Universidad
Rey Juan Carlos

ESCUELA DE INGENIERÍA DE FUENLABRADA

GRADO EN INGENÍERIA EN SISTEMAS AUDIOVISUALES Y MULTIMEDIA

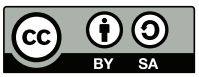
TRABAJO FIN DE GRADO

TÍTULO DEL TRABAJO CON LETRAS MAYÚSCULAS PARA SUSTANTIVOS Y ADJETIVOS

Autor : Juan José Arias Rojas

Tutor : Dr. Jesús María González Barahona

Curso académico 2024/2025



Dedicado a

todos aquellos que me animaron y apollaron

incluso en mis momentos de debilidad

II

Agradecimientos

Resumen

Summary

Índice general

Índice de figuras

Capítulo 1

Introducción

Capítulo 2

Tecnologías utilizadas

En este capítulo se mostraeán las distintas tecnologías que han sido necesarias para el desarroyo de este proyecto tanto de forma directa como indirecta. Del mismo modo, se explicará su funcionalidad y la manera en la que contribuyeron al proyecto. Las tecnologías se dividirán en 2 categorias, las principales y las auxiliares. Las principales son aquellas que han tenido un papel directo en el desarrollo del proyecto sin las cuales no se podria haber realizado, mientras que las auxiliares sen aquellas que han ayudado al desarrollo de forma mas indirecta.

2.1. Tecnologías principales

Aquí se nombran y explican aquellas tecnologías que han tenido una implicación directa con el proyecto y que han tenido un papel crucial e imprescindible con las cuales, sin ellas, no se habria popido llegar a los resultados obtenidos. Algunas de las principales tecnologias utilizadas han sido HTML5, JavaScript y A-Frame, estas tres tecnologias han sido el nucleo principal del proyecto. El resto de las tecnologías principales son aquellas en las que se apollan esas tres, permitiendo que funcionen como lo hacen.

2.1.1. A-Frame

A-Frame [1] es un web framework diseñado para construir experiencias de realidad virtual. A-Frame esta basado en HTML y en JavaScript, permitiendo realizar cualquier escena que uno pueda imaginar sin necesidad de conocimientos avanzados en gráficos 3D. Al estar basado en

HTML es posible realizar escenas simples directamente desde un archivo HTML simplemente importando la libreria de A-Frame y añadiendo dentro de la etiqueta `<a-scene>` cualquier elemento que desee el usuario.

Uno de los ejemplos más simples que se puede llegar a realizar utilizando A-Frame es el siguiente:

```
<html>
  <head>
    <script src="https://aframe.io/releases/1.6.0/aframe.min.js"></script>
  </head>
  <body>
    <a-scene>
      <a-box position="-1 0.5 -3" rotation="0 45 0" color="#4CC3D9"></a-box>
      <a-sphere position="0 1.25 -5" radius="1.25" color="#EF2D5E"></a-sphere>
      <a-cylinder position="1 0.75 -3" radius="0.5" height="1.5" color="#FFC65D"></a-
          cylinder>
      <a-plane position="0 0 -4" rotation="-90 0 0" width="4" height="4" color="#7BC8A4"></a
          -plane>
      <a-sky color="#ECECEC"></a-sky>
    </a-scene>
  </body>
</html>
```

Listing 2.1: Escena A-Frame básica

Este código de HTML genera una de las escenas más básicas que se pueden hacer en A-Frame. La cual consiste en un plano y, encima, una serie de figuras geométricas, como se aprecia en la figura 2.1.

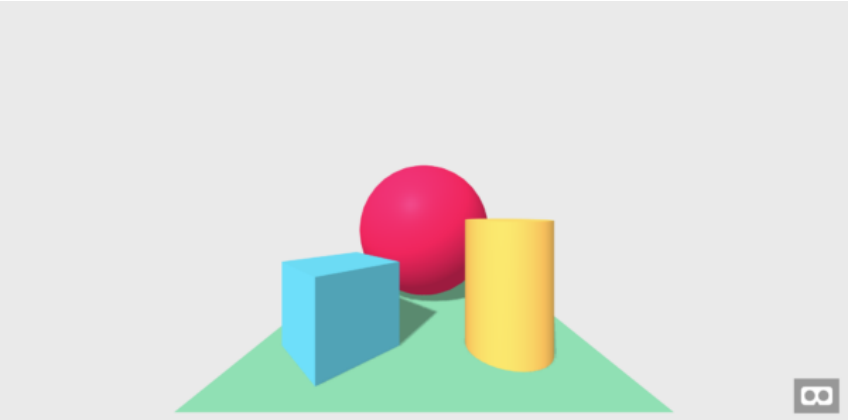


Figura 2.1: Escena básica de A-Frame

Una de las principales características de esta tecnología es su arquitectura basada en un

modelo de entidad-componente (*Entity Component System*), donde cada objeto dentro de la escena es una entidad que luego es integrada con Three.js para crear la escena. Una entidad en A-Frame es un objeto HTML que se crea añadiendo la etiqueta `<a-entity>` y el componente es la apariencia, comportamiento y funcionalidad que se le asigna mediante JavaScript.

Para hacer uso de un componente se requiere de una serie de pasos previos. Si el componente es externo es necesario importarlo en la cabecera del HTML del mismo modo que se importa A-Frame y posteriormente añadirlo a la entidad en la que uno desee usarlo. Por otra parte, si el componente es de nuestra creación, primero hay que registrar y definir el componente. Esto se hace en un archivo externo de JavaScript mediante el método de `A-Frame.registerComponente`. A este método, del mimo modo que una función primero se le asigna un nombre, este sera el nombre del componente. Luego, este metodo consta de varias funciones internas las cuales se encargan de definir la apariencia y lógica de dicho componente. Alguna de estas funciones son la función `schema`, `init` o `tick`. La función `schema`, define las propiedades principales del componente, estas se pueden modificar dependiendo de la lógica del componente desde el HTML a la hora de llamarlo. La función `init` se ejecuta una única vez al inicio cuando se inicializa el componente y la función `tick` se ejecuta a cada frame de la escena. Una vez el componente esta registrado y programado importamos el archivo que contiene el componente al HTML y lo insertamos a la escena en la entidad u objeto que deseemos.

Un ejemplo de la llamada de un componente dentro de un archivo de HTML se puede apreciar en 2.2

```
<html>
  <head>
    <script src="https://aframe.io/releases/1.6.0/aframe.min.js"></script>
    <script src = "componente1.js"></script>
    <script src = "componente2.js"></script>
  </head>
  <body>
    <a-scene>
      <a-entity componente1></a-entity>
      <a-box componente2></a-box>
    </a-scene>
  </body>
</html>
```

Listing 2.2: Ejemplo de llamada de entidad

En este ejemplo primero se muestra como a una entidad vacia se le añade un compoenn-

te llamado `componente1` y también se muestra como a una entidad de cubo se le añade el componente `componente2`. Del mismo modo, para la creación de un componente, en el archivo `.js` que posteriormente importaremos en la escena, unicamente es necesario registrar el componente y definir sus caracteristicas como en el siguiente ejemplo:

```
AFRAME.registerComponent('componente1', {
init() {
  const box = document.createElement('a-box');
  box.setAttribute('position', '-1 0.5 -3');
  box.setAttribute('rotation', '0 45 0');
  box.setAttribute('color', '#4CC3D9');

  const sphere = document.createElement('a-sphere');
  sphere.setAttribute('position', '0 1.25 -5');
  sphere.setAttribute('radius', '1.25');
  sphere.setAttribute('color', '#EF2D5E');

  this.el.appendChild(box);
  this.el.appendChild(sphere);
}
});
```

Listing 2.3: Ejemplo de creacion de entidad

Este fragmento de código registra el componente `componente1` el cual crea un cubo y un cilindro con unos colores, posición y rotación especificos y los añade a la escena.

Para este proyecto, A-Frame ha sido la tecnología principal, permitiendo elementos fundamentales como la creación y gestión de entornos de realidad virtual. También, permitio la integración de componentes diseñados mediante JavaScript y permitio el manejo de interacciones mas complejas a la vez que optimizaba el rendimiento de la escena 3D.

2.1.2. HTML5

HTML5 [4] es la quinta iteración del lenguaje de HTML (*marcado de hipertexto*), el cual es la estructura básica y principal de toda página web. Desarrollado conjuntamente por W3C (*World Wide Web Consortium*) [10] y WHATWG (*Web Hypertext Application Technology Working Group*) [11]. HTML5 introduce mejoras sustanciales respecto a sus anteriores versiones, incluyendo nuevas etiquetas semánticas, soporte multimedia mejorado y compatibilidad ampliada con diversos navegadores y dispositivos, dándole mayor flexibilidad y dinamismo a la creación de páginas web.

Una de sus nuevas introducciones clave es la introducción de etiquetas semánticas como: `<header>`, `<article>`, `<section>` y `<footer>`, que facilitan una estructuración clara y accesible del contenido web. Estas etiquetas no solo ayudan a mejorar la organización de la información, si no que también facilitan la búsqueda de información por parte de los motores de búsqueda y facilitan la interpretación por parte de los desarrolladores. Además, HTML5 incorpora nuevas interfaces de programación de aplicaciones (*API*), destacándose las funcionalidades de almacenamiento local mediante `localStorage` y `sessionStorage`, que permiten la gestión eficiente de datos directamente en el navegador, eliminando la necesidad de bases de datos externas.

HTML5 también introduce mejoras significativas en la gestión de contenido multimedia, eliminando la necesidad de complementos externos. Para esto se implementaron las etiquetas `<audio>` y `<video>` las cuales permiten la incorporación directa de archivos de sonido y video en las páginas web, permitiendo mayor dinamismo en el desarrollo. Otro elemento que se añadió fue el elemento `<canvas>` el cual habilita la generación de gráficos y animaciones en tiempo real a través de JavaScript, permitiendo el desarrollo de aplicaciones interactivas y videojuegos en línea.

HTML5 se ha establecido firmemente como un estándar en el desarrollo de aplicaciones web progresivas (*PWA*) y en entornos multiplataforma. Su integración con tecnologías como CSS3 y JavaScript permite la creación de interfaces dinámicas y adaptativas, compatibles con una amplia gama de dispositivos, desde ordenadores hasta tabletas y teléfonos.

Gracias a que HTML es una tecnologia extensible, tecnologias como A-Frame o WebXR han sido posibles de utilizar ya que son extensiones de HTML. WebXR permitio poder trabajar con entornos VR desde el navegador mientras que A-Frame fue el eje central del proyecto, donde se estructuro la escena. HTML fue una de las tecnologías mas importantes ya que ofrece una estructura esencial para el desarroyo del proyecto, haciendo uso de sus extensiones de A-Frame y WebXR para poder mostrar los elementos de la escena en la realidad virtual.

2.1.3. JavaScript

JavaScript es un lenguaje de programación ligero y multiplataforma utilizado principalmente para la creación de contenido dinámico e interactivo para páginas web. Su felixibilidad permite desarrollar elementos para mejorar la interacción del usuario en páginas web tanto del

lado del servidor como del lado del cliente. En 1997, JavaScript fue estandarizado por *ECMA* como ECMAScript y poco después como un éstandar *ISO*.

JavaScript, creado por Brendan Eich de Netscape en 1995 bajo el nombre de *Mocha*, posteriormente se renombró a LiveScript y finalmente a JavaScript. En el año 2000, JavaScript se extendió al lado de los servidores con la introducción de tecnologías como *Node.js*, y posteriormente su popularidad creció con la llegada de *AJAX*. Y Con la llegada de ECMAScript 6 en 2015, se introdujeron características mas avanzadas y actualizaciones anuales, haciendo que hoy en día sea uno de los lenguajes mas importantes en el desarrollo web.

JavaScript es un lenguaje de programación de alto nivel, lo que significa que su leguaje esta diseñado para ser lo mas 'humano' posible, permitiendo una rapida comprensión del codigo sin necesidad de un nivel alto de conocimientos. Es un lenguaje basado en eventos ya sean entradas de ratón o entradas de teclado, permitiendo la creación de interfaces de usuario interactivas. JavaScript puede integrarse en un HTML dentro de la etiqueta `<script>` de forma directa, escribiendo el codigo. También, se puede introducir codigos de JavaScript, los cuales estén en archivos distintos con la extensión correspondiente al lenguaje (.js), esto es posible vinculando dicho archivo en la cabecera del HTML.

Para este proyecto formo un papel crucial, ya que fue con JavaScript que se desarrollaron los distintos componentes de A-Frame que se usaron para el funcionamiento del proyecto. Permitio diseñar e implementar la lógica que hay tras cada componente permitiendo alcanzar los resultados obtenidos.

2.1.4. Three.js

Three.js [6] es una biblioteca de código abierto de JavaScript utilizada para la creación de gráficos 3D en los navegadores web. Three.js funciona sobre WebGL, la cual permite generar y visualizar animaciones y escenas 3D en el navegador sin necesidad de complementos. La tecnologia de WebGL tambiénpuede usarse junto con el elemento `<canva>` de HTML.

Esta biblioteca proporciona una API de alto nivel la cual permite al usuario la creación y manipulación de geometrías 3D como cubos, esfera o planos, así como la aplicación de texturas o la aplicación de tanto camaras como de efectos de iluminación en las escena, permitiendo que el desarrollo de dichas escenas sea mucho mas simplificado. También, Three.js permite la posibilidad de importar modelos 3D desde archivos distintos creados desde aplicaciones distintas.

La biblioteca de Three.js ofrece una alta variedad de herramientas que permiten la gestión y control de usuario, facilitando asi tanto la navegación como la interacción dentro de las escenas 3D. También, Three.js soporta animaciones suaves y dinámicas tanto para objetos en la escena como para camaras, lo cual permite la creación de efectos visuales mas dinámicos e impresionantes al igual que juegos interactivos.

Gracias a la amplia variedad de posibilidades que permite la biblioteca de Three.js las opciones de los elementos o escenas que se pueden llegar a crear son basicamente ilimitadas. Siendo capaz de crear escenas complejas como la de la figura 2.2

Figura 2.2: Escena de Three.js

Three.js permitio añadir una capa mas simplificada a WebGL, simplificando asi el proceso de creación y manipulación de los elementos 3D. Permitió crear un entorno 3D con mayor dinamismo y control para el proyecto.

2.1.5. WebXR

WebXR [9] es una tecnología desarrollada por W3C (*World Wide Web Consortium*) que permite crear experiencias inmersivas desde el navegador. Esta tecnología combina la Realidad Aumentada (*AR*) y la Realidad Virtual (*VR*) con la accesibilidad de un navegador web. Lo cual permite a los usuarios experimentar e interactuar con entornos tridimensionales a través de cualquier dispositivo con acceso a un navegador compatible.

A medida que la tecnología de WebXR ha ido evolucionando, se han desarrollado distintos tipos de experiencias para poder adaptarse a distintos contextos y necesidades. Hoy en día, se podria clasificar los distintos tipos en 3 categorias:

- **WebXR AR:** Este tipo de WebXR combina el mundo virtual y el mundo real, permitiendo que distintos elementos del mundo virtual puedan superponerse en el entorno físico a través de la cámara de un dispositivo compatible, pero sin que llegue a influir el mundo real en los elementos virtuales.

- **WebXR VR:** Este modo permite a los usuarios sumergirse en el entorno virtual creado, y con la ayuda de dispositivos suplementarios como auriculares, la experiencia es aún mayor. Esta tecnología permite a los usuarios interactuar y experimentar en primera persona distintos entornos virtuales, desde juegos y entretenimiento, hasta simulaciones virtuales y visualizaciones en 3D.

- **WebXR MR(Mixed Reality):** Esta tecnología combina el mundo real y el virtual de forma mas profunda que la tecnología AR. Permite a los usuarios poder interactuar con elementos tantos virtuales como físicos y que estos puedan interactuar entre si. Este tipo de combinación proporciona un nivel mayor de interacción entre el usuario y y su entorno, dando mayor numero de posibilidades para la creación de contenido.

Estos distintos tipos de WebXR ofrecen distintos tipos de experiencias dependiendo del entorno virtual que se quiera diseñar.

Fue gracias a WebXR, en este proyecto se pudieron realizar las distintas escenas inmersivas que permitieron ir comprobando el avance del proyecto con cada una de las demos que se creaban y que posteriormente se explicaran. Además de eso, fue gracias a WebXR que se pudo realizar la detección de las manos ya que ya posee una función integrada que permite el handtracking y al querer trabajar sobre el navegador era la mejor opción para ello.

2.1.6. WebGL

WebGL [3] (*Web graphics Library*), se trata de una tecnología de bajo nivel multiplataforma usada para la renderización de gráficos tanto tridimensionales como bidimensionales dentro de cualquier navegador que sea compatible.

WebGL fue lanzado en 2011 por el grupo Khronos. Esta tecnología se fundamenta en OpenGL ES, la cual es una variante simplificada de OpenGL, diseñada para dispositivos moviles. WebGL ha sufrido numerables actualizaciones, lo cual ha permitido una evolución continua

que ha ido mejorando tanto su funcionalidad como compatibilidad tanto en navegadores de escritorio como en navegadores en dispositivos móviles.

WebGL esta diseñado para trabajar directamente con la GPU (*Graphic Processing Unit*) del dispositivo, lo cual permite un mayor aprovechamiento de la computación para poder generar gráficos detallados y de alta calidad. Además, la tecnología de WebGL esta diseñada para poder integrarse de manera fluida con otros estandares de desarrollo web como HTML, CSS o DOM.

Gracias a esta tecnología fue posible renderizar en la escena las distintas entidades que forman las manos del mismo modo que las otras entidades que se crearon para ir comprobando si los componentes creados funcionaban.

2.2. Tecnologias auxiliares

En esta sección se detallan y describen las distintas tecnologías que aunque no hayan influido de manera directa en los resultados del proyecto, han tenido un papel crucial para el desarroyo de este. Entre dichas tecnologias se encuentra el *IDE* utilizado durante el proyecto al igual que el dispositivo utilizado para realizar las pruebas de las demos, la plataforma donde se almacenó el código y también el software utilizado para la redacción de esta memoria.

2.2.1. Visual Studio Code

Visual Studio Code (*VS Code*) es un potente entorno de desarrollo integrado (*IDE*) el cual esta disponible para Windows, Linux, MacOS y versión web. VS Code es una de las plataformas de edición de codigo más utilizadas a nivel mundial por toda clase de desarrolladores de software debido a su versatilidad, flexibilidad y amplia gama de extensiones que facilitan la edición y depuración de código.

Una de las pestañas mas importantes durante el desarrollo fue su pestaña de **Source Control**, la cual luego de conectar mi repositorio de GitHub, permite ir guardando el código en GitHub para tener un control de las distintas versiones del código.

Gracias a las numerosas extensiones que hay disponibles facilitan en gran medida la creación y depuración de código, permitiendo cosas como autocompletar palabras o expresiones o permitir compilar o depurar con algun lenguaje que VS Code no tenga por defecto. Algunas de las extensiones utilizadas en este proyecto han sido:

- **A-Frame completition**: Permite autocompletar rapidamente a la hora de escribir los distintos elementos o Snippets que posee A-Frame

- **Error lens**: A la hora de depuración, permite visualizar dentro del código si hay algun error de sintaxis o lógica

- **LaTeX Workshop**: Permite la escritura, compilación y previsualización de la memoria de este proyecto.

2.2.2. GitHub

GitHub es una plataforma basada en la nube que permite almacenar distintos repositorios y en cada repositorio codigo. GitHub esta basado en Git [5], el cual fue creado en 2005 por Linus Torvalds como un sistema de control de versiones de código abierto. Este sistema fue desarroyado con la intención de ayudar a los desarrolladores a tener en cualquier dispositivo con acceso a internet todo el historial de código que estan desarrollando.

Esta tecnología es ampliamente utilizada a nivel mundial por desarrolladores de todo tipo debido a su capacidad de almacenar, soportar y gestionar proyectos de toda clase. Una de las funciones clave que presenta la plataforma, es la capacidad de crear ramificaciones (branches). Esta opcion permite a los desarroyadores poder crear copias del codigo del repositorio en el que estan trabajando para poder trabajar en paralelo y posteriormete poder integrar los cambios realizados en paralelo al codigo principal. Esta opcion también permite que mas de un desarrollador pueda trabajar en el mismo codigo, haciendo que cada uno trabaje en paralelo en ramas distintas.

GitHub aprovecha todas las ventajas que permite Git, permitiendo crear repositorios en la nube para sus proyectos Git, que los desarrolladores pueden llegar a compartir con otras personas. Esta plataforma tambien facilita la colaboración entre desarrolladores y aparte de las ventajas ya mencionadas de Git permite también el uso de herramientas de gestión de proyectos al igual que acciones automaticas.

Durente este proyecto se creo el repositorio de GitHub *TFG*[1] donde se fue guardando todas las versiones del código al igual que todas las demos y pruebas realizadas.

[1] `https://github.com/JuJoarias/TFG`

2.2.3. Meta Quest 3

Meta Quest se trata de un dispositivo inalámbrico diseñado para disfrutar de contenido de realidad virtual (VR) por Meta [8]. Dicho dispositivo esta compuesto por unas gafas, un micrófono y auriculares integrados en un unico dispositivo que el usuario pone en su cabeza y un par de mandos inalámbricos. La primera version fue lanzada en 2020 con las Meta Quest 2 y 3 años mas tarde, en 2023 se lanzaron las Meta Quest 3.

Para este proyecto se han utilizado unas Meta Quest 3, las cuales coontaban con una memoria de 8 GBB de memoria RAM, una resolición de 2064x2208 pixeles por cada uno de los ojos. También, cuenta con mejoras respecto al campo de visión (FOV) respecto a su version anterior. Para este proyecto jugaron un papel cricial ya que al enfocarse en la detección de mandos el proyecto, sin las gafas no se habria podido probar los resultados que se iban obteniendo.

Figura 2.3: Meta Quest 3

2.2.4. LaTeX

LaTeX [7] es un software de uso libre de creación de documentación de alta calidad. Este sistema es altamente utilizado para la creación de documentación tecnica o cientifica de media o larga extensión, aunque gracias a su versatilidad se puede utilizar para cualquier tipo de documentación.

Entre las distintas caracteristicas de LaTeX cabe destacar su capacidad de manejar expresiones matematicas complejas, lo cual es una herramienta indispensable para cualquier documento cintifico. También gracias a su software motor TeX LaTeX es capaz de convertir los comandos

de texto, utilizados para expresar los resultados tipográficos, en un archivo PDF profesional.

Además de facilitar la creación de formulas matematicas complejas, LaTeX también ofrece otras facilidades avanzadas, como puede ser la creación de un índice del contenido, un índice de las figuras utilizadas, gestión de bibliografias o referencias simplificando la organización del documento y la citación de figuras o elementos externos en la bibliografia.

Para este proyecto, LaTeX facilito considerablemente el trabajo a la hora de crear este documento, ayudando con la creación y gestión de indices, imagenes y bibliografia al igual que a estructurar el texto del documento de forma correcta.

Capítulo 3

Desarrollo del proyecto

En este capítulo se describe de forma detallada como fue el desarrollo del proyecto. Dicho desarrollo se describe en forma de sprints siguiendo una estructura de una metodología *Agile*[2], pese a que el propio proyecto no siguio dicha metodología. El desarrollo explica desde los inicios y planteamiento del proyecto hasta obtener los resultados finales.

3.1. Sprint 0

Puesta en marcha del proyecto, planteamiento con el tutor y dominio del uso basico de A-Frame.

3.1.1. Objetivos

El objetivo principal de este sprint se puede dividir en 2 partes principales.

- **El planteamiento con el tutor**, donde se identificaron las necesidades para el proyecto del mismo modo que planificar las distintas etapas y duración de cada una y los objetivos que se deseaban obtener.

- **Dominio del uso basico de A-Frame**. Aprender a usar de forma correcta como funcionan las escenas al igual que la creación e implementacion de componentes o el uso de librerias de A-Frame.

3.1.2. Tareas Realizadas

La identificación de las necesidades y requisitos de un proyecto es una parte fundamental. Durante las primeras charlas con el tutor se concretaron los distintos objetivos y los posibles caminos que puede tomar este proyecto. Al principio se plantearon varios caminos, pero la idea general de todos ellos era la creación de un sistema de handtracking que sea capaz de interactuar con distintos elementos dentro de la escena.

Para lograr llegar a ese objetivo primero fue necesario dominar los elementos basicos de A-Frame como biene a ser la organización de la escena, el uso y creación de componentes y el uso de eventos. Para ello, primero se creo primero el repositorio de *Git* con el que trabajaremos durante todo el proyecto. Se crearon una serie de escenas sencillas con el objetivo de dominar el uso de componentes y eventos simples.

3.1.3. Resultados

Para familiarimarme aun mas con el entorno de A-Frame se crearon una serie de escenas muy sencillas donde se creaba y utilizaba un componente customizado y se hacia uso del evento nativo de A-Frame `click`. Estas escenas permitieron un mayor entendimiento del funcionamiento tanto de componentes como de eventos, que fueron fundamentales para el resto del proyecto. El resultado de dichas pruebas para mejorar la comprensión de A-Frame se pueden apreciar en las figura 3.1.

Figura 3.1: Escena de test de A-Frame

Tras la familiarización con el entorno de A-Frame y sus características el proyecto fue for-

malizado formalmente. Tras ellos se planteo el siguiente sprint del proyecto donde ya se empezarian los primeros pasos formales de este.

3.2. Sprint 1

Investigando el handtraking

3.2.1. Objetivos

En este primer sprint oficial del proyecto, el objetivo princpial era investigar y comprender el uso del handtraking en A-Frame. Este era el paso principal en el que se basaria todo el proyecto, ya que sin una comprensión del funcionamiento del handtracking en A-Frame este proyecto habria sido imposible de realizar.

3.2.2. Tareas Realizadas

Los primeros pasos fue investigar el handtracking en A-Frame con los elementos ya existentes como *super hands*[1], o los componentes nativos de A-Frame `hand-controls` o `hand-tracking-controls`. Para ellos se usaron pequeñas escenas de prueba para comprender mejor el comportamiento de las manos en la realidad virtual, y del mismo modo, tener una idea mas clara del objetivo final del aspecto de handtracking.

Tras la investigación de las manos y la sugerencia del tutor, se decidio utilizar la tecnologia de WebXR para el handtracking puesto que esta tecnología ya posee un sistema preparado para ello. Además, usando la tecnología de WebXR nos permitia trabajar con cada una de las articulaciones de la mano de forma independiente.

Siguiendo la documentación de *WebXR*[2], se empezo el desarrollo de las manos. para ello en el codigo fue necesario tratar cada una de las manos de forma independiente, del mismo modo que cada una de las articulaciones como se muestra en el esquema de las manos de WebXR en la figura 3.2.

[1] `https://github.com/c-frame/aframe-super-hands-component`
[2] `https://www.w3.org/TR/webxr-hand-input-1/`

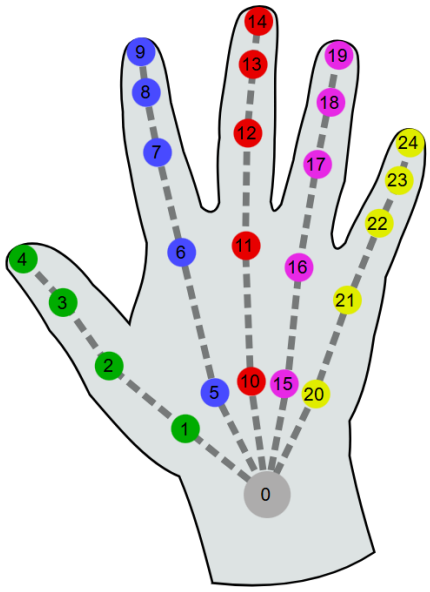


Figura 3.2: Esquema manos segun WebXR

Para poder crear las manos de forma exitosa dentro del entorno de realidad virtual siguiendo WebXR, primero se crea el componente que sera el encargado de detectar y dibujar las manos en este proyecto, dicho componente se llama `hand-skeleton`. Dicho componente no tenia un `schema`, en el `init` se definian las distintas variables que eran necesarias para el funcionamiento del código. En el HTML que define la escena unicamente era necesario añadir una vez el componente como una entidad. Pero era imprescindible definir todas y cada una de las articulaciones, como se muestra en el fragmento de código 3.1, pero esa definición unicamente se realizaba en el código de JavaScript.

```
const orderedJoints = [
  ["wrist"],
  ["thumb-metacarpal", "thumb-phalanx-proximal", "thumb-phalanx-distal", "thumb-tip"],
  ["index-finger-metacarpal", "index-finger-phalanx-proximal", "index-finger-phalanx-
      intermediate", "index-finger-phalanx-distal", "index-finger-tip"],
  ["middle-finger-metacarpal", "middle-finger-phalanx-proximal", "middle-finger-phalanx-
      intermediate", "middle-finger-phalanx-distal", "middle-finger-tip"],
  ["ring-finger-metacarpal", "ring-finger-phalanx-proximal", "ring-finger-phalanx-
      intermediate", "ring-finger-phalanx-distal", "ring-finger-tip"],
  ["pinky-finger-metacarpal", "pinky-finger-phalanx-proximal", "pinky-finger-phalanx-
      intermediate", "pinky-finger-phalanx-distal", "pinky-finger-tip"]
];
```

Listing 3.1: Definición de articulaciones

Después de definir todas las articulaciones era necesario procesarlas. Para ello, se creo la función que se muestra en el fragmento de código 3.2. En la función `renderHandSkeleton` se crea un bucle de código el cual, toma la sesión XR de la escena y por cada input de la sesion toma cada una de las articulaciones, las detecta en la mano y crea una esfera a traves de una segunda función unicamente encargada de crear esferas. Dichas esferas se crean en la posición corerspoondiente a la mano real y tambien se ajusta el radio de la esfera para que coincida con la mano.

```
renderHandSkeleton: function () {
  const session = this.el.sceneEl.renderer.xr.getSession();
  if (!session || !this.frame || !this.referenceSpace) {
      return;
  }
  const inputSources = session.inputSources;
  for (const inputSource of inputSources) {
      if (inputSource.hand) {
          const hand = inputSource.hand;
          const handedness = inputSource.handedness; // Determina si es la mano derecha o
              izquierda
          for (const finger of orderedJoints) {
              for (const jointName of finger) {
                  const joint = hand.get(jointName);
                  if (joint) {
                      const jointPose = this.frame.getJointPose(joint, this.referenceSpace);
                      if (jointPose) {
                          const position = jointPose.transform.position;
                          if (!this.spheres[handedness + '_' + jointName]) {
                              this.spheres[handedness + '_' + jointName] = this.drawSphere(
                                  jointPose.radius, position);
                          } else {
                              this.spheres[handedness + '_' + jointName].object3D.position.
                                  set(position.x, position.y, position.z);
                          }
                      }
                  }
              }
          }
      }
  }
},
```

Listing 3.2: Dibujo de las manos en la escena

Esta función se encuentra dentro de la función `tick` del componente manos, asi que a cada

frame el propio código revisa la posición de todas y cada una de las articulaciones para cada una de las dos manos de entrada. Además de detectar y dibujar las manos en la escena, en esta primera demo se implementó el gesto de `Pinch`, distinguiendo cada mano y realizando una acción distinta dependiendo de la mano que lo realizase.

Primero, para la detección del gesto se creo la función de `detectGestures`, como se muestra en el fragmento de código 3.3. En dicha función, se tomaba la posición del la punta del dedo índice y del pulgar y se calculaba la distancia entre ambos puntos.

```
detectGestures: function () {
  const session = this.el.sceneEl.renderer.xr.getSession();
  const inputSources = session.inputSources;
  let rightPinching = false;
  let leftPinching = false;
  for (const inputSource of inputSources) {
      if (inputSource.hand) {
          const thumbTip = this.frame.getJointPose(inputSource.hand.get("thumb-tip"), this.
              referenceSpace);
          const indexTip = this.frame.getJointPose(inputSource.hand.get("index-finger-tip"),
               this.referenceSpace);
          if (thumbTip && indexTip) {
              const distance = calculateDistance(thumbTip.transform.position, indexTip.
                  transform.position);
              if (distance < pinchDistance) {
                  if (inputSource.handedness === 'right') {
                      rightPinching = true;
                      if (!this.rightPinching) {
                        this.createBoxAtHand(inputSource.hand);
                      }
                  } else if (inputSource.handedness === 'left') {
                      leftPinching = true;
                      if (!this.leftPinching) {
                          this.selectObjectAtHand(inputSource.hand); // Selecciona un objeto
                               cercano para mover
                      }
                  }
              }
          }
      }
  }
  this.rightPinching = rightPinching;
  this.leftPinching = leftPinching;
},
```

Listing 3.3: Función detectGestures

Si la distancia era inferior a 0.02 (valor obtenido tras realizar varias pruebas), se comprobaba que mano era la que lo realizaba y luego se llamaba a la función correspondiente. Si se trataba de la mano derecha, se creaba un cubo verde en la posición de la punta del dedo índice. Y si se trataba de la mano izquierda, si esta se encontraba lo bastante cerda de uno de los cubos creados por la mano derecha (a una distancia inferior a 0.15) la posición del cubo copiaria la de la punta del dedo índice izquierdo hasta que se soltase el gesto.

3.2.3. Resultados

Como resultado de este sprint se creo la demo de *pinch test*[3] al igual que en la figura 3.3.

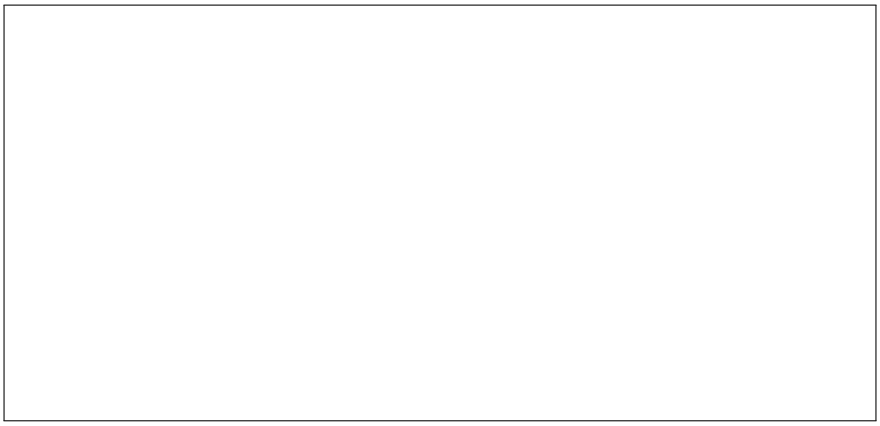


Figura 3.3: Primera demo de manos

Las esferas se dibujaban correctamente y los gestos funcionaban, pero habia bastantes aspectos a mejorar. Primero, por sujerencia del tutor, se debia modificar el componente para que fuese necesario añadirlo 2 veces a la escena, 1 por cada mano. También, respecto al gesto, con la mano derecha habia que cambiar el funcionamiento ya que el que creara un cubo no seria necesario mas adelante. Y respecto a la mano izquierda, se identificó que a la hora de tomar el cubo, este se transportaba hacia la posicion del dedo índice. Y aunque esto era lo deseado de primeras, visualmente no quedaba bien ya que si se agarraba una esquina el cubo se transportaba, tragandose la mano en vez de agarrar el cubo mientras mantenia la distance entre el dedo y el centro del cubo. Aparte de eso, en el caso de que hubiese mas de un cubo muy pegado,

[3] `https://github.com/JuJoarias/TFG/blob/main/first_steps/first_hands/pinch_test.html`

el código no identificaba siempre de forma correcta el mas cercano, haciendo que a veces se tomara el cubo no deseado.

Todos estos factores dieron paso al siguiente sprint, donde se corregirian los errores resaltados. Aparte se intentaria optimmizar la creación de las manos y se añadiria la emisión y gestión de eventos, ya que en esta primera demo, al realizar el gesto, se llama a la función correspondiente si utilizar eventos.

3.3. Sprint 2

Introducción de eventos y optimización del manejo de manos.

3.3.1. Objetivos

Tras la observaciones del sprint anterior, para este sprint se replanteo el funcionamiento de los gestos. También, se reorganizo la forma de creación de las manos en la escena y la detección de los gestos.

3.3.2. Tareas Realizadas

Primero, se reorganizo el componente encargado de dibujar las manos que se creo en el sprint anterior. Dicho componente se renombro a `manos`, nombre que se mantendría hasta el final del proyecto. A dicho componente se le añadio un `schema`. En dicho `schema` hay unacamente una variable llamada `hand`, dicha variable unicamente acepta `left` o `right` como entradas. Este cambio era para que el componente distinguiera las manos y unicamente dibujase una mano a la vez, esto hizo que para dibujar ambas manos en la escena fuese necesario añadir el componente 2 veces, 1 por cada mano. También, se modifico la función `init`. Ahora, aparte de definir los valores iniciales de las variables utilizadas, realiza un bucle que recorre la constante que contiene todas las articulaciones y por cada una crea una esfera. Dichas esferas son añadidas a la escena y tambien se crea un diccionario al que se le añade el nombre y entidad de cada articulación. Este bucle se puede apreciar en el fragmento de código 3.4. Como se puede apreciar, esto no nos permite detectar las manos y utilizarlas en la escena, simplemente prepara las entidades que representaran la mano dentro de la escena.

```
orderedJoints.flat().forEach((jointName) => {
  const jointEntity = document.createElement('a-sphere');
  jointEntity.setAttribute('color', 'white');
  this.el.appendChild(jointEntity);
  this.joints[jointName] = jointEntity;
});
```

Listing 3.4: Creación de las esferas de las articulaciones

Para la actualización de la posición de todas las articulaciones se creao la funcion que se muestra en el fragmetno de código 3.5. Esa nueva función es una versión actualizada de la que se creo en el sprint anterior. La principal diferencia con el código previo es que este unicamente trabaja con la mano que se definió en el `schema` y trabaja directamente con las entidades que estan guardadas en el diccionario que se creo en el `init`.

```
updateSkeleton: function () {
  const session = this.el.sceneEl.renderer.xr.getSession();
  const inputSources = session.inputSources;
  for (const inputSource of inputSources) {
    if (inputSource.handedness === this.data.hand && inputSource.hand) {
      for (const [jointName, jointEntity] of Object.entries(this.joints)) {
        const joint = inputSource.hand.get(jointName);
        const jointPose = this.frame.getJointPose(joint, this.referenceSpace);

        if (jointPose) {
          const { x, y, z } = jointPose.transform.position;
          const radius = jointPose.radius;
          jointEntity.setAttribute('position', { x, y, z });
          jointEntity.setAttribute('radius', radius || 0.008);
        } else {
          jointEntity.setAttribute('position', '0 0 0'); // Esconder si no hay datos
        }
      }
    }
  }
},
```

Listing 3.5: Actualización de las manos

Para la detección del gesto la lógica general permanece igual, pero en vez de llamar a otras funciones se emiten los eventos `pinchstart` y `pinchend` dependiendo de si se esta realizando el gesto o no y junto al evento se envia como argumento de este la mano que lo realiza, la derecha u izquierda.

Para escuchar los eventos, se creo el componente `detector`. Este componente posee 2

variables en su `schema`. La primera es que mano detectara, del mismo modo que el componente `manos`, y la segunda es un target. El target es principalmente para debug de la escena, ya que para esta demo se añadieron 2 entidades a la escena las cuales eran texto. Asi que el target seria el ID de una de esas entidades de texto. Este componente cada vez que reciviese un evento se encargaria de modificar el texto del target, señalando si se ha iniciado o terminado el gesto y de que mano.

3.3.3. Resultados

El resultado de este sprint fue el código *componentes*[4].

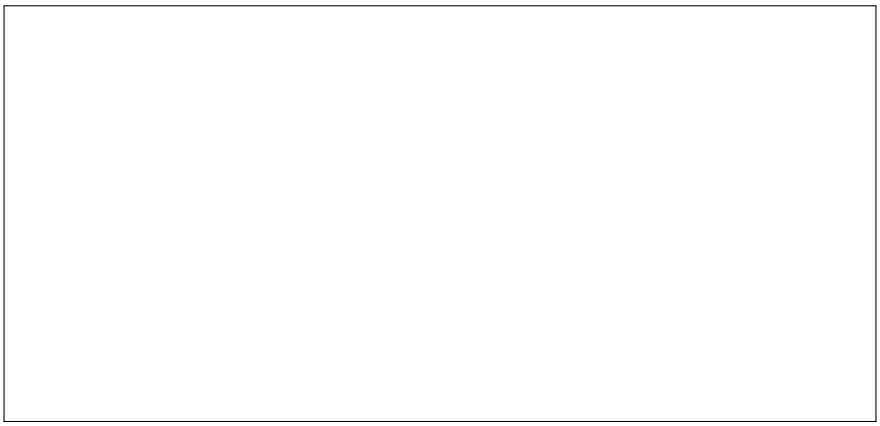


Figura 3.4: Primera demo de manos

En este sprint a nivel visual en la escena no habia grandes cambios ya que el objetivo principal era la optimización de la creación de las manos y la introducción de los eventos. Como se aprecia en la figura 3.4, dentro de la escena se dibujan las manos correctamente y los texton se modifican com oes debido. Para esta demo, para los gestos unicamente se implemento el cambio de texto ya que como aun no se trabajaba con los gestos y hacia falta añadir mas, se queria una forma mas sencilla de comprobar que los eventos se emitian y escuchaban correctamento.

3.4. Sprint 3

Nuevos gestos de las manos

[4] `https://github.com/JuJoarias/TFG/blob/main/first_steps/first_hands/componentes.html`

3.4.1. Objetivos

Para este nuevo sprint, el objetivo principal era introducir mas gestos. Hasta ahora unicamente estaba implementado el gesto de `Pinch`. Durante este sprint se planeo introducir la logica de los gestos `Fist`, `Point` y `Openhand`.

3.4.2. Tareas Realizadas

Para conseguir implementar los nuevos gestos principalmente hizo falta modificar la función de `detectGesture`, aparte de eso, para poder distinguir bien que gesto se esta realizando, en el `init` se definio una variable booleana por cada gesto, las cuales cambian dependiendo de si se esta realizando el gesto o no. Estas variables se implementaron ya que, como la comprobación de los gestos y emisión de estos se realizan en la función `tick`, si no se implementa esta lógica, se emitiria el mismo evento repetidas veces cuando unicamente es necesaria una vez.

Para ello, primero se buscaba la posicion de la punta de todos los dedos y de la muñeca. Una vez con todas las posiciones obtenidas, mediante una función se calculaba la distancia de la punta del dedo correspondiente a la muñeca y si esa distancia era inferior a 0.09 (obtenido luego de varias pruebas para que sea una distancia optima), se definia que el dedo en cuestion estaba doblado. Para cada dedo hay una constante booleana que esta True si el dedo esta doblado y false si esta estirado. Ahora con la información de cada dedo se comprueba si estan estirados o no y dependiendo del estado de los dedos y de si la variable de estado de los gestos esta a True o False, se emite el evento start o end correspondiente a cada gesto.

3.4.3. Resultados

Como resultado, se obtuvo la demo de *nuevos gestos*[5]

[5] `https://github.com/JuJoarias/TFG/blob/main/first_steps/second_hands/nuevos_gestos.html`

Figura 3.5: Primera demo de manos

El resultado fue bastante similar al del sprint anterior, pero con la principal diferencia de que se habian añadido mas gestos a la lógica de las manos como se aprecia en la figura 3.5

Ahora que teniamos una lógica de detección de manos y de gestos era tiempo de crear la la lógica necesaria para poder interactuar con los elementos dentro de la escena, para lo cual, primero era necesario saber cuando las manos estaban colisionando con los elementos de la escena.

3.5. Sprint 4

Incorporación de colliders

3.5.1. Objetivos

Como ultimo paso de preparación de las manos era necesario saber cuando las manos estaban interactuado con los elementos de la escena. El objetivo principal de este sprint era la incorporación de Incorporación de colliders en la escena.

3.5.2. Tareas Realizadas

Para añadir Incorporación de colliders a la escena se hizo uso del componente propio de A-Frame *obb-collider*[6]. La principal ventaja que nos proporciona el uso de este componente

[6] `https://aframe.io/docs/1.7.0/components/obb-collider.html`

nativo de A-Frame, era que cuando detecta una colisión entre 2 entidades que posean dicho componente este lanza un evento indicando el inicio de la colisión y cuando no hay colisión lanza otro indicando el final de la colisión. Los eventos que lanza el componente eran `obbcollisionstarted` y `obbcollisionended`.

Lo principal para añadir los Incorporación de colliders era añadirlos a la mano, ya que añadirlos a los distintos elementos de la escena era tan sencillo como añadir el componente de `obb-collider`. Para ello, habia que modificar la función encargada de actualizar las articulaciones de las manos.

Dentro de la función de `updateSkeleton` 3.5 se añade las siguientes lineas de código:

```
if (!jointEntity.hasAttribute('obb-collider')) {
  jointEntity.setAttribute('obb-collider', `size: ${radius * 2} ${radius * 2} ${radius *
      2}`);
}
```

Listing 3.6: Añadir Incorporación de colliders a la mano

Como se aprecia en el código 3.6, se comprueba si la articulación en cuestion tiene ya un colider y si no lo tiene lo añade.

3.5.3. Resultados

Como resultado de este sprint se obtuvo la demo de *colliders*[7].

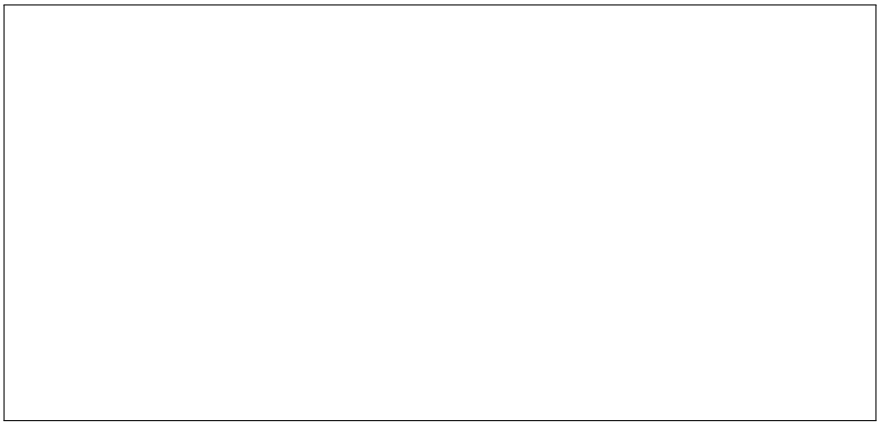


Figura 3.6: Primera demo de manos

[7] `https://github.com/JuJoarias/TFG/blob/main/first_steps/second_hands/colliders.html`

En dicha escena se mantienen los logros que ya habiamos obtenido en los sprints anteriores. Y para comrpobar el funcionamiento de los colliders se añadio a la escena un cubo al que le añadimos el componente de `obb-collider`. En dicha escena se implemento la lógica de que si se detectaba colisión entre el cubo y las manos el color del cubo cambiaria para tener una respuesta mas visual.

Mientras se comprobaba el funcionamiento se noto un error. Como la mano estaba formada por varias esferas con espacios entre medias, y cada esfera poseia un colliders, se emitian varias veces los eventos de inicio y fin de la colisión incluso si no se sacaba la mano del cubo. Esto era un problema ya que estos eventos repetidos cuando no corresponden hacian que cualquier función implementada cuando se detecta colisión actue de forma intermitente y por lo tanto erronea. Aunque este problema no fue identificado hasta el sprint siguiente.

3.6. Sprint 5

Función grabable

3.6.1. Objetivos

Para este sprint se quiso ya ir incorporando el funcionamiento de los gestos en la escena. Para empezar se quiso trabajar con el gesto `Pinch`. Se planteo que si se hacia con la manos derecha sobre el cubo poder arrastrar dicho cubo por la escena en cualquier dirección e imitando la rotación de la mano para que parezca que lo estamos agarrando realmente. Y si se hacia con la mano izquierda, el cubo unicamente se podria deslizar en un unico eje, el eje X.

3.6.2. Tareas Realizadas

Este sprint fue uno de los mas complejos hasta ahora, ya que a lo largo del proceso fuimos encontrando distintos problemas que hicieron replantear la lógica del código.

Para empezar con este sprint, se creo el componente de `Grabable`, dicho componente no posee un `schema`. Y en su init aparte de definir las distintas variables necesarias, se comprueba si la entidad a la que se añada el componente posee un collider, y en caso de no tener le añade uno. En este componente toda la lógica se ejecuta dentro la una función llamada `check`, la cual

se llama en el `tick`.

En dicha función, lo primero que se compruba es si hay una colisión, y si la hay ocurren 2 procesos importantes. El primero, es que se cambia el estado de una variable booleana la cual indica si hay colisión o no, y el segundo, se ejecuta una función llamada `hoover`. En esa función para saber de forma visual que estamos interactuando con el cubo, incluso antes de realizar ningun gesto, se cambia la trasparencia del cubo ligeramente.

Luego, para distinguir entre el pinch de la mano derecha y la mano izquierda escuchamos los eventos de `pinchstart` y `pinchend` ya que el evento lleva la información de la mano que esta realizando el gesto. Con esa información se cambia la variable boolenan correspondiente para saber que mano realiza el gesto. Una vez con esa información ya se podia realizar la lógica del gesto.

Para poder realizar correctamente la lógica del gesto, era necesario poder acceder a la información de las entidades que forman las articulaciones. Para ello, debiamos de ser capaces de acceder a la información del componente `manos` mediante las siguientes lineas de código:

```
this.leftHandEntity = document.querySelector('#left-hand');
this.rightHandEntity = document.querySelector('#right-hand');

const manoDerecha = this.rightHandEntity.components.manos;
const manoIzquierda = this.leftHandEntity.components.manos;
```

Listing 3.7: Acceso información componente manos

Con eso ya somos capaces de acceder a la información de `joints`, donde se almacenaban todas las entidades de las articulaciones.

Para la mano izquierda fue mas sencillo. Se comprobaba si habia solisión con el cubo y tambien si la variable correspondiente al pinch de la mano izquierda estaba en True, y de estarlo, el cubo cambiaba su color a azul y posteriormente se procede a acceder a la información de la punta del indice de la mano izquierda y se hace que el cubo copie la coordenada de dicha articulación mientras mantiene sus otras coordenadas, asi se desliza unicamente en un único eje.

Para la manos derecha fue mas complejo. Se queria poder agarrar el cubo y poder deslizarlo y rotarlo a antojo del usuario. Para ello se planteo hacer un `reparenting`, para volver la entidad del cubo hija de la entidad de la mano para que asi imitara sus movimientos y rotaciones.

Para realizar esto se realizo un *test externo*[8]. El test dio los resultados deseados, pero a la hora de aplicarlo al código resaltaron varios probelmas.

El primero, era que no se lograba realizar el `reparenting` con la entidad de la mano, asi que para solucionarlo se obto por que el nuevo padre del cubo fuese la punta del dedo índice mientras se realizaba el gesto. Otro fallo que resalto fue la rotación. Pese a que el `reparenting` se realizaba correctamente, el cubo no seguia las rotaciones de la mano. Y esto se debia a como se dibujaban las manos, ya que a cada frame se actualiza la posición de cada articulación, pero no la rotación, con lo cual esta se mantiene fija. Para solucionar este problema, se obto por tomar 3 articulaciones de la mano y con estas como referencia para los ejes X y Z se creo un eje de coordenadas relativo. Dicho eje relativo seguiria la orientación y rotación de la mano. Una vez obtenido ese eje cuando se realiza el gesto derecho el cubo empieza a imitar dicho eje, permitiendo que el cubo rote al mismo tiempo que se arrastra el cubo por la escena.

3.6.3. Resultados

El resultado de este sprint se refleja en la demo *hands grabable*[9].

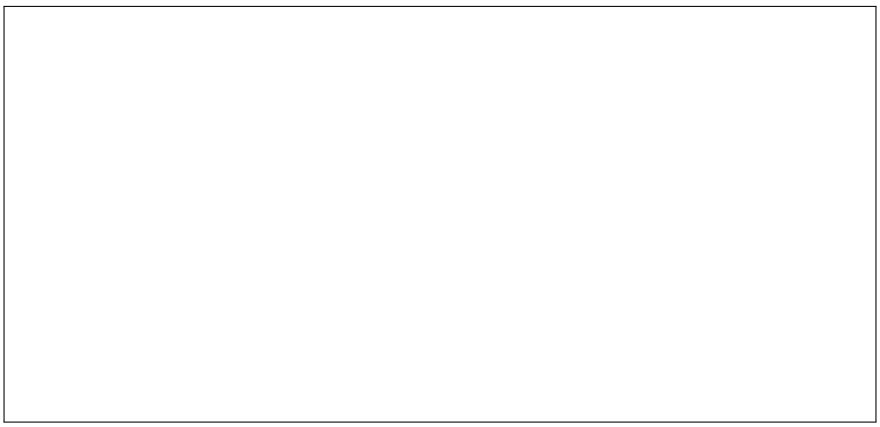


Figura 3.7: Manos Grabable

[8] `https://github.com/JuJoarias/TFG/blob/main/external_tests/reparenting_test.html`

[9] `https://github.com/JuJoarias/TFG/blob/main/first_steps/second_hands/hands_grabable.js`

Figura 3.8: Manos Grabable

Como se aprecia en las figuras 3.7 y 3.8 cada vez que se realiza el gesto con cada una de las manos sobre el cubo este reacciona de manera distinta. El color cabia para tener una respuesta mas visual además de que cada mano tenia se propia reacción.

Con esto se habia conseguido implementar las acciones de `Hoover`, `Drag` y `Slide`. De cara al siguiente sprint se decidio organizar mejor el código de dichas acciones al igual que añadir las acciones de `Stretch` y `Click`

3.7. Sprint 6

Manos finales

3.7.1. Objetivos

El objetivo de este ultimo sprint era añadir nuevas acciones y organizar mejor el código para que quedase lo mas limpio posible. También, por sugerencia del tutor se intento modificar el código para que las manos diseñadas en este proyecto funcionasen junto con el componente ya existente de `Superhands`, aunque esto no se logro al final.

3.7.2. Tareas Realizadas

Para empezar con este sprint, primero se decidio por organizar el código ya existente y las acciones. Para ello, por cada acción que ya estaba implementada , `Hoover`, `Drag` y `Slide`, se creo un componente. También, el componente `Grabable`, creado en el sprint anterior fue

modificado para que en vez de llamar directamente a las funciones encargadas de las acciones de los gestos, emita los eventos `Start` o `End` correspondeintes a cada acción.

Dichos eventos luego serian escuchados por el componente correspondiente. La estructura de los componentes de las acciones todos siguen la misma estructura. En el `Init` primero se comprueba si la entidad a la que fue añadido contiene también el componente `Grabable` y si no lo tiene lo añade ya que este componente es indispensable para que los componentes de acciones funcionen a excepción del componente correspondiente al `Click`. Luego de eso, el componente escucha los eventos de `Start` y `End` y si los escucha llama a una función , la cual inicializa los datos o variables correspondientes que posteriormente se usaran en la función `Tick` del componente. En el siguiente fragmento de código se aprecia el componente `hoover` como ejemplo:

```
AFRAME.registerComponent('hoover', {

  init: function(){
      if (!this.el.hasAttribute('grabable')) {
          this.el.setAttribute('grabable', '');
      }

     this.el.addEventListener('hooverStart', this.onHooverStart.bind(this));
     this.el.addEventListener('hooverEnd', this.onHooverEnd.bind(this));
     this.hooverState = false;
     this.isHoovering = false;
  },

  onHooverStart: function () {
     if (this.isHoovering) return;
     this.isHoovering = true;

     this.hooverState = true;
  },

  onHooverEnd: function () {
     this.hooverState = false;
     this.isHoovering = false;
  },

  tick: function () {
      if (this.hooverState){
         this.el.setAttribute('material', 'opacity', '0.8');
      } else{
```

```
            this.el.setAttribute('material', 'opacity', '1');
        }
    },
});
```

Listing 3.8: Componente Hoover

Para los componentes de `Slide` y `Drag` se ha mantenido la lógica obtenida en el sprint anterior, pero aplicando la estructura mostrada. También, al componente `Slide`, se le añadio un `Schema`. Este componente es el único que lo tiene y en el se puede definir en que eje queremos que se deslice el objeto, ya no se limita al eje X.

Respecto al componente de `Stretch`, la lógica detras del gesto es similar a las acciones previamente implementadas. La única diferencia es que este componente unicamente reacciona cuando se esta haciendo el gesto sobre el mismo objeto con ambas manos a la vez. Una vez activado, el componente comprueba la distancia entre la punta de los dedos indices de ambas manos y con esa distancia la guarda en una variable. Utilizando esa distasncia inicial, se crea un factor de escalado al dividir la distancia actual entre las manos y la inicial. Posteriormente se multiplica dicho factor por la escala del objeto, asi esta crecera o decrecera dependiendo de la distancia entre ambas manos.

Hasta ahora todas las acciones habian funcionado con el gesto `Pinch`, pero para la accion de `Click` era necesario otro. Para este componente se utilizo el gesto de `Point`, el cual consistia en tener el dedo índice estirado. Al relizar el gesto el código crearía un puntero desde la punta del dedo y a cada frame actualizaria su dirección utilizando como vector la línea que une la punta del dedo con su nudillo. Ahora que teniamos el puntero era necesaria una acción para que se emitiera el evento. Para ello, se utilizó el dedo pulgar. Como si se tratase de una pistola, se configuró el código para que cuando se haga el gesto de ”disparar una pistola”se emitiera el evento de click. Toda esta implementación del gesto se introdujo directamente dentro del componente principal, el componente `Manos`. Posteriormente, en el propio componente, al escuchar el evento este componente unicamente cambia el color del cubo a morado y al soltar el gesto de pistola, sin necesidad de soltar el gesto de `Point`, el color vuelve a su color original.

3.7.3. Resultados

El resultado de este sprint es la propia *demo*[10] presentada al final.

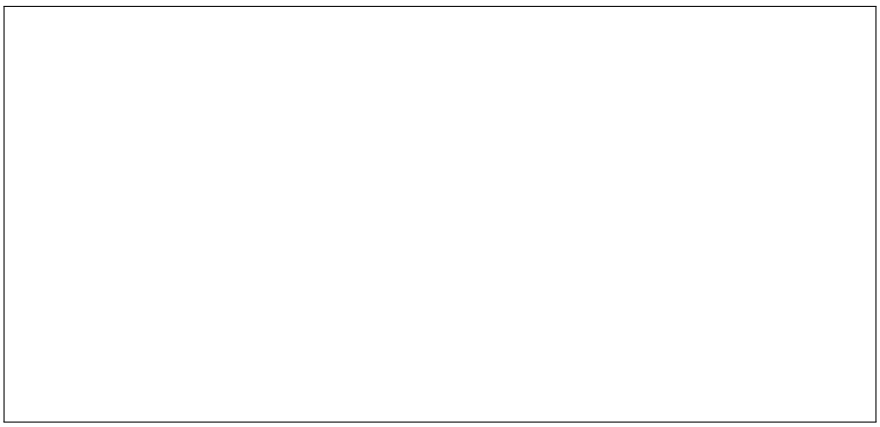


Figura 3.9: Manos finales 1

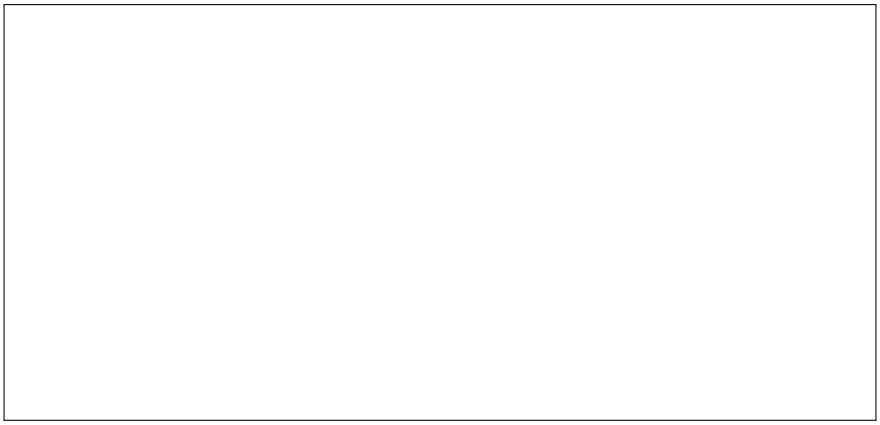


Figura 3.10: Manos finales 2

En las figuras 3.9 y 3.10 se muestran algunas de las acciones implementadas durante este proyecto. En esta demo se aprecia un cubo por cada acción que se implemento al igual que un cubo que posee todas ya que cada cubo unicamente cuenta con uno de los componentes de acción.

[10] `https://github.com/JuJoarias/TFG/blob/main/demo/demo.html`

Capítulo 4

Resultados

Capítulo 5

Pruebas y experimentos

Capítulo 6

Conclusiones

6.1. Aplicación de lo aprendido

6.2. Lecciones aprendidas

6.3. Trabajos futuros

Bibliografía

[1] A-Frame. A-frame - introduction, 2024.
https://aframe.io/docs/1.6.0/introduction/.

[2] Asana. ¿qué es la metodología ágil?, 2025.
https://asana.com/es/resources/agile-methodology.

[3] EncodeBiz. Parte 1: Webgl y su impacto en el desarrollo web moderno, 2024.
https://www.encodebiz.com/blog/parte-1-webgl-y-su-impacto-en-el-desar
cG9zdDozMjc=.

[4] E. Freeman and E. Robson. *Head First HTML and CSS*. O'Reilly Media, 2nd edition, 2018.

[5] Git SCM. Branching and merging, 2024.
https://git-scm.com/about/branching-and-merging.

[6] Google Developers. Introducción a three.js, 2025.
https://web.dev/articles/three-intro?hl=es-419.

[7] LaTeX Project. About the latex project, 2025.
https://www.latex-project.org.

[8] Meta. Company information, 2024.
https://www.meta.com/es-es/about/company-info/.

[9] Onirix. Webxr: Ejemplos y desarrollo en realidad extendida, 2024.
https://www.onirix.com/es/webxr-ejemplos-desarrollo-realidad-extendid

[10] W3C. The w3c mission, 2024.
`https://www.w3.org/mission/`.

[11] WHATWG. Whatwg - web hypertext application technology working group, 2024.
`https://whatwg.org/`.