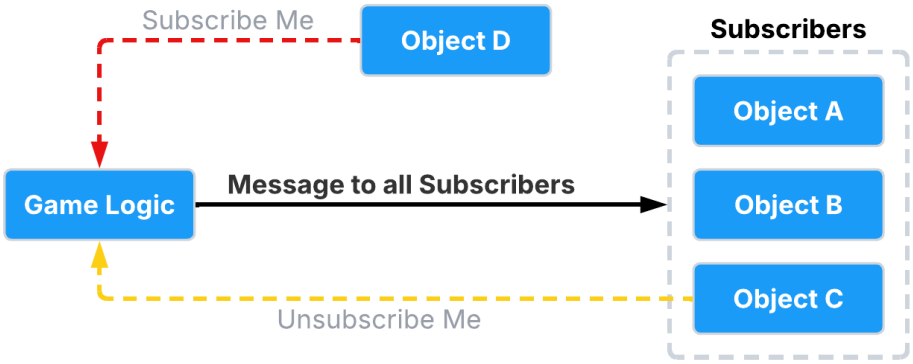

Figura 5.1: Diagrama del Patrón Observador
Fuente: Adaptado de Serrano (2016)

Este mecanismo permite implementar un flujo emitir-escuchar, el cual se ejemplifica en la Figura 5.1. El objeto emisor lanza un evento cuando ocurre algo, y múltiples receptores que se hayan suscrito reaccionarán ejecutando sus propios métodos, sin que el emisor necesite conocerlos explícitamente. Así se logra un fuerte desacoplamiento entre quien genera un suceso y quien lo recibe.

La principal ventaja de la POA es el desacoplamiento estricto del código. Al usar eventos y delegados, la clase que emite el evento no necesita saber quiénes están suscritos ni qué harán con esa información. Esto reduce dependencias directas y por consiguiente en tiempo de ejecución, los observadores pueden añadirse o removerse dinámicamente, otorgando flexibilidad. (Lafritz, 2022)

5.1.4. Uso de ScriptableObjects como contenedores de datos

Unity proporciona la clase especial `ScriptableObject` para definir objetos de datos que no representan entidades físicas en la escena, sino colecciones de información o configuración. Un `ScriptableObject` es un tipo serializable que puede existir como asset independiente en el proyecto almacenado como un archivo `.asset`. Estos objetos no necesitan estar ligados a un `GameObject` en la escena. Son esencialmente, contenedores de datos configurables que permiten separar los datos de la lógica. De esta manera una o más instancias pueden referencia ese mismo asset para consultar sus atributos. (Nordon, 2024)

En cuanto a persistencia, al ser assets, los `ScriptableObjects` existen fuera de las escenas. Pueden ser referenciados desde cualquier escena y, de hecho, facilitan compartir estado entre escenas sin depender de objetos Singleton. Además, dado que son objetos independientes, es posible utilizar varios sets de datos intercambiables. Otro uso importante es en arquitectura data-driven. En lugar de codificar comportamientos rígidos, se parametriza la lógica mediante datos. Los `ScriptableObjects` permiten diseñar activos de datos fáciles de manipular por diseñadores sin tocar código. Esto mejora la modularidad y reduce la necesidad de recompilar código para ajustar contenido. (Unity Technologies, 2023)

5.1.5. Enfoque data-driven frente a codificación rígida

El diseño impulsado por datos (data-driven) es un enfoque de ingeniería que contrasta con la codificación rígida (hard-coding). En un enfoque data-driven el comportamiento del software está gobernado por datos externos en lugar de lógica codificada. El código provee mecanismos genéricos para procesar esos datos, pero los detalles de comportamiento viven en archivos de datos que pueden modificarse independientemente del código fuente. En contraste, en un diseño rígido, los comportamientos están embebidos en el código, haciendo costosa cualquier modificación. (García, 2024)

Las ventajas en mantenimiento y escalabilidad son notables. Al no necesitar recompile, los diseñadores pueden ajustar equilibrio del juego, localización de textos, u otras variables fácilmente. El riesgo de errores se reduce porque el código manipula estructuras de datos en vez de tener valores mágicos dispersos. Además, escalar el proyecto se vuelve más cuestión de añadir datos que de duplicar lógica. En Unity, este enfoque se ve reforzado por herramientas como los mencionados `ScriptableObjects`, o incluso el uso del inspector para introducir valores en lugar de fijarlos en script. Esto hace al juego más fácil de actualizar y personalizar, y posibilita crear herramientas editor para modificar esos datos. (Zhang, 2025)

5.1.6. Separación de responsabilidades en sistemas interactivos

Un principio arquitectónico fundamental para sistemas interactivos (y software en general) es la separación de responsabilidades en distintas capas o componentes. Esto suele materializarse en patrones como MVC (Modelo-Vista-Controlador) y MVVM (Modelo-Vista-ViewModel), provenientes del desarrollo de interfaces de usuario. Aplicar estos modelos en proyectos interactivos ayuda a gestionar la complejidad y evitar dependencias circulares entre partes del código.

El patrón MVC es uno de los más antiguos y adoptados para interfaces. Su objetivo principal es dividir la aplicación en tres componentes interconectados, pero con responsabilidades claras:

- **Modelo:** Contiene los datos y la lógica de negocio o estado de la aplicación. Representa el dominio del problema. no conoce detalles de interfaz; proporciona métodos para manipular los datos y notifica cambios.

- **Vista:** Es la interfaz de usuario, encargada de presentar información al usuario y de recibir la interacción. Debe mostrar los datos del Modelo de forma adecuada. Idealmente, la Vista no contiene lógica de negocios; sólo lógica de presentación.

- **Controlador:** Actúa como intermediario entre la Vista y el Modelo. Procesa las entradas del usuario desde la Vista, las traduce en operaciones sobre el Modelo. Luego toma las actualizaciones del Modelo y decide qué Vista o qué parte de la Vista actualizar. En las implementaciones clásicas, el Controlador contiene la lógica de control de flujo e interacción.

El valor de MVC es que esta separación reduce dependencias: la Vista no debería tomar decisiones de negocios, solo refleja el estado del Modelo. Este es independiente de cualquier UI concreta y el Controlador orquesta, pero puede ser sustituido o modificado sin alterar la lógica interna del Modelo o las mecánicas de dibujo de la Vista. (Codecademy, s.f.)

Por otra parte, el MVVM es una variación más moderna, muy usada en entornos de aplicación, que introduce el concepto de ViewModel. Los componentes que componen este patrón se pueden observar en la Figura 5.2. El ViewModel cumple un rol similar al Controlador, pero con énfasis en estado y comunicación reactiva. El ViewModel es una abstracción del estado de la UI mantiene los datos provenientes del Modelo formateados de forma apta para la Vista, y expone propiedades y comandos a los cuales la Vista puede enlazarse. La Vista, no interactúa directamente con el Modelo, sino con el ViewModel; la Vista se enlaza a propiedades del ViewModel y muestra sus valores, y también puede invocar comandos del ViewModel. Mientras que el Modelo sigue siendo la fuente de los datos. (Microsoft, 2024)

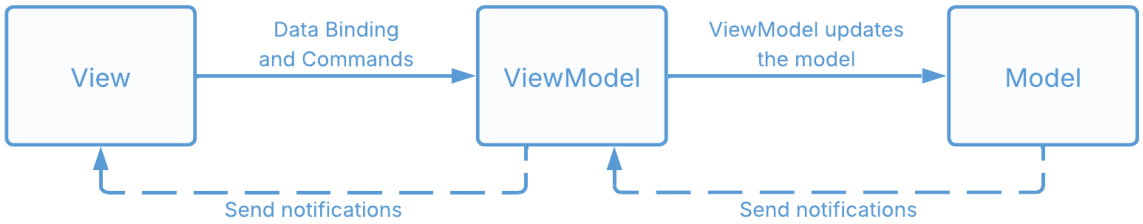

Figura 5.2: Diagrama de relaciones de los componentes MVVM
Fuente: Adaptado de Microsoft (2024)

La diferencia clave con MVC es que en MVVM la conexión entre Vista y ViewModel suele ser declarativa y automática. Cuando una propiedad del ViewModel cambia, la Vista lo refleja inmediatamente, sin código imperativo que pase datos como lo es el caso del MVC. Tanto MVC como MVVM buscan separar capas para lograr modularidad. Por ende, su integración trae beneficios directos. Se evita mezclar en un mismo componente código de lógica de juego con manipulación de interfaz. Esto hace el código más comprensible y permite cambiar la implementación de una capa sin reescribir las otras. (Kumar, 2024)

5.1.7. Patrón Mediator aplicado a sistemas desacoplados

El patrón Mediador centraliza la comunicación entre componentes UI y la lógica del juego. En lugar de que cada vista se comunique directamente con otras, todas reportan eventos a un único objeto mediador o coordinador. Este mediador recibe notificaciones y decide qué subsistema debe responder, evitando dependencias mutuas. Este patrón propone que se debe de cesar toda comunicación directa entre componentes y usar un mediador que redirija las interacciones. En la práctica, esto significa que un controlador central maneja los eventos de UI, consulta el modelo o ejecuta comandos, y así la UI permanece independiente de los detalles de la lógica de negocio. (Svetz, 2021)

5.2. Navegación en entornos virtuales con Unity

En aplicaciones de realidad aumentada o juegos 3D es común requerir que agentes autónomos o indicadores encuentren rutas óptimas en un entorno virtual. Unity proporciona un conjunto de herramientas para navegación basadas en conceptos tradicionales de path-finding (búsqueda de caminos) en grafos, adaptados al espacio tridimensional.

5.2.1. Fundamentos de path-finding en grafos

El problema del path-finding (búsqueda de caminos) consiste en hallar una ruta óptima entre dos puntos (nodos) en un grafo que representa el espacio de navegación. Óptima suele significar de menor costo. Dos algoritmos clásicos para este propósito son Dijkstra y A* (A estrella). Ambos son algoritmos de búsqueda en grafos, pero A* introduce heurística para mejorar la eficiencia. (Ionos, 2023)

El algoritmo A* (A estrella) es la solución más utilizada en sistemas de navegación, considerándose una extensión del Algoritmo de Dijkstra. A* es un algoritmo de búsqueda informada porque utiliza una función heurística para guiar la búsqueda de forma eficiente.

La función de costo $f(x)$ de A* es la base de su funcionamiento

$$f(x) = g(x) + h(x)$$

Donde:

- x es la distancia recorrida.
- $g(x)$ representa el real acumulado desde el nodo inicial hasta el nodo actual x.
- $h(x)$ representa la estimación heurística del costo restante desde el nodo actual x hasta el nodo objetivo.

Para que A* garantice la ruta óptima, la heurística $h(x)$ debe ser admisible, es decir, nunca debe sobreestimar el costo real restante. Una heurística común en entornos 3D es la distancia euclidiana. El algoritmo A* opera explorando el grafo, seleccionando continuamente el nodo en la lista abierta con el valor $f(x)$ más bajo, asegurando que los nodos más prometedores sean examinados primero hasta que se alcance el objetivo. (DataCamp, 2024)

5.2.2. Uso de NavMesh en navegación

Unity abstrae la complejidad de la navegación mediante el concepto de NavMesh. Una malla poligonal que marca las áreas del escenario por donde un agente puede caminar. En tiempo de diseño, Unity puede pre-calcular la NavMesh analizando la geometría estática. El resultado es una serie de polígonos convexos que recubren las superficies transitables.

Estos polígonos forman los nodos del grafo de navegación, conectados por aristas si comparten borde. Un agente de navegación considerará cada polígono como un estado del grafo y las conexiones como movimientos posibles entre regiones adyacentes.

La estructura interna de la NavMesh en Unity es tal que cada polígono tiene información de conectividad con sus vecinos. Para generarla generalmente se realiza desde el editor por medio de un bake, pero también hay componentes de `NavMeshSurface` que permiten generar en tiempo de ejecución si fuera necesario.

Al calcular un camino, Unity proyecta los puntos de inicio y fin sobre la NavMesh encontrando en qué polígono cae cada uno y luego realiza búsqueda A* de polígono a polígono. Esto produce una secuencia de polígonos que llevan del origen al destino. Después, coloca puntos de ruta dentro de esos polígonos, generalmente en los centros o en las intersecciones necesarias y aplica el algoritmo funnel para obtener puntos finales de la ruta. (Unity Technologies, 2025a)

Un punto importante son las Áreas de navegación y sus costos. Unity permite marcar regiones de la NavMesh con tipos de área, en la Figura 5.3 se muestran algunos tipos de área predeterminados como Caminable, Agua o NO Caminable. Cada tipo de área puede tener un costo de desplazamiento relativo. Por defecto, caminar en área Caminable tiene costo de 1.0 por cada unidad desplazada. Otras áreas como Agua pueden tener costos más elevados para indicar que cruzar por estas es más costoso que el área Caminable. Los desarrolladores pueden ajustar estos costos globalmente en la ventana de Navegación o incluso por agente vía código.

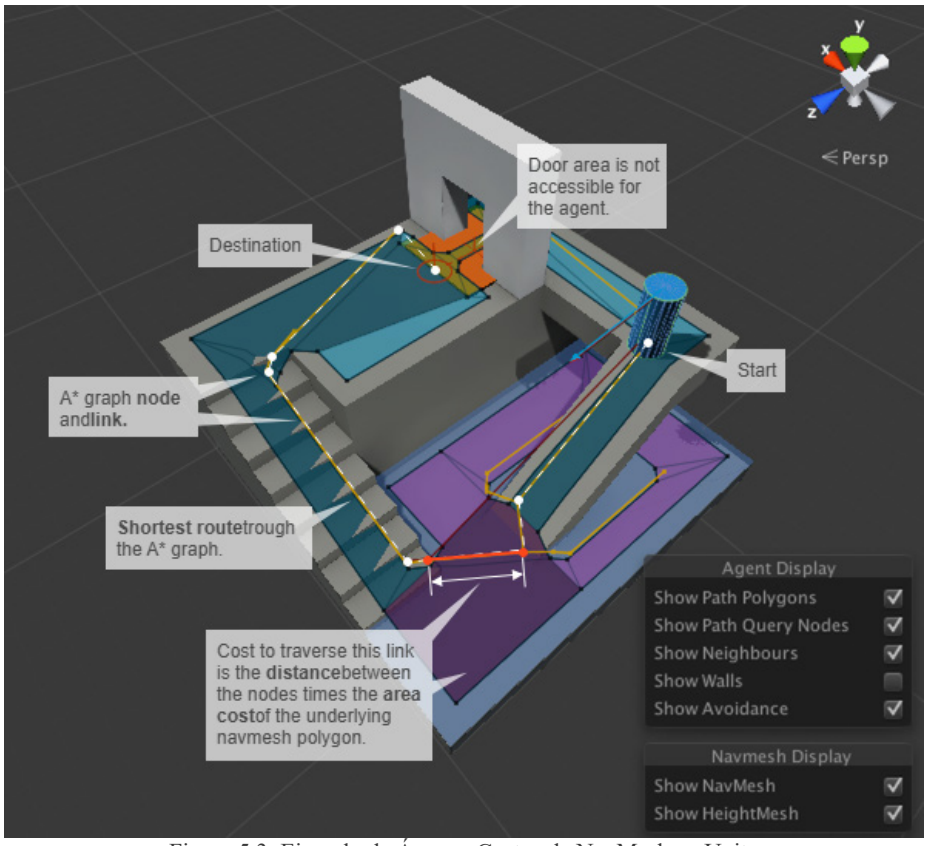

Figura 5.3: Ejemplo de Áreas y Costos de NavMesh en Unity
Fuente: Unity Technologies (2024)

El algoritmo de path-finding tomará esto en cuenta y preferirá rutas que minimicen el costo total, incluso si geométricamente no son las más cortas, evitando áreas costosas a menos que la alternativa sea mucho más larga.

Por ejemplo, si un camino recto atraviesa un área de alto costo y otra ruta da un rodeo por un área con costo normal, el path-finding podría optar por el rodeo si resulta menos costoso para el agente. Técnicamente, A* multiplica la distancia de cada tramo por el costo del área correspondiente antes de compararlo. (Unity Technologies, 2024)

5.2.3. Técnicas de corrección espacial de coordenadas

En entornos de realidad aumentada u otras aplicaciones con datos de sensores, es frecuente obtener posiciones que no caen exactamente sobre la `NavMesh` debido a imprecisiones o ruido. Unity proporciona la función `NavMesh.SamplePosition` para proyectar una posición arbitraria al punto más cercano de la NavMesh dentro de un cierto rango. Es decir, dada una coordenada en el mundo, esta función encuentra la posición válida más cercana en el terreno navegable.

Internamente, `NavMesh.SamplePosition` realiza una búsqueda alrededor de la posición dada, proyectando verticalmente hacia la NavMesh y expandiendo si es necesario. Retorna true y un `NavMeshHit` con la posición ajustada si encuentra algún polígono de la NavMesh cerca. Uno puede especificar un `maxDistance` (radio de búsqueda) y opcionalmente un mask de áreas permitidas para la muestra. La idea general de clamping es asegurar que las coordenadas usadas por los agentes sean válidas en el espacio navegable. (Unity Technologies, 2025o)

5.2.4. Visualización de trayectorias

Una técnica común para representar rutas de navegación dentro de Unity es utilizando el componente `LineRenderer`, el cual permite trazar líneas continuas en el espacio tridimensional a partir de una secuencia de puntos definidos. El componente tiene la capacidad de configurar y ajustar varias de sus propiedades como el ancho de línea a lo largo de su trayectoria, o aplicar gradientes de color para indicar progreso o intensidad. También se cuenta con modos de edición en la escena para facilitar la creación, subdivisión y ajuste manual de vértices, así como la opción de simplificar trayectorias automáticamente usando el algoritmo de Ramer-Douglas-Peucker. Trabajando en modo World Space, las líneas dibujadas se integran naturalmente en el entorno 3D, lo que es útil en entornos virtuales y en aplicaciones con AR para mantener el alineamiento entre elementos digitales y la geometría del mundo real.(Unity Technologies, 2025g)

5.3. Integración nativa Unity-iOS

Aunque Unity provee muchas funcionalidades multiplataforma, en ocasiones es necesario integrar características nativas de iOS. La interoperabilidad entre el código C# de Unity y el código nativo de iOS (Objective-C/Swift) se logra mediante plugins nativos y llamadas P/Invoke.

5.3.1. Puente entre Unity y bibliotecas nativas

La tecnología fundamental que permite la interoperabilidad entre el código C# gestionado de Unity y las librerías nativas es `P/Invoke` (Platform Invoke). Este mecanismo permite a las funciones en C# acceder a funciones, callbacks y estructuras definidas en código nativo (C, C++, Objective-C, o incluso Swift a través de un puente C).

Para iOS, existe una particularidad crítica; los plugins nativos se enlazan estáticamente en el ejecutable final. Por lo tanto, en lugar de especificar el nombre de una librería de carga dinámica, se utiliza `DllImport("__Internal")` en la declaración de funciones externas. Este último es un alias especial que indica a Mono/IL2CPP que la función a importar se encuentra en el binario principal de la aplicación, que en el caso de iOS es el ejecutable generado. En iOS no se cargan librerías dinámicas externas arbitrarias, así que Unity vincula los plugins nativos en tiempo de compilación de Xcode como código estático dentro del binario. Por ello se usa `"__Internal"` en lugar de un nombre de DLL. (Unity Technologies, 2025k, 2025f)

A continuación, un ejemplo del lado de C# de como definir una función externa:

```
1. [DllImport("__Internal")]
2. private static extern float FooPluginFunction(int valor);
3.
```

Del lado nativo (Objective-C, C o C++), se debe de tener una función C denominada con el mismo nombre que la compilada en la aplicación . Cuando en C# se llama la función externa, Unity localizará la función nativa correspondiente en el binario en tiempo de ejecución y la ejecutará, devolviendo el resultado.

5.4. Diseño de interfaces en Unity con UI Toolkit

El sistema UI Toolkit de Unity permite crear interfaces declarativas usando UXML (estructura, análogo a HTML) y USS (estilos, análogo a CSS). Los elementos básicos se representan con la clase `VisualElement`, que actúa como contenedor de otros elementos y agrupa estilos comunes. En UXML se define la jerarquía visual y se aplican estilos con USS, lo que separa la lógica de presentación del contenido. Este enfoque asocia cada `VisualElement` a un nivel en la jerarquía, estableciendo layout y apariencia de forma similar a un DOM web, pero optimizado para runtime en Unity. (Joffrineau, 2022)

5.4.1. Manipulación estructurada de elementos visuales

UI Toolkit organiza la UI como un árbol jerárquico de `VisualElement`. Para acceder o modificar elementos en tiempo de ejecución se usan consultas de `UQuery`. El método estático `Q<T>()` es un atajo que devuelve el primer elemento de tipo T que cumpla ciertos criterios ya sea por nombre, clase USS o tipo. También existe `Query<T>()` para obtener listas de elementos coincidentes. Se recomienda crear consultas al inicio del ciclo de vida de Unity y reutilizarlas para evitar gastos repetidos. La jerarquía visual se recorre con `UQuery` y `Q<T>()`, filtrando por atributos o clases CSS-like para manipular dinámicamente la UI. (Lin et al., 2025; Unity Technologies, 2025i)

5.4.2. Manejo de eventos y callbacks en UI Toolkit

UI Toolkit dispone de un sistema de eventos similar al del DOM donde los eventos pueden propagar de manera ascendente hacia los padres (bubble up) y propagar de manera descendente hacia los hijos (trickle-down). Cada `VisualElement` puede registrar callbacks con `RegisterCallback<TEvento>()`.

La clase `ClickEvent` se dispara al hacer clic sobre un elemento, y por defecto este evento se propaga tanto hacia abajo como hacia arriba en la jerarquía. Es decir, Unity permite que un elemento pueda estar escuchando a eventos específicos con callbacks, y el sistema se encarga de enrutar el evento correctamente en la jerarquía visual. (Unity Technologies, 2025b)

5.4.3. Comparación entre UI Toolkit y uGUI

UI Toolkit introduce mejoras de rendimiento, escalabilidad y consistencia frente al sistema clásico uGUI. Destacan la gestión global de estilos mediante USS reutilizable, inexistente en uGUI y la posibilidad de crear interfaces sin requerir texturas usando elementos vectoriales, lo que reduce memoria y llamadas de redibujo (draw calls). Además, UI Toolkit soporta de serie atlas dinámicos de texturas, anti-alias y animaciones de transición integradas, características para los cuales uGUI requiere plugins o no brinda. La Tabla 5.1 muestra una comparación de las características que ofrece cada uno de los sistemas de construcción de interfaces gráficas de Unity.

La documentación oficial señala que, para proyectos con múltiples resoluciones o UI en 3D/VR, UI Toolkit es la opción preferida, ya que "*ofrece mejor reutilización, desacoplamiento y escalabilidad tanto en código como en rendimiento*". En cambio, uGUI se reserva para casos con efectos gráficos muy específicos como el uso de shaders personalizados. (Unity Technologies, 2025c)

Característica	UGUI	UI Toolkit
Año de Lanzamiento	2014	2019
Arquitectura	Basado en *GameObject*	Basado en *Documentos*
Rendimiento	Decente para UI simple	Excelente para UI compleja
Animación	Animación completa con Animator/Timeline	Transiciones similares a CSS
Editor Visual	Vista en escena	UI Builder
Curva de Aprendizaje	Familiar para desarrolladores de Unity	Patrones similares a desarrollo web
Rendimiento Móvil	Decente	Superior
Data-Binding	Manual	Incorporado
Soporte para Asset Store	Extensivo	Limitado
Ideal para	Juegos y Animaciones	Apps, UI con datos

Tabla 5.1: Comparativa rápida entre uGUI y UI Toolkit
Fuente: Adaptado de Angry Shark Team (2025)

5.4.4. Animaciones declarativas mediante estilos

Las animaciones en UI Toolkit se definen de forma declarativa mediante USS. Usando propiedades CSS-like (`transition-duration`, `transition-delay`, etc.) en las hojas de estilo, los cambios de clases o propiedades entre estados provocan transiciones interpoladas automáticamente.

Un ejemplo directo es el agregar o quitar una clase USS, UI Toolkit interpolará los valores de estilo (color, posición, etc.) de manera suave. Internamente se generan eventos específicos: al iniciar una transición surge un `TransitionStartEvent` y al finalizar un `TransitionEndEvent` en el elemento implicado. Esto permite reaccionar al ciclo de vida de las animaciones (por ejemplo, encadenarlas o liberar recursos al acabar). En definitiva, la Toolkit aprovecha un modelo declarativo muy parecido a CSS para manejar animaciones y estados, simplificando la lógica de efectos visuales. (Unity Technologies, 2025l)

5.4.5. Instanciación dinámica de vistas por medio de patrón Factory

Para generar dinámicamente vistas desde plantillas, UI Toolkit ofrece la clase `VisualTreeAsset`, que representa un archivo UXML cargable en tiempo de ejecución. Al cargar un `VisualTreeAsset`, es posible llamar a `Instantiate()` o `CloneTree()` para crear instancias (templating) de la interfaz en cualquier `VisualElement` padre. (Unity Technologies, 2025h)

Este enfoque es análogo al patrón Factory, que es un patrón de diseño creacional que define una interfaz para la creación de objetos y da libertad a subclases de decidir que objeto instanciar, promoviendo el desacoplamiento encargando la creación de objetos a un método (Kartik, 2025). En el caso de UI se define una plantilla por medio de un UXML y el código C# es el que actúa como fábrica para instanciar o desechar paneles complejos bajo demanda. Así cada vez que se necesite una nueva vista, como una ventana emergente, simplemente se instancia un UXML correspondiente al árbol visual, logrando reutilizar diseños sin duplicar código.

Sin embargo, cabe notar que UI Toolkit no destruye automáticamente estos elementos; queda en manos del recolector de GC liberarlos cuando no existan referencias. Por eso, al generar o destruir muchos elementos consecutivamente, conviene habilitar la recolección de basura incremental de Unity para evitar caídas de fotograma por limpiezas repentinas. (Unity Technologies, 2025h)

5.4.6. Control de batching, reflujo y repintado

UI Toolkit usa el motor de layout Yoga (basado en Flexbox) para calcular posiciones. Esto significa que cambios en propiedades de layout pueden disparar un reflow, i.e. el recalculo de la disposición en todo un nodo del árbol visual. El reflow es costoso, ya que implica computar de nuevo posiciones y tamaños. Para optimizar, se debe evitar modificar constantemente estilos que afectan el layout. Por ejemplo, agrupar actualizaciones o alternar clases en vez de cambiar propiedades individuales repetidamente. Además, UI Toolkit construye por defecto un atlas dinámico de texturas que agrupa múltiples elementos visuales en una sola textura para reducir llamadas de redibujo (draw calls). Este atlas mejora el batching gráfico en tiempo de ejecución, combinando elementos relacionados. (Unity Technologies, 2025j)

5.4.7. Buenas prácticas de optimización de UI

Unity, a lo largo de su documentación, presenta diferentes buenas prácticas para evitar procesamiento y carga adicional en los dispositivos. En el contexto de UI Toolkit y la gestión del árbol visual, se recomiendan las siguientes:

- **Reutilización de estructuras de UI y pooling de elementos:** Se aconseja reutilizar estructuras UI en lugar de crearlas desde cero en cada actualización. Para ello, se deben almacenar referencias a árboles visuales y a los resultados de `UQuery`, evitando repetir búsquedas costosas. El uso de `QueryState` para cachear consultas contribuye a reducir la sobrecarga de instanciar listas y elementos de forma repetida.

- **Minimizar creaciones masivas y gestionar el Garbage Collector:** Cuando se generan o destruyen grandes cantidades de elementos en poco tiempo, se incrementa la presión sobre el recolector de basura. Unity recomienda habilitar el Garbage Collector incremental para repartir la carga de limpieza entre varios fotogramas y evitar picos de latencia. Además, se sugiere evitar ciclos intensivos en una sola actualización del sistema, distribuyendo las operaciones de creación y destrucción a lo largo del tiempo.

- **Reducir enlaces y recalculados de estilo:** Se recomienda vincular solo los datos estrictamente necesarios y evitar enlaces redundantes. Al actualizar el estado visual, es más eficiente agrupar las modificaciones de estilo que aplicar cambios propiedad por propiedad. Ajustar varias clases de estilo de forma conjunta reduce el número de recalculados de layout y de re-evaluación de estilos. (Unity Technologies, 2025c)

5.5. Fundamentos del audio en Unity

En el motor de juego Unity, la infraestructura de audio está diseñada para simular la experiencia auditiva humana, un componente crítico para la inmersión en sistemas interactivos.

5.5.1. Modelo fuente-escucha

La representación del audio en un entorno virtual se basa fundamentalmente en el modelo fuente-escucha, que define la relación espacial y la interacción entre el sonido emitido y el sonido percibido. El sistema de audio de Unity se estructura alrededor de dos componentes esenciales: el `AudioSource` y el `AudioListener`.

El `AudioSource` actúa como la fuente acústica dentro del mundo digital, siendo el responsable de reproducir un `AudioClip`. Por otro lado, el `AudioListener` representa al oyente o receptor del sonido. Este componente es el encargado de emular la posición, la velocidad y la orientación del usuario en el espacio tridimensional virtual.

El motor de Unity distingue entre dos modos principales de procesamiento de audio, controlados por el `AudioSource`:

- **Audio 2D (No Espacial):** Este modo ignora cualquier procesamiento tridimensional, como la atenuación por distancia o el posicionamiento direccional. El sonido 2D es adecuado para elementos de la interfaz de usuario, diálogos globales que no están asociados a una posición específica, o música de fondo no diegética, es decir, que existe fuera del mundo del juego.

- **Audio 3D (Espacial):** El audio 3D requiere que la señal sea procesada por el `AudioListener` en función de la distancia y el ángulo relativo entre la fuente y el oyente, lo que permite la localización del sonido en el espacio.

El control de mezcla entre estos dos modos se realiza mediante el parámetro `SpatialBlend`. Un valor de 0.0f indica audio puramente 2D, mientras que un valor de 1.0f indica audio completamente 3D. Valores intermedios representan una mezcla entre ambos. (Unity Technologies, 2025b, 2025p)

5.6. Tecnologías de síntesis de voz

La elección y la configuración de la tecnología de Síntesis de Voz (TTS) son determinantes para la calidad y la expresividad de la narrativa de un sistema interactivo. La evolución reciente en este campo ha pasado de los sistemas basados en reglas y concatenación a los modelos de lenguaje neural guiados por la intención.

5.6.1. Fundamentos del sistema texto a voz

Un sistema TTS tradicional opera en varias etapas para convertir texto plano en audio audible:

- **Análisis Lingüístico:** El texto de entrada se somete a un procesamiento para resolver ambigüedades, normalizar números y abreviaturas, y generar una transcripción fonética (la secuencia de sonidos del habla).
- **Análisis Prosódico:** Se genera un contorno que especifica cómo debe ser entonada la frase. Este contorno incluye información sobre la duración de las sílabas, la intensidad física de la onda acústica y la frecuencia fundamental, que determina la altura tonal.9
- **Conversión Acústica:** El modelo sintetiza la forma de onda de audio final basándose en los parámetros prosódicos y fonéticos calculados. (Zilliz, 2025)

Por otra parte, la prosodia, que incluye rasgos suprasegmentales como el ritmo, la entonación y la intensidad, no es meramente un adorno acústico, sino un elemento fundamental del significado lingüístico. Estos elementos prosódicos definen la función comunicativa del enunciado, diferenciando, por ejemplo, entre una modalidad declarativa y una interrogativa. (Akerberg et al., 2015)

5.6.2. SSML como control sintáctico y marcado de voz en TTS

El estándar Speech Synthesis Markup Language (SSML) es una herramienta declarativa basada en XML que permite al desarrollador ejercer un control explícito sobre la salida de voz sintética. Mediante etiquetas de marcado, se pueden especificar la voz, el lenguaje y el rol del locutor. Etapas de control prosódico, como la modificación del ritmo (rate), el tono (pitch) y el volumen, se logran mediante la etiqueta `<prosody>`.

El SSML ofrece la ventaja de un control estructurado y predecible. Es útil para la inserción precisa de pausas (`<break>`) o la incrustación de audio pregrabado dentro de una secuencia generada.11 Sin embargo, su control es fundamentalmente sintáctico.

La tarea de generar expresividad emocional o tonos sutiles requiere la manipulación manual de múltiples parámetros acústicos. Este proceso es tedioso, difícil de escalar y a menudo resulta en un habla que puede percibirse como robótica o inexpresiva. (Microsoft, 2025)

5.6.3. Síntesis expresiva con modelos neuronales

La nueva generación de sistemas TTS ha migrado hacia arquitecturas basadas en modelos de lenguaje (LLMs) y representaciones discretas del habla. Este enfoque permite desacoplar conceptualmente la tarea de "lectura" (convertir texto en semantic tokens) de la tarea de "habla" (convertir semantic tokens en acoustic tokens), posibilitando una generación de voz de mucha mayor fidelidad y controlabilidad. (Kharitonov et al., 2023)

Gemini-TTS ilustra esta evolución al permitir un control granular y avanzado sobre el audio generado utilizando prompts en lenguaje natural. En lugar de manipular directamente las propiedades acústicas, el desarrollador proporciona instrucciones de alto nivel, como dictar el estilo, el acento, el ritmo, el tono o la expresión emocional deseada. (Google, 2025b)

Este cambio de paradigma es altamente beneficioso para la tesis. Permite generar narrativas dinámicas con calidad de producción, donde la voz puede ser controlada con precisión para transmitir la intención requerida. Esta capacidad de control semántico simplifica la creación de una narrativa emocionalmente rica, superando significativamente las limitaciones de la rigidez paramétrica del SSML. (Google, 2025d)

5.6.4. Comparación entre SSML y Prompts

La justificación para la selección de una tecnología TTS de última generación se basa en el contraste entre las filosofías de control. El SSML requiere que el desarrollador prescriba explícitamente los atributos acústicos (control sintáctico). Por el contrario, el control basado en prompts se centra en la instrucción de la intención, confiando en la capacidad del modelo neuronal para inferir y generar la prosodia más natural y expresiva para cumplir con la intención descrita (control semántico). La Tabla 5.2 busca resumir los contrastes que hay entre ambos paradigmas en la síntesis de voz.

Criterio	**TTS Clásico (SSML)**	**TTS Guiado por Prompts (Gemini-TTS)**
Mecanismo de Control	Sintáctico y declarativo (XML).	Semántico, basado en lenguaje natural/instrucción.
Control de Estilo/Tono	Paramétrico (pitch, rate).	Holístico, por descripción narrativa (tono, emoción).
Arquitectura Subyacente	Paramétrica o neural anterior a LLM.	Modelo de lenguaje neural avanzado (LLM).
Ideal para	Recitación simple, control de pausa preciso.	Narrativa compleja, alta expresividad, control multispeaker.

Tabla 5.2: Comparativa entre TTS por medio de SSML y Prompts
Fuente: Elaboración propia

5.6.5. Parámetros técnicos del audio generado

Para preservar la alta calidad de la voz generada por un modelo TTS avanzado, es fundamental utilizar formatos de codificación sin pérdida. Se recomienda el uso de códecs como PCM Lineal (`LINEAR16`) o `FLAC`. El uso de formatos con pérdida, como `MP3`, durante la grabación o transmisión puede degradar las sutiles características prosódicas generadas por el modelo neuronal, reduciendo la exactitud y la calidad percibida de la voz.(Google, 2025c)

Se establece como práctica recomendada capturar y transmitir audio con una tasa de muestreo de 16,000 Hz o superior. Una tasa de muestreo adecuada es crucial no solo para la fidelidad general de la voz, sino también para mantener la integridad de las frecuencias necesarias para que el motor de audio de Unity pueda aplicar filtros de espacialización 3D de manera efectiva.(Google, 2025a)

6. Metodología

El desarrollo del proyecto se estructuró en tres macro fases que guiaron la ejecución de las actividades técnicas. La primera correspondió al diseño e implementación del sistema de tours y navegación, en el cual se definieron las estructuras de datos y los algoritmos de path-finding que permiten el desplazamiento controlado dentro del entorno virtual. La segunda fase se enfocó en la construcción del sistema de interfaz de usuario, mediante la cual se visualizaron el progreso, los mensajes y los eventos del recorrido, garantizando la separación entre la lógica interna y la presentación visual. Finalmente, la tercera fase comprendió el desarrollo del sistema de audio y guía por voz, que integró la generación, administración y reproducción de mensajes sonoros asociados a los puntos de interés del recorrido.

6.1. Arquitectura general del sistema

La arquitectura del sistema se refinó de manera iterativa conforme se ampliaron las responsabilidades del módulo de navegación y guía, hasta consolidar un modelo por capas, compuesto por tres subsistemas principales:

- Sistema de Tour y Navegación
- Sistema de Interfaz de Usuario (UI)
- Sistema de Audio y Voz

Cada subsistema cumple una función específica dentro del flujo general de la aplicación, garantizando independencia entre la lógica, la presentación y los servicios complementarios de audio y posicionamiento. La Figura 6.1 muestra la arquitectura general del sistema organizada en torno a estos tres subsistemas.

Este enfoque permitió mantener una separación clara de responsabilidades entre los componentes de posicionamiento, cálculo de rutas, gestión de pisos y control del flujo del recorrido, además de los módulos encargados de la interfaz gráfica y la reproducción de mensajes de voz.

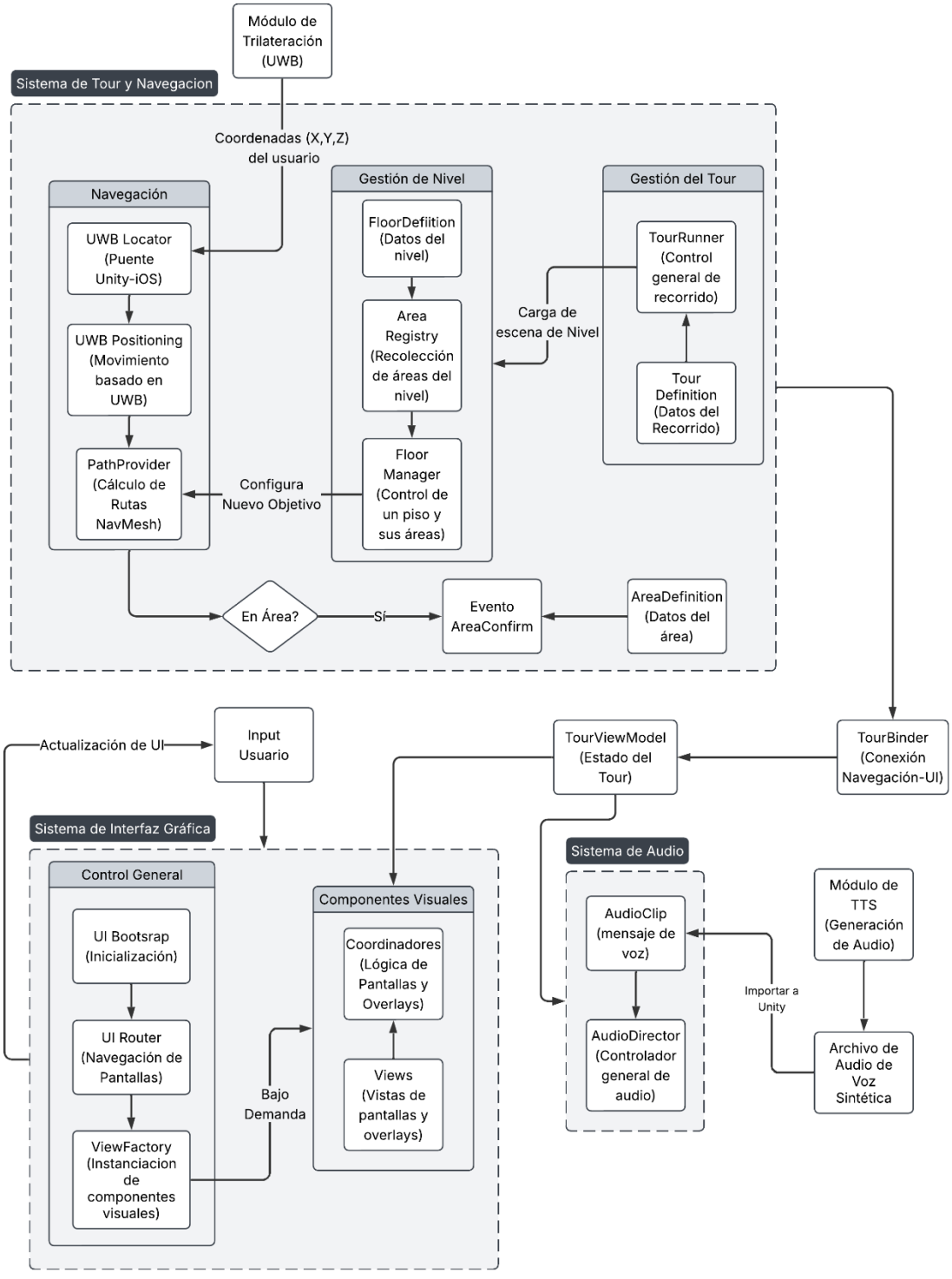

Figura 6.1: Diagrama de arquitectura general de la aplicación
Fuente: Elaboración propia

A continuación, se describen las fases que conformaron el desarrollo del sistema, siguiendo esta estructura arquitectónica como guía para el diseño e integración de cada componente.

6.2. Desarrollo de Sistema de Tours y Navegación

La primera fase de desarrollo se centró en el diseño e implementación del sistema de tours y navegación, cuyo propósito fue cumplir los objetivos específicos 1 y 2: calcular rutas entre puntos de interés dentro del Centro de Innovación y Tecnología (CIT) e integrar la localización en tiempo real del usuario mediante el módulo externo de trilateración por UWB.

6.2.1. Integración del módulo de posicionamiento UWB

El punto de partida consistió en integrar, dentro de Unity, el módulo de trilateración desarrollado como un plugin nativo para iOS. Para ello se implementó un puente entre Unity y el código nativo mediante el uso de funciones externas importadas con `DllImport`, de forma que el motor pudiera solicitar coordenadas calculadas por el módulo de posicionamiento y liberarlas adecuadamente en memoria.

Dado que el objetivo del proyecto fue ejecutar la aplicación en dispositivos iOS con soporte para UWB, se emplearon directivas de compilación condicional (`UNITY_IOS`, `UNITY_EDITOR`) para habilitar la integración únicamente en compilaciones para dispositivo y desactivar el posicionamiento en el editor de Unity u otras plataformas. La clase responsable de esta interfaz (`UWBLocator`) encapsula la llamada a las funciones nativas, transformando el JSON recibido en una estructura de coordenadas internas y lo convierte a posiciones tridimensionales en el entorno virtual. El código completo de esta implementación se presenta en el Anexo E. Código fuente del puente nativo Unity/iOS.

Para validar la integración se diseñó inicialmente una escena de prueba que representa una habitación reducida en 3D. En esta escena se colocó un objeto de tipo cápsula como representación del usuario, el cual se actualiza con las coordenadas proporcionadas por el módulo de trilateración. Las pruebas consistieron en verificar la existencia de comunicación efectiva con el plugin, la actualización continua de la posición y la correspondencia cualitativa entre el movimiento físico del usuario y su réplica en el entorno virtual.

Con el fin de garantizar la robustez del sistema de posicionamiento se implementó una capa adicional (`UWBPositioning`) encargada de:

- Consultar periódicamente el módulo UWB.
- Detectar pérdida de señal mediante conteo de lecturas nulas consecutivas.
- Proyectar las posiciones reales sobre la malla de navegación (NavMesh) para evitar posiciones inválidas.

Adicionalmente, debido a que no era posible realizar compilaciones directas a iOS fuera del laboratorio de Arte y diseño digital (por la necesidad de una Mac con Xcode), se desarrolló un componente alterno de posicionamiento por teclado (`KeyboardPositioning`). Este componente permite simular el desplazamiento del usuario en el editor de Unity, manteniendo el resto del sistema (cálculo de rutas, detección de áreas, etc.) sin cambios. Un componente unificador (`MovementAgent`) habilita dinámicamente el modo de movimiento apropiado según la plataforma de ejecución.

6.2.2. Modelado de navegación con NavMesh y ajuste de rutas

Una vez integrada la localización, se abordó el diseño del sistema de navegación y cálculo de rutas, orientado a cumplir el objetivo específico 1. Para ello se empleó el sistema de navegación de Unity basado en NavMesh. En la escena de prueba inicial se generó una superficie de NavMesh directamente sobre la geometría 3D de la habitación, y se implementó un primer prototipo de cálculo de ruta entre la posición actual del usuario y un objetivo. Sobre este prototipo se desarrolló un componente especializado (`PathProvider`) encargado de:

- Tomar la posición actual del usuario en el NavMesh.
- Tomar el punto objetivo asociado al área de interés.
- Calcular el camino más eficiente mediante `NavMesh.CalculatePath`.
- Exponer la ruta resultante y su longitud total para otros sistemas (por ejemplo, para retroalimentar la interfaz y el sistema de guía por voz).

Para facilitar la depuración y el análisis del comportamiento del algoritmo se implementó un visualizador de caminos (`PathRenderer`) que dibuja la secuencia de puntos de la ruta tanto en el editor como en ejecución. Esto permitió evaluar visualmente si las rutas seguían trayectorias naturales dentro de los pasillos. Este componente encargado del cálculo de rutas se documenta en el Anexo F. Código fuente del Path-Finding.

Durante las pruebas en entornos más complejos se observó que, debido a la naturaleza del algoritmo de path-finding y a la discretización del NavMesh, el agente tendía a acercarse a las orillas y esquinas de la geometría del entorno, produciendo rutas poco naturales. Para corregir este comportamiento se utilizó la configuración de áreas y costos del NavMesh, en combinación con NavMesh Modifiers. Inicialmente se consideraron volúmenes (volume modifiers) superpuestos sobre la geometría base para redefinir áreas con diferente costo de navegación. Sin embargo, en escenarios grandes esto implicaba manejar múltiples volúmenes.

Como solución más escalable se modeló una franja delgada de geometría adicional que recorre el centro de los corredores del CIT. Sobre esta geometría se aplicó un modificador de NavMesh que baja el costo de navegación, mientras que la malla base se configuró con un costo más alto. De esta manera, el algoritmo privilegia rutas centradas en los pasillos, sin marcar la malla base como no caminable, sino simplemente menos preferible. Este ajuste produce trayectorias más cercanas a cómo se desplazaría un usuario real, manteniendo flexibilidad para desviar el recorrido cuando es necesario.

6.2.3. Modelo de datos para tours, niveles y áreas de interés

Con la infraestructura de posicionamiento y navegación establecida, se diseñó un modelo de datos orientado a representar recorridos completos dentro del CIT, siguiendo un enfoque data-driven basado en `ScriptableObject`. El objetivo fue separar los datos de configuración del comportamiento, para facilitar la definición y edición de tours directamente desde el editor de Unity.

El modelo se estructuró en tres capas:

- **Área de interés**: unidad básica del recorrido. Cada área se representa mediante un `ScriptableObject` que almacena su nombre, descripción textual, recursos asociados (imágenes, audios) e información para la interfaz de usuario. En la escena, cada área se instancia como un prefab que incluye un `BoxCollider` configurado como trigger, lo que permite detectar la entrada y salida del usuario. Componentes auxiliares (`AreaGizmo`, `AreaInstance`, `AreaTrigger`) se utilizan para visualizar el volumen del área en el editor y para emitir eventos cuando el usuario entra o sale de ella.

- **Nivel o piso**: cada nivel del edificio se modela como un `ScriptableObject` de tipo piso, que contiene una lista ordenada de áreas a visitar y una referencia a la escena 3D correspondiente. A nivel de escena se mantiene únicamente la geometría y las áreas de un piso, junto con un punto de origen que permite alinear el mundo virtual con la disposición física de los sensores UWB. El piso mantiene también la referencia al archivo JSON de AnchorMap, que describe la distribución de sensores UWB. este JSON se carga mediante el módulo nativo de trilateración al entrar al piso correspondiente.

- **Tour**: un tour completo se representa como un `ScriptableObject` que contiene un listado ordenado de pisos. Este recurso define el orden en que se recorren los niveles del edificio y permite calcular el número total de áreas del tour, lo cual se utiliza posteriormente para el cálculo del progreso global del recorrido.

En tiempo de ejecución, varios componentes coordinan el funcionamiento de este modelo. Un registro de áreas (`AreaRegistry`) localiza en la escena las instancias de cada `AreaDefinition`, mientras que un gestor de piso (`FloorManager`) se encarga de activar únicamente el área relevante en cada momento, prevenir activaciones accidentales y actualizar el objetivo del `PathProvider` con el centro del área activa. Un coordinador global (`TourRunner`) controla la carga y descarga aditiva de escenas de piso, aplica el AnchorMap correspondiente a cada nivel y mantiene el conteo de áreas visitadas a lo largo del tour.

6.2.4. Estrategia de pruebas de la fase de navegación

Las pruebas de esta fase se realizaron de manera incremental. En la etapa de integración con el módulo UWB, las pruebas se centraron en:

- Verificar la recepción continua de coordenadas desde el plugin.
- Confirmar la transformación correcta de coordenadas reales a posiciones dentro del entorno virtual.
- Detectar y manejar adecuadamente situaciones de pérdida de señal.

Posteriormente, sobre la escena de prueba y más tarde sobre los modelos 3D del CIT, se llevaron a cabo pruebas funcionales del sistema de navegación:

- Validación de que el cálculo de rutas se actualiza al cambiar la posición del usuario y el objetivo.

- Observación de las trayectorias generadas mediante el visualizador de caminos, comparándolas con recorridos esperados en los pasillos.
- Verificación de que los ajustes de costos de área inducen rutas centradas en los pasillos, evitando bordes de la geometría salvo cuando es estrictamente necesario.

Finalmente, se validó el modelo de datos de tours y niveles comprobando que:

- El orden de áreas definido en los `ScriptableObject` se respeta en la ejecución.
- Solo el área activa se encuentra disponible para activación, reduciendo falsos positivos por errores de trilateración.
- La carga de AnchorMaps por piso se realiza de forma consistente, garantizando la alineación entre el posicionamiento UWB y la geometría 3D del nivel.

Todas las clases, escenas de prueba y recursos descritos en esta fase fueron desarrollados específicamente para este proyecto.

6.3. Integración de Interfaz de usuario con el sistema de navegación

La segunda fase consistió en la implementación de la interfaz de usuario basada en UI Toolkit y su integración con la lógica de navegación desarrollada en la fase anterior. El objetivo fue exponer al usuario el estado del tour (progreso, distancia a la siguiente área, mensajes contextuales) mediante una capa de presentación desacoplada de los detalles internos del motor de navegación, evitando refactorizaciones profundas del código ya existente.

6.3.1. Diseño arquitectónico de la capa de presentación

Para estructurar la interfaz se adoptó un enfoque inspirado en el patrón Model–View–ViewModel (MVVM), complementado con coordinadores que encapsulan la lógica de interacción por pantalla. La arquitectura de la capa de presentación se organizó en tres elementos principales:

- **Modelo de vista (ViewModel):** se definió una clase `TourViewModel` como fuente única de verdad respecto al estado visible del tour. Este modelo almacena, entre otros, el área actual, el siguiente destino, el progreso normalizado del recorrido, la distancia restante calculada por el sistema de navegación y el estado de conexión del módulo de posicionamiento UWB. Además, expone un conjunto de eventos mediante los cuales las vistas son notificadas de cambios.

- **Enlace con la lógica de dominio**: se implementó un componente `TourBinder` responsable de suscribirse a los eventos del sistema de dominio (`TourRunner`, `FloorManager`, `PathProvider`, `UWBPositioning`) y traducirlos a actualizaciones sobre el `TourViewModel`. Por ejemplo, al recibir un evento de "piso cargado", el binder actualiza el piso actual en el modelo de vista, coloca la interfaz en fase de "esperando conexión" y reinicia la distancia; cuando se confirma la entrada a una nueva área, actualiza el progreso global y dispara la reproducción de la secuencia de audio asociada a dicha área.

- **Coordinadores por pantalla**: sobre este modelo se declararon coordinadores específicos por tipo de pantalla (`HomeCoordinator`, `HUDCoordinator`, `OnboardingCoordinator`, etc.), cuya responsabilidad es conectar un ViewModel con una vista concreta, reaccionar a eventos de la UI y orquestar cambios de pantalla mediante el enrutador de interfaz `UIRouter`.

Este diseño permitió extender la interfaz sin modificar la lógica central de navegación, limitando los cambios a la capa de presentación y a la clase de enlace (`TourBinder`).

6.3.2. Infraestructura de UI Toolkit y gestión de pantallas

La implementación de la UI se basó en UI Toolkit y en el uso de plantillas UXML para describir las estructuras visuales. Con el fin de evitar dependencias directas y facilitar la reutilización, se definió un recurso `UIAtlas` (`ScriptableObject`) que centraliza las referencias a:

- Plantillas UXML de pantallas principales (inicio, minijuegos, onboarding, HUD del tour).
- Plantillas UXML de overlays (ventanas modales, popups de aviso, menú colapsable, pantalla de ajustes).
- Recursos asociados como íconos del HUD y configuraciones predefinidas de mensajes.

A partir de este atlas se implementó un `ViewFactory`, responsable de clonar dinámicamente las plantillas UXML y devolver instancias de vistas fuertemente tipadas (`HomeView`, `BaseHUDView`, `ActionPopupView`, etc.). De este modo, la lógica nunca manipula directamente UXML, sino objetos de vista que encapsulan los elementos relevantes de la jerarquía.

La navegación entre pantallas se gestionó a través de un componente `UIRouter`, construido sobre un layout base con varias capas visuales:

- Capa de base para las pantallas principales.
- Capa de modales informativos.
- Capa de popups de acción/aviso.
- Capa de menú.
- Capa de ajustes.

El enrutador se encarga de:

- Reemplazar la pantalla base actual por la nueva vista (Home, Onboarding, TourHUD, etc.).
- Mostrar u ocultar overlays según su tipo (informativo, confirmación de acción, menú, ajustes), asegurando que cada overlay se renderice en la capa correspondiente.
- Mantener un registro de los overlays activos para poder ocultarlos en bloque cuando el estado del tour lo requiera.

Adicionalmente, se implementaron utilidades de picking (`UIPickingUtils`) para controlar el modo de captura de eventos de puntero en cada capa, de manera que las capas invisibles no intercepten interacciones destinadas a otras capas.

6.3.3. Integración de estados del tour con la interfaz

Una vez establecida la infraestructura de vistas y enrutador, se desarrolló la lógica específica del HUD del tour y de las ventanas emergentes que guían al usuario a lo largo del recorrido. Esta integración pivota sobre el `TourViewModel` y un conjunto de estados discretos de la UI (`TourUIPhase`), entre ellos:

- **WaitingForConnection**: a la espera de establecer comunicación con el módulo UWB.
- **ConnectionLost**: pérdida temporal de señal.
- **ReadyPrompt**: el sistema está listo y solicita confirmación del usuario para iniciar o continuar el recorrido.
- **Navigating**: el usuario se encuentra en tránsito hacia la siguiente área.
- **InAreaInfo**: el usuario ha ingresado a un área de interés y se muestra la información correspondiente.
- **FloorTransition**: transición entre pisos.
- **TourComplete**: fin del recorrido.

El coordinador del HUD (`HUDCoordinator`) suscribe el `TourViewModel` y, en función de la fase, decide qué overlays mostrar:

- En fases de espera o error de conexión se muestra un popup de aviso (`NoticePopupView`) con mensajes preconfigurados de “conectando” o “conexión perdida”.

- En la fase ReadyPrompt se presenta un popup de acción (`ActionPopupView`) que solicita al usuario iniciar el tour o continuar en el siguiente piso, empleando datos configurados en `ActionData` (texto, arte, audio).

- En Navigating se mantiene visible únicamente el HUD base, que muestra la distancia al siguiente punto y el progreso global del tour.

- En InAreaInfo se activa un widget informativo (`InfoWidgetView`) que despliega el contenido asociado al área (`AreaDefinition`: nombre, texto descriptivo, imagen, ícono).

La vinculación de estos estados con la lógica de dominio se realiza a través del `TourBinder`, el cual actualiza la distancia en el modelo de vista a partir de los datos del `PathProvider` (NavMesh). También propaga los eventos de entrada a un área (`AreaConfirmed`) para que el HUD muestre la información correspondiente y, en caso de áreas sin información visual, avance automáticamente a la siguiente área mediante una llamada al sistema de tours. Y reacciona a cambios de estado en la conexión UWB para alternar entre pantallas de conexión, navegación o advertencia según corresponda.

6.3.4. Flujo de pantallas y onboarding del usuario

Además de la UI específica del tour, se implementó un flujo de inicio que incluye una pantalla de bienvenida y una secuencia de onboarding. El comportamiento se controló mediante:

- Un recurso `OnboardingSet` que agrupa las diapositivas (texto, arte y audio opcional) que se presentan al usuario la primera vez que abre la aplicación.

- Una clase estática `OnboardingGate` que almacena en preferencias (PlayerPrefs) la fecha de la última visualización de la secuencia de onboarding, así como un indicador para forzar su presentación en cada ejecución durante las pruebas.

- Un coordinador `OnboardingCoordinator` y una vista `OnboardingView` que permiten recorrer las diapositivas, controlar el texto del botón ("Siguiente" o "Empezar") y reproducir la narración asociada a cada una.

En la pantalla de inicio (`HomeView`), coordinada por `HomeCoordinator`, se ofrecen opciones para iniciar un tour "exprés" o un tour "completo". Al seleccionar una opción, el coordinador configura el tour correspondiente en `TourRunner`, inicia el recorrido y ordena al enrutador cambiar a la pantalla del HUD del tour.

6.3.5. Pruebas de la fase de interfaz

Las pruebas de esta fase se orientaron a validar el funcionamiento correcto de la capa de UI y su coherencia con el estado interno del sistema. Entre las verificaciones realizadas destacan:

- Comprobación de que cada transición de estado en `TourViewModel` produce el cambio esperado en la interfaz.

- Validación de que los overlays no interfieren entre sí, que se ocultan al cambiar de pantallas y que las capas invisibles no capturan eventos de puntero gracias al ajuste de PickingMode.

- Pruebas del flujo completo desde la pantalla de inicio: selección de tipo de tour, presentación de la pantalla de HUD, conexión/desconexión del módulo UWB, ingreso a áreas de interés, transición entre pisos y finalización del tour.

- Verificación de la persistencia del estado de onboarding mediante el uso de PlayerPrefs, garantizando que la secuencia de bienvenida no se repite en cada ejecución en condiciones normales.

Toda la infraestructura de UI descrita en esta fase, así como los recursos asociados fueron desarrollados específicamente para este proyecto.

6.4. Desarrollo de sistema de generación de audio y guía por voz

La tercera fase del desarrollo correspondió a la implementación del sistema de audio, dividido en dos componentes: la generación automatizada de audios de voz mediante modelos de síntesis de texto a voz (TTS) de Google, y la integración de dichos audios dentro del entorno de Unity mediante un controlador de audio general (`AudioDirector`). Esta fase tuvo como objetivo final proporcionar una experiencia auditiva natural y coherente con el recorrido virtual, sincronizando los mensajes narrativos con las acciones y eventos del usuario en tiempo real.

6.4.1. Generación de audios de voz con Gemini TTS

En la formulación inicial del proyecto se contempló el uso de SSML (Speech Synthesis Markup Language) junto con las voces Studio y Polyglot de Google Cloud TTS, disponibles durante las primeras fases de desarrollo. Estas voces ofrecían un nivel adecuado de calidad, pero requerían la escritura de etiquetas SSML específicas para lograr variaciones naturales de entonación, lo que resultaba costoso en tiempo y mantenibilidad.

Durante el segundo semestre del año, Google actualizó su infraestructura de síntesis de voz, deprecando los modelos antiguos y lanzando los modelos Chirp3HD y posteriormente Gemini 2.5 Pro TTS, basados en un enfoque de prompting con modelos de lenguaje. Este nuevo paradigma permitía reemplazar las instrucciones SSML por lenguaje natural, utilizando prompts expresivos como “di lo siguiente de forma entusiasmada”, obteniendo resultados más naturales y consistentes sin la necesidad de escribir etiquetas.

El cambio de modelo redujo el tiempo de producción de audios, mejoró la naturalidad del tono y simplificó la integración en la línea de generación, por lo que se abandonó definitivamente el uso de SSML. El modelo final adoptado fue “gemini-2.5-pro-tts”, utilizando la voz “Enceladus” en español de Estados Unidos (es-US), configurada por defecto en todos los scripts de generación.

Este ajuste representó una modificación metodológica respecto al diseño original planteado en los objetivos, que hacía referencia explícita a SSML; sin embargo, mantuvo intacto el propósito de dicho objetivo, al seguir proporcionando un mecanismo de generación de voz sintética configurable y adecuado para dispositivos móviles.

El sistema de generación de audios se implementó en Python, empleando la biblioteca oficial de Google Cloud para la API de Text-to-Speech. Se compone de tres módulos principales.

El primero es `gemini_tts.py` responsable de la interacción directa con la API de Google TTS. Este módulo define funciones para:

- Crear el cliente de conexión (`build_client`).
- Configurar los parámetros de voz (`build_voice`).
- Establecer la configuración de salida de audio (`build_audio_cfg`), en formato PCM lineal (`LINEAR16`) a 24 kHz.
- Ejecutar la síntesis (`synthesize`), recibiendo un prompt y un texto principal para generar un audio `.wav` lossless.

El flujo de esta función encapsula el nuevo paradigma de prompting, recibiendo como entrada un texto de estilo y un texto narrativo, enviándolos como un solo objeto `SynthesisInput` a la API de Gemini TTS. El resultado se devuelve como un arreglo de bytes PCM listo para ser grabado en disco. El código completo de este módulo de generación de voz utilizado para sintetizar los audios se presenta en el Anexo H. Código fuente principal del sistema de Generación de Voz.

El segundo modulo es `utilities.py`, módulo de utilidades para manejo de archivos y estructura de carpetas. Dentro se implementa:

- Detección recursiva de archivos JSON en un directorio (`discover_json`).
- Validación y lectura de datos de entrada (`read_job`), que esperan dos campos: prompt y text.
- Escritura del audio resultante en formato `.wav` (write_wav), asegurando compatibilidad con los parámetros de muestra predeterminados (16-bit mono a 24 kHz).
- Generación de rutas relativas para replicar la estructura de carpetas en la salida (`rel_out_path`).

Con ello, el sistema permite procesar una carpeta completa de guiones sin intervención manual, generando automáticamente una estructura espejo con los audios sintetizados.

Finalmente, el módulo `main.py`, script principal de ejecución por la línea de comandos. Es el que orquesta el flujo completo:

- Acepta parámetros mediante flags (`--input, --output, --model, --voice, --lang, --samplerate`).
- Descubre los archivos JSON de entrada y valida su contenido.
- Inicializa el cliente de Google TTS y las configuraciones predeterminadas.
- Recorre cada archivo, sintetiza el audio y lo guarda en la ruta correspondiente.

El script incluye control de errores, manejo de excepciones (`GoogleAPICallError`) y un resumen final de procesamiento con el número de archivos generados exitosamente y los fallidos. En la Tabla 6.1 se muestran los valores por defecto establecidos para la generación de voz.

Parámetro	Valor por defecto	Descripción
BYTE_LIMIT	900 bytes	Límite de tamaño por campo según la API de Google
DEFAULT_MODEL	"gemini-2.5-pro-tts"	Modelo de síntesis adoptado
DEFAULT_VOICE	"Enceladus"	Voz principal del sistema
DEFAULT_LANG	"es-US"	Idioma configurado
DEFAULT_SAMPLE_RATE	24000	Frecuencia de muestreo en Hz

Tabla 6.1: Valores por defecto para el módulo generador de voz
Fuente: Elaboración propia

De esta forma, el sistema podía ejecutarse con un único comando y procesar en lote los guiones del tour, generando audios uniformes y de alta calidad sin necesidad de edición manual posterior.

6.4.2. Integración del sistema de audio dentro de Unity

Una vez generados los audios, la segunda parte de esta fase consistió en integrar el sistema sonoro en la arquitectura de Unity, garantizando su sincronización con los eventos del recorrido y la interfaz.

Para centralizar la gestión de todos los audios del proyecto se desarrolló un componente llamado `AudioDirector`, encargado de la reproducción, secuenciación y control de volumen de los clips de voz. Este componente se implementó como un singleton persistente que asegura la existencia de un único punto de control de audio durante toda la ejecución de la aplicación. La implementación completa de este componente se presenta en el Anexo G. Código fuente del controlador general de voz en Unity.

El `AudioDirector` se construyó sobre la clase `AudioSource` de Unity, configurada con los siguientes parámetros:

- No espacialidad: spatialBlend = 0 para que las narraciones se reproduzcan en canal estéreo fijo.
- No repetición: loop = false para reproducir los audios una sola vez.
- Volumen dinámico: AudioListener.volume sincronizado con las preferencias del usuario (AppPrefs.LoadVolume()).

Este componente define tres métodos principales:

- `Play()` para reproducir un único audio, aplicando transiciones suaves.
- `PlaySequence()` para reproducir secuencias de clips consecutivos, útil para concatenar narraciones o efectos.
- `Stop()` para detener la reproducción con fundido gradual.

La lógica interna utiliza coroutines para aplicar fundidos lineales en el volumen, evitando cortes bruscos entre clips y permitiendo solapamiento controlado entre frases. Cada secuencia se gestiona mediante un identificador que permite abortar o reemplazar la reproducción si otro evento dispara una nueva orden de audio.

6.4.3. Integración con la interfaz y eventos del tour

El `AudioDirector` se comunica con la capa de presentación a través del `TourBinder` y los coordinadores de UI descritos en la Fase 2. Por ejemplo:

- Cuando el usuario ingresa a un área de interés, el `TourBinder` solicita al `AudioDirector` reproducir el clip de voz asociado a dicha área.
- Durante el onboarding o las pantallas de aviso, los coordinadores invocan `AudioDirector.Play()` con los clips correspondientes a cada mensaje contextual.
- Al cambiar de piso o finalizar el recorrido, se ejecutan secuencias de clips (`PlaySequence`) que acompañan los mensajes visuales de transición o cierre.

La gestión centralizada permite controlar la continuidad auditiva a lo largo del recorrido, garantizar que no existan reproducciones simultáneas de diferentes fuentes y aplicar ajustes globales de volumen o silenciamiento de manera uniforme.

6.4.4. Validación funcional del sistema de audio

Las pruebas de esta fase se enfocaron en los siguientes aspectos:

- **Verificación de audios generados**: confirmación de que todos los clips sintetizados se reproducen sin artefactos y con la entonación esperada.
- **Pruebas de integración en Unity**: comprobación de que los audios correctos se reproducen al ingresar en cada área, que los fundidos de entrada y salida funcionan, y que los eventos de navegación no interrumpen de forma abrupta las narraciones.
- **Validación multiplataforma**: ejecución tanto en el editor de Unity como en dispositivos iOS, asegurando que el sistema de audio se inicializa correctamente junto con el `AudioListener` y `AudioSource` creados dinámicamente.

Con esta fase se completó la arquitectura de la aplicación, consolidando la integración entre los tres subsistemas principales: navegación y posicionamiento, interfaz de usuario y audio narrativo.

6.5. Consideraciones éticas, privacidad y accesibilidad del sistema

El diseño del sistema de recorridos guiados consideró desde su planteamiento aspectos relacionados con la privacidad de los usuarios y la accesibilidad de la aplicación. Estas consideraciones forman parte de los criterios metodológicos que guiaron la implementación de la arquitectura descrita en este capítulo.

En cuanto a privacidad, la aplicación fue concebida para operar exclusivamente con datos de posicionamiento en tiempo real, sin almacenar de forma persistente información sensible sobre las trayectorias de los usuarios. Las coordenadas obtenidas desde los sensores UWB se utilizan únicamente para actualizar la posición del avatar virtual y el estado del recorrido, sin registrar historiales de movimiento ni asociar los datos a identificadores personales. Del mismo modo, el sistema de guía por voz no captura ni almacena muestras de voz del usuario; todos los audios reproducidos corresponden a narraciones previamente generadas y empacadas junto con la aplicación.

Adicionalmente, la solución no depende de servicios en la nube para el procesamiento en tiempo de ejecución, lo que reduce la exposición de datos de navegación a terceros. La comunicación con los sensores se limita al entorno local y el procesamiento principal se realiza en el dispositivo, acotando el uso de datos al contexto específico del recorrido.

Desde la perspectiva de accesibilidad, la aplicación incorpora una guía por voz que complementa la información visual presentada en pantalla, facilitando el uso del sistema por parte de personas que puedan tener dificultades para leer o interpretar el entorno tridimensional. La interfaz incluye controles para configurar preferencias de usuario que incluye: el volumen general del sistema y ajustes de tamaño de texto, gestionadas mediante el modelo de estado descrito en la sección 6.3.

La arquitectura modular adoptada deja abierta la posibilidad de incorporar, en versiones futuras, mejoras adicionales de accesibilidad, tales como modos de alto contraste, esquemas de color adaptados, o la presentación de subtítulos o transcripciones de los mensajes de voz. Estas extensiones permitirían ampliar el espectro de usuarios que pueden beneficiarse del sistema, manteniendo al mismo tiempo los principios de privacidad y uso responsable de la información de posicionamiento.

7. Resultados y Discusión

El resultado principal de este proyecto consiste en una aplicación móvil desarrollada en Unity que permite realizar recorridos guiados por las instalaciones del Centro de Innovación y Tecnología de la Universidad del Valle de Guatemala, incorporando navegación virtual y guía por voz.

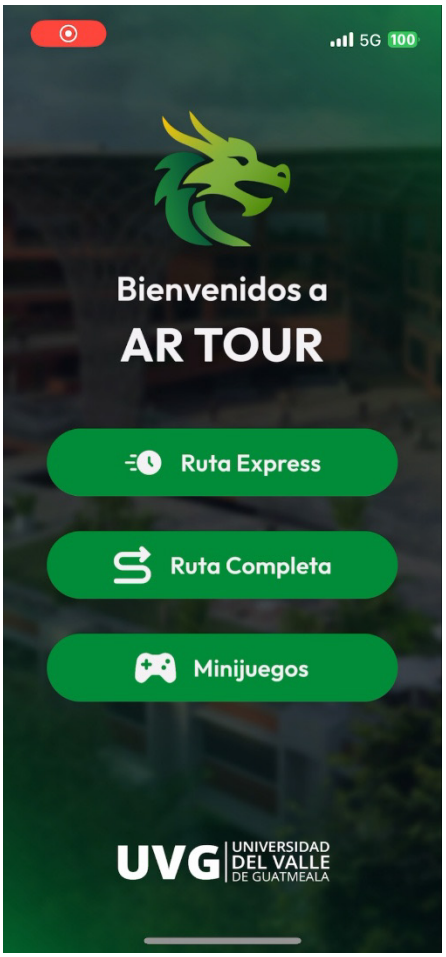

Figura 7.1: Pantalla principal de la aplicación
Fuente: Elaboración propia

En los apartados siguientes se presentan los resultados individuales de cada subsistema, acompañados de capturas, diagramas y evidencia audiovisual que demuestran su funcionamiento e integración final.

7.1. Funcionamiento del Sistema de Navegación

El sistema de navegación constituye el núcleo funcional del recorrido guiado. Su propósito es permitir el desplazamiento virtual del usuario dentro del entorno tridimensional del Centro de Innovación y Tecnología (CIT), mostrando en pantalla la ruta activa y la posición relativa en tiempo real.

La Figura 7.2 presenta una secuencia de tres capturas correspondientes a la ejecución de la aplicación en el nivel 6 del edificio CIT, frente al área de Administración de Maestrías. En ellas se observa la representación del usuario (esfera azul) desplazándose progresivamente a lo largo de la ruta generada por el sistema de path-finding. El indicador de progreso, mostrado en la parte inferior de la interfaz, refleja dinámicamente el porcentaje de recorrido completado y la distancia restante hacia el próximo punto de interés.

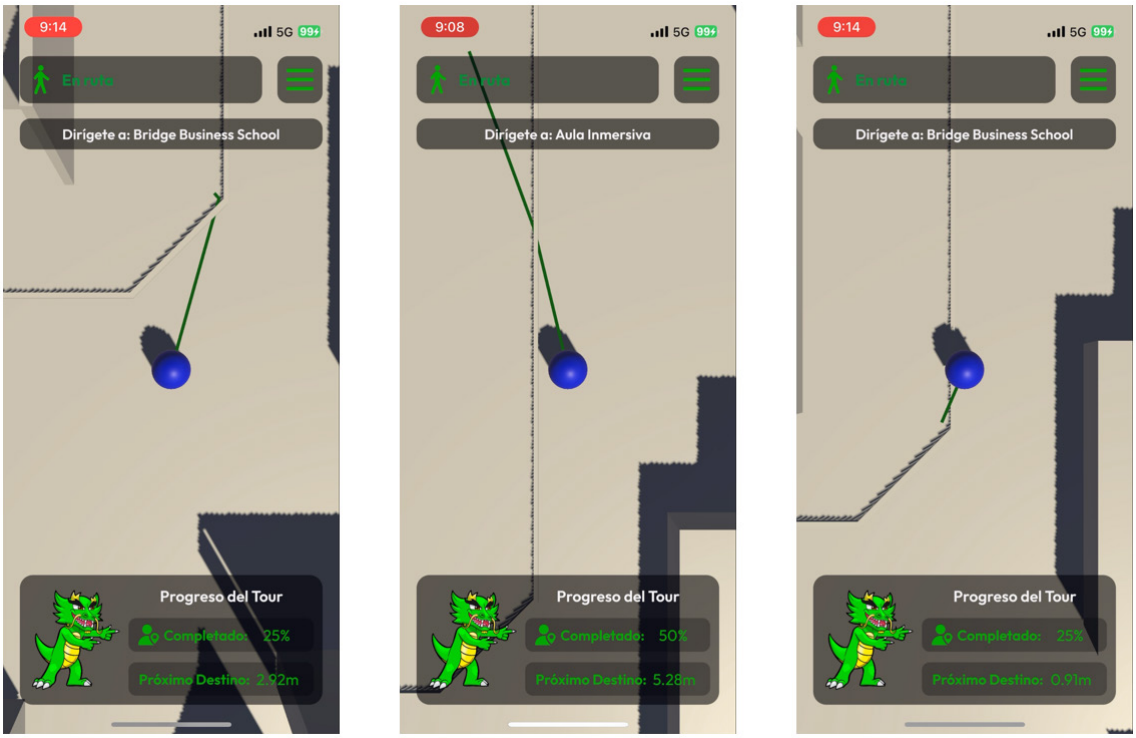

Figura 7.2: Ejecución del recorrido virtual sin modo AR.
Fuente: Elaboración propia

El desplazamiento visual se actualiza en función de los datos recibidos desde los beacons UWB instalados en el entorno físico, los cuales proporcionan la ubicación del usuario. No obstante, Unity no tiene acceso directo a las bibliotecas nativas de Estimote, fabricante de los beacons utilizados, ya que su SDK únicamente ofrece soporte para entornos nativos en iOS y Android. Por esta razón, el sistema implementa una capa intermedia de integración nativa, construida mediante la importación de funciones externas a través de `DllImport` como se muestra en el Anexo E. Código fuente del puente nativo Unity/iOS que actúa como puente entre el SDK móvil y la lógica de Unity.

Esta estrategia permitió desacoplar la lógica de posicionamiento del motor gráfico, facilitando la interoperabilidad entre plataformas. Además, aseguró que la navegación pudiera ejecutarse en el editor o en entornos simulados, sin depender de hardware físico durante el desarrollo.

Cada actualización de posición recibida se traduce en la modificación de las coordenadas del avatar virtual dentro de la escena, generando así un recorrido continuo sobre el modelo tridimensional del edificio. El sistema de rutas utiliza una malla de navegación (NavMesh) previamente configurada con áreas transitables, nodos de interés y modificadores de piso, garantizando que los desplazamientos mantengan coherencia con la geometría del entorno.

La Figura 7.3 muestra la vista interna del editor de Unity donde se observa la malla de navegación (NavMesh) utilizada para el cálculo de rutas, junto con las áreas de interés representadas mediante gizmos de color. Cada área corresponde a una sección del edificio y posee una definición propia (`AreaDefinition`), que almacena su posición, nombre y los mensajes de voz asociados. Los modificadores de piso permiten restringir o habilitar zonas navegables según la ubicación actual del usuario.

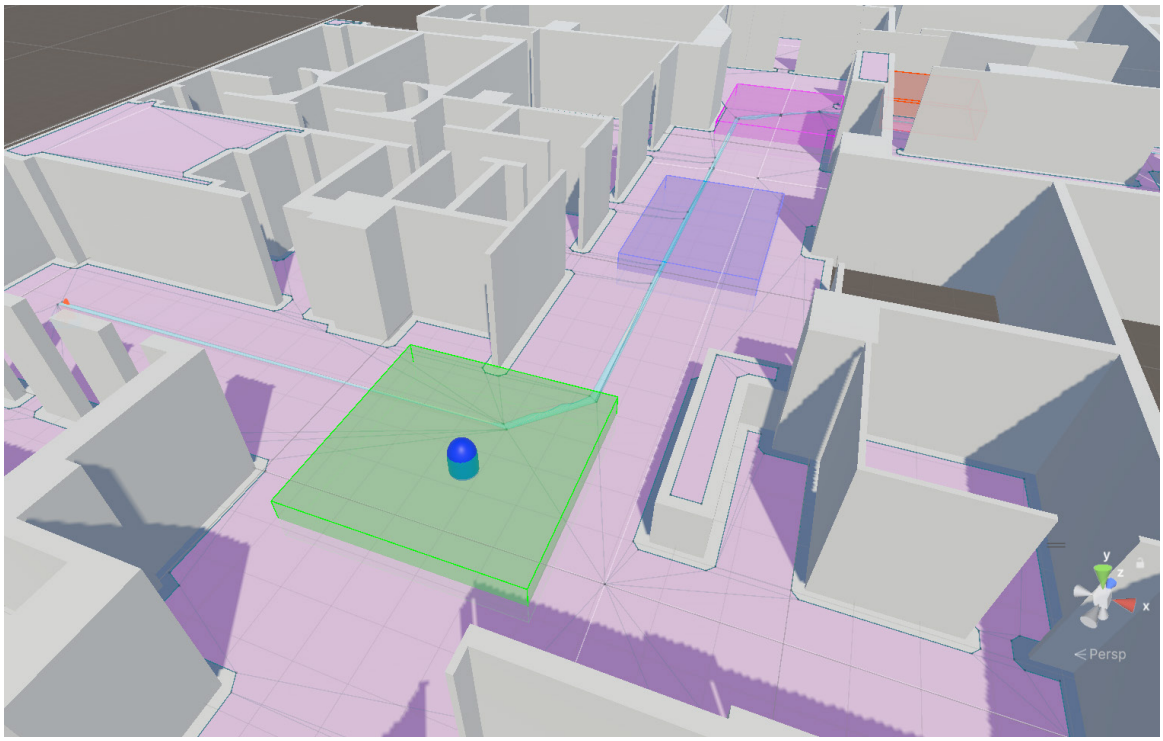

Figura 7.3:Configuración del NavMesh y áreas de interés en Unity.
Fuente: Elaboración propia

El conjunto de estos elementos posibilita la gestión dinámica de recorridos mediante el componente `TourRunner`, que coordina las actualizaciones del recorrido, el cambio de niveles y la sincronización con el sistema de audio. Este enfoque modular garantiza la consistencia visual y lógica del tour, aun cuando el origen de los datos de posición varíe entre simulaciones y dispositivos reales.

7.2. Funcionamiento de la Interfaz de Usuario

El sistema de interfaz de usuario fue diseñado bajo un modelo modular construido con UI Toolkit de Unity, empleando una arquitectura basada en el patrón de diseño MVVM para mantener la independencia entre la lógica del tour y los elementos visuales. Esta estructura permitió una gestión eficiente de pantallas, menús y preferencias, así como una comunicación fluida con los módulos de navegación y audio.

La Figura 7.4 muestra una secuencia de pantallas de la aplicación móvil si el usuario decide abrir las preferencias. La interfaz inicial ofrece acceso directo a los modos Ruta Express, Ruta Completa y Minijuegos, junto con la identidad visual institucional. Durante la ejecución del recorrido, el usuario visualiza una flecha direccional en el entorno de realidad aumentada, junto con un panel inferior que muestra el progreso del tour y la distancia al siguiente destino.

El menú lateral desplegable permite acceder a opciones de ayuda, reinicio del recorrido y configuración personalizada de volumen y tamaño de letra. Estos controles demuestran la interacción entre los componentes visuales y los datos del modelo de estado del tour, gestionados por el componente `TourViewModel`.

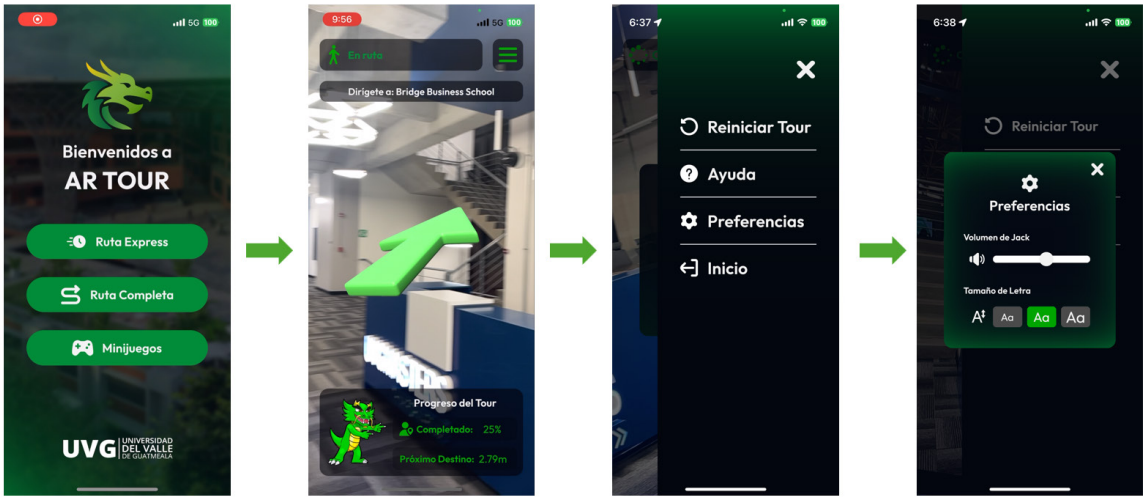

Figura 7.4: Secuencia de pantallas del sistema de interfaz de usuario
Fuente: Elaboración propia

Además de las vistas visibles al usuario final, se implementaron herramientas de desarrollo integradas en el editor de Unity con el propósito de facilitar la configuración y depuración del sistema. Estas herramientas, mostradas en las Figura 7.5 y Figura 7.6 , permiten manipular dinámicamente los parámetros de simulación, definir pisos, áreas y recorridos, así como conectar o desconectar fuentes de posicionamiento sin necesidad de modificar el código fuente.

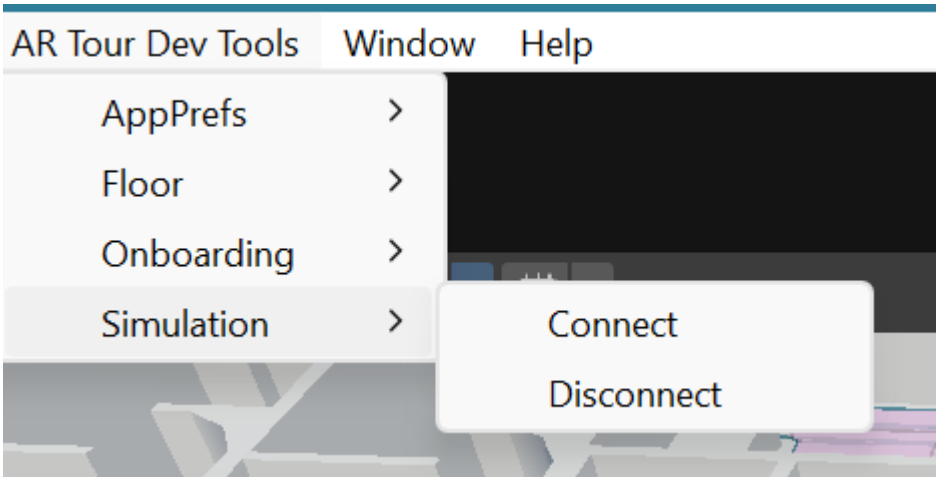

Figura 7.5: Menú de herramientas de desarrollo (AR Tour Dev Tools)
Fuente: Elaboración propia

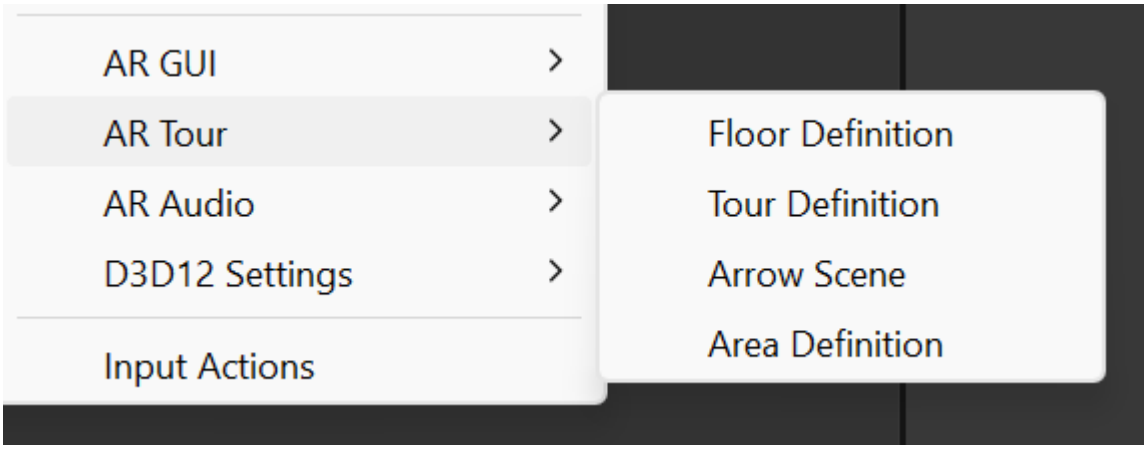

Figura 7.6: Menú de definiciones internas para pisos, áreas y escenas del recorrido.
Fuente: Elaboración propia

El uso de estas herramientas permitió mantener un flujo de iteración rápida durante el desarrollo, facilitando pruebas de navegación, audio y visualización sin depender de compilaciones o entornos externos. Asimismo, garantizó coherencia entre los módulos funcionales al centralizar las configuraciones de escena y definición de contenido en un mismo entorno de trabajo.

7.3. Funcionamiento del sistema de audio

El sistema de audio y voz evolucionó a lo largo del desarrollo del proyecto, pasando de un enfoque basado en SSML a uno basado en Prompts que es más flexible y eficiente sustentado en el nuevo modelo Gemini 2.5 Pro TTS de Google.

Durante las fases iniciales, se utilizaba SSML para definir la entonación, pausas y pronunciación de los mensajes de voz. Aunque el resultado era aceptable, este método requería una edición manual detallada de cada texto y un procesamiento más complejo para mantener naturalidad en las voces generadas. A continuación, un ejemplo de texto en formato SSML:

```
welcome_msg = """
    <speak>
        Hola, Soy Jack, y seré tu guía virtual.
        <break time="500ms"/>
        Te doy la bienvenida a la <say-as interpret-as="characters">UBG</say-as>
        <break time="600ms"/>
        ¡Vamos!
    </speak>
    """
```

Con la actualización de los servicios de Google a finales del año, se deprecó el modelo anterior (Polyglot/Studio) y se incorporó Gemini TTS, un modelo de lenguaje capaz de interpretar instrucciones naturales dentro del prompt de entrada. Esto permitió abandonar el uso de SSML, ya que la entonación y emoción pueden describirse directamente en lenguaje natural.

El nuevo formato simplificó notablemente el flujo de generación de audio: cada archivo JSON contiene un prompt y un text, donde el primero define el y el segundo el contenido a sintetizar. A continuación, una muestra de este nuevo formato adoptado:

```
{
  "prompt": "Habla con entusiasmo y amabilidad.",
  "text": " Hola, Soy Jack, y seré tu guía virtual. Te doy la bienvenida a la UVG"
}

```

El sistema de generación de audio fue implementado en Python utilizando la librería oficial `google-cloud-texttospeech`. El siguiente fragmento del módulo `gemini_tts.py` muestra la estructura principal del proceso de síntesis:

```
from google.cloud import texttospeech_v1beta1 as texttospeech

def synthesize(client, voice, audio_cfg, prompt: str, text: str) -> bytes:
    synthesis_input = texttospeech.SynthesisInput(
        text=text,
        prompt=prompt or None
    )
    resp = client.synthesize_speech(
        input=synthesis_input,
        voice=voice,
        audio_config=audio_cfg
    )
    return resp.audio_content

```

Este método recibe el texto y el prompt, genera el audio mediante la API de Google y devuelve los datos `PCM` para su almacenamiento como archivo `.wav`. La ejecución del proceso completo se automatizó en el script `main.py`, capaz de recorrer carpetas de entrada, procesar múltiples JSON y exportar los audios de forma masiva.

7.4. Integración Final

Una vez implementados los tres subsistemas navegación, interfaz de usuario y audio y voz, se realizó la integración final dentro del entorno de Unity. Esta etapa tuvo como objetivo consolidar la comunicación entre componentes y garantizar que los diferentes módulos operaran de forma coherente en tiempo real.

La aplicación resultante combina el posicionamiento mediante sensores UWB, el cálculo de rutas dinámicas, la guía por voz y una interfaz adaptable que asiste al usuario durante todo el recorrido. La arquitectura modular descrita en la metodología permitió que los distintos sistemas interactuaran sin dependencias directas.

En la Figura 7.7 se muestra una secuencia de pantallas que ejemplifica el flujo principal de uso: desde la pantalla de inicio, la bienvenida del asistente virtual Jack, el recorrido con realidad aumentada y el elemento visual que le muestra al usuario la información de un área en específico.

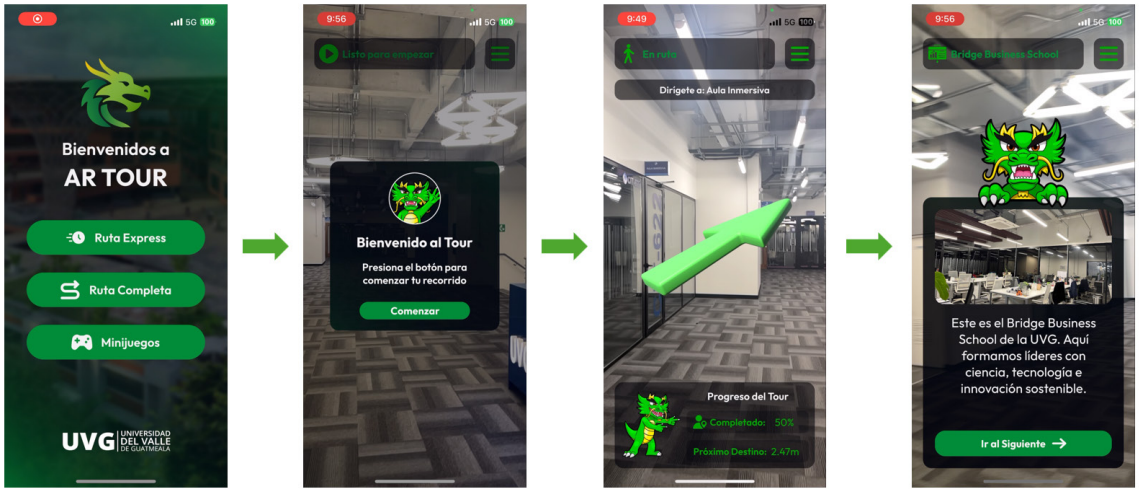

Figura 7.7: Secuencia del flujo principal de la aplicación AR Tour
Fuente: Elaboración propia

Durante las pruebas internas realizadas en el Centro de Innovación y Tecnología (CIT) de la Universidad del Valle de Guatemala, la aplicación demostró la correcta integración de los sistemas. Los desplazamientos del usuario fueron reflejados en tiempo real en la vista tridimensional del entorno, el sistema de voz respondió conforme al progreso del recorrido y los elementos de interfaz adaptaron su estado según las acciones del usuario.

Como complemento visual, en el Anexo A se incluye un video demostrativo del funcionamiento de la aplicación. Evidenciando la operación simultanea de los distintos subsistemas de la aplicación.

Para fines de transparencia y reproducibilidad, el código fuente completo del proyecto, incluyendo clases auxiliares, herramientas internas de editor y configuraciones que no forman parte de los Anexos E–H, se encuentra disponible como material suplementario en el repositorio del proyecto presentado en el Anexo D. Repositorio de la aplicación AR Tour.

7.5. Amenazas a la validez y limitaciones técnicas

La validación del sistema estuvo condicionada por limitaciones técnicas asociadas al ecosistema de posicionamiento UWB utilizado. En particular, se identificó una dependencia crítica del SDK de Estimote, responsable de gestionar la comunicación con los sensores UWB instalados en el Centro de Innovación y Tecnología (CIT).

Durante las pruebas se observó que, al incrementar el número de sensores UWB activos gestionados simultáneamente por el SDK, se producía un fallo de concurrencia interno que derivaba en un error de hilos (thread error). Este error no solo interrumpía la comunicación con los sensores, sino que provocaba un cierre inesperado de la aplicación (crash). En la práctica, se determinó que configuraciones con más de cinco sensores activos generaban este comportamiento, por lo que el entorno de pruebas se limitó estrictamente a ese número máximo para evitar inestabilidad.

Como consecuencia, la validación se realizó en dos entornos controlados. Primero, se ejecutaron pruebas extensivas en el editor de Unity utilizando un sistema de posicionamiento simulado (`KeyboardPositioning`), lo que permitió reproducir trayectorias de usuario sin depender de hardware físico. Posteriormente, se realizaron pruebas en el nivel 6 del edificio CIT, tanto en un subconjunto acotado de salones como en un montaje con cinco sensores UWB, respetando el límite de estabilidad impuesto por el SDK. En estas condiciones fue posible validar la coherencia entre navegación, interfaz y audio, así como el flujo general del recorrido.

Desde la perspectiva de validez, la principal amenaza identificada se relaciona con la inestabilidad del SDK de Estimote bajo condiciones de alta concurrencia, lo cual limita la capacidad de extrapolar los resultados hacia configuraciones con un mayor número de sensores. Aunque la lógica interna del sistema (navegación, guía por voz, interfaz y coordinación) demostró estabilidad y coherencia interna, su funcionamiento en entornos con mayor infraestructura UWB depende completamente de mejoras futuras del SDK o de ajustes internos en el módulo de trilateración.

No obstante, esta amenaza se mitigó parcialmente mediante una capa de abstracción entre Unity y el módulo nativo que usa de forma directa el SDK de Estimote (`UWBPositioning` y `MovementAgent`), lo cual permite incorporar ajustes sin modificar la lógica central del proyecto, reduciendo el impacto de esta limitación en evoluciones posteriores.

8. Conclusiones

- Se desarrolló un sistema de navegación y guía por voz integrado en una aplicación móvil, que combina de manera funcional los componentes de posicionamiento, cálculo de rutas, interfaz gráfica y narración por voz dentro de un entorno unificado en Unity. El sistema demostró su operatividad y coherencia interna al simular recorridos utilizando datos de posicionamiento controlados, cumpliendo con el objetivo principal del proyecto.

- Se diseño e implementó un sistema de navegación basado en NavMesh y algoritmos de path-finding que permite calcular rutas óptimas entre puntos de interés dentro del edificio CIT. Mediante el ajuste de costos de áreas y modificadores de piso se obtuvieron trayectorias centradas y coherentes con la geometría real del edificio.

- Se integró la localización en tiempo real con el sistema de navegación por medio de un puente nativo entre iOS y Unity. Durante las pruebas de campo se identificó una limitación técnica asociada al SDK de los sensores UWB Estimote, el cual presenta fallos de concurrencia al superar cinco dispositivos activos, lo que impidió validar un recorrido físico completo. No obstante, el sistema respondió de forma consistente al recibir datos de posición simulada y de un entorno controlado, lo que confirma la solidez del diseño y la independencia funcional de la capa de navegación.

- Se desarrolló un sistema de generación de audios de voz sintética que evolucionó desde un enfoque inicial basado en SSML hacia el uso de LLM como Gemini 2.5 Pro TTS, configurados mediante prompts en lenguaje natural. Migración que permitió producir narraciones más naturales y consistentes, reduciendo el esfuerzo de edición manual y manteniendo el enfoque del objetivo planteado de contar con una guía por voz sintética flexible y configurable.

- Se implementó la reproducción sincronizada de mensajes de voz según la posición del usuario, utilizando activadores espaciales asociados a áreas de interés y un componente centralizado de control de audio. La coordinación entre los distintos módulos de la aplicación permitió comunicar información contextual en los momentos adecuados del recorrido.

- El proyecto contribuye de forma directa a resolver el problema de orientación dentro del edificio CIT, proponiendo una herramienta de guía virtual que combina voz, navegación visual y elementos de realidad aumentada. Su arquitectura modular y enfoque data-driven sientan bases técnicas sólidas para futuras extensiones y aplicaciones en contextos educativos, museográficos o turísticos.

9. Recomendaciones

- Contar con un entorno de desarrollo macOS y iOS desde el inicio del proyecto. Dado que el sistema fue diseñado para ejecutarse en dispositivos iOS. Disponer de un entorno macOS es indispensable para generar builds, realizar firmas de aplicación y ejecutar pruebas directamente sobre hardware real. La falta de este entorno durante las etapas iniciales limitó el flujo continuo de compilación, retrasando la validación final del sistema.

- Evaluar alternativas de posicionamiento multiplataforma, incluyendo soluciones BLE o UWB de código abierto, que permitan una integración directa en Unity sin depender de SDKs propietarios limitados a un solo sistema operativo.

- Evaluar la implementación paralela para Android, en caso de no contar con infraestructura iOS, con el fin de disponer de una ruta de prueba funcional independiente y mantener continuidad en el desarrollo multiplataforma.

- Realizar pruebas de usuario estructuradas, enfocadas en evaluar la precisión de navegación, comprensión de instrucciones auditivas y usabilidad de la interfaz. Esto permitirá obtener métricas objetivas y ajustar el diseño a distintos perfiles de visitantes.

- Optimizar el manejo de casos límite dentro de la aplicación, como la pérdida de señal, errores de lectura o interrupciones del recorrido, mediante rutinas de reconexión o recalibración automática.

10. Referencias

Akerberg, M., Espinosa, A., & Santiago, F. (2015). *La enseñanza de la pronunciación* (1a ed.). Universidad Nacional Autónoma de México. https://publicaciones.enallt.unam.mx/adjuntos/La%20ense%C3%B1anza%20de%20la%20pronunciaci%C3%B3n_completo.pdf

Angry Shark Team. (2025). *Unity UI Toolkit vs UGUI: 2025 Developer Guide | Angry Shark Studio Blog*. Angry Shark Studio Blog. https://www.angry-shark-studio.com/blog/unity-ui-toolkit-vs-ugui-2025-guide/

Beyza. (2023). *Observer Pattern, Delegates & Events*. Medium. https://medium.com/@beyzaunity/observer-pattern-delegates-events-unity-series-72c5b39856cb

Codecademy. (s/f). *MVC Architecture Explained: Model, View, Controller*. Skillsoft. Recuperado el 2 de noviembre de 2025, de https://www.codecademy.com/article/mvc-architecture-model-view-controller

DataCamp. (2024). *El Algoritmo A*: Guía completa*. DataCamp. https://www.datacamp.com/es/tutorial/a-star-algorithm

French, J. (2021). *Events & Delegates in Unity*. GameDevBeginner. https://gamedevbeginner.com/events-and-delegates-in-unity/

García, F. (2024). *Qué es el data-driven design: el diseño basado en datos*. Arsys. https://www.arsys.es/blog/que-es-el-data-driven-design-el-diseno-basado-en-datos

Gonzalez, P. (2024). *Evaluación y Selección de Tecnologías para sistemas IPS para el CIT* [Tesis de licenciatura]. Universidad del Valle de Guatemala.

Google. (2025a). *Best practices | Cloud Speech-to-Text V2 documentation | Google Cloud*. Google Cloud. https://cloud.google.com/speech-to-text/v2/docs/best-practices

Google. (2025b). *Gemini-TTS | Text-to-Speech | Google Cloud Documentation*. Google Cloud. https://docs.cloud.google.com/text-to-speech/docs/gemini-tts

Google. (2025c). *Introduction to audio encoding for Speech-to-Text | Google Cloud Documentation*. Google Cloud. https://docs.cloud.google.com/speech-to-text/docs/encoding

Google. (2025d). *Speech generation (text-to-speech) | Gemini API | Google AI for Developers*. Gemini API. https://ai.google.dev/gemini-api/docs/speech-generation

Hernández, J. (2024). *Desarrollo de UX/UI en Aplicación de Recorridos Virtuales con Unity* [Tesis de licenciatura]. Universidad del Valle de Guatemala.

Ionos. (2023). *Pathfinding: búsqueda de rutas en informática*. Ionos Digital Guide. https://www.ionos.com/es-us/digitalguide/online-marketing/analisis-web/pathfinding-busqueda-de-rutas-en-informatica/

Joffrineau, M. (2022). *UI Toolkit at runtime: Get the breakdown*. Unity. https://unity.com/blog/engine-platform/ui-toolkit-at-runtime-get-the-breakdown

Kartik. (2025). *Factory method Design Pattern - GeeksforGeeks*. Geeks for Geeks. https://www.geeksforgeeks.org/system-design/factory-method-for-designing-pattern/

Kharitonov, E., Vincent, D., Borsos, Z., Marinier, R., Girgin, S., Pietquin, O., Sharifi, M., Tagliasacchi, M., & Zeghidour, N. (2023). Speak, Read and Prompt: High-Fidelity Text-to-Speech with Minimal Supervision. *Transactions of the Association for Computational Linguistics*, *11*, 1703–1718. https://doi.org/10.1162/tacl_a_00618

Kumar, P. (2024). *MVVM & MVC. 1. Separation of Concerns*. Medium. https://medium.com/@pradeepgpre/mvvm-mvc-7e36f508018c

Lafritz, J. (2022). *The Observer Pattern In Unity.* DevGenius. https://blog.devgenius.io/the-observer-pattern-in-unity-4ee8e12100aa

Lin, W., Oriz, E., & Krogh-Jacobsen, T. K.-J. (2025). *UI Toolkit for advanced Unity developers (Unity 6 edition)* (6a ed.). Unity Technologies. https://unity.com/resources/scalable-performant-ui-uitoolkit-unity-6

Microsoft. (2024). *Modelo-Vista-Modelo de vista*. Microsoft Learn. https://learn.microsoft.com/es-es/dotnet/architecture/maui/mvvm

Microsoft. (2025). *Voice and sound with Speech Synthesis Markup Language (SSML) - Speech service - Azure AI services | Microsoft Learn*. Microsoft Learn. https://learn.microsoft.com/en-us/azure/ai-services/speech-service/speech-synthesis-markup-voice

Mombiela, R. (2024). *Desarrollo de Modelado 3D y Algoritmo de Pathfinding en Aplicación de Recorridos Virtuales con Unity* [Tesis de licenciatura]. Universidad del Valle de Guatemala.

Nordon, S. (2024). *Unity Architecture: Scriptable Object Pattern*. Medium. https://medium.com/@simon.nordon/unity-architecture-scriptable-object-pattern-0a6c25b2d741

Santos, G. (2024). *Desarrollo de interfaz gráfica para aplicación de recorridos virtuales en la plataforma iOS* [Tesis de licenciatura]. Universidad del Valle de Guatemala.

Serrano, H. (2016). *Design Patterns in Game Engine Development — Harold Serrano - Game Engine Developer*. HaroldSerrano Blog. https://www.haroldserrano.com/blog/design-patterns-in-game-engine-development?rq=observer

Song, H. (2024). *The Most Suitable Paradigm in Unity: A Comparative Analysis of Object-Oriented, Component-Based, and Data-Oriented Approaches*. Medium. https://medium.com/@olgaphila40/the-most-suitable-paradigm-in-unity-a-comparative-analysis-of-object-oriented-component-based-499812974912

Svetz, A. (2021). *Mediator Pattern*. Refactoring Guru. https://refactoring.guru/design-patterns/mediator

Unity Technologies. (2023). *Separate Game Data and Logic with ScriptableObjects*. Unity Documentation. https://unity.com/how-to/separate-game-data-logic-scriptable-objects

Unity Technologies. (2024). *Navigation Areas and Costs | AI Navigation | 2.0.9*. Unity Manual. https://docs.unity3d.com/Packages/com.unity.ai.navigation@2.0/manual/AreasAndCosts.html

Unity Technologies. (2025a). *Inner Workings of the Navigation System | AI Navigation | 2.0.9*. Unity Manual. https://docs.unity3d.com/Packages/com.unity.ai.navigation@2.0/manual/NavInnerWorkings.html

Unity Technologies. (2025b). *Unity - Manual: Audio Listener*. Unity Documentation. https://docs.unity3d.com/6000.2/Documentation/Manual/class-AudioListener.html

Unity Technologies. (2025c). *Unity - Manual: Best practices for managing elements*. Unity Documentation. https://docs.unity3d.com/6000.2/Documentation/Manual/UIE-best-practices-for-managing-elements.html

Unity Technologies. (2025d). *Unity - Manual: Click events*. Unity Documentation. https://docs.unity3d.com/6000.2/Documentation/Manual/UIE-Click-Events.html

Unity Technologies. (2025e). *Unity - Manual: Comparison of UI systems in Unity*. Unity Documentation. https://docs.unity3d.com/6000.2/Documentation/Manual/UI-system-compare.html

Unity Technologies. (2025f). *Unity - Manual: Create a native plug-in for iOS*. Unity Documentation. https://docs.unity3d.com/6000.2/Documentation/Manual/ios-native-plugin-create.html

Unity Technologies. (2025g). *Unity - Manual: Draw and configure a line in 3D space*. Unity Documentation. https://docs.unity3d.com/Manual/draw-configure-line-3d-space.html

Unity Technologies. (2025h). *Unity - Manual: Event function execution order*. Unity Documentation. https://docs.unity3d.com/6000.2/Documentation/Manual/execution-order.html

Unity Technologies. (2025i). *Unity - Manual: Find visual elements with UQuery*. Unity Documentation. https://docs.unity3d.com/6000.2/Documentation/Manual/UIE-UQuery.html

Unity Technologies. (2025j). *Unity - Manual: Instantiate UXML from C# scripts*. Unity Documentation. https://docs.unity3d.com/6000.2/Documentation/Manual/UIE-LoadingUXMLcsharp.html

Unity Technologies. (2025k). *Unity - Manual: Native plug-ins*. Unity Documentation. https://docs.unity3d.com/6000.2/Documentation/Manual/plug-ins-native.html

Unity Technologies. (2025l). *Unity - Manual: Position element with the layout engine*. Unity Documentation. https://docs.unity3d.com/6000.2/Documentation/Manual/UIE-LayoutEngine.html

Unity Technologies. (2025m). *Unity - Manual: The GameObject class*. Unity Documentation. https://docs.unity3d.com/Manual/class-GameObject.html

Unity Technologies. (2025n). *Unity - Manual: Transition events*. Unity Documentation. https://docs.unity3d.com/6000.2/Documentation/Manual/UIE-Transition-Events.html

Unity Technologies. (2025o). *Unity - Scripting API: AI.NavMesh.SamplePosition*. Unity Documentation. https://docs.unity3d.com/6000.2/Documentation/ScriptReference/AI.NavMesh.SamplePosition.html

Unity Technologies. (2025p). *Unity - Scripting API: AudioSource*. Unity Documentation. https://docs.unity3d.com/6000.2/Documentation/ScriptReference/AudioSource.html

Unity Technologies. (2025q). *Unity - Scripting API: MonoBehaviour.Awake()*. Unity Documentation. https://docs.unity3d.com/ScriptReference/MonoBehaviour.Awake.html

Unity Technologies. (2025r). *Unity - Scripting API: MonoBehaviour.OnDestroy()*. Unity Documentation. https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnDestroy.html

Unity Technologies. (2025s). *Unity - Scripting API: MonoBehaviour.Start()*. Unity Documentation. https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html

Unity Technologies. (2025t). *Unity - Scripting API: MonoBehaviour.Update()*. Unity Documentation. https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html

Zhang, C. (2025). *Data-Driven Design: Leveraging Lessons from Game Development in Everyday Software*. Methodox. https://dev.to/methodox/data-driven-design-leveraging-lessons-from-game-development-in-everyday-software-5512

Zilliz. (2025). *From Text to Speech: A Deep Dive into TTS Technologies*. Medium. https://medium.com/@zilliz_learn/from-text-to-speech-a-deep-dive-into-tts-technologies-18ea409f20e8

11. Anexos

Anexo A. Video de demostración de la aplicación

Enlace: AR Tour Demo.mp4

(Disponible en OneDrive institucional de la Universidad del Valle de Guatemala)

Anexo B. Organización del proyecto en GitHub.

Repositorio central que agrupa los módulos desarrollados por los distintos integrantes del equipo y documentación activa.

Enlace: https://github.com/AR-Tour-UVG

Anexo C. Repositorio del sistema de generación de voz

Incluye los scripts en Python utilizados para la generación de audios con Google Text-to-Speech y el formato de Prompts.

Enlace: https://github.com/AR-Tour-UVG/Tour-TTS.git

Anexo D. Repositorio de la aplicación AR Tour

Contiene el entorno de desarrollo en Unity, los scripts principales y la lógica del sistema de navegación. Utilizado principalmente para control de versiones y sincronización con Unity DevOps:

Enlace: https://github.com/AR-Tour-UVG/AR-Tour-V2.git

Anexo E. Código fuente del puente nativo Unity/iOS

```
using System;
using System.Runtime.InteropServices;
using UnityEngine;

/// <summary>
/// Struct to hold coordinate data from UWB system.
/// </summary>
[Serializable]
public struct Coordinate
{
    public float x; // UWB x coordinate
    public float y; // UWB y coordinate
}

/// <summary>
/// UWBLocator provides methods to interact with the UWB positioning system.
/// It allows setting anchor maps and retrieving real-time position data.
/// </summary>
/// <remarks>
/// This class uses platform-specific native plugins for iOS devices.
/// On unsupported platforms, it provides stub implementations.
/// </remarks>
public static class UWBLocator
{
    // State to track initialization and current anchor map
    public static bool IsInitialized => isInitialized;
    private static bool isInitialized = false;
    private static string currentAnchorMap;

#if UNITY_IOS && !UNITY_EDITOR
    // Platform-specific native method bindings (iOS)
    [DllImport("__Internal")]
    private static extern IntPtr getCoords();

    [DllImport("__Internal")]
    private static extern void freeCString(IntPtr ptr);

    [DllImport("__Internal")]
    private static extern void setAnchorMap(string jsonUtf8);

    [DllImport("__Internal")]
    private static extern void start();
#elif UNITY_EDITOR && !UNITY_IOS
    // Stub implementations for unsupported platforms
    private static bool hasWarned = false;

    private static IntPtr getCoords() => IntPtr.Zero;

    private static void freeCString(IntPtr ptr) { }

    private static void setAnchorMap(string jsonUtf8) { }

    private static void start() { }
#else
    // Fallback stubs for all other platforms
    private static IntPtr getCoords() => IntPtr.Zero;

    private static void freeCString(IntPtr ptr) { }

    private static void setAnchorMap(string jsonUtf8) { }

    private static void start() { }
#endif

    /// <summary>
    /// Attempts to get the current position from the UWB system.
    /// </summary>
    /// <param name="position"></param>
    /// <returns> True if the position was successfully retrieved; otherwise, false. </returns>
```

```
70.     public static bool TryGetPosition(out Vector3 position)
71.     {
72.         position = default;
73.
74. #if UNITY_EDITOR && !UNITY_IOS
75.         // Warn once in the editor about unsupported platform
76.         if (!hasWarned)
77.         {
78.             Debug.LogWarning(
79.                 "[UWBLocator] Real time positioning is supported only on iOS device builds."
80.             );
81.             hasWarned = true;
82.         }
83.         return false;
84.
85. #elif UNITY_IOS && !UNITY_EDITOR
86.         // Call the native plugin to get coordinates
87.         IntPtr coordsPtr = getCoords();
88.
89.         // Check for null pointer
90.         if (coordsPtr == IntPtr.Zero)
91.         {
92.             Debug.LogWarning("[UWBLocator] Received null pointer for coordinates from UWB plugin.");
93.             return false;
94.         }
95.
96.         try
97.         {
98.             // Convert the C string to a managed string and parse JSON
99.             string json = Marshal.PtrToStringAnsi(coordsPtr);
100.            Debug.Log($"[UWBLocator] JSON from plugin: {json}");
101.
102.            // Validate JSON content
103.            if (string.IsNullOrEmpty(json) || json == "{}" || json.Contains("null"))
104.            {
105.                Debug.LogWarning(
106.                    "[UWBLocator] Received invalid JSON or null coordinates from UWB plugin."
107.                );
108.                return false;
109.            }
110.
111.            // Deserialize JSON to Coordinate struct
112.            var uwbPosition = JsonUtility.FromJson<Coordinate>(json);
113.            Debug.Log($"[UWBLocator] Parsed UWB Position - x: {uwbPosition.x}, y: {uwbPosition.y}");
114.            position = new Vector3(uwbPosition.x, 0f, uwbPosition.y);
115.            return true;
116.        }
117.        catch (Exception ex)
118.        {
119.            Debug.LogError($"[UWBLocator] Failed to parse UWB position JSON: {ex.Message}");
120.            return false;
121.        }
122.        finally
123.        {
124.            // Free the allocated C string to prevent memory leaks
125.            Debug.Log($"[UWBLocator] Freeing allocated string for coordinates.");
126.            freeCString(coordsPtr);
127.        }
128. #else
129.        Debug.LogWarning(
130.            "[UWBLocator] Real time positioning is supported only on iOS device builds."
131.        );
132.        return false;
133. #endif
134.    }
135.
136.    /// <summary>
137.    /// Sets the anchor map for the UWB system.
138.    /// </summary>
139.    /// <param name="anchorMap">The anchor map in JSON format.</param>
140.    /// <remarks>
```

```
141.     /// ThE Anchor Map should be a JSON string representing the beacons coordinates in real world
coordinates.
142.     /// </remarks>
143.     public static void SetAnchorMap(string anchorMap)
144.     {
145.         // Validate input
146.         if (string.IsNullOrWhiteSpace(anchorMap))
147.         {
148.             Debug.LogWarning("[UWBLocator] SetAnchorMap: Anchor map is null or empty.");
149.             return;
150.         }
151.         if (currentAnchorMap == anchorMap)
152.         {
153.             Debug.Log("[UWBLocator] SetAnchorMap: same Anchor map, no change.");
154.             return;
155.         }
156.
157. #if UNITY_IOS && !UNITY_EDITOR
158.         try
159.         {
160.             // Call the native plugin to set the anchor map
161.             setAnchorMap(anchorMap);
162.             currentAnchorMap = anchorMap;
163.             Debug.Log($"[UWBLocator] SetAnchorMap: Anchor map set to {anchorMap}.");
164.         }
165.         catch (Exception ex)
166.         {
167.             Debug.LogError($"[UWBLocator] SetAnchorMap: Failed setting anchor map: {ex.Message}");
168.         }
169. #else
170.         // On unsupported platforms, just store the anchor map
171.         currentAnchorMap = anchorMap;
172.         Debug.Log("[UWBLocator] Not supported on this platform.");
173. #endif
174.     }
175.
176.     /// <summary>
177.     /// Starts the UWB locator system.
178.     /// </summary>
179.     public static void Start()
180.     {
181. #if UNITY_IOS && !UNITY_EDITOR
182.         try
183.         {
184.             // Call the native plugin to start the UWB system
185.             start();
186.             Debug.Log("[UWBLocator] Native Plugin started.");
187.             isInitialized = true;
188.         }
189.         catch (Exception ex)
190.         {
191.             // If starting fails, log the error and set isInitialized to false
192.             isInitialized = false;
193.             Debug.LogError($"[UWBLocator] Start failed: {ex.Message}");
194.         }
195. #endif
196.     }
197. }
198.
```

Anexo F. Código fuente del Path-Finding

```
 1. using System;
 2. using UnityEngine;
 3. using UnityEngine.AI;
 4.
 5. /// <summary>
 6. /// Provides NavMesh-based path computation between the player and a target area,
 7. /// tracking distance and raising events when the path is updated.
 8. /// </summary>
 9. [DisallowMultipleComponent]
10. public class PathProvider : MonoBehaviour
11. {
12.     [Header("Settings")]
13.     [SerializeField]
14.     private float sampleRadius = 2f;
15.
16.     [SerializeField]
17.     private float recomputeThreshold = 0.01f;
18.
19.     [Header("Optional")]
20.     [Tooltip("Assign an Area GameObject in the scene to start with (uses its BoxCollider center).")]
21.     [SerializeField]
22.     private GameObject initialTarget;
23.
24.     [SerializeField]
25.     private Transform playerPosition;
26.
27.     // Path related
28.     public bool Paused;
29.     public NavMeshPath CurrentPath { get; private set; }
30.     public float CurrentDistance { get; private set; }
31.
32.     // Positioning related values
33.     private Vector3 _lastPlayerPos = Vector3.positiveInfinity;
34.     private Vector3 _lastTargetPos = Vector3.positiveInfinity;
35.     private bool _hasTargetPoint = false;
36.     private Vector3 _targetPoint;
37.
38.     /// <summary>
39.     /// Event raised whenever a path is updated. The argument is the new path, or null if no path is
available.
40.     /// </summary>
41.     public event Action<NavMeshPath> OnPathUpdated;
42.
43.     /// <summary>
44.     /// Initializes the NavMeshPath and attempts to locate the player transform if not explicitly
assigned.
45.     /// </summary>
46.     private void Awake()
47.     {
48.         // Create new path
49.         CurrentPath = new NavMeshPath();
50.
51.         // Validate user position
52.         if (playerPosition == null)
53.         {
54.             // Find the MovementAgent Object in the scene
55.             var movement = FindFirstObjectByType<MovementAgent>(FindObjectsInactive.Include);
56.             if (movement != null)
57.             {
58.                 // Update the user position to the transform of the MovementAgent
59.                 playerPosition = movement.transform;
60.             }
61.             else
62.             {
63.                 // If no MovementAgent found, fallback to player tagged GameObject
64.                 var tagged = GameObject.FindWithTag("Player");
```

```
                if (tagged != null)
                    // Update the user position to the transform of the GameObject with the "Player"
tag
                    playerPosition = tagged.transform;
            }

            // Check if playerPosition was successfully assigned
            if (playerPosition == null)
                Debug.LogWarning("[PathProvider] No player position found. Distance will stay 0.");
        }
    }

    /// <summary>
    /// Optionally sets an initial target if one has been configured.
    /// </summary>
    private void Start()
    {
        if (initialTarget)
            SetTarget(initialTarget);
    }

    /// <summary>
    /// Monitors player and target movement, recomputing the path when either moves
    /// beyond the configured threshold and raising <see cref="OnPathUpdated"/>.
    /// </summary>
    private void Update()
    {
        // Check if the MovementAgent is paused or has no target
        if (Paused || !_hasTargetPoint)
            return;

        // Compute coordinates of both user and target
        Vector3 p = playerPosition ? playerPosition.position : transform.position;
        Vector3 t = _targetPoint;

        // Apply recompute threshold
        float threshSq = recomputeThreshold * recomputeThreshold;
        // Check if both player and target positions are inside the threshold
        if (
            (p - _lastPlayerPos).sqrMagnitude < threshSq
            && (t - _lastTargetPos).sqrMagnitude < threshSq
        )
            return;

        // Update last positions
        _lastPlayerPos = p;
        _lastTargetPos = t;

        // Compute path from user to target
        if (TryComputePath(p, t, out var path))
        {
            // Update path
            CurrentPath = path;
            CurrentDistance = ComputePathDistance(CurrentPath);
            OnPathUpdated?.Invoke(CurrentPath);
        }
        else
        {
            // Update path as null value
            CurrentPath = null;
            CurrentDistance = 0f;
            OnPathUpdated?.Invoke(null);
        }
    }

    /// <summary>
    /// Sets the target area GameObject, using its <see cref="AreaInstance"/> or <see
cref="BoxCollider"/>
    /// as the destination point, and triggers a path recomputation.
    /// </summary>
    /// <param name="areaGO">The GameObject representing the target area.</param>
    public void SetTarget(GameObject areaGO)
```

```
135.     {
136.         if (!areaGO)
137.             return;
138.
139.         var ai = areaGO.GetComponent<AreaInstance>();
140.         var box = ai ? ai.NavTarget : areaGO.GetComponent<BoxCollider>();
141.         if (!box)
142.         {
143.             Debug.LogWarning("[PathProvider] Target GameObject has no AreaInstance/BoxCollider.");
144.             return;
145.         }
146.
147.         _targetPoint = box.transform.TransformPoint(box.center);
148.         _hasTargetPoint = true;
149.         ForceRecompute();
150.     }
151.
152.     /// <summary>
153.     /// Clears the current target and resets path and distance information.
154.     /// </summary>
155.     public void ClearTarget()
156.     {
157.         // Reset current values
158.         _hasTargetPoint = false;
159.         CurrentPath = null;
160.         CurrentDistance = 0f;
161.         OnPathUpdated?.Invoke(null);
162.     }
163.
164.     /// <summary>
165.     /// Forces the next Update to recompute the path, regardless of threshold.
166.     /// </summary>
167.     public void ForceRecompute()
168.     {
169.         // Set last known positions to infinite value
170.         _lastPlayerPos = Vector3.positiveInfinity;
171.         _lastTargetPos = Vector3.positiveInfinity;
172.     }
173.
174.     /// <summary>
175.     /// Attempts to compute a NavMesh path between two world positions, sampling
176.     /// both endpoints onto the NavMesh.
177.     /// </summary>
178.     /// <param name="from">The starting world position.</param>
179.     /// <param name="to">The target world position.</param>
180.     /// <param name="path">The resulting computed path, if successful.</param>
181.     /// <returns>True if a valid path was found; otherwise false.</returns>
182.     private bool TryComputePath(Vector3 from, Vector3 to, out NavMeshPath path)
183.     {
184.         path = new NavMeshPath();
185.         // Attempt to compute path between user and target positions
186.         if (!NavMesh.SamplePosition(from, out var fromHit, sampleRadius, NavMesh.AllAreas))
187.             return false;
188.         if (!NavMesh.SamplePosition(to, out var toHit, sampleRadius, NavMesh.AllAreas))
189.             return false;
190.         if (!NavMesh.CalculatePath(fromHit.position, toHit.position, NavMesh.AllAreas, path))
191.             return false;
192.
193.         // If succesfull, update status as valid
194.         return path.status != NavMeshPathStatus.PathInvalid;
195.     }
196.
197.     /// <summary>
198.     /// Calculates the total distance along a NavMesh path by summing the distances
199.     /// between its corner points.
200.     /// </summary>
201.     /// <param name="path">The path whose distance should be measured.</param>
202.     /// <returns>Total distance in world units, or 0 if the path is null or too short.</returns>
203.     private float ComputePathDistance(NavMeshPath path)
204.     {
205.         // Check if path is invalid
206.         if (path == null || path.corners.Length < 2)
```

```
207.             return 0f;
208.
209.         float dist = 0f;
210.         for (int i = 1; i < path.corners.Length; i++)
211.         {
212.             // Compute distance from user to target.
213.             dist += Vector3.Distance(path.corners[i - 1], path.corners[i]);
214.         }
215.         return dist;
216.     }
217. }
218.
```

Anexo G. Código fuente del controlador general de voz en Unity

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

/// <summary>
/// Centralized audio controller responsible for narration playback,
/// fading, and sequencing audio clips across the application.
/// </summary>
[DisallowMultipleComponent]
[RequireComponent(typeof(AudioSource))]
public sealed class AudioDirector : MonoBehaviour
{
    /// <summary>
    /// Singleton instance of the <see cref="AudioDirector"/>.
    /// </summary>
    public static AudioDirector Instance { get; private set; }

    [Header("Fades")]
    [Range(0f, 5f)]
    [Tooltip("Default fade-in duration in seconds")]
    [SerializeField]
    float defaultFadeIn = 0.0f;

    [Range(0f, 5f)]
    [Tooltip("Default fade-out duration in seconds")]
    [SerializeField]
    float defaultFadeOut = 0.25f;

    private AudioSource src;
    private Coroutine routine;
    private uint token;

    /// <summary>
    /// Ensures the singleton instance, prepares audio infrastructure,
    /// and applies volume preferences.
    /// </summary>
    void Awake()
    {
        // Singleton enforcement
        if (Instance && Instance != this)
        {
            // There can be only one AudioDirector
            Destroy(gameObject);
            return;
        }
        // Assign singleton instance
        Instance = this;
        // Persist across scenes
        EnsureAudioInfrastructure();
        ApplyVolumeFromPrefs();
        Debug.Log("[AudioDirector] Ready");
    }

    /// <summary>
    /// Ensures the necessary audio components (listener and source)
    /// exist and are configured on this GameObject.
    /// </summary>
    void EnsureAudioInfrastructure()
    {
        // Ensure AudioListener
        var listener = FindFirstObjectByType<AudioListener>();
        if (!listener)
        {
            // No AudioListener found, add one to this GameObject
            gameObject.AddComponent<AudioListener>();
            Debug.Log("[AudioDirector] Added AudioListener to Audio GO");
        }
        else if (!listener.enabled)
        {
```

```
            // Enable existing AudioListener
            listener.enabled = true;
        }
        // Ensure AudioSource
        src = GetComponent<AudioSource>();
        if (!src)
        {
            // No AudioSource found, add one to this GameObject
            src = gameObject.AddComponent<AudioSource>();
            Debug.Log("[AudioDirector] Added AudioSource to Audio GO");
        }
        // Configure AudioSource for narration
        ConfigureNarrationSource(src);
    }

    /// <summary>
    /// Configures the supplied <see cref="AudioSource"/> with
    /// appropriate settings for narration playback.
    /// </summary>
    /// <param name="s">The audio source to configure.</param>
    static void ConfigureNarrationSource(AudioSource s)
    {
        // Set narration-specific audio source settings
        s.playOnAwake = false;
        s.loop = false;
        s.spatialBlend = 0f;
        s.dopplerLevel = 0f;
        s.rolloffMode = AudioRolloffMode.Linear;
        s.minDistance = 1f;
        s.maxDistance = 10f;
        s.volume = 1f;
        s.bypassListenerEffects = false;
        s.bypassEffects = false;
        s.bypassReverbZones = true;
    }

    /// <summary>
    /// Applies the global audio volume from stored preferences.
    /// </summary>
    public static void ApplyVolumeFromPrefs()
    {
        AudioListener.volume = AppPrefs.LoadVolume() / 100f;
    }

    /// <summary>
    /// Plays a single <see cref="AudioClip"/> with optional fade-in and fade-out of any currently
playing clip.
    /// </summary>
    /// <param name="clip">The clip to play.</param>
    /// <param name="fadeIn">Fade-in duration in seconds.</param>
    /// <param name="fadeOutPrev">Fade-out duration for any currently playing clip.</param>
    public void Play(AudioClip clip, float fadeIn = -1f, float fadeOutPrev = -1f)
    {
        // Validate clip
        if (!clip)
            return;
        // Use default fades if negative values provided
        fadeIn = fadeIn < 0 ? defaultFadeIn : fadeIn;
        fadeOutPrev = fadeOutPrev < 0 ? defaultFadeOut : fadeOutPrev;
        token++;
        // Start or swap to the new clip
        StartOrSwap(new[] { clip }, fadeIn, fadeOutPrev);
    }

    /// <summary>
    /// Plays a sequence of <see cref="AudioClip"/> instances in order,
    /// with gaps and optional fades between them.
    /// </summary>
    /// <param name="clips">The ordered list of clips to play.</param>
    /// <param name="gapSeconds">Gap in seconds between clips.</param>
    /// <param name="fadeIn">Fade-in duration in seconds.</param>
    /// <param name="fadeOutPrev">Fade-out duration of previously playing clip.</param>
```

```
141.     public void PlaySequence(
142.         IReadOnlyList<AudioClip> clips,
143.         float gapSeconds = 0.05f,
144.         float fadeIn = -1f,
145.         float fadeOutPrev = -1f
146.     )
147.     {
148.         // Validate clips
149.         if (clips == null || clips.Count == 0)
150.             return;
151.         // Use default fades if negative values provided
152.         fadeIn = fadeIn < 0 ? defaultFadeIn : fadeIn;
153.         fadeOutPrev = fadeOutPrev < 0 ? defaultFadeOut : fadeOutPrev;
154.         token++;
155.         // Stop any existing routine
156.         if (routine != null)
157.             StopCoroutine(routine);
158.         // Start new sequence coroutine
159.         routine = StartCoroutine(CoSequence(clips, gapSeconds, fadeIn, fadeOutPrev, token));
160.     }
161. 
162.     /// <summary>
163.     /// Stops any currently playing audio, optionally fading it out.
164.     /// </summary>
165.     /// <param name="fadeOut">Fade-out duration in seconds.</param>
166.     public void Stop(float fadeOut = -1f)
167.     {
168.         // Use default fade-out if negative value provided
169.         fadeOut = fadeOut < 0 ? defaultFadeOut : fadeOut;
170.         token++;
171.         // Stop any existing routine
172.         if (routine != null)
173.             StopCoroutine(routine);
174.         // Start fade-out coroutine
175.         routine = StartCoroutine(CoFadeOut(src, fadeOut));
176.     }
177. 
178.     /// <summary>
179.     /// Starts playback of a set of clips, fading out any existing audio first.
180.     /// </summary>
181.     /// <param name="clips">The clips to swap to.</param>
182.     /// <param name="fadeIn">Fade-in duration in seconds.</param>
183.     /// <param name="fadeOutPrev">Fade-out duration for the previous audio.</param>
184.     void StartOrSwap(IReadOnlyList<AudioClip> clips, float fadeIn, float fadeOutPrev)
185.     {
186.         // Stop any existing routine
187.         if (routine != null)
188.             StopCoroutine(routine);
189.         // Start new swap coroutine
190.         routine = StartCoroutine(CoSwapTo(clips, fadeIn, fadeOutPrev, token));
191.     }
192. 
193.     /// <summary>
194.     /// Coroutine that fades out the current audio and then swaps to the provided clips in sequence.
195.     /// </summary>
196.     /// <param name="clips">Clips to play.</param>
197.     /// <param name="fadeIn">Fade-in duration.</param>
198.     /// <param name="fadeOutPrev">Fade-out duration for the previous audio.</param>
199.     /// <param name="tk">Token used to cancel stale coroutine instances.</param>
200.     IEnumerator CoSwapTo(IReadOnlyList<AudioClip> clips, float fadeIn, float fadeOutPrev, uint tk)
201.     {
202.         // Fade out any currently playing audio
203.         yield return CoFadeOut(src, fadeOutPrev);
204. 
205.         // Check token validity
206.         if (tk != token)
207.             yield break;
208. 
209.         // Play each clip in sequence
210.         for (int i = 0; i < clips.Count; i++)
211.         {
212.             // Check token validity
```

```
213.             var clip = clips[i];
214.             if (!clip)
215.                 continue;
216.             // Play clip with fade-in
217.             src.clip = clip;
218.             src.volume = 0f;
219.             src.Play();
220.             // Fade in to full volume
221.             yield return CoFadeTo(src, 1f, fadeIn);
222.             // Wait for clip to finish
223.             while (src.isPlaying && tk == token)
224.                 yield return null;
225.             // Check token validity
226.             if (tk != token)
227.                 yield break;
228.         }
229.     }
230.
231.     /// <summary>
232.     /// Coroutine that plays a sequence of clips with optional gaps and fades between them.
233.     /// </summary>
234.     /// <param name="clips">Clips to play.</param>
235.     /// <param name="gap">Gap in seconds between clips.</param>
236.     /// <param name="fadeIn">Fade-in duration.</param>
237.     /// <param name="fadeOutPrev">Fade-out duration before the sequence starts.</param>
238.     /// <param name="tk">Token used to cancel stale coroutine instances.</param>
239.     IEnumerator CoSequence(
240.         IReadOnlyList<AudioClip> clips,
241.         float gap,
242.         float fadeIn,
243.         float fadeOutPrev,
244.         uint tk
245.     )
246.     {
247.         // Fade out any currently playing audio
248.         yield return CoFadeOut(src, fadeOutPrev);
249.         // Iterate through clips
250.         for (int i = 0; i < clips.Count; i++)
251.         {
252.             // Check token validity
253.             if (tk != token)
254.                 yield break;
255.             // Get current clip
256.             var c = clips[i];
257.             if (!c)
258.                 continue;
259.             // Play clip with fade-in
260.             src.clip = c;
261.             src.volume = 0f;
262.             src.Play();
263.             yield return CoFadeTo(src, 1f, fadeIn);
264.             // Wait for clip to finish
265.             while (src.isPlaying && tk == token)
266.                 yield return null;
267.             // Wait for gap if not the last clip
268.             if (i < clips.Count - 1 && gap > 0f && tk == token)
269.                 yield return new WaitForSeconds(gap);
270.         }
271.         // Fade out at the end of the sequence
272.         yield return CoFadeOut(src, defaultFadeOut);
273.     }
274.
275.     /// <summary>
276.     /// Coroutine that fades out the given <see cref="AudioSource"/> over a duration,
277.     /// then stops it and resets its volume.
278.     /// </summary>
279.     /// <param name="s">The audio source to fade out.</param>
280.     /// <param name="dur">Fade-out duration in seconds.</param>
281.     static IEnumerator CoFadeOut(AudioSource s, float dur)
282.     {
283.         if (!s.isPlaying || dur <= 0f)
284.         {
```

```
285.             // Stop immediately if not playing or duration is zero
286.             s.Stop();
287.             s.volume = 1f;
288.             yield break;
289.         }
290.         // Perform fade-out
291.         float start = s.volume,
292.             t = 0f;
293.         while (t < dur)
294.         {
295.             // Increment time
296.             t += Time.unscaledDeltaTime;
297.             // Lerp volume down to zero
298.             s.volume = Mathf.Lerp(start, 0f, t / dur);
299.             yield return null;
300.         }
301.         // Stop and reset volume
302.         s.Stop();
303.         s.volume = 1f;
304.     }
305.
306.     /// <summary>
307.     /// Coroutine that fades the volume of the given <see cref="AudioSource"/>
308.     /// to the target value over the specified duration.
309.     /// </summary>
310.     /// <param name="s">The audio source to fade.</param>
311.     /// <param name="target">Target volume value.</param>
312.     /// <param name="dur">Fade duration in seconds.</param>
313.     static IEnumerator CoFadeTo(AudioSource s, float target, float dur)
314.     {
315.         // Immediate set if duration is zero or negative
316.         if (dur <= 0f)
317.         {
318.             s.volume = target;
319.             yield break;
320.         }
321.         // Perform fade
322.         float start = s.volume,
323.             t = 0f;
324.         while (t < dur)
325.         {
326.             // Increment time
327.             t += Time.unscaledDeltaTime;
328.             // Lerp volume towards target
329.             s.volume = Mathf.Lerp(start, target, t / dur);
330.             yield return null;
331.         }
332.         // Ensure final volume is set
333.         s.volume = target;
334.     }
335. }
336.
```

Anexo H. Código fuente principal del sistema de Generación de Voz

```python
from google.cloud import texttospeech_v1beta1 as texttospeech
from values import DEFAULT_SAMPLE_RATE

def build_client():
    """
    Build and return a Google TTS client.
    returns:
        texttospeech.TextToSpeechClient: Google TTS client.
    """
    return texttospeech.TextToSpeechClient()

def build_voice(model: str, voice_name: str, lang: str):
    """
    Build and return a voice selection params for Google TTS.
    args:
        model (str): TTS model name.
        voice_name (str): Voice name.
        lang (str): Language code.
    returns:
        texttospeech.VoiceSelectionParams: Voice selection parameters.
    """
    tts_voice = texttospeech.VoiceSelectionParams(
        name=voice_name,
        language_code=lang,
        model_name=model)

    return tts_voice

def build_audio_cfg(sample_rate_hz: int = DEFAULT_SAMPLE_RATE):
    """
    Build and return an audio config for Google TTS.
    args:
        sample_rate_hz (int): Sample rate in Hz.
    returns:
        texttospeech.AudioConfig: Audio configuration.
    """
    # Configure audio output as LINEAR16 PCM with specified sample rate
    audio_cfg = texttospeech.AudioConfig(
        audio_encoding=texttospeech.AudioEncoding.LINEAR16,
        sample_rate_hertz=sample_rate_hz)

    return audio_cfg

def synthesize(client, voice, audio_cfg, prompt: str, text: str) -> bytes:
    """
    Synthesize speech using Google TTS API.
    args:
        client (texttospeech.TextToSpeechClient): Google TTS client.
        voice (texttospeech.VoiceSelectionParams): Voice selection parameters.
        audio_cfg (texttospeech.AudioConfig): Audio configuration.
        prompt (str): Prompt text to style audio.
        text (str): Main text to synthesize.
    returns:
        bytes: Synthesized PCM audio data.
    """
    # Prepare synthesis input with prompt and text
    synthesis_input = texttospeech.SynthesisInput(
        text=text,
        prompt=prompt or None)

    # Call the TTS API to synthesize speech
    resp = client.synthesize_speech(
        input=synthesis_input,
        voice=voice,
        audio_config=audio_cfg)

    # Return the synthesized audio content
    return resp.audio_content
```