



**ESCUELA DE INGENIERÍA DE FUENLABRADA**

**GRADO EN INGENIERIA EN SISTEMAS  
AUDIOVISUALES Y MULTIMEDIA**

**TRABAJO FIN DE GRADO**

**SIMULADOR DE REDES DE ORDENADORES EN  
REALIDAD EXTENDIDA**

Autor : Enrique Estebaranz Redondo

Tutor : Dr. Jesús María González Barahona

Curso académico 2023/2024



# **Trabajo Fin de Grado/Máster**

Simulador de Redes de Ordenadores en Realidad Extendida

**Autor :** Enrique Estebaranz Redondo

**Tutor :** Dr. Jesús María González Barahona

La defensa del presente Proyecto Fin de Carrera se realizó el día                de  
de 2024, siendo calificada por el siguiente tribunal:

**Presidente:**

**Secretario:**

**Vocal:**

y habiendo obtenido la siguiente calificación:

**Calificación:**

Fuenlabrada, a                de                de 2024



©2024 Enrique Estebaranz Redondo

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  
mis padres y mi hermana*



# Agradecimientos

Quiero expresar mi más profundo agradecimiento a mis padres, quienes siempre han sido un pilar fundamental en mi vida. Ellos siempre me han dicho que mi educación es lo mas importante y el legado que llevare siempre conmigo y el cual me dejan, su apoyo constante y los sacrificios para garantizar que esta fuera una prioridad no tienen medida. Gracias por creer en mí y por estar a mi lado en cada paso de este camino.

No puedo dejar de mencionar a mi hermana, cuyo apoyo en los momentos más difíciles ha sido indispensable para mis éxitos. Su presencia y confianza en mis capacidades han sido una fuente de motivación.

También debo un enorme gracias a mis amigos, por todos esos momentos de risas y desconexión haciendo rutas por la montaña y jugando. Me ayudaron a mantener un equilibrio durante las épocas de exámenes y entregas.

Mi gratitud se extiende también a mi tutor Jesús María González Barahona, cuya orientación y sabiduría han sido esenciales para realizar este Trabajo Fin de Grado. Su paciencia y dedicación guiaron y también enriquecieron mi experiencia de aprendizaje.



# Resumen

En este Trabajo Fin de Grado se muestra una forma novedosa para visualizar redes de ordenadores de otra maneras, utilizando realidad virtual. Este simulador permite ver la evolución de como los paquetes se mueven en la red a lo largo del tiempo, en la que cualquier usuario puede interactuar con la red de una manera fácil, intuitiva y visualmente rica.

Se ha aprovechado las ventajas que proporciona la realidad extendida para representar algo tan complicado en 2D como es el envío de paquetes a través de redes de ordenadores, donde se requiere de la visualización de dos escenarios para entenderlo, uno correspondiente al mapa de comunicaciones o topología de redes y el otro que muestre el envío de paquetes dentro de el. La realidad extendida ha favorecido la visualización del proceso en un solo escenario proporcionando una experiencia inmersiva e interactiva la cual facilita que las redes de ordenadores sean mas accesibles y comprensibles. De esta manera el proyecto ayuda a cualquier tipo de usuario, principalmente estudiantes y profesionales a entender mejor la topología de las redes, el flujo de datos y como distintas variaciones de la red pueden afectar a esta.

El simulador ha sido desarrollado utilizando la biblioteca web A-Frame, esta permite crear experiencias de realidad virtual y aumentada directamente en el navegador. A-Frame está construida como una extensión de HTML en lenguaje JavaScript, complementándose con Three.js para el manejo de gráficos en 3D. También se ha utilizado otros lenguajes de programación como es Python, para convertir a formato JSON la información que necesita el simulador. Esta información que agregamos en el simulador es extraída de la interfaz gráfica NetGUI, con la cual dibujamos la topología de la red y nos proporciona un archivo con los datos, logrando un simulador de redes capaz de simular cualquier escenario creado con NetGUI junto unos paquetes que definamos.



# Summary

This Final Degree Project shows a novel way to visualise computer networks in another way, using virtual reality. This simulator allows us to see the evolution of how packets move in the network over time, in which any user can interact with the network in an easy, intuitive and visually rich way.

The advantages provided by extended reality have been used to represent something as complicated in 2D as the sending of packets through computer networks, where the visualisation of two scenarios is required to understand it, one corresponding to the communications map or network topology and the other showing the sending of packets within it. Extended reality has favoured the visualisation of the process in a single scenario providing an immersive and interactive experience which makes computer networks more accessible and understandable. In this way the project helps any kind of user, mainly students and professionals, to better understand network topology, data flow and how different variations of the network can affect it.

The simulator has been developed using the A-Frame web library, which allows you to create virtual and augmented reality experiences directly in the browser. A-Frame is built as an extension of HTML in JavaScript language, complemented by Three.js for handling 3D graphics. Other programming languages such as Python have also been used to convert the information needed by the simulator to JSON format. This information that we add in the simulator is extracted from the NetGUI graphical interface, with which we draw the network topology and provides us with a file with the data, achieving a network simulator capable of simulating any scenario created with NetGUI together with some packages that we define.



# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Contexto . . . . .	2
1.2. Objetivo General . . . . .	2
1.3. Objetivos Específicos . . . . .	3
1.4. Estructura de la memoria . . . . .	4
1.5. Mas información . . . . .	5
<b>2. Tecnologías utilizadas</b>	<b>7</b>
2.1. A-Frame . . . . .	7
2.2. Three.js . . . . .	11
2.3. WebGL . . . . .	12
2.4. WebXR . . . . .	13
2.5. JavaScript . . . . .	14
2.6. HTML5 . . . . .	14
2.7. DOM . . . . .	15
2.8. NetGUI . . . . .	16
2.9. Python . . . . .	17
2.10. JSON . . . . .	17
2.11. GitHub . . . . .	18
2.12. Visual Studio Code . . . . .	19
2.13. LaTeX . . . . .	20
2.14. Realidad Extendida . . . . .	21
2.15. Gafas realidad extendida Meta Quest . . . . .	22

<b>3. Desarrollo del proyecto</b>	<b>25</b>
3.1. Sprint 1: Primer contacto . . . . .	26
3.1.1. Objetivo . . . . .	26
3.1.2. Tarea . . . . .	26
3.2. Sprint 2: Trabajo con animaciones . . . . .	28
3.2.1. Objetivo . . . . .	29
3.2.2. Tarea . . . . .	29
3.3. Sprint 3: Comienzo del proyecto . . . . .	30
3.3.1. Objetivo . . . . .	30
3.3.2. Tarea . . . . .	31
3.4. Sprint 4: Creación de paquetes y puesta en movimiento . . . . .	36
3.4.1. Objetivo . . . . .	37
3.4.2. Tarea . . . . .	37
3.5. Sprint 5: Implementación de historial y mejora visual . . . . .	39
3.5.1. Objetivo . . . . .	39
3.5.2. Tarea . . . . .	40
3.6. Sprint 6: Interactividad con la simulación . . . . .	45
3.6.1. Objetivo . . . . .	45
3.6.2. Tarea . . . . .	45
3.7. Sprint 7: Creación de escenario final . . . . .	49
3.7.1. Objetivo . . . . .	49
3.7.2. Tarea . . . . .	50
<b>4. Descripción del resultado</b>	<b>53</b>
4.1. Descripción para usuario . . . . .	53
4.1.1. Uso General del Simulador . . . . .	53
4.1.2. Composición de Escenarios . . . . .	60
4.2. Arquitectura general . . . . .	65
4.2.1. Componente Simulacro . . . . .	66
4.2.2. Componente Envio-Paquetes . . . . .	70
4.2.3. Componente Pausa y Reproducir . . . . .	73

4.2.4. Componente Historial . . . . .	74
4.2.5. Componente Trazador . . . . .	75
4.2.6. Componente MostrarHistorial . . . . .	76
4.2.7. Componente Paquetes-Enviados . . . . .	77
4.2.8. Componentes y Funciones Auxiliares . . . . .	78
<b>5. Conclusiones</b>	<b>81</b>
5.1. Consecución de objetivos . . . . .	81
5.2. Aplicación de lo aprendido . . . . .	83
5.3. Lecciones aprendidas . . . . .	84
5.4. Esfuerzo realizado . . . . .	85
5.5. Trabajos futuros . . . . .	86
<b>Bibliografía</b>	<b>87</b>



# Índice de figuras

2.1. Referencia a entidades con selector de consulta . . . . .	8
2.2. Comunicación entre entidades usando eventos . . . . .	9
2.3. Manipulación DOM . . . . .	9
2.4. Selector de Atributos . . . . .	10
2.5. Se realizan las funciones escena, cámara y renderizador. . . . .	11
2.6. En estas líneas se modela la geometría y su aspecto. . . . .	11
2.7. Se agrega luces. . . . .	12
2.8. Se anima a un cubo a rotar. . . . .	12
2.9. Estructura HTML y estructura DOM . . . . .	16
2.10. Repositorio de GitHub del proyecto. . . . .	19
2.11. Extensiones de VS Code que han sido necesarias . . . . .	20
2.12. Gafas de realidad virtual modelo Meta Quest 3 . . . . .	22
3.1. Escenario de la demo CambiaAspecto. . . . .	27
3.2. Menú prueba 2 . . . . .	28
3.3. Se aprecia como se han generado 8 cubos y ya se ha interactuado con 6 de estos cubos. . . . .	28
3.4. Código utilizado para marcar el camino de la animación . . . . .	29
3.5. Animación de paquetes entre nodos con pausa y reanudar . . . . .	30
3.6. Escenario de simulación de redes en NetGUI . . . . .	32
3.7. Posición y enfoque inicial de la cámara en el mundo A-Frame . . . . .	33
3.8. Representación en 3D del escenario de una simulación de redes. . . . .	36
3.9. Simulación dinámica de tráfico de redes de ordenador en un entorno 3D. . . . .	39
3.10. Primera versión del simulador con su historial. . . . .	41

3.11. Captura extraída del contenido de la asignatura Arquitectura de internet . . . . .	42
3.12. Versión mejorada de la primera versión del simulador con su historial. . . . .	44
3.13. Botón pausa y reproducir fijados en la cámara de A-Frame . . . . .	45
3.14. Simulador interactivo de redes de ordenadores en realidad extendida. . . . .	49
3.15. Se aprecia el historial de paquetes recibidos en el sistema “dnsnet” . . . . .	51
3.16. Controles de la simulación, en este caso el estado es pausado y sonando . . . . .	52
4.1. Vista según se inicia la simulación. . . . .	54
4.2. Transcurso de tiempo en la simulación. . . . .	54
4.3. Simulador desde distintas perspectivas . . . . .	55
4.4. Aspecto del botón de reinicio. . . . .	56
4.5. Botón aspecto reanudar y botón aspecto pausa. . . . .	56
4.6. Botón aspecto sonando y silencio. . . . .	57
4.7. Historial de paquetes en el nodo “dnscom”. . . . .	57
4.8. Ejemplo de interacción al pulsar el router r1. . . . .	58
4.9. Se ha clicado el paquete 11 y únicamente se ve su historial en la escena. . . . .	58
4.10. Modo “historial” con 3 paquetes seleccionados. . . . .	59
4.11. Herramienta NetGUI. . . . .	61
4.12. Fichero “netgui.nkp”. . . . .	61
4.13. Generamos el archivo netgui.json . . . . .	62
4.14. Archivo “packages.nkp” a configurar. . . . .	62
4.15. Ejemplo de dos paquetes configurados. . . . .	64
4.16. Generamos el archivo package.json. . . . .	64
4.17. Extensión Live Server de Visual Estudio . . . . .	64
4.18. Función “iniciarSimulacon” de la componente“simulacro” . . . . .	66
4.19. Código Fetch de la función “construirmapa” . . . . .	67
4.20. Agrego la entidad “NKCompaq” . . . . .	68
4.21. Código en el que se conectan entidades según las conexiones . . . . .	69

# **Capítulo 1**

## **Introducción**

La constante evolución de las Tecnologías de la Información de Comunicación provoca la aparición de conceptos cada vez más complejos de comprender. Para entender estos conceptos conviene contar con herramientas innovadoras que faciliten su comprensión y manejo. Un ejemplo de esto son el campo de las redes informáticas, donde la visualización y la interactividad pueden mejorar significativamente el proceso de aprendizaje. Este Trabajo Fin de Grado aborda dicho problema en un entorno muy concreto que es el de la simulación de redes de ordenadores, tema recurrente que se trata en varias asignaturas de mi carrera y que se entiende mucho mejor a través de la visualización de casos prácticos. Es por ello que nos hemos centrado en hacer un modelo para simular las redes de ordenadores en Realidad Extendida.

La simulación de redes es un problema muy habitual que afecta a quienes están tratando de comprender la red y a los profesionales que trabajan con ella, por lo tanto este simulador tiene aplicación en un contexto estudiantil y profesional porque permite comprender el funcionamiento de las redes de ordenadores en distintas situaciones.

La necesidad de superar estas limitaciones para todas las personas que puedan estar interesadas en el tema, fueron los motivos por los que me decante para enfocar mi Trabajo Fin de Grado en desarrollar un simulador de redes interactivo en realidad extendida, es decir, combina la realidad virtual con la realidad aumentada. Este simulador además de ser mas intuitivo y envolvente; también es, como se ha dicho antes, interactivo, ya que se puede visualizar de distintas formas la topología de las redes y analizar el flujo de datos en tiempo real, proporcionandole una mejor visión y tridimensional al usuario.

Se ha creado una pagina web<sup>1</sup> del TFG donde se dispone de la memoria, la presentación, el enlace al repositorio de GitHub<sup>2</sup> donde se almacena todo el proyecto, las demos resultantes y un vídeo de las demos en funcionamiento.

En los siguientes puntos trataremos el contexto del proyecto mas a fondo, la motivación y objetivos de este y la estructura de la memoria.

## 1.1. Contexto

Es importante comenzar dando una definición sobre lo que es una red de ordenadores, en el ámbito de mi formación y experiencia, he llegado a entender una red de ordenadores como: Un conjunto de equipos de computación que están conectados y envían datos entre si. Esta definición que acabo de dar es incompleta, ya que es importante mencionar los protocolos en las redes, estos son los sistemas de reglas que usan los equipos de la red para transmitir información. Dependiendo del protocolo el envío de paquetes se puede realizar de distintas formas.

Tradicionalmente, los conceptos de redes se enseñan con diagramas estáticos o simuladores limitados que no permiten capturar la naturaleza dinámica y tridimensional que tienen las redes reales. Dejando al estudiante con una comprensión superficial o malentendidos sobre el funcionamiento de las redes.

Para abordar esto, he integrado tecnologías avanzadas como la realidad extendida para la simulación de redes, permitiendo visualizar la topología de la red e interactuar con el envío de paquetes en un entorno virtual tridimensional. De esta manera, los estudiantes pueden ver el impacto que tienen distintas configuraciones a través de la creación de simulaciones específicas para cada una. Además, cualquier otro usuario que trabaje con redes puede aprovechar la herramienta para tener otra forma de visualización.

## 1.2. Objetivo General

El objetivo de mi trabajo fin de grado es desarrollar un simulador de redes de ordenadores interactivo basado en realidad extendida que mejore la comprensión y el aprendizaje sobre

---

<sup>1</sup>Página Web: <https://enriqueestebaranz.github.io/EnriqueEstebaranz/>

<sup>2</sup>Repositorio GitHub: <https://github.com/EnriqueEstebaranz/EnriqueEstebaranz>

el funcionamiento y la gestión de redes informáticas. Este simulador se podrá visualizar en cualquier dispositivo con navegador web y, adicionalmente, con dispositivos de realidad virtual, ofreciendo una experiencia más inmersiva y detallada para aquellos que dispongan de dicho equipo. Esta dualidad de accesibilidad asegura que el simulador pueda ser utilizado en una amplia gama de entornos.

El simulador debe ser capaz de leer escenarios de redes de la interfaz gráfica NetGUI, crear la topología de la red y, en función de la información de los paquetes que introduzcamos, elaborar una animación de los paquetes, generando un historial de estos y permitiendo la interacción por parte del usuario con el escenario.

### 1.3. Objetivos Específicos

A continuación se detallan los objetivos específicos a seguir para el desarrollo de la simulación de redes de ordenadores:

- **Utilización de herramientas de desarrollo front-end:** El simulador debe ser desarrollado utilizando desarrollo web front-end, ya que es el encargado de construir y diseñar la interfaz de usuario de un sitio web. Esto implica trabajar con tecnologías del lado del cliente como HTML y JavaScript, para lo cual es muy útil el framework A-Frame, que permite desarrollar esta aplicación en realidad virtual.
- **Adaptabilidad a diferentes dispositivos:** El programa debe ser accesible en una variedad de dispositivos, incluidos aquellos sin capacidades de realidad virtual, sin necesidad de utilizar complementos o plugins externos.
- **Lectura de datos JSON:** Desarrollar e integrar un mecanismo dentro del simulador que permita leer y procesar eficientemente datos en formato JSON (JavaScript Object Notation).
- **Visualización dinámica de la red:** Desarrollar un programa que pueda crear representaciones visuales tridimensionales de redes a partir de cualquier archivo “netgui.nkp” de NetGUI.

- **Historial de paquetes:** Visualizar el historial para cada paquete que se envía en la simulación.
- **Interfaz intuitiva y fácil de usar:** Crear una interfaz de usuario intuitiva que permita a los usuarios interactuar de manera sencilla con la simulación de redes, sin necesidad de un manual.
- **Experiencia inmersiva:** Integrar elementos visuales y auditivos para mejorar la inmersión y la comprensión del proceso.
- **Interacción con la animación:** Desarrollar y crear funcionalidades de pausa y reanudación dentro del simulador de redes. Así el usuario puede detener en cualquier momento la animación del simulador.
- **Interacciones con la escena:** Al pulsar un paquete o dispositivo debe aparecer información sobre este.
- **Mostrar información relacionada:** Debe aparecer la información dentro de la escena, como cuadros de texto con identificadores de cada sistema y la información del paquete.
- **Creación de prototipos:** Crear demos para poder ver el resultado final con distintos escenarios.

## 1.4. Estructura de la memoria

El resto de la memoria se estructura de la siguiente forma:

- **2 Tecnologías utilizadas:** Se muestran todas las tecnologías utilizadas durante el desarrollo del proyecto, desde la mas importante y con mas peso, hasta las auxiliares. Se da una descripción de la tecnología, junto con algunos ejemplos y el porque de su uso en el proyecto.
- **3 Desarrollo del proyecto:** Se detalla todo el proceso por el que se ha ido pasando para lograr el desarrollo de un simulador interactivo de redes de ordenadores en realidad extendida. Se ha utilizado una metodología de desarrollo inspirada en la metodología

SCRUM de la cual utilizaremos su estructura para contar las distintas etapas por las que se ha pasado.

- **4 Descripción del resultado:** Aquí se muestra el resultado final de tres formas distintas:
  - **Descripción para el usuario:** Se explica qué hay, qué puede hacer y cómo funciona el simulador.
  - **Descripción para el usuario sobre la construcción de una escena:** Se detalla cómo construir una escena sin modificar el código, permitiendo al usuario crear una escena con los paquetes y la geometría que desee.
  - **Descripción de la implementación:** Se describe cómo está construida la versión final, con una explicación de cada componente y su funcionamiento, para que se entienda toda la complejidad del trabajo realizado.
- **5 Conclusión:** Este es el último capítulo, que consiste en las conclusiones de todo el trabajo y aprendizaje realizado, junto con posibles implementaciones para futuros trabajos a partir de este.

## 1.5. Mas información

Se dispone de una pagina web<sup>3</sup> con mas información sobre el proyecto y todo el código se encuentra en un repositorio de GitHub<sup>4</sup>.

---

<sup>3</sup>Página Web: <https://enriqueestebaranz.github.io/EnriqueEstebaranz/>

<sup>4</sup>Repositorio GitHub: <https://github.com/EnriqueEstebaranz/EnriqueEstebaranz>



# **Capítulo 2**

## **Tecnologías utilizadas**

En esta sección, se presentarán diversas tecnologías que han sido fundamentales para el desarrollo del proyecto. A-Frame 2.1 ha sido la tecnología de mas peso, esta construida como una extensión de HTML 2.6 en lenguaje JavaScript 2.5 y gracias a Three.js 2.2 y WebGL 2.3 permite crear entornos 3D en la web. Web XR 2.4 ha permitido la representación de escenas 3D tanto en realidad virtual como aumentada. NetGui 2.8 es una interfaz gráfica que nos proporciona la información de la red, para la cual se utiliza Python 2.9 para convertir dicha información a formato JSON 2.10. GitHub 2.11 ha sido esencial para la gestión del código, facilitando el seguimiento de versiones y ofreciendo un lugar seguro en el que guardar todo. Visual Studio Code 2.12 ha sido el editor de código utilizado, ofreciendo una integración fluida con Git y algunas extensiones como LiveServer. Latex 2.13 se ha utilizado para la elaboración de la memoria del proyecto y las gafas Meta Quest 2.15 han permitido una experiencia inmersiva en realidad virtual.

### **2.1. A-Frame**

A-Frame [1]<sup>1</sup> es la tecnología utilizada con mas peso de este proyecto. Se trata de un framework en JavaScript (Lenguaje de programación que se utiliza para generar webs interactivas) de código abierto, es decir, es un conjunto de herramientas, guías y estructuras predefinidas que colaboran en este caso en el desarrollo de sitios webs proporcionando una experiencia de realidad extendida. A-Frame se basa en HTML, facilitando a los desarrolladores su inicio en

---

<sup>1</sup>Página A-Frame: <https://aframe.io/>

el mundo de la realidad extendida. Esto es porque HTML es un lenguaje de código ordenado, claro y sencillo de aprender, de el se hablará mas tarde.

La forma de operar que tiene A-Frame es utilizando un sistema de componentes de entidad que ofrece un marco estructurado, ampliable y modular para Three.js (Una de las bibliotecas de JavaScript de gráficos 3D mas utilizadas). Este sistema de componentes de entidad utiliza la estructura HTML y el DOM para mejorar la interactividad y modularidad de las aplicaciones ER (Extended Reality).

Gracias a la naturaleza declarativa del framework, a que tiene HTML y a que está basado en el DOM, se pueden aprovechar estas capacidades para proporcionar mejoras en el sistema de componentes de entidad:

- **Selectores de consulta para referenciar y manipular entidades**

```
// Gracias al potente sistema selector de consultas del DOM
// Puedo obtener una entidad por su ID
var entidadId = document.querySelector('#ID');

// Obtener entidades que tengan una clase específica
var entidadClase = document.querySelectorAll('.clase');

// Obtener entidades que tengan una componente específica
var entidadComponente = document.querySelector('[componente]');

// Acceder al elemento de la escena de A-Frame
var escena = document.querySelector('a-scene');

// querySelector devuelve el primer elemento dentro del documento que coincide
// querySelectorAll devuelve todos los elementos que coinciden dentro del documento
```

Figura 2.1: Referencia a entidades con selector de consulta

- **Gestión de eventos para facilitar la comunicación desacoplada entre componentes**

```

<a-entity id="emite" emite-evento></a-entity>
<a-entity id="escucha" escucha-evento></a-entity>

<script>
// Componente que emite un evento
AFRAME.registerComponent('emite-evento', {
  init: function() {
    this.el.addEventListener('click', () => {
      this.el.emit('eventoEmitido', {clicked: true});
    });
  }
});

// Componente que escucha el evento
AFRAME.registerComponent('escucha-evento', {
  init: function() {
    this.el.addEventListener('eventoEmitido', (event) => {
      console.log('Se ha escuchado un evento que han emitido');
    });
  }
});
</script>

```

Figura 2.2: Comunicación entre entidades usando eventos

- **Integración de APIs del DOM para el manejo de elementos:** Permitiendo actualizar (“.setAttribute”), añadir (“.createElement”) o eliminar (“.removeChild”) como se muestra en el siguiente ejemplo.

```

// Actualizo color de una entidad al color azul
var entity = document.querySelector('a-entity');
entity.setAttribute('material', 'color', 'blue');

// Añadir un nuevo elemento a la escena
var caja = document.createElement('a-entity');
caja.setAttribute('geometry', {primitive: 'box', height: 2, width: 2});
caja.setAttribute('position', '0 0.5 0');
document.querySelector('a-scene').appendChild(caja);
// appendChild lo agrega a la escena

// Elimina un elemento de la escena
var entidad = document.querySelector('#entidad');
entidad.parentNode.removeChild(entidad);

```

Figura 2.3: Manipulación DOM

- **Selector de atributos en el filtro de entidades:** El DOM tiene selectores de atributos que permite consultar las entidades que poseen ciertos atributos de HTML.

```
// Seleccionar entidades con un componente específico
var entidadLuz = document.querySelectorAll('[light]');

// Seleccionar entidades que no tienen un atributo específico
var noEntidadAnimacion = document.querySelectorAll(':not([animation])');
```

Figura 2.4: Selector de Atributos

En cuanto al sistema de componentes de entidad, la lógica y los datos se dividen en componentes reutilizables, es decir, puedo asignar estos componentes a cualquier entidad, aun siendo estas entidades individuales en la escena.

Dentro de esta arquitectura tenemos:

- **Entidades:**<sup>2</sup> Son objetos genéricos que se representan mediante la etiqueta “<a-entity>”. No tienen comportamiento propio hasta que se les asignan componentes. Funcionan como contenedores modulares para estos componentes, permitiendo a los desarrolladores manipular, combinar o personalizar sin alterar el código subyacente de los componentes. Facilitando la construcción y organización de escenas complejas de realidad virtual de manera eficiente y estructurada.
- **Componentes:**<sup>3</sup> Son bloques de construcción reutilizables que definen todo (posición, modelo 3D, aspecto, interacciones específicas, etc) de las entidades, aportándoles el comportamiento propio a cada una. Están escritos en JavaScript, permitiendo una personalización detallada a través de HTML. Incluyen propiedades configurables, manejadores para eventos del ciclo de vida y métodos para funcionalidades extra.

A-Frame juega un papel fundamental en el proyecto, proporcionando una plataforma robusta y accesible para el desarrollo de realidad extendida. La integración de HTML facilita la participación de programadores de todos los niveles, mientras que su estructura modular asegura flexibilidad y amplia compatibilidad con otras tecnologías web. En definitiva, el framework es esencial para enriquecer la experiencia educativa, permitiendo explorar y aprender de manera interactiva y profundamente inmersiva.

---

<sup>2</sup>Información entidades A-Frame: <https://aframe.io/docs/1.6.0/core/entity.html>

<sup>3</sup>Información entidades A-Frame: <https://aframe.io/docs/1.6.0/core/component.html>

## 2.2. Three.js

Cuando se habló de A-Frame se menciono la ventaja que tiene este framework al contar con Three.js [2], es una biblioteca de desarrollo de JavaScript que destaca en la creación de entornos 3D en la web utilizando WebGL, del que se hablará en el siguiente punto.

Aquí se destacan cuatro componentes clave de THREE.JS:

- **Escena, cámara y renderizador:** Se coordina la visualización de objetos 3D, donde la escena alberga objetos, la cámara establece las perspectiva desde al cual se observa el escenario y el renderizador produce la imagen final.

```
// Crea una nueva escena 3D.
const escena = new THREE.Scene();

// Crea una cámara con perspectiva.
const cámara = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1, 1000);

// Crea un renderizador WebGL para mostrar la escena en el navegador.
const renderizador = new THREE.WebGLRenderer();
```

Figura 2.5: Se realizan las funciones escena, cámara y renderizador.

- **Geometrías y materiales:** Permiten modelar formas tridimensionales y modificar aspectos visuales, utilizando colores simples o texturas.

```
// Crea una geometría de caja.
const geometria = new THREE.BoxGeometry(1, 1, 1);

// Crea un material con un color básico rojo.
const material = new THREE.MeshBasicMaterial({ color: 0xff0000 });

// Crea un objeto de malla con la geometría y el material especificados.
const cubo = new THREE.Mesh(geometria, material);

// Añade el cubo a la escena.
scena.add(cubo);
```

Figura 2.6: En estas líneas se modela la geometría y su aspecto.

- **Luces y sombras:** Proporciona un realismo y sensación de profundidad al entorno gracias a distintos tipos de iluminación y por ello la creación de sombras.

```
// Crea una luz puntual blanca con intensidad y distancia máxima.
const luces = new THREE.PointLight(0xffffff, 1, 100);

// Establece la posición de la luz.
luces.position.set(10, 10, 10);

// Añade la luz a la escena.
scene.add(luces);
```

Figura 2.7: Se agrega luces.

- **Animación y controles de cámara:** Permiten crear movimientos fluidos y manipular la escena de forma interactiva.

```
// Crea controles de órbita para permitir la interacción con la cámara.
const controles = new THREE.OrbitControls(camera, renderer.domElement);

// Rota el cubo en el eje x.
cube.rotation.x += 0.01;

// Rota el cubo en el eje y.
cube.rotation.y += 0.01;
```

Figura 2.8: Se anima a un cubo a rotar.

Aunque luego en el código del proyecto se utiliza A-Frame para definir prácticamente todo y no se observa explícitamente fragmentos de Three.js, esta biblioteca esta trabajando por detrás, contribuyendo de manera integral al funcionamiento y eficacia del simulador.

### 2.3. WebGL

WebGL [3] [4] es una interfaz de programación de aplicaciones (del inglés API, Application Programming Interface) que permite representar gráficos 3D y 2D directamente en cualquier navegador compatible, evitando la utilización de plugins adicionales. Esta API se ejecuta en el cliente y permite el uso acelerado por hardware de la unidad de procesamiento gráfico de la que disponga el usuario. Es por ello que el programador puede crear escenas 3D complejas con la tranquilidad de que lo renderizará la tarjeta gráfica del usuario que entre en la Web (Se tendrá que tener en cuenta que en función de la potencia gráfica que necesite será más o menos accesibles para la gente).

Aunque es posible programar directamente con WebGL con solo JavaScript, la complejidad que tiene hace necesario el uso de bibliotecas como Three.js o Pixi.js, que facilitan el desarrollo de aplicaciones 3D.

## 2.4. WebXR

WebXR [5] es un conjunto de estándares que permiten la representación de escenas 3D en dispositivos diseñados para mostrar mundos virtuales (Realidad Virtual - VR siglas en inglés) o agregar gráficos al mundo real (Realidad Aumentada - AR siglas en inglés). Desarrollada bajo una especificación abierta por el consorcio World Wide Web (W3C), facilita la creación de experiencias inmersivas directamente en navegadores web sin necesidad de aplicaciones adicionales.

WebXR proporciona las siguientes capacidades:

- Encontrar dispositivos de salida de realidad virtual o aumentada compatibles.
- Renderizar una escena 3D a una velocidad de fotogramas adecuada.
- Mostrar la salida en una pantalla 2D.
- Rastrear y representar con precisión los movimientos y gestos de los dispositivos de entrada, como los mandos de un casco de realidad virtual.

Por otro lado WebXR es compatible con un gran numero de navegadores:

- Google Chrome
- Mozilla Firefox
- Microsoft Edge
- Safari
- Meta Quest Brouser

## 2.5. JavaScript

JavaScript [8] [9] [10] es el lenguaje mas utilizado del proyecto. Es un lenguaje de programación empleado principalmente para hacer páginas webs interactivas, ya que es el único que entienden los navegadores. Este lenguaje hizo que las paginas webs no se viesen como simples paginas de un libro y sean dinámicas, permitiendo manipular el DOM y gestionar eventos del usuario, alterando el contenido que se le muestra y comunicando información de manera asíncrona.

Estos son algunos de sus usos mas destacados:

- **Desarrollo web en el cliente:** Se utiliza para crear interacciones en el navegador, formando parte del trabajo frontend.
- **Aplicaciones variadas:** Gracias a NodeJS, se puede usar para desarrollar cualquier tipo de aplicación, desde web hasta sistemas de backend.
- **Apps móviles:** Se emplea en la creación de aplicaciones móviles, ya sean híbridas o convertidas a formato nativo para diferentes dispositivos.
- **Aplicaciones de escritorio:** Permite desarrollar programas para Windows, linux y Mac, usando un solo código que funciona en todas estas plataformas.

A través de JavaScript, se expanden las capacidades de A-Frame, brindando a los desarrolladores la posibilidad de diseñar componentes a medida que responden a las acciones del usuario, controlan animaciones y gestionan datos en tiempo real. Estas funciones son vitales para un simulador de redes que busca ser no solo visual, sino también interactivo y operativo.

## 2.6. HTML5

HTML versión 5 [11] [7] es la versión mas reciente del lenguaje de marcado HTML (HyperText Markup Language), este no es un lenguaje de programación, ya que no posee estructuras propias de estos lenguajes (bucles, funciones, condiciones, etc). Sirve para definir la estructura básica de una página web, facilitando la integración con lenguajes de backend y otras tecnologías como CSS Y JavaScript.

Es esencial para la creación de páginas web debido a los siguientes factores:

- **Universal:** Es compatible con todos los navegadores y por tanto accesible para cualquier dispositivo que disponga de navegador web, permitiendo que los contenidos puedan llegar a todo el mundo.
- **Fácil de aprender:** Gracias a su estructura simple y a las etiquetas intuitivas, HTML es sencillo de aprender y usar.
- **Soporte multimedia:** La versión 5 incluye soporte integrado para elementos multimedia como audio y vídeo, permitiendo a los programadores la incorporación de contenidos multimedia sin utilización de complementos o plugins externos.
- **Integración de tecnologías:** Como ya se ha comentado, HTML se integra perfectamente con otras tecnologías como CSS para el diseño y JavaScript para la funcionalidad, permitiendo crear sitios dinámicos y visuales.

Al utilizar HTML5 en el desarrollador de mi simulador de redes he asegurado una estructura que me permite integrar contenido multimedia, interacciones de manera eficiente y accesible a todo usuario, garantizando que cualquier usuario pueda acceder al simulador sin barreras técnicas.

## 2.7. DOM

Las siglas DOM significan Document Object Model y representan la estructura de los documentos HTML y XML. Fue creado por la compañía Netscape Communications como un conjunto de objetos que permiten a lenguajes de programación como es en nuestro caso JavaScript interactuar con el contenido, la estructura y el estilo de una página web.

En la figura se ve de manera visual como a la derecha se representa el arbol DOM de la estructura HTML de la izquierda



Figura 2.9: Estructura HTML y estructura DOM

Debido al DOM [12] podemos manipular elementos individuales de una página a través de JavaScript (como se muestra en la figura 2.3), proporcionando funcionalidades como la modificación de propiedades, la invocación de métodos y la gestión de eventos. Cada objeto en el DOM tiene la capacidad de representar partes del documento que permite cambios en tiempo real sin necesidad de recargar la página completa.

Aunque existen alternativas modernas como React o Lit, que ofrecen abstracciones para manejar el DOM de manera más eficiente, el uso directo del DOM sigue siendo indispensable en proyectos que requieren una alta interactividad y una respuesta inmediata.

## 2.8. NetGUI

Como no podía faltar en un proyecto dedicado a la simulación de redes, era esencial utilizar alguna herramienta visual e interactiva que facilite la comprensión y gestión de redes y de la que poder apoyarse en su desarrollo. Elegí NetGUI [13] ya que estoy familiarizado con ella por su uso en varias asignaturas de la carrera y proporciona una plataforma robusta para la visualización y configuración de topologías red.

NetGUI es la interfaz gráfica para Netkit, el cual es un software que simula redes de ordenadores virtuales utilizando máquinas virtuales (User-mode Linux), permitiendo operar varios nodos virtuales como ordenadores y routers sin necesidad de hardware real.

Estas son las funcionalidades que tiene:

- **Diseño de redes por selección y arrastre de componentes.**
- **Guardado y carga de configuraciones de red.**

- **Conexión fácil de dispositivos de red.**
- **Configuración y arranque de hardware emulado.**
- **Gestión de la red mediante consolas Linux.**

Gracias a esta herramienta se obtiene un fichero “netgui.nkp” que contiene una descripción del dibujo del escenario de red creado. El fichero nos servirá de gran ayuda para la generación de la escena dentro del proyecto, ya que se trabajara para adaptarlo a la realidad extendida.

## 2.9. Python

Python [14] [6] es un lenguaje de programación orientado a objetos, de alto nivel y de código abierto, creado por Guido van Rossum en 1991. Conocido por su legibilidad y simplicidad sintáctica, facilitando la escritura de código claro y lógico, muy útil para grandes proyectos. Se utiliza en distintas áreas como, desarrollo web, análisis de datos, inteligencia artificial, aprendizaje automático (más conocido como machine learning) y automatización de scripts.

Uno de los aspectos más poderosos y por el que se ha usado es su sistema de módulos, que son simplemente archivos Python con extensión .py que contienen definiciones y declaraciones de funciones, clases y variables. Estos módulos pueden ser importados y utilizados en otros programas de Python, lo que permite la reutilización del código y la organización del programa en componentes manejables.

En el proyecto se ha utilizado el módulo ”json”de Python, útil para realizar la conversión de datos entre formatos que son legibles por humanos y máquinas. Este módulo proporciona una forma de codificar y decodificar datos en formato JSON del que se habla mas abajo. En el contexto del proyecto ha sido esencial para adaptar el archivo “netgui.nkp” que nos proporcionaba la herramienta NetGUI a JSON favoreciendo una manipulación eficiente y un almacenamiento conveniente de la configuración de las redes simuladas.

## 2.10. JSON

JSON, acrónimo de JavaScript Object Notation como su nombre indica forma parte del sistema de JavaScript y deriva de su sintaxis, su función es facilitar el acceso, almacenamiento e

intercambio de datos. Uno de los motivos por los que JSON es tan importante y Python dispone de este modulo es la comunicación entre el backend desarrollado por Python y el frontend, asegurando que la información se transmita de manera eficaz y coherente.

Se utilizó JSON para configurar y almacenar los detalles de las redes virtuales, como posición, propiedades, tipo de dispositivo, conexiones y envíos de paquetes. Logrando una carga rápida de diferentes escenarios de red dentro del simulador.

## 2.11. GitHub

GitHub es un portal creado para alojar el código de las aplicaciones de cualquier desarrollador que utiliza el sistema de control de versiones Git.

- **Git:** Diseñado por Linus Torvalds, es un sistema de control de versiones que permite a los desarrolladores administrar su proyecto, facilitando la colaboración, el seguimiento de versiones y la preservación de copias de las versiones anteriores.

Es una plataforma de hospedaje para proyectos de software que facilita la gestión de repositorios públicos gratuitos para código abierto (también para código privado a través de una suscripción). Permite la colaboración entre usuarios para mejorar el código, reportar errores y compartir mejoras a través de herramientas como wikis, seguimiento de problemas, revisión de código, etc. También funciona como red social donde puedes seguir a otros desarrolladores para mantenerte actualizado de sus proyectos. Actualmente, según las estadísticas Clave de GitHub en 2024 alrededor de 100 millones de desarrolladores de todo el punto utilizan el portal.

Se ha elegido esta plataforma porque es una de las mas utilizadas en la comunidad de desarrolladores y ya se nos ha familiarizado con ella a lo largo de toda la carrera, en la siguiente figura se aprecia el repositorio donde se encuentra el proyecto del TFG dentro de GitHub<sup>4</sup>:

---

<sup>4</sup>Repositorio GitHub: <https://github.com/EnriqueEstebaranz/EnriqueEstebaranz>

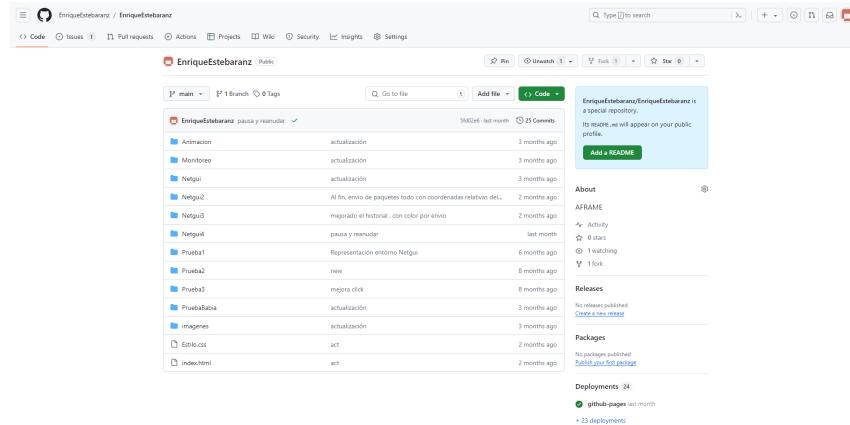


Figura 2.10: Repositorio de GitHub del proyecto.

## 2.12. Visual Studio Code

Visual Studio Code (VS Code) es un software libre y multiplataforma de editor de código fuente desarrollado por Microsoft. Muy útil debido a su buena integración con Git, facilitando la subida y recuperación de archivos al repositorio de GitHub, sin necesidad de escribir comando en la terminal y con una interfaz intuitiva y sencilla de entender. Si a esto le sumas la multitud de extensiones que posee tenemos una herramienta con una gran variedad de características útiles para agilizar el desarrollo del proyecto.

A lo largo de mi etapa universitaria he trabajado con distintos editores de código fuente como Atom, Geany y WebStorm. Me he terminado decantando por VS Code por el uso de control de versiones que tiene, la facilidad de uso y algo tan simple como el aspecto visual del editor. Es por ello que este es el editor de código fuente empleado para la realización del Simulador de Redes.

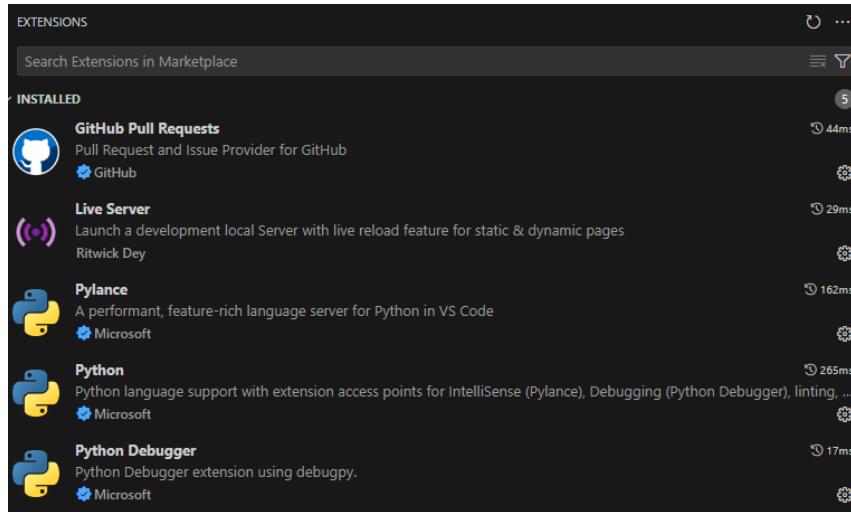


Figura 2.11: Extensiónes de VS Code que han sido necesarias

Estas son las extensiones que tengo instaladas, destacar Live Server que me permite visualizar con un clic el resultado de todo lo que se está programando en un navegador web.

## 2.13. LaTeX

LaTeX [15]<sup>5</sup> es un sistema tipográfico de composición de textos de alta calidad que contiene funcionalidades encargadas de producir documentos técnicos, ideal para memorias de TFG. Se basa en TeX y ofrece una gran variedad de comandos para insertar elementos técnicos en los textos.

Unas de las características que posee es su arquitectura modular que permite una fácil adición de nuevas funcionalidades y que es de código abierto permitiendo que distintos usuarios participen en la incorporación de nuevas utilidades al lenguaje.

Su proceso de edición se divide en dos fases: la escritura del texto en un editor de texto plano y su compilación para generar el documento final. Para la elaboración de esta memoria se ha utilizado la tecnología LaTeX que se combino con Overleaf<sup>6</sup>, una plataforma de edición en línea que simplifica la escritura y compilación de documentos, así el usuario puede centrarse en el contenido sin preocuparse por el formato.

<sup>5</sup>Información sobre LaTeX <https://www.latex-project.org/>

<sup>6</sup>Overleaf: <https://es.overleaf.com/project>

## 2.14. Realidad Extendida

La realidad extendida (XR) es un término colectivo que engloba varias formas de entornos digitales inmersivos, incluye tres principales tipos de tecnologías:

- **Realidad Virtual:** Sumerge completamente a los usuarios en un entorno virtual. El entorno real es completamente reemplazado por uno virtual, los cascos o gafas los transporta a un mundo virtual.
- **Realidad Aumentada:** Es cuando se superpone información digital en el mundo real, ampliando la realidad sin reemplazarla. Permitiendo a los usuarios ver el mundo real al mismo tiempo que añade imágenes, textos o efectos visuales sobre el entorno, un ejemplo reciente son las personas que se vieron que iban por la calle con la Apple Vision Pro aprovechando esta realidad aumentada para ver un vídeo de Youtube a la vez que ve por donde esta andando.
- **Realidad Mixta:** Combina elementos de realidad virtual con realidad aumentada para crear entornos donde objetos físicos y digitales coexisten e interactúan en tiempo real, existen dos tipos de realidades mixtas:
  - **Mezclando objetos virtuales en el mundo real.**
  - **Mezclando objetos del mundo real en el mundo virtual.**

La realidad extendida (XR) ha evolucionado significativamente desde sus orígenes en la realidad virtual (VR), que se utilizó inicialmente en sectores públicos para entrenamientos en simuladores de vuelo y en las industrias energética y automotriz para simulación y visualización. Estas aplicaciones iniciales de VR requerían supercomputadoras y espacios dedicados como las CAVE de VR. Durante mucho tiempo, la VR fue poco accesible y limitada a instituciones grandes e investigadores. Esto cambió con la llegada de dispositivos como HTC Vive y Oculus Rift que permitió que la VR llegara al consumidor, provocando un impulso de innovación en la tecnología.

Actualmente la realidad extendida está siendo utilizada en una variedad de campos como la educación en donde se está participando con este simulador, la medicina, el entretenimiento, la ingeniería, y el diseño, entre otros, ofreciendo nuevas formas de interacción y visualización de información que amplían las posibilidades del mundo real.

## 2.15. Gafas realidad extendida Meta Quest

Desde la universidad se me ha facilitado las gafas de realidad virtual Meta Quest, para la correcta realización de este proyecto.

Las gafas de realidad virtual Meta Quest representan una línea de dispositivos de realidad virtual autónomos desarrollados por Meta, anteriormente conocida por Facebook. Son un dispositivo que no necesitan de sensores de posición adicionales y permiten su utilización en cualquier lado. Se pueden usar de manera autónoma con una duración aproximada de 2 a 3 horas y también se pueden conectar al ordenador usando un cable de tipo c.

En cuanto a las especificaciones técnicas se destacan:

- El seguimiento, gracias a la tecnología de seis grados de libertad, que permiten realizar un seguimiento de movimientos tanto de la cabeza como del cuerpo adaptándolos con una precisión realista a la realidad virtual.
- Audio posicional 3D integrado directamente en las gafas, que permite escuchar todo lo que te rodea.
- Las lentes son una pantalla LCD de cambio rápido, con 1832 x 1920 de resolución en cada ojo permiten una frecuencia de muestreo de 60, 70 y 90 hercios en el modelo Meta Quest 2, mientras que la versión 3 de las Meta Quest tiene un resolución de 2064 x 2208 permitiendo una frecuencia de muestreo de 72, 80, 90, 120 hercios.

En la siguiente figura 2.12 se muestra los dispositivos que conforman estas gafas de realidad virtual.



Figura 2.12: Gafas de realidad virtual modelo Meta Quest 3

- Gafas Meta Quest 3 con interfaz facial estándar ajustable preinstalada.
- Dos controladores Touch Plus con correas para las muñecas.



# Capítulo 3

## Desarrollo del proyecto

En este capítulo se detallará el proceso de desarrollo del proyecto, describiendo las distintas etapas por las que se han pasado para lograrlo. Para la realización y, sobre todo, la descripción del proyecto, se ha utilizado una metodología de desarrollo inspiradas en las metodologías ágiles, específicamente SCRUM. Esta metodología se caracteriza por dividir el trabajo en ciclos cortos y repetitivos conocidos como “sprints” (duran entre dos semanas y un mes), con el objetivo de entregar prototipos funcionales a final de cada sprint.

El proyecto ha llevado un tiempo aproximado de nueve meses, dedicándole los fines de semana y una hora o dos de lunes a jueves, complementándolo con mi trabajo a tiempo completo y una asignatura. Por el contexto académico y los requisitos del proyecto, el desarrollo se ha inspirado en la metodología SCRUM en los siguientes puntos:

- El equipo esta formado por dos miembros, mi tutor Jesus Maria Gonzalez Barahona que ha tenido un rol de guía y dirección (rol de Product Owner y Scrum Master), optimizando mi trabajo, manteniendo una supervisión y asegurando la aplicación de esta metodología. Por mi parte he tenido un rol mas de desarrollador, encargado en realizar los objetivos que se ponían en común en las reuniones.
- No se definieron todos los objetivos desde el inicio, se creó un Product Backlog (lista ordenada de tareas para el desarrollo de un proyecto) que evolucionaba en función del feedback y lo aprendido durante los sprints.
- Cada sprint se centro en tareas específicas según la prioridad y viabilidad del momento.

- Las reuniones con el tutor por vía teams ha sido el equivalente a las revisiones de Sprint. En estas revisiones se ha realizado un seguimiento y marcado los objetivo a tratar para el siguiente Sprint.

## 3.1. Sprint 1: Primer contacto

Este Sprint es la primera toma de contacto con A-Frame fundamental para el desarrollo del proyecto.

### 3.1.1. Objetivo

El objetivo principal del Sprint 1 es familiarizarnos con A-Frame para entender su estructura, funcionalidad y modularidad proporcionada por las entidades y componentes. Después de esto para aplicar los conocimientos aprendidos se realizan algunos programas de prueba donde se crea nuestra primera componente en A-Frame. Se sigue el siguiente backlog de tareas:

- **Familiarización con A-Frame:** Dedicar un tiempo a conocer los principios fundamentales de A-Frame mediante su documentación oficial y tutoriales.
- **Desarrollo de programas en A-frame:** Crear dos escenas básicas en A-Frame con sus componentes como primera toma de contacto con el framework con el que trabajaremos para desarrollar el proyecto.

### 3.1.2. Tarea

**Programa 1 de prueba:** El primer programa realizado es siguiendo paso a paso un tutorial de Youtube, donde una vez terminado se comenzó a realizar modificaciones al código para visualizar sus efectos en la escena. Principalmente se hizo cambios menores: las texturas, tamaños y agregar algunas entidades. Así es como resulta mi primera escena de A-Frame, a la que llamo “CambiaAspecto” por su funcionalidad <sup>1</sup>.

---

<sup>1</sup>Demo CambiaAspecto: <https://enriqueestebaranz.github.io/EnriqueEstebaranz/Sprint1/CambiaAspecto/>

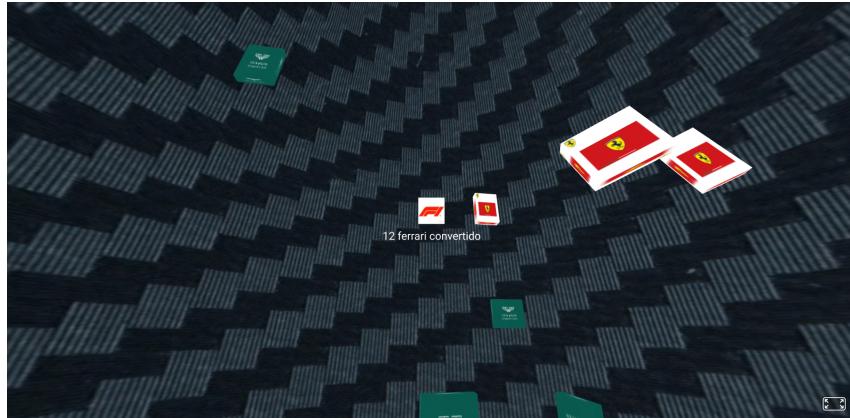


Figura 3.1: Escenario de la demo CambiaAspecto.

Lo que hace este programa es básicamente modificar el aspecto de las entidades cuando pasa el punto central por encima de entidad sin modificar y lleva un seguimiento de las entidades que se han convertido. Como soy un seguidor de la F1 le quise dar una temática de esta competición al escenario.

Esta primera prueba me permitió familiarizarme con las estructura de A-Frame y ver algunas posibilidades de modificación pero era necesario tratar mas a fondo y de una manera menos guiada el funcionamiento de las componentes. Es por ello que se decidió hacer otra prueba para poner en practica los conocimientos aprendidos.

**Programa 2 de prueba:** Este programa <sup>2</sup> se realiza desde cero con el objetivo de crear una componente que a la hora de interactuar cambie de color la entidad. La escena comienza con un pequeño menú donde se debe indicar a través de dos botones "+" y "-" el numero de cubos que se desean generar en la escena y al clicar aceptar se generará dichos cubos con un color aleatorio.

---

<sup>2</sup>Demo CubosARojo: <https://enriqueestebaranz.github.io/EnriqueEstebaranz/Sprint1/CubosARojo/>

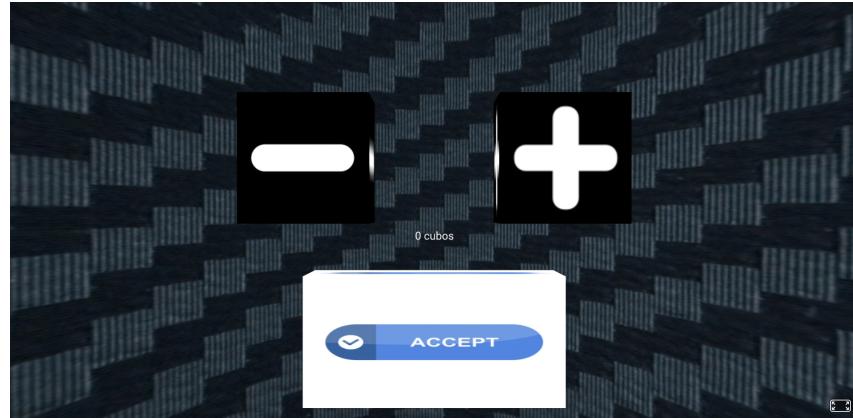


Figura 3.2: Menú prueba 2

Una vez se tiene la escena con los cubos generados podemos interactuar con estos cubos haciendo clic para que se cambien al color rojo.

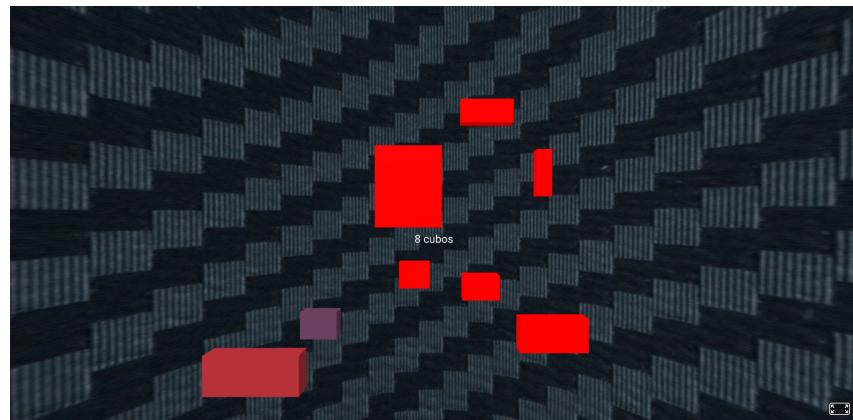


Figura 3.3: Se aprecia como se han generado 8 cubos y ya se ha interactuado con 6 de estos cubos.

Con este segundo programa se ha aprende a manejar los parámetros de las componentes y crear las entidades necesarias para poder interactuar con el ratón del ordenador en la escena.

### 3.2. Sprint 2: Trabajo con animaciones

Esta etapa forma parte de la formación con A-Frame pero ya con vistas a una idea de lo que va a necesitar el proyecto.

### 3.2.1. Objetivo

El objetivo principal es profundizar en el uso de A-Frame, con un enfoque particular en la creación de animaciones, puesto que es crucial el uso de estas animaciones para crear un simulador de redes de ordenadores, dinámico, vistoso e intuitivo. Se sigue el siguiente backlog de tareas:

- **Desarrollo de animación:** Diseñar una escena en la que se realicen distintas animaciones y analizar las propiedades de estas.
- **Diseño de una escena animada mas avanzada:** Crear una escena que imite el envío de paquetes entre ordenadores y permitir que el usuario pueda manejar las animaciones.

### 3.2.2. Tarea

Se han realizado dos programas:

**Prueba animación 1:**<sup>3</sup> No requiere de interacción con el usuario, es una mera animación, donde unos paquetes se activan secuencialmente con intervenciones temporizadas gracias a un componente reloj que he creado. Esta activación hace que los paquetes realicen una serie de movimientos animados que si nos fijamos estos movimientos no son aleatorios, salen los paquetes apilado de un lugar y se apilan en otro manteniendo el mismo orden de salida.

```
activar:function () {
    const contador = this.data.contador;
    if (!this.data.activo) {
        this.data.activo = true;

        const camino = [
            {property: "position", to: `${this.posicionInicial.x + 6} ${this.posicionInicial.y} ${this.posicionInicial.z}`, dur:1000},
            {property: "position", to: `${this.posicionInicial.x + 6} ${this.posicionInicial.y + 2 + 2 * contador} ${this.posicionInicial.z}`, dur:1000},
            {property: "position", to: `${this.posicionInicial.x + 17} ${this.posicionInicial.y + 2 + 2 * contador} ${this.posicionInicial.z}`, dur:1000},
            {property: "position", to: `${this.posicionInicial.x + 17} ${this.posicionInicial.y + 2 * contador} ${this.posicionInicial.z}`, dur:1000}
        ];

        const ejecutarAnimacion = (i) => {
            if (i < camino.length) {
                // Activo la animación correspondiente a i en la cadena
                this.el.setAttribute('animation', camino[i]);
                // Ejecuta una animación cuando se termina la duración de la animación que se activo.
                setTimeout(() => {
                    ejecutarAnimacion(i + 1);
                },camino[i].dur);
            }
        };
        // Inicia la animación secuencial de los movimientos del paquete activo desde el primer paso.
        ejecutarAnimacion(0);
    }
};
```

Figura 3.4: Código utilizado para marcar el camino de la animación

<sup>3</sup>Primera animación: <https://enriqueestebaranz.github.io/EnriqueEstebaranz/Sprint2/Animacion/>

**Animación con interacción:** <sup>4</sup> Esta prueba ya no es una simple animación sin posibilidad de interacción, el usuario puede ponerla en pausa y reproducirla cuando quiera. Se puede valorar esta prueba como una idea muy inicial de representación para el simulador de redes de ordenadores en el que cada barra representa un nodo y se disponen de paquetes moviéndose de nodo a nodo acompañados de un sonido que hacen al llegar al destino y otro que producen al moverse.

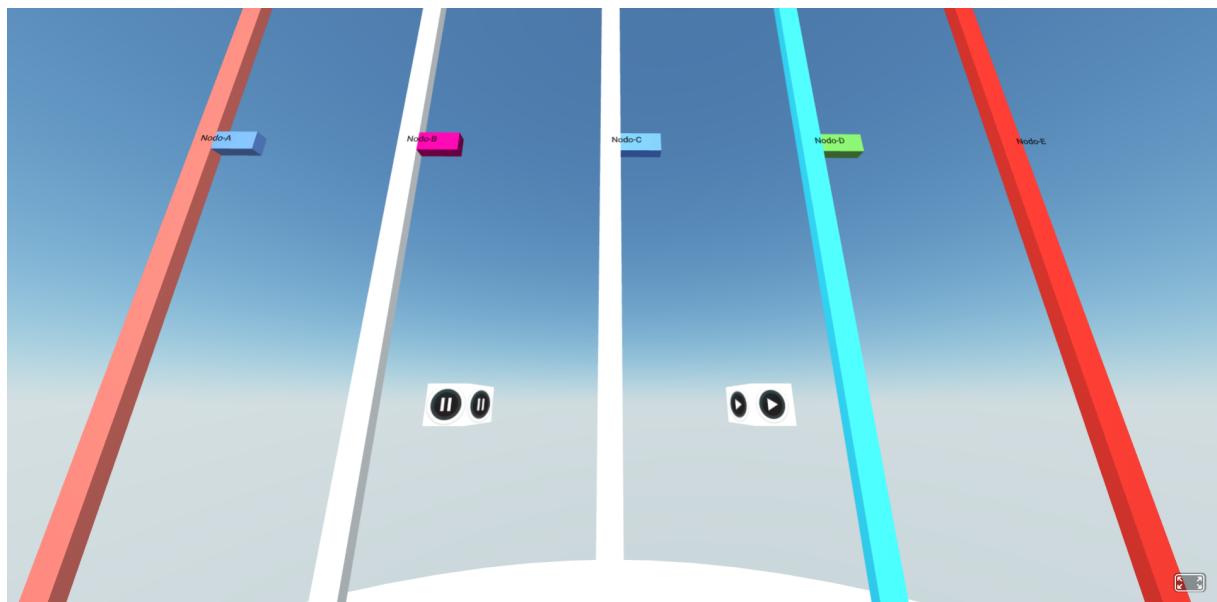


Figura 3.5: Animación de paquetes entre nodos con pausa y reanudar

Tras terminar satisfactoriamente estos dos primeros Sprints se comenzó a especificar algunos de los objetivos del TFG y formalizarlo en el aula.

### 3.3. Sprint 3: Comienzo del proyecto

Se explica todo el proceso realizado para representar el archivo “netgui.nkp” que nos proporciona la herramienta NetGUI en una escena de A-Frame.

#### 3.3.1. Objetivo

En este sprint, el objetivo principal es representar un entorno de simulación de redes de ordenadores basado en NetGUI. Para lograrlo es necesario realizar la siguiente lista de tareas:

---

<sup>4</sup>Primera animación: <https://enriqueestebaranz.github.io/EnriqueEstebaranz/Sprint2/PaquetesIdaVuelta/>

- **Adaptación de las Coordenadas de Nodos:** Se debe adaptar las coordenadas de los nodos del plano 2D de NetGUI al espacio 3D de A-Frame.
- **Conversión a JSON:** Convertir el archivo que proporciona NetGUI con la información de la topología a formato JSON, para una mejor manipulación y transmisión de los datos.
- **Representación de Nodos:** Representar los distintos tipos de sistemas que aparecen en la red con geometrías y texturas apropiadas.
- **Conexiones entre Nodos:** Renderizar las conexiones entre nodos utilizando entidades.
- **Configuración de una demo:** Diseñar un programa en el que ver la simulación de la red de NetGUI en A-Frame.

### 3.3.2. Tarea

**Adaptación de las coordenadas:** El primer paso para representar un entorno de simulación de redes de ordenadores de NetGUI, es diseñar un escenario en su interfaz gráfica y extraer el archivo “netgui.nkp” el cual contiene toda la información del escenario creado (posiciones de los nodos y conexiones que existen entre ellos). En el siguiente fragmento de código se muestra como aparece la descripción de la ubicación de los nodos.

```
<nodes>
position(449.0,154.0); NKHub("hub2")
position(171.0,198.0); NKHub("hub1")
position(820.0,219.0); NKCompaq("dnsnet")
position(5.0,117.0); NKCompaq("dnscom")
position(341.0,224.0); NKRouter("r2")
position(630.0,381.0); NKRouter("r4")
position(315.7491584544007,53.25747062210026); NKCompaq("dnsroot2")
position(560.8754207727998,42.20265163060037); NKCompaq("dnsroot1")
</nodes>
```

La posición es en un plano bidimensional formado por el eje X e Y y todo esto se ha adaptado para implementarlo en nuestro escenario de realidad extendida. En la siguiente figura se muestra una captura del escenario en NetGUI para facilitar el entendimiento del proceso seguido para adaptarlo.

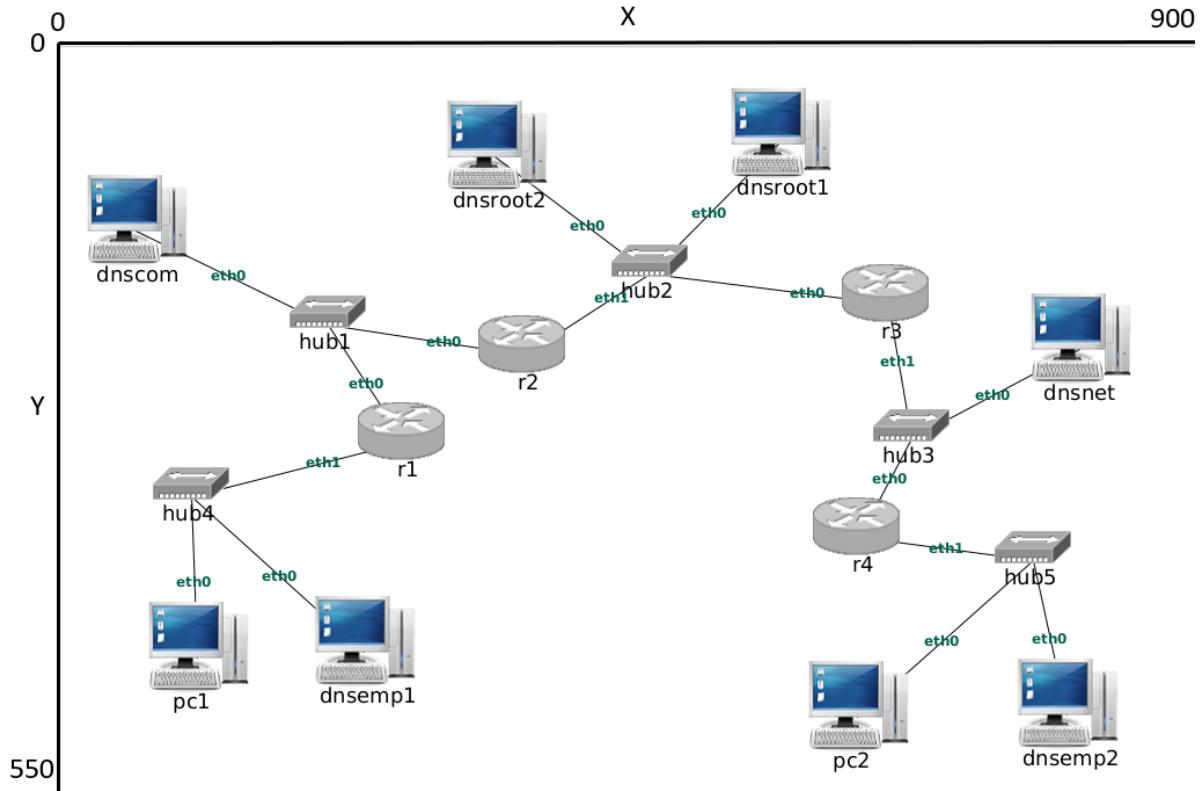


Figura 3.6: Escenario de simulación de redes en NetGUI

- **Reducción de las coordenadas:** Lo primero que se tiene que hacer es adaptar las coordenadas al mundo A-frame, donde se utilizan los metros como unidades. Como el archivo “netgui.nkp” proporciona números muy altos se ha reducido al cinco por ciento dividiendo entre veinte todos los valores, de esta forma reducimos la distancia a algo mas visual y realista.
- **Posicionado centrado al frente del usuario:** En este punto se busca que el escenario se genere en frente de la cámara del usuario, para ello es importante conocer la posición inicial y hacia donde enfoca esta. En la siguiente figura se muestra de manera visual la posición de la cámara y donde esta enfoca, utilizando un cilindro como representación de la cámara.

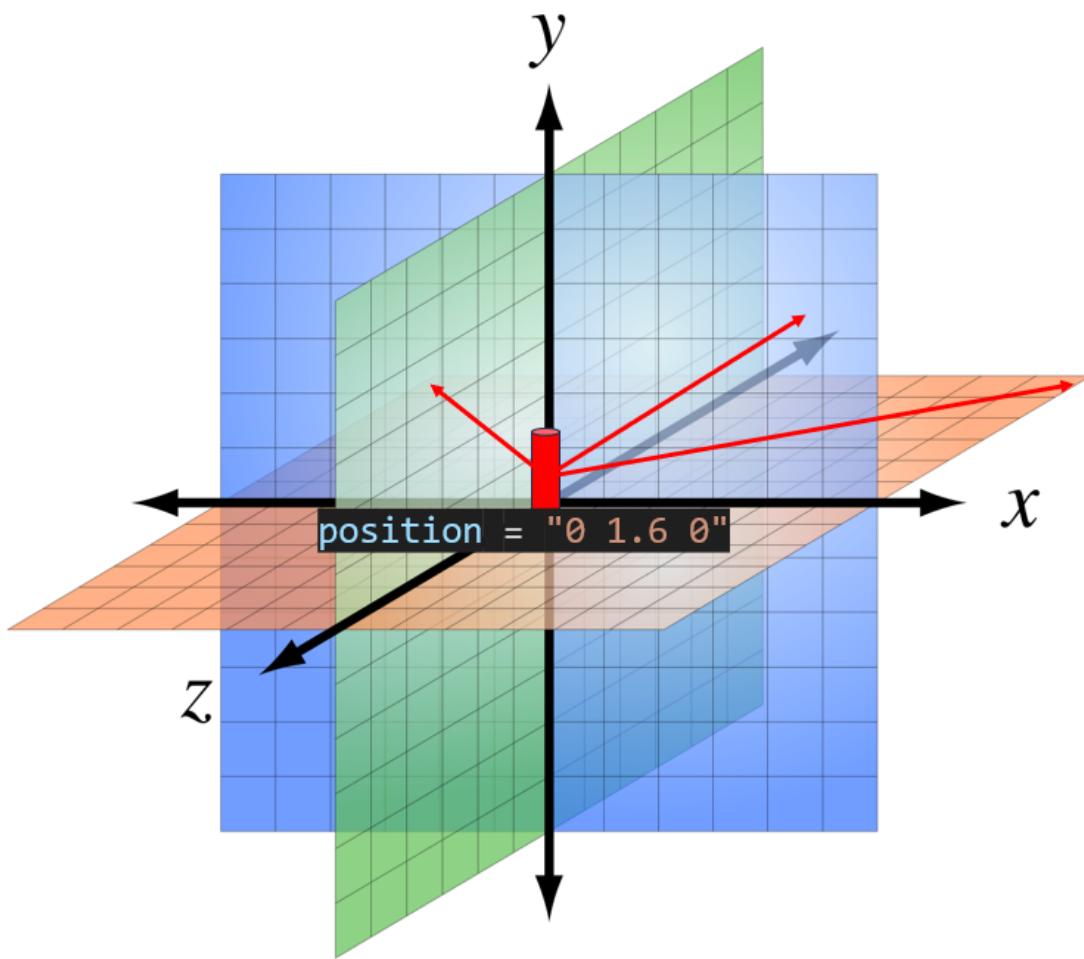


Figura 3.7: Posición y enfoque inicial de la cámara en el mundo A-Frame

Como se puede observar la cámara se encuentra situada prácticamente en la intersección entre los tres planos a una altura (1.6 metros) propia del ser humano y mirando hacia el eje negativo de Z. Con esto, ya se tiene toda la información necesaria para adaptar las coordenadas a lo que se busca:

- Primero se adapta el plano 2D (X,Y) a uno 3D (X,Y,Z) para ello se pasa al plano (X, 0, Z) las coordenadas en el eje X se mantienen en en dicho eje mientras que las correspondientes al eje Y pasan a ser el eje Z en el mundo A-Frame. En la siguiente linea de código se ejecuta el cambio:

```
entidad.object3D.position.set((node.position[0]),0,(node.position[1]));
```

Esta línea es la responsable de establecer la posición de una entidad en el espacio

de la escena. “entidad.object3D” es una instancia de Three.js que proporciona base para las operaciones 3D, en este caso el posicionamiento. “position.set” define la posición de la entidad en las coordenadas (x, y, z), donde “(node.position[0])” es donde se tiene almacenado el valor de X de “netgui.nkp” y “(node.position[1])” el de Y, se aprecia que están representados en el plano X y Z.

- Para que este centrado es suficiente con restarle 25 al eje x ya que va de 0 a 45 (recordemos que lo primero que se hizo fue reducir los valores a un 5 %  $900/20 = 45$ ) y al realizar esta operación se mueve hacia la izquierda las coordenadas del eje x permitiendo su centrado.
- Por ultimo, para que aparezcan de cara todos los valores de la componente Y que posteriormente se pasarán a la Z deben ser negativos, por lo que si van de 0 a 28 con restar 30 al valor ya se tiene.

A continuación se comparte la línea donde se ejecutan estos dos puntos junto con la reducción al 5 %:

```
x, y = (float(position_parts[0])/20)-25, (float(position_parts[1])/20)-30
```

Una vez se ha explicado el proceso seguido y los puntos claves para lograr adaptar el fichero “netgui.nkp” de una representación 2D a una 3D toca explicar el como se representa.

**Cambiar formato de archivo “.nkp” a “.json”:** En un principio se trato directamente con el archivo “netgui.nkp” extrayendo los datos importantes y adaptándolos posteriormente. Esta forma fue efectiva puesto que se lograba el resultado deseado, pero se generaba un gasto computacional evitable, es por ello que se decidió crear un programa en python para aprovechar el módulo “json” que posee a partir del cual permite codificar y decodificar en formato JSON. Dicho formato optimiza la transmisión y manipulación de datos en JavaScript.

**Representación de los nodos y las conexiones:** Para lograr una correcta representación del simulador se ha creado un componente llamado “simulacro” encargado de crear todas las entidades necesarias para representar cada nodo y generar los hilos que representan las conexiones entre nodos. Una vez se tiene toda la información del escenario en JSON a través del método “fetch”, del que se explica su funcionamiento en el capítulo 4.2.1, se extrae los datos que nos proporciona la información para generar las entidades:

- **Hub:** Son todos los nodos que poseen el tipo “NKHub”, se crea una entidad con forma de cubo y textura propia (la imagen de un hub), posicionado en función de los datos adaptados y con un identificador único. Una vez se tiene, lo añadimos a la escena.

```
const entidad = document.createElement('a-entity');
entidad.setAttribute('geometry', {primitive: "box", width:1, height: 0.2});
entidad.object3D.position.set((node.position[0]), 0, (node.position[1]));
entidad.setAttribute("material", {src: "#hub", side: "double"});
entidad.setAttribute("id", node.name);
escenario.appendChild(entidad);
```

- **Router:** Son todos los nodos que poseen el tipo “NKRouter”, se crea una entidad con forma cilíndrica y color gris (simulando un router real), posicionado en función de los datos adaptados y con su propio identificador. Por ultimo se añade a la escena.

```
const entidad = document.createElement('a-entity');
entidad.setAttribute('geometry', {primitive: "cylinder", radius:1, height: 0.5});
entidad.object3D.position.set((node.position[0]), 0, (node.position[1]));
entidad.setAttribute('material', { color: "#808080" });
entidad.setAttribute("id", node.name);
escenario.appendChild(entidad);
```

- **PC:** Aquellos nodos que pertenezcan al tipo “NKCompaq”, se representan como un cubo perfecto con texturas (PC de NetGUI), posicionado según las coordenadas adaptadas y con su propio identificador.

```
const entidad = document.createElement('a-entity');
entidad.setAttribute('geometry', {primitive: "box", width:2, height: 2, depth:2 });
entidad.object3D.position.set((node.position[0]), 0, (node.position[1]));
entidad.setAttribute("material", {src: "#pc", side: "double"});
entidad.setAttribute("id", node.name);
escenario.appendChild(entidad);
```

- **Conexiones:** Se recorre un array llamado “connections” que procede de la información extraída de “netgui.nkp” pasado a JSON y a través de sus identificadores “from” y “to”, si existen, creo una entidad que dibuja una linea entre las posiciones de las entidades que se encuentran en “from” y “to”. Por ultimo las agrego a la escena.

```
connections.forEach(connection => {
  const fromEntity = document.querySelector(`#${connection.from}`);
  const toEntity = document.querySelector(`#${connection.to}`);
  if (fromEntity && toEntity) {
```

```

const line = document.createElement('a-entity');
line.setAttribute('line', {
  start: fromEntity.getAttribute('position'),
  end: toEntity.getAttribute('position'),
});
escenario.appendChild(line);
}
});

```

**Demo:** <sup>5</sup> Finalizando este sprint con el resultado de la representación de la escena inicial extraída de la interfaz gráfica NetGUI en la escena de realidad extendida de A-Frame como se muestra en la siguiente figura.

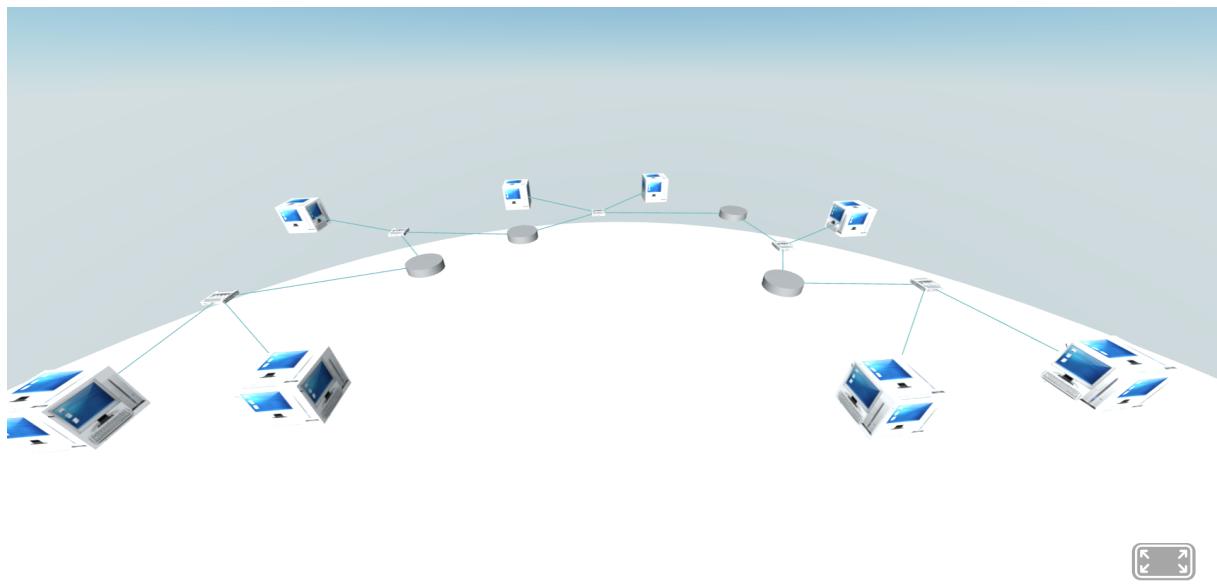


Figura 3.8: Representación en 3D del escenario de una simulación de redes.

### 3.4. Sprint 4: Creación de paquetes y puesta en movimiento

Ya tenemos una red de ordenadores representada en realidad extendida, el siguiente paso a continuar es implementar unos paquetes que realicen unas rutas dentro de la red.

<sup>5</sup>Representación Red en A-Frame: <https://enriqueestebaranz.github.io/EnriqueEstebaranz/Sprint3/SimuladorNetGUI/>

### 3.4.1. Objetivo

En este sprint se marca como objetivo dar vida a la simulación añadiendo unos paquetes animados que simulen el envío de datos de una red. Para ello el backlog de tareas a realizar sigue el siguiente orden:

- **Archivo de paquetes:** Se elabora un archivo con la información necesaria para crear un paquete: tiempo de salida y ruta a realizar.
- **Animación de Paquetes:** Desarrollar una animación recursiva para mover los paquetes a través de los nodos definidos en sus rutas.
- **Demo resultado del sprint:** Como resultado del sprint debemos tener una escena en la que se pueda visualizar el tráfico de red en tiempo real.

### 3.4.2. Tarea

**Archivo de paquetes:** Siguiendo el backlog de tareas comenzamos por realizar un fichero similar al que se obtiene de NetGUI con la información del escenario, pero en este caso se ha realizado por elaboración propia con información de rutas y tiempo de salida del paquete. Destacar que esto se ha realizado para tener información que poder simular y aprovechar que ya se disponía de un programa de python que pasaba la información a formato JSON, el cual con una pequeña modificación nos sirve para cambiar de formato.

```
<packages>
time(2.0); route("pc1","hub4","r1","hub1","dnscom")
time(4.0); route("dnsnet","hub3","r4","hub5","pc2")
time(4.0); route("dnscom","hub1","r2","hub2","dnsroot2")
time(10.0); route("pc2","hub5","r4","hub3","dnsnet")
time(8.0); route("dnsemp1","hub4","r1","hub1","r2","hub2","dnsroot1")
time(9.0); route("dnscom","hub1","r1","hub4","dnsemp1")
time(13.0); route("pc1","hub4","r1","hub1","dnscom")
time(14.0); route("pc2","hub5","r4","hub3","dnsnet")
time(16.0); route("dnscom","hub1","r2","hub2","dnsroot2")
</packages>
```

Ya disponemos de una serie de rutas y tiempos asociados para el envío de paquetes en una simulación de red especificando la secuencia de nodos por los que cada paquete debe pasar, toca representarla.

**Animación de paquetes:** Para representar esta información dentro del simulador creado en el anterior sprint, se ha añadido la función “envioPaquetes” dentro del componente “simulacro”. La función comienza solicitando el archivo JSON denominado “packages.json” a través de la función “fetch”, así obtengo un array de paquetes y con “foreach” se itera sobre cada uno de los paquetes. Ahora se trabaja con cada paquete por separado utilizando “setTimeOut()”, método explicado en el apartado 4.2.2, ejecuto un fragmento de código, una vez transcurrido un tiempo determinado, ideal para que funcione como temporizador y se introduzca el paquete con su animación cuando toque.

Una vez el temporizador termina se genera un paquete que inicia su camino en el lugar del primer sistema que se indica en la ruta, gracias a que en la creación del escenario se hizo referencia de cada nodo con su nombre correspondiente se puede utilizar el método “getElementById” para localizar los dispositivos que aparecen en la ruta y extraer su posición. Se muestra el código donde se añade el paquete en la escena utilizando la posición del primer dispositivo de la ruta:

```
const pqt = document.createElement('a-entity')
pqt.setAttribute('geometry', {primitive: "box", width:0.3,height:0.3, depth:0.3});
pqt.setAttribute('material', {color: 'red' });
const salida = document.getElementById(paquete.route[0])
pqt.object3D.position.copy(salida.object3D.position);
escenario.appendChild(pqt)
```

Para realizar el movimiento se crea una animación recursiva hasta que alcanza el final de la ruta. Si no existen mas nodos a los que moverse, el paquete se elimina. Este proceso permite visualizar el movimiento del paquete a través de los diferentes puntos definidos en su ruta. Dicho movimiento es generado gracias al atributo “animation” estudiado en el segundo sprint. La siguiente línea es clave para el funcionamiento recursivo de la animación porque ejecuta la siguiente animación una vez finaliza la que se estaba realizando.

```
pqt.addEventListener('animationcomplete', () => movimiento(i+1), {once: true});
```

Destacar que también se realiza un manejo de errores para poder informar al usuario si algo no funciona y su motivo, por ejemplo, el programa informa si se intenta agregar un paquete con una ruta inferior a dos ya que esta no se puede ejecutar.

**Demo resultado:** <sup>6</sup> Con estas implementaciones ya se dispone de una simulación dinámica de tráfico de redes de ordenador en un entorno 3D. En la siguiente figura se ve una captura tomada en el momento en el que se realiza envíos de paquetes.

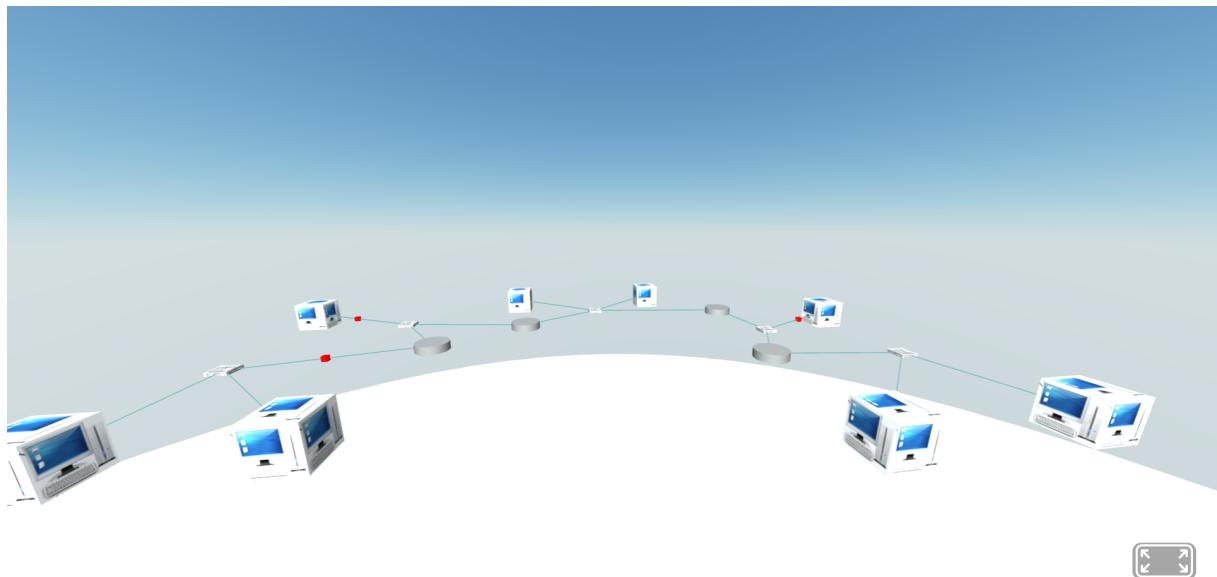


Figura 3.9: Simulación dinámica de tráfico de redes de ordenador en un entorno 3D.

## 3.5. Sprint 5: Implementación de historial y mejora visual

Implementar un historial en una simulación de redes en realidad virtual es crucial, especialmente cuando se aprovecha la funcionalidad 3D para visualizar algo que en 2D es difícil. Por esto es muy importante tener un diseño acertado e intuitivo para el historial, ya que se explota una de las ventajas con las que se diferencia este simulador con el resto de simuladores 2D. A lo largo de este sprint se verá como se desarrolla el historial hasta tener un programa que satisfaga los objetivos.

### 3.5.1. Objetivo

El objetivo es generar un historial del envío de paquetes en el programa obtenido del anterior sprint, para lograrlo se ha comenzado por hacer una primera versión y posteriormente trabajar

<sup>6</sup>Demo EnvioPaquetes: <https://enriqueestebaranz.github.io/EnriqueEstebaranz/Sprint4/EnvioPaquetes/>

en mejoras visuales de está. Como tareas se han hecho las siguientes:

- **Versión inicial:** Desarrollo de una versión en la que se puede visualizar de alguna manera el historial, para ello se han creado distintas funciones:
  - Desplazamiento vertical: implementar una función que mueva el escenario en el eje “Y” para simular el progreso temporal.
  - Implementar un rastro: agregar geometrías en la escena que represente el rastro de los paquetes en movimiento.
- **Mejorar la versión y organizar el código:** El código ya comienza a ser extenso y es importante organizarlo para que este sea legible y más cómodo de depurar. También nos centramos en el aspecto visual del historial.

### 3.5.2. Tarea

**Versión inicial:**<sup>7</sup> Para esta primera versión se ha pensado en el historial como un rastro que van dejando los paquetes según se van moviendo, de esta manera visualizando dicho rastro se podrá conocer el camino realizado por cada paquete. Para el desarrollo de este rastro se ha hecho lo siguiente:

- **Desplazamiento vertical del simulacro:** Se mueve el escenario hacia arriba en el eje Y, para simular una progresión o desplazamiento temporal en la visualización. Como toda la simulación creada hasta ahora está dentro de una entidad llamada escenario, se puede seleccionar esta entidad y desplazarla (desplazando todo lo que contiene), modificando únicamente la posición en el eje Y de la entidad. Para que se visualice como un movimiento suave se actualiza cada 200 milisegundos con una subida de 20 centímetros, de no ser así se vería como el escenario pega saltos.

```
const posicionEscenario = escenario.getAttribute('position')
let initialY = 0; // Comienza en la posición Y = 0
setInterval(() => {
  initialY += 0.2; // Incrementa en 1 metro cada segundo
  escenario.setAttribute('position', `0 ${initialY} 0`);
}, 200); // Actualiza cada 200 milisegundos (0.2 segundo)
```

---

<sup>7</sup>Versión inicial del historial: <https://enriqueestebaranz.github.io/EnriqueEstebaranz/Sprint5/Version-Inicial/>

- **Implementación del rastro:** Lo primero que se ha hecho es implementar una función que únicamente agrega un cubo en “a-scene” del color que se le indique en la posición que reciba, la clave de esta función es mezclar las coordenadas X y Z que poseen el paquete con la altura en la que se encuentra el escenario, de esta forma al inyectar el cubo en la escena permite crear un rastro.

```
function rastroHistorial(posicion, posicionEscenario, color){
    const cubo = document.createElement('a-entity');
    cubo.setAttribute('geometry', {primitive: "box", width:0.4,height:0.1, depth:0.4});
    cubo.setAttribute('material', {color: color });
    cubo.setAttribute('position', `${posicion.x}${posicionEscenario.y}${posicion.z}`);
    document.querySelector('a-scene').appendChild(cubo);
}
```

Y desde la función “envioPaquetes” dentro del componente simulacro agrego un intervalo que me permite llamar a la función “rastroHistorial” he indicarle las coordenadas en las que se encuentra el paquete en ese momento y el color que posee para poder diferenciar cada rastro en función del paquete que lo ha creado.

Tras estas implementaciones ya tenemos la primer versión del simulador con su historial.

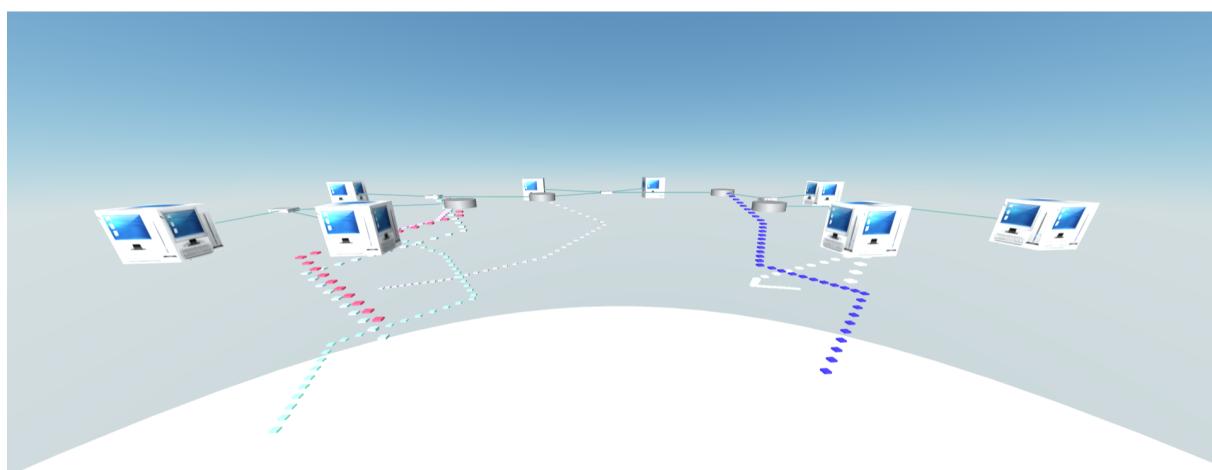


Figura 3.10: Primera versión del simulador con su historial.

Esta primera versión es ”pobre” visualmente y difícil de entender para el usuario pero ha servido de ayuda para formar una idea de lo que se busca. Para mejorar esta versión se ha inspirado

de algunos esquemas que se hacen en las asignaturas de redes para explicar su funcionamiento, en la siguiente figura se puede ver un ejemplo.

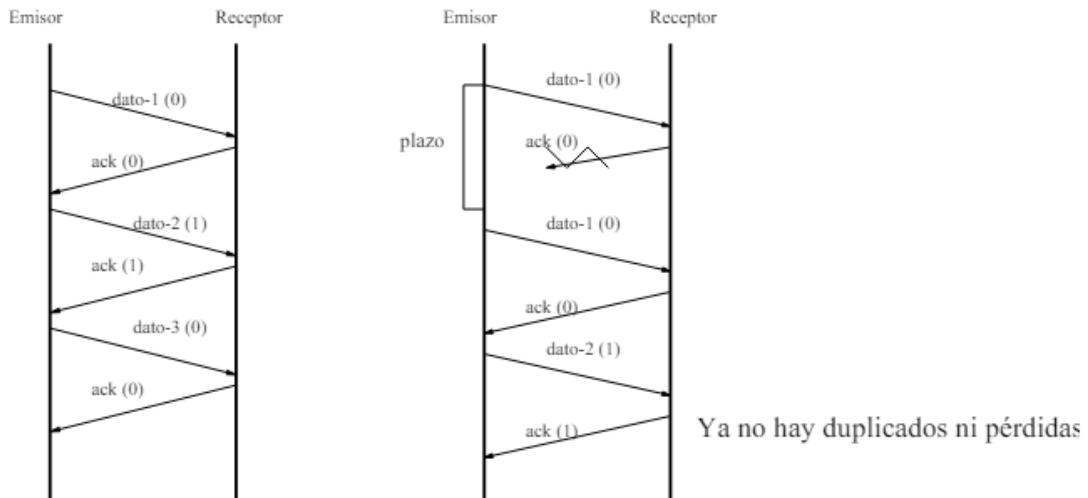


Figura 3.11: Captura extraída del contenido de la asignatura Arquitectura de internet.

De este ejemplo se extraen dos ideas a realizar para la siguiente versión:

- La representación del transcurso del tiempo va de arriba a abajo, a diferencia de la versión inicial que va de abajo a arriba.
- La incorporación de líneas guía para facilitar un seguimiento visual de la localización de los sistemas.

**Versión mejorada:** <sup>8</sup> En esta versión no solo se modifica la representación del transcurso del tiempo e implementa líneas guías, también se incorporan mejoras visuales en el rastreo y se separa las funcionalidades en distintas componentes. A continuación se va a tratar punto por punto los cambios realizados, con respecto la versión inicial.

- **Organización del código:** Separar el código en componentes más pequeños y específico provoca una mejora en la claridad y modularidad, facilitando la comprensión del programa y permitiendo que cada componente funcione de manera autónoma. Por este motivo el componente ‘simulacro’ se ha dividido en 3 componentes:

<sup>8</sup>Versión mejorada del historial: <https://enriqueestebaranz.github.io/EnriqueEstebaranz/Sprint5/version-mejorada/>

- **Simulacro:** Este componente mantiene el nombre original puesto que prácticamente realiza la misma función, construye el mapa de redes de ordenadores, crea los paquetes y el historial. Realmente se encarga de crear las entidades con las componentes donde se realiza dicha función para crear paquetes (componente -‘envio-paquetes’) y crear historial (componente -‘historial’), en cuanto a la construcción de mapa de redes de ordenadores mantiene su funcionalidad dentro de este componente. Dicha funcionalidad no ha sufrido ninguna modificación.
- **Historial:** Es la encargada del desplazamiento vertical para registrar el rastreo a lo largo del tiempo. Se ha modificado con respecto la anterior versión, ya no va creciendo en el sentido positivo del eje Y, ahora crece hacia el negativo, de esta forma la representación en el transcurso de tiempo va de arriba a abajo.
- **Envio-paquetes:** No ha sufrido ninguna modificación en cuanto a código, realiza la misma función que es crear los paquetes, moverlos de un lado a otros y eliminarlos.
- **Componente “trazador”:** Esta es una de las mejoras principales de la versión, se ha creado una componente que se encarga de crear una traza visual de la ruta de los paquetes en el simulador. Tiene una funcionalidad similar a la función “rastroHistorial” de la versión anterior, pero esta incorpora la creación de una línea de guiado en función de cada sistema del mapa de la red.
  - **Pc:** Crea una fina linea recta de cubos con la textura del PC apilados hacia abajo.
  - **Router:** Crea una fina linea recta de cilindros grises apilados hacia abajo.
  - **Hub:** Crea una linea recta compuesta por cubos mas estrechos con la imagen del hub.

Con esto facilito un seguimiento visual de la localización de los sistemas. A parte de esto se ha añadido un grado de personalización al componente, permitiendo elegir a la hora de crearlo el tiempo del intervalo que crea las entidades y si el rastro del paquetes se quiere en forma de caja o esfera. Se comparte la línea de código donde se añade esta componente al paquete:

```
pqt.setAttribute('trazador', {forma: 'esfera', intervalo: 30});
```

Dicha línea se localiza dentro de la componente “envio-paquetes” a la hora de crear el paquete. En caso de no indicar ningún valor se mantendrá los que tiene por defecto (cubo e intervalo de 200 milisegundos)

```
AFRAME.registerComponent("trazador", {
  schema: {
    forma: {type: 'string', default: 'caja'},
    intervalo: {type: 'int', default: 200}
  },
  //...
```

- **Componente “difuminado”:** Esta componente difumina de manera gradual el rastro que dejan los paquetes, así se puede apreciar de forma más visual el rastro mas reciente. Este atributo se asigna a todas las entidades que marcan el rastro del paquete, se puede modificar el tiempo que tarda en difuminarse y el nivel de opacidad (va de 0 a 1, con 0 deja de ser visible y con 1 y totalmente opaco).

Este es el resultado del simulador tras la implementación de las mejoras:

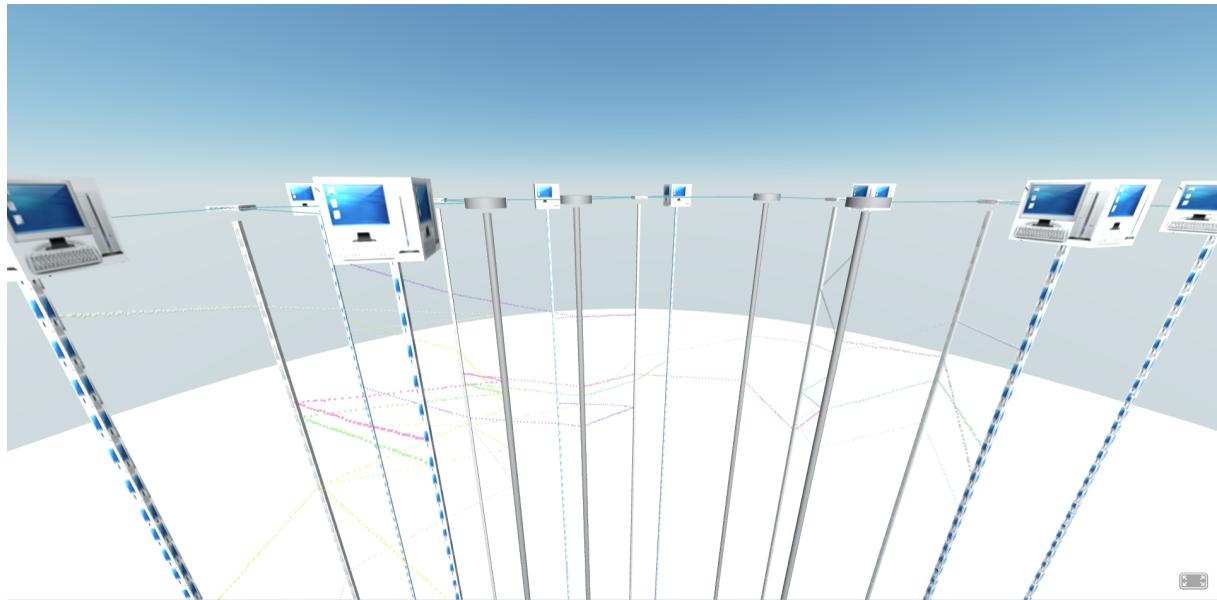


Figura 3.12: Versión mejorada de la primera versión del simulador con su historial.

Se puede apreciar una mejora considerable de la versión inicial, mas entendible y con lo que ya se puede trabajar en añadir funcionalidades para que el usuario pueda interactuar.

## 3.6. Sprint 6: Interactividad con la simulación

En este sprint se va a trabajar en añadirle interactividad al simulador.

### 3.6.1. Objetivo

El objetivo es incluir dos botones, pausa y reanudar, de tal forma que se pueda pausar toda la simulación y reanudarla para continuar en el punto donde se dejó cuando quiera. A simple vista puede parecer sencillo, pero por el calibre del programa, el gran número de intervalos que están en funcionamiento y las animaciones ha supuesto un quebradero de cabeza trabajar en esta implementación. Además se le ha dado una segunda vuelta a la forma en la que se representa el historial y se ha optado por modificarla. Backlog de tareas:

- **Agregar botones de control:** Crear y posicionar los botones de pausa y reproducción.
- **Pausar todo el simulador:** Emitir eventos para pausar todas las animaciones y pausar temporizadores en ejecución.
- **Reanudar la simulación:** Restaurar el estado previo a la pausa.
- **Mejorar la gestión del historial:** Cambiar la forma en la que se representa el historial para que sea más intuitiva.

### 3.6.2. Tarea

**Implementación de controles:** El primer paso que se ha realizado es la implementación de dos botones de control dentro de la cámara del usuario. Esta función se agrega dentro de la componente simulacro para que cree todo una vez se inicie. Los botones se agregan a la cámara del usuario para que de esta forma estén disponibles en todo momento. El aspecto es el siguiente:

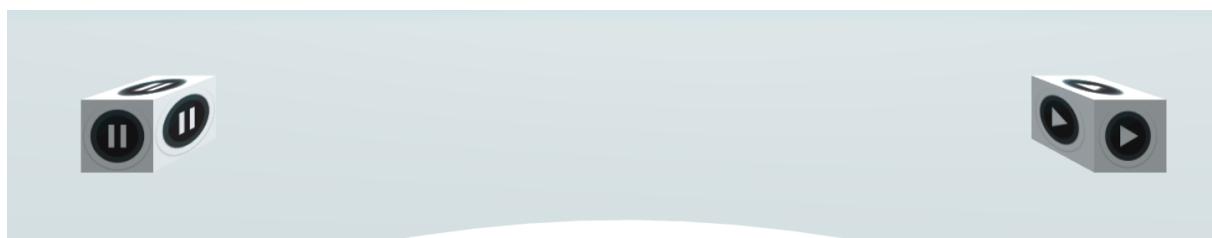


Figura 3.13: Botón pausa y reproducir fijados en la cámara de A-Frame

**Funcionalidad pausa:** Como se puede apreciar en la creación de los botones, cada uno tiene agregado un atributo “pausa” o “reproducir”, esto es un indicativo de que se ha creado una componente “pausa”. Debido a su complejidad se va a indicar todo lo que se debe pausar y la forma en la que se pausa, para cada cosa:

- **Envío paquete:** En este punto se debe pausar dos cosas:

- **Animación de los paquetes:** Debido a las propiedades de A-Frame pausar las animaciones es sencillo. Basta con emitir el evento “animation-pause” a todas las animaciones provocando que ejecuten su lógica de pausa interna.

```
document.querySelectorAll("[animation]").forEach((element) => {
  element.emit('animation-pause');
});
```

Pero para que esto funcione es necesario que a la hora de crear la animación se indique el evento pausa dentro de las propiedades de “animation”:

```
pauseEvents: 'animation-pause'
```

- **Intervalos de tiempo para lanzar paquetes:** Si se recuerda, los paquetes vienen con la información de la ruta que se va a hacer y el segundo en el que se lanza, es importante llevar un registro de los intervalos, de tal forma que si se pausa, poder retomar el intervalo con el tiempo que se pauso. Esto ha supuesto un reto ya que el intervalo no puede ser pausado una vez ha comenzado. La solución que se ha tenido es detener todas las ejecuciones futuras del intervalo eliminándolo y luego reiniciarlo con el tiempo que le quedaba antes de eliminar. Para ello, se itera sobre un array “almacenTimeouts” que contiene información sobre cada “setTimeout” en ejecución, incluyendo su identificador, el tiempo de inicio del timeout, el tiempo total que debería durar, y el tiempo restante (que se calculará mas adelante).

```
const delay = paquete.time * 1000;
const timeout = setTimeout(() => {
  if (!this.pausa) {
    numeroPaquetes +=1
    this.animacion(paquete, numeroPaquetes);
  }
  // Elimina el timeout del almacen después de ejecutarse
  this.almacenTimeouts = this.almacenTimeouts.filter(t =>
  t.id !== timeout);
}, delay);
this.almacenTimeouts.push({ id: timeout, package: paquete,
delay: delay, startTime: Date.now(), remainingTime: delay });
```

Con el array “almacenTimeouts” ya se puede trabajar en la pausa. Primero se cancela cada timeout que se encuentra en el array, y para cada timeout cancelado se le calcula el tiempo restante hasta que debería haberse disparado la orden. Este tiempo restante se actualiza en el array “almacenTimeouts” para disponer de ello mas tarde cuando toque reanudarlo.

```
pausar: function() {
    this.pausa = true;
    this.almacenTimeouts = this.almacenTimeouts.map(t => {
        clearTimeout(t.id); // Cancela el timeout actual
        const currentTime = Date.now();
        t.remainingTime = t.remainingTime - (currentTime
            - t.startTime);
        return t
    });
}
```

Se debe incluir la siguiente linea en el componente pausa para ejecutar la función pausar de la componente “envio-paquetes”.

```
const envioEntity = document.querySelector('[envio-paquetes]');
envioEntity.components['envio-paquetes'].pausar();
```

- **Trazador:** Se debe pausar el componente encargado de crear las entidades que forman parte del historial, como las lineas de guía y el rastro de los paquetes, puesto que de no hacerlo se saturaría el escenario por la creación de entidades en un mismo punto. Para ello se ha añadido un control de pausa en el trazador.

```
// this.pausado = false; por defecto
if(this.pausado) return;
```

De tal forma, si pausado es true retornara sin hacer nada dentro del trazador, de lo contrario creará las entidades. Para modificar el estado de ”this.pausado” ha bastado con crear dos pequeñas funciones (pausa y reanudar) que cambian el estado para poder llamarlas desde la componente pausa.

- **Difuminado:** Similar a las animaciones de “envio-paquetes”, pero con el difuminado, se emite “fadeout-pause” a todas “animation fadeout”.
- **Historial:** Por ultimo se pausa el historial que es donde se representa el avance del tiempo a través del incremento del eje Y, y este no puede incrementar durante la pausa de la simulación. Para evitar que siga se detiene la ejecución del intervalo que lo mueve.

```

pausado: function() {
    this.pausa = true;
    if (this.interval) {
        clearInterval(this.interval);
        this.interval = null;
    }
}

```

Y al igual que se ha hecho con el resto , se llama a esta función de la componente historial dentro de la componente pausa.

**Funcionalidad reanudar:** La lógica de reanudar es más sencilla de implementar, ya que depende de revertir el estado de pausa y reanudar los procesos suspendidos, es por ello que esta parte es mas corta, se sigue la metodología anterior.

- **Envío paquete:** Para reanudar el envío de paquetes se comienza por emitir el elemento “animation-resume” a todas las animaciones y recorrer el array de paquetes pendientes por lanzar con su correspondiente tiempo pendiente “almacenTimeouts” e iniciar el timeout.
- **Trazador:** Se modifica el estado de ‘pausado’ a “false”.
- **Difuminado:** Se emite “fadeout-resume” a todas “animation\_fadeout”.
- **Historial:** Para continuar con el incremento del historial se crea un intervalo que siga la razón de movimiento en el eje Y respecto al tiempo que se tenía antes.

Ya tenemos las dos funcionalidades fundamentales para poder considerar lo que se tiene como un simulador interactivo de redes de ordenadores en realidad extendida.

**Mejora del historial:** Por ultimo, en este mismo sprint se ha aprovechado para realizar un nuevo cambio en la gestión del historial. Se ha hecho una mera modificación en las configuraciones del componente trazador que se encarga de “dibujar” el historial. Antes se “dibujaba” con coordenadas globales en la escena y la modificación se ha realizado para que todas estas entidades que se crean sea dentro de la entidad ‘historial’.

```

const localPosition = this.historial.object3D;
worldToLocal(globalPosition.clone());

```

Esta línea convierte la posición global de un objeto a una posición local relativa a la entidad ‘historial’, permitiendo trabajar con coordenadas relativas en el contexto del historial en A-Frame. En la siguiente figura se comparte una captura de la simulación<sup>9</sup>:

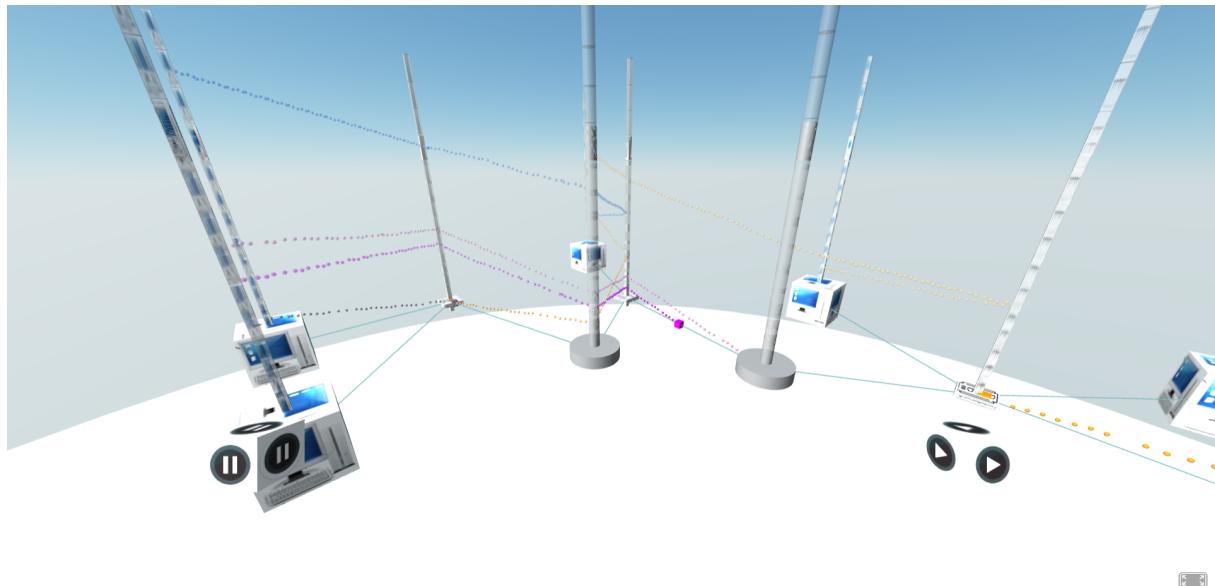


Figura 3.14: Simulador interactivo de redes de ordenadores en realidad extendida.

Esta modificación en el historial nos permite tener fijado el mapa de redes con sus paquetes en un plano. Y es mas intuitivo mantener un historial de tal manera que el paquete esta seguido del rastro mas reciente que deja.

## 3.7. Sprint 7: Creación de escenario final

Último sprint, en el que se pone a punto el simulador.

### 3.7.1. Objetivo

El objetivo de este sprint es mejorar el simulador resultante del sprint anterior añadiéndole funcionalidades nuevas y cuadros informativos. Además, se generarán unas demos de prueba que serán las que se muestren en la presentación del TFG. En la siguiente lista de tareas se enumeran las mejoras y objetivos a realizar en este sprint:

---

<sup>9</sup>Versión mejorada del historial: <https://enriqueestebaranz.github.io/EnriqueEstebaranz/Sprint6/SimuladorControlado/>

- **Cuadros informativos:** Agregar información del simulador como identificadores de los sistemas y cuadros informativos con los paquetes.
- **Interacción historial:** Aprovechando el rastro que dejan los paquetes para agregar funcionalidades como la visualización del historial de un paquete en concreto.
- **Agregar historial de recepción de paquetes:** de esta forma llevaremos un seguimiento de los paquetes que ha recibido cada sistema.
- **Mejora de la interactividad con la simulación:** Modificación de los botones “pausar” y “reanudar” y agregar la funcionalidad reiniciar.
- **Sonido y control:** Agregar sonidos en la simulación y un control para poder ensordecerlos o no.
- **Demos:** Diseñar distintos escenarios en los que se pueda visualizar el funcionamiento del simulador.

### 3.7.2. Tarea

**Cuadros informativos:** Creo una componente llamada “nombre-en-plano” con la que a través de un texto y un plano de color distinto de fondo represento el nombre de cada sistema de la topología. Con esta componente cubrimos una parte informativa de la simulación, pero aun quedan los paquetes, para ello creamos una función “crearCuadroNombrePaquete” donde agrego el nombre del paquete y su origen y destino.

**Interacción historial:** Se ha agregado una componente llamada “mostrar-historial” en la que aprovecho el identificador (dependiendo del paquete que genera el rastro) que tiene las entidades creadas con componente trazador y las vuelvo invisibles o visibles en función de los paquetes que seleccione. Es decir cuando clique en un paquete se mostrara únicamente el historial de los paquetes clicados. Cuando se interactúe con el historial la simulación estará pausada.

**Historial de recepción de paquetes:** La interacción con el historial hizo que nos planteásemos con el siguiente problema: si para visualizar el historial necesito clicar un paquete, y estos

se eliminan una vez llegan a su destino, como puedo interactuar con el. La solución es la componente “paquetes-enviados” que se encarga de almacenar los paquetes en orden de llegada al lado del sistema destino de tal forma que se tienen a manos todos los paquetes de la simulación. Por otro lado, para evitar que los paquetes sigan dejando un rastro se les quita el atributo trazador.

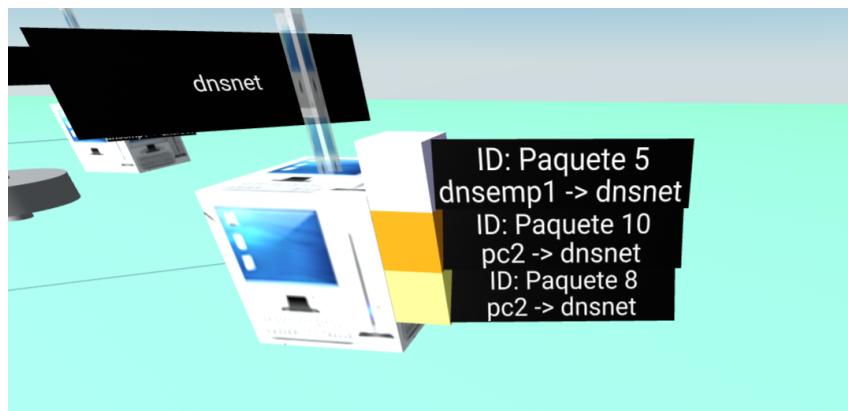


Figura 3.15: Se aprecia el historial de paquetes recibidos en el sistema “dnsnet”.

**Mejoras:** Se optimiza el control de los botones pausar y reanudar manteniendo un único botón que cambie el aspecto en función del estado, similar al botón que poseen todas las herramientas de reproducción de vídeo. Y agrego el botón reiniciar cuya funcionalidad es reiniciar la simulación. Básicamente recarga la página en la que se encuentra.

**Sonido y control:** Para esta parte utilice las siguientes líneas de código para hacer sonar una notificación que indique que el paquete ha llegado a su destino :

```
const sonar = document.getElementById('logrado');
sonar.components.sound.playSound();
```

Donde en función del estado que tenga la componente “sonido” se escuchará o no. Esta componente se maneja a través de otro botón que cambia el aspecto en función del estado. En la siguiente figura se comparte una imagen de como quedan todos los controles dentro de la escena.

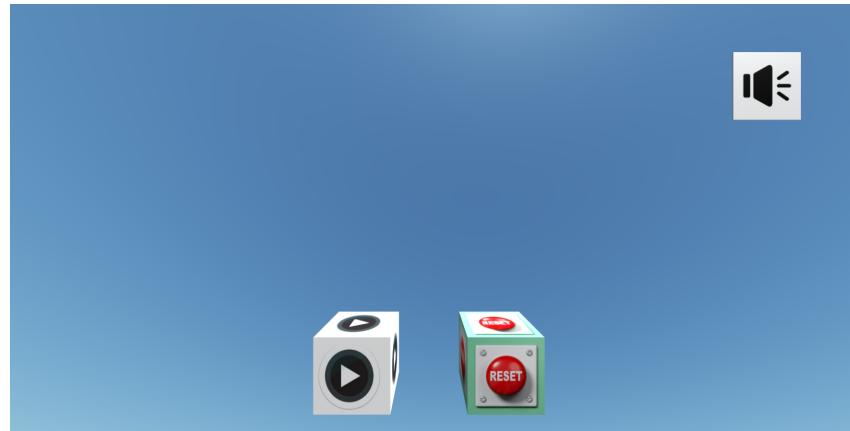


Figura 3.16: Controles de la simulación, en este caso el estado es pausado y sonando

**Demos:** Como ya tenemos todo, únicamente se debe crear unos paquetes que imiten casos reales que sucedan en una simulación de redes de ordenadores. Se ha optado por simular en una demo la ejecución de pings<sup>10</sup> y en otra la de traceroute<sup>11</sup>.

---

<sup>10</sup>Demo ping:: <https://enriqueestebaranz.github.io/EnriqueEstebaranz/Sprint7/Ping/>

<sup>11</sup>Demo ping:: <https://enriqueestebaranz.github.io/EnriqueEstebaranz/Sprint7/Traceroute/>

# **Capítulo 4**

## **Descripción del resultado**

En este capítulo de la memoria se detalla el resultado obtenido con el desarrollo del simulador de redes de ordenadores en realidad extendida. Proporcionando un manual adaptado al objetivo del usuario y describiendo la arquitectura completa del sistema y desarrollo del programa para aquellos usuarios interesados en extender el código subyacente del simulador.

### **4.1. Descripción para usuario**

En esta sección se proporciona una guía detallada sobre cómo interactuar y explotar al máximo el simulador adaptado a las necesidades del usuario. Se divide en dos subsecciones para tratar distintos enfoques de uso. En la primera se le cuenta al usuario que puede hacer dentro de la simulación y en la segunda como se componen esos escenarios.

#### **4.1.1. Uso General del Simulador**

En este apartado se da una explicación de la interfaz del simulador, controles dentro del entorno e interacciones con los elementos visuales. El usuario accede a la simulación, la cual se encuentra completamente configurada y operativa dentro del sitio web donde se hospeda el simulador, GitHub Pages<sup>1</sup> en este caso.

Para este punto se diferencia entre dos tipos de usuario, dependiendo del dispositivo con el que desee interactuar en la web. Por un lado están los usuarios que abren la simulación desde

---

<sup>1</sup>Enlace GitHub Pages: <https://pages.github.com/>

dispositivos como ordenadores, teléfonos y tablets y por otro el de los usuarios que utilizan gafas de realidad virtual para la visualización de la simulación.

- **Usuario de Plataforma Web (no-VR):**

El primer paso que se debe realizar es acceder a la URL de cualquiera de las dos demos para visualizar, TopologíaSimple<sup>2</sup> y TopologíaCompleja<sup>3</sup>, elegimos la que se deseé y abrimos su web en cualquier navegador.

Una vez se accede al escenario lo primero que vemos será esto:

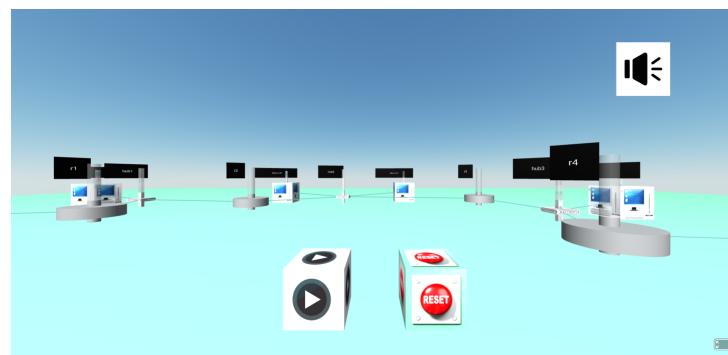


Figura 4.1: Vista según se inicia la simulación.

Se aprecia la topología de una red que va generando un rastro hacia arriba según pasa el tiempo, de tal manera que el primer segundo de la simulación queda arriba del todo y el instante más reciente abajo. Si dejamos que la simulación avance más tiempo el resultado será este:

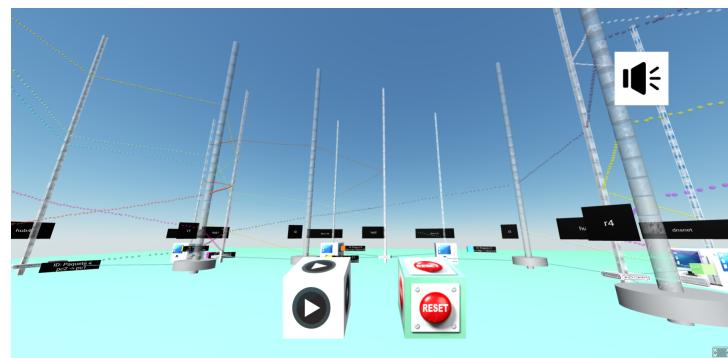


Figura 4.2: Transcurso de tiempo en la simulación.

---

<sup>2</sup>Demo1:<https://enriqueestebaranz.github.io/EnriqueEstebaranz/Sprint7/Ping/>

<sup>3</sup>Demo2:<https://enriqueestebaranz.github.io/EnriqueEstebaranz/Sprint7/>

Para **movernos** dentro de la simulación podemos usar tanto las teclas de las flechas como ‘‘W’’, ‘A’’, ‘S’y ‘D’, tienen la misma funcionalidad, moverse adelante ( $W\uparrow$ ), hacia atrás ( $S\downarrow$ ), izquierda ( $A\leftarrow$ ) y derecha ( $D\rightarrow$ ) respecto a la dirección y sentido de la cámara. El movimiento de la cámara se controla con el ratón, manteniendo presionado el clic izquierdo en la escena y moviendo el ratón podemos mover la cámara, modificando de esta forma su dirección y sentido. En cuanto a la **interacción** basta con apuntar con la cámara directamente a los elementos con los que se desea interactuar (botones, dispositivos y paquetes) y hacer clic en el (no importa si es izquierdo o derecho).

Gracias a la configuración con la que se tiene la cámara se puede mover libremente por todo el entorno, permitiendo visualizar el simulador desde distintos ángulos, como se muestra en la siguiente figura.

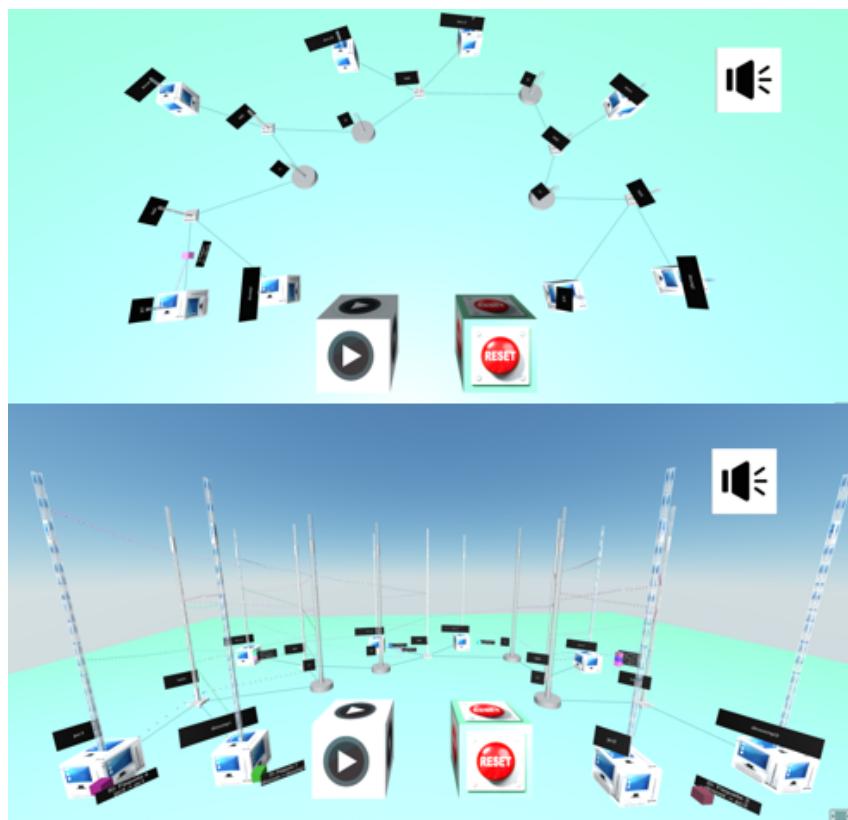


Figura 4.3: En la figura superior nos hemos situado encima de la simulación para apreciar mejor la topología de la red, mientras que en la figura inferior nos hemos alejado para tener una visión mas global de la simulación.

En la escena se dispone de tres botones que están fijos en la cámara, para interactuar

con ellos es importante que no estén los botones cerca de otras entidades, ya que puede interferir el área de la otra entidad evitando que se clique de forma correcta en los botones. En cuanto a los botones son tres:

- **Botón de Reinicio:** El botón está ubicado a la derecha y reinicia el escenario por completo, tiene el siguiente aspecto.



Figura 4.4: Aspecto del botón de reinicio.

- **Botón Reanudar/Pausar:** Este botón se encuentra ubicado a la izquierda, sirve para pausar o reanudar toda la simulación. En función del estado en el que se encuentre la simulación, este botón alterna entre pausar y reanudar. De tal forma que aparecerá un botón reanudar, si la animación esta pausada o pausar si la simulación esta en movimiento. En la siguiente figura 4.5 se muestra los dos aspecto que puede tener este botón en función del estado en el que se encuentre la simulación

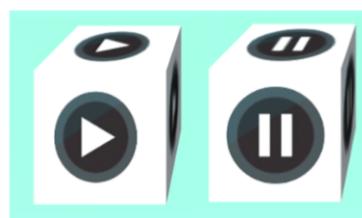


Figura 4.5: Botón aspecto reanudar y botón aspecto pausa.

- **Botón sonido:** Se ubica en la esquina superior derecha y se encarga de silenciar o no el sonido de la simulación. Si el estado de este tendrá una apariencia u otra. En el caso de que la simulación tenga sonido, el aspecto del botón será el de un altavoz “sonando”, mientras que si el estado es silencioso.

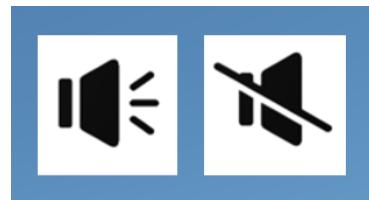


Figura 4.6: Botón aspecto sonando y silencio.

Cuando los paquetes llegan a su fin se amontonan por orden de llegada al lado del nodo destino. Teniendo así un **historial de paquetes** que queda como se ve en la siguiente imagen:

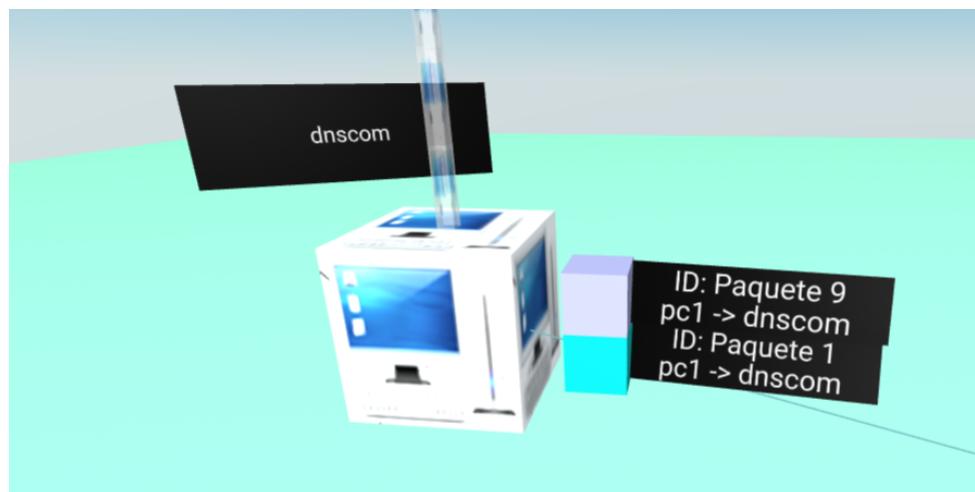


Figura 4.7: Historial de paquetes en el nodo “dnscom”.

Aprovechando esta imagen se puede ver como en toda la simulación tenemos cuadros informativos, unos se encuentran encima de cada dispositivo para informar del identificador que este tiene y otros junto a los paquetes proporcionando la información de nombre del paquete, nodo origen y nodo destino.

Además de los botones mencionados, en el escenario se puede **interactuar con los dispositivos**, al hacer clic en PCs, routers, hubs y switches, se mostrará un cuadro de texto con las conexiones del dispositivo y este desaparece pasado cinco segundos.

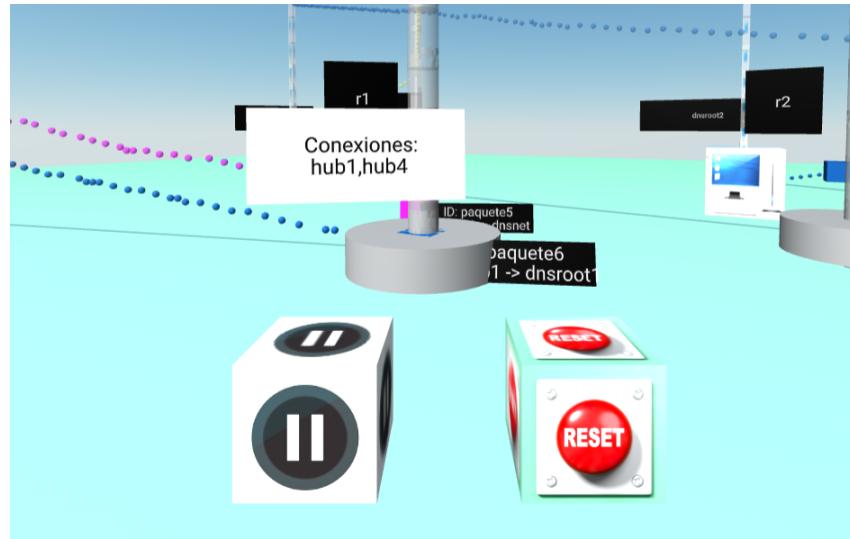


Figura 4.8: Ejemplo de interacción al pulsar el router r1.

En cuanto a si queremos **visualizar el historial** de un paquete en concreto es recomendable pausar previamente la simulación para que sea mas fácil clicar el paquete o ir a un paquete que ya se ha enviado, los cuales aparecen ubicados de manera fija al lado del ordenador donde ha finalizado su ruta, como se ve en la figura 4.7. Una vez seleccionemos el paquete se pausara el simulacro si no lo estaba antes y se mostrara en el escenario únicamente el historial del paquete pulsado, de tal forma que se pueda estudiar su ruta realizada mas fácilmente.

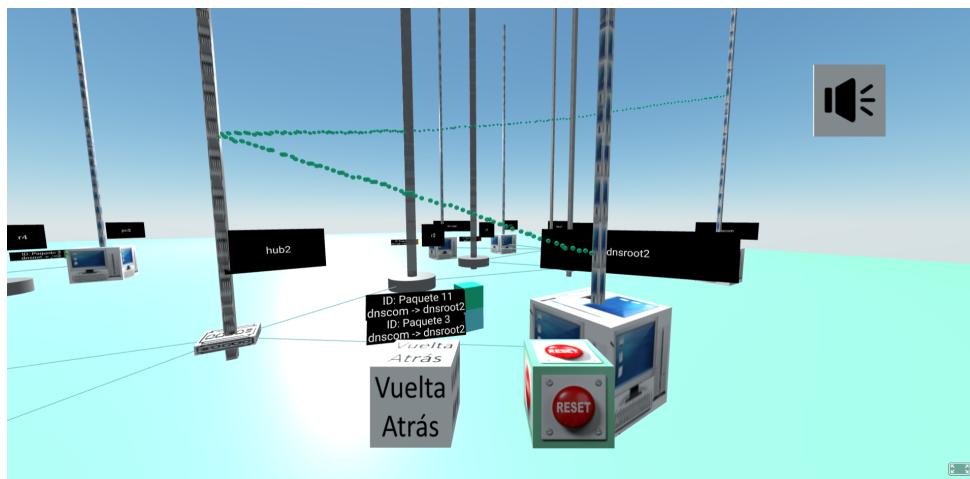


Figura 4.9: Se ha clicado el paquete 11 y únicamente se ve su historial en la escena.

Podemos ver como el botón de pausar/reanudar ha desaparecido y en su lugar tenemos

un botón de vuelta atrás, eso es debido a que estamos en el modo “historial”, a este modo se entra pulsando en cualquier paquete. Dentro de dicho modo podemos pulsar otros paquetes y se visualizará su rastro junto con el del resto de paquetes que se hayan clicado. Por otro lado si se clica en un paquete anteriormente seleccionado se le realiza el proceso inverso, de tal forma que si antes su rastro era visible porque se había pulsado, ahora el historial de dicho paquete no es visible debido a que hemos clicado una segunda vez. Al igual que para los usuarios sin gafas es importante En la siguiente imagen 4.10 se muestra una captura donde se ha pulsado en tres paquetes.

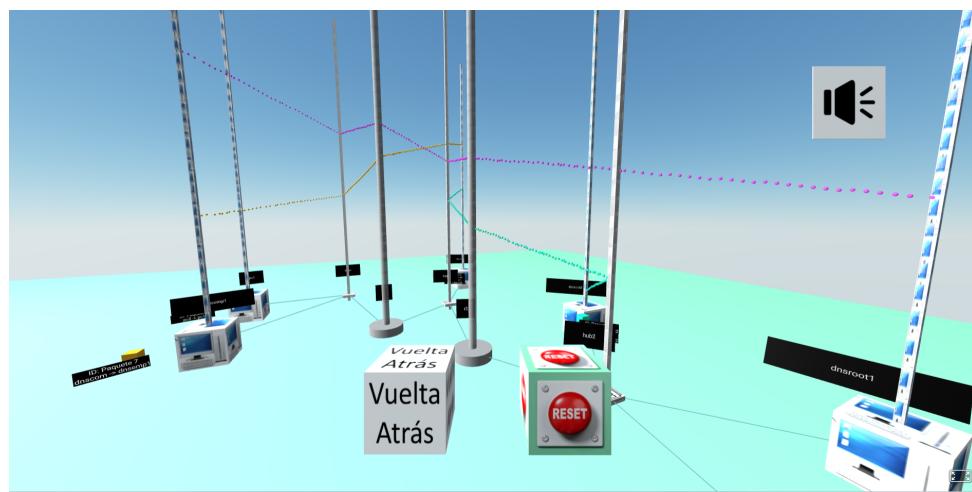


Figura 4.10: Modo “historial” con 3 paquetes seleccionados.

Cuando se pulsa el botón de vuelta atrás se muestra el resto del historial, se elimina el botón vuelta atrás y trae de vuelta el botón con aspecto de reanudar ya que el estado del simulador al volver atrás es pausado.

Una apreciación respecto a la simulación es que el rastro comparte el mismo color que los paquetes que lo dejan. Y no se repite el color de los paquetes que están en envío de tal forma que se evita la mezcla del historial de distintos paquetes.

- **Usuario de Realidad Virtual (VR):** Para este tipo de usuario el acceso a la escena es el mismo, se introduce la URL de la demo que se desee ver en el navegador de las gafas de realidad Virtual. Una vez se abra la escena con el simulador debemos pulsar en el botón que aparece en la esquina inferior derecha con siglas VR (virtual reality).

Para pulsar tenemos que presionar el gatillo de atrás del mando una vez estemos señalando con el el botón. Tras pulsar se carga la escena en realidad virtual.

Dentro de la escena la forma que tenemos de movernos es utilizando el joystick del mando izquierdo, este tiene un funcionamiento similar al de las flechas ya que nos mueve en función del lado al que inclinemos el joystick respecto a la dirección y sentido de la cámara. La forma en la que se mueve la cámara es con nuestra cabeza ya que hacia donde miremos es hacia donde la cámara apunte. Por ejemplo, si queremos elevarnos en la escena se puede hacer mirando hacia arriba y mover el joystick hacia delante, parando cuando estemos a la altura deseada, o mirando hacia abajo y mover el joystick hacia atrás. Son dos formas distintas de hacer lo mismo.

Para interactuar con la simulación tenemos en la mano derecha el mando que aparece con un láser azul. Con este láser apuntamos a cualquier elemento con el que se quiera y pueda interactuar (botones, paquetes y nodos) y presionando el gatillo trasero lo accionamos.

Respecto al funcionamiento del simulador, es exactamente el mismo que el explicado en el punto anterior solo que en un entorno de realidad virtual.

#### 4.1.2. Composición de Escenarios

En este apartado se guía a un usuario avanzado para que el mismo cree su propia escena sin necesidad de tocar código. La guía se ha dividido en dos partes, en la primera se desarrollara la forma en la que se dibuja la topología de la red que deseamos implementar y en la segunda se indica como se configuran los paquetes para su envío en la simulación.

- **Diseño de la topología:** Primeramente realizamos la topología que se deseé simular a través de la interfaz gráfica NetGUI<sup>4</sup>. Esta herramienta es para Linux y una vez esta instalada se arranca con la orden “netgui.sh”. Con la herramienta ya abierta podemos seleccionar entre PC, router, switch y hub que son los dispositivos con los que se puede trabajar en los escenarios de NetGUI. Para dibujarlo basta con pulsar sobre el botón del dispositivo a utilizar y pinchar en la zona donde queramos que se sitúe dentro de la zona de dibujo. Una vez tengamos todos los dispositivos dibujados se conectan utilizando el botón que representa el cable. Una vez seleccionado este botón, pulsaremos una vez sobre

---

<sup>4</sup>Guia NetGUI: <https://mobiquo.gsy.c.urjc.es/dns-and-bgp/intro-netgui.pdf>

el primer dispositivo que queremos conectar y luego pulsamos en el segundo dispositivo. Por ultimo, para borrar cualquier elemento dibujado se selecciona el botón que muestra las tijeras y después presionamos sobre lo que se quiera borrar. En la siguiente figura 4.11 se muestra la interfaz gráfica NetGUI con indicaciones como soporte a lo explicado:



Figura 4.11: Herramienta NetGUI.

Una vez se termina de dibujar el escenario de la red lo guardamos, clicando File->Save, donde elegiremos un nombre de la carpeta que no exista (yo la he llamado GuíaNetGUI). En esa carpeta se almacenará un fichero “netgui.nkp” que contiene toda la información del dibujo del escenario.



Figura 4.12: Fichero “netgui.nkp”.

Con este fichero ya tenemos toda la información del escenario y podemos cerrar la herramienta NetGUI.

A continuación hay que descargarse la carpeta “Configuración” de mi repositorio de

GitHub<sup>5</sup> donde están todos los archivos necesarios para correr la simulación. Una vez tenemos la carpeta, el siguiente paso para continuar con la configuración es cambiar formato del fichero nkp a formato JSON, ejecutando un programa de python de nombre “Ajson.py” localizado en la carpeta descargada. Para ejecutarlo tenemos que tener en la misma carpeta los archivos “netgui.nkp” y “Ajson.py” y abrir una terminal en esa carpeta donde ejecutar el programa de python:

```
PS C:\Users\enriq\Desktop\TFG\Codigo\EnriqueEstebaranz\Configuracion> python3 Ajson.py
El archivo JSON se ha creado correctamente: netgui.json
```

Figura 4.13: Generamos el archivo netgui.json

Con la obtención del archivo “netgui.json” ya tenemos configurado el diseño de la topología.

- **Configuración de paquetes:** La simulación gestiona unos paquetes sintéticos , que los usuarios pueden configurar de acuerdo a sus necesidades específicas. Estos paquetes son elementos clave para simular distintas situaciones de tráfico de red. Para configurar estos paquetes basta con abrir el archivo “packages.nkp” que se encuentra dentro de la carpeta configuración que hemos descargado previamente. Una vez lo abramos vamos a ver lo siguiente:

```
Configuracion > ≡ packages.nkp
1   <packages>
2   time(); route("", "", "", "", ""); id(""); duracion("")
3   <\packages>
```

Figura 4.14: Archivo “packages.nkp” a configurar.

Esta es la plantilla que debemos de llenar, en función del numero de paquetes que se quieran enviar basta con añadir mas líneas con la configuración deseada. A continuación hacemos un desglose de los parámetros a configurar:

---

<sup>5</sup>Configuración GitHub: <https://github.com/EnriqueEstebaranz/EnriqueEstebaranz/tree/main/>

- “**time()**”: Este parámetro especifica el tiempo de inicio del envío de paquete dentro de la simulación. Se mide en segundos, por ejemplo, si deseamos que se lance un paquete en el segundo cuatro deberemos de rellanar el campo con un 4.0, en caso de que queramos enviarlo siete segundos y medio mas tarde pondremos 7.5.
- “**route(“”,“”,“”)**”: Define la ruta que debe seguir el paquete dentro de la red simulada. Los valores dentro de los paréntesis representan los identificadores de los nodos de red por los que el paquete transitará desde el origen hasta el destino. Es importante fijarnos bien en los identificadores de los nodos que tenemos en el archivo “netgui.json” generado para que funcione. Esta secuencia es crucial para simular el paso del paquete a través de diferentes equipos de la red. Por ejemplo si deseo que salga un paquete del pc1 y que pase a través del hub4, r1, hub1 hasta llegar al destino dnscom, tendremos que indicarlo de la siguiente forma “route(“pc1”,“hub4”,“r1”,“hub1”,“dnscom”);”.
- “**id()**”: Un identificador único para el paquete donde le podremos dar un nombre. En caso de dejarlo vacío el programa generará un nombre por defecto, paquete “X”, siendo “X” el numero en orden de los paquetes que se ha enviado, si es el quinto paquete que se envía será paquete5. Por ejemplo si queremos que el id del paquete sea ACK lo indicaremos de la siguiente manera “id(“ACK”);”.
- “**duracion(“”)**”: En este parámetro se define en milisegundos la duración que tarda en desplazarse el paquete entre dispositivos, es decir si pongo 2000 tardara dos segundos en desplazarse de un nodo a otro por lo que si en la ruta el paquete tiene que pasar del nodo1->nodo1->nodo3 tardará cuatro segundos en total , dos segundos de moverse del nodo1 al nodo2 y otros 2 segundos de desplazarse del nodo 2 al nodo3.Por ejemplo si deseamos que el tiempo sea de 4 segundos lo indicaremos de la siguiente forma “duracion(“4000”)”.

En la siguiente figura 4.15 se muestra como quedaría con dos paquete configurado, en caso de querer configurar mas basta con añadir mas líneas siguiendo la misma estructura.

```
Configuracion > packages.nkp
1   <packages>
2     time(1.0); route("dnscom","hub1","r1","hub4","pc1"); id("PING"); duracion("2000")
3     time(9.0); route("pc1","hub4","r1","hub1","dnscom"); id("ACK"); duracion("2000")
4   <\packages>
```

Figura 4.15: Ejemplo de dos paquetes configurados.

Al igual que hicimos con el archivo “negui.nkp” falta por pasarlo a formato JSON para que lo interprete el programa, para ello en este caso tendremos que ejecutar el programa “Ajson2.py” abriendo la terminal de la carpeta donde se encuentran los dos archivos y ejecutándolo de la siguiente forma:

```
PS C:\Users\enriq\Desktop\TFG\Codigo\EnriqueEstebaranz\Configuracion> python3 Ajson2.py
El archivo JSON se ha creado correctamente: packages.json
```

Figura 4.16: Generamos el archivo package.json.

Ya tenemos la topología y los paquetes configurado, queda el último paso que es lanzar el servidor. Podemos subirlo a GitHub Pages para visualizarlo en línea o lanzarlo en local.

Para lanzarlo en local se puede hacer de muchas formas, en esta guía utilizaremos la aplicación Visual Estudio. En esta aplicación podemos ir a la pestaña de extensiones y buscar Live Server donde descargaremos esta extensión:

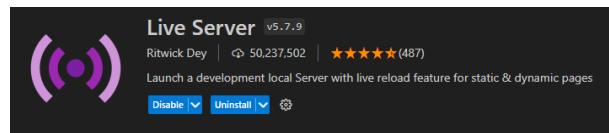


Figura 4.17: Extensión Live Server de Visual Estudio

Para usarlo una vez esta instalada debemos dirigirnos al archivo “index.html” dentro de la carpeta configuración y abrirlo con Visual Estudio. Una vez esta abierto tenemos que pulsar en el botón que aparece en la esquina inferior derecha con el nombre de “Go Live”. Nos abrirá una pestaña del navegador y correrá el escenario en local.

## 4.2. Arquitectura general

En esta sección vamos a desglosar y explicar la estructura detallada del programa y los componentes que lo conforman. El programa se basa principalmente en dos archivos: “app.js” y “index.html”.

En el archivo “index.html” únicamente cargamos los recursos necesarios para la escena, como imágenes y sonido, agregamos la iluminación y el suelo con la siguiente línea de código.

```
<a-plane rotation="-90 0 0" width="100" height="100" color="#7BC8A4"
material="src: #groundTexture; repeat: 10 10;" position="0 -1 0"></a-plane>
```

Para que se genere la simulación basta con tener en el “index.html” la entidad con la componente simulacro , tal que así:

```
<a-entity simulacro></a-entity>
```

Y sobre la cámara y los controles de movimientos agregamos la siguientes entidad con los atributos necesarios para configurarlos:

```
<a-entity id="rig" movement-controls position="0 1.6 0">
  <a-entity camera look-controls movement-controls="fly: true"
    wasd-controls="fly: true" id="camara"></a-entity>
  <a-entity cursor="rayOrigin:mouse"></a-entity>
  <a-entity laser-controls="hand:right"></a-entity>
</a-entity>
```

En “app.js” se definen los componentes y la lógica principal del simulador y como indica su extensión es un archivo que contiene código en JavaScript (lenguaje utilizado por A-Frame). Los elementos principales del programa son los componentes, en A-Frame son la base de la funcionalidad de las entidades, definen su comportamiento, apariencia y funcionalidad.

Los componentes que forman el simulador son varios, cada uno tiene un propósito específico y contribuye en el funcionamiento general de la simulación. Basándonos en la funcionalidad y la correlación entre los componentes se va a explicar todos los componentes siguiendo un orden de mayor a menor importancia:

### 4.2.1. Componente Simulacro

Este componente se encarga de inicializar todo el entorno de la simulación, creando un escenario, agregando los controles de interfaz y algunos componentes específicos para manejar la lógica de la simulación. A continuación se explicará las funciones que se realizan dentro de este componente.

La función “init” es una de las múltiples funciones opcionales que pueden definirse dentro de un componente A-Frame, se ejecuta el comportamiento del componente una vez se añade a una entidad. En este caso solo ejecuta la función “iniciarSimulacion” para empezar la simulación.

La función “iniciarSimulacion” crea una nueva entidad y la agrega al escenario de A-Frame. A esta entidad la llamaremos escenario y contendrá todas las entidades que tengan que ver con el escenario. Luego llamo a las funciones “agregarControles”, “construirMapa”, “crearpaquete” y “crearhistorial”. En la siguiente figura 4.18 se comparte la función “iniciarSimulacion”:

```
iniciarSimulacion: function () {
    // creo la entidad donde se localizará todo el escenario y la agrego a la escena
    const scene = document.querySelector('a-scene');
    const escenario = document.createElement("a-entity")
    escenario.setAttribute("id", "escenario");
    scene.appendChild(escenario);

    this.agregarControles();           // Agrego controles para manejar el entorno
    this.construirMapa(escenario);    // Dibuja la topología de la red
    this.crearpaquete(escenario);     // Crea los paquetes
    this.crearhistorial(escenario);   // Crea el historial
},
```

Figura 4.18: Función “iniciarSimulacion” de la componente “simulacro”

“agregarControles” añade tres controles a la cámara de la escena: botón para pausar/reanudar, reiniciar y manejar el sonido. La forma en la que se agrega cada botón es la siguiente:

- Se crea la entidad con “createElement(“a-entity”)”.
- Le atribuimos una geometría junto con sus dimensiones. Para atribuir propiedades a la entidad usaremos “setAttribute()”.
- Se le indican las coordenadas para posicionarlo.
- Proporcionamos un aspecto a la entidad.

- Se le atribuye un identificador y al componente encargado de realizar esa función (Estos se analizarán mas adelante).
- Para finalizar lo agregamos a la cámara utilizando la función “appendChild()”

En el apartado 3.6 del sprint 6 se puede ver algunos ejemplos del proceso para agregar controles.

En cuanto a la función “construirMapa” lee el archivo JSON “netgui.json” y con esto se crea entidades para cada nodo definido en el archivo, junto con sus atributos y posiciones y aparte se crean las conexiones entre dispositivos. En el sprint 3 ?? es donde se explica el proceso que se ha seguido para interpretar la información que proporciona la interfaz gráfica NetGUI y agregarla al escenario.

Primero se explica la forma en la que leemos el archivo JSON llamado “netgui.nkp”. Utilizamos el método “fetch”<sup>6</sup>, este método realiza una solicitud HTTP GET al archivo y cuando obtiene la respuesta la convierte en un objeto JSON mediante “response.json()”, de esta forma se permite manipular fácilmente los datos contenidos en el archivo JSON. En la siguiente imagen 4.19 se muestra la parte del “fetch” del código junto con las constantes que extraen los valores. La constante “nodes” maneja el valor de la propiedad “nodes” del objeto JSON y la constante “connections” el valor de la propiedad “connections” del JSON, obteniendo un array para cada propiedad.

```
fetch('netgui.json')
  .then(response => response.json())
  .then(json => {
    const nodes = json.nodes;
    const connections = json.connections;
    // contenido...
  })
  .catch(error => console.error("Error al leer el archivo:", error));
```

Figura 4.19: Código Fetch de la función “construirmapa”

Una vez se tienen los arrays se itera sobre ellos con el método “.forEach” se trata cada “node” dentro de “nodes”. Un ejemplo de como queda cada node es el siguiente:

---

<sup>6</sup>Uso de Fetch: [https://developer.mozilla.org/es/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/es/docs/Web/API/Fetch_API/Using_Fetch)

```
{position: [-2.44, -17], type: 'NKHub', name: 'hub1'}
```

Y para acceder a estos valores se usa la siguiente notación node.type (accedo a “NKHub”) o en el caso de que un valor contenga un array como node.position hace falta añadirle [0] (accedo a -2.44) para referirse al primer valor que contiene e incrementándolo para los siguientes en adelante.

En esta función se hacen 3 cosas:

- **Crear entidades A-Frame para cada nodo:** En función del tipo de nodo se creará una entidad personalizada para cada tipo.

- **“NKCompaq”:** Se configura entidad “PC” en la escena, asignándole una geometría de caja, una textura específica y posicionándola según las coordenadas del nodo. Después se le añade los componentes “trazador”, “nombre-en-plano”, “direcciones” y en el caso de los PCs se le añade la componente “paquetes-enviados”. Esta ultima componente solo se le agrega al tipo “NKCompaq” porque son los únicos nodos de donde acaban los paquetes. Finalmente la entidad se añade al escenario (entidad creada en la función “iniciarsimulacion”).

```
(node.type.includes("NKCompaq")){
    entidad.setAttribute('geometry', {primitive: "box", width:2, height: 2, depth:2 })
    entidad.object3D.position.set((node.position[0]), 0, (node.position[1]))
    entidad.setAttribute("material", {src: "#pc", side: "double"});
    entidad.setAttribute("id", node.name)
    entidad.setAttribute('trazador', {forma: 'pc', intervalo: 1000});
    entidad.setAttribute('paquetes-enviados', ''); // Añadir el componente paquetes-enviados
    entidad.setAttribute('nombre-en-plano', {nombre: node.name,x: node.position[0],y: 0,z: node.position[1]});
    entidad.setAttribute('direcciones','');
    escenario.appendChild(entidad);
```

Figura 4.20: Agrego la entidad “NKCompaq”

La creación de las entidades del resto de nodos es similar, con algunas modificaciones en el aspecto (“material”), la geometría (“geometry”) y el valor forma que le introducimos al componente trazador.

- **“NKHub”:** Se crea una entidad con geometría de caja baja y ancha, se posiciona, se le añade la textura “hub”, la cual es el identificador de una imagen cargada previamente en “index.html”, y por último se le asignan los componentes de “trazador” (con su forma específica), “direcciones”, “paquetes-enviados” y “nombre-en-plano”.

- “**NKRouter**”: Generamos una entidad con forma cilíndrica, de color gris, se posiciona según sus coordenadas y se le añaden las componentes “trazador”, “direcciones”, “paquetes-enviados” y “nombre-en-plano”.
- “**NKSwitch**”: Crea la entidad con geometría caja el doble de alto que el nodo “NKHub”, la posicionamos, se le asigna su textura propia y por ultimo se agrega las cuatro componentes comunes de todos los nodos (“trazador”, “direcciones”, “paquetes-enviados” y “nombre-en-plano”).
- **Crear conexiones entre nodos:** En esta parte se itera sobre cada conexión en el array “connections” para acceder a las entidades origen y destino en la escena A-Frame a través de los identificadores “from” y “to” como se ha explicado anteriormente. Si ambas entidades se localizan en la escena se procede a crear una entidad con atributo “line” que define una línea que conecta la posición de las dos entidades y la añade al escenario como se aprecia en el siguiente fragmento de código.

```
const connections = json.connections;
// Conectar entidades según las conexiones
connections.forEach(connection => {
  const fromEntity = document.querySelector(`#${connection.from}`);
  const toEntity = document.querySelector(`#${connection.to}`);
  if (fromEntity && toEntity) {
    const line = document.createElement('a-entity');
    line.setAttribute('line', {
      start: fromEntity.getAttribute('position'),
      end: toEntity.getAttribute('position'),
    });
    escenario.appendChild(line);
  }
});
```

Figura 4.21: Código en el que se conectan entidades según las conexiones

- **Guardar conexiones entre nodos:** Este fragmento itera sobre “connections”, asegurando que cada nodo “from” y “to” tengan un array en “conexionesPorNodo” variable global (let conexionesPorNodo = []). En esta parte se agrega cada nodo destino al array del nodo origen y viceversa, de esta forma agregamos todas las conexione posibles que existen en cada nodo.

En cuanto a la función “crearpaquete” únicamente crea una entidad a la que se le asigna la componente “envio-paquetes” y el identificador “envio”.

La función “creahistorial” es muy similar a la función anterior, en este caso crea una entidad con identificador “EntidadHistorial” y le asigna la componente “historial” con el valor “movimientovertical” y “tiempo” que necesitan.

#### 4.2.2. Componente Envio-Paquetes

Este componente se encarga de gestionar la creación y envío de paquetes en la simulación. Al inicializar el componente, se define una lista para almacenar los “timeouts” llamada “almacenTimeouts” y un estado “pausa” con valor false. La función “init” ya explicada asegura que la entidad con identificador “escenario” este presente y luego llama a la función “envioPaquetes”.

“envioPaquetes” se encarga de leer el archivo “packages.json” que contiene los datos del paquete a enviar. Una vez lo carga, utilizando el método “fetch” recorre cada paquete para programar su envío usando el método “setTimeout”. Este método ejecuta una función después del numero de milisegundos que se indique. En la siguientes líneas de código podemos ver el “timeout”:

```
const delay = paquete.time * 1000;
const timeout = setTimeout (() => {
    if (!this.pausa) {
        numeroPaquetes++
        this.animacion(paquete, identificador, duración
    }
    this.almacenTimeouts = this.almacenTimeouts.filter(t => t.id !== timeout);
}, delay);
```

Se declara la constante “timeout” al que se le asigna el valor devuelto por “setTimeout”, de esta forma controlamos mas adelante cada “timeout” del paquete. El método suma un valor a la variable global “numero Paquetes” para llevar un seguimiento, ejecuta la función “animacion” que se encarga de mover los paquetes entre nodos y también actualiza la lista “almacenTimeouts” eliminando el timeout ya ejecutado. Todo esto lo ejecuta según se cumpla el tiempo de “delay” que es la constante encargada de almacenar el tiempo al que sale el paquete.

Con el objetivo de manejar el tiempo de los paquetes por si se pausa antes de que se lancen y eliminar el timeout sin llevar un registro del tiempo transcurrido desde su inicio agrego toda la información necesaria a la variable “almacenTimeouts”.

```
this.almacenTimeouts.push({ id: timeout, package: paquete, delay: delay,
startTime: Date.now(), remainingTime: delay, idpaquete: identificador,
durpaquete:duracion});
```

Esta estructura permite pausar, reanudar o cancelar timeouts de manera eficiente, ya que toda la información necesaria se encuentra almacenada en “almacenTimeouts”, en la función pausar y reanudar trataremos los datos que se manejan.

“animacion” es la función encargada de animar el movimiento de un paquete a través de la ruta definida de tal manera que cada movimiento corresponde al desplazamiento que hace de un nodo a otro. Una vez el paquete llega a su destino emite un sonido para notificar al usuario, llama a la función “agregarPaquete” de la componente “paquetes-enviados” que se encarga de posicionar los paquetes uno encima de otro al lado del nodo según su orden de llegada, también se le quita al paquete el componente trazador para que no siga dejando rastro y por último se registra el final del envío de dicho paquete para llevar una cuenta de los paquetes enviados.

Dentro de esta función se realiza las siguientes tareas para su correcto funcionamiento, ordenadas según su aparición en el código:

- **Verificación de las rutas:** Se comprueba que la ruta del paquete tenga al menos dos puntos, de lo contrario no se puede realizar el movimiento entre nodos y se imprime un error para informar.
- **Asignación de valores por defecto:** En esta parte le otorgamos valores por defecto en el caso de que no se le haya indicado este en la configuración del archivos “packages.nkp”. Se asigna un identificador único al paquete (paquete + numero de este) en caso de que no se le proporcionase uno y se establece la duración de la animación entre nodos, utilizando un valor por defecto (2000 milisegundos) si no se ha especificado.
- **Creación y posicionamiento de la entidad paquete:** Se crea una entidad visual para representar el paquete con geometría de caja y un color aleatorio al paquete, el cual no se

puede repetir si un paquete en tránsito con ese color. Además se coloca el paquete en la posición del nodo de salida inicial.

- “**crearCuadroNombrePaquete**”: Llamo a esta función para añadir un plano con información del paquete, como origen destino y su identificador y va siempre pegado a él.
- **movimiento**: Es una función recursiva que gestiona el desplazamiento de un paquete a través de una secuencia de nodos en una ruta específica. Cuando se llama a la función, comprueba si el índice actual “i” ha superado la longitud de la ruta, y si es así ejecuta algunas acciones como comprobar si ha llegado al nodo destino y si es así reproduce un sonido en caso de que este no esté silenciado, también se agrega a la lista de paquetes entregados del nodos destino por la componente “paquetes-enviados”, se elimina el atributo de la componente “trazador” para evitar que siga generando un rastro de paquetes innecesarios y se incrementa el contador de paquetes entregados. Cuando el contador de paquetes entregados es igual al total de paquetes se finaliza la simulación una vez pasado dos segundos.

Por ultimo, dentro de movimiento se encuentra la parte de A-Frame que se encarga de la animación de los paquetes entre nodos que corresponde a las siguientes líneas de código:

```
const siguiente = document.getElementById(paquete.route[i]);
if (siguiente) {
    pqt.setAttribute('animation', {
        property: 'position',
        to: `${siguiente.object3D.position.x} ${siguiente.object3D.position.y}
              ${siguiente.object3D.position.z}`,
        dur: duracion,
        easing: 'linear',
        resumeEvents: 'animation-resume',
        pauseEvents: 'animation-pause'
    });
    pqt.addEventListener('animationcomplete', () => movimiento(i+1), {once: true});
}
```

Si el paquete no ha llegado a su destino se obtiene el siguiente nodo en la ruta utilizando el indice “i”, si existe el siguiente nodo de la ruta se configura una animación para mover al paquete a la posición del siguiente nodo (“paquete.route[i]”). La animación se define mediante los siguientes atributos de A-Frame:

- **property:** Define qué propiedad del objeto se anima, en este caso la posición “position”, esto significa que la posición del paquete cambiará a lo largo del tiempo definido.
- **to:** Indica el valor final al que debe llegar la propiedad animada, en este caso la posición final del paquete corresponde a la posición del siguiente nodo.
- **dur:** Determina la duración de la animación en milisegundos.
- **easing:** Es la forma en la que se acelera la animación, esto afecta la velocidad con la que se realiza la transición. En el caso de nuestro simulador hemos seleccionado “linear” porque nos proporciona una velocidad constante de principio a fin.
- **resumeEvents:** Especifica los eventos que reanudan la animación pausada, en este caso cuando se emite el evento “animation-resume”.
- **pauseEvents:** Especifica los eventos que pausan la animación en curso, en este caso cuando se emite el evento “animation-pause”.

Por ultimo una vez se finaliza la animación se incrementa el índice “i” en 1 y se llama de manera recursiva a la función movimiento.

“pausar” tiene como función detener la ejecución de todas las animaciones y eventos de tiempo en curso. Lo primero que se hace es marcar el sistema del estado como pausado y posteriormente cancela todos los eventos de tiempo “timeouts” en curso, calcula el tiempo restante para cada “timeout” en “almacenTimeouts” para finalmente actualizar la lista de eventos de tiempo con los valores calculados.

“reanudar” es la función encargada de restablecer y reiniciar todos los eventos de tiempo que fueron pausados, actualizando los timeouts con los tiempos proporcionados al pausar el tiempo y asegura que el estado del sistema sea reanudado.

#### 4.2.3. Componente Pausa y Reproducir

La componente “pausa-reproducir” permite alternar entre el estado pausa y reproducción del simulador cuando el usuario hace clic en el botón. En función de su estado se modifica el material del botón, para facilitar al usuario la acción que sucederá al darle al botón, por ejemplo

si la simulación esta pausada el botón tendrá el signo de reanudar, indicando que al clicarlo reanuda la simulación. Además, emite eventos para pausar o reanudar las animaciones y permite lo mismo con otro componentes como el “historial”, encargado de representar el transcurso del tiempo a través del eje “y” y también se llama a las funciones “pausar” y “reanudar” de la componente “envio-paquetes”.

Dentro de la componente tenemos 4 funciones:

- **“setPause”:** Cambia el estado a pausa, actualiza el aspecto del botón al de “reproducir”, llama al método “pausar” del componente “envio-paquetes” y al método “pausado” del componente “historial”, y emite el evento pausa para todos los elementos con animación. Cuando se le llama a esta función toda la simulación se pone en pausa y se cambia el aspecto del botón de pausa al botón reproducir.
- **“setReproducir”:** Realiza los mismos pasos de la función anterior pero cambiando el estado a “reproducir” y llamando al método “reanudar” de la componente “envio-paquetes” y “reanuda” de la componente “historial” al igual que emite el elemento reanudación para todos los elementos con animaciones. Es decir reanuda toda la simulación haciendo que esta continúe desde el mismo punto en el que se pauso.
- **“finalSimulacion”:** Fuerza la pausa de la simulación, elimina el botón “pausa-reanudar” y llama a la función “crearCuadroFin” que crea un recuadro indicando que la simulación ha finalizado. A este método se le llama en la componente “envio-paquetes” cuando se termina de enviar todos los paquetes.
- **“getState”:** Esta función sirve únicamente para devolver el estado actual del componente (“reproducir” o “pausa”).

En resumen esta componente gestiona la lógica de pausa y reanudación de la simulación y actualiza el estado y la apariencia del botón.

#### 4.2.4. Componente Historial

La componente “historial” controla el movimiento vertical del elemento que contiene las entidades que se dejan como rastro de los paquetes. Haciendo que este eje vertical “y” represente el transcurso del tiempo en la simulación. Su función principal es animar el movimiento

de la entidad que lo posee dentro de la escena y gestiona el estado pausa y reanudación para poder controlar el movimiento. Esta componente tiene dos propiedades configurables “movimientoVertical” y “tiempo”, la primera sirve para indicar que distancia se desea que suba la entidad respecto al eje “y” y la segunda para determinar el tiempo que debe transcurrir para cambiar la posición en el eje “y”. Los valores que están asignados por defecto permiten generar un movimiento fluido del historial. Dentro de la componente tenemos 3 funciones:

- **“movimientohistorial”:** Establece un intervalo que incrementa la posición vertical “escalay” del elemento, dependiendo del valor de “movimientoVertical” y si el estado no esta en pausa actualiza el atributo posición de la entidad. setInterval es un método de JavaScript que permite llamar a una función en intervalos específicos de tiempo
- **“pausado”:** Cambia el estado de “pausa” a “true” y detiene el intervalo encargado del “movimientohistorial”.
- **“reanuda”:** Esta función cambia el estado “pausa” a “false” y reanuda el movimiento del historial llamando a la función “movimientohistorial” vista anteriormente.

Gracias a esta componente se obtiene una representación realista del transcurso del tiempo en la simulación a través del eje “y” de A-Frame.

#### 4.2.5. Componente Trazador

Es la componente encargada de crear una traza visual del paquete en movimiento, generando periódicamente entidades que siguen la posición del paquete a modo de rastro, también genera trazas de los dispositivos de la red para dejar una guía de la ubicación de los nodos junto con el rastro de los paquetes. Junto con la componente “historial” estas dos componentes se encargan en gran medida de representar toda la simulación a lo largo del tiempo, ya que la componente trazador inyecta las entidades en la entidad de la componente historial, y esta se mueve para representar el transcurso del tiempo.

Inicializa configurando las propiedades “forma” e “intervalo”, la propiedad forma determina la geometría y aspecto de la entidad que se deja como traza o rastro mientras que intervalo marca el tiempo necesario para poner dicha entidad en el historial.

Lo primero que hace la componente es obtener la entidad “historial” y los atributos “material”, “color” e “id” de cada elemento que posee la componente. Con esta información ya puede iniciar la función traza.

“traza” es la función encargada de establecer un intervalo que crea una nueva entidad en la posición actual de la entidad cada cierto tiempo (según “intervalo”). En función de la forma que tenga atribuida se generará una entidad con una geometría y aspecto específico, dentro de la función se manejan estas formas:

- **Caja:** Esta es una de las dos opción que tiene un paquete para dejar su rastro en el historial, como su nombre indica deja una caja de dimensiones pequeñas y del mismo color que el paquete.
- **Esfera:** Es la otra opción, deja un rastro de esferas con radio de cinco centímetros del mismo color que el paquete.
- **PC:** Añade una caja alta y fina con aspecto de PC.
- **Router:** Agrega un cilindro alto y con un radio de dos decímetros.
- **Hub:** Una caja alta y estrecha con aspecto de Hub.
- **Switch:** Una caja alta y fina con aspecto de Switch.

A las entidades se les agrega la componente “difuminado” que se encarga de difuminar un 50 % las entidades agregadas y se posicionan con “localPosition” dentro de la entidad historial, de tal manera que si esta se mueve, mueve todas las entidades agregadas en su interior.

Por ultimo tenemos la función “remove” que permite cancelar el intervalo cuando se elimina la componente “trazador” de la entidad, de esta forma evitamos que los paquetes sigan agregando geometrías a la escena una vez ha finalizado su envío.

#### 4.2.6. Componente MostrarHistorial

La componente “mostrar-historial” permite al usuario interactuar con los paquetes en la simulación de redes de ordenadores, de tal manera que se muestra únicamente el historial de los paquetes clicados y se oculta el resto de tal manera que se puede revertir todos los

cambios realizados dando al botón “vuelta atrás”. Cuando el usuario hace clic en un paquete, la componente alterna su estado entre seleccionado y no seleccionado mostrando únicamente los seleccionados.

En este componente se define el array vacío “mostrar-historial” donde se almacenan los paquetes seleccionados. Y una vez se clica con la entidad, lanza las dos funciones que posee:

**“mostrar”:** Esta función pausa la simulación en caso de que este reproduciendo. Obtiene el identificador paquete clicado a través de “getAttribute” y se agrega en el array paquetesClicados, en caso de que se encuentre ya en el array se saca, de tal manera que si se clica dos veces el mismo paquete con el primer clic se muestra y con el segundo se oculta.

```
const entidades = document.querySelectorAll('[difuminado]');
entidades.forEach(entidad => {
    if (this.paquetesClicados.includes(entidad.getAttribute('id')) ||
        (entidad.getAttribute('id') == "guia")) {
        entidad.setAttribute('visible', 'true');
    } else {
        entidad.setAttribute('visible', 'false');
    }
});
```

En estas líneas de código se recorre todas las entidades con componente “difuminado” y se muestra aquellas que poseen un identificador registrado en “paquetesClicados”. Por ultimo en función de si la simulación se ha finalizado y tiene el cuadro de texto que indica el fin o no, y tiene el botón pausar/reproducir los elimina para posicionar en su lugar al botón que revierte los cambios.

**“botonAtras”** crea un botón para revertir la visualización del historial posicionándolo en la cámara y se elimina al hacer clic, mostrando todos los elementos del historial y restaurando el botón pausar/reanudar o el cuadro informativo de fin de simulación en función del estado que tuviese anteriormente.

#### 4.2.7. Componente Paquetes-Enviados

Este componente llamado “paquetes-enviados” se encarga de gestionar una pila de paquetes, la llamamos pila debido a que se guardan los paquetes según su orden de llegada de tal

manera que el primero estará abajo del todo y se irán apilando el resto encima según su orden de menor a mayor tiempo de llegada. El componente tiene una propiedad “alturaPaquete” que indica la altura que cada paquete ocupara en la pila, de tal forma que aparecen los paquetes uno encima de otro sin espacio entre medias. Esta componente es útil para visualizar la acumulación de paquetes de manera ordenada.

#### 4.2.8. Componentes y Funciones Auxiliares

En este apartado se detallan las funciones y componentes auxiliares que, aunque no sean los mas importantes para el funcionamiento del simulador, son esenciales para garantizar su correcto desempeño.

**Componente Difuminado:** Esta diseñado para crear un efecto de desvanecimiento a los objetos agregados por el componente “trazador”. Para simular este efecto de desvanecimiento se ha realizado reduciendo gradualmente su opacidad durante un periodo específico de tiempo (4 segundos). Este componente es útil porque permite diferenciar cuales son las trazas mas recientes ya que estas tienen mayor opacidad respecto al resto. Se definen dos propiedades “tiempo” que marca la duración de la animación y “opacidad” que representa el nivel de opacidad al que llega pasado el tiempo. Una vez se inicializa la componente se configura la animación con “animation\_fadeout” especificando la propiedad que se va animar (“material.opacity” controla la opacidad del material), el valor objetivo de la propiedad, la duración de la animación y la aceleración (constante en este caso) como podemos apreciar n la siguientes líneas del código.

```
this.el.setAttribute('animation_fadeout', {
  property: 'material.opacity',
  to: this.data.opacidad,
  dur: this.data.tiempo,
  easing: 'linear'
});
```

**Componente Direcciones:** Se utiliza para mostrar las conexiones de un nodo en la red cuando el nodo es clicado, y esta información desaparece transcurrido cinco segundos. Para su funcionamiento se crea un plano y una entidad texto que muestra las conexiones del nodo almacenadas en “conexionesPorNodo” ambas entidades se agregan a una entidad aparte llamada “grupo” y se

le añade el atributo “look-at , [camera]” para que siempre este mirando a la cámara. En cuanto a la posición es respecto a la entidad propia modificando las coordenadas del eje “y” y “z”.

**Componente Sonido:** Gestionamos el estado del sonido, permitiendo alternar entre el estado “sonando” y “muteado”. Al hacer clic en la entidad (botón de sonido), esta cambia de aspecto según el estado de la componente, para informar al usuario si el sonido esta activado o no. Es importante la función “getState” porque se utiliza en la componente “envio-paquetes” 4.2.2 para conocer el estado de sonido.

**Componente Reinicio:** Reinicia la página actual cuando se hace clic en la entidad a la que está asociado, en este caso es al “botonReinicio” generado en la función “agregarControles” de la componente “simulacro”. Gracias a la función “location.reload()” de JavaScript se recarga la pagina actual desde la URL original.

**Componente Nombre en Plano:** Este componente esta diseñado para crear y posicionar una etique de texto encima de cada nodo dentro de la escena. De tal manera que se coloca una etiqueta de texto con el identificador del nodo en una posición específica (encima del nodo) y siempre mirando a la cámara (atributo “look-at”), de tal forma que se permite visualizar el nombre del nodo desde cualquier ángulo.

**Función crearCuadroNombrePaquete:** Crea una entidad que muestra un cuadro con información del paquete con el que va. Esta información es el identificador del paquete, su nodo origen y el nodo destino. Esta función es útil para proporcionar información sobre un paquete en la simulación de redes de ordenadores. La estructura es muy similar a la componente “direcciones”, cambiando la información que imprime, el color de la letra a blanco y del plano a negro.

**Función crearCuadroFin:** Tiene como objetivo crear el cuadro que informa al usuario que la simulación ha finalizado, siguiendo la misma estructura que la función anterior.

**Función colorAleatorio y liberarColor:** Estas funciones permiten crear un color aleatorio de tal manera que no se pueda repetir ese mismo color sin antes liberarlo. Es una función

simple con la que generamos un color hexadecimal aleatorio siempre y cuando no este ya en uso dentro del conjunto “coloresActivos”, para comprobarlo utilizamos el bucle “do...while” que genera colores aleatorios hasta encontrar uno que no este en activo. Cuando se saca ese color que no esta en activo se añade a “coloresActivos” y para liberarlo se debe usar la función “liberarColor”, a la cual se llama una vez el paquete ha terminado el simular el envío.

# Capítulo 5

## Conclusiones

Una vez se ha finalizado el proyecto podemos hablar del esfuerzo que ha costado llevarlo a cabo, lo que se ha aprendido y el resultado obtenido, que ha sido este simulador de redes de ordenadores en realidad extendida. En este apartado se presentan las conclusiones, las cuales se estructuran en varios apartados: consecución de objetivos, aplicación de lo aprendido, lecciones aprendidas, esfuerzo realizado y trabajos futuros.

### 5.1. Consecución de objetivos

Esta sección evalúa el grado de cumplimiento de los objetivos planteados al inicio del proyecto donde se diferenció entre objetivos generales y específicos con el fin de crear un simulador de redes de ordenadores.

El simulador de redes de ordenadores interactivo basado en realidad extendida que se marcó como objetivo general se ha desarrollado exitosamente, permitiendo una visualización tanto en navegadores webs como en dispositivos de realidad virtual (meta quest). Este permite la dualidad de accesibilidad, lo que hace que el simulador pueda ser utilizado en una amplia gama de entornos, cumpliendo así el objetivo principal de crear una herramienta que mejore la comprensión y aprendizaje sobre la gestión de redes informáticas.

Además se presentaron una serie de objetivos específicos a seguir para el desarrollo de la simulación de redes de ordenadores. A continuación se presenta el estado de estos objetivos:

- **Utilización de herramientas de desarrollo front-end:** El simulador ha sido desarrollado utilizando desarrollo web front-end, en este caso ha nos hemos centrado en utilizar el

framework A-Frame.

- **Adaptabilidad a diferentes dispositivos:** Se ha logrado tener una herramienta accesible en cualquier dispositivo con acceso a internet sin necesidad de utilizar extensiones o plugins externos. Asegurando compatibilidad con dispositivos de realidad virtual.
- **Lectura de datos JSON:** Hemos desarrollado un mecanismo eficiente dentro del simulador para leer y procesar datos en formato JSON, permitiendo una carga dinámica de la topología de la red (netgui.nkp) y paquetes (packages.nkp).
- **Visualización dinámica de la red:** Se ha logrado una herramienta que genera una representación visual tridimensional de cualquier red diseñada en NetGUI.
- **Historial de paquetes:** El simulador permite la visualización del historial de todos los paquetes y configurarlo de manera que solo sean visibles el rastro de paquetes específicos que se hayan seleccionado. Así los usuarios pueden rastrear y analizar el recorrido de cada paquete enviado en la red.
- **Intuitivo y fácil:** Se ha diseñado una interfaz de usuario intuitiva y sencilla en lo que a los botones respecta. De esta forma el usuario puede interactuar con la simulación sin la necesidad de manual, respecto a algunas funcionalidades como la visualización del historial y las conexiones entre nodos es necesario informar sobre ello.
- **Experiencia inmersiva:** Se han integrado elementos visuales y auditivos para mejorar la inmersión y comprensión del proceso de simulación, ejemplo de ello son el sonido que notifica la llegada del paquete a su destino y la animación de los paquetes y difuminado del rastro que dejan según el transcurso del tiempo.
- **Interacción con la animación:** Inicialmente se marco como objetivo este mínimo de funcionalidades con la simulación, las cuales han sido desarrolladas, de tal forma que el ususario posee el control total de sobre el flujo de la animación. Además se ha añadido otra funcionalidad que permite reiniciar la simulación al completo.
- **Interacciones con la escena:** Se puede interactuar con los paquetes (mostrando su historial) y dispositivos (visualización de información sobre conexiones) obteniendo información extra del simulador.

- **Mostrar información relacionada:** Se han implementado cuadros de texto que muestran los identificadores de cada sistema y paquete, junto con información extra de este, proporcionando información clara y accesible durante la simulación.
- **Creación de demos:** Se han desarrollado varias demos que permite visualizar el resultado final del simulador con diferentes escenarios para mostrar

En resumen, todos los objetivos planteados han sido cumplidos de manera satisfactoria, logrando desarrollar un simulador de redes de ordenadores.

## 5.2. Aplicación de lo aprendido

En este apartado se refleja como los conocimientos adquiridos durante la carrera de Ingeniería en Sistemas Audiovisuales y Multimedia se han integrado y aplicado en el desarrollo del simulador de redes de ordenadores.

A lo largo de la carrera he recibido información de diversas ramas, para la realización de este proyecto nos vamos a centrar en dos ramas programación y redes:

**Programación:** En esta rama incluyo todas las asignaturas de programación, las cuales contribuyen de manera significativa a proporcionarnos un pensamiento lógico y capacidad de resolución de problemas en la programación. A continuación, detallo las asignaturas y los conocimientos utilizados:

- **Informática II y Protocolos para la transmisión de Audio y Vídeo en Internet:** He juntado estas dos asignaturas en este apartado porque en ambas se trabaja con Python, lenguaje utilizado para programar un archivo que pase la información de los archivos “.nkp” a formato JSON. Además en la asignatura de Protocolos para la transmisión de Audio y Vídeo en Internet también se ha aportado conocimientos de la rama de redes.
- **Construcción de Servicios y Aplicaciones Audiovisuales en Internet y Laboratorio de Tecnologías Audiovisuales en la Web:** Estas dos asignaturas van dadas de la mano ya que Laboratorio es la continuación de Construcción, en ambas se trabaja con JavaScript, lenguaje principal de la herramienta desarrollada, además en la asignatura de construcción

se centra en ensañarnos conocimientos necesarios para frontend, indispensable para el proyecto, ya que se trabaja el desarrollo del lado del usuario.

- **Gráficos y Visualización en 3D:** Conocimientos en WebGL y Three.js, biblioteca con la que hemos trabajado para el posicionamiento de objetos y su geometría.
- **Informática I:** Nos introdujo en el mundo de la programación enseñándonos conceptos básicos y fundamentales.

**Redes:** Se engloba todo lo relacionado con las redes de comunicaciones, protocolos de red, fundamentos y arquitecturas de red junto con ejemplo y utilización de estas redes.

- **Arquitectura de Internet y Sistemas Telemáticos para Medios Audiovisuales:** Estas dos asignaturas van dadas de la mano , en ellas se enseñan los protocolos de encaminamiento , redes de ordenadores, calidad de servicio, OSPF, BGP ... Y lo mas importante de cara al proyecto, se nos enseña a utilizar la interfaz gráfica NetGUI, indispensable para crear las topologías de nuestra simulación de redes.
- **Protocolos para la transmisión de Audio y Vídeo en Internet:** Mencionada en la parte de programación, en la asignatura se imparten conocimientos de protocolos, entre ello la realización de la practica final basado en una comunicación servidor cliente utilizando protocolo TCP. De esta forma nos aporta contexto necesario sobre las redes.
- **Tecnologías de televisión en internet:** Al igual que las anteriores nos aporta conocimientos de redes fundamentales, en este caso se centra sobre todo en entender las características de las redes de manera practica.

### 5.3. Lecciones aprendidas

Durante la realización del Trabajo Fin de Grado he adquirido una serie de conocimientos clave. He profundizado en el uso de A-Frame, necesario para crear experiencias de realidad extendida en el navegador. Entendiendo la arquitectura y creando componentes personalizados para crear y hacer funcionar una simulación. He mejorado mi manejo en programación con JavaScript, esencial para la manipulación y animación de entidades en la escena y la manipulación

del DOM. A parte de todo esto he adquirido experiencia en el diseño y desarrollo de interfaces de usuario intuitivas y accesibles gracias a la mejora constante de las demos pensando en el usuario.

Se me han proporcionado gafas de realidad virtual para realizar pruebas, aprendiendo así a utilizarlas y ver las distintas funcionalidades que ofrece.

Por ultimo he aprendido a utilizar LaTeX para la realización de documentos técnicos como es esta memoria

## 5.4. Esfuerzo realizado

Inicie el proyecto en noviembre del año 2023, por lo que el tiempo dedicado al TFG es entorno a 9 meses. Debido a mi situación laboral la gran mayor parte del tiempo dedicado a la semana a realizar el proyecto se daba los fines de semana y aparte 1 hora o dos de lunes a jueves antes de dormir. Las horas semanales oscilan entorno a 14 y 18, varia sobre todo en función de la etapa del proyecto, ya que en algunos casos se obtenían los resultados sin mucha dificultad. Destacar que durante el mes de enero se hizo un parón de 15 días para terminar de preparar la ultima asignatura pendiente de la carrera.

El tiempo dedicado para cada sprint es el siguiente:

- Sprint 1 primer contacto: Realizado en la segunda quincena de Noviembre
- Sprint 2 Trabajo con animaciones: Comienza a principios de diciembre, finalizando entorno al 22 de diciembre justo antes de un parón debido a las fiestas y a la preparación de un examen.
- Sprint 3 Comienzo de proyecto: Realizado en la segunda quincena de Enero y principios de Febrero
- Sprint 4 Creacion de paquetes y puesta en movimiento: Desde la segunda semana de Febrero hasta principios de Marzo.
- Sprint 5 Implementación de historial y mejora visual: Realizado en la segunda quincena de marzo y primera de Abril.
- Sprint 6 Interactividad con la simulación: Desde finales de Abril hasta finales de mayo.

- Sprint 7 Creación de escenario final: Durante el mes de Junio

## 5.5. Trabajos futuros

En este apartado se mencionan posibles mejoras o implementaciones que se puedan hacer en un futuro en este proyecto.

- Utilización de trazas reales en lugar de generar los paquetes de manera manual, es una posibilidad que se puede realizar a través de wireshark (por ejemplo) que nos proporciona trazas reales en formato JSON y habría que adaptar esa información para imprimirla en forma de paquetes. Como en el proyecto se ha centrado en el aspecto visual no se ha podido abarcar este tema.
- Agregar funcionalidades en el simulador como rebobinar o avanzar en el tiempo para que si la simulación es larga se pueda ir directamente al punto deseado de manera rápida o retroceder para repetir el movimiento de un paquete.
- Implementar un menú de configuración del simulador a tiempo real, donde se pueda modificar los ajustes que se configuran al inicio o tengan valores por defecto en su ausencia. Por ejemplo la velocidad a la que asciende el historial, la geometría que deja de rastro los paquetes, la duración de las animaciones al mover los paquetes entre nodos etc. De tal forma que se pueda visualizar en tiempo real esos cambios.
- Creación de topologías de la red en el mismo escenario, crear un escenario previo que sirva para dibujar la topología de la red, de igual manera que hacemos con NetGUI. Básicamente en lugar de crear la red en NetGUI sería simular el funcionamiento de este en realidad virtual, donde una vez se termina de diseñar la red habría que configurar los paquetes y luego se procedería a simular todo. De esta manera se le facilita a los usuarios que quieran realizar su propia escena el proceso.

# Bibliografía

[1] A-Frame: Documentación web disponible en:

<https://aframe.io/docs/1.6.0/introduction/>. [Accedido: 26-jun-2024].

[2] Three.js: Documentación web disponible en:

<https://threejs.org/>. [Accedido: 26-jun-2024].

[3] WebGL: Documentación con tutorial de WebGL en:

[https://developer.mozilla.org/es/docs/Web/API/WebGL\\_API/Tutorial](https://developer.mozilla.org/es/docs/Web/API/WebGL_API/Tutorial). [Accedido: 26-jun-2024].

[4] WebGL: Documentación estandar WebGL en:

<https://www.khronos.org/webgl/>. [Accedido: 26-jun-2024].

[5] WebXR: Documentación web disponible en:

[https://developer.mozilla.org/en-US/docs/Web/API/WebXR\\_Device\\_API](https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API). [Accedido: 26-jun-2024].

[6] S. M. J. Z. Arturo Montejo Ráez. *Curso de Programación Python*. ANAYA, Junio 2019.

[7] J. D. L. Castillo. *HTML5, CSS y JavaScript. Crea tu web y apps con el estandar de desarrollo*. RC Libros, Enero 2016.

[8] D. Flanagan. *JavaScript The Definitive Guide*. O'Reilly Media, Inc, Mayo 2011.

[9] JavaScript: Documentación web disponible en:

<https://developer.mozilla.org/es/docs/Web/JavaScript>. [Accedido: 26-jun-2024].

- [10] Estandar internacional que describe JavaScript: ECMA - 262 <https://ecma-international.org/publications-and-standards/standards/ecma-262/>. [15<sup>a</sup> edición, junio de 2024].
- [11] HTML5: Documentación web disponible en:  
<https://www.manualweb.net/html5/>. [Accedido: 26-jun-2024].
- [12] DOM: Documentación web disponible en: <https://developer.mozilla.org/es/docs/Glossary/DOM>. [Accedido: 26-jun-2024].
- [13] NetGUI: Pdf informativo de NetGUI:  
<https://mobiquo.gsync.urjc.es/dns-and-bgp/intro-netgui.pdf>.  
[Fecha creación: Septiembre de 2021 por la Universidad Rey Juan Carlos].
- [14] Python: Documentación con tutorial de Python en:  
<https://docs.python.org/es/3/tutorial/>. [Accedido: 26-jun-2024].
- [15] Latex: Documentación web disponible en: :  
<https://www.latex-project.org/>. [Accedido: 26-jun-2024].