Universidad
Rey Juan Carlos

ESCUELA DE INGENIERÍA DE FUENLABRADA

INGENIERÍA EN SISTEMAS AUDIOVISUALES Y MULTIMEDIA

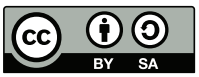
TRABAJO FIN DE GRADO

EDICIÓN DE ESCENAS 3D MEDIANTE VOZ

Autor : Pablo Esteban Camuendo Carlosama

Tutor : Dr. Jesús María González Barahona

Curso académico 2024/2025



©2025 Pablo Esteban Camuendo Carlosama

Algunos derechos reservados

Este documento se distribuye bajo la licencia

“Atribución-CompartirIgual 4.0 Internacional” de Creative Commons,

disponible en

https://creativecommons.org/licenses/by-sa/4.0/deed.es

Dedicado a
mi familia.

Agradecimientos

Quisiera expresar mi más sincero agradecimiento a todas las personas e instituciones que han hecho posible la realización de esta tesis de grado.

En primer lugar, mi familia, durante todos estos años de estudio, por su apoyo incondicional y esfuerzo, su comprensión y aliento continuo. Su fe en mí es la mayor aliento. Ellos siempre han dicho que la educación es la base de todo lo que puedo construir a continuación y es gracias a ellos que he podido llegar aquí. Gracias, mamá y papá.

A mi hermana, que ha sido un constante apoyo y de la cual quiero ser un ejemplo a seguir, que sepa que el desarrollo de uno mismo es difícil, pero satisfactorio al final.

A mi tutor de tesis, por su guía, paciencia y conocimientos compartidos a lo largo de este proceso. Su experiencia ha sido crucial para el desarrollo de este trabajo.

Finalmente, a todos aquellos que de una u otra manera han contribuido a mi crecimiento personal y profesional durante mi etapa universitaria.

A todos, mi más profunda gratitud.

Resumen

En este trabajo de fin de grado se ha investigado la integración de técnicas de reconocimiento de voz en entornos 3D. Como resultado, se ha creado un prototipo de creación y edición de escenas mediante comandos de voz, usando la unión del framework llamado A-Frame y una API de reconocimiento de voz. Así, creando una caja de herramientas con componentes reutilizables y modulares en A-Frame.

Los sistemas de realidad virtual y aumentada cada vez son más importantes porque permiten experiencias inmersivas intentando emular el mundo real. En la actualidad, se están explorando nuevas interfaces en realidad virtual (VR) y realidad aumentada (AR). Entre ellas, una de las más interesantes es la utilización de voz, ya que parte como una de las más prometedoras para mejorar la inmersión y la usabilidad. El objetivo es explorar cómo esta tecnología puede transformar la manera en la que se puede llegar a consumir el contenido digital.

Hay varios aspectos a destacar en este proyecto. En primer lugar, se desarrolló un sistema de componentes que permite incorporar voz a cualquier escena hecha con A-Frame. Esto es muy relevante, ya que A-Frame es una de las plataformas más populares para la creación de escenas en realidad virtual que funcionan sobre un navegador. Además de esta funcionalidad, también se añaden componentes modulares que no solo facilitan la incorporación de comandos, sino que también abren la puerta al diseño de diferentes aplicaciones usando esta caja de herramientas. Esto no solo mejora la accesibilidad y la experiencia de usuario, sino que también impulsa la innovación en el diseño de interfaces de VR/AR.

Summary

In this thesis, we have investigated the integration of voice recognition techniques in 3D environments. As a result, a prototype for creating and editing scenes using voice commands has been developed, by combining the A-Frame framework with a speech recognition API. Thus, creating a toolbox with reusable and modular components in A-Frame.

Virtual and augmented reality systems are becoming increasingly important because they enable immersive experiences that try to emulate the real world. Currently, new interfaces are being explored in virtual reality (VR) and augmented reality (AR). Among these, one of the most interesting is the use of voice, as it stands out as one of the most promising ways to improve immersion and usability. The goal is to explore how this technology can transform the way digital content is consumed.

There are several notable aspects of this project. First, a component system was developed that allows the integration of voice commands into any scene created with A-Frame. This is highly relevant, as A-Frame is one of the most popular platforms for building browser-based virtual reality scenes. In addition to this functionality, modular components were also added. These not only simplify the incorporation of voice commands but also open the door to designing diverse applications using this toolbox. This not only enhances accessibility and user experience but also drives innovation in VR/AR interface design.

Índice general

1. Introducción **1**

1.1. Objetivo general 2

1.2. Objetivos específicos 2

1.3. Estructura de la memoria 3

2. Tecnologías utilizadas **5**

2.1. Tecnologías de reconocimiento de voz 5

2.1.1. Whisper 5

2.1.2. Whisper Web 6

2.1.3. Web Speech API 8

2.1.4. AssemblyAI 10

2.2. A-Frame 10

2.3. Three.js 13

2.4. WebXR 14

2.5. WebGL 16

2.6. HTML 17

2.7. JavaScript 18

2.8. CSS 19

2.9. Node.js 20

2.9.1. Node.js y el Manejo de Peticiones HTTP 20

2.9.2. Certificados SSL/TLS 21

2.10. Aplicaciones relacionadas 21

2.10.1. Arkio 21

2.10.2. VR Sculpting 22

2.10.3. Bridge Crew (Star Trek) 23
2.11. Tecnologías auxiliares 24
2.11.1. GitHub 24
2.11.2. Visual Studio Code 25
2.11.3. Gafas Meta Quest 3 25
2.11.4. LaTeX 26

3. Desarrollo del proyecto **28**
3.1. Estructura del proceso de desarrollo 29
3.2. Sprint 1 30
3.2.1. Objetivos 30
3.2.2. Tareas Realizadas 30
3.2.3. Resultados y Lecciones aprendidas 32
3.3. Sprint 2 33
3.3.1. Objetivos 33
3.3.2. Tareas Realizadas 33
3.3.3. Resultados 35
3.3.4. Lecciones aprendidas 36
3.4. Sprint 3 36
3.4.1. Objetivos 37
3.4.2. Tareas Realizadas 37
3.4.3. Resultados 38
3.4.4. Lecciones aprendidas 38
3.5. Sprint 4 39
3.5.1. Objetivos 39
3.5.2. Tareas Realizadas 39
3.5.3. Resultados 41
3.5.4. Lecciones aprendidas 41
3.6. Sprint 5 42
3.6.1. Objetivos 42
3.6.2. Tareas Realizadas 42

3.6.3. Resultados 43
3.6.4. Lecciones aprendidas 43
3.7. Sprint 6 44
3.7.1. Objetivos 44
3.7.2. Tareas Realizadas 44
3.7.3. Resultados 46
3.7.4. Lecciones aprendidas 46

4. Sistema resultante **47**
4.1. Aplicación desarrollada 47
4.1.1. Entorno de ejecución 48
4.1.2. Objetos en escena 48
4.1.3. Comandos de Voz 49
4.1.4. Uso de los Comandos 50
4.1.5. Notas Adicionales 53
4.2. Descripción de la implementación 54
4.2.1. Arquitectura General 54
4.2.2. Componentes Principales 54
4.2.3. Componentes Auxiliares 58
4.2.4. Funciones de utilidad 58
4.3. Arquitectura de funcionamiento de la aplicación 59
4.4. Reutilización y Modularidad 60
4.4.1. Ejemplo de la aplicación completa del proyecto 61
4.4.2. Ejemplo de creación de aplicación de transcripción de voz en pantalla con A-Frame 61

5. Experimentos y validación **64**
5.1. Objetivos 64
5.2. Entorno de pruebas 64
5.3. Resultados 67
5.3.1. Precisión del Reconocimiento de Voz 67
5.3.2. Tiempo por Tarea 67

5.3.3. Opiniones de los Usuarios 68
5.4. Conclusión de Validación 69

6. Conclusiones **70**
6.1. Consecución de objetivos 70
6.2. Esfuerzo y recursos dedicados 72
6.3. Aplicación de lo aprendido 75
6.4. Lecciones aprendidas 76
6.5. Trabajos futuros 76

Bibliografía **78**

Índice de figuras

2.1. Imagen de los modelos de Whisper extraida de su GitHub 6
2.2. Imagen de Whisper Web 7
2.3. Compatibilidad en exploradores segun MDN web docs 9
2.4. Ejemplo basico de WebSpeechAPI 9
2.5. Escena simple A-Frame 12
2.6. Escena de ejemplo en three.js 14
2.7. Escena de ejemplo en WebXR 15
2.8. Ejemplo de aplicación en WebGL 16
2.9. Ejemplo de escena de Arkio 22
2.10. Ejemplo de uso de Shapelab 23
2.11. Ejemplo de uso de voz en Star Trek Bridge Crew 24
2.12. Gafas Meta Quest 3 26

3.1. Funcionamiento de la demo de reconocimiento de voz en A-Frame 32
3.2. Flujo de creación de componentes 34
3.3. Flujo de creación de componentes avanzados 34
3.4. Funcionamiento del comando de creacion de objetos básico 35
3.5. Flujo de creación de componentes avanzados 37
3.6. Funcionamiento del generador de objetos con funcionalidades extra 38
3.7. Flujo de creación de componentes avanzados 39
3.8. Funcionamiento de eliminar objetos con ID 40
3.9. Funcionamiento de editar objetos con ID 40
3.10. Funcionamiento del generador de assets 41
3.11. Funcionamiento de htmlembed 45

3.12. Componentes finales agregados 45

4.1. Objetos en escena que puede ver el usuario. 48
4.2. Creación de objetos, ejemplo 'contenedor' 50
4.3. Escena de edición con objeto resaltado 51
4.4. Escena de eliminación del objeto paco 52
4.5. Escena de ayuda con los comandos posibles 53
4.6. Como cerrar panel de ayuda al usuario 53
4.7. Arquitectura de la Aplicación 59

5.1. Tabla de tiempos de los usuarios 68

6.1. Cronograma del desarrollo del proyecto 74

Capítulo 1

Introducción

Con el gran avance en tecnologías, la realidad virtual (VR) ha experimentado un gran crecimiento en los últimos años, gracias a la evolución de tecnologías web accesibles y de código abierto. Actualmente, existe una búsqueda para nuevas interfaces de usuario y forma de interaccionar en la escena que superen las limitaciones de las formas tradicionales, como son el teclado, controles físicos o gestos manuales.

La interacción natural se vuelve un punto clave, en donde se están explorando diferentes modos que permitan al usuario interactuar con entornos 3D de manera más intuitiva. Aquí es donde el reconocimiento del habla cobra más importancia. La integración de comandos de voz permite al usuario manipular objetos o activar funciones sin necesidad de interfaces tangibles. Esto no solo es una mejora en la accesibilidad, sino que también aumenta la inmersión, ya que la interacción con la escena se siente más natural y fluida, acercándose a cómo se interactúa en el mundo real. Por ello, el reconocimiento de voz se vuelve un punto importante en la mejora del potencial en experiencias de realidad virtual o aumentada.

Además, estas tecnologías se están implementando en el navegador, aprovechando los estándares web que tanto éxito han tenido en el acceso a la información y a las aplicaciones. En este punto, la unión entre estas tres tecnologías no solo facilita la creación de experiencias inmersivas más accesibles y usables, sino que también impulsa la innovación en la forma en la que interactuamos con el contenido.

Este proyecto surge de la inquietud por investigar nuevas formas de interacción en entornos 3D web. A día de hoy existen avances significativos en la creación visual de escenas virtuales, la edición y manipulación de objetos puede resultar en ocasiones compleja y poco intuitiva. La

idea de un editor controlado por voz sería una alternativa prometedora para agilizar el flujo de trabajo y mejorar la accesibilidad para usuarios con diferentes necesidades.

Actualmente, se ha avanzado en la exploración de diferentes arquitecturas y sistemas de componentes para la implementación de un editor 3D dirigido por voz. Como resultado de esta investigación, se ha desarrollado un prototipo funcional que permite la creación y edición básica de objetos en una escena 3D mediante comandos de voz. Sin embargo, es importante señalar que la aplicación aún se encuentra en una fase de desarrollo y no está completamente pulida. El trabajo futuro se centrará en refinar la funcionalidad existente, ampliar el conjunto de comandos de voz y mejorar la estabilidad y el rendimiento general del sistema.

Se ha creado una página web[1] del TFG. En ella, se encuentran disponibles la memoria del proyecto, la presentación y enlaces a diferentes demostraciones de la aplicación, así como al repositorio donde se almacena todo el contenido.

1.1. Objetivo general

El trabajo de fin de grado consiste en explorar diversas técnicas para la construcción de un editor 3D dirigido por voz, basado en un sistema de componentes que demuestre la viabilidad de la creación y edición de escenas 3D mediante comandos de voz en navegadores web compatibles.

1.2. Objetivos específicos

- Explorar qué opciones se tienen para 'Speech-to-text' en un navegador y en dispositivos tipo Quest 3.
- Conseguir transcripciones del usuario con la aplicación.
- Reconocimiento de órdenes concretas para interactuar con el editor de escenas.
- Explorar distintas soluciones para encontrar una que funcione bien en gafas tipo Quest 3 y otra en el navegador de escritorio.
- Hacer que el prototipo soporte comandos como crear, editar, eliminar...

[1]Página web: `https://r4cc00n.github.io/SpeechRecognition_in_Browser/`

- Realización de un prototipo modular.
- Construir un prototipo funcional en navegadores
- Construir un prototipo funcional en equipos VR/AR como las gafas Quest 3.
- Utilizar para la construcción el framework A-Frame, porque se adapta especialmente bien a los objetivos anterior.

1.3. Estructura de la memoria

El resto de la memoria se estructurara de la siguiente manera:

- Tecnologías utilizadas (**Capítulo 2**): Se muestran todas las tecnologías utilizadas durante el desarrollo del proyecto, desde las principales como A-Frame para la construcción de entornos inmersivos en el navegador, hasta herramientas auxiliares como Node.js o WebXR. Se da una breve descripción de cada tecnología, el contexto de uso y su papel dentro de cada una de las demos.
- Desarrollo del Proyecto (**Capítulo 3**): Se detalla el proceso completo de creación, donde se implementaron dos versiones de la demo: una que usa la Web Speech API directamente en el navegador, y otra que envía audio a un servidor local para procesarlo con Assembly AI.
- Resultados (**Capítulo 4**): Aquí se muestra el resultado final de ambas demos
  - Descripción para el usuario: Se explica qué funcionalidades incluye cada demo, cómo se activan los comandos por voz y cómo se muestra el resultado del proceso dentro del entorno 3D.
  - Descripción del comportamiento de la interfaz: Se detalla cómo se representa cada componente de la escena VR/AR, así como la lógica detrás de la creación/edición de objetos en la escena 3D.
- Experimentos y validación (**Capítulo 5**): Se describen los pasos que se realizaron para el experimento, incluyendo los valores clave a medir y los resultados que se obtuvieron de la interacción directa de los usuarios con el prototipo.

- Conclusiones (**Capítulo 6**): Se aborda el cumplimiento de los objetivos planteados, destacando el tiempo y esfuerzo dedicado en el desarrollo del proyecto. Así como la aplicación de conocimientos y lecciones aprendidas. Finalmente, se hablará de los trabajos futuros identificados a lo largo del desarrollo, los cuales tienen potencial de expansión.

Capítulo 2

Tecnologías utilizadas

En este capítulo se repasarán todas las tecnologías que se emplearon en el desarrollo del proyecto, abordando con detalle aquellas herramientas que fueron más importantes, se presentará una descripción de cada tecnología, el contexto en el que se usó y el papel que desempeñó en el prototipo final.

2.1. Tecnologías de reconocimiento de voz

El reconocimiento de voz en navegadores permite a los usuarios interactuar con aplicaciones web y contenido utilizando su voz en lugar de escribir. Esto abre un mundo de posibilidades para la accesibilidad, la productividad y la creación de experiencias de usuario más interactivas.

2.1.1. Whisper

En primera instancia, se planeaba trabajar con Whisper[1] [23], un modelo de 'Speech Recognition' (reconocimiento de voz) de OpenAI que se destaca por su capacidad para transcribir audio en múltiples idiomas con alta precisión, incluso en condiciones de ruido o con variaciones en el acento del hablante.

Whisper tiene una arquitectura robusta y de código libre. Como parte de la investigación, se analizaron los requisitos técnicos para la implementación, incluyendo dependencias, consumo de recursos, y compatibilidad con otros componentes del sistema.

[1] Whisper OpenAI: `https://openai.com/es-ES/index/whisper/`.

Size	Parameters	English-only model	Multilingual model	Required VRAM	Relative speed
tiny	39 M	`tiny.en`	`tiny`	~1 GB	~10x
base	74 M	`base.en`	`base`	~1 GB	~7x
small	244 M	`small.en`	`small`	~2 GB	~4x
medium	769 M	`medium.en`	`medium`	~5 GB	~2x
large	1550 M	N/A	`large`	~10 GB	1x
turbo	809 M	N/A	`turbo`	~6 GB	~8x

Figura 2.1: Imagen de los modelos de Whisper extraida de su GitHub [24]

Whisper presenta los modelos que se pueden ver en la Figura 2.1. Se pueden observar diferentes tamaños de modelos, cada uno con distintos niveles de precisión, velocidad y requerimientos de hardware.

Los modelos más grandes (medium, large) son altamente demandantes de GPU, esto hace que los modelos tengan mucha más precisión, pero que su velocidad sea más afectada. Para una menor latencia, los modelos más pequeños (tiny, base, small) son mejores, ya que su demanda de hardware es menor.

Alguno de los problemas que presentó Whisper fue el tamaño de los modelos, ya que incluso el modelo más pequeño (tiny) pesa varios cientos de MB. Otro problema es la necesidad de PyTorch y otras dependencias que no están disponibles en entornos web estándar. Además del uso intensivo de GPU/CPU, lo cual no es viable en la mayoría de navegadores móviles o de escritorio.

2.1.2. Whisper Web

Actualmente, algunos proyectos han empezado a portar versiones ligeras al navegador (construcciones en WebAssembly u ONNX). Un ejemplo de esto es Whisper Web[2] [34].

En la Figura 2.2 se muestra un proyecto de la comunidad que busca llevar Whisper al navegador, sin la necesidad de un back-end. Para ello se usan tecnologías como:

- WebAssembly[3] (WASM): para correr código pesado directamente en el navegador.

[2]Demo oficial en: `https://huggingface.co/spaces/Xenova/whisper-web`
[3]WASM: `https://developer.mozilla.org/es/docs/WebAssembly`

- ONNX[4] (Open Neural Network Exchange): para convertir modelos de PyTorch a un formato más portable.
- WebGPU o WebGL[5]: para aprovechar el hardware del usuario, siempre que sea compatible.

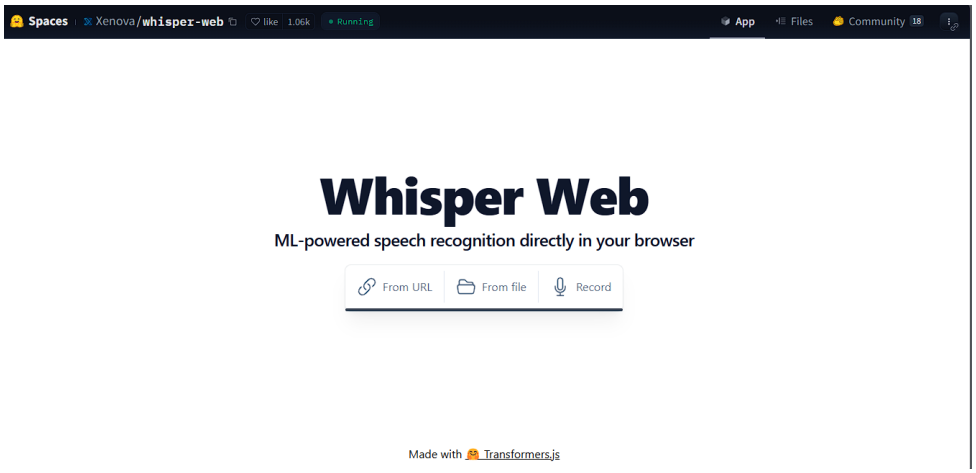


Figura 2.2: Imagen de Whisper Web [35]

Para ello se da uso a la plataforma y comunidad de Hugging Face[6] enfocada en el desarrollo, distribución y uso de modelos de inteligencia artificial.

El funcionamiento en esta aplicación web es el siguiente:

- Se carga un archivo de audio o se graba la voz.
- En el navegador elige una versión comprimida del modelo Whisper y la elección del lenguaje si el usuario lo desea.
- El modelo se ejecuta localmente en el navegador (no sube tu audio a la nube).
- La app procesa el audio y muestra la transcripción en pantalla.
- El navegador puede exportar la información.

Esta idea se descartó debido a la complejidad de implementación en el navegador, ya que ejecutar modelos de reconocimiento de voz como Whisper directamente en la web implica múltiples

[4]ONNX: `https://learn.microsoft.com/es-es/windows/ai/windows-ml/get-onnx-model`
[5]Se hablará de él en la sección 2.5
[6]Hugging Face: `https://huggingface.co/`

desafíos técnicos. Entre ellos destacan el alto consumo de recursos, la falta de soporte en entornos como WebAssembly/WebGPU, y la necesidad de transcribir y optimizar los modelos para que puedan ejecutarse localmente en el cliente sin afectar la experiencia del usuario.

Además, el peso de los modelos de en sus versiones más pequeñas, puede afectar significativamente los tiempos de carga y procesamiento, especialmente en dispositivos móviles o con hardware limitado. Por estas razones, se optó por explorar soluciones alternativas que pudieran integrarse de manera más eficiente dentro del proyecto.

2.1.3. Web Speech API

La principal tecnología nativa para reconocimiento de voz directamente en los navegadores es la Web Speech API[7] [2], específicamente su interfaz SpeechRecognition [16].

La Web Speech API fue diseñada para el análisis y síntesis de voz, creando la posibilidad al usuario de enviar voz a aplicaciones web. Dichos audios pueden ser transformados en texto. Fue introducida como un borrador por el W3C[8] (World Wide Web Consortium) en 2012. Aunque algunas partes de la API, como la síntesis de voz (SpeechSynthesis), han alcanzado un estado de recomendación, la parte de reconocimiento de voz (SpeechRecognition) aún se considera un borrador de trabajo. Sin embargo, ha sido implementada en la mayoría de los navegadores modernos, aunque con diferentes niveles de soporte y posibles requisitos de prefijos específicos del navegador (ej.: webkitSpeechRecognition en Chrome), por ejemplo el uso de esta API en navegadores específicos para gafas VR/AR como las Quest 3 es directamente imposible, debido a que no hay soporte para ello, por esto lo más factible para el uso de reconocimiento de voz en casos como estos es usar un back-end que reciba las peticiones y chuncks de audio del cliente y está mismo las transcriba devolviendo así el texto transcrito.

En este proyecto se usará el 'SpeechRecognition' para convertir la voz en texto. Es cierto que la compatibilidad de navegadores es muy grande, pero el soporte de SpeechRecognition no es estándar aún, y muchos navegadores solo lo permiten con el prefijo webkit (webkitSpeechRecognition). Funciona principalmente en Chrome y navegadores basados en Chromium, como se ve en la Figura 2.3.

[7] Web Speech API según Julius Adorf

[8] W3C: `https://www.w3.org/`

Figura 2.3: Compatibilidad en exploradores segun MDN web docs [16]

Finalmente, una de las razones por las que se optó por esta opción es su facilidad de implementación, ya que no requiere de librerías externas, se ejecuta directamente en el navegador y es ideal para el uso de prototipos, demos o apps ligeras.

```
// Verificar compatibilidad con la API de reconocimiento de voz
if ('webkitSpeechRecognition' in window) {
    recognition = new webkitSpeechRecognition();
    recognition.continuous = true;
    recognition.interimResults = false;
    recognition.lang = 'es-ES';
} else {
    console.error('API de reconocimiento de voz no soportada en este navegador');
}
```

Figura 2.4: Ejemplo basico de WebSpeechAPI

Como se observa en la Figura 2.4 la integración se hace en pocas líneas de código. Este código muestra la activación de reconocimiento continuo **'recognition.continuous = true; '**, esto quiere decir que se procesara la transcripción continuamente, pero con la condición de devolver el resultado final gracias a **'recognition.interimResults = false; '** además de seleccionar el idioma que se desea transcribir.

Por otro lado, no todas son ventajas, existen limitaciones para el uso de esta API, tales como el requerimiento de conexión a internet, ya que utiliza los servidores de Google para el reconocimiento. Además, el soporte y el comportamiento pueden variar entre diferentes navegadores. Y sobre todo, cuenta con una menor precisión en comparación con modelos como Whisper, especialmente en ambientes ruidosos o con acentos fuertes.

2.1.4. AssemblyAI

Assembly AI[9] [5] es una plataforma de Inteligencia Artificial que ofrece potentes APIs para el procesamiento de audio, incluyendo reconocimiento de voz (Speech-to-Text), comprensión del lenguaje natural (NLP) y otras funcionalidades de inteligencia de audio.

Cuenta con características similares a Web Speech API, salvo que tiene algunas mejoras en otros puntos.

Assembly AI se enfoca en ofrecer modelos de reconocimiento de voz de última generación entrenados en grandes cantidades de datos, lo que generalmente resulta en una mayor precisión, especialmente en entornos ruidosos o con acentos variados, con la posibilidad de identificar al hablante, detectar las emociones expresadas en el audio, incluso admite diferentes formatos de archivo y fuentes de audio/video que pueden ser en tiempo real o no.

La API es robusta y escalable, fue diseñada para desarrolladores que necesitan integrar capacidades de reconocimiento de voz en aplicaciones complejas y a gran escala. Por ello al ser una API externa, funciona de manera consistente en diferentes navegadores y plataformas, su uso en navegadores se realizará a través de una API REST y las respuestas se reciben como respuesta a dicha API por esto requiere una conexión a internet activa para enviar y recibir datos de la API. Assembly AI es un servicio comercial y tiene costos asociados en función del volumen de audio procesado, aunque a menudo ofrecen planes gratuitos o de prueba.

Finalmente, se usa esta tecnología para el proyecto en una segunda demo para el uso de gafas VR/AR, debido a que los navegadores de Meta de las gafas de pruebas no soportaban el manejo de la API anterior de Web Speech Recognition.

2.2. A-Frame

Para la implementación en AR/VR se usó A-Frame[10] [1], este es un framework web de código abierto usado para el desarrollo de experiencias en realidad virtual y aumentada. Utilizando HTML como lenguaje principal para la definición de escenas. Este framework se basa en Three.js, el cual es una biblioteca de JavaScript para gráficos en 3D.

A-Frame se caracteriza por el uso simplificado de Three.js para la creación de escenas,

[9] Documentación de Assembly: `https://www.assemblyai.com/docs`.

[10] A-Frame:`https://aframe.io/`.

evitando el directo y complejo gestionamiento de los detalles de renderización en 3D.

Usando una arquitectura llamada ECS (Entity Components System) que es aplicada a los videojuegos, en donde cada objeto es una entidad diferenciada, que puede o no albergar otras entidades. Se debe tener en cuenta que para acceder a la librería de este framework se puede usar la línea de código del Listado 2.1. Esta etiqueta `<script>` permite cargar la versión 1.6.0 de A-Frame directamente desde la web. Dicha versión puede variar.

```
<script src='https://aframe.io/releases/1.6.0/aframe.min.js'></script>
```

Listado 2.1: Línea de código de A-Frame

Una de las características más claras de A-Frame es la definición de entidades utilizando un etiquetado similar al de HTML. Esto permite a desarrolladores construir mundos virtuales de manera rápida y simple, como se observa en este fragmento de código del Listado 2.2 y en la Figura 2.5.

```
<a-scene>
  <a-box position='-1 0.5 -3' rotation='0 45 0' color='#4CC3D9'></a-
     box>
  <a-sphere position='0 1.25 -5' radius='1.25' color='#EF2D5E'></a-
     sphere>
  <a-cylinder position='1 0.75 -3' radius='0.5' height='1.5' color='#
     FFC65D'></a-cylinder>
  <a-plane position='0 0 -4' rotation='-90 0 0' width='4' height='4'
     color='#7BC8A4'></a-plane>
  <a-sky color='#ECECEC'></a-sky>
</a-scene>
```

Listado 2.2: Escena A-Frame básica

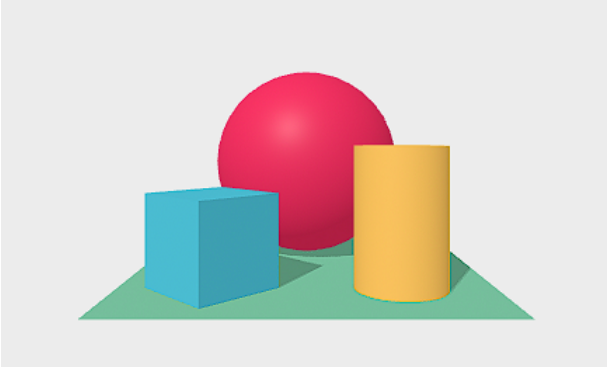


Figura 2.5: Escena simple A-Frame

Cada entidad de esta escena tiene sus propiedades especificadas en la página de la documentación de A-Frame[11].

El núcleo de A-Frame es su sistema de **entidades-componentes**. Las **entidades**[12] son objetos genéricos que adquieren propiedades y comportamientos mediante componentes. Los **componentes**[13] son módulos reutilizables que añaden comportamientos, propiedades visuales, interacciones y varias acciones más a las entidades. A-Frame proporciona componentes predefinidos (position, rotation, scale, material, geometry...). Se puede observar en el Listado 2.3, el componente añade una entidad con un `<a-text>` que dará una información al usuario.

```
  AFRAME.registerComponent('text-message', {
    init: function () {
        const userMessage = document.createElement('a-text');
        userMessage.setAttribute('id', 'userMessage');
        userMessage.setAttribute('value', 'Bienvenido a VOICE VR');
        userMessage.setAttribute('align', 'center');
        userMessage.setAttribute('color', 'black');
        userMessage.setAttribute('position', '0 20 -45');
        userMessage.setAttribute('width', '25');
        userMessage.setAttribute('look-at', '[camera]');
        this.el.appendChild(userMessage);
    }
});
```

Listado 2.3: Crear Componente

[11] Primitivas de A-Frame: `https://aframe.io/docs/1.7.0/introduction/html-and-primitives.html`
[12] Detalles sobre entidades de A-Frame: `https://aframe.io/docs/1.6.0/core/entity.html`
[13] Detalles sobre componentes de A-Frame: `https://aframe.io/docs/1.6.0/core/component.html`

Una entidad puede tener muchos componentes a la vez. Se pueden modificar los valores de un componente en tiempo real usando JavaScript. La creación de componentes depende de varias cosas como:

- **Definición del componente:** Se realiza utilizando el método `AFRAME.registerComponent`.
- **schema:** Define los datos que el componente puede recibir, como configuraciones o parámetros personalizables.
- **init():** Función que se ejecuta al iniciar el componente. Se utiliza para establecer comportamientos iniciales o escuchar eventos de la escena.
- **this.el:** Hace referencia a la entidad (`<a-entity>`) sobre la cual está aplicado el componente.

2.3. Three.js

Como se mencionó anteriormente, A-Frame se construye sobre la potente biblioteca de gráficos 3D Three.js[14][15] [7] el cual es una biblioteca de JavaScript que facilita la creación de gráficos 3D. A-Frame abstrae la complejidad de Three.js, permitiendo a los desarrolladores centrarse en la creación de contenido sin tener que lidiar directamente con la configuración de WebGL, shaders, matrices, etc. Sin embargo, para necesidades más avanzadas, los desarrolladores pueden acceder directamente al objeto Three.js subyacente dentro de los componentes personalizados de A-Frame. Con Three.js, puedes crear y manipular formas básicas como cubos o esferas, importar modelos complejos (en formatos como GLTF), aplicar materiales y texturas, e iluminar escenas con luces realistas. Además, incluye cámaras y controles intuitivos para navegar, así como soporte para animaciones fluidas.

La construcción de escenas mediante Three.js utilizará componentes como:

- **Escena (Scene)**: El contenedor de todos los elementos 3D, como objetos, luces y cámaras.
- **Cámara (Camera)**: Define el punto de vista desde el que se observa la escena. Las más comunes son la cámara de perspectiva y la ortográfica.

[14] Learning Three.js: The JavaScript 3D Library for WebGL.

[15] Three.js:`https://threejs.org/`.

- **Renderizador (Renderer)**: Convierte la escena y la cámara en una imagen visible en el lienzo HTML (`<canvas>`), usando WebGL.
- **Objetos (Mesh)**: Representan modelos 3D, creados a partir de geometrías (formas básicas como cubos, esferas, planos, etc.) y materiales (color, textura, iluminación).
- **Luces (Light)**: Iluminan la escena y permiten que los materiales reaccionen visualmente según el tipo de luz aplicada.

Un ejemplo claro es la Figura 2.6 en donde se puede observar el potencial de esta biblioteca de gráficos 3D.

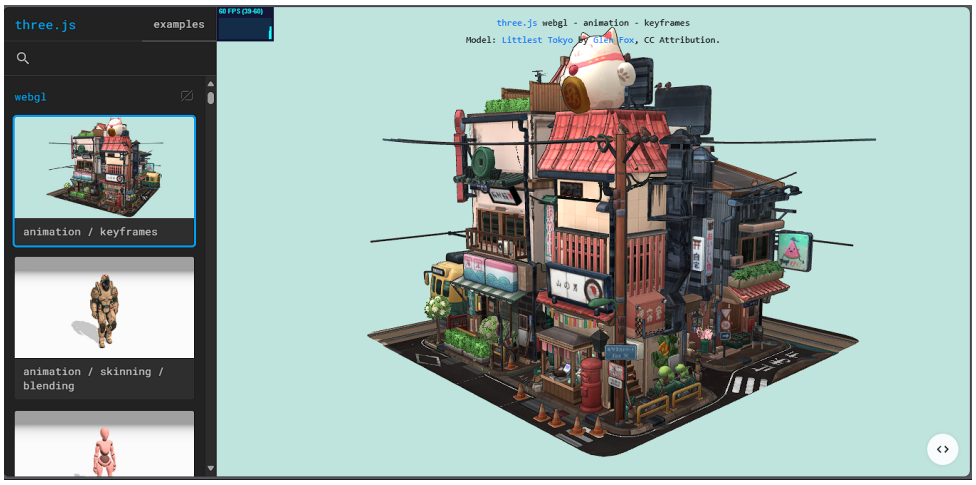


Figura 2.6: Escena de ejemplo en three.js [30]

2.4. WebXR

Contextualizando este apartado, se tiene que conocer la historia y desarrollo de WebXR. Fue desarrollada bajo las especificaciones del consorcio de W3C [32] (World Wide Web)[16].

Su predecesor fue WebVR:

> *'WebVR is an open specification that makes it possible to experience VR in your browser.' [33].*

Esta era una API experimental creada únicamente para el desarrollo en realidad virtual. Este estándar fue algo fundamental para el desarrollo actual, pero presentaba varias limitaciones,

[16]WebXR según W3C.

ya que no abordaba los problemas relacionados con la realidad aumentada. La evolución y la integración de la realidad aumentada es la que creó el actual WebXR.

WebXR (Web Extended Reality) es un conjunto de tecnologías web que permite crear experiencias en realidad virtual (VR) y en realidad aumentada (AR) dentro de navegadores. Esto implica que cualquier usuario puede acceder a estas experiencias sin necesidad de descargar aplicaciones separadas.

WebXR presenta varias funcionalidades como:

- **Renderizado de Escenas 3D:** Facilita el renderizado de escenas 3D en estos dispositivos a las velocidades de fotogramas adecuadas, creando que la experiencia sea inmersiva.
- **Seguimiento del Movimiento:** WebXR puede rastrear el movimiento y la orientación de la cabeza y los controladores del usuario, lo que permite la interactividad.
- **Manejo de Entrada:** Admite la entrada de varios controladores y dispositivos XR, lo que permite la interacción del usuario.

Estas funcionalidades mencionadas más otras, crean las beneficiosas características que hacen que este tipo de tecnologías web sean accesibles y compatibles en diferentes plataformas, aprovechando las diferentes tecnologías web familiares como JavaScript, HTML y WebGL para crear experiencias WebXR como se muestra en la Figura 2.7.

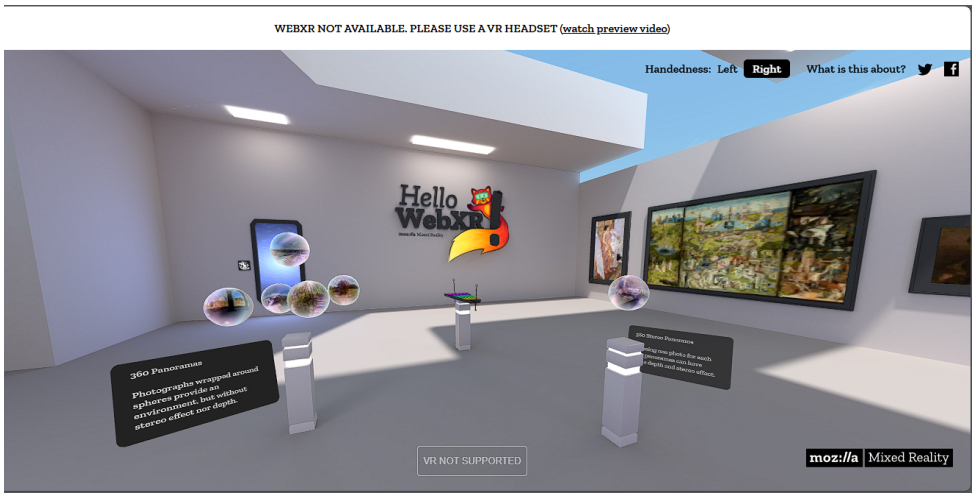


Figura 2.7: Escena de ejemplo en WebXR [17]

2.5. WebGL

WebGL surgió por la necesidad de llevar los gráficos 3D directamente a un navegador. Antes de que los gráficos 3D se limitaban a tecnologías basadas en plugins como 'Adobe Flash' o renderización puramente con JavaScript, esto resultaba en una caída abrupta de rendimiento.

Creada en 2007 por el grupo Khronos[17] [11] fue desarrollada basándose en una API conocida como OpenGL ES 2.0 diseñada para dispositivos embebidos.

> *'WebGL (Web Graphics Library) is a JavaScript API for rendering high-performance interactive 3D and 2D graphics within any compatible web browser without the use of plug-ins. WebGL does so by introducing an API that closely conforms to OpenGL ES 2.0.' [13]*[18]

Esta tecnología fue impulsada gracias a empresas como Mozilla y Google, quienes fueron actores clave para la implementación y desarrollo del WebGL.

Una de las características más importantes de WebGL es la capacidad de aprovechamiento de la GPU de los dispositivos del usuario, permitiendo una renderización más rápida y eficiente de gráficos en 3D. Por otro lado, la integración en el DOM junto con otras tecnologías web como HTML, CSS y JavaScript ayudan al desarrollo de escenas en 3D como la Figura 2.8. La multiplataformeidad que ofrece WebGL permite que se pueda ejecutar en cualquier navegador, independientemente de su sistema operativo.

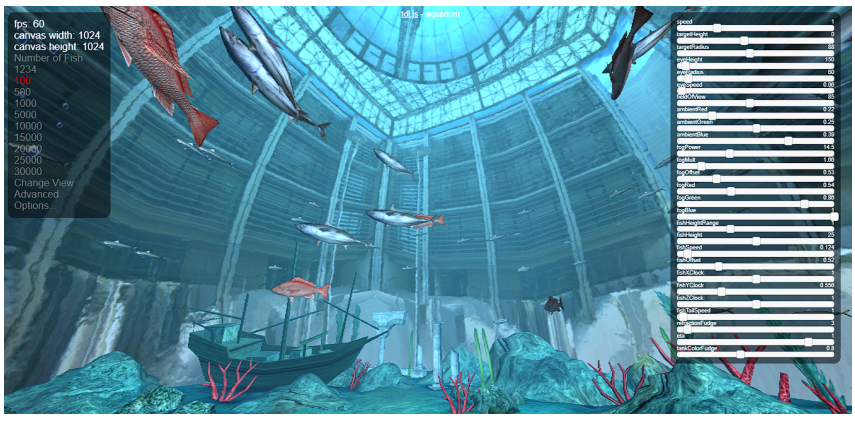


Figura 2.8: Ejemplo de aplicación en WebGL

[17] Khronos Group: `https://www.khronos.org/webgl/`.
[18] Cita según MDN web docs

2.6. HTML

HTML[19] fue inventado por Tim Berners-Lee en el CERN (Organización Europea para la Investigación Nuclear) a principios de la década de 1990. Se buscaba que los científicos pudieran compartir información fácilmente a través de una red, en donde el hipertexto de HTML jugaba una parte esencial.

> *'HTML (HyperText Markup Language) is the set of markup symbols or codes inserted in a file intended for display on a World Wide Web browser page. The markup tells the web browser how to display a web page's words and images for the user.'* [6]

Esta definición de W3C muestra el propósito del HTML. A medida que la web evolucionó, HTML también lo hizo creando diferentes versiones que se iban adaptando al uso de navegadores.

La versión más moderna, el HTML5 [9], es un conjunto de estándares y tecnologías web que ofrece una experiencia más interactiva y eficiente al usuario. Es el lugar donde las aplicaciones cobran vida, y el centro de todo son los navegadores. Cada navegador cuenta con una máquina de JavaScript que es el encargado de la interactividad de la página. Como complemento, se introdujo CSS, que se encarga de la presentación y estilos visuales de las páginas HTML. Para gráficos 3D, HTML5 se une con WebGL, permitiendo renderizado de gráficos avanzados en 3D acelerados por el propio hardware del navegador. Además, introdujo una gran cantidad de características, como el soporte multimedia de audio y video, además de la posibilidad de Canvas para gráficos 2D. Incorporó además un conjunto de APIs para aplicaciones web que permiten funcionalidades avanzadas.

El HTML5 como tal se conoce como lenguaje de marcado, lo que significa que utiliza etiquetas (tags) para estructurar el contenido. Estas etiquetas indican el significado y la presentación de diferentes elementos (encabezados, párrafos, imágenes, enlaces, etc.). Los documentos HTML5 cuentan con una estructura jerárquica basada en el anidamiento de elementos; esto permite introducir elementos o etiquetas dentro de otras, definiendo una relación entre dichos objetos de 'Padre-Hijo'.

[19] HTML W3C [6].

Es importante recordar que la mayoría de las etiquetas usadas en HTML en general se deben cerrar tras su uso, aunque existen excepciones.

2.7. JavaScript

> *'JavaScript es un lenguaje de programación interpretado que permite implementar funcionalidades complejas en páginas web' [19]*

JavaScript [18] [8] fue diseñado originalmente para ejecutarse en el navegador del cliente (front-end). Se caracteriza por la posibilidad de ser ejecutado línea a línea por el navegador, sin tener que compilarse antes de ello, además al ser un lenguaje de alto nivel facilita la escritura y comprensión de código, además también cuenta con un tipado de variables lo que significa que dichas 'variables' pueden cambiar a lo largo de la ejecución.

Actualmente, se rige por el estándar ECMAScript (la última versión es la ES2023[20]); este asegura la interoperabilidad y evolución. Debido a este estándar y el desarrollo de API Web, ha podido transcender al navegador, esto hace que hoy en día también se pueda usar en el lado del servidor (back-end) con entornos como Node.js, así como en desarrollo de aplicaciones móviles y de escritorio.

- **Cliente (front-end):** En el lado del cliente, JavaScript permite manipular el DOM (Document Object Model), gestionar eventos del usuario, y hacer peticiones asincrónicas al servidor (AJAX o Fetch API), lo que lo convierte en una herramienta fundamental para crear experiencias de usuario dinámicas y reactivas.

- **Servidor (back-end):** En el entorno del servidor, JavaScript ha tomado gran relevancia gracias a Node.js, que permite ejecutar código JavaScript fuera del navegador. Se explicará mejor en la seccion 2.9

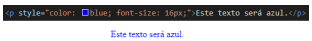
JavaScript juega un papel fundamental en el desarrollo de páginas web dinámicas, dotando al HTML antes mencionado con diferentes posibilidades de acción.

A lo largo del desarrollo del proyecto se ha usado en diferentes escenarios, desde la creación de un servidor simple, hasta la codificación de componentes para A-Frame con la finalidad de dar funcionalismo a escenas en 3D desarrolladas en dicho framework.

[20]Documentación ECMAScript2023: `https://tc39.es/ecma262/2023/`

2.8. CSS

CSS [9] (Cascading Style Sheets) es como la cobertura y la decoración de una página web (el HTML sería el esqueleto). Se usa para decirle al navegador cómo mostrar los elementos de HTML: qué colores usar, qué fuentes, cómo se deben organizar en la pantalla, si deben tener márgenes, bordes, etc. Gracias a CSS, es posible crear interfaces web atractivas, responsivas y coherentes en distintos dispositivos. Para su uso se podrá utilizar de la siguiente forma:

Método	Descripción	Ejemplo Visual
`Inline`	Se aplica directamente en el atributo `style` de un elemento HTML. Útil para aplicar estilos rápidos.	
`Internal`	Se define dentro del bloque `<style>` en la sección `<head>` del HTML.	
`External`	Consiste en enlazar un archivo externo con extensión `.css` mediante `<link>`. Es la forma más escalable.	

Cuadro 2.1: Formas de aplicar CSS en un documento HTML

Esta tecnología se usa particularmente en una sección en la que se une un HTML con estilos dentro de un panel de A-Frame conocido como 'HTML Embed Component' [28].

2.9. Node.js

Node.js [20][21][22] es un entorno de ejecución de JavaScript de código abierto y multiplataforma. Permite ejecutar código JavaScript fuera de un navegador web.

Node.js fue creado por Ryan Dahl y se lanzó inicialmente en 2009. Se estaba buscando una forma más eficiente de manejar conexiones y concurrencia en la web después de experimentar problemas con el servidor web Apache. Se inspiró en lenguajes orientados a eventos y propuso una arquitectura basada en un bucle de eventos no bloqueante. La clave de esto fue la utilización del motor V8 presente en Google Chrome, el cual compila JavaScript de manera rápida y eficiente.

Node.js se centra en una arquitectura orientada a eventos y no bloqueante. Esto significa que, en lugar de esperar a que una operación de entrada/salida (E/S) se complete (bloqueando el hilo), Node.js registra una función de callback que se ejecutará una vez que la operación finalice. Esto permite manejar muchas conexiones simultáneas de manera eficiente con un solo hilo.

Existen problemas con lo anterior mencionado; uno de los más importantes era el anidamiento de callbacks, ya que con un número excesivamente grande de estos, la lectura y mantención de estas se vuelven difíciles. Este problema se mitigó con la llegada de las promesas y las funciones `async/await`. También el uso de tareas complejas podría sobrecargar la CPU y afectar negativamente al hilo principal.

2.9.1. Node.js y el Manejo de Peticiones HTTP

Node.js es excelente para manejar peticiones HTTP, tanto para crear servidores web que responden a las solicitudes de los clientes (navegadores, otras aplicaciones) como para realizar peticiones a otros servidores (APIs externas).

- **Módulo `http` y `https`:** Proporcionan las bases para crear servidores y clientes HTTP/HTTPS de bajo nivel.

- **Frameworks (Express):** Simplifican la creación de servidores web con funcionalidades para enrutamiento, middleware y gestión de solicitudes/respuestas.

- **Clientes HTTP:** Para realizar peticiones a otros servidores, se pueden usar los módulos nativos (`http`, `https`) o bibliotecas de terceros más convenientes como `axios` o `node-fetch`.
- **Manejo de Métodos HTTP:** Node.js facilita el manejo de diferentes métodos HTTP (`GET`, `POST`, `PUT`, `DELETE`, etc.) para construir APIs RESTful.
- **Streams:** Node.js utiliza *streams* para manejar grandes cantidades de datos de manera eficiente durante las peticiones y respuestas, evitando cargar todo en la memoria.

2.9.2. Certificados SSL/TLS

Para asegurar las comunicaciones a través de la web (HTTPS), Node.js permite trabajar con certificados SSL/TLS. Estos certificados son credenciales digitales que cumplen una función fundamental en la seguridad de las comunicaciones por internet. Son la base sobre la que se construye la confianza en línea y la protección de datos. Estos certificados constan:

- **Clave privada (.key) :** Archivo que se usa para encriptar la información que el servidor envía y desencripta la información que reciba.
- **Clave pública (.crt) :** Archivo que contiene la clave pública del servidor y la información de identificación.
- **Certificado de Autoridad(.ca, .pem o .crt):** Archivos que son necesarios para construir una cadena de confianza entre el usuario y el servidor, deben estar firmados por una autoridad de certificación, pero para pruebas locales puede ir autofirmado.

2.10. Aplicaciones relacionadas

En este apartado se mencionarán aplicaciones que, desde algunos puntos de vista, están relacionadas con el proyecto o han servido de inspiración para el desarrollo del proyecto.

2.10.1. Arkio

Arkio [4] es una herramienta de diseño espacial colaborativo que permite a las personas crear y colaborar utilizando Realidad Virtual (VR), así como en PC, tablet y móvil.

Figura 2.9: Ejemplo de escena de Arkio [3]

- Se centra en el diseño de interiores, edificios, espacios virtuales y planificación urbana.
- Facilita la colaboración multiplataforma en tiempo real como se ve en la Figura 2.9.
- Dispone de plugins para integrar con herramientas de diseño 3D como `Revit`, `Rhino` y `SketchUp`, permitiendo importar modelos existentes y exportar el trabajo realizado en Arkio.
- Arkio permite a los usuarios dibujar y manipular elementos arquitectónicos de forma intuitiva en el espacio VR mediante el uso de los mandos de que traen las gafas como la Quest 3.

2.10.2. VR Sculpting

VR Sculpting con Comandos de Voz: En la escultura VR (como en Shapelab[12] Figura 2.10), se han explorado o implementado (a menudo a través de plugins o herramientas de terceros como VoiceAttack) comandos de voz para tareas específicas durante el proceso de modelado.

Estos comandos no forman parte nativa del software, pero permiten extender su funcionalidad, especialmente en contextos de accesibilidad, productividad o flujo de trabajo manos libres. Por ejemplo, podrías usar la voz para:

- Seleccionar herramientas.
- Modificar parámetros.
- Realizar acciones.

Si bien esto no es la creación del objeto desde cero con la voz, sí permite una manipulación y modificación significativa mediante comandos hablados.

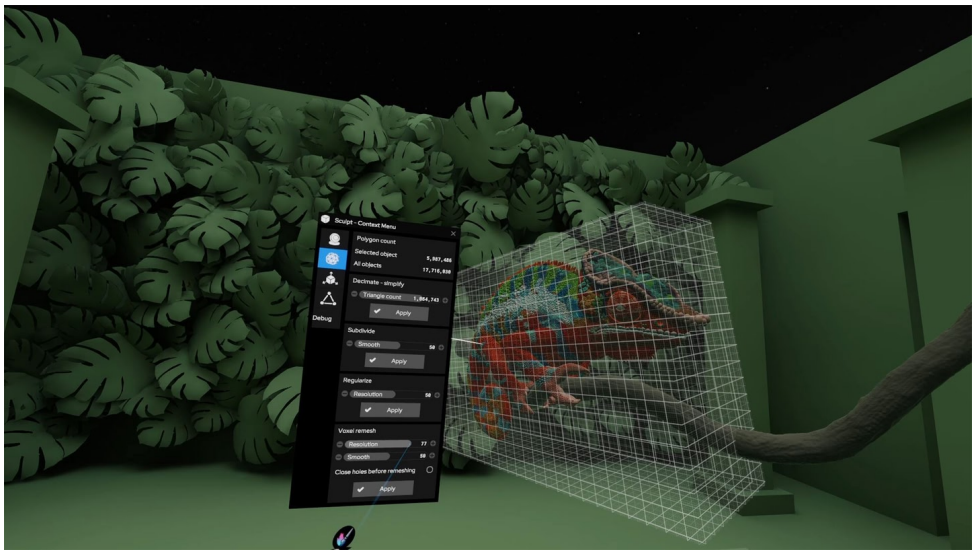


Figura 2.10: Ejemplo de uso de Shapelab [25]

2.10.3. Bridge Crew (Star Trek)

Inicialmente, se añadió soporte para comandos de voz mediante la integración con IBM Watson, permitiendo a los jugadores emitir órdenes verbales a la tripulación en misiones para un solo jugador. Esta funcionalidad estaba disponible exclusivamente en inglés y requería el uso de un micrófono durante el juego, como se puede ver en la Figura 2.11. **Ubisoft**[31] desactivó esta característica debido a la finalización del contrato con IBM, lo que resultó en la eliminación del soporte de comandos de voz en todas las plataformas.

Figura 2.11: Ejemplo de uso de voz en Star Trek Bridge Crew [26]

Estas aplicaciones sirvieron como inspiración tanto por sus mecanismos de creación de objetos como por su capacidad para detectar y ejecutar comandos por voz. Este proyecto fue desarrollado con el objetivo de aprovechar esas funcionalidades, integrándolas en una sola herramienta accesible. El resultado en este proyecto busca combinar ambas facetas, creación y control por voz, en una plataforma accesible desde cualquier navegador web.

2.11. Tecnologías auxiliares

2.11.1. GitHub

GitHub[21] [10] se ha consolidado como la plataforma líder para el desarrollo colaborativo de software. Basándose en el sistema de control de versiones de Git, ofrece un entorno robusto para que desarrolladores de todo el mundo puedan trabajar simultáneamente en proyectos, gestionando los cambios de código de manera eficiente y organizada.

Su principal función radica en el alojamiento de repositorios, que actúan como almacenes centrales para el código fuente de un proyecto. Git permite rastrear cada modificación realizada en el código a lo largo del tiempo, facilitando la reversión a versiones anteriores, la identificación de la autoría de los cambios y la fusión del trabajo de diferentes colaboradores. Las pull

[21] GitHub: `https://github.com/`

requests (solicitudes de extracción) sirven como mecanismo para proponer cambios, iniciar discusiones sobre el código y, finalmente, integrarlos a la rama principal.

Gracias a estas funcionalidades se ha ido desarrollando poco a poco el proyecto, del que se está hablando en este documento. Si bien no se explotó toda la funcionalidad que GitHub aporta, ha sido de gran ayuda para la recuperación de versiones anteriores y guardar avances.

2.11.2. Visual Studio Code

Visual Studio Code[22][15] es un editor de código fuente con un gran equilibrio entre ligereza y potencia. Se distingue por su rendimiento rápido y su capacidad para manejar proyectos de gran envergadura sin comprometer la fluidez. Una de sus características más destacadas es su soporte integrado para una amplia gama de lenguajes de programación, incluyendo JavaScript y Python, entre otros.

Para cada lenguaje, VS Code ofrece funcionalidades específicas como el resaltado de sintaxis, que facilita la lectura del código, además de constar con un sistema de autocompletado inteligente que sugiere código, muestra información sobre funciones y parámetros, y ayuda a prevenir errores.

La depuración integrada es otra funcionalidad clave, permitiendo a los desarrolladores ejecutar y analizar su código directamente desde el editor. Se pueden establecer puntos de interrupción, inspeccionar el valor de las variables y seguir la ejecución paso a paso para identificar y corregir errores de manera eficiente.

El Marketplace de extensiones ofrece una vasta colección de herramientas. En este proyecto se usaron extensiones como Live Server y Live Preview para visualizar los cambios en tiempo real que se hacen tanto en JavaScript como en el cuerpo del archivo de A-Frame.

2.11.3. Gafas Meta Quest 3

Las gafas de realidad virtual (VR) y realidad mixta (MR) de Meta llamadas Quest 3[14]. Ofrecen experiencias inmersivas en VR y la capacidad de superponer elementos virtuales en el mundo real (MR).

Cuenta con un potente procesador llamado Qualcomm Snapdragon XR2 Gen 2. Integra dos

[22]Visual Studio Code: `https://code.visualstudio.com/`

camaras RGB de 4MP (megapixeles) para un passthrough a color en alta definicion. Además, mejora la inmersion visual con una resolucion de pantalla de 2064 x 2208 píxeles por ojo, mostrando una imagen mas nitida.

Figura 2.12: Gafas Meta Quest 3

Se utilizaron para hacer las pruebas de funcionamiento de la aplicación en los navegadores de Meta, pero contaba con ligeros problemas de compatibilidad para el uso de APIs que se usan en navegadores normales.

2.11.4. LaTeX

LaTeX [29], por otro lado, es un sistema de composición de textos de alta calidad, ideal para documentos técnicos. Se centra en la estructura y el contenido, dejando el formato en manos del sistema.

LaTeX automatiza la generación de elementos estructurales como la numeración de secciones, subsecciones, figuras y tablas. También se encarga de la creación automática de índices, bibliografías y referencias cruzadas, asegurando la coherencia y precisión en todo el documento. La salida final se caracteriza por su alta calidad tipográfica, gracias a los algoritmos de composición y al uso de fuentes profesionales. Es usado para el desarrollo de la memoria del

trabajo de fin de grado. Se tomó esta elección debido a que permite centrarse únicamente en el contenido textual, mientras que el sistema se encarga del diseño y formato del documento.

Capítulo 3

Desarrollo del proyecto

El desarrollo del proyecto se ha inspirado conceptualmente en las metodologías ágiles. Para su descripción se han utilizado algunos conceptos de **Scrum**[1], para su explicación y seguimiento del progreso.

Aunque para el desarrollo de este proyecto no se contó con un equipo de trabajo amplio, ni sea un proyecto real, la responsabilidad de Scrum Master fue asumida por el tutor del TFG, quien se encargó de definir los requisitos y evaluar los incrementos iterativos. El equipo de desarrollo ejecutó las tareas técnicas, haciendo avanzar el proyecto de forma incremental a través de una serie de sprints.

Esto facilitó la gestión del proyecto mediante ciclos iterativos conocidos como sprints. La progresión a un sprint posterior dependía del logro exitoso de los objetivos del sprint precedente. Estos sprints a su vez se dividen en tareas mucho más pequeñas que persiguen objetivos más concretos y ayudan a la consecución del objetivo final del sprint.

Estructura mínima de cada sprint:

- Objetivos
- Tareas Realizadas
- Resultado
- Lecciones aprendidas

[1] Guía de Scrum oficial [27].

3.1. Estructura del proceso de desarrollo

- **Sprint 1: Investigación y Prototipado Inicial (3 semanas)**
  - Familiarización con las tecnologías de reconocimiento de voz y seleccionar una para uso en el navegador.
  - Familiarización con la tecnología de A-Frame.
  - Construir la primera demo con reconocimiento de voz sobre A-Frame.
- **Sprint 2: Familiarización con el desarrollo en A-Frame Base (2 semanas)**
  - Integrar funcionalidades de interacción por voz en una escena de A-Frame.
  - Permitir la creación de objetos primitivos mediante comandos de voz.
  - Diseñar y desarrollar componentes reutilizables.
  - Desarrollo de una demo condensando todos los cambios.
- **Sprint 3: Funcionalidades y Componentes (2 semanas)**
  - Creación de funcionalidades para los componentes.
  - Aumento de comandos para la funcionalidad del proyecto.
  - Creación de escena simple para funcionalidad en escritorio.
- **Sprint 4: Mejora de componentes y Mejora Visual (2 semanas)**
  - Crear componentes para editar y eliminar objetos.
  - Modificar la escena para un mejor uso del usuario.
  - Crear comandos nuevos para la funcionalidad de los nuevos componentes.
  - Mejora de la estética y colocación de objetos.
- **Sprint 5: Integración Quest 3 (1 semana)**
  - Explorar la posibilidad de integración del proyecto con dispositivos VR/AR, específicamente las gafas Quest 3.
  - Creación de servidor propio para peticiones de transcripción.

- **Sprint 6: Demo y unión final (3 semanas)**
  - Mejora de la estética y colocación de objetos.
  - Que los elementos que aparezcan en la escena y proporcionen información al usuario sean fáciles de ver.
  - Maquetado final de la demo para escritorio.
  - Maquetado y unión del back-end y front-end para demo con gafas Quest 3.

3.2. Sprint 1

Con este sprint, se tuvo la primera toma de contacto con el servicio de reconocimiento de voz y del uso complejo en A-Frame. Además, se contó con diferentes programas que se fueron desarrollando en el transcurso de este sprint. Estos programas se irán explicando a continuación.

3.2.1. Objetivos

- Familiarización con las tecnologías de reconocimiento de voz y seleccionar una para uso en el navegador.
- Familiarización con la tecnología de A-Frame.
- Construir la primera demo con reconocimiento de voz sobre A-Frame.

3.2.2. Tareas Realizadas

Tarea 1: Explorar distintas tecnologías de reconocimiento de voz y su funcionamiento.

Tras la recolección y análisis de información técnica relevante. Para ello, se harán pruebas con distintos sistemas de transcripción de voz. Y finalmente, se desarrollaron 3 ejemplos de cómo debía realizarse la funcionalidad para generar una transcripción, disponibles en un repositorio de GitHub[2].

1. Reconocimiento de Voz en Python con Whisper

[2]Ejemplo disponible en GitHub: `https://github.com/R4CC00N/SpeechRecognition_in_Browser/blob/main/Sprint2`.

Se utilizó el modelo Whisper de OpenAI en local para transcripción de audio. Este programa se llevó a cabo mediante Python, grabando la voz mediante el micrófono, y guardando el audio para luego usar Whisper en él y obtener una transcripción.

2. Servidor híbrido Node.js + Python

Para el segundo caso, se usó un servidor propio creado a base de Node.js que se comunicaba con el mismo script de Python en tiempo real para realizar las transcripciones del mismo modo que en el caso anterior. Este caso se desarrolló para poder implementar la transcripción de voz en el navegador y desarrollar la idea principal del proyecto. La transcripción obtenida se transmitiría al cliente web a través de WebSockets.

3. Reconocimiento en el navegador (Web Speech API)

Como caso de uso final se encontró una API dedicada a los navegadores, Web Speech API dio paso a la transcripción rápida y efectiva. Es necesario recalcar que depende mucho de la calidad de audio, y el navegador. Dio unos buenos resultados, además de que, en el contexto del proyecto, cumplía los estándares que se intentaban alcanzar con el reconocimiento y transcripción de voz.

Tarea 2: Investigar componentes de A-Frame que se puedan utilizar.

Para este punto se planteó el primer contacto con el framework haciendo uso de `<a-text>` para observar los cambios en la escena producidos por la transcripción de voz. Además de observar las otras primitivas dentro del framework disponibles junto con sus características y posibles usos.

Tarea 3: Desarrollo y descripción del prototipo final.

Para condensar toda la información, se realizó una demo de Reconocimiento de Voz en A-Frame disponible en GitHub[3]. En esta demo se trabajó con el reconocimiento de voz desde el navegador (usando 'Web Speech API'). De la tal manera que al pulsar el botón principal que aparece en la Figura 3.1 se activa el evento de detección de voz que obtendrá permisos para el uso del micrófono y este al detectar al usuario hablar transcribe la voz, todo esto dentro del mismo navegador, gracias al componente **`input-text`**. Tras la obtención de la transcripción,

[3]Demo de reconocimiento de voz disponible en GitHub: `https://github.com/R4CC00N/SpeechRecognition_in_Browser/blob/main/Sprint3/v2_A-Frame.html`.

esta aparece automáticamente en el elemento `<a-text>` dentro de la escena. La experiencia permite mostrar mensajes de voz directamente dentro de un entorno 3D. Si se detectan palabras como crear, editar u otro texto, el fondo cambia de color.

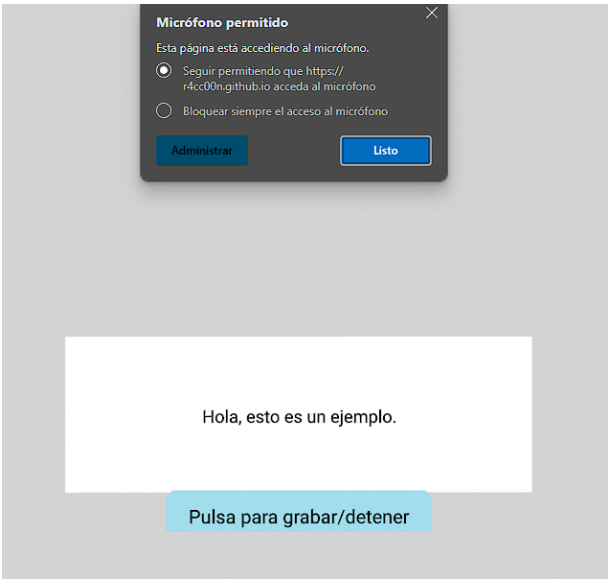


Figura 3.1: Funcionamiento de la demo de reconocimiento de voz en A-Frame

3.2.3. Resultados y Lecciones aprendidas

- Para conseguir el objetivo de elección de tecnología de reconocimiento de voz se ha optado por la opción de Web Speech API ya que fue la más eficiente y directa para pruebas en escritorio. También tuvo varias cosas negativas, la principal fue que en el uso de la detección y transcripción de voz en gafas de VR/AR, como son las Quest 3, no fuesen soportadas en el navegador de Meta. No obstante, esta incompatibilidad de Web Speech API con navegadores de gafas como la Quest 3 llevó a plantear dos líneas de trabajo: una demo basada en navegador para escritorio, y una segunda demo soportada por back-end para su funcionamiento en gafas de realidad mixta, empleando Node.js y alguna API de reconocimiento de voz en el servidor.
- Para completar los objetivos de familiarización con A-Frame y el desarrollo de una demo funcional se creó la Demo de Reconocimiento de Voz en A-Frame y una vez conocidos los elementos a interactuar, el añadir un factor como el reconocimiento de voz fue un desafío de grado medio. Se encontró que la transcripción podía ser usada directamente

para el cambio de valor dentro del `<a-text>`.

3.3. Sprint 2

Este sprint contiene demos interactivas en las que se combina A-Frame con reconocimiento de voz para crear objetos 3D dentro de una escena usando comandos hablados. Cada demo mejora la anterior, especialmente en el reconocimiento y manejo de coordenadas y números, permitiendo finalmente la creación precisa de objetos con posición y tamaño definidos por voz. Además del incremento y mejoras en los componentes de A-Frame.

3.3.1. Objetivos

- Integrar funcionalidades de interacción por voz en una escena de A-Frame.
- Permitir la creación de objetos primitivos mediante comandos de voz.
- Diseñar y desarrollar componentes reutilizables.
- Desarrollo de una demo condensando todos los cambios.

3.3.2. Tareas Realizadas

Tarea 1: Comando Básico y Reconocimiento Inicial

En esta demo se creó un componente llamado **`input-text`** que, aparte de transcribir el audio, también detectará comandos de voz. Estos comandos serán procesados de la transcripción, como se puede ver en la Figura 3.2. Se trabajó principalmente con el comando 'crear [objeto]', este permite la generación de cubos, solicitando a continuación la 'posición' y el 'tamaño' del objeto.

Sin embargo, presenta ciertas limitaciones, una de ellas es la fiabilidad en el reconocimiento numérico (valores numéricos que la transcripción decide automáticamente poner como letras e incapacidad para interpretar valores negativos en las transcripciones). Otra de las limitaciones que se encontraron es la captura de coordenadas como un bloque único.

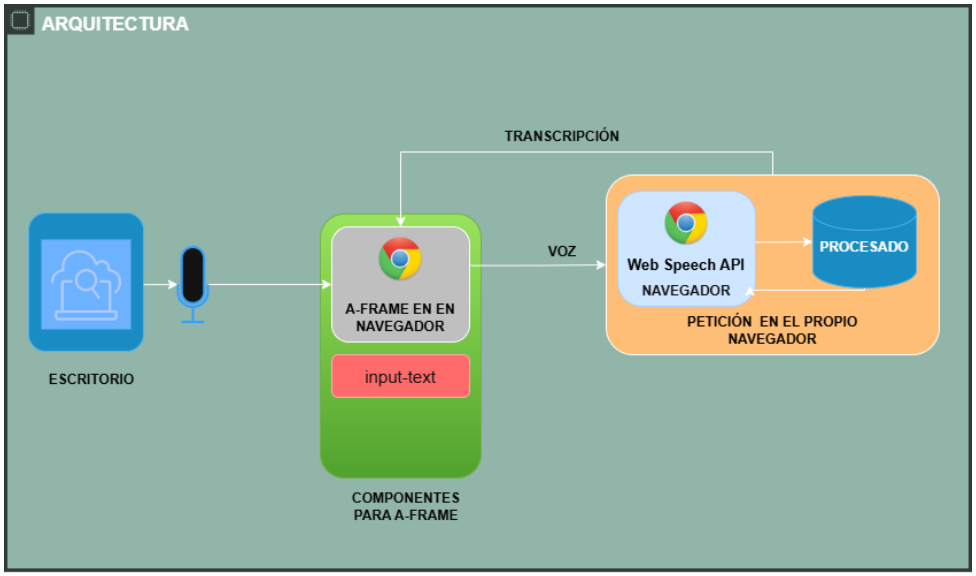


Figura 3.2: Flujo de creación de componentes

Tarea 2: Creación de Componentes en A-Frame y Reestructuración de Código

Este sprint se centró en modularizar la escena en A-Frame mediante la creación de componentes reutilizables como se ve en la Figura 3.3, en ella se puede ver:

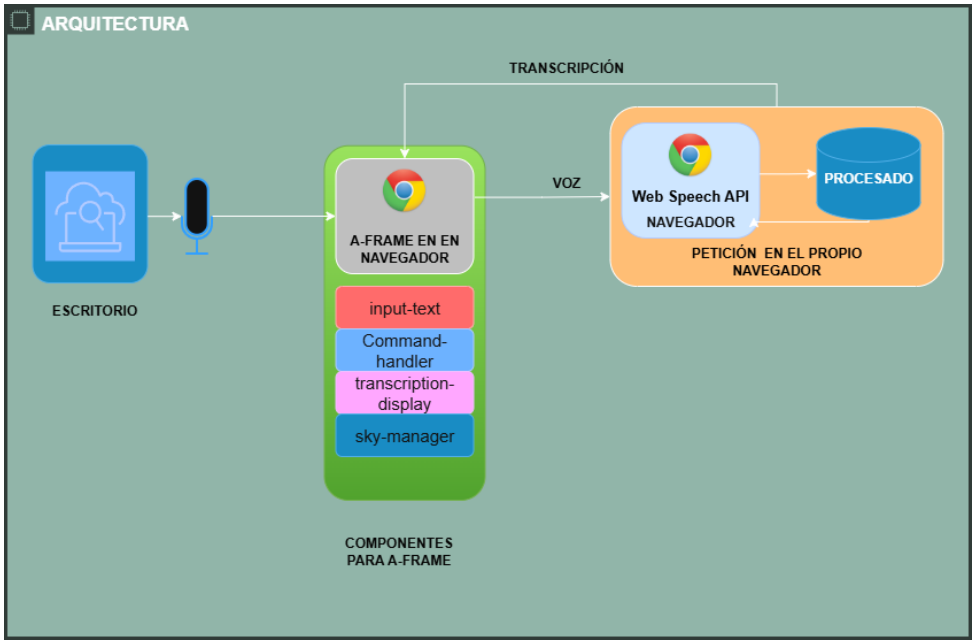


Figura 3.3: Flujo de creación de componentes avanzados

command-handler: Es un componente que interpreta comandos de voz para crear objetos en la escena, pero con una entrada de audio configurable. Depende de que otro componente en

la escena realice el reconocimiento de voz y emita un evento 'transcript' con el texto transcrito. Durante su funcionamiento, emite eventos a nivel de escena (enter-create-mode, exit-create-mode) para señalar el inicio y el fin del proceso de creación.

Además de aplicar una reestructuración del código con el fin de mejorar la organización, escalabilidad y mantenibilidad del proyecto.

transcription-display: Tiene la función de mostrar en un elemento de texto 3D con un ID definido (`<a-text>`) la transcripción de voz proveniente de otro componente de la escena. Se puede observar su uso en la Figura 3.4 en donde aparecen los mensajes para el usuario en la parte superior de la caja de texto normal.

sky-manager: Se encarga de gestionar el color del fondo de la escena (`<a-sky>`) en respuesta a eventos personalizados emitidos a nivel de la escena provenientes del `command-handler`.

Añadido a estos componentes, se dedidió separar el código de JavaScript y el HTML, para organizar así mejor los archivos y la estructuración del proyecto. También se han solucionado los problemas con los números creando un manejador de números.

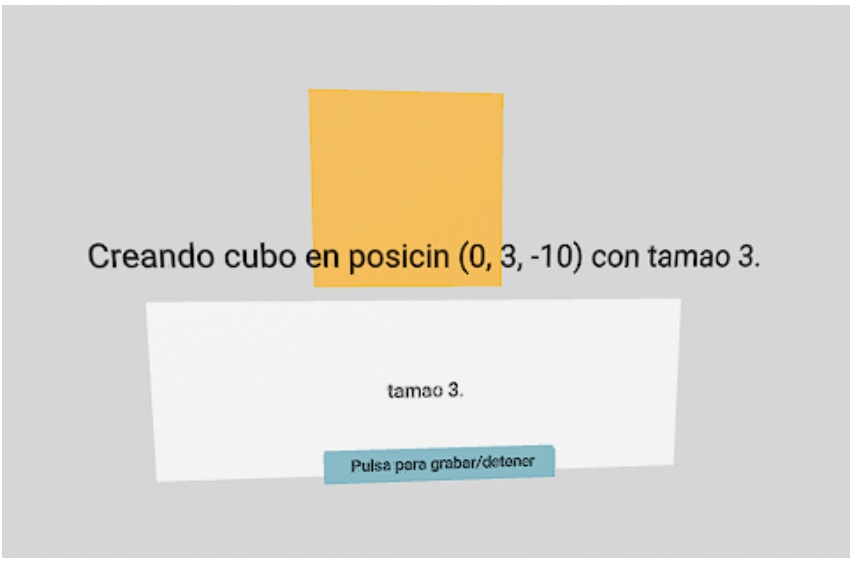


Figura 3.4: Funcionamiento del comando de creacion de objetos básico

3.3.3. Resultados

- Se logró una implementación funcional del reconocimiento de voz para la creación de objetos 3D en un entorno A-Frame.

- Se demostró una mejora progresiva en la capacidad del sistema para comprender y procesar comandos de voz relacionados con la creación, posición y tamaño de objetos.
- La interacción para especificar la posición de los objetos evolucionó de una entrada de bloque (123) a una solicitud secuencial (x=1,y=2,z=3), mejorando la usabilidad.
- En la última demo, se implementó con éxito la interpretación de números tanto en formato numérico como en palabras, incluyendo valores negativos, lo que amplía la flexibilidad de la entrada por voz.
- Se proporcionó retroalimentación visual constante al usuario sobre el estado de la grabación y el texto reconocido a través de la interfaz en la escena.
- Se mejoró la legibilidad del texto al separar en diferentes archivos el JavaScript y el HTML.

3.3.4. Lecciones aprendidas

- La creación de componentes en A-Frame facilita la organización y la reutilización de la lógica de la aplicación.
- La interpretación del lenguaje natural, incluso para tareas específicas, requiere una consideración cuidadosa de las posibles formas en que los usuarios pueden expresar sus comandos. El manejador para números es un ejemplo de cómo abordar esta complejidad.
- La retroalimentación visual es esencial para mantener al usuario informado y para depurar el proceso de reconocimiento e interpretación de voz.
- La iteración y la mejora progresiva, como se ve en la secuencia de los cuatro demos, es un enfoque efectivo para abordar desafíos técnicos.

3.4. Sprint 3

En este sprint se desarrollaron las capacidades del proyecto, añadiendo más modificadores, creando componentes e incluso mejorando la división del código. Durante el desarrollo de este sprint se crearon componentes para el accionar del proyecto.

3.4.1. Objetivos

- Creación de funcionalidades para los componentes.
- Aumento de comandos para la funcionalidad del proyecto.
- Creación de escena simple para funcionalidad en escritorio.

3.4.2. Tareas Realizadas

Creador de objetos funcionalidades

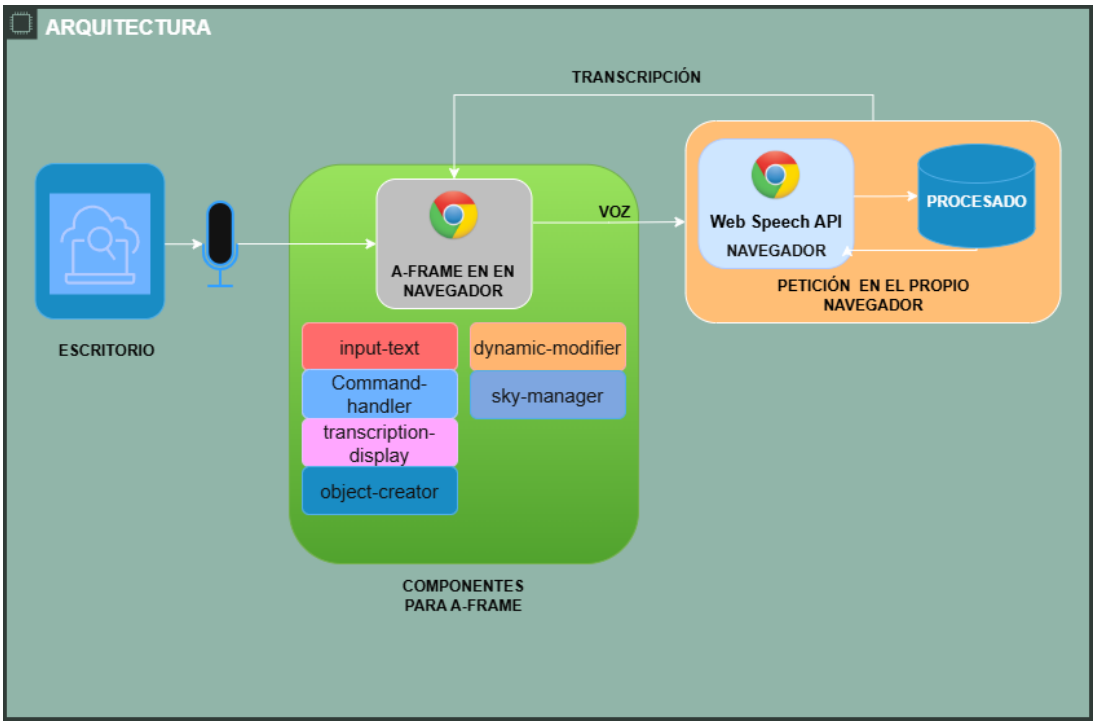


Figura 3.5: Flujo de creación de componentes avanzados

object-creator: Escucha el evento start-object-creation y, basándose en el tipo de objeto recibido, crea la entidad con una geometría predefinida, una posición y rotación fijas.

dynamic-modifier: Este componente edita en tiempo real el objeto creado con diferentes funcionalidades de modificación Paso a Paso (stepsPOS, stepCOLOR, stepID, stepSIZE) para manejar la modificación de cada atributo de forma conversacional.

Se introduce una variable global step (inicializada en null). Esta variable se utiliza para gestionar un flujo de conversación paso a paso durante la modificación de los atributos (posición, tamaño, color, ID).

La palabra clave 'cambio' ahora se utiliza para salir del modo de modificación. Este cambio envía un mensaje al usuario que se utiliza para guiarlo a través del proceso de modificación.

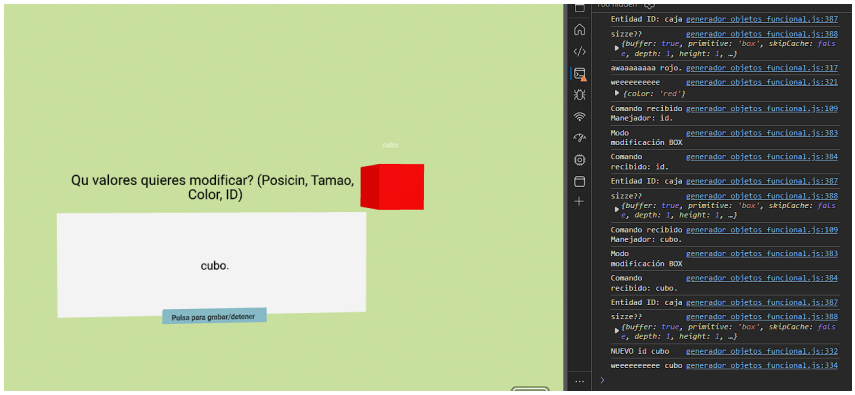


Figura 3.6: Funcionamiento del generador de objetos con funcionalidades extra

3.4.3. Resultados

- Se cumplió con el aumento de comandos y funcionaidades.

- Por último, la creación de una escena simple condensando todos estos detalles se puede encontrar en GitHub[4], en donde se pueden ver la aplicación de los comandos antes mencionados.

3.4.4. Lecciones aprendidas

Componentes para modularidad: Organizar la lógica en componentes facilita el desarrollo y mantenimiento.

Gestión de estados en voz: Las interacciones conversacionales requieren un seguimiento claro del estado.

Asincronía de la voz: La transcripción y respuesta necesitan manejar la latencia.

Comandos de voz intuitivos: El lenguaje claro y la guía mejoran la experiencia del usuario.

[4]**Demo Final del Sprint** `https://r4cc00n.github.io/SpeechRecognition_in_Browser/Sprint4/Create_Obj_v3.html`

3.5. Sprint 4

En este sprint se dieron la mayoría de los avances en la creación de componentes, como se puede ver en la Figura 3.7. Pudiendo así tener una estructura de componentes variados, con su respectiva funcionalidad, que se mantendrían hasta el final del desarrollo.

3.5.1. Objetivos

- Crear componentes para editar y eliminar objetos.
- Modificar la escena para un mejor uso del usuario.
- Crear comandos nuevos para la funcionalidad de los nuevos componentes.
- Mejora de la estética y colocación de objetos.

3.5.2. Tareas Realizadas

Nuevas funcionalidades

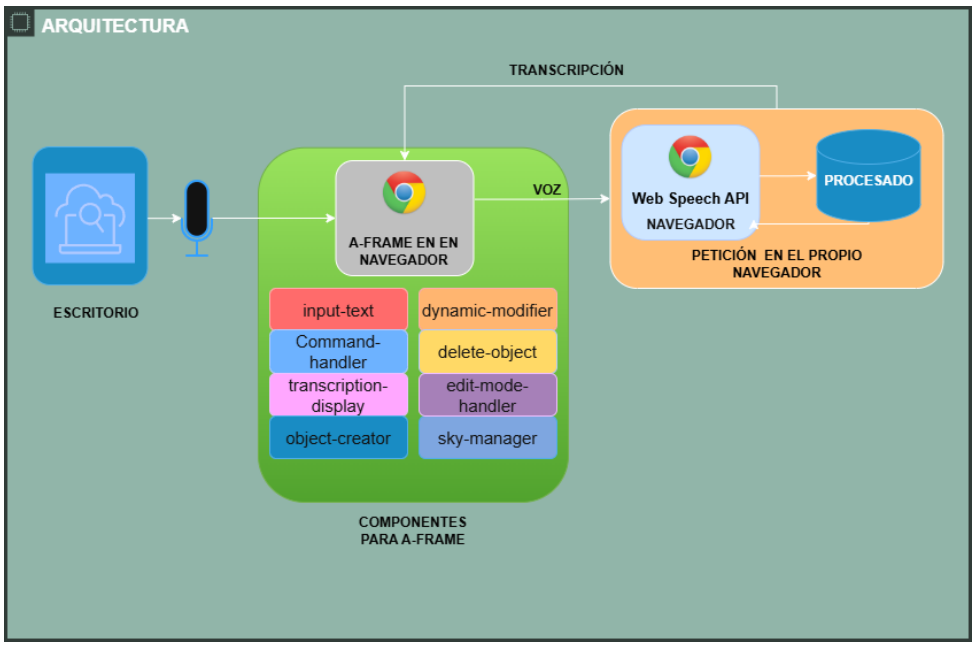


Figura 3.7: Flujo de creación de componentes avanzados

Esta demo es la más completa de todas; se agrupó la funcionalidad presentada anteriormente y añadiendo unas mejoras de las que se hablará a continuación. Primero se añadió la lógica para

crear diferentes tipos de objetos (cubo, esfera, plano, luz, y assets como contenedor y radio) al recibir comandos de voz. Se agregaron nuevos componentes con funcionalidades diversas:

- **delete-object:** Este componente maneja la lógica para eliminar objetos de la escena mediante comandos de voz, solicitando el ID del objeto a eliminar como se ve en la Figura 3.8.

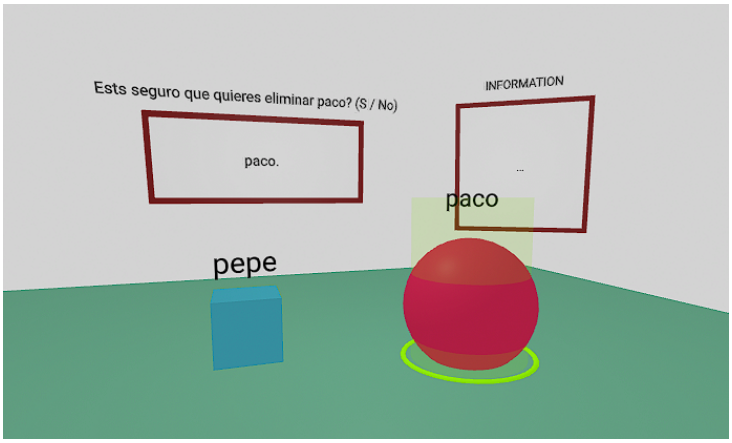


Figura 3.8: Funcionamiento de eliminar objetos con ID

- **edit-mode-handler:** Este componente gestiona el modo de edición de objetos existentes en la escena, según el ID del objeto, como se ve en la Figura 3.9.

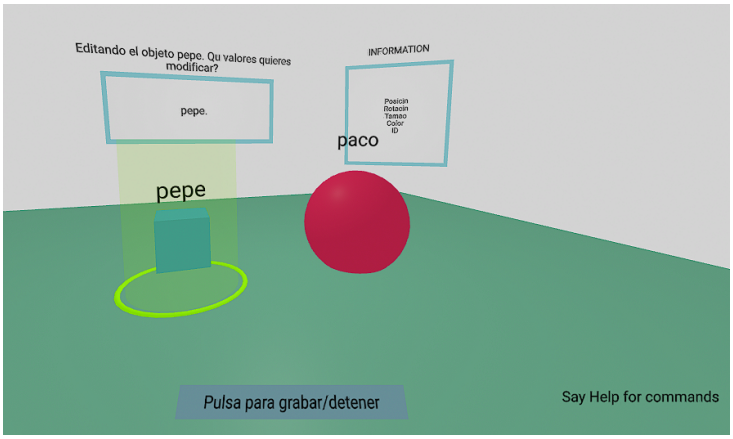


Figura 3.9: Funcionamiento de editar objetos con ID

- **object-creator:** Este componente se encarga de instanciar los diferentes tipos de objetos en la escena basándose en los eventos recibidos, como se ve en la Figura 3.10.

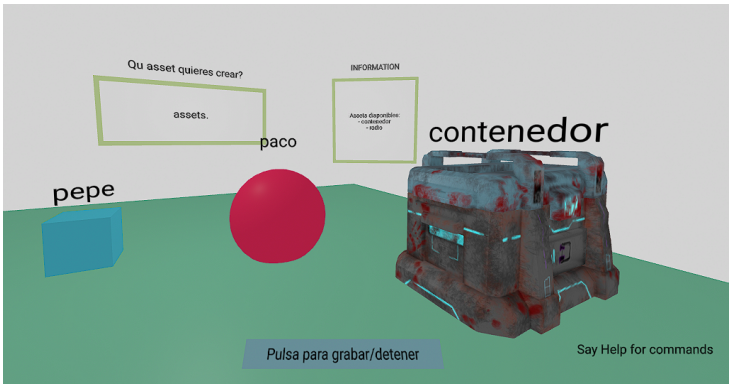


Figura 3.10: Funcionamiento del generador de assets

Además, se definen varias funciones utilitarias (updateUserMessage, stepsPOS, stepsROT, stepCOLOR, stepID, stepSIZE, getEntityById, getLastCreatedEntity, createTorus) para facilitar la manipulación de la escena y los objetos.

También se cambiaron y agregaron comandos nuevos al uso del command-handler, así como crear, editar, ayuda, salir. Se añadieron paneles nuevos para añadir información extra que puede ser útil al usuario tanto escrita como visual, como se puede ver en los bordes de los paneles que cambian de color, dependiendo del estado en el que se encuentre la aplicación (verde=crear, rojo=eliminar...).

3.5.3. Resultados

Gestión de nuevos comandos: Integración de creación, edición y eliminación por voz mediante componentes dedicados (delete-object, edit-mode-handler, object-creator) y modificación avanzada con 'dynamic-modifier'.

Cambio en la estética y colocación de objetos: Se modificó el HUD para hacerlo más agradable visualmente y de ayuda para el usuario, creando así una escena sencilla.

3.5.4. Lecciones aprendidas

Identificación de objetos: Es crucial darle un nombre a los objetos para luego poder dar uso de las funciones de edición y eliminación específicas.

Reutilización de código: Las funciones utilitarias evitan la repetición.

Retroalimentación al usuario: Informar sobre los comandos y el estado es esencial.

3.6. Sprint 5

Este sprint se enfocó en la funcionalidad de la aplicación en gafas tipo Meta Quest 3. Para lograrlo, se abordaron dos aspectos clave: primero, al ver cómo se puede integrar en navegadores de meta, esto llevó a una restructuración de código. Por otro lado, cuando se observó que la API con la que se había trabajado hasta el momento no funcionaba en este tipo de navegadores, se decidió optar por la creación de un servidor privado para el reconocimiento de voz.

3.6.1. Objetivos

- Explorar la posibilidad de integración del proyecto con dispositivos VR/AR, específicamente las gafas Quest 3.
- Creación de servidor propio para peticiones de transcripción.

3.6.2. Tareas Realizadas

En primer lugar, se quería realizar este proyecto íntegro en gafas VR/AR Quest 3. Pero la API de reconocimiento de voz integrada en navegadores no fue soportada por el navegador de Meta, entonces se volvió a replantear el flujo para obtención de la transcripción por voz. Se probó con tecnologías diferentes de reconocimiento de voz, refactorizando el componente que se encargaba de reconocimiento de voz dentro de este proyecto de A-Frame. Pero no se encontró ninguna solución, ya que no soportaba la detección del audio.

Por ello, se creó un servidor privado que fuese capaz de manejar los audios obtenidos del cliente y luego procesarlos y mandarlos a una API de Speech-to-text llamada Assembly AI, que devolverá la transcripción al servidor y este finalmente al cliente por medio del servidor. Esta fue la parte desafiante del sprint. Ya se había manejado un servidor con anterioridad, pero este manejaba unos procesos simples y todo ocurría en el mismo ordenador, pero a la hora de conectarse otros dispositivos, dependía de permisos y certificaciones para hacer esta conexión posible.

La implementación de un servidor HTTPS funcional presentó un nuevo reto, ya que se necesitaba garantizar la conexión desde otros dispositivos dentro de la red local. Para ello, se abordaron los siguientes aspectos técnicos:

-Certificado SSL: SSL/server.key (clave privada) y SSL/server.crt (certificado público). En los navegadores aparecerá como “no seguro” si es autofirmado.

-Configurar un servidor HTTPS

-Acceso desde otros dispositivos en red local: Ahora la conexión deberá hacerse al servidor, entonces la URL deberá apuntar a la IP del servidor que esté en la misma conexión de internet y con el puerto correspondiente (ej. `https://«IP»:8080/`).

3.6.3. Resultados

Tras la creación de certificados y configuraciones, se pudo conectar el servidor con las gafas de VR/AR Quest 3. El problema era la detección del micrófono, que solo se hacía si el servidor tenía los permisos para HTTPS.

Como resultado, se consiguió la interacción completa esperada: el cliente envía el audio al servidor, este lo reenvía a Assembly AI para su transcripción, y finalmente el texto transcrito es devuelto al cliente para su visualización.

También hay que recalcar que para el uso de algunos complementos nos tocó usar una versión de A-Frame pasada, para poder usar el desplazamiento por la escena.

3.6.4. Lecciones aprendidas

- Las limitaciones de plataformas específicas (como navegadores embebidos en dispositivos VR) pueden requerir rediseños profundos en la arquitectura del sistema.
- Contar con un servidor propio proporciona flexibilidad y control, pero también exige cumplir con requisitos de seguridad como HTTPS para trabajar con APIs modernas o acceder a funciones sensibles como el micrófono.
- La comunicación segura entre dispositivos en red local es posible incluso con certificados autofirmados, siempre y cuando se configuren correctamente.

3.7. Sprint 6

En este sprint se llevaron a cabo los últimos ajustes para la implementación final tanto en escritorio como para gafas VR/AR Quest 3.

Se tratará de mejorar la usabilidad; uno de los problemas que se encontró es que los carteles o elementos que debía ver el usuario quedaban en posiciones en las que el usuario no podía ver, por lo que se tratará de arreglar este problema. También otro problema presentado es la limitación que tienen los `<a-text>` para mostrar información.

3.7.1. Objetivos

- Mejora de la estética y colocación de objetos.
- Que los elementos que aparezcan en la escena y proporcionen información al usuario sean fáciles de ver.
- Maquetado final de la demo para escritorio.
- Maquetado y unión del back-end y front-end para demo con gafas Quest 3.

3.7.2. Tareas Realizadas

Look At

Para que el usuario pueda observar bien la información se investigó y aplicó un componente a los objetos de interés para que siempre estén apuntando al usuario (la cámara de la escena); de esta manera, independientemente de la posición del usuario en la escena, podrá observar la información que se desea mostrar. Esto se realizó con un **`look-at`**.

Mejora de la estética

Como se quería introducir más información en la escena se consideró usar un componente extra llamado **`html embed`** que permite incrustar HTML con todas sus riquezas, incluidas CSS, en este caso en paneles de A-Frame, quedando de la forma que se ve en la Figura 3.11.

Figura 3.11: Funcionamiento de htmlembed

Maquetado y unión

Durante este sprint se añadieron también nuevos componentes al funcionamiento del proyecto, unos para la creación de objetos que llevasen otros componentes incrustados, de tal manera que con un solo componente se pudiese estructurar todo. Por otro lado, también se agregaron componentes para nuevos comandos tales como Guardar y Cargar escenas Figura 3.12.

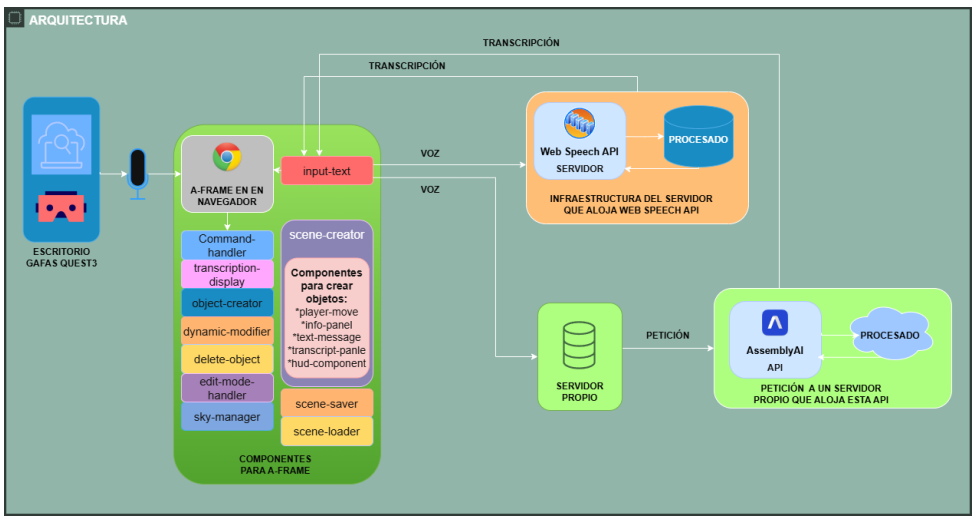


Figura 3.12: Componentes finales agregados

Finalmente, se hizo la inclusión de las demos para escritorio y gafas Quest 3, donde el cambio más significativo entre ellas es el componente **`input-text`**, el cual en la versión de escritorio usa WebSpeechAPI y en la versión de AR/VR se usará un servidor conectado a AssemblyAI.

3.7.3. Resultados

- Para cumplir con el objetivo de la mejora de la estética y posicionamiento de los objetos, se trabajaron con componentes externos haciendo que sea más fácil de ver para el usuario.
- Para cumplir el objetivo del maquetado final, el desarrollo de componentes reutilizables mejoró la modularidad del sistema, permitiendo construir objetos más complejos con menos código y mayor coherencia visual.
- También se logró una integración efectiva de entrada por voz, adaptada según la demo, lo que permitió mantener una experiencia coherente en ambos entornos; esto sirvió tanto para el maquetado final en escritorio como en gafas tipo Quest 3, cumpliendo así esos objetivos.

3.7.4. Lecciones aprendidas

- Se valoró el uso de herramientas externas y componentes, que ampliaron las posibilidades de diseño y funcionalidad sin comprometer el rendimiento.
- El componente `look-at` resolvió perfectamente el problema de la ubicación de objetos de información.
- El componente `htmlembed` permitió añadir información más variada a la escena, pero no se logró conseguir el dinamismo esperado durante el uso de la aplicación.
- Adaptar interfaces y métodos de interacción según el dispositivo resultó ser lo más adecuado para tener dos demos diferenciadas.
- Finalmente, se reafirmó la importancia de mantener una arquitectura limpia y desacoplada, de esta manera, el uso del front-end y el back-end solo requirió de mínimos cambios en la demo de escritorio.

Capítulo 4

Sistema resultante

El resultado de este proyecto fue la creación de un conjunto de componentes modulares que sirven como base para el desarrollo de diversas aplicaciones interactivas. Para demostrar la funcionalidad y versatilidad de estos componentes, se ha creado una aplicación que tiene dos modos de empleo, una para navegador en escritorio y otra para navegador de gafas tipo Quest 3 descrita a continuación.

4.1. Aplicación desarrollada

Como resultado principal de este trabajo de fin de grado, se ha construido una aplicación que ha sido diseñada para facilitar la creación y manipulación de objetos geométricos primarios mediante el uso de comandos de voz. Incluye funciones como la generación de figuras primitivas geométricas, modificación paramétrica y gestión de objetos ya creados, y la posibilidad de guardar y cargar una escena reciente, todo ello usando comandos de voz.

La aplicación cuenta con dos versiones: una para escritorio basada en el reconocimiento de voz que ofrece Web Speech API, y otra versión para navegadores de gafas tipo Quest 3 que no aceptan la API de reconocimiento de voz mencionada, sino que se ha creado un servidor privado que hace el mismo procedimiento pero con otra API de reconocimiento de voz llamada AssemblyAI. La diferencia fundamental reside en la API de reconocimiento de voz empleada en cada implementación; las demás funcionalidades y componentes permanecen idénticos.

Descripción de la aplicación:

1. Se explicará el entorno de ejecución de la aplicación.

2. Posteriormente, se describirá la escena 3D que se presenta al usuario al ejecutarla.
3. Luego, se detallarán los comandos disponibles y su modo de utilización.
4. Además, se explicará el flujo de trabajo de la aplicación, una vez ya entendido el uso de cada componente.
5. Finalmente, se explicará cómo este sistema de componentes es capaz de crear otras aplicaciones combinando de diferente manera los componentes.

4.1.1. Entorno de ejecución

La aplicación funciona en navegadores; estos deberán soportar las APIs de reconocimiento de voz (Web Speech API o AssemblyAI) implementadas en este sistema de componentes. Se necesita además que pueda manejar A-Frame y las tecnologías sobre las que se construye.

4.1.2. Objetos en escena

A continuación se describirán los objetos que el usuario puede ver en la escena y el porqué de su implementación.

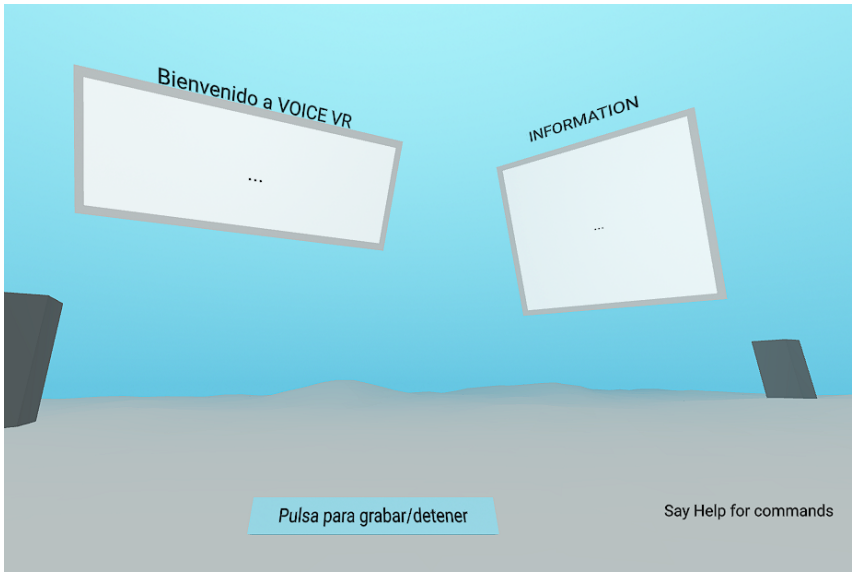


Figura 4.1: Objetos en escena que puede ver el usuario.

- Se ha creado un recuadro 3D que permite iniciar o detener el reconocimiento de voz, para implementar la funcionalidad de interacción con voz. Dicho recuadro está situado en la

parte inferior de la cámara del usuario, como se ve en la Figura 4.1.

- Se ha creado un panel de transcripción situado a la izquierda en la Figura 4.1, ya que en algunos casos es necesario ver las transcripciones que se han realizado durante la conversación con la aplicación, cumpliendo así con la función de retroalimentación.
- Se creó el panel de información situado a la derecha en la Figura 4.1, para poder contextualizar la situación de la escena, donde su contenido se adapta al modo de interacción en el que se encuentre (estado según el comando seleccionado).
- Se creó un texto para los mensajes de usuario situado en la parte superior del panel de transcripción, para mostrar notificaciones críticas del sistema y confirmaciones de acciones, asegurándose de que el usuario siempre esté informado y siga los pasos necesarios para el avance de la aplicación.

4.1.3. Comandos de Voz

La escena permite interactuar usando los siguientes comandos de voz. Una vez que haya iniciado la grabación de voz, dicho inicio se realiza haciendo clic en la caja de entrada de voz antes de decir un comando. Se tiene que decir exactamente las palabras que se mencionan a continuación:

- **CREAR:** Activa el modo de creación. El panel de información mostrará los tipos de objetos que puedes crear (cubo, esfera, plano, cilindro, luz, assets).
- **EDITAR:** Activa el modo de edición. El panel de información te pedirá el ID del objeto que deseas modificar.
- **ELIMINAR:** Activa el modo de eliminación. El panel de información te pedirá el ID del objeto a eliminar y luego te pedirá confirmación (Sí/No).
- **GUARDAR:** Guarda el estado actual de la escena en la memoria de tu navegador.
- **CARGAR:** Carga la última escena que fue guardada en la memoria de tu navegador.
- **AYUDA / HELP / COMANDOS:** Muestra la lista de comandos disponibles en el panel de información.
- **SALIR / ATRÁS / CAMBIO:** Utiliza cualquiera de estas palabras para salir del modo actual (creación, edición o eliminación) y volver al estado de espera de un nuevo comando

principal.

4.1.4. Uso de los Comandos

- Para la creación de objetos: El proceso se inicia con un clic en la caja de entrada de voz si no está ya activa. A continuación, el usuario debe pronunciar el comando '**CREAR**'. El sistema responderá mostrando en el panel de información una lista de los objetos disponibles. El usuario selecciona el objeto deseado vocalizando su nombre (por ejemplo, 'cubo', 'esfera' o 'luz'). Una vez creado el objeto, la aplicación entra automáticamente en modo de modificación para dicho elemento, como se muestra en la Figura 4.2. En este punto, el panel de información guiará al usuario, indicando los comandos disponibles para ajustar la posición, rotación, tamaño, color o ID del objeto. Para finalizar las modificaciones y salir de este modo, el usuario debe decir 'cambio'.

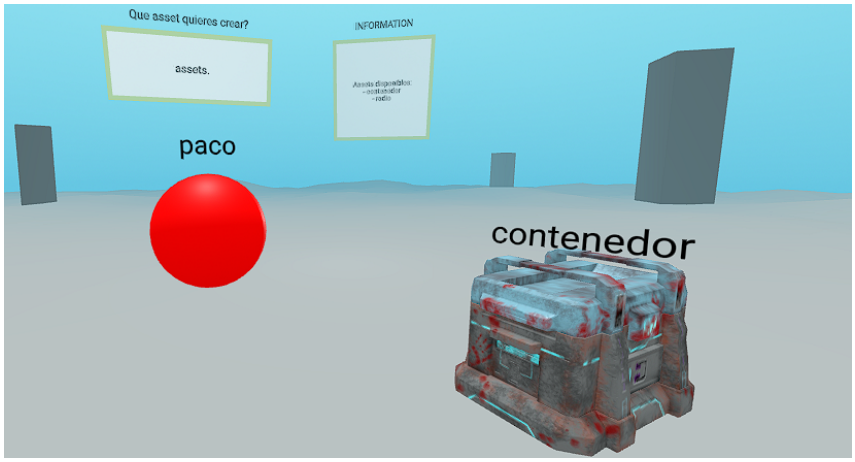


Figura 4.2: En la imagen se puede ver la creación de un asset llamado 'contenedor' junto a dos objetos previamente creados.

- Para editar un objeto: El proceso es similar al de creación. Primero, asegúrate de que la caja de entrada de voz esté activa; si no lo está, haz clic en ella. Luego, pronuncia el comando '**EDITAR**'. El panel de información te solicitará el identificador único (ID) del objeto que deseas modificar. Una vez que digas el ID, el sistema señalizará visualmente el objeto seleccionado en la escena como se ve en la Figura 4.3. A partir de ese momento, el panel de información te guiará con las instrucciones para ajustar la posición, rotación, tamaño, color o ID del objeto. Cuando hayas finalizado las modificaciones deseadas, se

deberá decir ’cambio’ para abandonar el modo de edición.

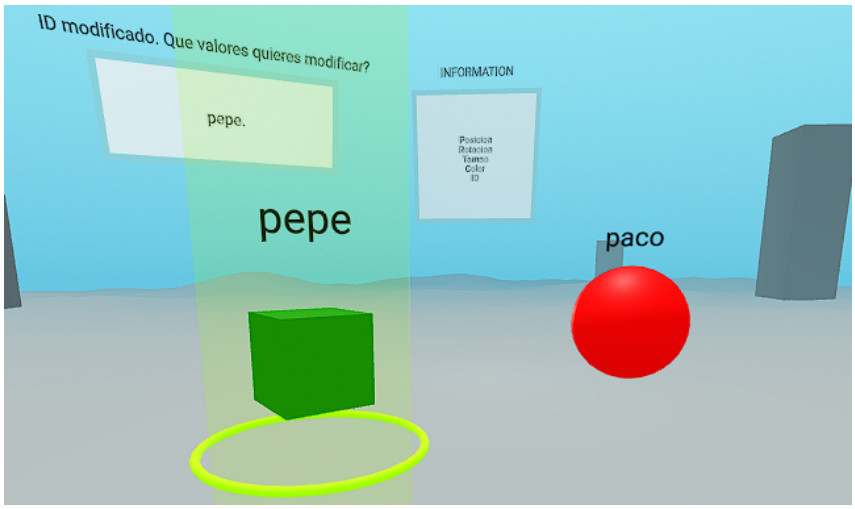


Figura 4.3: Funcionamiento al editar objetos con ID, donde el objeto a editar está marcado y la interfaz está esperando el siguiente paso para modificar algún parámetro del objeto, se puede observar como el marco de los paneles cambio de color.

- Para eliminar un objeto: El procedimiento comienza asegurándose de que la caja de entrada de voz esté activa; si no es así, actívala con un clic. Luego, vocaliza el comando ’**ELIMINAR**’. El panel de información te solicitará el ID del objeto que deseas suprimir. Una vez que pronuncies el ID, el sistema señalizará visualmente el objeto en la escena para confirmación. Acto seguido, el panel de información te pedirá que confirmes la acción como se ve en la Figura 4.4. Para proceder con la eliminación, diga ’sí’; para cancelar, diga ’no’. Finalmente, tras confirmar o cancelar la operación, diga ’salir’ o ’atrás’ para regresar al estado de funcionamiento normal de la aplicación.

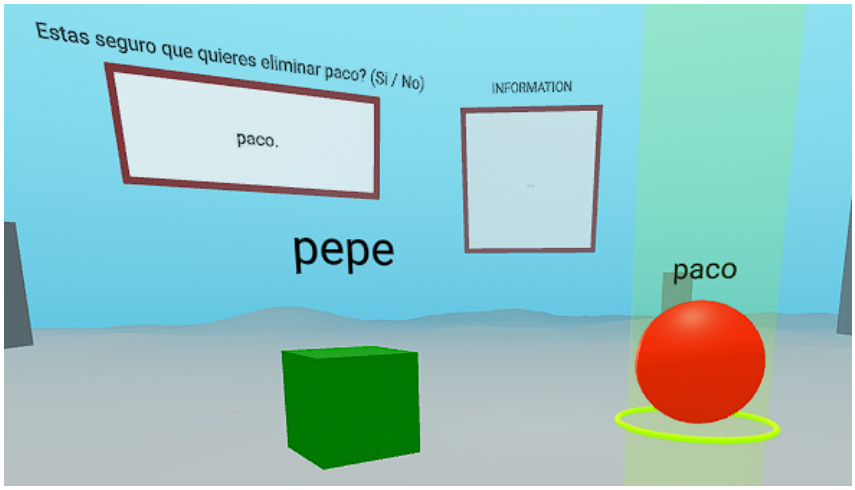


Figura 4.4: Se muestra en la escena el objeto a eliminar, donde la interfaz está esperando la confirmación de la eliminación del objeto 'paco', al lado del objeto 'pepe', se puede observar como el marco de los paneles cambio de color.

- Para guardar la escena actual, se deberá activar el reconocimiento de voz, si no lo está. Posteriormente, pronuncia el comando '**GUARDAR**'. Una vez ejecutado, el sistema proporcionará una confirmación visual indicando que la escena ha sido guardada satisfactoriamente.
- Para cargar la escena actual, se deberá activar el reconocimiento de voz, si no lo está. Posteriormente, pronuncia el comando '**CARGAR**'. Una vez ejecutado, el sistema proporcionará una confirmación visual indicando que la escena ha sido cargada satisfactoriamente.
- Para solicitar ayuda o ver los comandos disponibles, se deberá activar el reconocimiento de voz, si no lo está. Luego, pronuncia uno de los siguientes comandos: '**AYUDA**', '**HELP**' o '**COMANDOS**'. El panel de información mostrará entonces la lista de comandos principales que puedes utilizar, como se ve en la Figura 4.5. Para salir de un modo específico (como '**CREAR**', '**EDITAR**' o '**ELIMINAR**') en cualquier momento, se deberá hacer clic en la caja de entrada de voz y decir '**SALIR**', '**ATRÁS**' o '**CAMBIO**'.

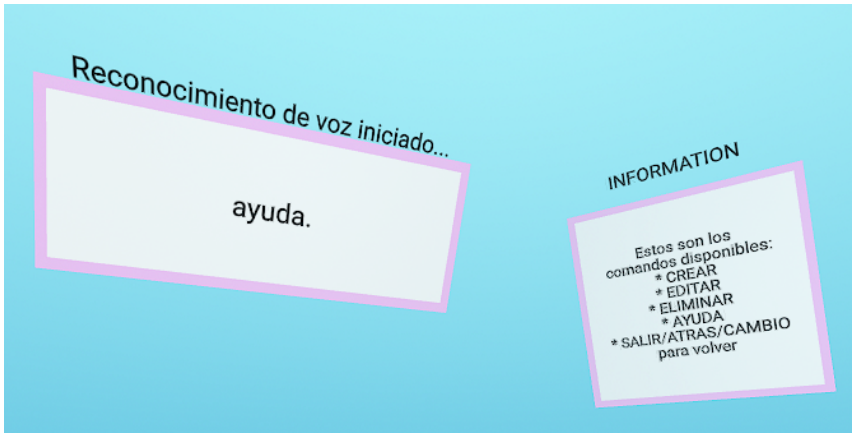


Figura 4.5: Se muestra la escena donde el panel de información contiene los diferentes comandos que se pueden usar, además de distinguirse el estado de la escena con los bordes de color rosa.

4.1.5. Notas Adicionales

La funcionalidad de guardar y cargar escenas utiliza la memoria local del navegador, por lo que las escenas no se conservarán si se borra la caché o se utiliza otro navegador/dispositivo. La precisión del reconocimiento de voz puede variar dependiendo del micrófono, el acento y el ruido ambiental. El panel interactivo inicial (`htmlembed`) proporciona una guía visual adicional dentro de la escena. Se puede cerrarlo haciendo clic en el botón 'Close' al final de las diapositivas, como se ve en la Figura4.6.

Figura 4.6: Se visualiza el panel creado con htmlembed donde se muestra cómo cerrarlo al final de las diapositivas de ayuda al usuario.

4.2. Descripción de la implementación

A continuación se describe la arquitectura y el funcionamiento interno del conjunto de componentes, diseñado para extender A-Frame y permitir la interacción por voz en un entorno VR. Dichos componentes están implementados en el fichero componente.js[1]

Esta sección detalla los aspectos técnicos clave del código para facilitar su comprensión y mantenimiento.

4.2.1. Arquitectura General

La aplicación sigue un patrón basado en componentes de A-Frame. Cada componente encapsula una funcionalidad específica (manejo de entrada, creación de UI, gestión de comandos, modificación de objetos, etc.). La comunicación entre componentes se realiza principalmente a través de eventos de A-Frame emitidos en la escena (`a-scene`) o en elementos específicos (`a-entity`). Dado que los componentes creados son componentes de A-Frame normales, interaccionan perfectamente con otros componentes de A-Frame en una escena, lo que facilita la creación de entornos más completos y complejos.

4.2.2. Componentes Principales

- **input-text**: Este componente utiliza **Web Speech API** como dependencia principal. Su funcionalidad consiste en capturar audio del micrófono, realizar transcripción en tiempo real, aplicar normalizaciones (quitar acentos, convertir números de texto a dígitos) y emitir un evento **`transcription`** con el texto limpio. El componente responde a un evento clic en el propio elemento para iniciar/detener la grabación.

  Existe una versión para gafas tipo Quest 3 que usa una conexión a un servidor privado, en el que se realiza el mismo procedimiento que Web Speech API pero con otra aplicación de transcripción de voz llamada AssemblyAI.
- **transcription-display**: Este componente depende de un elemento de entrada especificado por el selector `input`. Su función principal es mostrar las transcripciones de voz recibidas de este elemento de entrada dentro de la escena de A-Frame.

[1] fichero alojado en GitHub:`https://github.com/R4CC00N/SpeechRecognition_in_Browser/blob/main/DemoEscritorio/componente.js`

Escucha los eventos **`transcription`** emitidos por el elemento de entrada y toma el texto de la transcripción (`event.detail.transcription`). Luego, busca un elemento **`<a-text>`** con el ID **'transcriptionText'** dentro de sí mismo y actualiza su atributo `value` con la transcripción recibida.

- **command-handler:** Este componente depende del componente **`input-text`** especificado por el selector `input`, este componente necesita un ID (create) que será usado de `input` en otro componente (`object-creator`). Su función es actuar como controlador de eventos para componentes como el `object-creator` y el `sky-manager`.
  Escucha los eventos **`transcription`** y, basándose en palabras clave, determina el modo de operación y emite eventos de A-Frame correspondientes en el elemento **`<a-scene>`** como los mostrados en el Cuadro4.1 a continuación:

Acción detectada	Comando Emitido
Crear	enter-create-mode, start-object-creation
Editar	edit-mode
Eliminar	delete-mode
Ayuda	help-mode
Guardar	No emite
Cargar	No emite
Salir	exit-create-mode

Cuadro 4.1: Relación entre acciones y comandos emitidos

  En resumen, este componente interpreta los comandos hablados y los traduce en acciones concretas dentro de la escena de A-Frame.

- **object-creator:** Este componente escucha el evento **start-object-creation** emitido por el **command-handler** cuya ID es un parámetro de entrada para este componente ('create'). Al recibir el evento, se crean dinámicamente entidades primitivas o modelos GLTF en A-Frame basadas en el `type` especificado en el detalle del evento. Asigna un ID predeterminado al objeto recién creado, el cual aparecerá en la parte superior y se podrá ser modificado. Este ID será importante, ya que otros componentes lo usarán para interactuar con el objeto. Adicionalmente, durante la creación se establecen atributos iniciales y se añade la nueva entidad al contenedor **`'objsCreated'`**. También asocia una variable de estado global (`modifyingBox, modifyingSphere`, etc.) a `true` para indicar que

se ha iniciado la modificación.

- **dynamic-modifier:** Este componente depende del componente **`input-text`** especificado por el selector `input`. Su función se basa en manejar la modificación de atributos (posición, rotación, tamaño, color, ID) del último objeto creado mientras las variables de estado 'modifying' correspondientes sean **true**. Para la gestión del flujo de datos de entrada, utiliza la variable global '`step`' para seguir el flujo de entrada de valores (ej. pedir X, luego Y, luego Z para la posición). También escucha comandos de modificación o el comando 'cambio' para finalizar el modo de modificación. Por último, emite eventos en el elemento **`<a-scene>`** tales como `exit-create-mode`, `start-create-mode`.
- **edit-mode-handler:** Este componente depende del componente **`input-text`** especificado por el selector `input`. Tiene una función similar a `dynamic-modifier`, pero para editar un objeto existente. Se activa con el comando '**editar**' a través del evento **`transcription`**. Una vez activado, se pide el ID (el nombre que aparece sobre objeto) del objeto a editar, lo busca en el DOM, le añade temporalmente un 'torus' de señalización y permite modificar sus atributos utilizando la variable `step` y las funciones auxiliares. Para salir del modo edición se deberá usar el comando 'cambio'. Durante el proceso, el componente puede emitir eventos en el elemento **`<a-scene>`**, como `exit-create-mode`, cuando se abandona el modo de edición.
- **delete-object:** Este componente depende del componente **`input-text`** especificado por el selector `input`. La función principal es la eliminación de objetos. Se activa con el comando '**eliminar**' a través del evento **`transcription`**. Una vez iniciado, se pide el ID (el nombre que aparece sobre objeto) del objeto que se quiere eliminar, lo busca en el DOM y lo señaliza con un torus y espera una confirmación (Sí/No) antes de eliminar la entidad del DOM. Para salir del modo de eliminación se utilizan los comandos de salida o cambio. El componente finaliza su funcionamiento al recibir comandos de salida y, al hacerlo, emite el evento `exit-create-mode` en el elemento **`<a-scene>`**.

- **scene-saver:** Este componente se activa cuando el `command-handler` detecta el comando 'guardar'. Ejecutando la función `saveScene` que exporta el estado de las entidades que se encuentran dentro de la entidad cuyo ID es **`'objsCreated'`** a un objeto

JSON. Guarda atributos clave como `id`, `class`, y los valores de componentes como `position`, `rotation`, `scale`, `geometry`, `material`, `light` y `gltf-model`. También guarda los hijos de cada entidad (incluyendo los textos de los IDs). El JSON resultante se almacena en `localStorage` bajo la clave '**savedScene**'. Para facilitar su uso desde otros componentes o scripts, la función `saveScene` se expone globalmente a través del objeto `window`.

- **scene-loader:** Este componente se activa cuando el `command-handler` detecta el comando 'cargar'. Dicho comando activa la función `loadScene` que toma un objeto JSON (compatible con el formato de `scene-saver`). Limpia el contenido actual de **`'objsCreated'`** y recrea las entidades y sus hijos a partir de los datos en el JSON. La función `loadSceneFromStorage` recupera el JSON de `localStorage` y llama a `loadScene`. Para permitir su uso desde cualquier parte de la aplicación, `loadSceneFromStorage` se expone globalmente a través del objeto `window`.
- **sky-manager:** Este componente escucha eventos en **`<a-scene>`** emitidos por el componente `command-handler` (enter-create-mode, edit-mode, help-mode, delete-mode, exit-create-mode). Cuando se recibe uno de estos eventos, el componente cambia dinámicamente el color del material del propio elemento al que está asociado (con un 'setAttribute'); en el caso del Listado 4.1 el objeto que cambiará de color será el panel.

```
<a-plane sky-manager></a-plane>
```

Listado 4.1: Escena de uso para cambio de color en el componente sky-managar

De este modo, cada modo activo (crear, editar, ayuda, eliminar, etc.) puede reflejarse visualmente mediante un color distinto, facilitando la comprensión del estado actual de la interfaz.

- **scene-creator:** Este componente es uno de inicialización. Asegura la existencia de **`<a-assets>`** y añade los assets GLTF necesarios. Crea y adjunta a la entidad que lleva este componente (presumiblemente la escena principal o un contenedor) instancias de los otros componentes personalizados (info-panel, text-message, player-move, command-handler, object-creator, dynamic-modifier, edit-mode-handler, delete-object, scene-saver, scene-loader, transcript-panel).

4.2.3. Componentes Auxiliares

- **info-panel:** Gestiona y muestra información al usuario. Es un panel de ayuda, mensajes de estado y detalles sobre la escena. Esta información se escribe en el elemento `<a-text>` cuyo ID es 'typeMessage' en donde el parámetro modificable es **value**.
- **text-message:** Dedicado a la visualización de mensajes de texto en la escena. Esto se usa para notificaciones, diálogos o cualquier comunicación textual con el usuario. Esta información se escribe en el elemento `<a-text>` cuyo ID es 'userMessage', en donde el parámetro modificable es **value**.
- **player-move:** Este es un componente crucial que se encarga del movimiento del jugador (la cámara y los controles de VR). Dentro de su `init function`, se encarga de:
  - Crear un `voiceInputBox` (una caja interactiva para entrada de voz) con atributos para posición, rotación, tamaño, color, animación, y un componente **`input-text`**.
  - Crear un texto dentro de `voiceInputBox` que indica 'Pulsa para grabar/detener'.
  - Crear la entidad del jugador con controles de movimiento.
- **transcript-panel:** Muestra una transcripción recibida por parte del evento `transcript` en las interacciones de voz o comandos, lo que podría ser útil para depuración o para que el usuario revise lo que se ha dicho. Esta información se escribe en el elemento `<a-text>` que aloja un componente llamado **`transcription-display`** en donde el parámetro modificable es el **value**.

4.2.4. Funciones de utilidad

Hay varias funciones que son usadas en los componentes que son capaces de cambiar las propiedades de los objetos, por ello se han escrito una sola vez fuera de los componentes para que puedan ser reutilizadas en ellos. Son las siguientes:

- **stepsPOS(posKey, entity, transcript):** Permite modificar las coordenadas X, Y, Z de la posición en pasos. Al decir 'posición', se pregunta por la 'x', luego 'y', luego 'z'.
- **stepsROT(posKey, entity, transcript):** Permite modificar las coordenadas X, Y, Z de la rotación en pasos. Al decir 'rotación', se pregunta por la 'x', luego 'y', luego 'z'.

- **stepCOLOR(entity, transcript):** Intenta encontrar un color válido en el `transcript` utilizando un mapa (`colorsMap`) y actualiza el color del material del objeto.
- **stepID(entity, transcript):** Permite cambiar el `id` de la entidad y también actualiza el texto que se muestra sobre el objeto.
- **stepSIZE(entity, transcript):** Intenta extraer un número del `transcript` y lo aplica como un nuevo tamaño a todos los parámetros de la geometría del objeto (excepto `primitive` y otros parámetros específicos).

4.3. Arquitectura de funcionamiento de la aplicación

El diagrama de la Figura 4.7 muestra dos arquitecturas distintas para transcribir voz en la aplicación web VR/AR construida con A-Frame, también muestra la estructura de componentes que comparten ambas arquitecturas a execpción del componente **`input-text`**. Existe un flujo que usará **Web Speech API** para la versión de escritorio y otro flujo que usará una reinterpretación del proceso de transcripción que usa Web Speech API pero con la API de transcripción de **AssemblyAI** para su uso en gafas tipo Quest 3.

Para cada acción o uso de un comando se seguirá el flujo presentado a continuación que se repetirá cada vez que se acceda de nuevo:

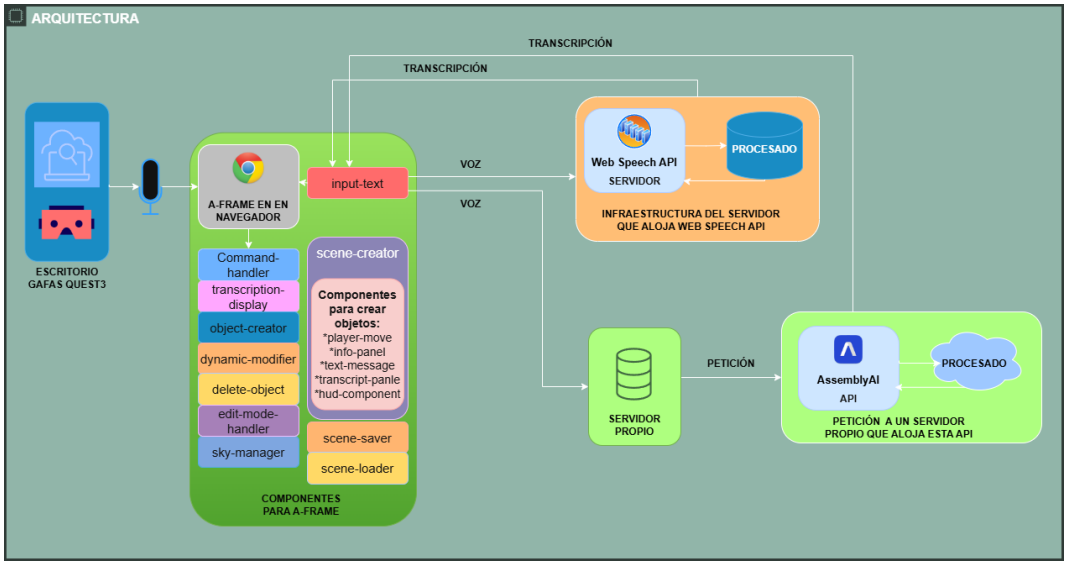


Figura 4.7: Arquitectura del funcionamiento de la aplicación que use los componentes desarrollados.

1. El usuario desde la aplicación de escritorio o VR/AR dará inicio a la grabación de la voz.

2. La señal de audio capturada por el componente **`input-text`** se envía por el flujo 'VOZ'. Dependiendo de si el usuario está en escritorio o gafas Quest 3 se enviará a **Web Speech API** del navegador o a la API del servidor basada en **AssemblyAI** respectivamente.

3. El navegador o servidor, utilizando estas APIs, son los encargados de todo el PROCESAMIENTO del audio. El componente **`input-text`** interactúa con estas APIs para iniciar/detener la grabación y recibir los resultados.

4. Una vez que se ha terminado de procesar una frase o ha detectado una pausa, devuelve la transcripción al componente **`input-text`** por medio del flujo 'TRANSCRIPCIÓN' y este emitirá el evento **`transcription`** a la escena.

5. Luego, la transcripción es manejada por otros componentes de A-Frame como **command-handler** que se encargará de ejecutar los otros escenarios posibles dependiendo del comando usado.

4.4. Reutilización y Modularidad

La naturaleza modular de estos componentes los hace reutilizables en diferentes aplicaciones. Al estar diseñados para encapsular funcionalidades específicas y comunicarse a través de eventos de A-Frame, pueden ser 'intercambiados' o 'combinados' para construir diferentes aplicaciones sin necesidad de reescribir grandes porciones de código.

Al ser componentes por separado, se puede usar una mezcla de estos componentes para gestionar otras aplicaciones.

El centro de todo esto es el evento **transcript** que es emitido a la escena por el componente **`input-text`**. La gran mayoría de componentes creados en este prototipo utiliza este evento para la gestión interna de funciones.

4.4.1. Ejemplo de la aplicación completa del proyecto

A continuación se explicará la creación de una escena para la aplicación final simple desde el HTML. El Listado 4.2 es la base fundamental de la creación de la escena con sus respectivos componentes. En este contexto, se usó un componente principal:

- **scene-creator**: Se encargará de crear en escena los demás componentes cuando se inicie la aplicación en el navegador.

Por otro lado, en la entidad con ID `'objsCreated'` se almacenarán como hijos todos los objetos creados durante el uso de la aplicación.

También podemos observar los diferentes scripts que ejecutan el HTML para su funcionamiento, en donde encontramos desde la librería de A-Frame utilizada hasta el código JavaScript del proyecto.

```
<head>
  <script src="https://aframe.io/releases/1.6.0/aframe.min.js"></script>
  <script src="https://unpkg.com/aframe-look-at-component@0.8.0/dist/aframe
     -look-at-component.min.js"></script>
  <script src="componente.js"></script>
</head>
<body>
  <a-scene>
      <a-entity scene-creator></a-entity>
      <a-entity id="objsCreated">
      </a-entity>
  </a-scene>
</body>
```

Listado 4.2: Escena de ejemplo para ilustrar los dos componentes principales para la creación de la aplicación final.

4.4.2. Ejemplo de creación de aplicación de transcripción de voz en pantalla con A-Frame

A continuación, se ilustrará el uso de la caja de herramientas mediante un ejemplo práctico:

Se quiere una aplicación que transcriba la voz y la muestre en pantalla. Se usará una mezcla de componentes auxiliares y componentes principales. Como componentes principales se usarán:

- **player-move:** Este componente está compuesto por entidades, dentro de las cuales se encuentran todos los elementos visuales que verá el usuario en la cámara en todo momento. Cuenta con el botón del componente **`input-text`** y textos auxiliares.

- **transcript-panel:** Este componente está compuesto de entidades de las cuales una está en el componente **`transcription-display`**. Este se encargará de crear y actualizar un elemento de texto 3D para mostrar las transcripciones recibidas.

Como componentes auxiliares se usarán:

- **input-text:** Se adjuntará dentro de algún elemento interactuable para la grabación de voz. Este componente procesará la voz y emitirá la transcripción (evento **transcript**).

- **transcription-display:** Su función principal es mostrar las transcripciones de voz recibidas de este elemento de entrada dentro de la escena de A-Frame. Buscando un elemento **`<a-text>`** con un ID específico llamado **'transcriptionText'**.

```
 <head>
  <script src="https://aframe.io/releases/1.6.0/aframe.min.js"></script>
  <script src="https://unpkg.com/aframe-look-at-component@0.8.0/dist/aframe
     -look-at-component.min.js"></script>
  <script src="componente.js"></script>
</head>
<body>
  <a-scene>
    <a-entity player-move></a-entity>
    <a-entity transcript-panel></a-entity>
  </a-scene>
</body>
```

Listado 4.3: Escena de ejemplo para ilustrar la modularidad donde se ven los dos componentes principales.

Composición de componentes en la escena

Otros componentes, por ejemplo, no usan ID específicos. El caso es, por ejemplo, el **command-handler** que directamente usa una nomenclatura en HTML en donde tiene como parámetro de entrada la entidad que contiene el componente **`input-text`**. De esta forma, gran cantidad de los componentes aquí creados pueden ser llamados a escena por separado.

```
<a-entity id="create" command-handler="input:#voiceInputBox"></a-entity>
<a-entity id="object" object-creator="input:#create"></a-entity>
<a-entity id="dynamic-modifier" dynamic-modifier="input:#voiceInputBox"></a
   -entity>
<a-entity id="edit" edit-mode-handler="input:#voiceInputBox"></a-entity>
<a-entity id="delete" delete-object="input:#voiceInputBox"></a-entity>
<a-entity id="saver" scene-saver="input:#voiceInputBox"></a-entity>
<a-entity id="loader" scene-loader="input:#voiceInputBox"></a-entity>
<a-entity id="transcriptPanel" transcript-panel="input:#voiceInputBox"></a-
   entity>
```

Listado 4.4: Crear Componentes en la escena de HTML por separado.

Entonces, si se quiere usar esta aplicación sin el **`input-text`** o usarla con una entrada de texto diferente, entonces a los componentes se les deberá cambiar el ID de entrada.

Capítulo 5

Experimentos y validación

Con el fin de validar la funcionalidad del sistema desarrollado, se realizó una serie de experimentos centrados en evaluar tanto el correcto reconocimiento de comandos como la experiencia de usuario.

Las pruebas se realizaron en un entorno controlado dentro de una habitación que estaba equipada con una mesa de trabajo despejada, una silla ergonómica y la infraestructura tecnológica necesaria para ejecutar la aplicación.

5.1. Objetivos

El experimento se enfocó en medir tres aspectos principales del sistema:

- **Precisión del reconocimiento de voz.**
- **Facilidad de uso e intuición de los comandos.**
- **Tiempo medio desde la grabación de voz hasta la finalización de la tarea.**
- **Conseguir realimentación de los usuarios sobre la experiencia.**

5.2. Entorno de pruebas

Se utilizaron dos versiones del sistema para comparar el rendimiento entre plataformas, lo que resultó en tres posibilidades de uso distintas.

La primera versión cuenta con un reconocimiento de voz en el navegador proporcionado por la API de Web Speech API, la cual cuenta con dos posibilidades de uso:

Uso en escritorio:[1] Con un monitor de 24 pulgadas, utilizando el navegador Google Chrome

Uso en móvil:[2] El móvil será de gama media (Redmi Note 11 Pro), usando el navegador de Edge.

La segunda versión de la aplicación cuenta con un reconocimiento de voz que se conecta a un servidor privado. Este servidor realiza la misma funcionalidad que Web Speech API, pero empleando una API diferente llamada AssemblyAI. Cuya posibilidad de uso es:

Uso en gafas VR/AR:[3] Se usaron las gafas de meta Quest 3 para trabajar en un entorno virtual.

Descripción de los usuarios y las pruebas.

Participaron cinco individuos con perfiles técnicos diversos, que no habían probado la aplicación con anterioridad. Además, usuarios como el 1 y el 5 no habían experimentado el uso de gafas tipo Quest 3. Esta variedad de perfiles se buscó intencionadamente para obtener una retroalimentación representativa de diferentes niveles de familiaridad con la tecnología.

Las pruebas se llevaron a cabo utilizando la versión final de la demostración de la aplicación.

Para cada una de las pruebas, se realizaron 20 comandos de voz en donde un único experimentador guio a cada participante a través de la sesión, asegurándose de que las instrucciones se proporcionaran de manera estandarizada y consistente. Antes de comenzar cada tarea, el experimentador se dirigía al participante utilizando las siguientes palabras textuales:

"A continuación, te guiaré a través de una serie de tareas sencillas para evaluar cómo interactúas con el sistema. Por favor, sigue mis indicaciones y los mensajes informativos que aparecerán en la pantalla o dentro del entorno virtual."

[1] **Uso en escritorio:** `https://r4cc00n.github.io/SpeechRecognition_in_Browser/DemoEscritorio/`

[2] **Uso en móvil:** `https://r4cc00n.github.io/SpeechRecognition_in_Browser/DemoEscritorio/`

[3] **Uso en gafas Quest 3:** `https://github.com/R4CC00N/SpeechRecognition_in_Browser/tree/main/DemoVR`

Secuencia de tareas.

Una vez que el participante estaba preparado, el experimentador indicaba cada tarea de forma secuencial, apoyándose también en los carteles de información integrados en la aplicación que proporcionaban recordatorios visuales de los comandos esperados. La secuencia de tareas que cada participante debía completar fue la siguiente:

1.- Creación de un cubo: El experimentador indicaba verbalmente: "Por favor, intenta crear un cubo utilizando el comando de voz **'crear'** y luego **'cubo'**." El participante debía accionar el sistema de reconocimiento de voz (pulsando el botón) y pronunciar el comando. Después de la acción del usuario, se observaba si el cubo se creaba correctamente en la escena y se avisaba al experimentador.

2.- Cambio de color del objeto: Una vez creado el cubo, el experimentador dirá: "Ahora, por favor, intenta cambiar el color del cubo a rojo utilizando el comando **'color'** y acto seguido **'rojo'**." El participante debía activar el reconocimiento de voz si se desactivó previamente y pronunciar el comando. En el momento en el que el color del cubo cambiaba a rojo según lo esperado, se avisaba al experimentador.

3.- Posicionamiento del objeto: Tras el cambio de color, el experimentador instruía: "Ahora, intenta mover el cubo a otra ubicación dentro de la escena. Puedes usar el comando **'posición'** y seguir las instrucciones en la información de la aplicación para especificar coordenadas." Se observaba si el participante lograba desplazar el objeto utilizando los comandos de movimiento disponibles y, una vez conseguido, se avisaba al experimentador.

4.- Eliminación del objeto: Finalmente, el experimentador pedía: "Por favor, elimina el cubo de la escena utilizando el comando **'eliminar'**." Se verificaba si el objeto desaparecía de la escena tras la ejecución del comando por parte del usuario.

5.- Manejo libre de la aplicación: Después de completar las tareas guiadas, el experimentador indicaba: "Ahora tienes unos minutos para explorar libremente la aplicación. Intenta crear otros objetos, cambiar sus propiedades o realizar cualquier acción que te parezca interesante utilizando los comandos de voz que recuerdes o que veas en la información." Durante este tiempo, se observaba cómo el participante interactuaba con el sistema sin indicaciones directas, los comandos que intentaba utilizar y cualquier dificultad que pudiera encontrar.

5.3. Resultados

De 20 comandos de reconocimiento de voz se han obtenido los siguientes resultados en cada versión, añadiendo las observaciones más destacadas de cada una en el Cuadro 5.1.

5.3.1. Precisión del Reconocimiento de Voz

Usuario	Plataforma	Comandos exitosos	Observaciones
Usuario 1	Quest 3	19/20	Reconocimiento lento, modalidad de grabación de audio molesta.
Usuario 2	Escritorio	18/20	Reconocimiento rápido, errores con las transcripciones en algunos casos.
Usuario 3	Móvil	17/20	Problemas con la interfaz para grabar audio; una vez obtenido el manejo, bastante fluido gracias a los mensajes de información.
Usuario 4	Escritorio	17/20	Sin demoras en respuesta, no tiene comandos encadenados.
Usuario 5	Quest 3	18/20	Alta precisión en acento marcado, demoras en respuesta.

Cuadro 5.1: Número de comandos exitosos por usuario y plataforma

5.3.2. Tiempo por Tarea

Usuario	Crear objeto	Cambiar color	Mover objeto	Eliminar objeto
Usuario 1	7 segundos	9 segundos	13 segundos	7 segundos
Usuario 2	4 segundos	4 segundos	8 segundos	4 segundos
Usuario 3	6 segundos	7 segundos	10 segundos	6 segundos
Usuario 4	3 segundos	5 segundos	9 segundos	5 segundos
Usuario 5	7 segundos	8 segundos	15 segundos	8 segundos

Cuadro 5.2: Tiempo de realización de tareas por usuario

Se observó que los tiempos en la versión Quest 3 fueron ligeramente más altos debido al retardo en la conexión al servidor externo, como se ve en el Cuadro 5.2 y en la Figura 5.1 más gráficamente.

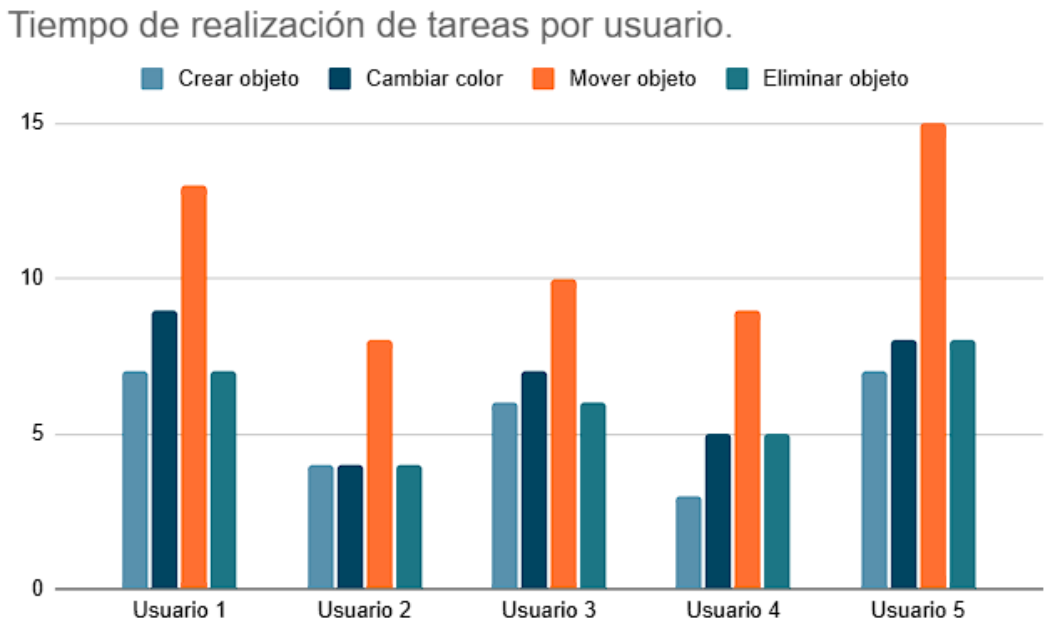


Figura 5.1: Tabla del tiempo que a cada usuario le tomo realizar una tarea.

5.3.3. Opiniones de los Usuarios

Las opiniones recogidas al final de las pruebas fueron mayoritariamente positivas. A continuación se resumen algunos comentarios:

- **Usuario 1:** "La experiencia en realidad virtual es muy inmersiva. Me gustaría poder usar frases más naturales y mejorar el tiempo de espera. Además, el manejo de grabar y parar de grabar por cada comando me pareció molesto y esperar por respuesta me parece lento."
- **Usuario 2:** "Sorprendido por la precisión del reconocimiento. Muy útil para crear escenas rápidamente y además muy vistosas."
- **Usuario 3:** "La aplicación me pareció interesante, es un poco molesto tener que darle al botón cada vez que quiero decir un comando; por lo general, bastante rápido y entretenido."

- **Usuario 4:** “Me sorprendió la velocidad de crear los objetos; los comandos, como están, me parece que hacen más lenta la interacción, pero la aplicación es bastante fácil de manejar.“
- **Usuario 5:** “Solo bastan unos minutos para entender cómo manejar la aplicación; la parte de ejecución libre fue la mejor, intentaría hacer que no se dependiera de darle al botón para grabar los comandos.“

Usuario	Tiempos de espera	Comandos	Botón Grabar voz	Uso fácil
Usuario 1	Alto	Modulares, hubiera sido mejor con detección de frases	Molesto, se debe interactuar cada vez que hablas.	Sí
Usuario 2	Bajo	Un poco lento el accionar en ocasiones	Sin comentarios	Sí
Usuario 3	Medio	Normales para edición	Tenía que activarlo de vez en cuando	Sí
Usuario 4	Medio	Fáciles de usar	Sin problemas	Sí
Usuario 5	Alto	Le gustan	Manejable	Sí

Cuadro 5.3: Datos obtenidos de los comentarios de los usuarios

Estos datos han sido obtenidos de la retroalimentación dada por el usuario y conversaciones extras, dotándonos de unas posibles mejoras a futuro.

5.4. Conclusión de Validación

Los resultados obtenidos muestran que el proyecto cumple con los objetivos planteados: permite la creación y manipulación de objetos 3D mediante comandos de voz de forma eficiente y accesible, en entornos de escritorio y VR. La precisión en el reconocimiento fue alta en ambos entornos, y los usuarios consideraron la experiencia intuitiva y útil.

Es cierto que aún existen limitaciones como el manejo de envío de audios para las transcripciones y una latencia alta en una versión del proyecto. También se destaca el funcionamiento como ’incómodo’ del repetido accionar el botón de grabar en una de las versiones; esto se debe al funcionamiento de la propia API de transcripción de voz. Sin embargo, nada de esto afecta al funcionamiento como tal, pero sí al rendimiento.

Capítulo 6

Conclusiones

6.1. Consecución de objetivos

En este Trabajo Fin de Grado se propuso como objetivo general explorar diversas técnicas para la construcción de un editor 3D dirigido por voz, culminando en el desarrollo de un prototipo funcional basado en un sistema de componentes que demostrara la viabilidad de la creación y edición de escenas 3D mediante comandos de voz en navegadores web compatibles.

Para alcanzar este **objetivo general**, se plantearon los siguientes **objetivos específicos**:

- Explorar qué opciones se tienen para 'Speech-to-Text' en un navegador y en dispositivos tipo Quest 3. Este objetivo se cumplió, para ello se realizó un análisis exhaustivo de las opciones de Speech-to-Text disponibles para ambos entornos, considerando factores como la precisión, la latencia, la compatibilidad y los costos asociados. Se investigaron y evaluaron diferentes tecnologías de reconocimiento de voz, incluyendo la Web Speech API para navegadores de escritorio y la API de AssemblyAI para las gafas Quest 3, identificando sus fortalezas y debilidades en cada plataforma.

- Conseguir transcripciones del usuario con la aplicación. Este objetivo se cumplió implementando las soluciones de Speech-to-Text seleccionadas en ambas versiones de la aplicación, logrando obtener transcripciones de los comandos de voz emitidos por los usuarios, siendo reflejados en una escena 3D con A-Frame.

- Reconocimiento de órdenes concretas para interactuar con el editor de escenas. Este objetivo se cumplió. Se desarrolló la lógica necesaria para identificar y procesar un conjunto

predefinido de comandos de voz para interactuar con los elementos de la escena 3D, se dividen en comandos de 'creación', 'edición', 'eliminación'.

- Explorar distintas soluciones para encontrar una que funcione bien en gafas tipo Quest 3 y otra en el navegador de escritorio. Este objetivo se cumplió satisfactoriamente, ya que se crearon dos prototipos diferentes para el funcionamiento, uno con Web Speech API para escritorio y AssemblyAI para gafas tipo Quest 3.

- Hacer que el prototipo soporte comandos como crear, editar, eliminar... Este objetivo se cumplió parcialmente. Se logró implementar el soporte para los comandos de 'crear', 'editar', 'guardar' y 'eliminar', demostrando la viabilidad de la edición básica por voz. Sin embargo, la gama completa de posibles comandos de edición no se implementó en la totalidad del prototipo debido a las limitaciones de tiempo.

- Realización de un prototipo modular. Este objetivo se cumplió parcialmente. Se adoptó una arquitectura basada en componentes de A-Frame, lo que permitió una cierta modularidad en la estructura del prototipo. Sin embargo, la modularidad podría haberse profundizado aún más para facilitar futuras extensiones y el mantenimiento del código.

- Construir un prototipo funcional en navegadores. Este objetivo se cumplió utilizando la Web Speech API y A-Frame. La funcionalidad es buena, pero el reconocimiento de voz no es perfecto, ya que depende mucho del entorno en el que se encuentre y el tipo de micrófono que se esté usando.

- Construir un prototipo funcional en equipos VR/AR como las gafas Quest 3. Este objetivo se cumplió utilizando un servidor y la API de AssemblyAI con A-Frame. La funcionalidad, la detección de voz y las transcripciones son buenas; el principal inconveniente reside en la interacción. Para cada comando se debe iniciar y detener la grabación de voz para su envío al servidor. Este proceso, cuando sabes usar la aplicación, no es problemático, pero para usuarios nuevos resulta en una latencia muy alta, afectando a la fluidez de la experiencia.

- Utilizar para la construcción el framework A-Frame, porque se adapta especialmente bien a los objetivos anteriores. Este objetivo se cumplió. A-Frame demostró ser un framework

adecuado para el desarrollo de la aplicación en ambas plataformas, facilitando la creación de escenas 3D inmersivas y la integración con las diferentes APIs de reconocimiento de voz.

En conclusión, el objetivo general del proyecto se considera que se ha cumplido de manera significativa, ya que se exploraron diversas técnicas de edición por voz y se desarrolló un prototipo funcional que demuestra la viabilidad de la interacción por comandos de voz para la creación y edición de escenas 3D en navegadores web y dispositivos VR/AR. La mayoría de los objetivos específicos se cumplieron total o parcialmente, sentando una base sólida para futuras investigaciones y desarrollos en esta área. Las limitaciones de tiempo impidieron la implementación completa de todas las funcionalidades planeadas y una modularidad más exhaustiva del prototipo.

6.2. Esfuerzo y recursos dedicados

Se ha dividido el proyecto por sprints y se ha hablado en cada sprint de su duración en semanas, pero aún no destaca el tiempo que se le ha dedicado. En el Cuadro6.1 se observará el tiempo dedicado en horas cada sprint.

Cabe recalcar que el proyecto se empezó a inicios de diciembre y se ha ido compaginando con el trabajo y las semanas de vacaciones. Por ello, se ha trabajado en la parte del código durante 4 meses aproximadamente y 1 mes más en la parte de la memoria, como se observa en el diagrama de Gantt (ver Figura 6.1).

SPRINTS	Tiempo absoluto	Tiempo calendario	Observaciones
Sprint 1	31 Horas	3 Semanas	En esta fase gran parte del trabajo fue leer documentación y comprenderla.
Sprint 2	18 Horas	2 Semanas	La mayor parte del esfuerzo se empleó en explorar el manejo y creación de componentes para crear objetos de A-Frame.
Sprint 3	21 Horas	2 Semanas	Gran parte del tiempo se empleó en hacer la comunicación entre componentes para que funcionasen correctamente.
Sprint 4	16 Horas	2 Semanas	La creación de componentes extras y modificación de escena fue muy educativa, pero de la misma manera compleja.
Sprint 5	6 Horas	1 Semana	Crear el servidor con todos sus certificados para una conexión segura fue de lo más desafiante.
Sprint 6	37 Horas	3 Semanas	La implementación de un componente para visualizar HTML incrustado en un panel fue muy interesante durante este sprint.
Memoria	entre 3 y 4 horas diarias	4 Semanas	Fue la parte más compleja y extensa del desarrollo del proyecto.

Cuadro 6.1: Tiempo dedicado

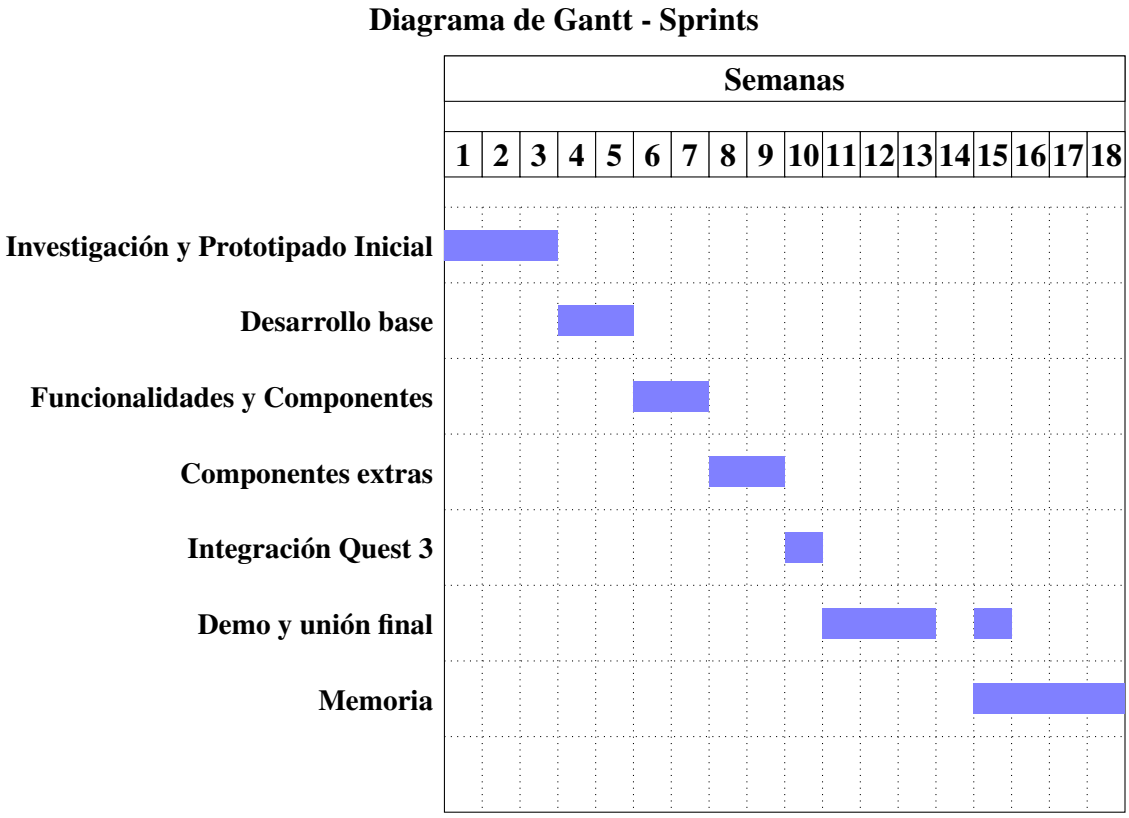


Figura 6.1: Cronograma del desarrollo del proyecto

6.3. Aplicación de lo aprendido

- **Informática I y II:** Estas asignaturas crearon una base sólida en los principios de la programación, como la lógica algorítmica, y la resolución de problemas mediante código. Si bien el lenguaje principal que se usaba en estas asignaturas fue Python, la forma de construcción de un programa, funciones y el conocimiento en general de cómo resolver problemas fueron esenciales para la estructura de los componentes de A-Frame, definir las clases y objetos que representan los elementos de la escena 3D y la lógica detrás del reconocimiento e interpretación de los comandos de voz. La gestión de estados de la aplicación y la manipulación de datos también se basaron en estos fundamentos.

- **Construcción de Servicios y Aplicaciones Audiovisuales en Internet y Laboratorio de Tecnologías Audiovisuales en la Web:** Estas asignaturas fueron cruciales para profundizar en lenguajes de programación Web, centrándose especialmente en JavaScript, HTML y CSS. Los conocimientos adquiridos permitieron comprender la estructura del DOM y cómo manipularlo. Por otro lado, la comprensión de los mecanismos de interacción tanto del lado del cliente (la interpretación de los comandos de voz en el navegador) como del lado del servidor (en la versión de las gafas Quest 3 con AssemblyAI) fue directamente aplicada.

- **Gráficos y Visualización en 3D:** Los conocimientos adquiridos en esta asignatura me proporcionaron un entendimiento sólido de conceptos como la geometría de los objetos, la aplicación de materiales y texturas, la gestión de la cámara y la iluminación en la escena virtual. Esto fue esencial para poder crear y modificar los objetos 3D mediante los comandos de voz.

- **Cursos/Seminario de A-Frame:** En este curso se aprendió a manejar el framework en general, dotando de conocimiento básico sobre primitivas, objetos 3D diferentes, animaciones, etc. Gracias a esto se tenían nociones básicas de cómo funciona la construcción de una escena, el posicionamiento de objetos y características.

6.4. Lecciones aprendidas

A lo largo del desarrollo de este proyecto, se ha adquirido un conocimiento variado y profundo en tecnologías involucradas en la creación de la aplicación.

- Se ha explorado cómo desarrollar una interfaz de usuario dirigida por voz.
- Se ha profundizado en el uso de A-Frame principalmente, necesario para la creación de experiencias en realidad virtual.
- Se indagó profundamente en la construcción de componentes para A-Frame, con diferentes funcionalidades.
- Se profundizó en diferentes programas de detección de voz y sus aplicaciones.
- Se experimentó la necesidad de realizar cambios estructurales en la aplicación debido a restricciones tecnológicas o de compatibilidad, y cómo esto impacta el diseño general.
- Creación de un servidor privado para conexiones mediante HTTPS.
- Se desarrollaron habilidades en la depuración y solución de problemas complejos
- Trabajar con dispositivos de VR/AR como las Quest 3.
- Se mejoró el manejo de JavaScript, aunque aplicado al entorno de A-Frame.
- Se ha aprendido a utilizar LaTeX para la realización de documentos de investigación y técnicos como es la memoria.

6.5. Trabajos futuros

En este apartado se van a mencionar posibles mejoras e ideas que se pueden implementar para mejorar esta aplicación:

- Se puede mejorar la detección de comandos encadenados, de tal manera que si detectasen en una frase todos los atributos necesarios, no haría falta la respuesta del componente para pedir cada dato. Esto ayudaría al dinamismo para la creación de escenas, además de evitar tiempos de espera entre comandos.

- Se pueden añadir comandos para poder editar todos los parámetros de cada objeto primitivo en la librería de A-Frame. Por si es necesaria la super edición de objetos.

- Se podría integrar un modelo de lenguaje para que el programa entienda diminutivos, sinónimos, etc.

- En el tema visual, se podría añadir una interfaz que muestre al usuario qué propiedades puede modificar dentro de una entidad. De esta manera, ayudaría a saber cómo acceder y qué valores se pueden dar.

- También en el tema de retroalimentación del programa al usuario, se podría añadir una función de voz (text-to-speech) para que devuelva los mensajes que ahora son de ayuda en texto, mediante la voz.

- Como el objetivo era probar la viabilidad y funcionalidad del reconocimiento de voz y comando, se ha logrado. Para mejorar la usabilidad de la aplicación, se debería solucionar el inconveniente de la grabación por chuncks de audio. Esto implicará una implementación en el servidor para que funcione mediante streaming de manera continua. Esto requeriría una reestructuración completa de la aplicación y un tiempo considerable.

- Finalmente, se podría crear un juego de plataformas simple en donde la dinámica principal para pasar el mapa sea la creación de objetos con colisiones mediante comandos de voz e incluso crear objetos por voz para poder enfrentar enemigos. Esta idea de juego aprovecharía al máximo la creación de objetos y transcripción de comandos, de tal manera que la experiencia de juego se vería enriquecida.

Bibliografía

[1] A-Frame. A web framework for building virtual reality experiences, 2025. Consultado el 6 de mayo de 2025. URL: https://aframe.io/docs/1.7.0/introduction/.

[2] Julius Adorf. Web speech API. *KTH Royal Institute of Technology*, 1, 2013.

[3] Arkio. Arkio - collaborative spatial design, 2025. Imagen obtenida del sitio web oficial de Arkio. URL: https://www.arkio.is/.

[4] Arkio. Arkio workflow documentation, 2025. Documentación oficial sobre las integraciones de flujo de trabajo de Arkio con herramientas como Revit, Unity, Rhino, SketchUp y BIM 360. URL: https://www.arkio.is/workflow/.

[5] AssemblyAI. AssemblyAI - AI models to transcribe and understand speech. Official website of AssemblyAI. URL: https://www.assemblyai.com/docs/speech-to-text/streaming.

[6] World Wide Web Consortium. What is HTML? Introduction to HTML by the W3C. URL: https://www.w3.org/html/.

[7] Jos Dirksen et al. *Learning Three. js: the JavaScript 3D library for WebGL*, volume 3. Packt Publishing Livery Place, UK, 2013.

[8] D. Flanagan. *JavaScript: The Definitive Guide*. Definitive Guides. O'Reilly, 2002. URL: https://books.google.es/books?id=vJGlu9t9LNYC.

[9] Gauchat, Juan Diego. *El gran libro de HTML5, CSS3 y Javascript*. Marcombo, 2012.

[10] GitHub Docs. Acerca de GitHub y Git, 2024. Consultado el 13 de mayo de 2025. URL: https://docs.github.com/es/get-started/start-your-journey/about-github-and-git.

[11] Khronos Group. WebGL overview. Official website of the WebGL standard by Khronos Group. URL: https://www.khronos.org/webgl/.

[12] Leopoly Kft. Shapelab workflows, 2025. Documentación oficial sobre los flujos de trabajo en Shapelab, incluyendo aplicaciones en creación de personajes, impresión 3D, artes visuales y procesamiento de escaneos 3D. URL: https://shapelabvr.com/workflows.

[13] MDN Web Docs. WebGL API - Web APIs — MDN, 2025. Documentation for the WebGL API on MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API.

[14] Meta Platforms. Meta Quest 3, 2024. Consultado el 13 de mayo de 2025. URL: https://www.meta.com/es/quest/quest-3/.

[15] Microsoft. Visual Studio Code Documentation, 2024. Consultado el 13 de mayo de 2025. URL: https://code.visualstudio.com/docs.

[16] Mozilla. Web Speech API - Web APIs — MDN, 2025. Consultado el 5 de Mayo de 2025. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API.

[17] Mozilla Mixed Reality Team. Hello WebXR!, 2025. Experiencia interactiva de realidad virtual desarrollada para celebrar el lanzamiento de la especificación WebXR. URL: https://msub2.github.io/hello-webxr/.

[18] Mozilla Developer Network. Estructuras de datos y tipos en javascript. Documentación sobre tipos de datos primitivos, objetos y arrays en JavaScript por MDN en español. URL: https://developer.mozilla.org/es/docs/Web/JavaScript/Data_structures.

[19] Mozilla Developer Network. ¿Qué es JavaScript? Introducción a JavaScript en MDN en español. URL: https://developer.mozilla.org/es/docs/Learn_web_development/Core/Scripting/What_is_JavaScript.

[20] Node.js Contributors. *Node.js API Documentation*, 2025. Documentación oficial de la API de Node.js, versión 24.1.0. URL: https://nodejs.org/docs/latest/api/.

[21] Node.js Contributors. Node.js JavaScript Runtime, 2025. Repositorio oficial del entorno de ejecución JavaScript Node.js, mantenido por la comunidad y respaldado por la OpenJS Foundation. URL: https://github.com/nodejs/node.

[22] Node.js Contributors. Sobre Node.js, 2025. Descripción oficial del entorno de ejecución JavaScript asíncrono basado en eventos, diseñado para construir aplicaciones de red escalables. URL: https://nodejs.org/es/about.

[23] OpenAI. Whisper, 2024. Consultado el 5 de Mayo de 2025. URL: https://openai.com/es-ES/index/whisper/.

[24] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision, 2024. URL: https://github.com/openai/whisper.

[25] Real o Virtual. Shapelab 2024 (pc), 2024. Ficha del juego Shapelab 2024 en la base de datos de Real o Virtual, con detalles sobre sus características y compatibilidad en realidad virtual. URL: https://www.realovirtual.com/rovdb/juegos/25122/shapelab-2024.

[26] Rob Cram. Star trek bridge crew voice commands demo, 2017. Demostración de juego en realidad virtual de Star Trek: Bridge Crew, publicada por Rob Cram en YouTube. URL: https://www.youtube.com/watch?v=FH4bY4ZG9Pw.

[27] Ken Schwaber and Jeff Sutherland. *The Scrum Guide: A Complete Guide to the Most Popular Agile Framework*. Scrum.org, 2020. Consultado el 16 de Mayo de 2025.

[28] Supereggbert. aframe-htmlembed-component, 2025. Consultado el 23 de Mayo de 2025. URL: https://github.com/supereggbert/aframe-htmlembed-component.

[29] The LaTeX Project. The LaTeX Project, 2025. Consultado el 14 de mayo de 2025. URL: https://www.latex-project.org/.

[30] Three.js Contributors. Three.js Example: WebGL Animation Keyframes, 2025. Demostración de animación de modelos 3D utilizando fotogramas clave en Three.js. URL: https://threejs.org/examples/#webgl_animation_keyframes.

[31] Ubisoft. Voice commands in star trek: Bridge crew, 2025. Artículo de soporte oficial que explica cómo utilizar comandos de voz en el juego Star Trek: Bridge Crew. URL: https://www.ubisoft.com/en-us/help/star-trek-bridge-crew/gameplay/article/voice-commands-in-star-trek-bridge-crew/000060554.

[32] W3C. WebXR Device API, 2024. [Accedido: 7 de mayo de 2025]. URL: https://www.w3.org/TR/webxr/.

[33] WebVR Contributors. WebVR - Bringing Virtual Reality to the Web, 2025. WebVR is an open specification that makes it possible to experience VR in your browser. The goal is to make it easier for everyone to get into VR experiences. URL: https://webvr.info/.

[34] Xenova. Whisper web - robust speech recognition in the browser, 2022. Hugging Face Space, Consultado el 6 de Mayo de 2025. URL: https://huggingface.co/spaces/Xenova/whisper-web/tree/main.

[35] Xenova. Whisper web - robust speech recognition in the browser, 2022. Hugging Face Space, Consultado el 6 de Mayo de 2025. URL: https://huggingface.co/spaces/Xenova/whisper-web.