



**Universidad
Rey Juan Carlos**

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

Curso Académico 2021/2022

Trabajo Fin de Grado

**VISUALIZACIÓN DE TRAZAS DE NETGUI
EN REALIDAD VIRTUAL**

Autor: Ignacio Cruz de la Haza
Tutor: Dr. Jesús María González Barahona

Trabajo Fin de Grado

Visualización de trazas de NetGUI en realidad virtual

Autor: Ignacio Cruz de la Haza

Tutor: Dr. Jesús María González Barahona

La defensa del presente Proyecto Fin de Grado/Máster se realizó el día de 202X, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Móstoles/Fuenlabrada, a _____ de _____ de 202X

©2022 Ignacio Cruz de la Haza

Algunos derechos reservados

Este documento se distribuye bajo la licencia

“Atribución-CompartirIgual 4.0 Internacional” de Creative Commons,

disponible en <https://creativecommons.org/licenses/by-sa/4.0/deed.es>

*Dedicado a
mi familia*

Agradecimientos

Me gustaría dedicar este proyecto en primer lugar a mi familia, por haberme inculcado unos valores extraordinarios y una formación académica que hoy en día me permiten ser quien soy. También por su apoyo desde el minuto cero y por haberme servido de conejillo de indias en mis diferentes pruebas y testeos. Este proyecto es lo que es gracias a vuestras críticas constructivas.

También me gustaría dedicárselo a mi pareja, por haber compartido con orgullo los buenos momentos y haberme iluminado en los momentos de más oscuridad, ya no solo a lo largo de este proyecto sino durante toda la carrera.

Por último, quería agradecer por su dedicación y esfuerzo a todos los profesores de la universidad Rey Juan Carlos con los que he tenido el placer de coincidir a lo largo de mis años de estudio, por haber despertado en mi inquietudes y por haberme hecho entender muchas cosas del mundo, y en especial agradecer al profesor Jesús María González Barahona por haber sido el encargado de llevarme este proyecto, por estar siempre abierto a ayudar y por hacer el camino mucho más ameno.

Gracias a todos.

AGRADECIMIENTOS

Resumen

Este proyecto describe la construcción de una herramienta para visualizar trazas de comunicación entre nodos y presentar la escena en un entorno de realidad virtual. La topología del escenario a representar en 3D y las trazas de comunicación han sido extraídas de la herramienta NetGUI que sirve para crear interfaces de comunicación de redes e interconexión de datos entre sus diferentes nodos.

La herramienta será capaz de interpretar y unificar las capturas tomadas y visualizar en un entorno de realidad virtual los paquetes que forman parte de un escenario de comunicación real.

La escena 3D será interactiva y el usuario podrá navegar por ella libremente, además de iniciar, detener y reanudar la animación de los paquetes en cualquier momento, así como pinchar en ellos para desplegar la información que transporta de cada nivel de comunicación.

También permite diferenciar máquinas según el nivel al que se comunican, distinguiendo entre ordenadores, conectores o hubs y routers. Cada uno presentará un modelo visual diferente para distinguirlos fácilmente.

La tecnología utilizada para el desarrollo de este proyecto es el framework A-Frame que basado en los lenguajes de programación HTML y Javascript es capaz de construir escenas y experiencias en realidad virtual. Otras tecnologías utilizadas a lo largo del proyecto son NetGUI, Wireshark, Python o LaTeX para el desarrollo de la memoria.

RESUMEN

Summary

This project describes the construction of a tool for display communication traces between nodes and present the scene in a virtual reality environment. The scene topology to represent in 3D and the communication traces have been extracted from the tool NetGUI, which helps to create network communication interfaces and connect data between its different nodes.

The tool will be able to interpret and unify the captures taken and visualize in a virtual reality environment the packets that are part of a real communication scenario.

The 3D scene will be interactive and the user will be able to navigate through it freely, in addition to starting, pausing and resuming the packets animation anytime, as well as click over them to deploy the information that they carry of each communication level.

It also allows to differentiate machines according to the level they communicate, distinguishing between computers, hubs and routers. Each one will present a different visual model to distinguish them easily,

The technology used to develop this project is the framework A-Frame, which is based on programming languages such as HTML and Javascript and it is able to build scenes and experiences in virtual reality. Other used technologies throughout this project are NetGUI, Wireshark, Python or LaTeX for the memoir writing.

SUMMARY

Índice general

1	Introducción	1
1.1	Objetivos del proyecto	2
1.1.1	Objetivo general	2
1.1.2	Objetivos específicos	2
1.2	Planificación temporal	3
1.3	Materiales	3
1.4	Estructura de la memoria	3
2	Estado del arte	5
2.1	NetGUI	6
2.2	Wireshark	7
2.3	Three.js	9
2.4	WebXR	11
2.5	WebGL	12
2.6	HTML5	12
2.7	A-Frame	14
2.8	Javascript	17

ÍNDICE GENERAL

2.9 Python	18
2.10 JSON	19
2.11 Visual Studio Code	19
2.12 LaTeX	20
2.13 Overleaf	20
3 Diseño e implementación	23
3.1 Metodología	23
3.2 Etapa 0: Aprendizaje	25
3.2.1 Probando los primeros componentes	26
3.2.2 Manejadores de eventos y cambios en la escena	27
3.3 Etapa 1: Primeras implementaciones	29
3.3.1 Posicionamiento de entidades en la escena	29
3.3.2 Uso y manejo de componentes	31
3.3.3 Anidando componentes	32
3.3.4 Manejo de eventos entre componentes	33
3.3.5 Unificación de lo aprendido	35
3.3.6 Animaciones	36
3.3.7 Profundizando en las posibilidades de A-Frame	38
3.3.8 Prototipo de la etapa 1	38
3.4 Etapa 2: Representación de la topología de un escenario	38
3.4.1 Representación de nodos por paso de parámetros	39
3.4.2 Representación de nodos en topología circular	39

ÍNDICE GENERAL

3.4.3	Lectura de parámetros desde archivo JSON	40
3.4.4	Lectura de un fichero de paquetes real	41
3.4.5	Representación de la topología de un escenario real de NetGUI	42
3.5	Etapa 3: Lectura de paquetes y animación en la escena	44
3.5.1	Lectura y representación de paquetes reales en la escena	44
3.5.2	Creando componente para detener y reanudar la escena	48
3.5.3	Controlando el inicio de las animaciones	49
3.6	Etapa 4: Añadiendo información a los paquetes	50
3.6.1	Información de niveles al clicar en los paquetes	50
3.6.2	Representación de la información de los paquetes	53
3.7	Etapa 5: Experiencias en realidad virtual	58
3.7.1	Mejorando aspecto visual de los componentes	58
3.7.2	Unificando capturas	62
3.7.3	Preparando la escena para la realidad virtual	65
3.7.4	Añadiendo modelos visuales a los componentes	66
4	Resultados	71
4.1	Manual de usuario	71
4.1.1	Usuario final	71
4.1.2	Composición de la escena	75
4.1.3	Proceso completo	78
4.2	Detalles de implementación	81
4.2.1	Componente selector	81

ÍNDICE GENERAL

4.2.2 Componente immersiveMode	82
4.2.3 Componente network	83
4.2.4 Componente packet	85
5 Conclusiones y trabajos futuros	93
5.1 Consecución de objetivos	93
5.2 Aplicación de lo aprendido	94
5.3 Lecciones aprendidas	95
5.4 Planificación temporal	96
5.5 Trabajos futuros	96
Referencias	99

Capítulo 1

Introducción

Entender una traza de comunicación no es algo trivial y aunque existen herramientas para su análisis y estudio como Wireshark que nos permite ver los detalles de un paquete desglosando cada uno de sus niveles, no siempre es fácil entender la escena. Es por ello que hemos pensado en una solución que nos permita representar un escenario real y entender el contexto de todos los paquetes que forma parte de él, aprovechando además un entorno novedoso como es el de la realidad virtual.

Esta realidad virtual que genera un marco artificial, mediante el cual se hace sentir al usuario que está inmerso en un entorno real está viviendo sus mejores momentos desde la llegada del Metaverso, la idea de interconexión global a través de un mundo virtual. Su uso extendido en videojuegos y experiencias inmersivas hacen de este tipo de tecnologías una opción muy atractiva para la ejecución de mi trabajo.

La herramienta nos permitirá visualizar e interactuar con la escena en cualquier navegador o haciendo uso de periféricos especializados en realidad virtual gracias a las tecnologías de WebGL y WebXR, que detallaremos más adelante.

Para el desarrollo nos vamos a centrar en un entorno concreto como es el de NetGUI, herramienta utilizada en la Universidad Rey Juan Carlos para el aprendizaje de redes de comunicación.

Esta herramienta es capaz de solucionar la problemática expuesta al inicio del capítulo, ya que permitirá visualizar y estudiar en profundidad todas estas trazas de comunicación de un escenario real a cualquier persona que esté aprendiendo sobre protocolos o para un perfil más técnico que quiera estudiarlo más en profundidad.

Para el desarrollo del proyecto nos hemos basado en la tecnología A-Frame que es un software especializado en la construcción de escenas pensadas para su visualización en entor-

nos virtuales.

1.1 Objetivos del proyecto

1.1.1 Objetivo general

Mi Trabajo Fin de Grado consiste en crear una herramienta de visualización de trazas de comunicación en realidad virtual, concretamente se va a trabajar en trazas de comunicación de NetGUI.

1.1.2 Objetivos específicos

En este apartado entraremos más en detalle de cuáles han sido los objetivos específicos en los que se desglosa el proyecto.

- Se deberá poder ejecutar y probar la escena a partir de una url única.
- Se permitirá la lectura de archivos en formato JSON para la extracción de datos.
- El programa principal se desarrollará haciendo uso de la tecnología A-Frame, adecuada para el manejo de escenas en realidad virtual.
- Se deberá poder replicar la topología de un escenario de NetGUI, definiendo una entidad en 3D para cada uno de los nodos y representando las conexiones entre ellos.
- Las fuentes de datos de donde extraeremos las capturas de comunicación que pasaremos al programa principal serán NetGUI y Wireshark.
- La herramienta debe permitir analizar trazas de varias capturas, ordenando los paquetes por orden de aparición y filtrando duplicados.
- Se deberá poder visualizar el tránsito de paquetes entre los nodos de la escena. Para ello replicaremos el comportamiento de los paquetes de la captura en el escenario de realidad virtual.
- La escena permitirá animar las trazas.
- Podremos movernos libremente por la escena y permitir la interacción con los paquetes que la forman.
- Se deberá poder entender la estructura de cualquier paquete en todos sus niveles.

- Dotaremos al escenario una visión realista.
- Exista una documentación que describa todo el proceso de uso y desarrollo de la aplicación.

1.2 Planificación temporal

El desarrollo de este trabajo de fin de grado me ha llevado alrededor de 8 meses, empezando en el mes de noviembre de 2021 y dándolo por finalizado en junio de 2022, atendiendo a la siguiente división de etapas:

- Meses de noviembre y diciembre de 2021: Etapa de aprendizaje y familiarización con las tecnologías utilizadas y entornos virtuales.
- Meses de enero y febrero de 2022: Etapa de realización de los primeros ejercicios para ir cogiendo soltura con los lenguajes aplicados y para entender los intrínsecos de A-Frame.
- Meses de marzo, abril y mayo de 2022: Implementación de la herramienta.
- Mes de junio de 2022: Redacción de la memoria, revisión de código y aplicación de mejoras o correcciones.

1.3 Materiales

He preparado todos los recursos referentes a este proyecto en una página de GitHub a la que se puede acceder públicamente mediante el siguiente enlace <https://nachocru.github.io/TFG/MainPage/>

1.4 Estructura de la memoria

En este apartado se detalla en qué consistirá cada una de las secciones de la memoria:

- Capítulo 1: Introducción.

En esta capítulo se hace una breve introducción al proyecto a desarrollar, se exponen los objetivos tanto generales como específicos del proyecto, se presenta la planificación del mismo y por último se explica cuál es la estructura de la memoria.

- Capítulo 2: Estado del arte.

En esta sección se presentan cuáles han sido las tecnologías utilizadas durante el desarrollo del proyecto, cuáles son sus principales características y el motivo por el que se han seleccionado para el desarrollo de este proyecto.

- Capítulo 3: Diseño e implementación.

En esta sección entraremos más en detalle de cuál ha sido el recorrido del proyecto, cómo ha sido el aprendizaje de las tecnologías y cómo hemos implementado cada uno de los prototipos en diferentes etapas basándonos en una metodología SCRUM.

- Capítulo 4: Resultados.

En este capítulo describimos un manual de usuario a tres niveles diferentes para saber cómo probar la herramienta y cuál es el proceso completo para poder replicar otras escenas de NetGUI. Además incluimos los detalles de implementación con la explicación de todo el código y estructuración de los componentes.

- Capítulo 5: Conclusiones y trabajos futuros.

Por último, en este apartado revisaremos la consecución de los objetivos propuestos, estableceremos los diferentes conocimientos adquiridos durante el Grado Y revisaremos qué proyectos futuros se podrían llevar a cabo para mejorar este trabajo.

Capítulo 2

Estado del arte

Este proyecto tiene un alto contenido tecnológico, y para ello se han utilizado diferentes herramientas que han contribuido en el desarrollo del mismo.

Con algunas de estas tecnologías he podido trabajar anteriormente en asignaturas de la carrera, que sentaron los cimientos para la realización de este proyecto. Sin embargo otras las he visto por primera vez para el desarrollo de este trabajo y han sido especialmente útiles para mi formación personal de cara a un futuro laboral.

Debido a que el propósito del trabajo es poder visualizar trazas en realidad virtual, hemos utilizado el lenguaje de programación A-Frame que se fundamenta en HTML5 y Javascript ya que está especializado para trabajar con este tipo de entornos. Por ello son algunas de las tecnologías sobre las que hablaremos en este apartado.

A continuación voy a presentaros algunas de las herramientas más importantes que se han utilizado para la realización de este trabajo, exponiendo una descripción sencilla de cada una y algún ejemplo o caso práctico.

En primer lugar hablaremos de las tecnologías que hemos seleccionado para la extracción de datos: NetGUI y Wireshark. Después hablaremos de los estándares que nos permiten desarrollar un programa en realidad virtual, como son Three.js, WebXR y WebGL. Acto seguido trataremos cada uno de los lenguajes en los que se ha escrito la herramienta, como son HTML5, A-Frame principalmente, Javascript y en menor medida Python. Por último hablaremos de otras tecnologías complementarias que nos han servido de ayuda para la realización del trabajo como son JSON para estructurar datos, Visual Studio Code para el desarrollo del código y LaTeX y Overleaf para la escritura de la memoria.

2.1 NetGUI

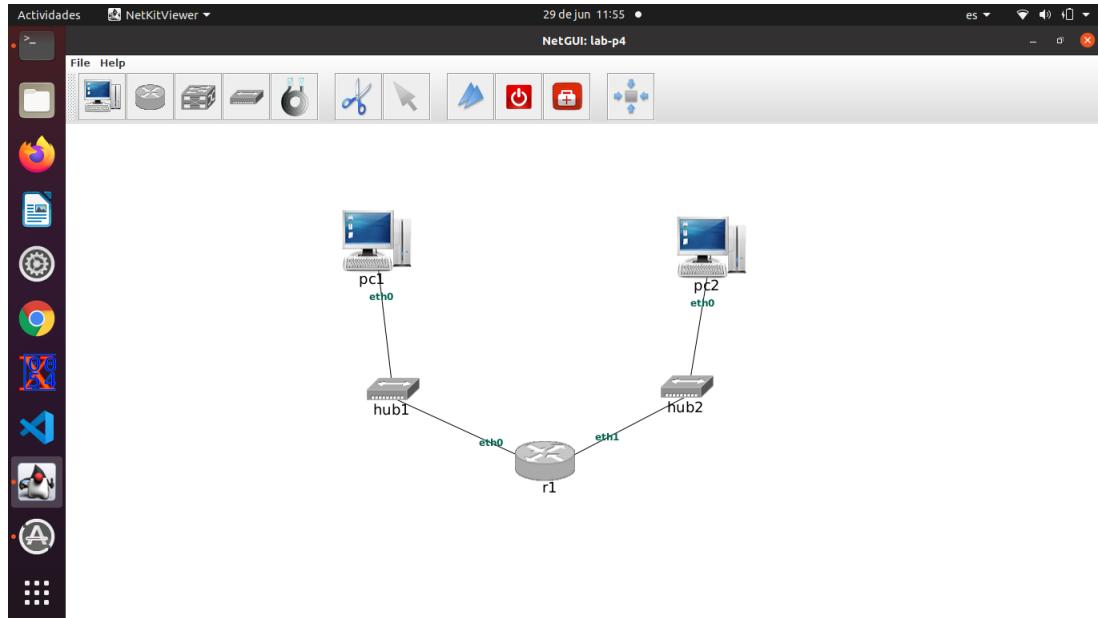
NetGUI es una herramienta usada dentro de la Universidad Rey Juan Carlos construida sobre el software Netkit y cuyo objetivo es la creación de una interfaz de comunicación de redes, así como el almacenamiento de estos escenarios, interconexión de elementos y simulaciones de comunicaciones.

Es un software libre que se puede descargar en <https://mobiquo.gsyc.urjc.es/netgui/> [14].

Netkit es un entorno software que nos permite experimentar con redes de ordenadores en entornos virtuales. Permite arrancar nodos que ejecuten el kernel y las aplicaciones de GNU/Linux. Usa máquinas virtuales UML (User-mode Linux) que es un kernel de Linux que puede ser arrancado como un proceso de usuario en una máquina que lo tenga instalado.

La interfaz de NetGUI es sencilla y se pueden hacer cosas como cargar o crear un escenario con máquinas virtuales, hubs y routers e interconectarlos entre sí. Todas las máquinas son operables con un terminal Linux integrado desde el que se pueden indicar funciones para interactuar con el resto del escenario.

Figura 2.1: Interfaz gráfica de NetGUI



Descargando el software necesario podemos arrancar el programa con la orden netgui.sh, además de limpiar restos de escenarios anteriores con clean-netgui.sh.

NetGUI es uno de los pilares fundamentales de nuestro proyecto, porque es el programa

donde generaremos los escenarios de comunicación, configuraremos cada una de las máquinas para establecer rutas o requisitos entre ellos y activaremos una comunicación de trazas para su intercambio de datos. Después todo este comportamiento será lo que repliquemos en nuestro escenario A-Frame, por lo que si no trabajamos bien con la base del proyecto que es NetGUI, no obtendremos un buen resultado final en nuestro entorno de realidad virtual,

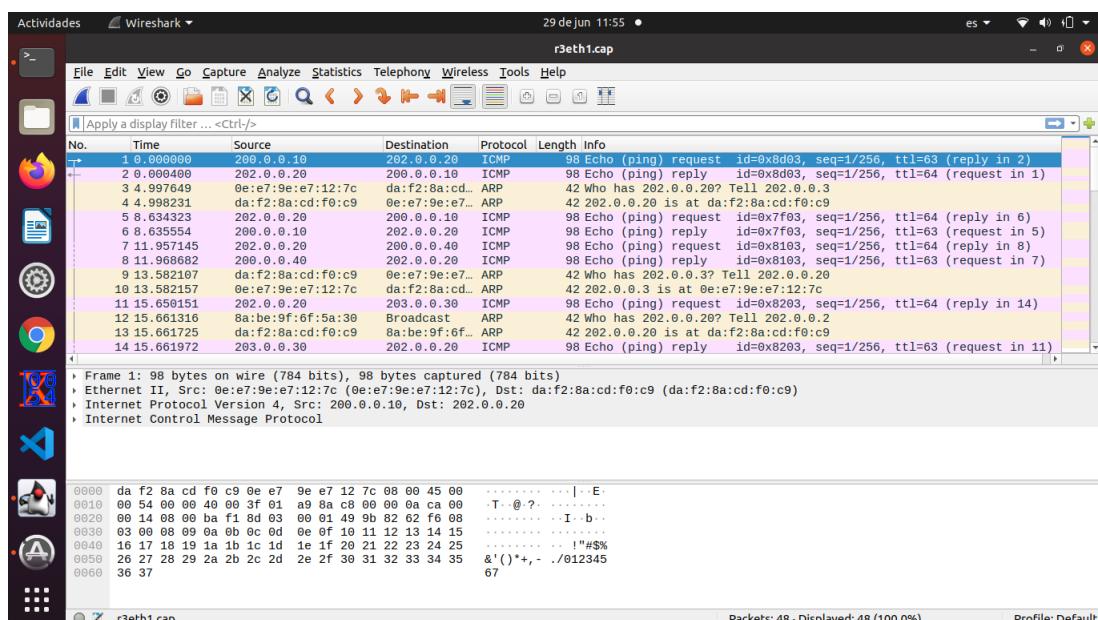
2.2 Wireshark

Dentro de nuestro escenario de pruebas realizaremos capturas de comunicación para que sean analizadas en Wireshark ¹ [22].

Wireshark es un analizador de protocolos de red que nos permite observar qué pasa en las trazas de comunicación que se transportan en la red. Este software se desarrolló gracias a contribuciones voluntarios de expertos en redes desde el 1998.

Está mantenido bajo la licencia GLP y es muy robusto tanto en modo promiscuo como no promiscuo. Permite la captura o lectura de datos almacenados en un archivo y está basado en la librería pcap. Tiene una interfaz muy flexible, gran capacidad para filtrar y admite el formato estándar de archivos tcpdump. Se puede ejecutar en más de 20 plataformas, es compatible con más de 480 protocolos y puede traducir protocolos TCP IP entre otras funcionalidades.

Figura 2.2: Interfaz gráfica de Wireshark



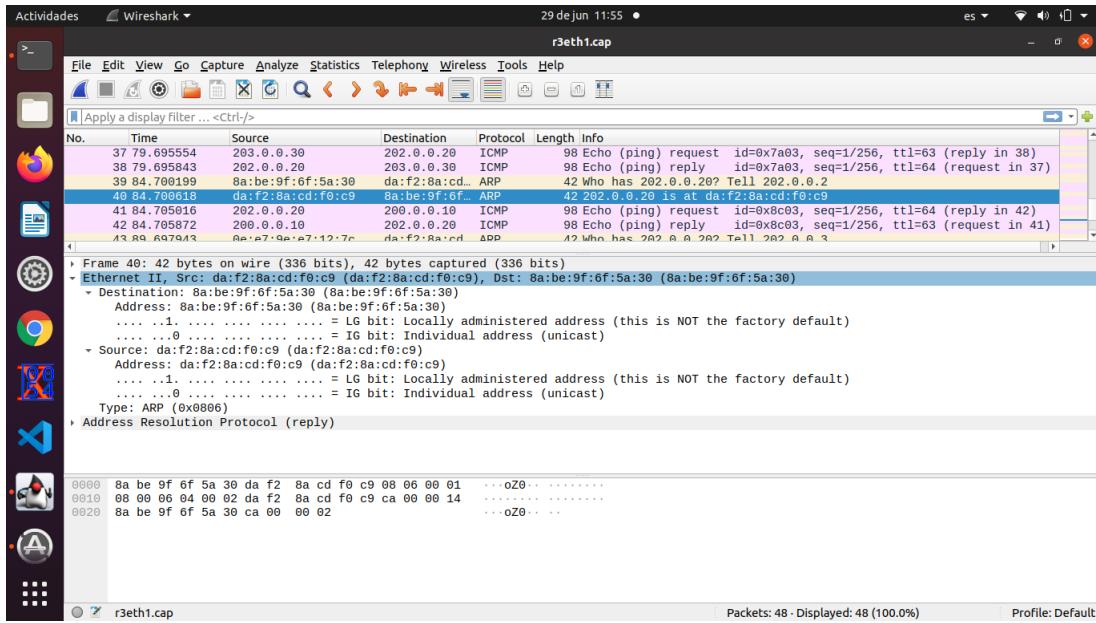
¹<https://www.wireshark.org/>

Su uso está muy generalizado, ya que nos permite, entre otras cosas, las siguientes:

- Captura de tramas directamente desde la web.
- Mostrar y filtrar dichas tramas.
- Captura de tramas en remoto.
- Modificar estas tramas y transmitirlas por la red.
- Estudiar y analizar las diferentes tramas para sacar estadísticas.
- Exportar las capturas en diferentes formatos.
- Realizar seguimiento de parámetros, flujos de datos o protocolos.

Wireshark ofrece un interfaz gráfico para desplegar y visualizar la información de cada paquete distinguiendo los diferentes niveles de comunicación (Ethernet, IP, UDP ...).

Figura 2.3: Desglose de información de niveles en Wireshark



El uso de esta aplicación para nuestro proyecto ha sido fundamental para el estudio de los diferentes niveles que transportan los paquetes. Además hemos podido exportar dichas capturas en formato JSON para que nuestro programa principal de A-Frame sea capaz de leerlas e interpretarlas.

2.3 Three.js

Three.js² [6] es una librería de Javascript que se orienta a la creación y el diseño de gráficos en 3D desde un navegador web. Esta librería se puede usar junto con el elemento canvas de HTML5, SVG o WebGL, siendo esta última para la que se desarrollan las animaciones más importantes. El contenido de esta librería se aloja en un repositorio público de GitHub y podemos acceder a su documentación oficial desde el siguiente enlace <https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene> [4].

Three.js nos ofrece muchas facilidades para la creación de escenas en 3D mediante la inclusión de elementos gráficos que permiten una alta interactividad con el usuario. El elemento de la cámara define el punto de vista del usuario, un mesh (o malla) detalla cualquier colección de vértices, bordes y caras que nos definen objetos poliédricos y la luz determina la iluminación de la escena, entre otras características.

Para crear una escena Javascript, debemos añadir el siguiente identificador en nuestro código HTML para poder hacer uso de este lenguaje:

```
<script src="js/three.js"></script>
```

Ahora mediante código Javascript podemos crear una escena a la que iremos añadiendo cada uno de los elementos que queremos que formen parte de ella. Es importante definir la cámara que nos indicará el punto de vista del usuario y renderizar la escena haciendo uso de WebGL:

```
const scene = new THREE.Scene();
const camera = new THREE.PerspectiveCamera( 75,
window.innerWidth / window.innerHeight, 0.1, 1000 );

const renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );
```

Una vez tenemos la escena creada podemos añadir una gran variedad de elementos a ella definiendo por un lado su geometría y por otro lado su material:

```
const geometry = new THREE.BoxGeometry( 1, 1, 1 );
const material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
const cube = new THREE.Mesh( geometry, material );
scene.add( cube );
```

²<https://threejs.org/>

Esta simple dinámica nos ofrece muchas posibilidades a la hora de la creación de gráficos y escenas en 3D, ya que no sólo podemos colocar elementos en la escena, sino también animarlos y detectar eventos por parte del usuario como clicks o entradas de teclado que hacen de Three.js una opción perfecta para la creación de escenas interactivas en 3D.

Three.js nos permite generar escenas tan alucinantes como estas.

Figura 2.4: Ejemplo de escena construida con Three.js

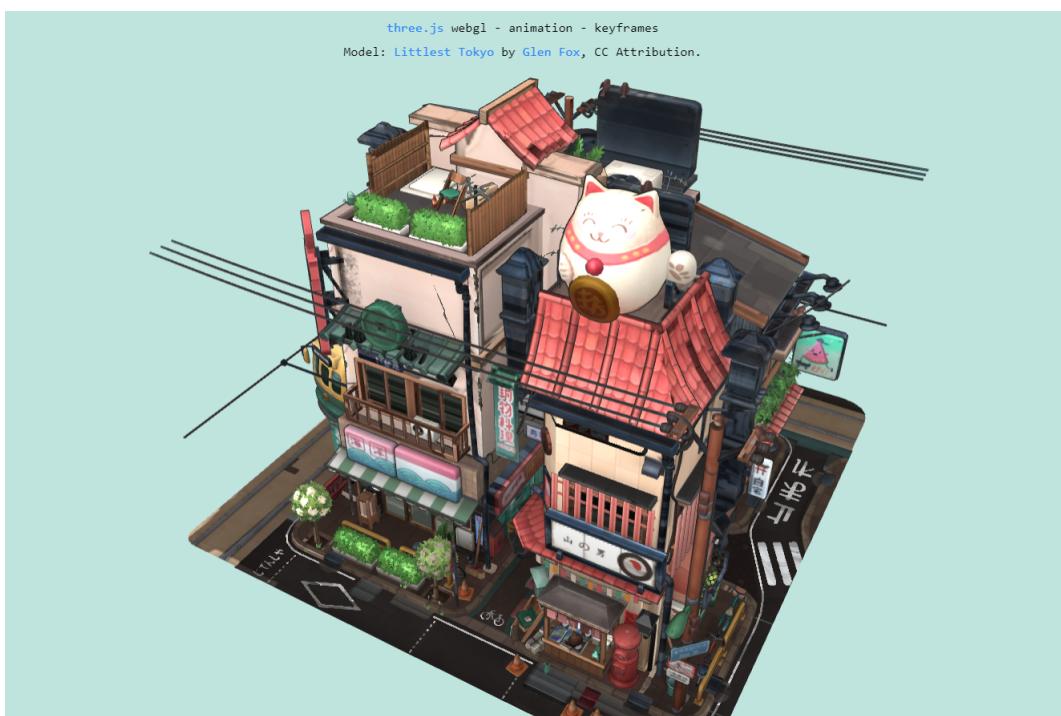
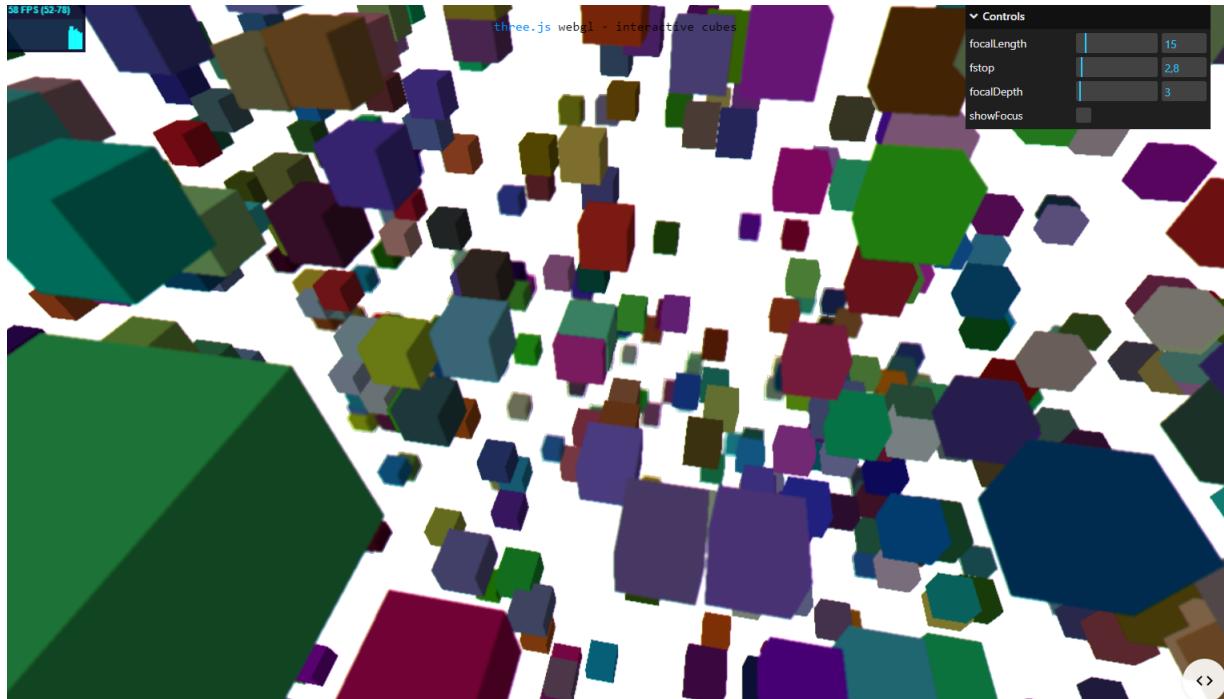


Figura 2.5: Ejemplo de escena construida con Three.js



De cara a nuestro proyecto hemos seleccionado esta tecnología por su facilidad y dinamismo a la hora de crear escenas en 3D que ayudarán en la sensación de inmersión por parte del usuario final.

2.4 WebXR

WebXR³ [20]⁴ [21] es un grupo de estándares que se usan para la representación de gráficos en 3D en dispositivos especializados de realidad virtual o realidad aumentada.

WebXR es el encargado de gestionar la renderización de la escena. Mediante un sistema de percepción del movimiento en los distintos periféricos donde se visualizan las vistas de realidad virtual WebXR es capaz de detectar qué parte de la escena hay que renderizar y mostrar por pantalla, ofreciendo al usuario una sensación de inmersión en un entorno envolvente en 3D. Además, no solo la imagen va acorde con los distintos movimientos del usuario, sino que WebXR controla la salida de audio de los auriculares en función de nuestra posición en la escena.

La API de WebXR nos proporciona las siguientes características:

³<https://immersiveweb.dev/>

⁴<https://www.w3.org/TR/webxr/>

- Localizar e identificar dispositivos de salida de realidad virtual o aumentada.
- Traducir los vectores de movimiento (tanto dirección como magnitud) de los dispositivos de entrada y sensores en vectores que se utilizan en la aplicación.
- Se encarga de la renderización de escenas 3D en dispositivos a una adecuada velocidad de fotogramas.

2.5 WebGL

WebGL (Web Graphics Library)⁵ [19] es una librería estándar que define una API desarrollada en Javascript y pensada para renderizar gráficos 3D sobre navegadores web. Sobre cualquier dispositivo que soporte OpenGL se puede hacer uso de este estándar sin necesidad de plug-ins externos. Se pueden combinar elementos estándar de HTML con otros de WebGL para formar el lienzo de la escena o "canvas". WebGL está desarrollado y gestionado por el consorcio Khronos Group.

Para nuestro proyecto haremos uso de esta tecnología para optimizar el uso de la visualización de gráficos 3D desde cualquier tipo de navegador, ofreciendo la posibilidad de crear escenas hiperrealistas como la que mostramos en la imagen:

Figura 2.6: Escena hiperrealista con WebGL



2.6 HTML5

A-Frame renderiza su escena sobre HTML5⁶ [10] [9] o HyperText Markup Language, que se trata de la quinta versión del lenguaje básico de la World Wide Web, es decir, el lenguaje

⁵https://www.khronos.org/webgl/wiki/Main_Page

⁶<https://devdocs.io/html/>

estándar que se usa como referencia en el visionado de sitios web.

Las características de este tipo de lenguaje, como su propio nombre indica son:

- Hypertext: El pilar de las páginas web tal y como lo conocemos, se trata de recursos interconectados con otros recursos, ya sean páginas, textos, archivos ...
- Markup: HTML funciona mediante un sistema de etiquetado que identifica diferentes secciones dentro de la página, tales como párrafos, bloques, listas ...
- Lenguaje: Forma un lenguaje ya que tiene sus propias normas y convenciones que nos sirve para definir la forma y estructura de la web.

Dentro del contenido HTML podemos definir etiquetas referentes a componentes A-Frame, como veíamos en la primera sección, tales como `<a-entity>`, la propia escena `<a-scene>` y muchos otros.

Figura 2.7: Ejemplo tipográfico en HTML



The screenshot shows a browser-based code editor interface. On the left, the code area contains the following HTML:

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Heading</h1>
<p>My first paragraph.</p>
</body>
</html>
```

On the right, the preview area displays the rendered content:

My First Heading

My first paragraph.

At the top of the interface, there are various icons and buttons, including a 'Run' button. In the top right corner, it says 'Result Size: 753 x 528' and 'Get your own website'.

Además de su definición como lenguaje propiamente dicho podemos definir HTML como el conjunto de tecnologías que se desarrollan en torno a la nueva versión de este lenguaje HTML5, entre las que destacan HTML, CSS3, Javascript o XML.

HTML5 tiene un extenso número de aplicaciones y usos, y algunas de las características de su lenguaje son las siguientes.

- Semántica: Mediante el uso de etiquetas podemos diferenciar cuál es el contenido principal.
- Conectividad: Comunicación con el servidor de diversas formas, como puede ser mediante WebSockets, WebRTC ...

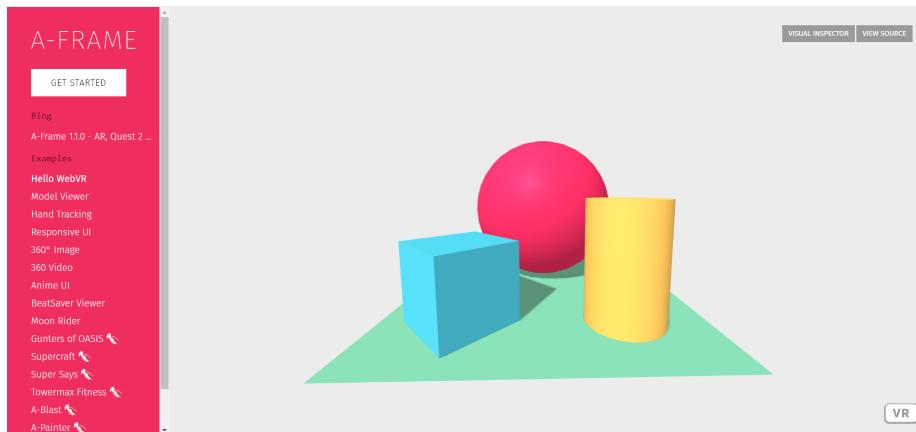
- Soporte de contenido multimedia, permitiendo la reproducción de audio y vídeo sin necesidad de plug-ins externos.
- Amplia gama de efectos y gráficos en 2D y 3D.
- APIs externas para gran variedad de funcionalidades: Acceso a la cámara, ubicación, detección de eventos ...
- Compatibilidad con CSS para la gestión de estilos.

Para nuestros proyectos hacemos uso de HTML5 porque es el lenguaje sobre el que se basa nuestra aplicación A-Frame y el que nos permite servir escenas en navegadores web.

2.7 A-Frame

A-Frame⁷ [1] es un framework de código abierto de Three.js basado en HTML que se utiliza para construir escenas y experiencias en realidad virtual. Su núcleo principal consta de una arquitectura ECS (Entity Component System) donde cada elemento de la escena representa una entidad que puede a su vez tener componentes.

Figura 2.8: Página de documentación de A-Frame



Además A-Frame tiene una alta compatibilidad con la mayoría de tecnologías de realidad virtual y aumentada que se encuentran actualmente en el mercado: Vive, Rift, Windows Mixed Reality, Daydream, GearVR, Cardboard, Oculus Go, etc.

En cuanto a rendimiento, el hecho de que los elementos de la escena se carguen en memoria ya que A-Frame se basa en la manipulación del DOM hace que podamos visualizar escenas en realidad virtual a 90fps e interactuar con ellas.

⁷<https://aframe.io/>

A-Frame también ofrece una alta escalabilidad y ha sido probado y usado por enormes compañías tales como Google, Disney, Samsung, Toyota, Ford, Chevrolet, etc.

Los principales elementos de los que consta un proyecto escrito con A-Frame son los siguientes:

- Entidades: Son cada uno de los elementos que aparecen en la escena, tales como figuras, objetos ... Para insertar en la escena una de estas entidades lo podremos hacer escribiendo directamente en el archivo HTML:

```
<a-entity geometry="primitive: box" material="color: red">
```

A estas entidades se las puede dotar de ciertas características como comportamiento, apariencia, funcionalidad ...

- Componentes: Dentro de la arquitectura ECS un componente es un pedazo de código que puede ser reutilizado por cualquier entidad de nuestro proyecto y sirven generalmente para dar apariencia, función o comportamiento al elemento que lo referencia.

```
<a-entity position="1 2 3"></a-entity>
```

En este ejemplo le estaríamos dotando al elemento de un atributo de posición, que nos indica dónde estará ubicada la entidad en la escena.

- Sistemas: Siguiendo con la arquitectura ECS un sistema proporciona un marco global a los componentes para interactuar con la escena. Un sistema se declara de forma parecida a un componente mediante:

```
AFRAME.registerSystem('my-component', {
  schema: {}, // System schema. Parses into `this.data`.
  init: function () {
    // Called on scene initialization.
  },
  // Other handlers and methods.
});
```

- Escena: La escena es un elemento referenciado con la etiqueta <a-scene> y es el objeto raíz global, donde se alojarán todos los demás elementos de la escena.
- Mixins: Son un tipo de elementos pensados para reutilizar propiedades de componentes comunes, se definen con la etiqueta <a-mixin> y se sitúan dentro de los assets de la escena:

```

<a-scene>
  <a-assets>
    <a-mixin id="red" material="color: red"></a-mixin>
    <a-mixin id="blue" material="color: blue"></a-mixin>
    <a-mixin id="cube" geometry="primitive: box"></a-mixin>
  </a-assets>

  <a-entity mixin="red cube"></a-entity>
  <a-entity mixin="blue cube"></a-entity>
</a-scene>

```

- Otros activos: Estos son elementos especiales que nos permiten manejar otro tipo de activos en nuestra escena y son el audio, el vídeo, las imágenes y los ítems, que nos permiten dotar a los objetos de modelos 3D. Aquí vemos un ejemplo de como declarar y usar este tipo de elementos:

```

<a-scene>
  <!-- Asset management system. -->
  <a-assets>
    <a-asset-item id="horse-obj" src="horse.obj"></a-asset-item>
    <a-asset-item id="horse-mtl" src="horse.mtl"></a-asset-item>
    <a-mixin id="giant" scale="5 5 5"></a-mixin>
    <audio id="neigh" src="neigh.mp3"></audio>
    
    <video id="kentucky-derby" src="derby.mp4"></video>
  </a-assets>

  <!-- Scene. -->
  <a-plane src="#advertisement"></a-plane>
  <a-sound src="#neigh"></a-sound>
  <a-entity geometry="primitive: plane" material="src: #kentucky-derby">
  </a-entity>
  <a-entity mixin="giant" obj-model="obj: #horse-obj; mtl: #horse-mtl">
  </a-entity>
</a-scene>

```

A-Frame es la tecnología principal de nuestro proyecto, ya que está orientada la visualización de escenas en realidad virtual y al estar apoyado sobre HTML y Javascript permite una interactividad con el usuario que define los conceptos clave de nuestro trabajo.

2.8 Javascript

Toda la definición de la lógica del programa, tanto componentes, funciones e interacciones entre ambos se hace con lenguaje de programación ligero Javascript⁸ [11] [7], que se compila a tiempo real y usa funciones de primera clase, es decir, que dichas funciones son tratadas de la misma manera que cualquier otra variable, pudiendo pasar una función como argumento a otra función, puede ser devuelta otra función como resultado o incluso asignar una función a una variable.

Javascript es altamente usado en entornos web como scripting (o secuencia de comandos) y en muchos entornos fuera del navegador, entre los que destacan Node.js, Apache CouchDB y Adobe Acrobat. Fue desarrollado por Brendan Eich de Netscape originalmente a finales de siglo pasado y sus autores lo propusieron para que fuera adoptado por ECMA (European Computer Manufacturers' Association) donde finalmente se incluyó en junio de 1997 adoptando el nombre de ECMAScript y poco después se adoptó también como estándar ISO.

Javascript nos permite hacer cosas como:

- Almacenar valores dentro de variables para usarlos en cualquier parte de nuestro programa.
- Tratamiento del texto, para hacer operaciones como unir cadenas de caracteres o strings.
- Detectar eventos que pueda realizar el usuario (como clicks en algún elemento de la escena) y provocar alguna reacción ante ellos.
- Conectar con distintas APIs (interfaces de programación de aplicaciones) que son un conjunto de herramientas, definiciones y protocolos que sirven para integrar servicios de aplicaciones y software externos. Generalmente podemos distinguir 2 tipos de APIs de las que se hacen uso en programas Javascript:
 - APIs del navegador: Integradas en el propio navegador web y pueden exponer datos que se encuentran en la misma o realizar otro tipo de tareas complejas. Algunas de las APIs de este tipo más comunes son la API del DOM (que nos permite manejar el HTML y el CSS), la API de geolocalización (que maneja información geográfica), las APIs de Canvas y WebGL (para crear gráficos y animaciones en 2D y 3D, como es el caso de A-Frame) o las APIs de audio y vídeo (que permiten explorar el mundo de la multimedia).
 - APIs de terceros: No están integrada en el navegador sino que hay que obtener la información de algún otro lugar de la web. Algunas interesantes pudieran ser

⁸<https://developer.mozilla.org/es/docs/Web/JavaScript>

la API de Twitter (que nos permite recopilar información de esta aplicación) o la API de Google Maps (para interactuar con mapas).

La mayor parte de mi proyecto está escrito sobre Javascript, donde definiremos cada uno de los componentes que mostraremos en la escena y modificaremos el DOM de forma dinámica. Su versatilidad como lenguaje y compatibilidad con A-Frame y HTML han sido detonantes para la selección de este lenguaje en el desarrollo.

2.9 Python

En cierta parte del proyecto hacemos uso del lenguaje de programación de alto nivel Python⁹ [16] [8], que es un lenguaje interpretado ya que no necesita compilación, sino que se ejecuta directamente con un interpretador a diferencia de otros lenguajes como Java que sí hace uso de un compilador.

La principal característica de Python es la legibilidad de código, para acercar al usuario a un lenguaje más parecido al humano. Su paradigma es orientado a objetos, es decir, la escena se construye en torno a objetos que tienen información en campos y código en forma de métodos. Estos objetos son capaces de interactuar entre sí. Es administrado por Python Software Foundation y tiene licencia de código abierto (Python Software Foundation License).

Se creó a finales de los 80 por Guido van Rossum y hoy en día es uno de los lenguajes más utilizados por la simplicidad de su sintaxis. Dentro del lenguaje, podemos encontrar diferentes elementos como funciones, variables (a las que se les puede dotar de un tipo de dato), condicionales, bucles, listas y tuplas, diccionarios, sentencias ...

Para nuestro proyecto, hacemos uso de este lenguaje para leer todas las capturas en formato JSON que se han realizado en los nodos del escenario, las procesamos y las unificamos en una nueva captura. El motivo de haber hecho esta parte en este lenguaje es la facilidad que tiene para la lectura y escritura de ficheros y por el uso de librerías internas que nos facilitan mucho trabajo para diferentes funciones y métodos.

⁹<https://docs.python.org/3/>

2.10 JSON

JavaScript Object Notation¹⁰ [12] es un sencillo formato de texto para intercambio de datos. Aunque tiene una alta similitud con la notación de objetos en Javascript se le considera un lenguaje independiente debido a su extendido uso. La principal ventaja de este tipo de formato es la sencillez de leer y escribir información en él, ya que al ser un lenguaje organizado en listas, se puede recorrer fácilmente obteniendo su información. Este es un ejemplo de un fragmento de código escrito en lenguaje JSON:

```
{
  "departamento":8,
  "nombrededepto":"Ventas",
  "director": "Juan Rodríguez",
  "empleados": [
    {
      "nombre": "Pedro",
      "apellido": "Fernández"
    },
    {
      "nombre": "Jacinto",
      "apellido": "Benavente"
    }
  ]
}
```

2.11 Visual Studio Code

Visual Studio Code¹¹ [18] es un editor de texto optimizado para el desarrollo de lenguajes de programación. Se fundamenta en ofrecer al desarrollador las herramientas necesarias para construir un flujo de código óptimo así como para debuggear fragmentos, correr tareas o llevar un control de versiones adecuando para nuestra aplicación.

Una característica fundamental de Visual Studio Code es que nos permite instalar extensiones que faciliten el desarrollo del código. Dentro de estas extensiones tenemos una gran variedad de utilidades como puede ser una terminal integrada dentro del propio programa (y así poder llevar un mejor control de los logs y ejecuciones de nuestra aplicación), un código de estilos según el lenguaje en el que estemos desarrollando, un corrector de código limpio ...

Hemos hecho uso de este editor por la versatilidad que ofrece en cuanto a posibilidades

¹⁰<https://www.json.org/json-es.html>

¹¹<https://code.visualstudio.com/>

de extensiones (terminal integrada dentro del editor, poder previsualizar en un entorno web el código que estamos desarrollando o añadir una detección automática de errores de escritura de código, entre otros) y por estar previamente familiarizado con este editor por haber trabajado con él en asignaturas de la carrera.

2.12 LaTeX

Para la redacción de esta memoria he utilizado LaTeX ¹² [13], que es un lenguaje de marcado utilizado generalmente para la redacción de artículos de alta calidad tipográfica, como artículos académicos, científicos o técnicos. LaTeX se enmarca dentro de una licencia pública libre lo que ha permitido que numerosos usuarios mejoren y amplíen las capacidades que este lenguaje ofrece.

LaTeX se fundamenta en órdenes creadas a partir de comandos Tex, que es un lenguaje de bajo nivel. Hace uso de una arquitectura modular, con el compilador como núcleo central y una serie de paquetes a los cuales se invocan solo en caso de ser necesarios.

Algunas de las numerosas cualidades que nos ofrece este tipo de lenguaje son, una estructuración del documento en diferentes bloques y secciones, ofreciendo una paginación e índice automáticos, diferentes tipologías de letra, elementos visuales como guiones, títulos, puntos suspensivos, y otras muchas ventajas que hacen de este lenguaje una muy buena opción para escribir un artículo formal.

El motivo de haber elegido este lenguaje para la escritura de mi memoria es porque nos facilita la redacción de textos formales y no tenemos que preocuparnos por los estilos para poder centrarnos en el contenido, que es lo verdaderamente importante de estos textos.

2.13 Overleaf

Overleaf ¹³ [15] es un editor online colaborativo usado principalmente para desarrollar documentos escritos en LaTeX y que no requiere de ninguna instalación local, lo que hace muy sencillo el desarrollo de documentos por parte de los usuarios. Fue creado por dos matemáticos: John Hammersley y John Lees-Miller en el 2014.

Aparte de para el desarrollo de esta memoria, Overleaf ha ofrecido servicio a la redacción de artículos y publicaciones, entre los que destacan algunos para las revistas Nature, Science o documentos de Red Hat's opensource.com.

¹²<https://www.latex-project.org/help/documentation/>

¹³<https://es.overleaf.com/learn>

He hecho uso de este editor compatible con LaTeX por la facilidad de uso que tiene, ya que funciona online sin la necesidad de ningún tipo de instalación y me permite poder descargar el contenido escrito en cualquier momento en formato pdf. Además nos permite en tiempo real ver una previsualización del resultado mientras escribimos el contenido.

Figura 2.9: Interfaz gráfico de Overleaf

The screenshot shows the Overleaf LaTeX editor interface. On the left, there's a sidebar with project files like 'helloworld.png', 'html.png', 'logoURJC.png', etc., and a 'File outline' section. The main area has tabs for 'Source (legacy)', 'Rich Text', and 'Preview'. The 'Source (legacy)' tab displays LaTeX code. The 'Rich Text' tab shows a preview of the document content. The 'Preview' tab shows a screenshot of the Wireshark application. The Wireshark preview shows a network traffic capture with several frames listed in the list pane and their details in the details pane. The Overleaf interface includes a top bar with 'Review', 'Compartir', 'Submit', 'History', 'Layout', and 'Chat' buttons.

Capítulo 3

Diseño e implementación

A lo largo de esta sección explicaré cómo ha sido el periodo de aprendizaje y el desarrollo del proyecto hasta llegar al resultado final.

3.1 Metodología

Para encarar este proyecto, nos hemos basado en una metodología SCRUM, que se trata de un proceso de desarrollo orientado a la consecución de tareas de forma colaborativa. Este tipo de metodología es adecuada en proyectos en los que no queda bien definido cuál debe ser el resultado final, sino que a medida que se avanza este objetivo se puede ir redefiniendo o tomando un mayor alcance. Esto casa muy bien con la realización de mi trabajo, que pese a tener un objetivo claro desde el principio, este puede ir variando o redefiniendo según avanza el proyecto, adecuándose al tiempo y las necesidades del equipo de desarrollo.

Esta metodología distingue las siguientes fases o etapas del proyecto:

- Planificación: Product Backlog.

En esta fase se definen las principales tareas que deberemos conseguir a lo largo del desarrollo, describiendo brevemente qué aportará cada una al proyecto final.

Debido a que la metodología SCRUM es una forma de trabajar de forma incremental, no es necesario definir desde el inicio del proyecto todas las tareas, sino que se pueden ir añadiendo, eliminando o modificando según se exija con el tiempo.

- Ejecución: Sprint.

El sprint es el núcleo de la metodología SCRUM, donde se deberá llegar a un prototípico entregable del proyecto a desarrollar en un corto periodo de tiempo. Este tiempo

suele variar según la organización, pero suele comprender entre una o dos semanas, y rara vez alcanza una duración superior a los 30 días.

- Control: Burn Down.

Esta etapa se lleva acabo entre sprints, para actualizar la situación del proyecto y realizar una retrospectiva de los avances del mismo y los objetivos alcanzados.

Un equipo SCRUM suele estar dividido en los siguientes perfiles:

- Product Owner:

Es el dueño del producto y por tanto el que tiene que aportar valor al mismo, definiendo los requisitos y transmitiendo al resto del equipo los objetivos que se deben alcanzar. Es el encargado de tomar las decisiones del proyecto. Es una única persona y puede formar parte de los roles que veremos a continuación.

- SCRUM Master:

Por un lado es la persona que se encarga de gestionar el proceso SCRUM, asegurándose que se están cumpliendo las diferentes fases y facilitando al resto del equipo la consecución del trabajo. Por otro lado es el encargado de lidiar y eliminar los impedimentos, ofreciendo ayuda al equipo de desarrollo y estableciendo las pautas que dicta la metodología.

- Equipo de desarrollo:

Son las personas encargadas de la consecución de las diferentes tareas técnicas definidas en el periodo de planificación SCRUM. El equipo se auto-organiza a la hora de afrontar las diferentes tareas y habitualmente son equipos cross-funcional, es decir, con la capacidad de generar un incremento terminado de principio a fin sin la necesidad de dependencias externas.

En lo relativo a este proyecto, no hemos seguido a rajatabla esta metodología pero si nos hemos basado ligeramente en ella. En cuanto a los distintos perfiles yo he adoptado el rol del equipo de desarrollo por ser el encargado de la realización de las tareas y manejo del código técnico. Por otro lado, mi tutor Jesús María González Barahona ha adoptado los roles de Product Owner y SCRUM Master, ya que ha sido el encargado de definir y controlar la consecución de cada una de las tareas.

Basándonos en esta metodología se pueden alcanzar beneficios tan importantes como los siguientes:

- Flexibilidad: Poder redefinir los objetivos y tareas nos permite que podamos adaptarnos a las distintas necesidades del cliente o exigencias del propio proyecto.

- Mejora en la calidad de código: El hecho de obtener una versión funcional al finalizar cada sprint hace que obtengamos un código más limpio y depurado.
- Reducción en los tiempos: La manera en la que se van atacando los problemas en porciones más pequeñas y el hecho de que se vayan generando prototipos al final de cada sprint reduce riesgos de cara al proyecto final.

Dentro del proyecto podemos definir las diferentes etapas que asumiremos como sprints. En cada una de ellas hemos tenido por lo menos una reunión orientativa entre el tutor y yo en la que hemos definido las tareas a desarrollar y realizar una retrospectiva de la etapa anterior. Sin embargo en muchas etapas las reuniones han sido simplemente de seguimiento y en otras hemos tenido más de una reunión por el gran alcance definido:

- Etapa 0: Aprendizaje
- Etapa 1: Primeras implementaciones
- Etapa 2: Representación de la topología de un escenario
- Etapa 3: Lectura de paquetes y animación en la escena
- Etapa 4: Añadiendo información a los paquetes
- Etapa 5: Experiencias en realidad virtual

3.2 Etapa 0: Aprendizaje

Esta etapa está orientada principalmente a aprender el lenguaje de A-Frame, cómo funcionan cada uno de los componentes, manejadores de eventos y poder realizar algunos programas sencillos que sienten las bases de lo que usaré más adelante en la herramienta final.

Es quizás la parte más importante del proyecto, en la que me toparía por primera vez con esta nueva tecnología y sentaría los conocimientos que necesitaría para todo el resto del proyecto.

Para adquirir el conocimiento previo antes de empezar a tirar líneas de código o ejecutar demos sencillas, es fundamental navegar sobre la documentación principal de A-Frame¹ donde podremos investigar y aprender todo acerca de este lenguaje.

¹<https://aframe.io/>

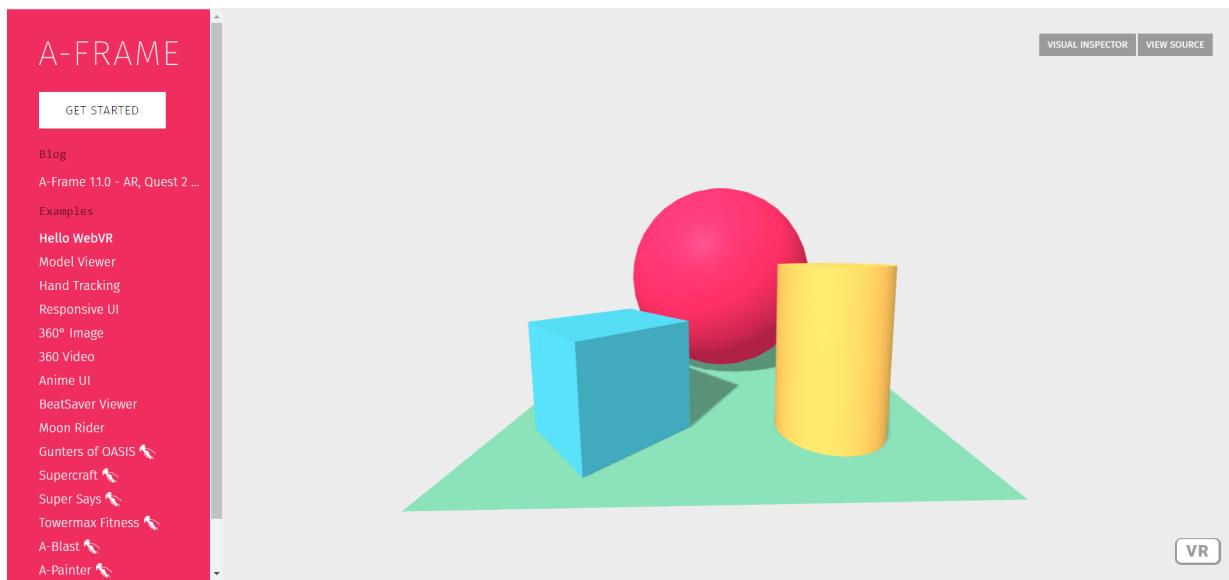
Hemos podido realizar una lectura exhaustiva de cada una de las secciones, entendiendo bien cómo funciona cada una de sus características, como se forman las diferentes escenas, cómo introducir elementos en ella, cómo establecer atributos en estos elementos o incluso la creación y reutilización de componentes.

Tras la lectura inicial, procedemos a copiar algún ejemplo sencillo en nuestra máquina local para entender de primera mano cómo funciona y se estructura un proyecto de este estilo, e incluso probando a cambiar alguna característica de los elementos o manejadores para ir cogiendo más soltura al trabajar con esta tecnología.

3.2.1 Probando los primeros componentes

Empezamos probando el ejemplo “Hola Mundo”² [3] que referencia la documentación oficial y que nos enseña algo tan simple como la construcción de una escena a la que se añaden algunos elementos como un cubo, un cilindro, una esfera o un plano.

Figura 3.1: Ejemplo Hello World de A-Frame



Variando esta escena hemos probado a introducir otro tipo de elemento y a cambiar sus atributos, para ir entendiendo la infinidad de posibilidades que nos ofrece A-Frame, entre los que podemos destacar la introducción de imágenes con textura, animaciones e interacciones con el usuario, texto e incluso audio, lo que nos da una idea de la versatilidad de esta tecnología.

²<https://aframe.io/examples/showcase/helloworld/>

3.2.2 Manejadores de eventos y cambios en la escena

Habiendo probado los ejemplos más sencillos de la documentación, hemos aumentado un poco la dificultad para probar algún ejemplo más elaborado antes de iniciar nuestras propias implementaciones.

En primer lugar probaremos una demo para construir una galería de imágenes en 360°³ [5]. En esta demo nos situaremos ante un selector que, mediante carteles con texto, nos permitirá cambiar el aspecto del entorno que nos rodea. Con esta demo tenemos un primer acercamiento al manejo de eventos e interacciones por parte del usuario, ya que la escena cambiará en función del evento click que se capture. Además esta demo nos será muy útil en un futuro porque sentará las bases de una pantalla inicial en nuestro proyecto final que nos permitirá seleccionar entre un modo inmersivo y un modo navegador.

Figura 3.2: Ejemplo galería de imágenes en A-frame: Imagen 1



³<https://aframe.io/docs/1.3.0/guides/building-a-360-image-gallery.html>

Figura 3.3: Ejemplo galería de imágenes en A-frame: Imagen 2

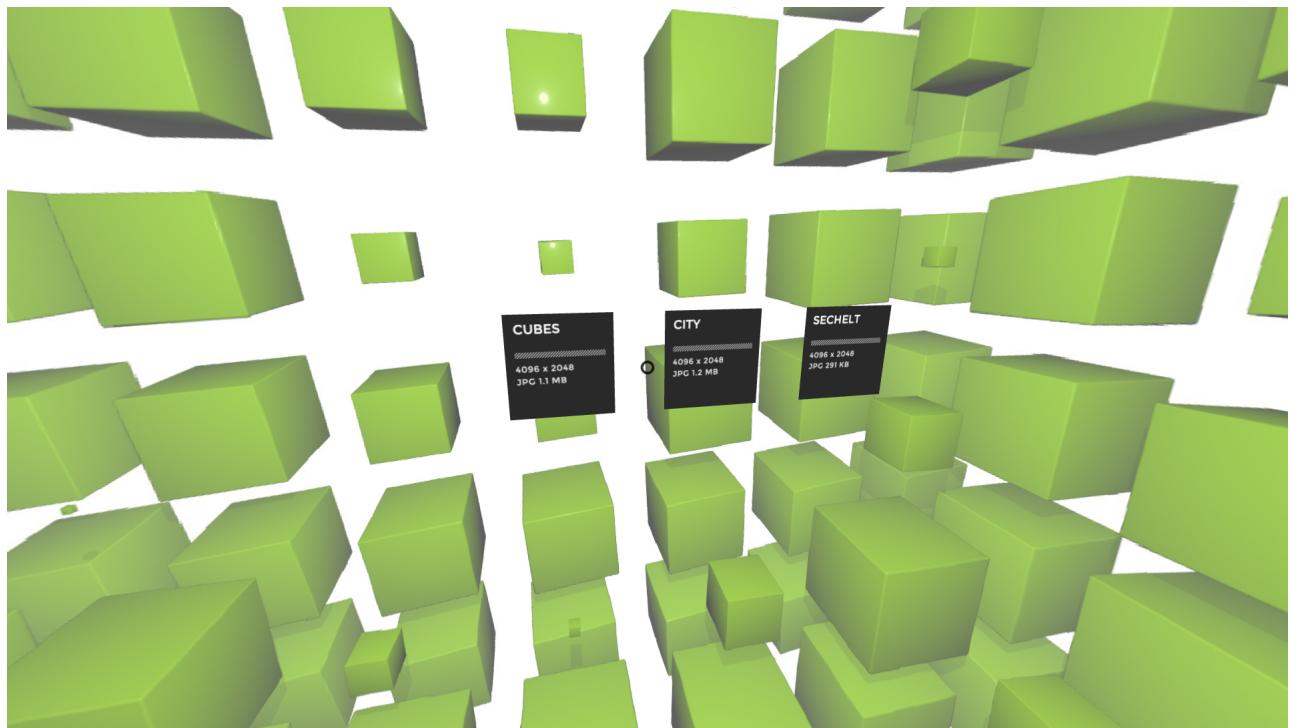
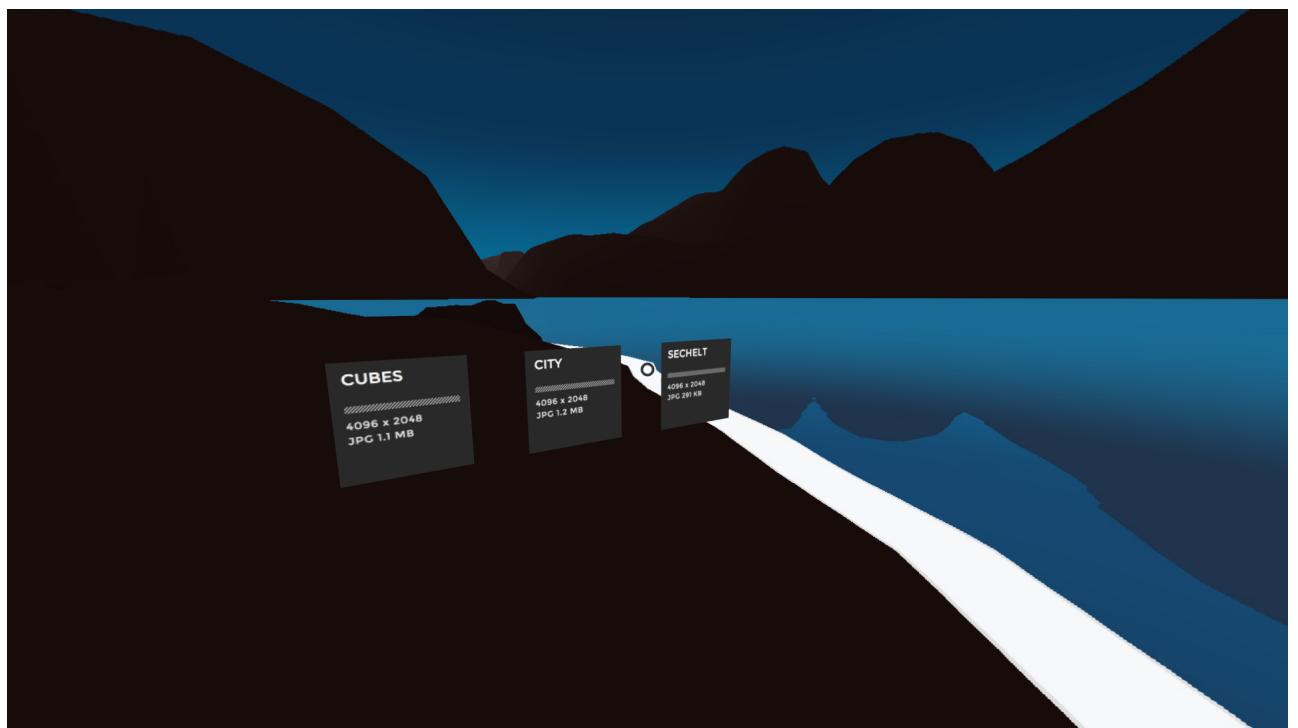


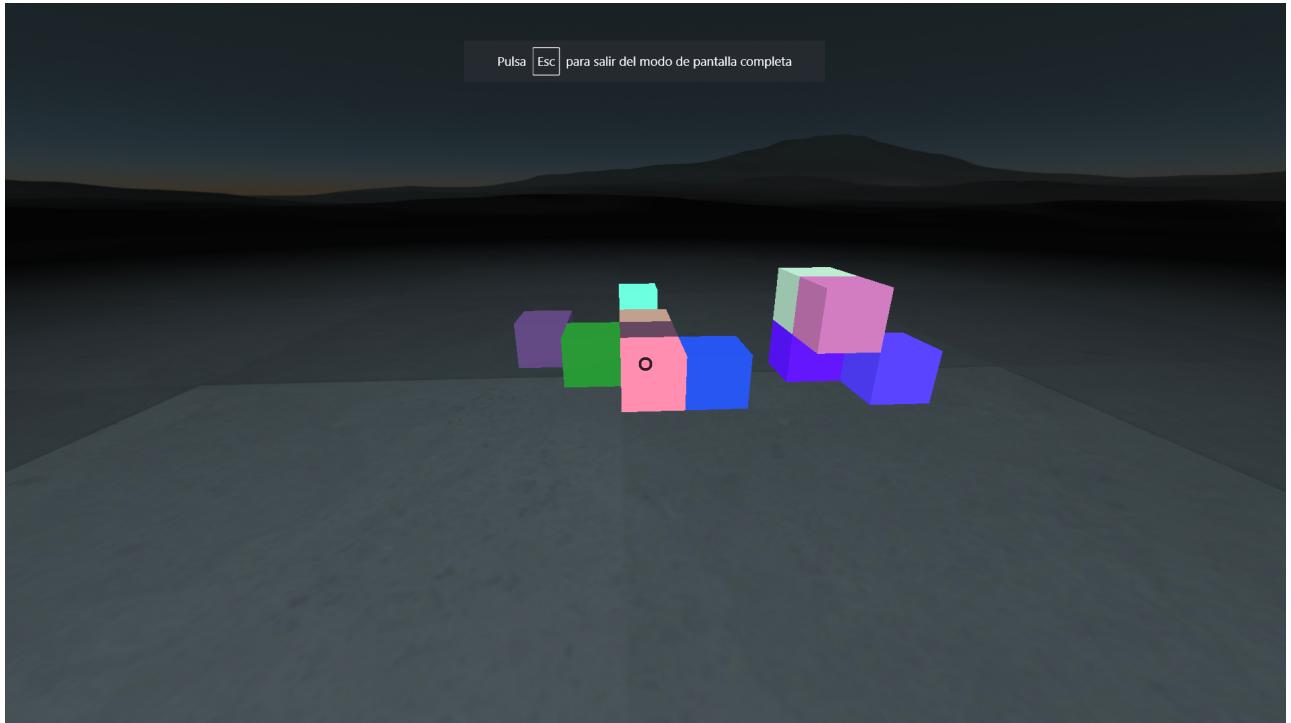
Figura 3.4: Ejemplo galería de imágenes en A-frame: Imagen 3



Por último, antes de proceder a realizar nuestras propias implementaciones y a modo de

conclusión de la etapa de aprendizaje, probaremos una demo al estilo "Minecraft"⁴ [2] en la que podremos insertar cubos en la escena con un color aleatorio en la posición donde se encuentre el cursor al hacer click, y si pulsamos en una posición en la que ya hay un cubo existente, la acción de click cambiará su color.

Figura 3.5: Ejemplo Minecraft en A-frame



3.3 Etapa 1: Primeras implementaciones

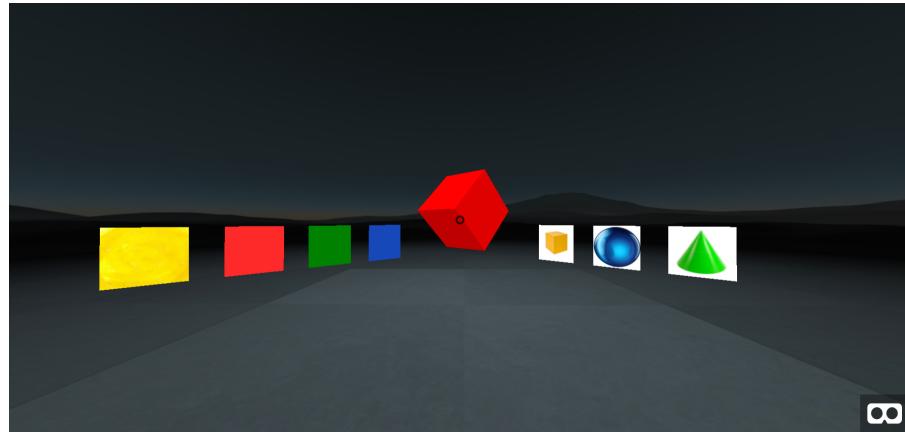
De cara a familiarizarnos con el lenguaje, entender sus intrínsecos y manejarlos mejor con las diferentes funcionalidades y características que nos ofrece A-Frame, hemos realizado una serie de tareas sencillas en lo que se definiría como primer sprint de cara a probar unas primeras implementaciones de código.

3.3.1 Posicionamiento de entidades en la escena

En esta tarea de aprendizaje tenemos en la escena una figura principal, que inicialmente será un cubo y una serie de selectores a los lados. A la izquierda 4 cubos con los colores parchís y a la derecha unos carteles con imágenes de distintas figuras.

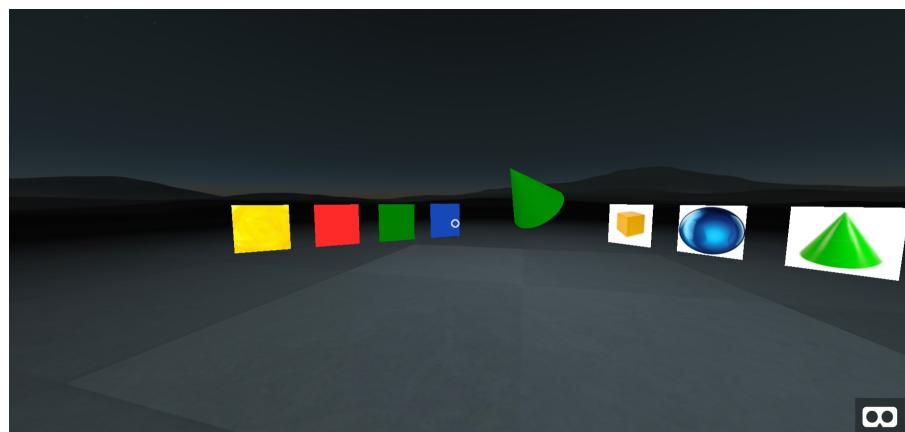
⁴<https://aframe.io/docs/1.3.0/guides/building-a-minecraft-demo.html>

Figura 3.6: Ejemplo posicionamiento de entidades en la escena



El funcionamiento de este ejercicio consiste en poder cambiar el color de la figura principal pinchando en cada uno de los selectores de color y su forma haciendo lo propio con los selectores de forma.

Figura 3.7: Ejemplo cambio de color y forma del elemento



Todos los elementos se han puesto en el archivo HTML para que aparezcan en la escena. Además, tanto para el caso de los colores como el de las figuras, hemos establecido una entidad para que los agrupe a todos y con un componente que hará el manejo de la lógica del programa:

```
<a-entity colorselector id="links"
layout="type: line; margin: 1.5; align: center"
position="-6 1 1" rotation="0 45 0">
</a-entity>
```

En el archivo Javascript definimos estos componentes, y establecemos un manejador que

permita a cada elemento emitir una variación de la figura principal para variar su forma y color:

```
var yellow = document.querySelector('#yellow');
var figure = document.querySelector('#mainFigure')
this.el.appendChild(yellow);

yellow.addEventListener('click', function () {
    changeColor(figure, 'yellow')
});
```

Además en esta escena añadimos otros detalles menos importantes como cambios de color en el puntero de la cámara para ir familiarizando y jugando con todos los componentes que nos ofrece A-Frame.

3.3.2 Uso y manejo de componentes

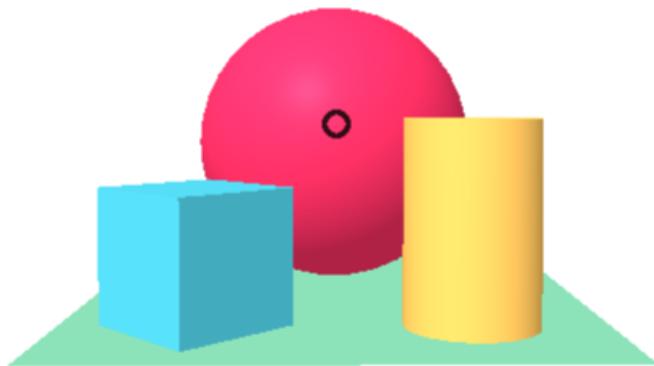
En esta tarea el objetivo es aprender más acerca de los componentes y de cómo crearlos de manera dinámica desde el archivo Javascript. Para ello en nuestra escena añadiremos un tag hacia ese componente que gestionaremos enteramente desde nuestro fichero de lógica.

```
<a-entity basic-scene></a-entity>
```

Dentro del archivo Javascript, definiremos este componente y crearemos de manera dinámica cada una de las entidades que queremos que aparezcan en la escena, dotando a cada una de ellas de sus atributos deseados:

```
AFRAME.registerComponent('basic-scene', {
  init: function() {
    // Box
    // <a-box position="-1 0.5 -3" rotation="0 45 0" color="#4CC3D9"></a-box>
    let box = document.createElement('a-box');
    box.setAttribute('color', 'red');
    box.setAttribute('position', {x: -1, y: 0.5, z: -3});
    box.setAttribute('rotation', {x: 0, y: 45, z: 0});
    this.el.appendChild(box);
  }
});
```

Figura 3.8: Ejemplo componentes A-Frame

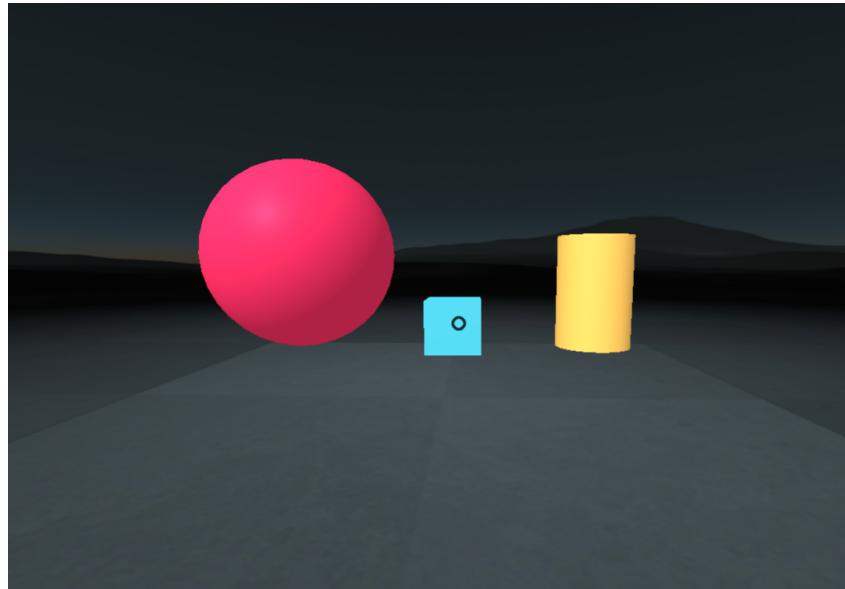


3.3.3 Anidando componentes

El propósito de esta tarea es seguir familiarizándose con el funcionamiento de los componentes, y en este caso anidando unos a otros, de forma que un componente incluya otro dentro de él mismo. Esto se consigue si al componente padre se le añade como atributo la referencia del nuevo componente en el que se crearán las nuevas figuras.

```
box.setAttribute('extra-figure', null);
AFRAME.registerComponent('extra-figure', {})
```

Figura 3.9: Ejemplo componentes anidados en A-Frame



De esta manera podemos decir que el cilindro y la esfera son elementos hijos del cubo.

3.3.4 Manejo de eventos entre componentes

En esta tarea tendremos un cubo como figura principal al cual se le añadirán dos figuras con un componente anidado cuando hagamos click en él. Esto se controla mediante un manejador de eventos, desde el cuál añadiremos o borraremos el atributo que anida el segundo componente:

```
box.addEventListener('click', function () {
  if(box.hasAttribute('extra-figure')) {
    console.log('Le quitamos las figuras extras')
    box.removeAttribute('extra-figure');
    box.emit('anEvent');
  } else {
    console.log('Le ponemos las figuras extras')
    box.setAttribute('extra-figure', {'event': 'anEvent'});
  }
});
```

Figura 3.10: Ejemplo manejo de eventos en A-Frame antes del evento

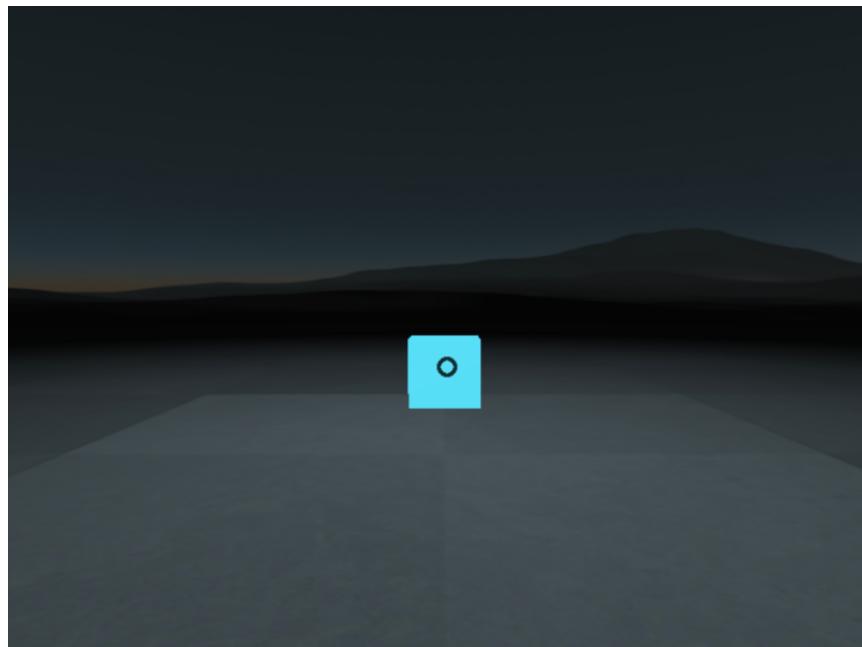
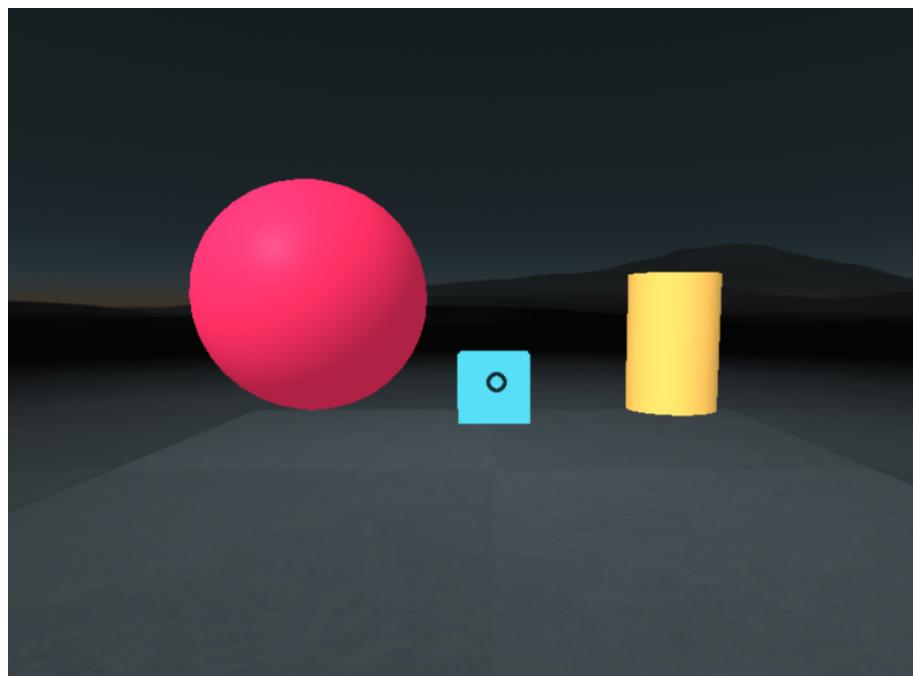


Figura 3.11: Ejemplo manejo de eventos en A-Frame después del evento



Para borrar el componente anidado al volver a hacer click en el cubo, debemos añadir la funcionalidad de borrado, que encuentra las figuras por su id y las desvincula de su componente padre.

```
remove: function () {
```

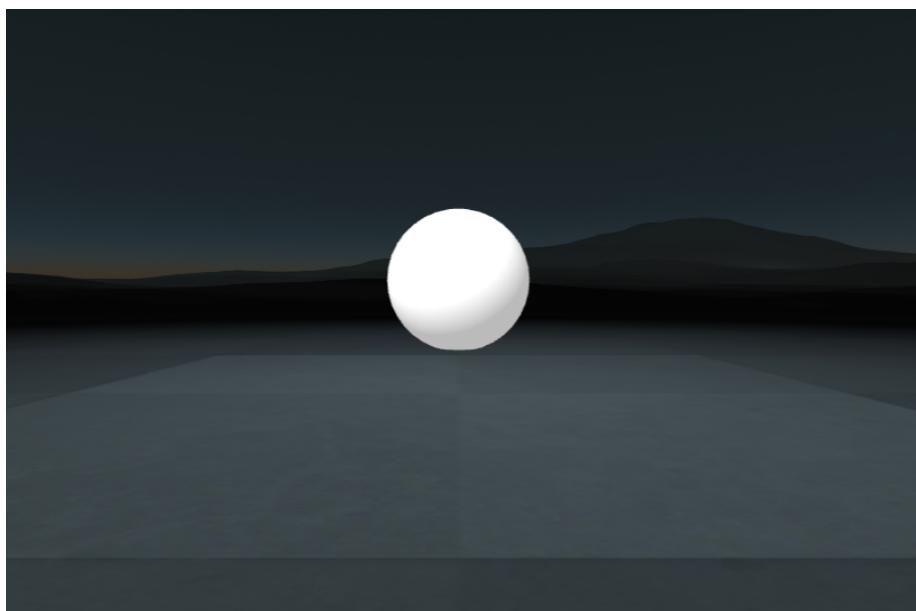
```
var data = this.data;
var el = this.el;

// Remove event listener.
if (data.event) {
    var el = document.querySelector("#sphere");
    el.parentElement.removeChild(el);
    var el = document.querySelector("#cylinder");
    el.parentElement.removeChild(el);
}
}
```

3.3.5 Unificación de lo aprendido

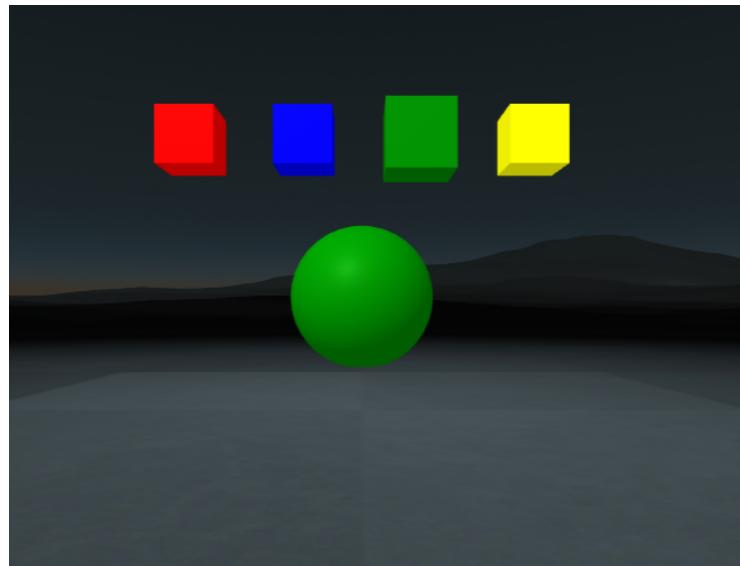
Ahora vamos a ir juntando todo lo que hemos podido aprender en tareas anteriores. Partiremos de una escena que muestra una esfera como figura principal, la cual no tiene color.

Figura 3.12: Ejemplo unificación de lo aprendido antes del evento



Al pulsar en ella desplegaremos (haciendo uso de componentes anidados) unos selectores de color que al pinchar en cada uno de ellos dotará a la figura principal del color indicado.

Figura 3.13: Ejemplo unificación de lo aprendido después del evento

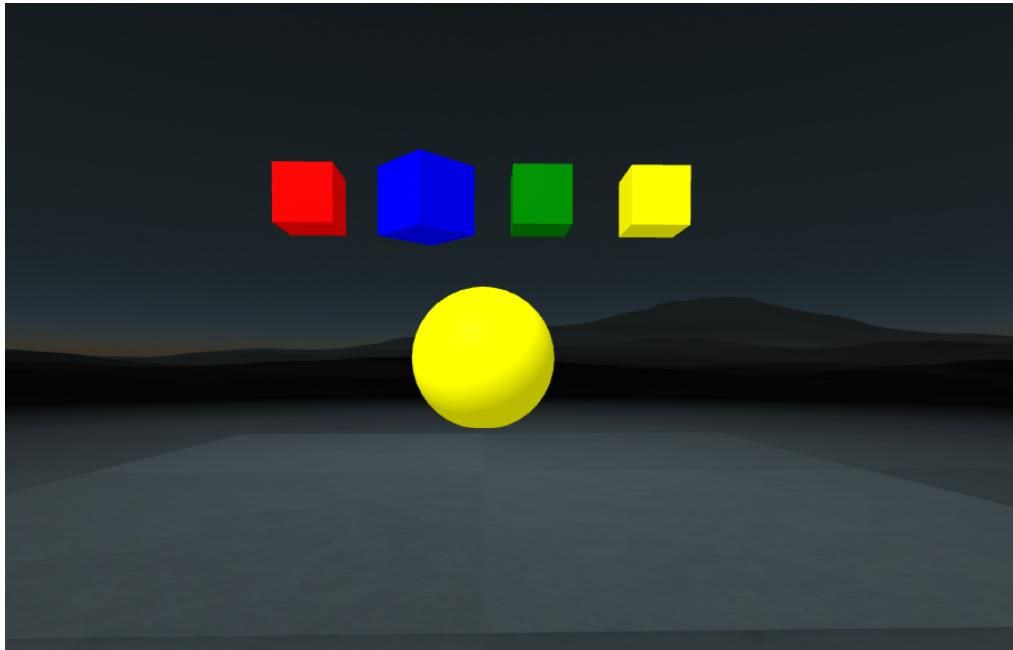


Al igual que en el ejercicio anterior, si volvemos a hacer click en la esfera desaparecerá el componente de los selectores de color, pero la figura principal mantendrá el color que le hayamos indicado.

3.3.6 Animaciones

Esta tarea tiene un funcionamiento parecido al anterior con la diferencia de que añadiremos animaciones mientras situemos el puntero encima de un selector de color. El objetivo es ir familiarizándose con las animaciones que posteriormente tendremos que usar en la herramienta final.

Figura 3.14: Ejemplo animaciones en A-Frame



El efecto de la animación se consigue dotando al componente de eventos que controlan la entrada y salida del puntero del cubo:

```

newBox.addEventListener('mouseenter', function () {
  newBox.setAttribute('scale', {x: 1.2, y: 1.2, z: 1.2});
  newBox.setAttribute('animation', 'property: rotation;
    to: 0 360 0;
    loop: true;
    dur: 3000;
    easing: linear');
});

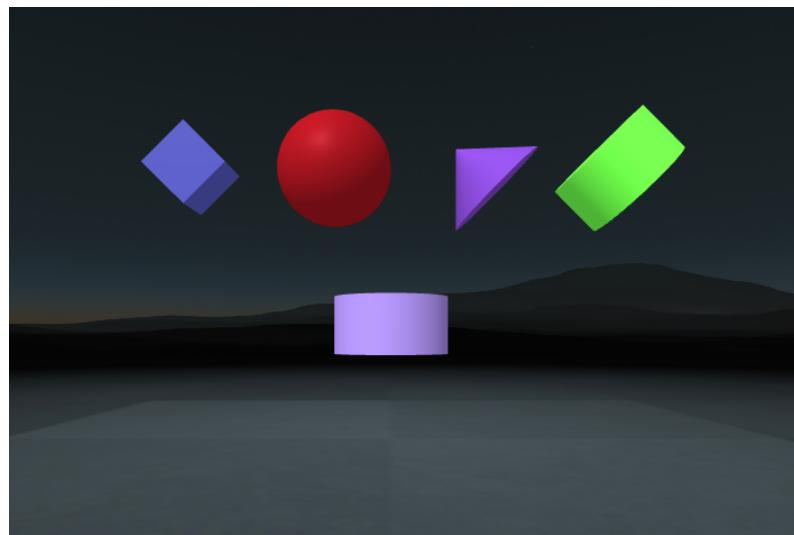
newBox.addEventListener('mouseleave', function () {
  newBox.setAttribute('scale', {x: 1, y: 1, z: 1})
  newBox.removeAttribute('animation');
  newBox.setAttribute('rotation', {x: 0, y: 0, z: 0});
});

newBox.addEventListener('click', function () {
  document.getElementById("escena").children[0].setAttribute('color',
    data.color[indice]);
  newBox.setAttribute('scale', {x: 1, y: 1, z: 1});
});
  
```

3.3.7 Profundizando en las posibilidades de A-Frame

La funcionalidad de esta tarea es muy similar a la anterior, solo que ahora en vez de tener selectores de color tendremos selectores de forma, que cambiarán el aspecto de la figura principal entre cubo, esfera, cono y cilindro.

Figura 3.15: Ejemplo cambio de formas en A-Frame



Como en este caso el color no es importante no lo hemos especificado en ninguno de los elementos de la escena, que por defecto toman uno aleatorio y si se recarga la página estos colores pueden variar.

3.3.8 Prototipo de la etapa 1

Como resultado final de unificar todas las tareas propuestas para este primer sprint juntamos ambos casos de uso, tanto el selector de color como el de forma en el mismo proyecto. Inicialmente tendremos la figura inicial que al clicar sobre ella nos mostrará el despliegue visto antes de los selectores de color. Si volvemos a pulsar en la figura principal estos selectores cambiarán al selector de forma y clicando de nuevo en la figura principal se cierran los selectores.

3.4 Etapa 2: Representación de la topología de un escenario

Una vez hemos finalizado el proceso de aprendizaje y familiarizarnos con los componentes, vamos a realizar una serie de prototipos con funcionalidades útiles que nos vayan acercan-

do cada vez más al proyecto final.

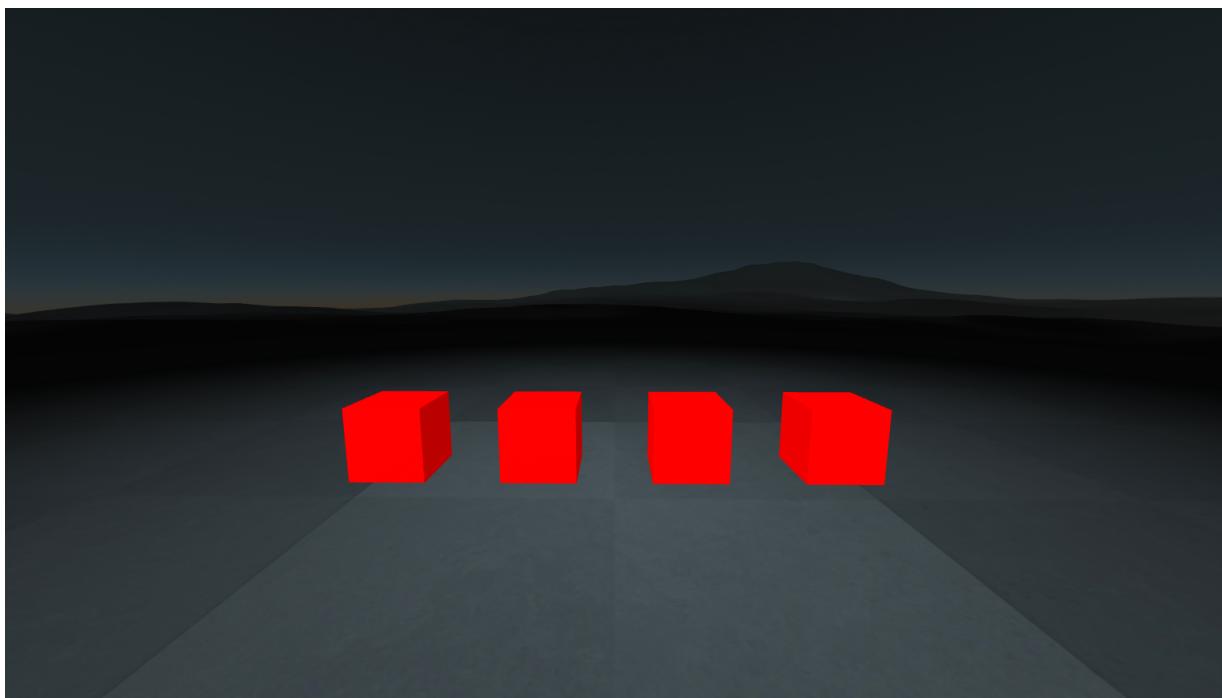
3.4.1 Representación de nodos por paso de parámetros

En esta primera tarea, trataremos de dibujar sobre la escena una red de nodos que pasaremos por parámetro a nuestro componente A-Frame.

Como parámetros del componente indicaremos el número de nodos a pintar la posición donde situaremos el primero de ellos, a partir del cual pintaremos el resto de manera lineal.

```
network="nodes: 4; position: -3 1 0"
```

Figura 3.16: Representación de nodos por paso de parámetros en A-Frame



Dentro del componente crearemos un bucle que recorra el número de nodos a crear y en cada iteración se creará un elemento box que se irá situando en la escena dejando un margen con los nodos anteriormente creados.

3.4.2 Representación de nodos en topología circular

El objetivo de esta tarea es similar a la anterior solo que ahora representaremos los nodos en formato circular y no lineal uno detrás de otro.

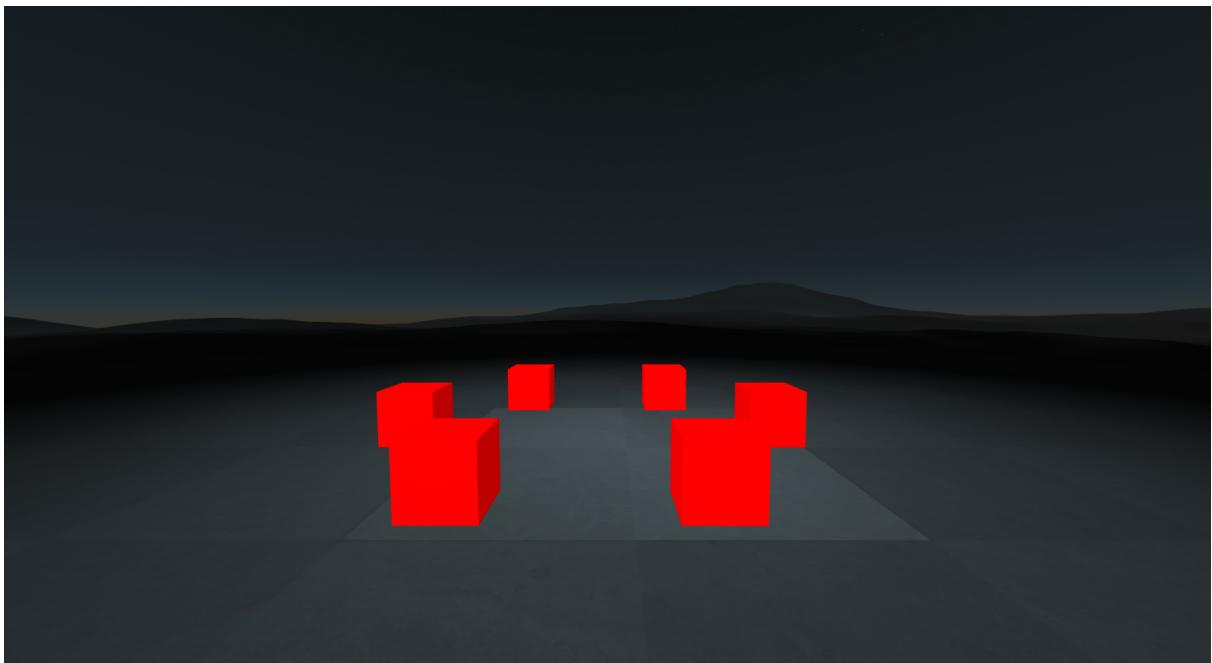
Para ello, aparte de los parámetros de número de nodos y posición del nodo inicial que establecimos anteriormente pasaremos el radio de la circunferencia a representar.

```
network="nodes: 6 ; position: 1 1 -2; radius: 20"
```

La manera de generar la circunferencia con los diferentes nodos se consigue con unos sencillos cálculos con senos y cosenos para establecer la posición x y z del nodo en cuestión.

```
data.position.x =
positionFirst.x + (data.radius / data.nodes) * Math.cos(alpha);
data.position.z =
positionFirst.z + (data.radius / data.nodes) * Math.sin(alpha);
newBox.setAttribute('position', data.position);
```

Figura 3.17: Representación de nodos en topología circular en A-Frame



3.4.3 Lectura de parámetros desde archivo JSON

Un aspecto muy importante del que haremos mucho uso en el proyecto final es la lectura de datos desde archivos JSON, por lo que en esta tarea se aborda el mismo funcionamiento que en el ejemplo anterior pero pasando todos los datos de la escena por un fichero JSON. El archivo en cuestión contiene la siguiente información:

```
{
  "nodes": 6,
  "position": {
    "x": 1,
    "y": 1,
    "z": -2
  },
  "radius": 20
}
```

Ahora dentro del componente buscamos ese fichero configurado sobre el cuál hacemos una petición para poder acceder a su contenido y parseamos este contenido para poder leer fácilmente en formato JSON y tener un mejor manejo de los datos e información que incluye nuestro fichero.

```
file = 'index.json'
scene = this.el
request = new XMLHttpRequest();
request.open('GET', file);
request.responseType = 'text';
request.send();
request.onload = function() {
  response = request.response;
  responseParse = JSON.parse(response);
  network = responseParse
}
```

3.4.4 Lectura de un fichero de paquetes real

En esta tarea tenemos un primer acercamiento de la lectura de un fichero con datos e información reales. Esta captura la hemos realizado desde un escenario de NetGUI, que posteriormente hemos pasado por el programa Wireshark para convertirla a formato JSON.

En la tarea vemos como realizamos la lectura del fichero y vamos recorriendo cada uno de los paquetes que se han realizado en la captura. En cada iteración revisamos la máquina desde la que se envía el paquete para ir detectando cada uno de los nodos que intervienen en la escena y los guardamos en una lista de máquinas.

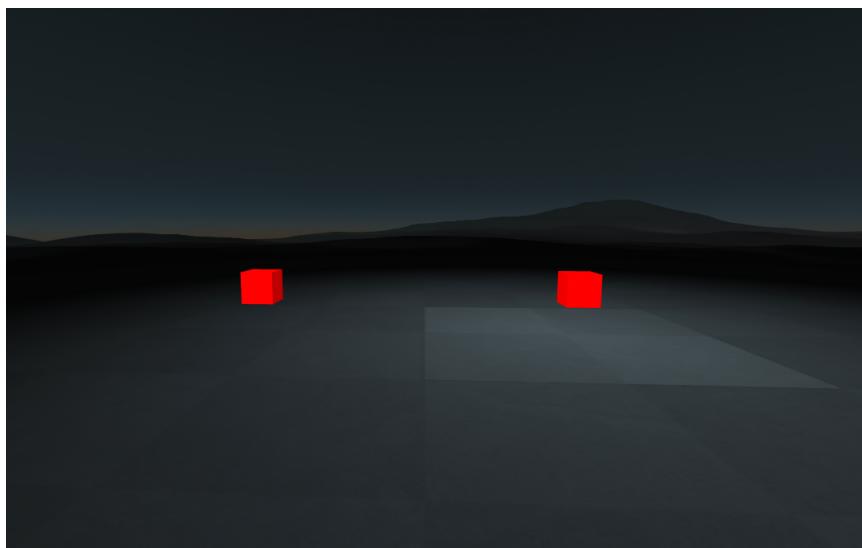
```
machines = []
for (i in responseParse) {
  if (machines.includes(responseParse[i]._source.layers.eth['eth.src'])) {
```

```
    null
} else {
    machines.push(responseParse[i]._source.layers.eth['eth.src'])
}
}
```

Una vez recorridos todos los paquetes, realizamos una representación de los paquetes de forma parecida a como lo hacíamos en la tarea anterior, calculando para cada nodo su posición en la escena.

```
0: "8e:d5:68:dd:a2:ce"  
1: "de:02:94:29:fc:d4"
```

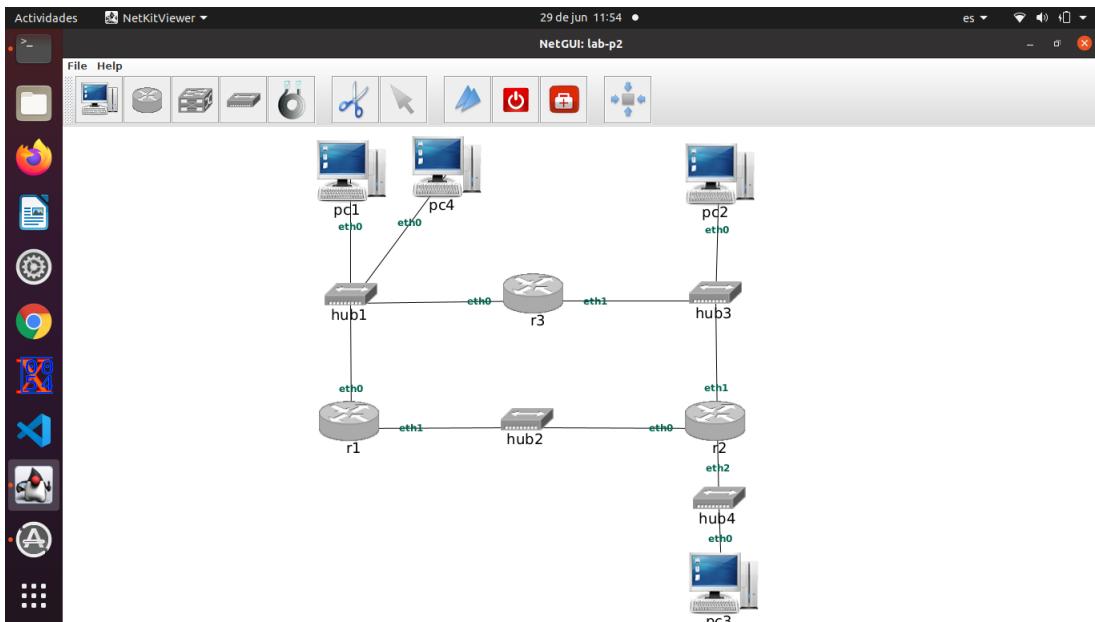
Figura 3.18: Representación de nodos desde fichero real en A-Frame



3.4.5 Representación de la topología de un escenario real de NetGUI

Para esta tarea haremos uso del fichero de arquitectura de un escenario de red de Net-GUI, acabado en la extensión nkp, para extraer la información de los nodos y conexiones y representarlo en la escena.

Figura 3.19: Escena de NetGUI original



El archivo nkp tiene por un lado información de los nodos con su nombre y su posición en un entorno 2D y por otro lado las conexiones que existen entre diferentes nodos:

```

<nodes>
position(-58.83836497991466,3.38431124866689); NKCompaq("pc1")
position(-58.05233993441159,158.46288729523633); NKHub("hub1")
...
<\nodes>
<connections>
<link>
Connect("pc1")
To("hub1")
<\link>
...
<\connections>

```

Cargamos el fichero en el programa y leemos cada uno de los nodos guardando en una lista un objeto compuesto por su nombre y su posición para posteriormente representar una caja simbolizando ese nodo acompañado con un letrero donde aparece el nombre.

```

newNode = {
  position: nodesInfo[0].slice(1),
  name: nodesName[1]
}

```

Posteriormente recorremos las conexiones y comprobamos desde la lista de nodos el nombre del nodo origen y destino de la conexión para representarlo con líneas en la escena.

```

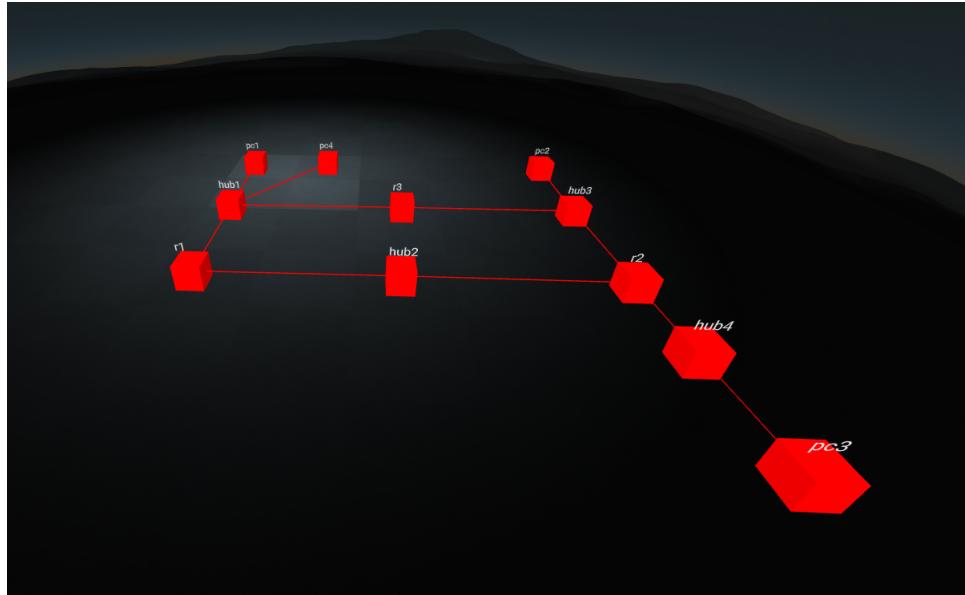
nodeFrom = nodeList.find(o => o.name === from)
nodeFromPosition = nodeFrom.position.split('”')

nodeTo = nodeList.find(o => o.name === to)
nodeToPosition = nodeTo.position.split('”')

let newLine = document.createElement('a-entity');
newLine.setAttribute('line', 'start: ' +
nodeFromPosition[0].split(',')[0] / 30 + ' 1 ' +
nodeFromPosition[0].split(',')[1] / 30 + '; end: ' +
nodeToPosition[0].split(',')[0] / 30 + ' 1 ' +
nodeToPosition[0].split(',')[1] / 30 + '; color: red');
scene.appendChild(newLine);

```

Figura 3.20: Resultado de la representación de la escena de NetGUI en A-Frame



3.5 Etapa 3: Lectura de paquetes y animación en la escena

3.5.1 Lectura y representación de paquetes reales en la escena

Partiendo del prototipo anterior donde extraemos la información de una escena de comunicación vamos a añadir información de los paquetes que se emiten entre los nodos. En

primer lugar realizaremos esa captura desde NetGUI para posteriormente convertirla a formato json desde el programa Wireshark.

Una vez hemos cargado la escena, empezaremos la lectura de la captura de paquetes, que tienen la siguiente estructura cada uno de ellos:

```
{
  "_index": "packets-2022-04-08",
  "_type": "doc",
  "_score": null,
  "_source": {
    "layers": {
      "frame": {
        "frame.encap_type": "1",
        "frame.time": "Apr 8, 2022 10:31:09.506950000 CEST",
        "frame.offset_shift": "0.000000000",
        "frame.time_epoch": "1649406669.506950000",
        "frame.time_delta": "0.000000000",
        "frame.time_delta_displayed": "0.000000000",
        "frame.time_relative": "0.000000000",
        "frame.number": "1",
        "frame.len": "42",
        "frame.cap_len": "42",
        "frame.marked": "0",
        "frame.ignored": "0",
        "frame.protocols": "eth:ethertype:arp",
        "frame.coloring_rule.name": "ARP",
        "frame.coloring_rule.string": "arp"
      },
      "eth": {
        "eth.dst": "ff:ff:ff:ff:ff:ff",
        "eth.dst_tree": {
          "eth.dst_resolved": "Broadcast",
          "eth.dst.oui": "16777215",
          "eth.addr": "ff:ff:ff:ff:ff:ff",
          "eth.addr_resolved": "Broadcast",
          "eth.addr.oui": "16777215",
          "eth.dst.lg": "1",
          "eth.lg": "1",
          "eth.dst.ig": "1",
          "eth.ig": "1"
        },
        "eth.src": "86:5b:92:e6:f6:af",
        "eth.src_tree": {
          "eth.src_resolved": "86:5b:92:e6:f6:af",
        }
      }
    }
  }
}
```

```

    "eth.src.oui": "8805266",
    "eth.addr": "86:5b:92:e6:f6:af",
    "eth.addr_resolved": "86:5b:92:e6:f6:af",
    "eth.addr.oui": "8805266",
    "eth.src.lg": "1",
    "eth.lg": "1",
    "eth.src.ig": "0",
    "eth.ig": "0"
  },
  "eth.type": "0x00000806"
},
"arp": {
  "arp.hw.type": "1",
  "arp.proto.type": "0x00000800",
  "arp.hw.size": "6",
  "arp.proto.size": "4",
  "arp.opcode": "1",
  "arp.src.hw_mac": "86:5b:92:e6:f6:af",
  "arp.src.proto_ip4": "11.0.0.1",
  "arp.dst.hw_mac": "00:00:00:00:00:00",
  "arp.dst.proto_ip4": "11.0.0.10"
}
}
}
}
}

```

En el archivo nkp de la arquitectura de la escena solo nos vienen los nombres asociados a cada nodo, 'pcX' en el caso de los ordenadores, 'hubX' para los hubs y 'rX' en el caso de los routers. Sin embargo en las capturas de paquetes, la información que obtenemos es la dirección ethernet de origen y destino de cada una de las máquinas.

Por lo tanto, para unificar la información que tenemos de la arquitectura de la escena y el intercambio de paquetes entre nodos deberemos dotar a cada nodo de su nombre de máquina correspondiente:

```

nodeList.find(o => o.name === 'pc1').machineName = '4e:da:80:75:67:a6'
nodeList.find(o => o.name === 'r1').machineName = '86:5b:92:e6:f6:af'

```

Ahora podremos recorrer cada uno de los paquetes de comunicación, extraer la dirección ethernet del nodo de origen y destino y crear una animación que vaya de uno a otro.

```

newPacketAnimation.setAttribute('animation', {
  property: 'position',

```

```

    to: packetDst.position.split(',')[0] / 30 + ' 1 ' +
    packetDst.position.split(',')[1] / 30,
    dur: 5000,
    delay: packetDelay,
    easing: 'linear'
});

}

```

Las dimensiones de la escena están divididas en un factor de 30 para que los espacios entre nodos no se vean excesivamente lejanos.

Por último, para representar el paquete hemos utilizado un pequeño cilindro. Para dar una mayor sensación de paquete de comunicación, hemos rotado dicho cilindro para que se oriente en la dirección de la línea de conexión entre los nodos por los que se transporta. Para calcular esa inclinación, hemos realizado una pequeña función que haciendo uso de propiedades trigonométricas devolvemos el ángulo de inclinación del paquete en cuestión:

```

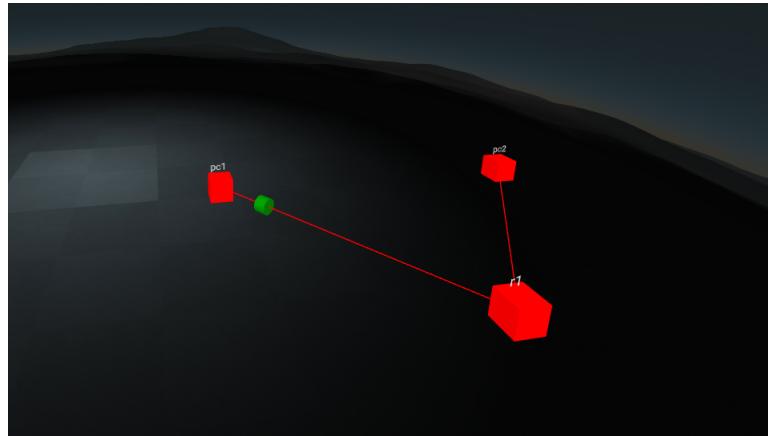
function calculateAngle(x1, z1, x2, z2) {
    cateto1 = Math.abs(parseFloat(x1) - parseFloat(x2))
    cateto2 = Math.abs(parseFloat(z1) - parseFloat(z2))
    hipotenusa = Math.sqrt(cateto1 * cateto1 + cateto2 * cateto2);
    A = Math.asin(cateto1 / hipotenusa);

    A_s = A * 180 / Math.PI;

    if (parseFloat(x1) > parseFloat(x2)) {
        if (parseFloat(z1) > parseFloat(z2)) {
            return (A_s * (-1))
        } else {
            return A_s
        }
    } else {
        if (parseFloat(z1) > parseFloat(z2)) {
            return A_s
        } else {
            return (A_s * (-1))
        }
    }
}

```

Figura 3.21: Representación de paquetes en A-Frame



3.5.2 Creando componente para detener y reanudar la escena

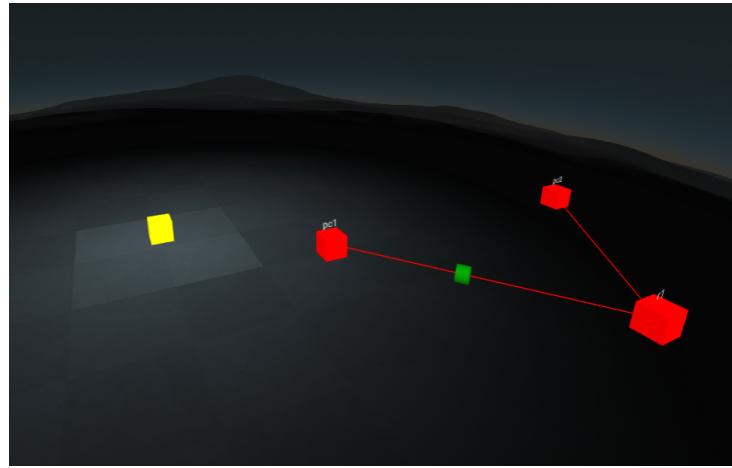
Para esta tarea, aparte de todo lo anteriormente implementado vamos aadir una caja que de momento detecte el evento de click en ella. En el futuro, haremos algo cuando se detecte este evento, pero en esta tarea solo escribiremos un log por pantalla indicando que la caja se ha pulsado.

Para realizar esto es tan sencillo como aadir a la escena el elemento de la caja que hemos creado y dotarle de un evento para identificar clicks en ella:

```
let starter = document.createElement('a-box');
starter.setAttribute('color', 'yellow');
starter.setAttribute('position', {x: 0, y: 1, z: 0});

escena = document.querySelector('#escena');
escena.appendChild(starter);
starter.addEventListener('click', function () {
  console.log('Se pulsa la caja')
});
```

Figura 3.22: Controlador dentro de la escena para detección de eventos en A-Frame



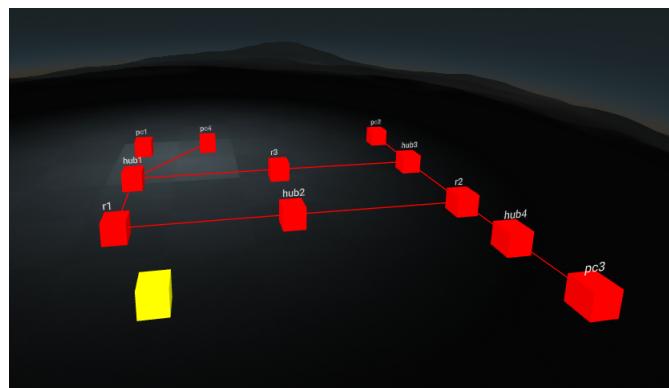
3.5.3 Controlando el inicio de las animaciones

Ahora sí vamos a hacer uso de esta caja añadida en el ejercicio anterior para activar la animación de los paquetes cuando se pulsa.

Al detectar el evento de click en ella, la caja llamará a la función para activar las animaciones de los paquetes, que previamente hemos leído de nuestro fichero con las capturas y lo hemos almacenado en una lista llamada process donde cada paquete intercambiado lleva la información del nodo origen y el nodo destino.

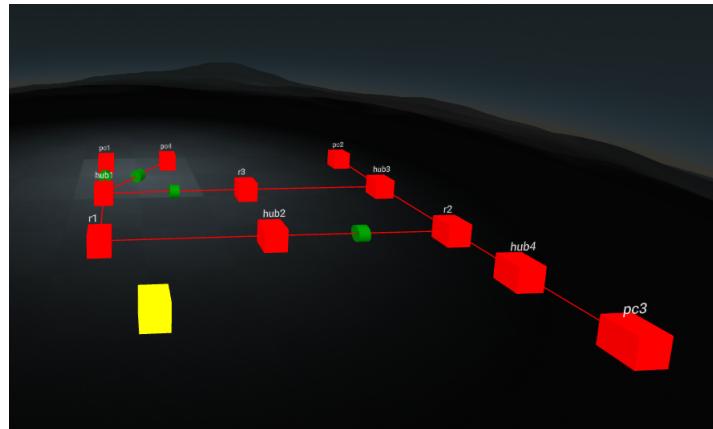
```
startButton.addEventListener('click', function () {
    animatePackets(process, finalConnectionsLinks)
});
```

Figura 3.23: Representación de una escena de NetGUI con controlador para inicio de animaciones en A-Frame



También debemos destacar, que cuando el nodo destino de un paquete es la dirección ff.ff.ff.ff.ff (es decir, difusión en broadcast a todos los nodos conocidos) implementamos un mecanismo para que cada nodo por el que pase ese paquete reenvíe a todos los nodos con los que tenga conexión. Esto lo haremos mediante un pequeño bucle for en el nodo en cuestión para que recorra todas las conexiones que tiene ancladas y reenvíe el paquete por ahí.

Figura 3.24: Representación del tránsito de paquetes en A-Frame



3.6 Etapa 4: Añadiendo información a los paquetes

3.6.1 Información de niveles al clicar en los paquetes

En este ejercicio hemos implementado principalmente dos nuevas funcionalidades además de una mejora en la estructuración del código. En primer lugar podemos parar/reanudar las animaciones pulsando en el cubo amarillo, de forma que ya no solo funcionará como el inicio de la animación, sino como un controlador de play/pause.

Para realizar esto, en primer lugar a la hora de construir un paquete debemos pasarle una clase general que llevarán todos los paquetes (y más adelante entenderemos para qué sirve) y un parámetro denominado start que nos indica cuándo arrancará ese paquete su animación en la escena (lo hacen por orden, yendo un paquete detrás del otro).

```
newPacket.setAttribute('packet', 'class', 'packetClass')
newPacket.setAttribute('packet', 'start',
finalPackets[currentPacket].packetDelay);
```

En primer lugar, guardamos en una variable tanto el cubo que funciona como botón de inicio de la animación y todos los paquetes dibujados en la escena mediante el método

getElementsByClassName con el que podemos seleccionar todos aquellos elementos que comparten una clase determinada como es este caso.

Definimos el estado de la animación que según el momento en el que se encuentre puede estar sin empezar, en marcha o en pausa. Cuando se pulsa el cubo amarillo y comienza la animación, evaluamos el estado de la animación para actualizarlo al nuevo estado, y en el caso de que se esté iniciando la animación, llamaremos a la función de empezar la animación:

```
var startButton = document.querySelector('#startButton');
var packets = document.getElementsByClassName('packetClass');

var animationStatus = 'animation-starts'
startButton.addEventListener('click', function () {
    for (var a=0; a< packets.length; a++) {
        packets[a].emit(animationStatus,null,false)
    }

    switch(animationStatus) {
        case 'animation-starts':
            setInterval(startAnimation, 1000);
            animationStatus = 'move-pause'
            break
        case 'move-resume':
            animationStatus = 'move-pause'
            break
        case 'move-pause':
            animationStatus = 'move-resume'
            break
    }
});
```

En la función de la animación propiamente dicha, primero comprobaremos que estamos en el estado de movimiento. A continuación comprobamos que estamos en el momento donde tiene que empezar el movimiento del paquete, mediante el parámetro start que hemos pasado como parámetro en la definición del componente. Una vez se cumplen estas dos condiciones, obtenemos por id el paquete en cuestión y le pasamos la propiedad de la animación para que vaya del nodo origen al destino. Para poder detener la animación deberemos añadir en el atributo los eventos de pause y resume como vemos en el siguiente código:

```
let i = 0;
function startAnimation() {
```

```

if (animationStatus == 'move-pause') {
    if (i == Math.ceil(packetParams.start/1000)) {
        var packet_move = document.getElementById(packetParams.id);
        packet_move.setAttribute('animation', {
            property: 'position',
            to: packetParams.toXPosition +
                packetParams.toYPosition +
                packetParams.toZPosition,
            dur: packetParams.duration,
            easing: 'linear',
            pauseEvents: 'move-pause',
            resumeEvents: 'move-resume'
        });
    }
    i++;
}
}

```

La otra gran implementación que hemos añadido en esta tarea es la muestra de información al clicar en cada paquete. Ahora del fichero de paquetes realizados en la captura no extraeremos solo la dirección origen y destino de cada uno de los paquetes, sino que guardaremos toda la información que tenga cada paquete, nivel Ethernet, IP, ARP, etc. Esta información se la pasaremos a los paquetes a la hora de crearlos:

```

if (finalPackets[currentPacket].ip){
    newPacket.setAttribute('packet','ip', finalPackets[currentPacket].ip);
}
if (finalPackets[currentPacket].eth){
    newPacket.setAttribute('packet','eth', finalPackets[currentPacket].eth);
}
...

```

A la hora de crear el paquete, añadiremos un manejador que detecte si se pulsa en dicho paquete para mostrarnos por consola la información que transporte de cada uno de los niveles.

```

let packetParams = this.data

...
packet.addEventListener('click', function () {
    console.log('IP: ');
    console.log(packetParams.ip);
}

```

```

    console.log('ETH: ');
    console.log(packetParams.eth);
    console.log('ARP: ');
    console.log(packetParams.arp);
    console.log('DATA: ');
    console.log(packetParams.dataInfo);
    console.log('TCP: ');
    console.log(packetParams.tcp);
});

}

```

Aquí vemos un ejemplo de la información que mostraría por consola un paquete que transporta diferentes niveles de comunicación:

Figura 3.25: Información transportada por un paquete de la escena

IP:	index.js:95
	index.js:96
▶ {ip.version: '4', ip.hdr_len: '20', ip.dsfield: '0x00000000', ip.dsfield_tr ee: {...}, ip.len: '57', ...}	
ETH:	index.js:97
	index.js:98
▶ {eth.dst: '4e:da:80:75:67:a6', eth.dst_tree: {...}, eth.src: '86:5b:92:e6:f6: af', eth.src_tree: {...}, eth.type: '0x00000800'}	
ARP:	index.js:99
null	index.js:100
DATA:	index.js:101
▶ {data.data: '68:6f:6c:61:0a', data.len: '5'}	index.js:102
TCP:	index.js:103
	index.js:104
▶ {tcp.srcport: '11111', tcp.dstport: '33333', tcp.port: '33333', tcp.stream: '0', tcp.len: '5', ...}	
>	

3.6.2 Representación de la información de los paquetes

En esta tarea se han abordado numerosos cambios e importantes avances de cara a la versión final del proyecto.

En primer lugar, debido a que la forma en la que dotábamos de los nombres de máquina a cada uno de los nodos de la escena se hacía en el propio código y no escalaba a otro tipo de escenarios, hemos decidido pasar como parámetro un fichero json que contiene los nombres de todos los nodos de la escena, el cual se lee y se asocia a las entidades. Este es el fichero que hemos usado para esta demo:

```
[
{
  "pc1": "4e:da:80:75:67:a6",
  "pc2": "4e:da:80:75:67:a5",
  "pc3": "4e:da:80:75:67:a7"
}
```

```

"pc3": "4e:da:80:75:67:a4",
"pc4": "4e:da:80:75:67:a3",
"r1": "86:5b:92:e6:f6:af",
"r2": "86:5b:92:e6:f6:aa",
"r3": "86:5b:92:e6:f6:ab"
}
]

```

Y aquí mostramos como asociamos a cada nodo el nombre de máquina correspondiente:

```

for (const machineNamesElement in responseParse) {
    nodeList.find(o => o.name ===
        machineNamesElement).machineName = responseParse[machineNamesElement]
}

```

El siguiente paso será mostrar de forma visual para el usuario la información de cada uno de los niveles de comunicación que contiene cada paquete (lo que en la tarea anterior hicimos mediante textos por consola, ahora habrá que mostrarlo en la escena).

Para ello, definimos en primer lugar una variable con los niveles de información que tendrá cada paquete, un booleano para indicar si la información se encuentra o no visible en la escena, una variable con el texto a mostrar en cuanto a información del nivel y otro booleano para indicar la validez. Ahora creamos la entidad del cartel con la información y la asociamos al paquete, estableciendo su visibilidad como oculta, ya que esto no se mostrará hasta que pinchemos en el selector de alguno de los niveles.

```

let levels = []
let closeInfo = true
let actualInfoShown = ''
notValid = false

let newInfoText = document.createElement('a-entity');
newInfoText.setAttribute('geometry', {primitive:'plane',height: 3, width: 6});
newInfoText.setAttribute('material', 'color', 'white');
newInfoText.setAttribute('position',
{x: 0, y: -1 + Math.sign(packetParams.angle) *
Math.round(Math.abs(packetParams.angle)/90) + yPositionPlus, z: -5 });
newInfoText.setAttribute('rotation',
{x: -90 + packetParams.angle, y: 90, z: - 90 });
newInfoText.setAttribute('visible', false);
newInfoText.setAttribute('isPoster', true);

packet.appendChild(newInfoText);

```

Comprobamos los niveles de información que transporta el paquete y los añadimos a la variable levels de la siguiente manera:

```
if(packetParams.eth){
  const ethInfo = {
    eth: packetParams.eth
  }
  levels = Object.assign(levels,ethInfo);
}
```

Una vez tenemos en la variable levels todos los niveles presentes en el paquete, vamos comprobando uno por uno para añadir el selector de información correspondiente a dicho nivel. Crearemos una caja que se situará encima del paquete (más a la izquierda o a la derecha en función del número de niveles que tengamos) y acompañada de unos manejadores que detecten cuando se haga click en la propia caja.

Cuando se pulsa, distinguiremos 3 posibles casuísticas: que no haya información visible de ningún nivel, en cuyo caso se muestra por pantalla la del nivel pulsado; que haya información visible y sea de otro nivel, en cuyo caso se cambiará al nivel indicado; o que haya información visible y sea justo la del nivel que hemos pulsado, en cuyo caso forzará el cierre de la información. Solo si no hay información visible, podremos volver a pulsar en el paquete para cerrar los selectores de niveles. Toda la lógica de cuando se muestran los selectores o la información de los paquetes se controla con una serie de validadores en forma de variables.

Aquí vemos el ejemplo de código referente al nivel ethernet, el cual habría que replicar para el resto de niveles existentes en el escenario de comunicación:

```
if(levels.hasOwnProperty('eth')){
  index = Object.keys(levels).findIndex(item => item === 'eth')

  let newEthBox = document.createElement('a-box');
  newEthBox.setAttribute('position',
  { x: -(Object.keys(levels).length - 2) + (2 * index) -1, y: 0, z: -2 });
  newEthBox.setAttribute('color', 'red');
  newEthBox.setAttribute('visible', false);
  packet.appendChild(newEthBox);

  newEthBox.addEventListener('mouseenter', function () {
    newEthBox.setAttribute('scale', {x: 1.2, y: 1.2, z: 1.2});
    newEthBox.setAttribute('animation',
    'property: rotation;
    to: 0 0 -360;
```

```

        loop: true;
        dur: 3000; easing: linear');
    });
    newEthBox.addEventListener('mouseleave', function () {
        newEthBox.setAttribute('scale', {x: 1, y: 1, z: 1})
        newEthBox.removeAttribute('animation');
        newEthBox.setAttribute('rotation', {x: 0, y: 0, z: 0});
    });
    newEthBox.addEventListener('click', function () {
        if(newEthBox.hasAttribute('isVisible')){
            let infoText = 'Nivel Ethernet:\n\nOrigen: ' +
                packetParams.eth['eth.src'] + '\nDestino: ' +
                packetParams.eth['eth.dst'] + '\nTipo: ' +
                packetParams.eth['eth.type']
            if(closeInfo == true){
                closeInfo = false
                actualInfoShown = 'eth'
                newInfoText.setAttribute('visible', true);
                newInfoText.setAttribute('text', {width:10, color:'black',
                    value: infoText, align:'center'});
            }else if(closeInfo == false && actualInfoShown == 'eth'){
                actualInfoShown = ''
                newInfoText.setAttribute('visible', false);
            }else if(closeInfo == false && actualInfoShown != ''){
                closeInfo = false
                actualInfoShown = 'eth'
                newInfoText.setAttribute('text', {width:10, color:'black',
                    value: infoText, align:'center'});
            }
        } else {
            notValid = true
        }
    });
}

```

Figura 3.26: Representación de una escena de NetGUI antes de iniciarse las animaciones

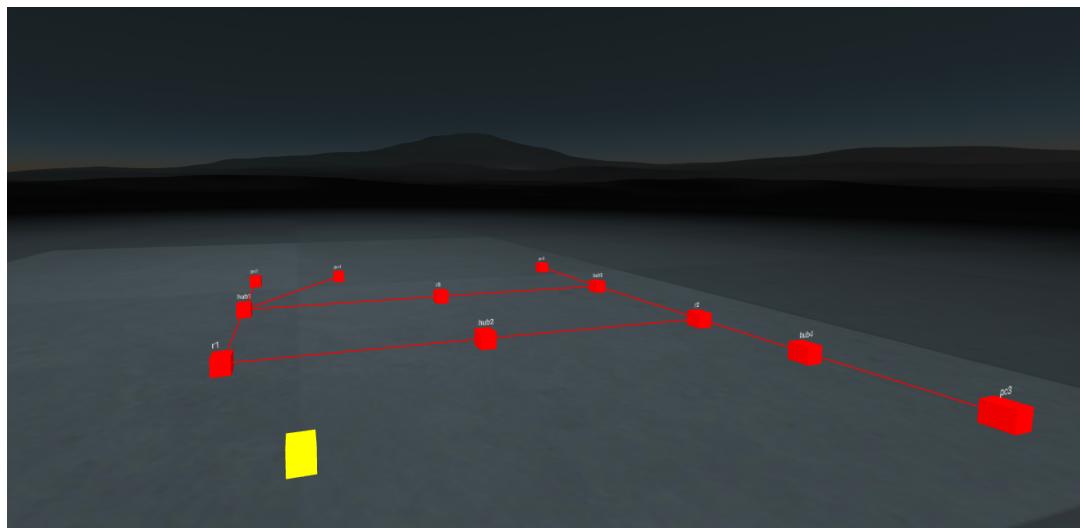


Figura 3.27: Representación de una escena de NetGUI con paquetes en tránsito y sus niveles desplegados

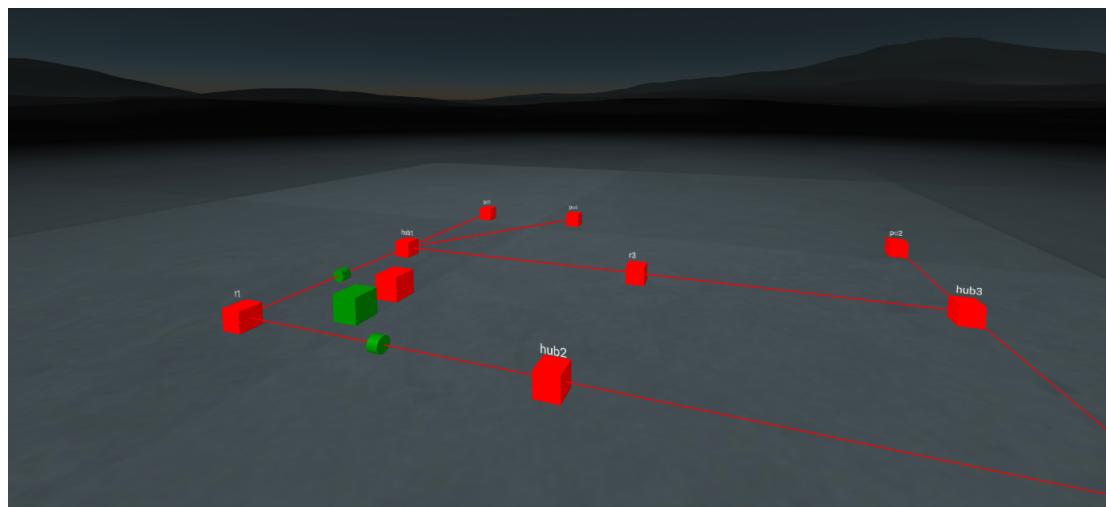


Figura 3.28: Muestra de información del nivel ARP

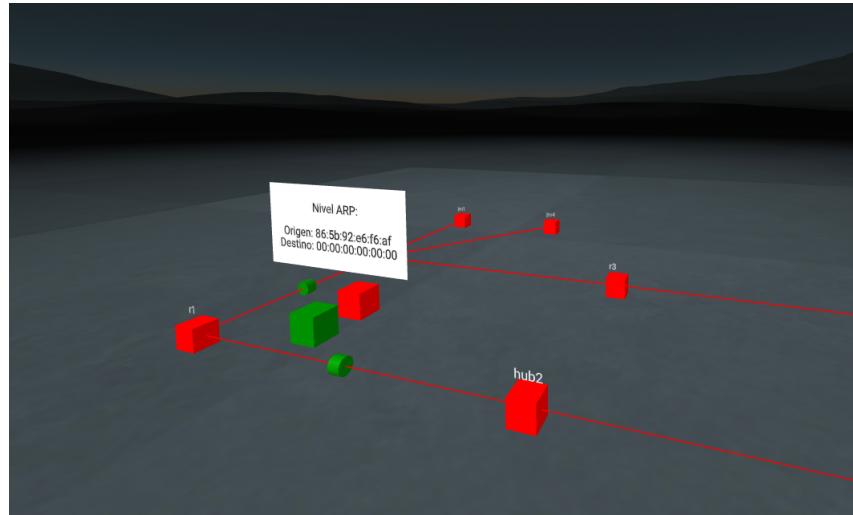
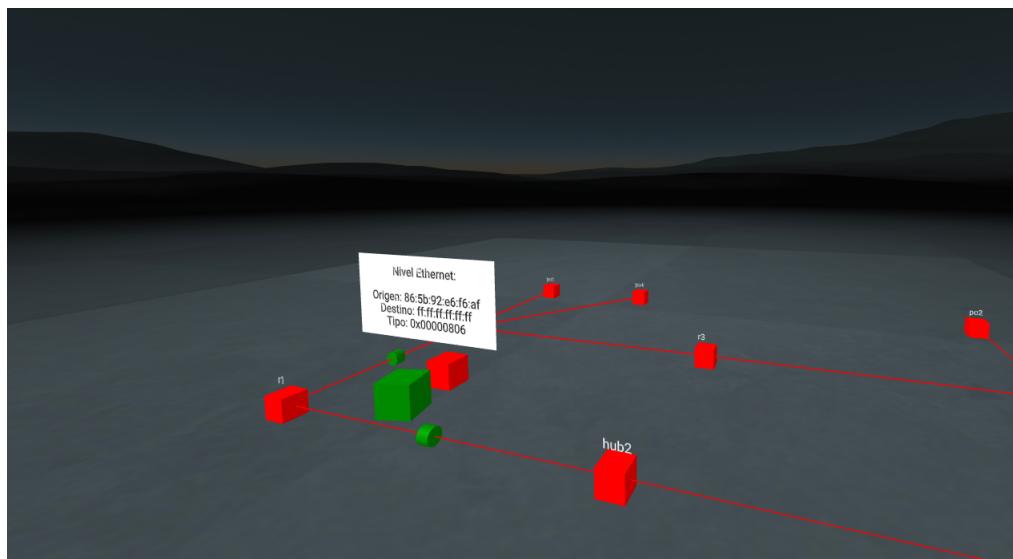


Figura 3.29: Muestra de información del nivel Ethernet



3.7 Etapa 5: Experiencias en realidad virtual

3.7.1 Mejorando aspecto visual de los componentes

Dentro de este apartado, hemos modificado tanto el aspecto de los paquetes como su desglose de información por niveles, para adecuarlo más a un estándar y se visualice mejor.

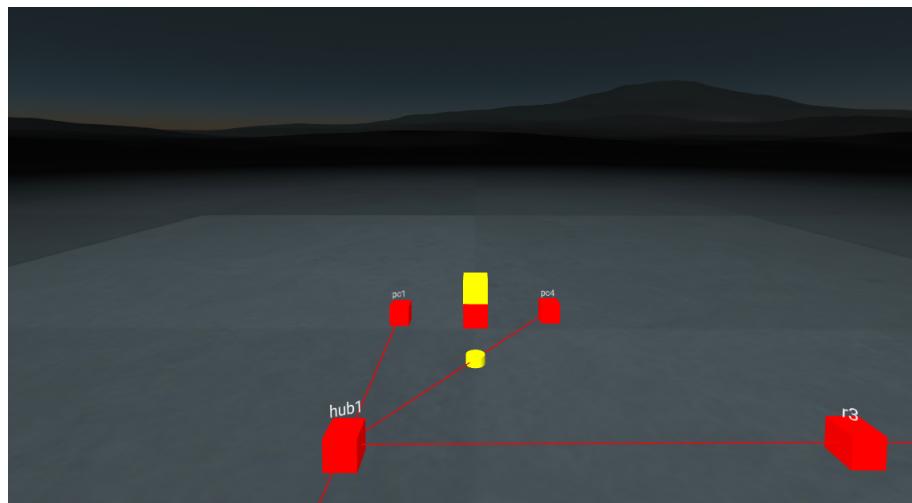
Ahora el cilindro que simboliza a los paquetes no estará girado y orientado desde el nodo origen hacia el nodo destino, ya que esto daba muchos conflictos a la hora de realizar

cálculos con los ángulos tanto para el propio paquete como para los selectores de niveles que ofrecía el paquete al clicar sobre él. Ahora mostraremos el paquete como un cilindro vertical sin ninguna orientación. Su color tampoco va a ser un color estándar, sino que el paquete adoptará el color del nivel de comunicación más alto que transporte, siguiendo el siguiente orden de colores:

```
{
  "ethernet": "red",
  "ip": "yellow",
  "arp": "green",
  "tcp": "blue",
  "data": "white"
}
```

Cuando cliquemos en un paquete, nos aparecerá el desglose de selectores de los niveles que transporta, pero esta vez en vez de ofrecerlo en formato horizontal, lo ofreceremos verticalmente, cada nivel encima del otro, y cada uno representado con un cubo de su color asociado:

Figura 3.30: Muestra de niveles de información de un paquete



De esta manera calculamos la altura donde tiene que ir cada cubo, en función del número de niveles que transporta dicho paquete y del nivel que representa dicho cubo:

```
newEthBox.setAttribute('position',
{ x: -(Object.keys(levels).length - 2) + (2 * index) - 1, y: 0, z: -2 });
```

Pulsando en alguno de esos selectores de nivel, mostraremos el letrero con la información, cuyo aspecto hemos cambiado ligeramente respecto a la tarea anterior. Ahora tendrá un

fondo gris predeterminado y el color de las letras será el color del nivel seleccionado, siguiendo el orden de colores mostrado anteriormente. Además, tanto a estos carteles como a los títulos de cada uno de los nodos le hemos añadido el atributo look-at, que permite orientarse a una posición concreta para que el elemento siempre mire hacia esa dirección. En nuestro caso lo orientamos hacia la cámara, para que nos movamos donde nos movamos, siempre podamos leer con facilidad todos los textos.

```
newInfoText.setAttribute('look-at', "[camera]");
```

Figura 3.31: Muestra de información del nivel IP

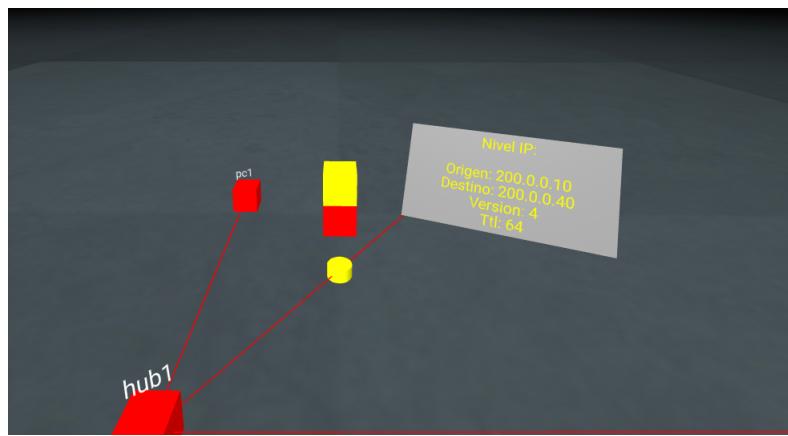
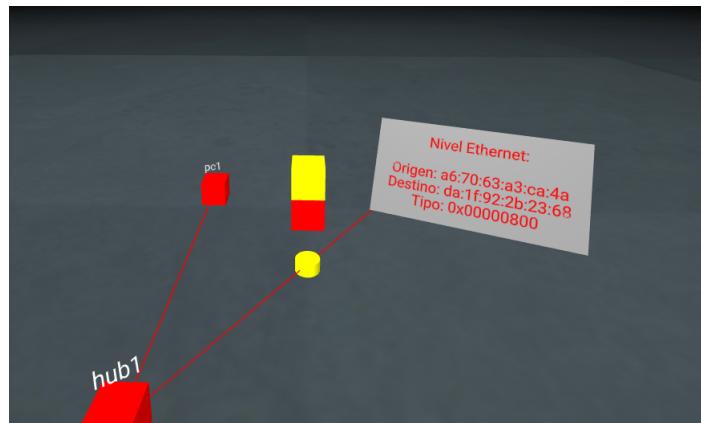


Figura 3.32: Muestra de información del nivel Ethernet



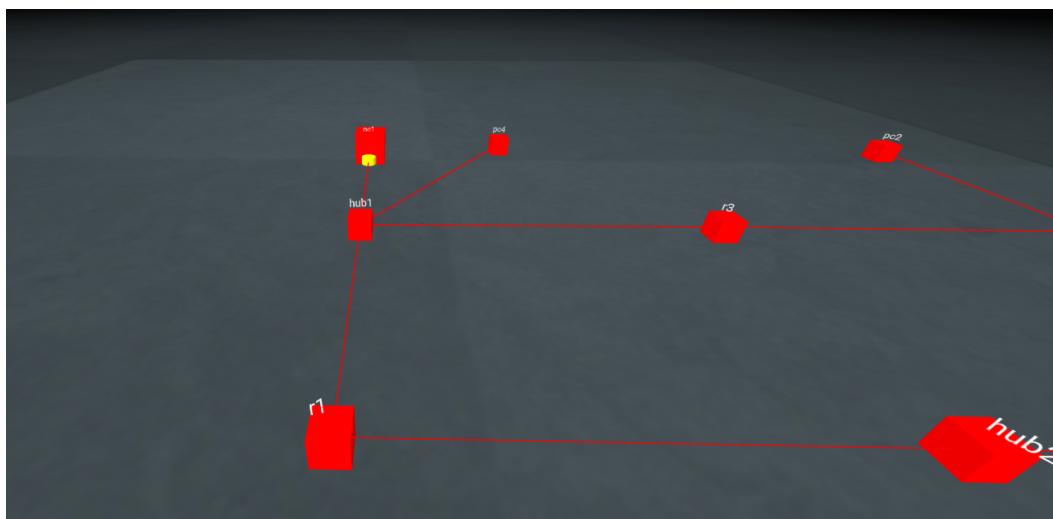
El siguiente cambio aplicado ha sido la introducción de animaciones en el nodo del que sale un nuevo paquete, para que visualmente el usuario pueda ver con facilidad dónde se encuentra el nuevo paquete y de forma intuitiva entienda mejor la escena y el recorrido que ha tenido la comunicación.

Para crear esto, debemos identificar el nodo del que sale el paquete. Esto lo haremos por correspondencia a la posición de salida del paquete, que coincidirá con la posición del

nodo en cuestión. Obtenemos ese elemento por su Id y le añadimos la propiedad de la animación, que variará su escala momentáneamente para producir una leve sensación de latir del nodo.

```
nodeAnimation = nodeList.find(o => o.position === nodePositionFrom)
var nodeFromAnimation = document.getElementById(nodeAnimation.name);
nodeFromAnimation.setAttribute('animation',
{property: 'scale', from: '2 2 2', to: '1 1 1',
loop: '2', dur: '1000', easing: 'linear'})
```

Figura 3.33: Animaciones en los nodos de los que sale un paquete



La última implementación referente a la escena consiste en la introducción de sonidos envolventes, cada vez que se realiza algún click o aparece en escena algún nuevo paquete. En primer lugar deberemos descargar los sonidos, y añadirlos a nuestro proyecto A-Frame en el apartado de assets del archivo HTML, donde definimos nuestra escena.

```
<audio id="packetIn" crossorigin="anonymous" src="packetIn.mp3"></audio>
<audio id="showLevels" crossorigin="anonymous" src="showLevels.mp3"></audio>
<audio id="showInfo" crossorigin="anonymous" src="showInfo.mp3"></audio>
```

Como cada audio está referenciado por un identificador, cuando queramos reproducir el sonido simplemente añadimos el atributo de audio al elemento haciendo referencia a ese identificador para reproducir el sonido. A-Frame nos permite controlar ciertas propiedades del audio como por ejemplo el volumen. Además, a la hora de visualizar la escena en realidad virtual este sonido será totalmente envolvente, percibiendo con total precisión de dónde viene el sonido y siendo este más intenso mientras más cerca estemos del elemento que lo produce.

```
packet.setAttribute('sound', {src: '#packetIn', volume: 5, autoplay: "true"});
```

3.7.2 Unificando capturas

Pasemos ahora a los cambios referentes a como trataremos los paquetes que se han capturado en la escena real de comunicación.

Hasta ahora, tan solo habíamos representado paquetes de la captura de una máquina en la vida real, por lo que todo el tráfico que se iba a mostrar iba a pasar sí o sí por ese nodo desde el que se había tomado la captura. Pero la realidad es que en una escena de comunicación son muchos nodos los que se ven involucrados y no necesariamente todos los paquetes deben pasar por alguno de ellos.

Por ese motivo, en primer lugar deberemos tomar capturas de la comunicación en todas las máquinas que pertenecen a la escena, y en aquellas donde haya varias interfaces deberemos tomar una captura para cada una de ellas.

De esta manera tendremos la siguiente ristra de capturas en formato json, habiéndolas previamente exportado desde Wireshark:

Figura 3.34: Listado de capturas tomadas en la escena

pc1	21/05/2022 14:40	Archivo de origen ...	246 KB
pc2	21/05/2022 12:26	Archivo de origen ...	131 KB
pc3	21/05/2022 12:27	Archivo de origen ...	138 KB
pc4	21/05/2022 12:27	Archivo de origen ...	246 KB
r1eth0	21/05/2022 12:27	Archivo de origen ...	246 KB
r1eth1	21/05/2022 12:28	Archivo de origen ...	101 KB
r2eth0	21/05/2022 12:28	Archivo de origen ...	101 KB
r2eth1	21/05/2022 12:29	Archivo de origen ...	131 KB
r3eth0	21/05/2022 12:29	Archivo de origen ...	246 KB
r3eth1	21/05/2022 12:29	Archivo de origen ...	131 KB

Para saber cuántas capturas de paquetes hemos tomado, crearemos un pequeño archivo JSON que posteriormente iremos recorriendo para indicarle cuáles son los nombres de las capturas y los pueda recorrer fácilmente para su lectura.

```
[  
    "pc1.json",  
    "pc2.json",  
    "pc3.json",  
    "pc4.json",  
    "r1eth0.json",  
    "r1eth1.json",  
    "r2eth0.json",  
    "r2eth1.json",  
    "r3eth0.json",  
    "r3eth1.json"]
```

```

    "r1eth1.json",
    "r2eth0.json",
    "r2eth1.json",
    "r3eth0.json",
    "r3eth1.json"
]
```

Una vez tenemos organizadas las diferentes capturas, vamos a crear un pequeño script de python para realizar la lectura de todas las capturas, filtrar duplicados (ya que un mismo paquete se puede capturar tanto en el nodo origen como en el nodo destino, e incluso también en algún nodo intermedio por el que pase) y guardar los resultados en un nuevo fichero JSON que será el que ya le pasemos a nuestro programa de A-Frame.

Para llevar a cabo esta funcionalidad, en primer lugar nos definiremos una clase con las propiedades necesarias que necesitarán los paquetes en nuestro programa. Definimos también una variable que contendrá el listado de todos los paquetes:

```

class Packet(object):
    src = ""
    dst = ""
    time = ""
    id = 0
    tcp = []
    data = []
    arp = []
    ip = []
    eth = []

allPackets = []
```

Ahora abriremos el fichero JSON con el listado de todas las capturas, e irá recorriendo una por una para primero leer la captura correspondiente y segundo extraer los datos necesarios de cada paquete.

```

for packet in capFileParsed:
    dictionary = {
        'src': packet['_source']['layers']['eth']['eth.src'],
        'dst': packet['_source']['layers']['eth']['eth.dst'],
        'time': packet['_source']['layers']['frame']['frame.time_relative'],
        'id': packet['_source']['layers']['frame']['frame.time_epoch'].split('.')[0],
        'date': packet['_source']['layers']['frame']['frame.time'],
        'tcp': [],
```

```

'data': [],
'arp': [],
'ip': [],
'eth': []
}

```

También incluiremos en un paso posterior la información de los niveles que posea ese paquete. Una vez formado este diccionario con la información del paquete, lo añadiremos al listado de todos los paquetes guardado en la variable allPackets que hemos definido al principio.

```
allPackets.append(dictionary)
```

Ahora crearemos un nuevo fichero JSON donde guardaremos la nueva información de los paquetes. En primer lugar, puesto que hemos ido añadiendo primero todos los paquetes de una captura, luego todos los de la siguiente, etc; no tienen por qué estar ordenados, ya que un paquete perteneciente al tercer fichero de capturas puede haber incurrido en la escena antes de otro paquete perteneciente al primer fichero de capturas. Por ello, los ordenaremos por su variable time, que es el momento exacto en el que se ha capturado ese paquete en la escena real, lo cual nos confirma qué paquetes han ido antes y cuáles han ido después. Para ordenar los paquetes, hacemos uso del método sort en Python que permite ordenar un listado en base a una de sus propiedades:

```
allPackets.sort(key=lambda x: x['date'])
```

Creamos ahora una variable en la que iremos metiendo los ids de esos paquetes que hayan aparecido por primera vez en la escena. De esta manera, cuando llegue otro paquete con el mismo id que otro que ya esté presente en este listado de ids, significa que se trata de un duplicado y no lo añadiremos al listado final de paquetes. No solo nos tendremos que fijarnos en el id, ya que el paquete conserva el mismo id aunque pase por distintos nodos, y en ese caso solo consideraremos duplicados los que van desde el mismo origen hacia el mismo destino. Todo este filtrado de duplicados lo observamos en esta secuencia de comandos:

```

ids = []
idsAndInfo = []
packetsFinal = []
for packet in allPackets:
    if packet['id'] not in ids:
        newRegister = {

```

```

'id': packet['id'],
'src': [],
'dst': []
}
newRegister['src'].append(packet['src'])
newRegister['dst'].append(packet['dst'])

packetsFinal.append(packet)
ids.append(packet['id'])
idsAndInfo.append(newRegister)

else:
    for x in idsAndInfo:
        if x['id'] == packet['id']:
            if packet['src'] not in x['src']
            and packet['dst'] not in x['dst']:
                x['src'].append(packet['src'])
                x['dst'].append(packet['dst'])
                packetsFinal.append(packet)
            break

```

Ahora tendremos en la variable packetsFinal todos los paquetes de la captura real, ordenados por orden de aparición y filtrando duplicados para que no se repita el mismo paquete capturado en nodos diferentes. Ya solo tenemos que plasmar esta variable en un fichero JSON, que será el fichero que pasemos a nuestro proyecto A-Frame como variable para poder representar los paquetes en nuestra escena:

```

json_object = json.dumps(packetsFinal, indent = 2)
f.write(json_object)
print("The json file is created")

```

3.7.3 Preparando la escena para la realidad virtual

Para esta tarea prepararemos la escena para que se pueda visualizar y controlar en realidad virtual con las gafas Oculus Quest.

Para realizar esto, simplemente hay que realizar un sencillo cambio que nos dará acceso al modo inmersivo de la realidad virtual y los controladores del mismo. Deberemos cambiar la entidad de la cámara que teníamos hasta ahora por el siguiente código:

```

<a-entity movement-controls="fly: true">
<a-entity camera position="0 10 45" look-controls></a-entity>

```

```
<a-entity laser-controls="hand: right"></a-entity>
</a-entity>
```

La entidad camera nos especificará dónde nos situaremos en el inicio de la escena. Laser controls nos permite utilizar los controles de las Oculus Quest, en concreto tenemos activado el de la mano derecha. Por último, englobamos estas dos entidades en otra con la propiedad de movement-controls que nos permite desplazarnos libremente por la escena con un efecto de "volar" hacia donde estemos enfocando con las gafas de realidad virtual usando el jostick para movernos a cada una de las direcciones.

3.7.4 Añadiendo modelos visuales a los componentes

Para finalizar este prototipo hemos añadido dos mejoras notables. En primer lugar una pantalla inicial previa en la que podremos seleccionar entre un modo navegador y un modo inmersivo:

Figura 3.35: Modelos visuales a entidades de A-Frame



Inicialmente cargaremos esta pantalla al iniciar la página mediante el componente selector:

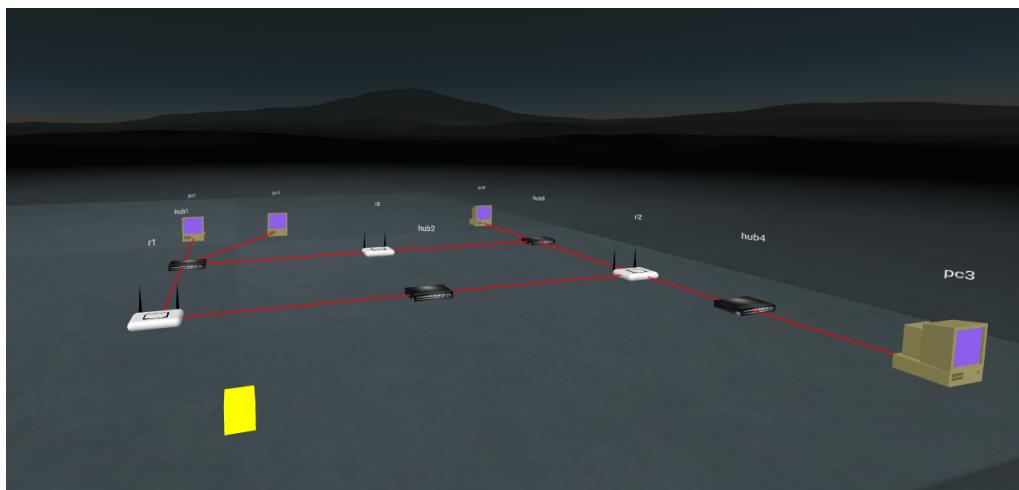
```
<a-entity id="escena" cursor="rayOrigin:mouse" selector>
</a-entity>
```

Dentro del archivo Javascript, definiremos este componente con el que situaremos en pantalla ambos modos: el modo navegador representado con un modelo de un ordenador y

el modo inmersivo representado con unas gafas de realidad virtual. Añadimos manejadores para detectar el click en alguna de las dos opciones, momento en el cual cargaremos la escena correspondiente.

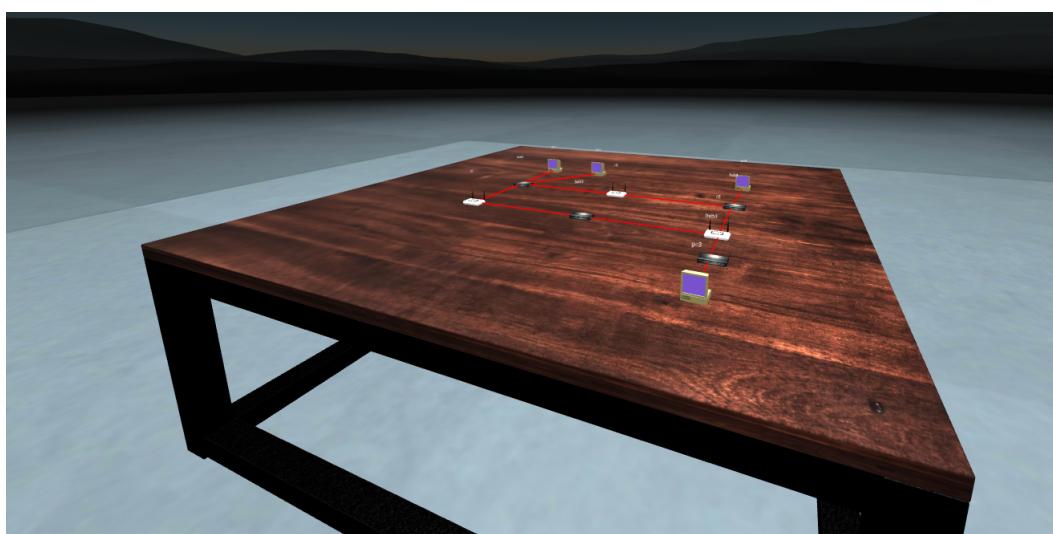
Para el caso navegador, cargaremos el modo conocido hasta ahora. En lo referente a las escalas de los elementos las pasaremos por parámetro al llamar al componente de la escena, ya que las escalas cambiarán respecto al modo inmersivo.

Figura 3.36: Modo navegador



Para este segundo modo, crearemos un nuevo elemento simulando una mesa y situaremos toda la escena encima de la mesa. Como hemos comentado, pasaremos por parámetro la escala de los elementos que será menor que el modo navegador ya que los elementos se sitúan encima de una mesa.

Figura 3.37: Modo inmersivo



```

AFRAME.registerComponent('selector', {
  init: function() {
    scene = this.el

    let inmersiveElement = document.createElement('a-entity');

    inmersiveElement.setAttribute('gltf-model', '#inmersive');
    inmersiveElement.setAttribute('position', '-7 7 30');
    inmersiveElement.setAttribute('scale', '10 10 10');
    scene.appendChild(inmersiveElement);
    inmersiveElement.addEventListener('click', function () {
      desktopText.parentNode.removeChild(desktopText);
      inmersiveText.parentNode.removeChild(inmersiveText);
      desktopElement.parentNode.removeChild(desktopElement);
      inmersiveElement.parentNode.removeChild(inmersiveElement);

      // scene.setAttribute('network', {filename: 'netgui.nkp'});
      scene.setAttribute('inmersiveMode', '');

    });

    let inmersiveText = document.createElement('a-entity');
    inmersiveText.setAttribute('geometry',
    {primitive:'plane',height: 4, width: 8});
    inmersiveText.setAttribute('material', 'color', 'grey');
    inmersiveText.setAttribute('position', { x: -7 , y: 16, z: 30 });
    inmersiveText.setAttribute('look-at', "[camera]");
    inmersiveText.setAttribute('text',
    {width:23, color:'yellow', value: 'Modo inmersivo', align:'center'});
    scene.appendChild(inmersiveText);

    let desktopElement = document.createElement('a-entity');

    desktopElement.setAttribute('gltf-model', '#desktop');
    desktopElement.setAttribute('position', '7 6 30');
    desktopElement.setAttribute('rotation', '0 -90 0');
    // desktopElement.setAttribute('scale', '10 10 10');
    scene.appendChild(desktopElement);
    desktopElement.addEventListener('click', function () {
      desktopText.parentNode.removeChild(desktopText);
      inmersiveText.parentNode.removeChild(inmersiveText);
      inmersiveElement.parentNode.removeChild(inmersiveElement);
      desktopElement.parentNode.removeChild(desktopElement);
      scene.setAttribute('network', {filename: 'netgui.nkp',
        elementsScale: 1, height: 1});

    });

  }
});

```

```

let desktopText = document.createElement('a-entity');
desktopText.setAttribute('geometry', {primitive:'plane',
height: 4, width: 8});
desktopText.setAttribute('material', 'color', 'grey');
desktopText.setAttribute('position', { x: 7 , y: 16, z: 30 });
desktopText.setAttribute('look-at', "[camera]");
desktopText.setAttribute('text', {width:23, color:'yellow',
value: 'Modo navegador', align:'center'});
scene.appendChild(desktopText);

}
);

```

La otra mejora añadida en esta tarea es el uso de modelos, para darle un aspecto más estético a la escena y que mejore la sensación de inmersión en la experiencia de usuario. Para realizar esto hemos recurrido a la página de modelos sketchfab⁵ [17] para buscar y descargar cada uno de los modelos que usaremos en la escena. Todos los modelos utilizados son de descarga gratuita.

Una vez añadidos estos modelos en nuestro proyecto, debemos incluirlos en nuestra escena referenciándolos de la siguiente manera:

```

<a-asset-item id="inmersive" src="inmersive/scene.gltf"></a-asset-item>
<a-asset-item id="desktop" src="desktop/scene.gltf"></a-asset-item>

```

Y para generar el modelo y mostrarlo en la escena le añadimos el atributo con la referencia al modelo deseado:

```
inmersiveElement.setAttribute('gltf-model', '#inmersive');
```

⁵https://sketchfab.com/3d-models?features=downloadable&sort_by=-likeCount

Capítulo 4

Resultados

Como resultados de los apartados anteriores se ha desarrollado una herramienta que permite la lectura de información de NetGUI sobre sus nodos y conexiones y permite representarla en un escenario de realidad virtual. También leeremos las trazas de comunicación entre estos nodos y replicarlas en nuestra escena de forma interactiva.

A continuación voy a describir cómo se utiliza la aplicación a distintos niveles y cómo está construida desde un punto de vista técnico.

4.1 Manual de usuario

4.1.1 Usuario final

Lo que el usuario final se va a encontrar es una escena en realidad virtual de un escenario de red real intercambiando paquetes de información entre cada uno de sus nodos.

Para ello, nos situamos directamente en la escena final ya ejecutada en un navegador o con unas gafas de realidad virtual mediante la url de acceso. Lo primero que nos encontramos es una pantalla para seleccionar el modo en el que queremos disfrutar de la escena.

Figura 4.1: Pantalla de selección de modos

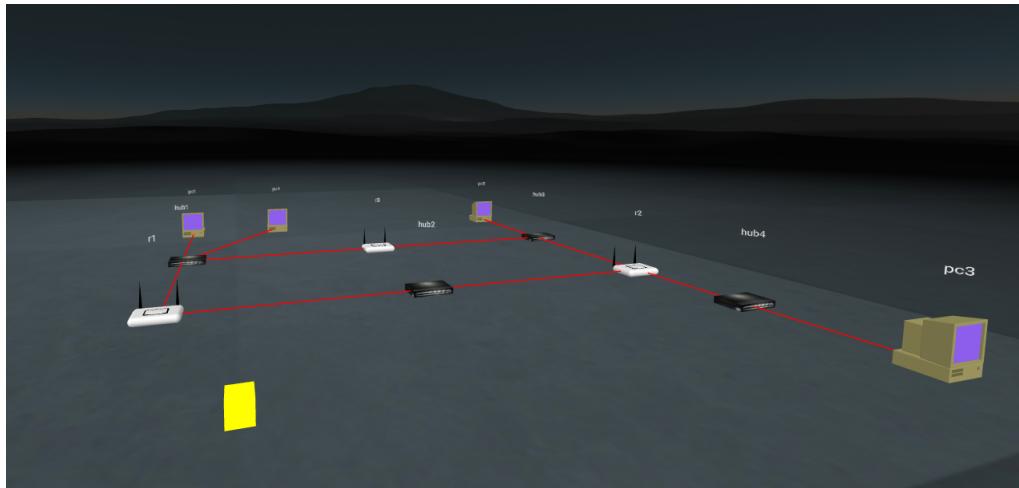


Podremos elegir entre un modo inmersivo del que podremos disfrutar clicando en la figura de las gafas de realidad virtual ya sea con el ratón si estamos visualizando la aplicación desde un navegador o con el mando controlador si lo hacemos mediante unas gafas especializadas o el modo navegador al que podremos acceder de igual modo pulsando en la figura del ordenador.

A grandes rasgos ambos modos tienen un funcionamiento similar, simplemente cambia un poco la manera de visualizar la escena.

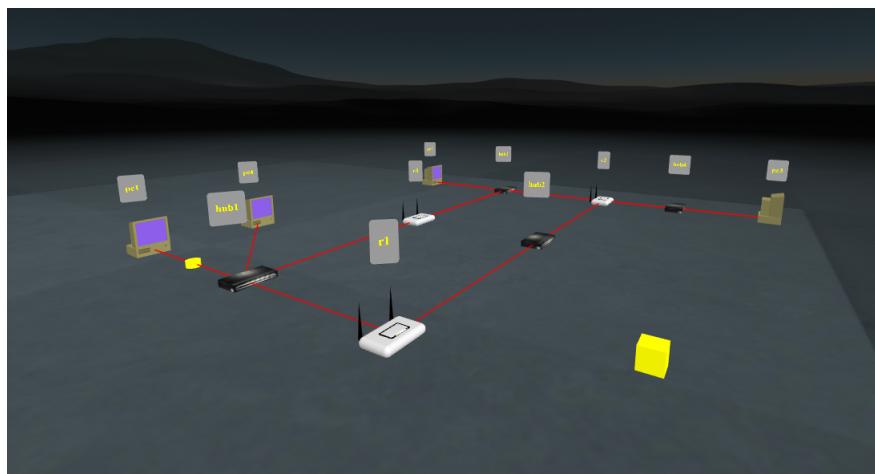
Una vez hemos seleccionado alguno de los modos, veremos la topología de un escenario real de NetGUI que se carga automáticamente. En este escenario podremos distinguir los diferentes nodos que interactúan en la escena distinguiendo entre máquinas (pcs), hubs y routers que se representan con diferentes modelos en pantalla y sobre ellos un pequeño cartel con el nombre asociado a ese nodo en el escenario de NetGUI.

Figura 4.2: Modo navegador



También podemos distinguir un botón representado con un cubo amarillo que será el que inicie las animaciones de los paquetes. Si pinchamos en él (ya sea con el ratón o con el mando de realidad virtual en cada caso) empezaremos a ver los diferentes paquetes transitando por la escena e intercambiando información entre cada uno de los nodos. Estas animaciones de los paquetes no son arbitrarias sino que previamente se ha realizado una lectura de una serie de ficheros con capturas reales en un escenario real.

Figura 4.3: Modo navegador con paquetes en tránsito

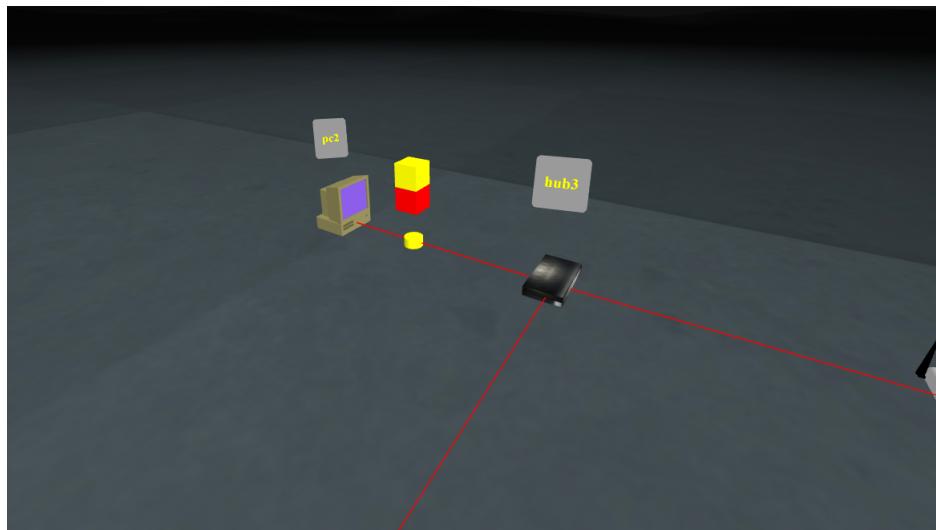


Volviendo al botón de inicio de la animación, una vez los paquetes han empezado a moverse, si volvemos a pulsar en él, las animaciones se detendrán, y podremos reanudarla de nuevo volviendo a pulsar. Es decir, este botón funciona como un controlador de inicio/detención/reanudación de la animación de la escena.

Si ahora nos centramos en los paquetes, cada uno tendrá el color asociado al nivel de información más alto que transporte, definiendo un color diferente para cada nivel (rojo para

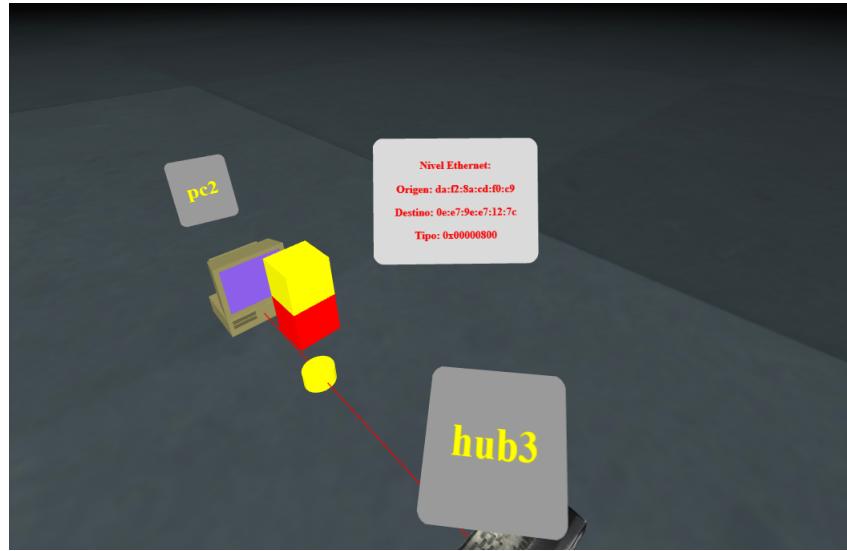
Ethernet, verde para ARP, amarillo para IP y azul para TCP). Estos también serán clicables. Al hacerlo, veremos sobre el propio paquete una serie de cubos que representan cada uno de los niveles de información que transporta también coloreados con su color correspondiente.

Figura 4.4: Desglose de niveles de información de un paquete



Si situamos el cursor sobre alguno de estos cubos, vemos como su tamaño aumenta significativamente para dar al usuario a entender que se puede clicar en este nivel. Al hacerlo veremos un pequeño panel a la derecha con la información más relevante de dicho nivel de información del paquete. Esta información se representa con un color de letra igual al color asociado al nivel correspondiente siguiendo la asociación presentada anteriormente. Por ejemplo si pinchamos en el el cubo rojo correspondiente al nivel de información Ethernet veremos un panel escrito con letras rojas donde se muestra información sobre la dirección ethernet origen y destino entre otros.

Figura 4.5: Muestra de información de un nivel de comunicación



Para ocultar de nuevo esta información deberemos volver a pulsar en el selector de nivel correspondiente al que tiene su información desplegada en el panel, ya que si por ejemplo se nos está mostrando información sobre el nivel Ethernet y pinchamos en el selector de nivel IP, el panel no se cerrará sino que cambiará su contenido al nuevo nivel seleccionado.

Además, la escena incluye una serie de sonidos localizados para cada uno de los elementos clicables de la escena, ya sea el botón de inicio, los diferentes paquetes o cada uno de los niveles de información.

4.1.2 Composición de la escena

Para la composición de la escena hemos hecho uso de la tecnología A-Frame que como ya hemos comentado en otros apartados es una muy buena plataforma para la construcción de escenas en realidad virtual. Esta tecnología se basa en HTML y Javascript.

En nuestro HTML, tenemos por un lado el apartado de head donde estableceremos cada uno de los ficheros externos que nos harán falta para la composición de nuestra escena: el fichero para poder hacer uso de A-Frame, fichero para hacer uso de componentes o funcionalidades especiales, el fichero Javascript donde definiremos toda la lógica de nuestra aplicación ...

```
<head>
  ...
  <script src="index.js"></script>
```

```
...
</head>
```

A continuación encontramos el body, que es donde irán todos los elementos que verá el usuario. Dentro del body tenemos por un lado la escena, donde situaremos cada uno de los elementos visuales que se representarán y por otro lado una serie de plantillas de las que más tarde haremos usos para representar letreros y carteles en la escena.

```
<body>
  <a-scene>
    ...
  </a-scene>

  <div id="htmltemplates">
  </div>

</body>
```

Dentro de la parte de la escena nos encontramos con un apartado de assets, donde situaremos cada uno de los elementos de audio, imágenes o modelos que utilizaremos en nuestra aplicación.

```
<a-assets>
  
  ...
  <audio id="packetIn" crossorigin="anonymous" src="packetIn.mp3"></audio>
  ...
  <a-asset-item id="inmersive" src="inmersive/scene.gltf"></a-asset-item>
  ...
</a-assets>
```

Después procedemos a referenciar cada uno de los elementos o entidades que se sitúan en la escena. Haciendo uso de las imágenes definidas en el apartado anterior situamos un suelo y un fondo en la escena para ambientar el aspecto visual.

```
<a-cylinder id="ground" src="#groundTexture" radius="300" height="0.1"></a-cylinder>
<a-sky id="background" src="#skyTexture" theta-length="90" radius="300"></a-sky>
```

También situamos la cámara que será el punto desde el que el usuario observe la escena cuando inicie la aplicación. Esta entidad varía en función de si estamos en el modo navegador:

```
<a-camera position="0 10 45">
```

o ejecutándolo en realidad virtual:

```
<a-entity movement-controls="fly: true">
  <a-entity camera position="0 10 45" look-controls></a-entity>
  <a-entity cursor="rayOrigin:mouse"></a-entity>
  <a-entity laser-controls="hand: right"></a-entity>
</a-entity>
```

Ya por último nos encontramos una entidad con la referencia al componente selector:

```
<a-entity id="escena" cursor="rayOrigin:mouse" selector>
</a-entity>
```

Este componente lo definimos en nuestro archivo Javascript, y a partir del cual se realizará toda la gestión de la escena y gestionando de forma dinámica el resto de elementos que pueden aparecer o desaparecer de la escena.

Para insertar en la escena elementos seguiremos el siguiente procedimiento. En primer lugar tendremos que guardar en una variable la escena para poder insertar más adelante entidades en ella. Creamos estas entidades y las dotamos de atributos (posición, color, modelos, otros componentes ...) y si queremos de eventos mediante addEventListener. Finalmente, volviendo a la variable de la escena y haciendo uso del appendChild podremos insertar en la escena estos elementos que creamos de forma dinámica. Aquí vemos un ejemplo cómo insertar en la escena una entidad dotada de atributos y eventos:

```
scene = this.el

let inmersiveElement = document.createElement('a-entity');

immersivElement.setAttribute('gltf-model', '#inmersive');
immersivElement.setAttribute('position', '-7 7 30');
immersivElement.setAttribute('scale', '10 10 10');
scene.appendChild(inmersiveElement);
immersivElement.addEventListener('click', function () {
  desktopText.parentNode.removeChild(desktopText);
  inmersiveText.parentNode.removeChild(inmersiveText);
  desktopElement.parentNode.removeChild(desktopElement);
  immersivElement.parentNode.removeChild(inmersiveElement);
  scene.setAttribute('inmersiveMode', '');
});
```

De esta manera, según vamos recorriendo el código de la aplicación observamos cómo todos los elementos se van creando e insertando de forma dinámica en la escena, ya sean nodos, paquetes, niveles, paneles con información ...

Si por el contrario lo que queremos es eliminar de la escena alguna entidad o componente que queramos que deje de ser visible para el usuario, hacemos uso del remove child para desanclarlo de su elemento padre y por tanto de la escena.

```
desktopElement.parentNode.removeChild(desktopElement);
});
```

4.1.3 Proceso completo

Ya hemos explicado cómo funciona nuestra aplicación tanto a nivel de usuario como en cuanto a la composición de la escena, pero falta por explicar cuál es el proceso completo a seguir para probar nuestras propias escenas.

Partimos de un escenario de NetGUI que podremos haber exportado o bien haberlo creado nosotros mismos. Este escenario tiene un archivo con la extensión nkp con la información de la topología de la escena, el cual deberemos copiar en nuestro proyecto y pasarlo como parámetro en la escena para que sepa que es el archivo del que tiene que leer para componer nuestro escenario.

Para capturar el tráfico dentro de NetGUI, deberemos iniciar cada una de las máquinas que componen la escena e iniciar una captura en cada una de sus interfaces. Esto lo podremos hacer mediante el comando:

```
tcpdump -i NOMBRE_INTERFAZ
```

Estas capturas se guardan con formato .cap y para pasarlas a formato .json (el cual nos interesa para el desarrollo de nuestro programa) deberemos abrirnos con el programa Wireshark, y en el menú superior, donde pone Archivo, veremos la opción de exportar la captura como JSON.

Una vez hemos convertido todas las capturas y las hemos almacenado localmente, las deberemos copiar en el directorio de nuestro proyecto y llenar el archivo caps.json con los nombres que le hemos dado a cada una de las capturas tomadas.

```
[  
    "pc1.json",
```

```

"pc2.json",
"pc3.json",
"pc4.json",
"r1eth0.json",
"r1eth1.json",
"r2eth0.json",
"r2eth1.json",
"r3eth0.json",
"r3eth1.json"
]
```

También debemos llenar el archivo machineNames.json asociando a cada máquina de la escena su correspondiente nombre ethernet, ya que tanto para la lectura de la topología de red como para la de los paquetes nos hará falta saber estas direcciones. En el caso de que una máquina tenga varias interfaces de red, tendremos que poner todas ellas dentro del fichero, como ocurre con nuestro caso de la máquina r2 en el siguiente ejemplo:

```

[
{
    "pc1": "a6:70:63:a3:ca:4a",
    "pc2": "da:f2:8a:cd:f0:c9",
    "pc3": "12:b5:d4:0b:46:a9",
    "pc4": "da:1f:92:2b:23:68",
    "r1": "6e:99:b4:dd:2b:01",
    "r2": "36:95:2e:79:23:b3",
    "r3": "62:f9:1b:ea:6b:34"
},
{
    "r1": "26:e9:06:70:1c:96",
    "r2": "f2:d8:5c:29:9b:30",
    "r3": "0e:e7:9e:e7:12:7c"
},
{
    "r2": "8a:be:9f:6f:5a:30"
}
]
```

Ahora abrimos una terminal en nuestro equipo y ejecutamos el programa unifycaps.py. Esto lo haremos situándonos en el directorio del proyecto y escribiendo

```
python3 unifycaps.py
```

siempre que tengamos el paquete de python instalado localmente (En caso contrario previamente deberemos instalarlo ejecutando pip install python3).

Este programa se encargará de leer cada una de las capturas tomadas y unificarla en una sola captura, ordenando los paquetes por orden de aparición y filtrando aquellos que están duplicados. El resultado se guardará en el archivo new-file.json que será el que tome nuestra aplicación para la lectura y representación de los paquetes.

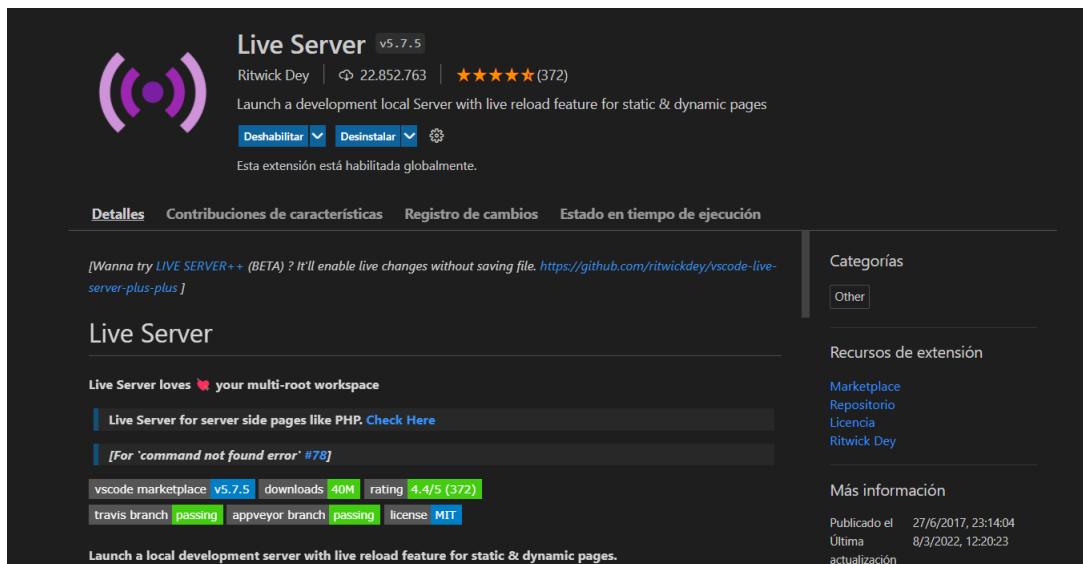
Ahora que ya tenemos todos los ficheros que necesita el programa para funcionar, debemos ejecutar un servidor local para poder empezar a disfrutar de nuestro programa. Esto se puede hacer abriendo una terminal en el equipo, situándonos en el directorio del proyecto y ejecutando

```
python3 -m http.server PORT
```

siendo PORT el número de puerto donde queramos probar nuestro programa en un entorno local. Si ahora accedemos a dicha dirección http://localhost:PORT podremos empezar a disfrutar de nuestro programa.

Existen otras alternativas para probar nuestra aplicación. Yo para desarrollar de forma más dinámica me he descargado una extensión en Visual Studio Code (el programa que uso para desarrollar el código) llamada Live Server que me permite ejecutar desde la propia consola un servidor y poder ir viendo en tiempo real los cambios que se puedan ir haciendo en el código.

Figura 4.6: Extensión Live Server para Visual Studio Code



4.2 Detalles de implementación

En este apartado vamos a profundizar en cómo está escrito el código y cuál es el funcionamiento del programa principal.

Este proyecto está basado en el uso de componentes, que son fragmentos de código reutilizables y modulares que asociamos a una entidad para dar apariencia, comportamiento y/o funcionalidad.

Mediante estos componentes hemos definido cada uno de los elementos que aparecerá en la escena, veremos cómo se componen y cómo se relacionan entre ellos.

Primero tenemos el componente selector, desde donde elegiremos en qué modo queremos visualizar nuestra escena. Si la opción elegida es la inmersiva llamaremos al componente `immersiveMode`, que añadirá una serie de entidades correspondientes a este modo. Desde aquí o desde el componente selector si hemos elegido el modo navegador llamaremos al componente `network`, que es el encargado de construir y representar los nodos y sus conexiones. Por último desde el componente `network` llamaremos a los componentes `packet`, en los cuales se generan y animan los diferentes paquetes de comunicación que protagonizan la escena.

4.2.1 Componente selector

Este componente es el encargado de componer la pantalla inicial de selección de modos. No tiene argumentos de entrada y en su cuerpo se crean los diferentes elementos de selección de modos en la aplicación.

Figura 4.7: Componente Selector



4.2.2 Componente immersiveMode

Este componente será el encargado de construir los elementos adicionales de la escena que no posee el modo navegador, en concreto la mesa sobre la que se apoya la escena y varias entidades de iluminación. Tampoco tiene ningún parámetro de entrada.

```
scene = this.el

let immersiveElement = document.createElement('a-entity');

immersiveElement.setAttribute('gltf-model', '#table');
immersiveElement.setAttribute('position', '2 0 3');
immersiveElement.setAttribute('scale', '15 10 15');
immersiveElement.setAttribute('rotation', '0 -90 0');
scene.appendChild(immersiveElement);

let light = document.createElement('a-entity');

light.setAttribute('light', {type: 'ambient', color: '#FFF'});
scene.appendChild(light);

let ambientLight = document.createElement('a-entity');

ambientLight.setAttribute('light', {color: '#FFF', intensity: '1.5'});
```

```

ambientLight.setAttribute('position', '0 30 0');
scene.appendChild(ambientLight);

scene.setAttribute('network',
{filename: 'netgui.nkp', elementsScale: 4, height: 6, connectionscolor: 'blue'});

```

4.2.3 Componente network

Este será el componente principal de nuestro proyecto sobre el cual construiremos toda la escena de la red. Tiene un parámetro de entrada que es el fichero desde el cual tiene que leer la topología de red de NetGUI que se tiene que representar en nuestro escenario.

```

schema: {
  filename: {type: 'string', default: ''},
},

```

En primer lugar este componente leerá el fichero que le hemos pasado por parámetro y procesará su contenido para guardar en una variable todos los nodos. Posteriormente se llamará a una función que se encarga de construir cada nodo y colocarlo en su posición correspondiente de la escena.

```

nodeList = [];
data = this.data
file = data.filename
request = new XMLHttpRequest();
request.open('GET', file);
request.responseType = 'text';
request.send();
scene = this.el
request.onload = function() {
  response = request.response;
  response.split('<nodes>')
  nodes = response.split('position')
  setNodes(nodes, nodeList, data)
}

```

A continuación abriremos el fichero con los nombres Ethernet asociados a cada máquina de la escena y lo guardaremos dentro del cuerpo de cada nodo para posteriormente poder establecer relaciones entre nodos (ya que en el fichero con la correspondiente topología de red las conexiones de máquinas vienen asociadas mediante este nombre de dirección ethernet).

```

machineNamesFile = 'machineNames.json'
requestMachineNames = new XMLHttpRequest();
requestMachineNames.open('GET', machineNamesFile);
requestMachineNames.responseType = 'text';
requestMachineNames.send();
requestMachineNames.onload = function() {
    response = requestMachineNames.response;
    responseParse = JSON.parse(response);

    for (const interface in responseParse) {
        for (const currentNode in responseParse[interface]) {
            nodeList.find(
                o => o.name === currentNode).machineName.push(
                    responseParse[interface][currentNode])
        }
    }
}

```

Ahora podremos establecer las conexiones entre nodos y representarlas en la escena mediante la función setConnectionsLinks.

```

connections = nodesInfo[1].split('link')
connectionsLinks = []

finalConnectionsLinks =
setConnectionsLinks(connections, connectionsLinks, nodeList, data)

```

Acto seguido, leemos el fichero con las capturas de cada uno de los paquetes y llamamos a la función readPackets para que se procese y se cree cada traza de comunicación que tiene lugar en la escena.

```

filePackets = 'new_file.json'
requestPackets = new XMLHttpRequest();
requestPackets.open('GET', filePackets);
requestPackets.responseType = 'text';
requestPackets.send();
requestPackets.onload = function() {
    response = requestPackets.response;
    responseParse = JSON.parse(response);

    packets = readPackets(responseParse)
}

```

Por último creamos e insertamos en la escena el botón encargado del inicio de las animaciones de los paquetes y llamamos a la función de animatePackets para empezar todo el proceso de simulación de un escenario de comunicación.

```

let startButton = document.createElement('a-box');

startButton.setAttribute('position', '0 1 30');
startButton.setAttribute('color', 'yellow');
startButton.setAttribute('id', 'startButton');
startButton.setAttribute('sound', {on: 'click', src: '#playPause', volume: 5});
scene.appendChild(startButton);

animatePackets(packets, finalConnectionsLinks, data)

```

4.2.4 Componente packet

Se trata del último componente que conforma la escena y corresponde a cada uno de los paquetes de comunicación que se intercambian en el escenario. Este posee una gran variedad de parámetros de entrada:

```

schema: {
    xPosition: {type: 'number', default: 0},
    yPosition: {type: 'number', default: 1},
    zPosition: {type: 'number', default: 0},
    toXPosition: {type: 'string', default: ''},
    toYPosition: {type: 'string', default: ' 1 '},
    toZPosition: {type: 'string', default: ''},
    duration: {type: 'number', default: 0},
    elementsScale: {type: 'number', default: 0},
    id: {type: 'number', default: 0},
    class:{type: 'string'},
    start:{type: 'number', default: 0},
    ip:{default: null},
    eth:{default: null},
    arp:{default: null},
    dataInfo:{default: null},
    tcp:{default: null},
},

```

- xPosition: Indica la posición en el eje x donde se inicia el paquete.
- yPosition: Indica la posición en el eje y donde se inicia el paquete.
- zPosition: Indica la posición en el eje z donde se inicia el paquete.
- toXPosition: Indica la posición en el eje x donde finaliza el paquete.
- toYPosition: Indica la posición en el eje y donde finaliza el paquete.

- toZPosition: Indica la posición en el eje z donde finaliza el paquete.
- duration: Lo que tarda el paquete en realizar su animación desde el nodo origen al nodo destino.
- elementsScale: La escala en tamaño que tendrá el paquete. Variará dependiendo de si estamos en modo inmersivo o navegador.
- id: Un identificador numérico asociado a cada paquete. Son identificadores numéricos en orden ascendente por orden de aparición en escena, siendo el identificador 0 el correspondiente al primer paquete en aparecer en escena.
- class: Una clase asociada a cada paquete. Esto lo usaremos a la hora de guardarnos en una variable todos los paquetes que comparten esa clase mediante el método getElementsByClassName.
- start: Indica el momento exacto en el que un paquete de iniciar su animación.
- ip: La información que posea el paquete del nivel de comunicación IP.
- eth: La información que posea el paquete del nivel de comunicación ETH.
- arp: La información que posea el paquete del nivel de comunicación ARP.
- dataInfo: La información que posea el paquete del nivel de comunicación dataInfo.
- tcp: La información que posea el paquete del nivel de comunicación TCP.

Dentro del componente, definiremos unos manejadores que detectarán cuando se pulse en el botón de inicio de la escena o bien cuando se pulse/reanude la escena. Esto irá cambiando la variable animationStatus al estado correspondiente.

```
var startButton = document.querySelector('#startButton');
var packets = document.getElementsByClassName('packetClass');

var animationStatus = 'animation-starts'
startButton.addEventListener('click', function () {

    for (var a=0; a< packets.length; a++) {
        packets[a].emit(animationStatus,null,false)
    }

    switch(animationStatus) {
        case 'animation-starts':
            setInterval(startAnimation, 1000);
            animationStatus = 'move-pause'
            break
    }
})
```

```

        case 'move-resume':
            animationStatus = 'move-pause'
            break
        case 'move-pause':
            animationStatus = 'move-resume'
            break
    }
});
```

También dentro del componente definiremos la función startAnimation que se encargará de crear y animar cada uno de los paquetes que entran en escena. Para que esta función se ejecute, debemos comprobar que el estado de la animación es activo y que ha llegado el momento de que el paquete actual entre en escena mediante el parámetro start que hemos comentado anteriormente.

```

function startAnimation() {
    if (animationStatus == 'move-pause') {
        if (i == Math.ceil(packetParams.start/1000)) {
```

Si ambos requisitos se cumplen, entramos en la función donde lo primero que hacemos es detectar el nodo del que sale el paquete mediante sus parámetros de posición (que deben coincidir con los de dicho nodo) para añadir una pequeña animación a la máquina de la que parte la traza.

```

// Búsqueda del nodo del que sale el paquete
let nodePositionFrom = ''
if(Number.isInteger((packetParams.xPosition * 15)*packetParams.elementsScale)){
    nodePositionFrom +=
    parseFloat(
        (packetParams.xPosition * 15)*packetParams.elementsScale).toFixed(1).toString()
}else{
    nodePositionFrom +=
    ((packetParams.xPosition * 15)*packetParams.elementsScale).toString()
}
nodePositionFrom += ','
if(Number.isInteger((packetParams.zPosition * 15)*packetParams.elementsScale)){
    nodePositionFrom +=
    parseFloat(
        (packetParams.zPosition * 15)*packetParams.elementsScale).toFixed(1).toString()
}else{
    nodePositionFrom +=
    ((packetParams.zPosition * 15)*packetParams.elementsScale).toString()
```

```

}

nodeAnimation = nodeList.find(o => o.position === nodePositionFrom)

// Añadir animación en dicha máquina
// La animación será diferente en función del tipo de máquina
var nodeFromAnimation = document.getElementById(nodeAnimation.name);

if(nodeAnimation.name.startsWith('pc')){
    nodeFromAnimation.setAttribute('animation', {property: 'scale',
        from: {x: 0.012/packetParams.elementsScale,
            y: 0.012/packetParams.elementsScale,
            z: 0.012/packetParams.elementsScale},
        to: {x: 0.006/packetParams.elementsScale,
            y: 0.006/packetParams.elementsScale,
            z: 0.006/packetParams.elementsScale},
        loop: '2', dur: '1000', easing: 'linear' })
}

...

```

Ahora creamos el elemento del paquete y le dotamos de ciertos atributos como color en función de los niveles que transporte, posición, clase ...

```

packet.setAttribute('geometry',
{primitive: 'cylinder', 'height': 0.4/packetParams.elementsScale,
radius: 0.4/packetParams.elementsScale });
let packetColor = ''
if(packetParams.tcp){
    packetColor = 'blue'
}else if(packetParams.ip){
    packetColor = 'yellow'
}else if(packetParams.arp){
    packetColor = 'green'
}else{
    packetColor = 'red'
}
packet.setAttribute('material', 'color', packetColor);
packet.setAttribute('position', { x: packetParams.xPosition,
y: packetParams.yPosition, z: packetParams.zPosition });
packet.setAttribute('class', packetParams.class);
packet.setAttribute('sound', {src: '#packetIn', volume: 5, autoplay: "true"});
packet.setAttribute('id', packetParams.id);

```

También le damos al paquete la propiedad de la animación para que se mueva de su nodo origen a su nodo destino:

```
var packet_move = document.getElementById(packetParams.id);

packet_move.setAttribute('animation', {
    property: 'position',
    to: packetParams.toXPosition
    + packetParams.toYPosition + packetParams.toZPosition,
    dur: packetParams.duration,
    easing: 'linear',
    pauseEvents: 'move-pause',
    resumeEvents: 'move-resume'
});

});
```

Creamos una plantilla HTML con un identificador igual al del paquete y lo añadimos a la escena. Lo usaremos más adelante para mostrar en él la información relativa al nivel de información seleccionado.

```
var htmltemplates = document.getElementById("htmltemplates");
var newSectionTemplate = document.createElement("section");
newSectionTemplate.style = "display: inline-block; background: #EEEEEE;
color: purple; border-radius: 1em; padding: 1em; margin:0;";
newSectionTemplate.id = packetParams.id + '-template'
htmltemplates.appendChild(newSectionTemplate);
```

También asociamos al paquete el elemento donde irá dicha información haciendo uso de la plantilla que acabamos de crear:

```
let newInfoText = document.createElement('a-entity');

newInfoText.setAttribute('position', { x: 5 , y: 3, z: 0 });
newInfoText.setAttribute('look-at', "[camera]");
newInfoText.setAttribute('visible', false);
newInfoText.setAttribute('scale', {x: 20, y: 20, z: 20});
newInfoText.setAttribute('isPoster', true);

packet.appendChild(newInfoText);
```

Ahora comprobamos qué niveles de información transporta el paquete para irlos guardando en la variable levels:

```
if(packetParams.eth){
    const ethInfo = {
```

```

        eth: packetParams.eth
    }
    levels = Object.assign(levels,ethInfo);
}

```

Posteriormente, comprobaremos nivel por nivel para crear en caso de que exista el selector correspondiente que situaremos encima del paquete aunque no se mostrará hasta que no se haga click en el paquete. A este selector irán asociados unos manejadores que nos mostrarán en el cartel que hemos creado antes la información desplegada correspondiente a ese nivel. Distinguiremos 3 casos: cuando aún no hay información desplegada, en cuyo caso mostraremos la del nivel que acabamos de pulsar; cuando hay información de otro nivel diferente al que acabamos de pulsar, en cuyo caso lo cambiaremos a la información actual; y cuando la información desplegada es la misma que la del nivel que acabamos de pulsar, en cuyo caso cerramos la información.

```

if(levels.hasOwnProperty('eth')){
    index = Object.keys(levels).findIndex(item => item === 'eth')

    let newEthBox = document.createElement('a-box');
    newEthBox.setAttribute('position', { x: 0, y: 2 + (index), z: 0 });
    newEthBox.setAttribute('color', 'red');
    newEthBox.setAttribute('visible', false);
    packet.appendChild(newEthBox);

    newEthBox.addEventListener('mouseenter', function () {
        newEthBox.setAttribute('scale', {x: 1.2, y: 1.2, z: 1.2});
    });
    newEthBox.addEventListener('mouseleave', function () {
        newEthBox.setAttribute('scale', {x: 1, y: 1, z: 1});
        newEthBox.removeAttribute('animation');
        newEthBox.setAttribute('rotation', {x: 0, y: 0, z: 0});
    });
    newEthBox.addEventListener('click', function () {
        if(newEthBox.getAttribute('isVisible')){
            let infoText = '<p>Nivel Ethernet:</p><p>Origen: ' +
                + packetParams.eth['eth.src'] + '</p><p>Destino: ' +
                + packetParams.eth['eth.dst'] + '</p><p>Tipo: ' +
                + packetParams.eth['eth.type'] + '</p>'
            if(closeInfo == true){
                closeInfo = false
                actualInfoShown = 'eth'
                newInfoText.setAttribute('visible', true);
                newInfoText.removeAttribute('html');
                var textTemplate =

```

```

document.getElementById(packetParams.id + '-template');
textTemplateContent =
'<h1 style="padding: 0rem 1rem; font-size: 1rem; font-weight: 700;
text-align: center; color: red">' + infoText + '</h1>'
textTemplate.innerHTML = textTemplateContent;
newInfoText.setAttribute('html',
'#' + packetParams.id + '-template');
newInfoText.setAttribute('visible', true);
newEthBox.removeAttribute('sound');
newEthBox.setAttribute('sound',
{src: '#showLevels', volume: 5, autoplay: "true"});
}else if(closeInfo == false && actualInfoShown == 'eth'){
    actualInfoShown = ''
    newInfoText.setAttribute('visible', false);
    newEthBox.removeAttribute('sound');
    newInfoText.removeAttribute('html');
    newEthBox.setAttribute('sound',
{src: '#showLevels', volume: 5, autoplay: "true"});
}else if(closeInfo == false && actualInfoShown != ''){
    closeInfo = false
    actualInfoShown = 'eth'
    newInfoText.removeAttribute('html');
    var textTemplate =
document.getElementById(packetParams.id + '-template');
textTemplateContent =
'<h1 style="padding: 0rem 1rem; font-size: 1rem; font-weight: 700;
text-align: center; color: red">' + infoText + '</h1>'
textTemplate.innerHTML = textTemplateContent;
newInfoText.setAttribute('html',
'#' + packetParams.id + '-template');
newEthBox.removeAttribute('sound');
newEthBox.setAttribute('sound', {src:
'#showLevels', volume: 5, autoplay: "true"});
}
}else {
    notValid = true
}
});
}

```

Ya solo queda detectar el evento de click sobre el paquete, momento en el cual comprobaremos si tiene visible los selectores de niveles (en cuyo caso los ocultaremos) o si no (en cuyo caso los mostraremos por pantalla).

```
packet.addEventListener('click', function () {
```

```
if(notValid){
    notValid = false
} else{
    if(closeInfo){
        for (var a=0; a< packet.children.length; a++) {
            if(packet.children[a].hasAttribute('isVisible')){
                packet.children[a].setAttribute('visible', false);
                packet.children[a].removeAttribute('isVisible');
                packet.removeAttribute('sound');
                packet.setAttribute('sound',
                    {src: '#showInfo', volume: 5, autoplay: "true"});
            }else{
                if(!packet.children[a].hasAttribute('isPoster')){
                    packet.children[a].setAttribute('isVisible', null);
                    packet.children[a].setAttribute('visible', true);
                    packet.removeAttribute('sound');
                    packet.setAttribute('sound',
                        {src: '#showInfo', volume: 5, autoplay: "true"});
                }
            }
        }
    }

    if(actualInfoShown == ''){
        closeInfo = true
    }
}

});
```

Capítulo 5

Conclusiones y trabajos futuros

5.1 Consecución de objetivos

A lo largo de esta sección vamos a revisar cómo hemos conseguido los objetivos propuestos al principio de esta memoria.

En cuanto al objetivo general de crear una herramienta de visualización de trazas de NetGUI en realidad virtual hemos desarrollado desde cero la aplicación necesaria para leer la topología de un escenario real, interpretar las capturas de sus paquetes y animarlos en la escena mostrando la información de cada uno de sus niveles.

En lo referente a los objetivos específicos, también hemos podido cumplir cada uno de los propuestos en la sección introductoria. Una demo para probar el proyecto está disponible gracias a una url única que hemos generado a través de la herramienta que nos ofrece github de pages, mediante la cual somos capaces de redireccionar cualquiera de nuestros repositorios hacia una url segura donde cualquier usuario podrá probar desde un navegador el contenido que se encuentra en dicho repositorio.

Siguiendo el estándar del lenguaje A-Frame como nos propusimos al principio hemos sido capaces de replicar en realidad virtual la topología de un escenario real de NetGUI leyendo el archivo que contiene el esquemático con sus nodos y conexiones. Podremos por tanto generar cualquier escenario desde la plataforma de NetGUI, exportar su arquitectura y pasándosela a nuestro programa lograremos ver el escenario en realidad virtual.

Y no solo replicaremos la topología, sino la representación de trazas de información entre cada uno de los nodos que componen la escena extraídas de los programas de Wireshark y NetGUI como fuente de datos gracias a la unificación en un fichero JSON de cada una de las capturas tomadas en las interfaces de todas las máquinas reales y la lectura de dicho

fichero por parte del programa para interpretar su contenido y animar los paquetes en la escena.

Cada paquete a su vez contendrá información de los distintos niveles que transporta, la cual podremos ver desplegada pulsando en cada uno de ellos que se representan mediante cubos situados sobre el paquete. Esta información nos ofrece conocimiento de niveles como Ethernet, IP, ARP, TCP y sus datos asociados.

Además, hemos podido dotar al escenario de una visión realista haciendo usos de los diferentes modelos gltf adheridos a entidades y componentes consiguiendo una mayor sensación de inmersión y calidad.

Por último hemos conseguido una gran interactividad por parte del usuario con la escena, pudiendo navegar libremente por el escenario, desplegar los distintos niveles asociados al paquete y desglosar su información, todo ello desde un entorno 3D en realidad virtual.

En definitiva, la consecución de todos estos objetivos dota al proyecto de un enorme valor por haber realizado el trabajo desde cero sin ningún tipo de conocimiento previo acerca de la tecnología A-frame y el funcionamiento de sus elementos y componentes. Además, el hecho de que no haya ninguna referencia previa sobre proyectos de este estilo y este sea prácticamente el primer proyecto que trata un tema como la representación de trazas de NetGUI en realidad virtual añade un enorme valor tanto personal como profesional a este trabajo.

5.2 Aplicación de lo aprendido

Para la consecución de este trabajo he podido aplicar numerosos conocimientos que he adquirido a lo largo de la carrera. Sin duda aquellas asignaturas relacionadas con la programación han sido fundamentales para el desarrollo del proyecto, ya que este se basa en la escritura y ejecución de un código software. Las asignaturas de Informática I e Informática II son las que sentaron las bases para entender la estructuración y funcionamiento de un lenguaje de programación y las que me despertaron el gusanillo en este mundo del desarrollo. También gracias a la asignatura de Gráficos y Visualización 3D tuve un mayor acercamiento a modelar escenarios visuales que han sido de gran ayuda a la hora de manejar elementos en realidad virtual. Por último, las asignaturas de Construcción de Servicios y Aplicaciones Audiovisuales en Internet y Laboratorio de Tecnologías Audiovisuales en la Web que se centran más en el desarrollo web en HTML5 y Javascript desde el punto de vista del cliente y del servidor respectivamente han sido las más importantes de cara a la realización de este trabajo, ya que estos lenguajes han sido los seleccionados para construir mi programa, basados en la tecnología de A-Frame.

Por otro lado y ya más relacionado con el contenido de la práctica y no tanto con su desarrollo debemos hablar de asignaturas de redes, ya que mi práctica se fundamenta en estos conceptos y en la visualización de trazas de un escenario de comunicación. Entre ellas debemos destacar las de Arquitectura de Internet y Sistemas Telemáticos para Medios Audiovisuales por ser las primeras en las que veíamos conceptos de redes y en las que usamos programas como NetGUI o Wireshark, que han sido fundamentales para la realización del proyecto. También en un mayor nivel de profundidad, asignaturas como Tecnologías de Televisión en Internet, Protocolos para la Transmisión de Audio y Vídeo en Internet y Codificación de Información Audiovisual han aportado sus conocimientos de redes para el desarrollo del trabajo.

5.3 Lecciones aprendidas

A lo largo de la realización de mi trabajo de fin de grado he adquirido los siguientes conocimientos.

1. Aprender a trabajar en un entorno ágil haciendo uso de la metodología SCRUM, orientada a proyectos de desarrollo software.
2. Aprendizaje desde cero del framework A-Frame.
3. Manejo en profundidad de lenguajes de programación HTML, Javascript y Python.
4. Desarrollar programas y construir escenas en A-Frame.
5. También he adquirido soltura a la hora de trabajar la programación orientada a componentes y a eventos.
6. Procesar datos y leer ficheros externos desde nuestro programa principal.
7. Hacer uso de distintos modelos visuales para dotar a nuestra escena de un aspecto más inmersivo, además de utilizar distintos sonidos y efectos para trabajar en la sensación de realismo.
8. Probar el desarrollo en un entorno de realidad virtual real, haciendo uso de unas gafas especializadas y poder adaptar nuestra escena a lo que esta situación requiere.
9. Poder realizar una documentación técnica siguiendo las reglas del lenguaje de escritura LaTeX.

5.4 Planificación temporal

El desarrollo de este trabajo de fin de grado me ha llevado alrededor de 8 meses, empezando en el mes de noviembre de 2021 y dándolo por finalizado en junio de 2022.

Durante toda esta etapa he tenido que compaginar el desarrollo del trabajo con una jornada laboral a tiempo completo de 40 horas semanales, por lo que no siempre he podido sacar el tiempo necesario que me hubiera gustado para profundizar en este proyecto. Ya no es solo la jornada laboral que evita poder dedicarle tiempo al proyecto durante las horas de trabajo, sino los largos desplazamientos hacia los lugares de trabajo que limitan aún más el tiempo libre y consumen las energías que hacen falta para centrarse en un proyecto de este calibre. Es por ello que durante los primeros meses sólo podía avanzar con el proyecto los fines de semana, dedicándole largas jornadas intensivas de trabajo. Durante la etapa navideña y debido a un nuevo brote del ya conocido Covid-19 el trabajo pasó a ser totalmente remoto desde casa, por lo que pude ahorrar esos largos tiempos de desplazamiento y me ayudaron a dedicarle alrededor de 1 o 2 horas diarias al proyecto, además de esas jornadas intensivas en fines de semana. Tras otro par de meses de jornadas presenciales que limitaban mucho mi tiempo, en el mes de abril realicé un traslado laboral que me permitía la opción de trabajar en remoto cuando quisiera, lo cual me permitió poder avanzar mucho más rápido en la consecución del proyecto coincidiendo con la etapa final y más importante de este.

5.5 Trabajos futuros

Este trabajo es un proyecto dinámico, lo que quiere decir que no está definido un objetivo claro que nos indique que ya hemos finalizado el desarrollo, sino que se pueden ir añadiendo cambios o mejoras de forma casi infinita.

Este hecho nos indica que en ningún caso este proyecto se define como finalizado ya que hay infinitud de mejores que podrían dotar al proyecto de un valor todavía mayor. Aquí vamos a mencionar algunas de esas mejoras.

- En primer lugar se podría mejorar el aspecto visual de la escena, importando nuevos modelos para los paquetes y sus niveles de información y en definitiva hacer mejor la experiencia de usuario.
- Poder distinguir diferentes formas de visualización del tráfico. Por ejemplo visualizar solo los paquetes que tengan cierto nivel de información, aquellos que pertenezcan a una subred determinada o visualizar solo aquellos paquetes que pasen por un nodo en concreto.

- Tener la posibilidad no solo de parar o reanudar la escena sino también de poder adelantarla o dar marcha atrás.
- Mostrar por pantalla un pequeño resumen visual de la información que hay en la escena. Qué paquetes están en acción, qué nodos intervienen o cuál será el próximo paquete en aparecer.
- Sacar más partido de la realidad virtual, añadiendo más funcionalidades a la hora de visualizar la escena con unas gafas. Hacer uso de sus controladores para poder interactuar más con los elementos que aparecen en pantalla.
- Iterar directamente con NetGUI. Tener la opción en este programa de pasar a realidad virtual para visualizar el escenario haciendo uso de esta herramienta.
- Poder crear escenarios propios en realidad virtual y tener la opción de abrir dichos escenarios en NetGUI.
- Integrar la aplicación completamente con NetGUI. Poder clicar en cada una de las máquinas para desplegar terminales para la generación de tráfico entre nodos.

Referencias

- [1] *A-Frame*. URL: <https://aframe.io/> (visitado 03-07-2022).
- [2] *A-Frame Building a Minecraft demo*. URL: <https://aframe.io/docs/1.3.0/guides/building-a-minecraft-demo.html> (visitado 03-07-2022).
- [3] *A-Frame Hello World*. URL: <https://aframe.io/examples/showcase/helloworld/> (visitado 03-07-2022).
- [4] *A-Frame. Creating a scene*. URL: <https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene> (visitado 03-07-2022).
- [5] *Building a 360 image gallery*. URL: <https://aframe.io/docs/1.3.0/guides/building-a-360-image-gallery.html> (visitado 03-07-2022).
- [6] Ricardo Cabello. *Three.js*. threejs.org. URL: <https://threejs.org/> (visitado 03-07-2022).
- [7] Douglas Crockford. *The Good Parts*. Yahoo Press, 2008.
- [8] Allen B. Downey. *Think Python*. O'Reilly Media, 2015.
- [9] Jon Duckett. *HTML and CSS: Design and Build Websites*. Wiley, 2014.
- [10] *Html*. World Wide Web Consortium. URL: <https://devdocs.io/html/> (visitado 03-07-2022).
- [11] *Javascript*. ECMAScript.
URL: <https://developer.mozilla.org/es/docs/Web/JavaScript> (visitado 03-07-2022).
- [12] *Json*. ECMA-404 The JSON Data Interchange Standard.
URL: <https://www.json.org/json-es.html> (visitado 03-07-2022).
- [13] *LaTex*. LaTeX team. URL: <https://www.latex-project.org/help/documentation/> (visitado 03-07-2022).
- [14] *Netgui*. MobiQuo/GSyC group. URL: <https://mobiquo.gsync.urjc.es/netgui/> (visitado 03-07-2022).

- [15] *Overleaf*. European Organization for Nuclear Research (CERN). URL: <https://es.overleaf.com/learn> (visitado 03-07-2022).
- [16] *Python*. Python Software Foundation. URL: <https://docs.python.org/3/> (visitado 03-07-2022).
- [17] *Sketchfab 3D models*. URL: https://sketchfab.com/3d-models?features=downloadable&sort_by=-likeCount (visitado 03-07-2022).
- [18] *Visual Studio Code*. Microsoft. URL: <https://code.visualstudio.com/> (visitado 03-07-2022).
- [19] *WebGL*. WebGL. URL: https://www.khronos.org/webgl/wiki/Main_Page (visitado 03-07-2022).
- [20] *WebXR*. W3C Immersive Web Working y Community Groups. URL: <https://immersiveweb.dev/> (visitado 03-07-2022).
- [21] *WebXR Device API*. W3C Immersive Web Working y Community Groups. URL: <https://www.w3.org/TR/webxr/> (visitado 03-07-2022).
- [22] *Wireshark*. wireshark.org. URL: <https://www.wireshark.org/> (visitado 03-07-2022).