



INGENIERÍA EN TECNOLOGÍAS DE LA TELECOMUNICACIÓN

Curso Académico 2019/2020

Trabajo Fin de Grado

INTERFACES FOR BUILDING SCENES IN VIRTUAL REALITY

Autor : Javier Jesús Bravo Donaire

Tutor : Dr. Jesús María González Barahona

Trabajo Fin de Grado/Máster

Interfaces for Building Scenes in Virtual Reality

Autor : Javier Jesús Bravo Donaire

Tutor : Dr. Jesús María González Barahona

La defensa del presente Proyecto Fin de Carrera se realizó el día de
de 20XX, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 20XX

*Dedicado a
mi familia / mis amigos / mis compañeros*

Agradecimientos

Página de agradecimientos en proceso.

Resumen

El proyecto realizado se centra en la creación de una interfaz para la construcción de escenas en realidad virtual, esto es, crear una escena de realidad virtual, manipulando entidades desde dentro de una escena de realidad virtual. Este trabajo ha sido creado como un programa JavaScript que se apoya en el módulo A-Frame para producir experiencias en realidad virtual, mediante la abstracción y manipulación del DOM.

La escena es accesible desde un navegador como con un dispositivo de realidad virtual, siempre que estos dispositivos tengan la capacidad para ello, la cual aporta la API WebVR.

Fusionando los elementos HTML con el lenguaje JavaScript, podemos manejar distintos eventos, para añadir dinamismo a la escena. A-Frame, el cual está construido sobre Three.js, se encargará de generar el entorno, y agregando la biblioteca WebGL se renderizarán todos los elementos.

Todos los ejemplos y el código se encuentran alojados en GitHub, para el uso y disfrute de cualquier usuario.

Summary

The current project's aim is the creation of an interface for building scenes in virtual reality, that is, creating a virtual reality scene, manipulating entities from within another virtual reality scene. This work has been created as a JavaScript program that relies on the A-Frame module to produce experiences in virtual reality, through the abstraction and manipulation of the DOM.

The scene is accessible from a browser or a virtual reality device, as long as these devices have the capacity to do so, which provides WebVR API.

Fusing HTML elements with JavaScript, we can handle different events, with the aim of providing dynamism. Furthermore, A-Frame is built on Three.js. It will generate the environment, rendering all the objects through WebGL.

All the code and examples are hosted on GitHub, for the use and enjoyment of any user.

Indice General

1	Introducción	1
1.1	Contexto	1
1.2	Objetivo principal	2
1.3	Objetivos específicos	3
1.4	Planificación temporal	3
1.5	Estructura de la memoria	4
2	Tecnologías	7
2.1	A-Frame	7
2.2	HTML5	8
2.3	JavaScript	10
2.4	WebGL	11
2.5	Three.js	11
2.6	WebXR	12
2.7	Git	13
2.8	GitHub	14
2.9	Atom	15
2.10	LaTeX	15
3	Diseño e implementación	17
3.1	Sprint 0	18
3.1.1	Objetivo	19
3.1.2	Desarrollo	19
3.1.3	Resultado	24

3.2	Sprint 1	24
3.2.1	Objetivo	24
3.2.2	Desarrollo	25
3.2.3	Resultado	30
3.3	Sprint 2	30
3.3.1	Objetivo	31
3.3.2	Desarrollo	31
3.3.3	Resultado	34
3.4	Sprint 3	34
3.4.1	Objetivo	35
3.4.2	Desarrollo	35
3.4.3	Resultado	38
4	Resultado Final	41
4.1	Manual de Usuario	41
4.2	Arquitectura resultante	43
5	Conclusiones	49
5.1	Consecución de objetivos	49
5.2	Aplicación de lo aprendido	50
5.3	Lecciones aprendidas	51
5.4	Trabajos futuros	51
	Bibliography	53

Lista de Figuras

2.1	Escena de ejemplo Introducción a Aframe	9
2.2	Estructura en forma de árbol DOM	10
2.3	Estructura de ramas Git	13
3.1	Estructura de la metodología Scrum	19
3.2	Punto de partida Minijuego	20
3.3	Plataformas para subir de nivel	20
3.4	Piedras que bloquean el camino y bloque gigante	21
3.5	Obstáculo de baldosas con movimiento	22
3.6	Escena victoria	25
3.7	Panel con botones para generar entidades	26
3.8	Entidades generadas con el panel	27
3.9	Mesa con entidades holográficas	28
3.10	Paleta de colores perteneciente a una entidad	29
3.11	Versión funcional conseguida al final del Sprint 1	30
3.12	Panel con botones para colores y texturas	33
3.13	Escena con panel de edición y barra deslizable	34
3.14	Wireframe de distintas entidades	36
3.15	Escena en modo edición	37
3.16	Panel con botones de modo edición versión VR	38
4.1	Versión con gafas de realidad virtual	42
4.2	Escena inicial versión de Escritorio	43
4.3	Escena compleja con todas las funcionalidades	44

4.4 Escena compleja mediante dispositivo de realidad virtual 48

Capítulo 1

Introducción

En este capítulo se tratan los objetivos tanto principales como específicos, el contexto en el que surgió esta idea, la planificación temporal para la elaboración del mismo, finalizando con una explicación clarificadora sobre la estructura seguida en este escrito.

El fin de este estudio es la elaboración de una interfaz en realidad virtual, disponible desde cualquier navegador. El nacimiento de esta idea surgió con el deseo de introducir a usuarios con un conocimiento bajo sobre los distintos programas de edición de escenas o diseño de modelos 3D, a un entorno intuitivo y de veloz aprendizaje.

El esqueleto del trabajo se compone del framework de creación de experiencias en realidad virtual, A-Frame. A este se le unen otras tecnologías que ayudan a moldear y mejorar este esqueleto, donde se incluyen JavaScript o HTML, entre otras.

1.1 Contexto

En los últimos años, el sector tecnológico es uno de los departamentos que más crecimiento ha demostrado, consiguiendo récords de desarrollo en 2018 y 2019. Una de las ramas que se diferencian son la realidad virtual y realidad aumentada, originada en la Segunda Guerra Mundial por la necesidad de crear simuladores de vuelo para instruir a los pilotos. Ya en el año 1960, Morton Heilig¹ patenta el primer prototipo de visor que reproduce diapositivas en 3D y sonidos estéreos.

Durante décadas, los avances en este ámbito se han ido desarrollando positivamente hasta

¹https://en.wikipedia.org/wiki/Morton_Heilig

lo que tenemos actualmente:

- Distintas marcas desarrollan videojuegos en realidad virtual (Sony, Microsoft).
- Múltiples empresas exponen distintos dispositivos para acceder a estos entornos (Oculus, HTC Corporation).
- Simulaciones de vuelo y entrenamiento de pilotos (Fórmula 1, MotoGP).
- En medicina, se usan como simuladores de operaciones y al igual que el anterior punto, para el adiestramiento de personal sanitario lo que evita el gasto de material.
- Muchos museos usan el sistema de realidad aumentada para mostrar escenas de distintas épocas al espectador.

Todas ellas coinciden en su sencillez para un usuario inexperto y en el atractivo de imaginarte en un mundo paralelo donde puedes hacer lo que te plazca. Siguiendo las anteriores aplicaciones, muchas se encuentran en constante desarrollo y son el aliciente del surgimiento de este proyecto.

El modelado 3D, la creación de escenas o la representación de datos en realidad virtual son ejemplos de las muchas ideas que se encuentran en auge en la actualidad. Aplicaciones de escritorio, como podría ser Blender², adaptadas en un entorno de realidad virtual donde tu puedas modificar el modelo 3D con tus propias manos, o recorrer una ciudad donde cada edificio representa un archivo de tu ordenador e incluso una red de ordenadores donde puedas observar todos los paquetes de comunicación y desplegarlos para ver su información, son ejemplos del alcance al que se puede llegar gracias a la realidad virtual. Es en esta línea donde se ha querido centrar el proyecto fin de grado que nos ocupa, la creación de escenas.

1.2 Objetivo principal

El objetivo principal de este proyecto es el desarrollo de un entorno de realidad virtual que permita al usuario realizar sus propias escenas, desde dentro de otra escena de realidad virtual, mediante la selección y edición de múltiples entidades de un modo veloz, comprensible y enriquecedor.

²<https://www.blender.org/>

1.3 Objetivos específicos

Objetivos específicos a los que se irán dando solución según avance el trabajo:

- La interfaz debe funcionar sobre el framework A-Frame.
- La interfaz debe funcionar dentro del navegador.
- El editor de escenas debe disponer de un panel donde el usuario pueda decidir que objeto crear y poder crear copias infinitas de él.
- Cada entidad contará con una paleta de colores y texturas que permita al usuario decidir el estilo.
- El usuario tendrá la capacidad de elegir el tamaño de las figuras.
- Adaptación de la versión de escritorio a una versión de realidad aumentada compatible con cualquier dispositivo de visualización.
- Dotar de movilidad al usuario tanto para la versión de escritorio como la de realidad virtual.
- Asignar dinamismo a la cámara para simular el movimiento y conectarlo con el sonido estéreo con el objetivo de hacer sentir al usuario una experiencia completa.
- La escena debe tener un modo de edición que represente el esqueleto de las entidades así como los ejes de esta y de la propia escena, para facilitar al usuario la visión y obtener una referencia sobre el ancho y alto real.
- La accesibilidad para cualquier usuario al código y a los ejemplos mediante la plataforma GitHub, además de permitir modificarlo y añadir funcionalidades.

1.4 Planificación temporal

Este proyecto se inició con la apertura del segundo semestre de curso, y se ha ido desarrollando mientras terminaba este y realizaba las prácticas en empresa. Para ello, seguimos el modelo

Scrum, en el cual, el proyecto se divide en distintas etapas denominadas "Sprints". Para distinguir estas etapas, manteníamos reuniones (al principio presenciales y más tarde por videoconferencia) en las que se resolvían dudas y se aclaraban los distintos objetivos a conseguir en cada "Sprint". La duración de este han sido un total de 9 meses, en los cuales ha habido temporadas más activas y otras más calmadas debido a exámenes o trabajo, pero en todas ellas el periodo de trabajo era por las tardes después de clase, y sobre todo los fines de semana. En general, todo el proceso de creación de este proyecto se puede simplificar en cuatro fases:

- Decisión del objetivo. Esta fase consta de las reuniones que llevé a cabo mi tutor Jesús. En ellas, se hablaba de las distintas corrientes de estudio que se podían seguir, decantándose al final por la presente.
- Adaptación a la tecnología. Para adentrarme en las tecnologías mencionadas más adelante en el capítulo 2, decidimos realizar un minijuego que me permitiese aprender el funcionamiento de Aframe 2.1 o aprender lenguajes de programación como JavaScript 2.3.
- Ejecución de los objetivos. Las distintas reuniones marcaban el inicio y final de las distintas etapas con las que avanzaba el proyecto. Es la fase más amplia de todas ya que es en la que mas documentación es necesaria para avanzar y más tiempo se invierte en la escritura de código, pruebas, etc.
- Redacción de la memoria.

1.5 Estructura de la memoria

Para una correcta lectura del presente proyecto, se aclara la estructura que se sigue a continuación:

- En este primer capítulo se presenta una introducción al proyecto, exponiendo sus objetivos y contexto actual, así como la planificación temporal de todo el proyecto.
- En el capítulo 2 se muestran las distintas tecnologías que se enlazan a lo largo del proyecto con algunos ejemplos de uso.

- A continuación, se presenta el proceso de desarrollo en el capítulo 3. Además, en él se explica detalladamente el modelo Scrum, los diferentes Sprints y tanto los problemas como objetivos que se van solucionando.
- El capítulo 4 muestra el resultado final desde dos distintas perspectivas, una en la que se expone una guía para el usuario y otra más técnica donde se explica la arquitectura resultante, focalizándose en los componentes que implementa la escena.
- Las conclusiones quedan recogidas en el capítulo 5, donde se analizará todo lo aprendido durante este periodo, los objetivos logrados y los problemas resueltos.

Capítulo 2

Tecnologías

2.1 A-Frame

La gran base de este proyecto es A-Frame[7], un framework cimentado sobre JavaScript que nos ofrece la oportunidad de crear escenas en VR (Virtual Reality o Realidad Virtual) sin la necesidad de instalar ninguna dependencia. Gracias a la unión de HTML, JavaScript, WebGL, WebXR y Three.js (comentados en sus respectivos apartados más adelante), A-Frame lleva la arquitectura ECS (Entity-Component-System o Entidad-Componente-Sistema) a otro nivel. Con esta arquitectura, podemos enlazar componentes a entidades, actualizarlos e incluso eliminarlos. Al obtener el DOM característico de HTML, podemos acceder a las entidades con sentencias simples (Query Selectors).

Se definen así, entidades y componentes. Una entidad es un objeto que tiene sus propiedades como cualquier elemento HTML; en cambio, un componente es un comportamiento especial que se puede asignar a una entidad. La relación entidad-componente será la esencia de aquellos trabajos realizados en A-Frame.

A-Frame tiene su propia sección de tutorial práctico en el que se introduce al usuario a los distintos elementos y entidades, realizando pequeñas escenas. El código de ejemplo¹ que propone la documentación de A-Frame, y que da lugar a la imagen de la Figura 2.1 es el siguiente:

```
<html>
  <head>
    <script src="https://aframe.io/releases/1.0.4/aframe.min.js"></script>
  </head>
```

¹<https://aframe.io/examples/showcase/helloworld/>

```

<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>

```

Como se puede observar, dentro del campo "head" del código, añadiremos todas las dependencias necesarias. En este caso, al ser un ejemplo sencillo, es necesario únicamente el script básico de A-Frame. En el elemento "body" se inserta la escena (que agrupará todos los elementos que posteriormente queramos añadir). En él, se encuentran un total de cinco entidades: un cubo, una esfera, un cilindro, un plano y el cielo, respectivamente. Todas ellas contienen unos atributos únicos que definen su tamaño, color o posición.

Por el momento, no se añade ningún componente en la escena, ya que se usarán en el capítulo 3, pero se añadirían como si de un atributo más se tratase. Estos son creados mediante JavaScript, y se dividen en schema (propiedades del componente) y las funciones init (se ejecuta al inicializar la entidad), update (se ejecuta cuando se modifica la entidad), tick (se ejecuta cada milisegundo), remove (se ejecuta al eliminar la entidad), pause y play. Lo adecuado sería alojarlos en un fichero distinto a este y enlazarlo en el elemento "head" al igual que se ha añadido el script básico de A-Frame.

2.2 HTML5

HTML (HyperText Markup Language)[10] es el lenguaje de marcado empleado para la creación y estructuración de páginas web. Actualmente se encuentra en su versión HTML5 publicada en 2014 y es en la que se ha desarrollado este proyecto. Algunas de sus novedades frente a su anterior versión son:

- Se incluyeron nuevas etiquetas al documento: footer, header, section, nav, aside, etc.

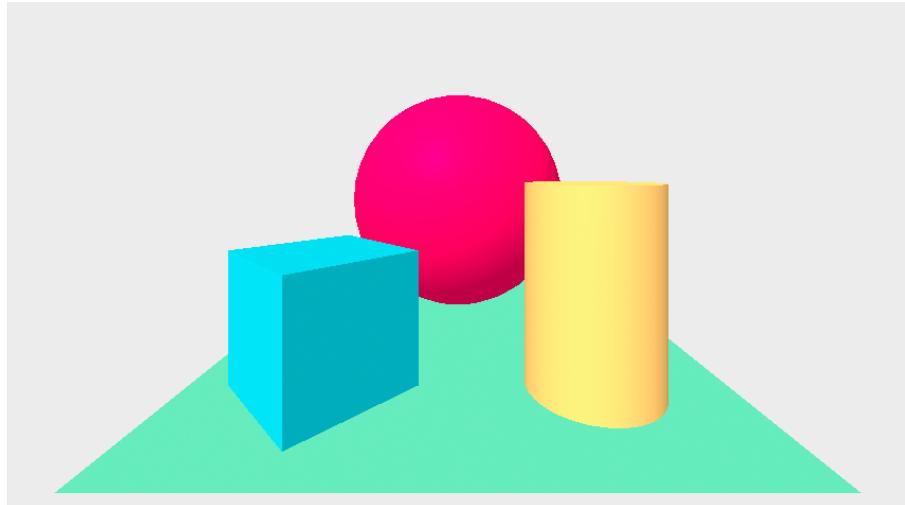


Figure 2.1: Escena de ejemplo Introducción a Aframe

- Nuevos componentes de audio y video para la manipulación de elementos multimedia.
- Los formularios ahora contienen valores adicionales para la etiqueta input y se incorpora el elemento output.
- Nueva interfaz para arrastrar y soltar elementos.
- Trabajo off-line descargando todos los contenidos necesarios.
- Geolocalización
- Ejecución de tareas paralelas (WebWorkers) e interfaz de comunicación entre páginas (WebSockets).
- Almacenamiento de datos en el propio navegador gracias a una base de datos (API Storage).
- Se incluyen muchas APIs, entre las que se encuentran WebGL, de la cual se habla más adelante, o Canvas.

Para finalizar, resta detallar el papel del DOM (Document Object Model) en un documento HTML. Consiste de una interfaz que nos proporciona una estructura lógica de todos los elementos implicados en el documento y los organiza en nodos, apodando a esta unión árbol de nodos (Figura 2.2). Esto crea una potente unión entre las entidades mencionadas antes en Aframe y

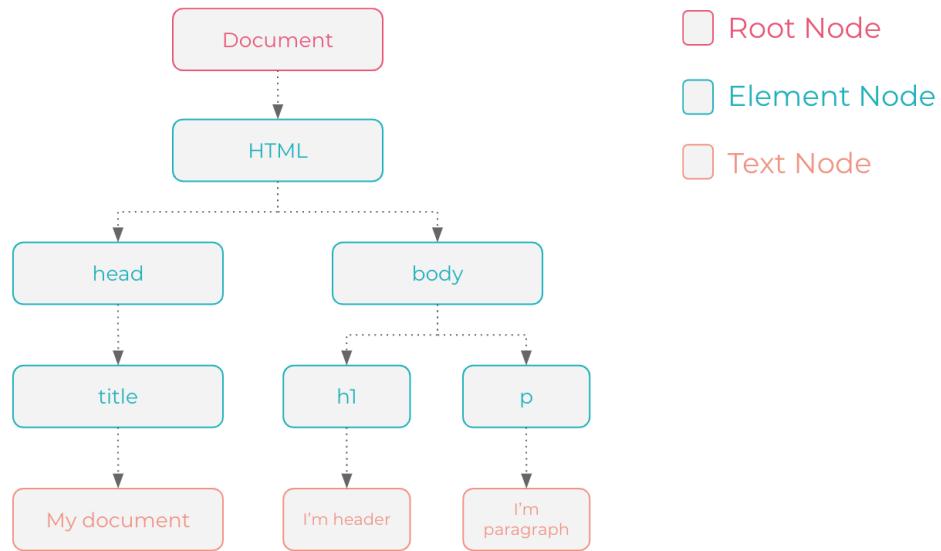


Figure 2.2: Estructura en forma de árbol DOM

lo que comentaremos a continuación en el apartado de JavaScript, gracias al DOM podremos acceder a ellas y manejarlas a nuestro antojo.

2.3 JavaScript

JavaScript[4] es un lenguaje de programación interpretado, lo que significa que no necesita de compilación para su ejecución. Su fin consiste en infundir dinamicidad a nuestras aplicaciones web mediante distintas animaciones, manejo de eventos y modificaciones del DOM en general. Como cualquier otro lenguaje de programación puede realizar operaciones, tratamiento de strings, almacenamiento de variables y métodos algorítmicos.

Acceder al DOM gracias a JavaScript se convierte en una tarea sencilla además de potente. El código HTML es estático, pero con JavaScript podemos alterarlo en vivo con distintos eventos. Gracias a sentencias como la siguiente podemos acceder a cualquier elemento, por ejemplo, al título:

```
document.getElementsByTagName("h1");
```

Dicho esto, igual que accedemos al título, podemos acceder a un componente de nuestra es-

cena en A-Frame: cubos, cilindros, el cielo o incluso a la cámara de nuestra escena. Nos aporta infinidad de opciones: crear botones que activen eventos, movimiento a distintas entidades, etc. Junto con todo lo que nos ofrece A-Frame, JavaScript será la base este proyecto.

Habitualmente, el código JavaScript se encuentra en el lado del cliente, pero debido a su gran popularidad se creó una versión con la capacidad de ser ejecutada en el lado del servidor, Node.js.^[5] Al estar basado en JavaScript, mantiene las ventajas de este, sigue siendo software libre y además, es asíncrono. Su función principal es el manejo de solicitudes de entrada y la salida de respuesta, llegando incluso a poder realizar una solicitud HTTP.

2.4 WebGL

WebGL^[9] define una API (Application Programming Interfaces o Interfaz de Programación de Aplicaciones) basada en OpenGL² cuyo fin es la representación tanto en 3D como en 2D de elementos gráficos. Como se ejecuta en el elemento "canvas" de HTML, se integra a la perfección con la interfaz del DOM. Su creación nació en 2006 con el primer prototipo de Canvas 3D presentado por Vladimir Vukicevic³, y no fue hasta principios de 2009 que el consorcio Kronos Group consolidó el WebGL Working Group junto con Google, Mozilla, Apple y Opera, entre otros. Finalmente, en 2011 fue lanzada la primera versión con sus correspondientes especificaciones.

Al estar asentado en OpenGL, su código está escrito en JavaScript y "shading language". Shading language o lenguaje de sombreado, es un lenguaje de programación basado en la creación de efectos de sombreado, así como superficies, volúmenes, etc. Usa unos tipos de datos más específicos que otros lenguajes, como pueden ser: vectores, matrices o colores. Para el caso de OpenGL, recibe el nombre de GLSL o "glslang".

2.5 Three.js

Three.js^[6] es una biblioteca escrita en JavaScript que se ayuda de WebGL para pintar en 3D. Three se encarga de tareas más específicas que las que aporta WebGL, lo que conlleva a es-

²<https://www.opengl.org//documentation/>

³https://en.wikipedia.org/wiki/Vladimir_Vuki%C4%87evi%C4%87

cribir un poco más de código. Estas tareas son la inyección en nuestro proyecto de texturas, iluminación, sombras, etc.

Actualmente se encuentra en constante desarrollo, y al estar alojada en GitHub⁴, se pueden crear funcionalidades nuevas siguiendo determinadas reglas comentadas en el apartado correspondiente a GitHub.

2.6 WebXR

WebXR [8] es un grupo de estándares que apoyan a la creación de escenas 3D tanto en realidad virtual como en realidad aumentada, que junto con Aframe, WebGL y Three.js permitirán el correcto funcionamiento de la escena y la apropiada ejecución en un dispositivo XR. Es la sucesora de WebVR, la cuál se ha dado como desactualizada este mismo año (2020), y añade la compatibilidad con realidad aumentada, de ahí que el nombre cambie de WebVR a WebXR. La API de WebXR aporta unas funcionalidades extra de las cuales, el usuario puede no necesitar hacerse cargo:

- Duplicar la salida 3D (en el dispositivo) a una 2D en el caso que fuese necesaria, por ejemplo, para la depuración de una aplicación.
- Crear los vectores que representan los movimientos que transmiten los controles de entrada (mandos).
- Renderizar la escena que se envía al dispositivo para que tenga el ratio de frames por segundo adecuado para su correcto funcionamiento.
- Averiguar si el dispositivo de salida es compatible con realidad aumentada o con realidad virtual.

Los dispositivos XR serán entonces, aquellos que pueden presentar imágenes al usuario e incorporen un sistema de rastreo de movimiento y orientación, permitiendo la inmersión completa de este.

⁴<https://github.com/mrdoob/three.js/>

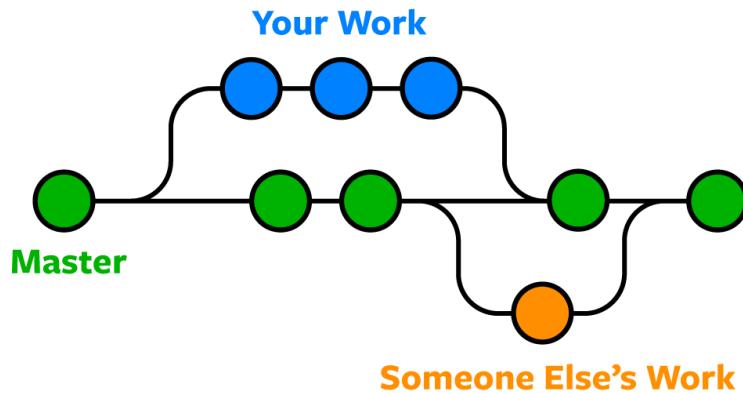


Figure 2.3: Estructura de ramas Git

2.7 Git

El propósito de Git es la eficiencia de producción y el mantenimiento de un registro de los cambios que se producen, con el fin de coordinar el trabajo compartido. Para lograr esto, se aclaran unas normas a seguir que reciben el nombre de Flujo de Trabajo.

Git sigue una estructura de árbol, en la que cada rama representa un flujo de trabajo. La rama principal recibe el nombre de "master", esta rama contiene la versión funcional del proyecto que se esté llevando a cabo, es la rama de producción. Para crear esta rama necesitamos establecer un repositorio en el que poder añadir nuestros archivos y añadirlos. Git tiene una serie de comandos para realizar todas estas acciones, por ejemplo, para crear el repositorio usamos *git init*, y para añadir nuestros archivos (consiguiendo así protegerlos) *git commit* seguido de *git push*.

Para prevenir que se trabaje todo sobre la versión de producción, Git aporta un comando para crear ramas adicionales en las que distintos grupos de trabajo pueden trabajar para evitar modificar los mismos archivos y eliminar sus propios avances, *git branch*. Una vez terminada la funcionalidad en la que se trabaja en una rama aparte, se debe enlazar con la rama maestra, para ello empleamos el comando *git merge*. Se forma así una estructura de árbol como la que podemos observar en la Figura 2.3, en la que cada nodo corresponde con una representación de nuestros archivos comprometidos, lo que se llama "log". Gracias a esto, podemos volver a antiguos nodos del árbol para recuperar versiones anteriores, eliminar otros, etc.

2.8 GitHub

GitHub[12] es una de las mayores plataformas de almacenamiento de software y proyectos basado en el sistema de control de versiones Git[2], creado por Linus Torvalds⁵ en el año 2005.

Tomando como base Git, GitHub nos permite visualizar en su plataforma web el flujo de trabajo conseguido. La principal diferencia entre ambos consiste en que GitHub nos aporta una interfaz gráfica de nuestro árbol, mientras que Git solo funciona a través de comandos en una terminal. Asimismo, GitHub añade más herramientas que nos aportan claridad y orden al proyecto:

- Creación de perfiles para cada usuario donde se muestran los repositorios en los que trabaja (tanto propios como conjuntos), junto con estadísticas de trabajo: lenguajes de programación más usados, historial de producción, etc.
- Página de "wiki" para cada repositorio en la que se puede comentar cada apartado de este para que los lectores puedan apoyarse.
- Una sección llamada "issues" donde los usuarios pueden dejar un mensaje comentando cualquier fallo que hayan encontrado en nuestro trabajo.
- La casilla "projects" permite crear tareas que se pueden asignar a distintos grupos de trabajo para tener un control de las funcionalidades que se están creando, así como creación de columnas para ordenar estas tareas según si se encuentran finalizadas, en proceso o por iniciar.
- Capacidad de copiar un repositorio ajeno para trabajar sobre él. Esto es lo que se conoce como "fork", nos permite editar los archivos y en el caso de mejorar alguna funcionalidad, podemos solicitar al autor una unión (merge) con lo que se denomina "pull request".

Todas estas cualidades mostradas, han sido las responsables para decidirme a alojar y proteger mi proyecto en GitHub⁶.

⁵https://es.wikipedia.org/wiki/Linus_Torvalds

⁶<https://github.com/JavierBravoDonaire/A-frame>

2.9 Atom

Atom[1] ha sido el editor de textos elegido para aposentar todo el código necesario para el funcionamiento de esta idea. Gracias a tratarse de un programa de código abierto, podemos encontrar infinidad de bibliotecas para modificarlo a nuestro gusto. Entre sus ventajas se encuentran:

- Autocompletado de sentencias. Si existe un paquete relacionado con el lenguaje de programación que se va a usar, Atom completará las sentencias añadiendo los paréntesis, corchetes, tabuladores o elemento que se necesite.
- Reglas de estilo. Semejante al anterior punto, dotará de distintos estilos al código, las variables serán de un color, los nombres de función de otro, etc.
- Sincronización con repositorios. Una de las herramientas más potentes es la capacidad de sincronizar con GitHub tu proyecto. Esto permite a Atom mostrarte los archivos que han sido modificados, los que no se han comprometido, los nuevos o los que siguen intactos.
- Edición múltiple. Tolera varias flujos de escritura a la vez, por ejemplo si necesitamos añadir a varios elementos el mismo atributo, escribiéndolo una vez, se escribiría en todos ellos sin necesidad de copiar y pegar.

2.10 LaTeX

LaTeX[11] trata de un procesador de textos de software libre típicamente utilizado para textos técnico-científicos. Su objetivo principal es separar las reglas de estilo del contenido. La diferencia más importante con respecto a otros editores de textos es su estructura basada en instrucciones. Entre sus ventajas se encuentran la capacidad de establecer órdenes al inicio de nuestro texto que afectarán a este, o la despreocupación a la hora de situar figuras y tablas, ya que como se ha comentado antes, él se encarga del estilo, nosotros del contenido.

Al tratarse de un software libre su uso es completamente gratuito (además, de que se mantiene siempre en desarrollo) y se encuentra disponible para múltiples sistemas operativos.⁷

⁷<https://www.latex-project.org>

Capítulo 3

Diseño e implementación

Para poder explicar todo el proceso de diseño del proyecto, se debe comenzar hablando de Scrum[3]. Scrum es una metodología de trabajo desarrollada en el año 1993 por Jeff Sutherland¹ y Ken Schwaber², y finalmente formalizada en el año 1995. El objetivo de esta es conseguir un desarrollo ágil y productivo con la mejor organización posible para llegar a un producto completo. En ella, se define que el trabajo realizado debe organizarse en distintos espacios de tiempo denominados iteraciones o "Sprints" en los que se definen una serie de objetivos, y no se da paso al siguiente "Sprint" hasta que se solucionan los problemas y se consigue todo lo planeado en el anterior.

Algunas de las ventajas del uso de Scrum son las siguientes:

- Los "Sprints" se realizan de manera mensual (o de varias semanas), lo que permite un avance rápido y marcado del trabajo, además de conseguir gradualmente resultados antes del producto final.
- Existe una flexibilidad entre el cliente y el equipo de desarrollo que permite el intercambio de ideas o soluciones de problemas, debido a las reuniones mensuales.
- Rígida definición de plazos y tareas.
- Reducción de la complejidad de un proyecto complejo gracias a su desglose en etapas.
- Mayor satisfacción moral del equipo y por consiguiente, mejora del ambiente de trabajo.

¹<https://www.scrumguides.org/jeff.html>

²<https://www.scrumguides.org/ken.html>

El equipo que emplee esta metodología debería dividirse en tres partes: el Scrum Master, el Equipo de Trabajo y el Cliente. En este caso, el papel de Scrum Master lo lleva a cabo el profesor tutor, (aunque también podría adquirir el papel de Cliente) y el papel del Equipo de Trabajo lo representa el alumno que realiza el trabajo fin de grado. El objetivo del Scrum Master consiste en asegurar el entendimiento de las reglas y prácticas llevadas acabo. Además, mantiene una estrecha relación con el Cliente haciendo de intermediario entre este y el Equipo de Trabajo, consiguiendo y aclarando los objetivos propuestos y aportando técnicas para una efectiva producción. Por otro lado, el Equipo de Trabajo esta formado por los profesionales que se encargar del desarrollo y resolución de problemas que propone el Cliente. Son los encargados de la consecución de los objetivos para poner fin a los distintos "Sprints".

Para este caso en particular, cada reunión entre el tutor (Scrum Master/Cliente) y el alumno (Equipo de Desarrollo), marcarán el inicio y final de cada "Sprint". Inicialmente, una reunión marcará los objetivos a conseguir en ese "Sprint", durante unas semanas se trabaja en ello y finalmente se prepara una nueva reunión. En esta nueva reunión, se hablan de los problemas que han surgido y lo conseguido. Si se han logrado con éxito cada una de las metas propuestas, esta reunión será la que dará fin a ese "Sprint" e iniciará el siguiente. Normalmente, en cada "Sprint" se crea una versión funcional que implementa lo conseguido y da paso como versión inicial al siguiente. En caso contrario, se tratan las dudas y problemas surgidos intentando llegar a una solución. Se trabaja en esto durante un periodo y se vuelve a tener la reunión hasta conseguir la versión funcional y dar paso al siguiente "Sprint" hasta llegar a la versión final del producto, como podemos observar en la Figura 3.1.

3.1 Sprint 0

Esta iteración comienza con la primera reunión que se mantuvo entre el Scrum Master y el equipo de desarrollo. En ella se acordó que para empezar el equipo debía iniciarse en A-Frame junto con JavaScript (hasta el momento desconocido) y HTML entre otros, por lo que la idea sería crear un minijuego básico que uniese todas las tecnologías.

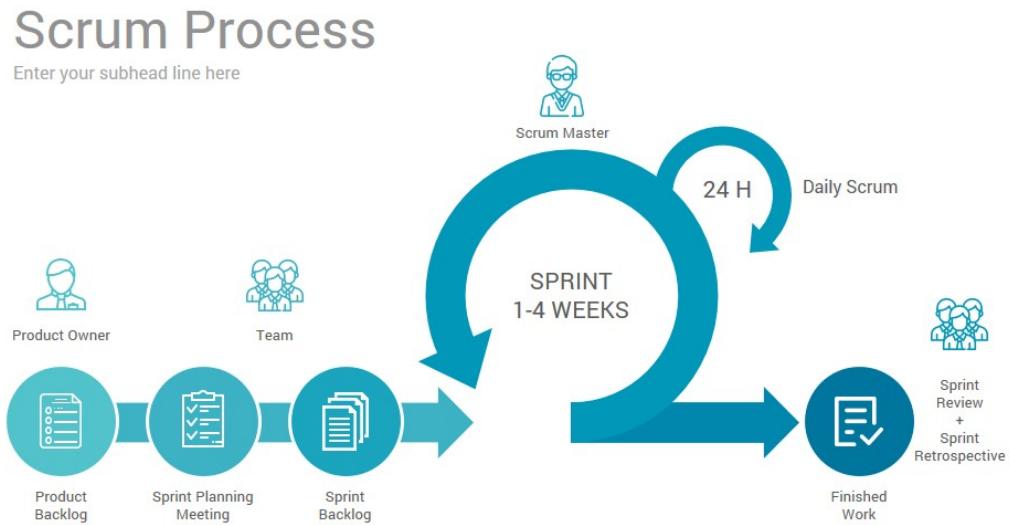


Figure 3.1: Estructura de la metodología Scrum

3.1.1 Objetivo

El objetivo de esta fase es aprender a usar A-Frame, en esto se incluye: aprender que son las entidades, crear entidades, crear eventos, crear componentes, dinámicas y físicas, etc. Como JavaScript es aun desconocido, otro gran objetivo es ir aprendiendo a la vez que se enlaza con A-Frame. El fin del minijuego es hacer un pequeño recorrido en el tengas que realizar distintas acciones para llegar a la meta.

3.1.2 Desarrollo

La idea es crear un recorrido con distintos obstáculos que implementen distintas acciones y eventos para adecuarme a ellos. El recorrido es el que sigue a continuación:

1. Apareces en un estrecho pasillo con una puerta delante, la cual debes abrir llamando al timbre y evitando los dos obstáculos que tratarán de tirarte como se puede observar en la Figura 3.2.
2. Una vez abiertas las puertas, el pasillo continúa dejando paso a dos plataformas que te permiten llegar al segundo piso (Figura 3.3). Para ello, debes subirte en una de las dos plataformas en el momento correcto.

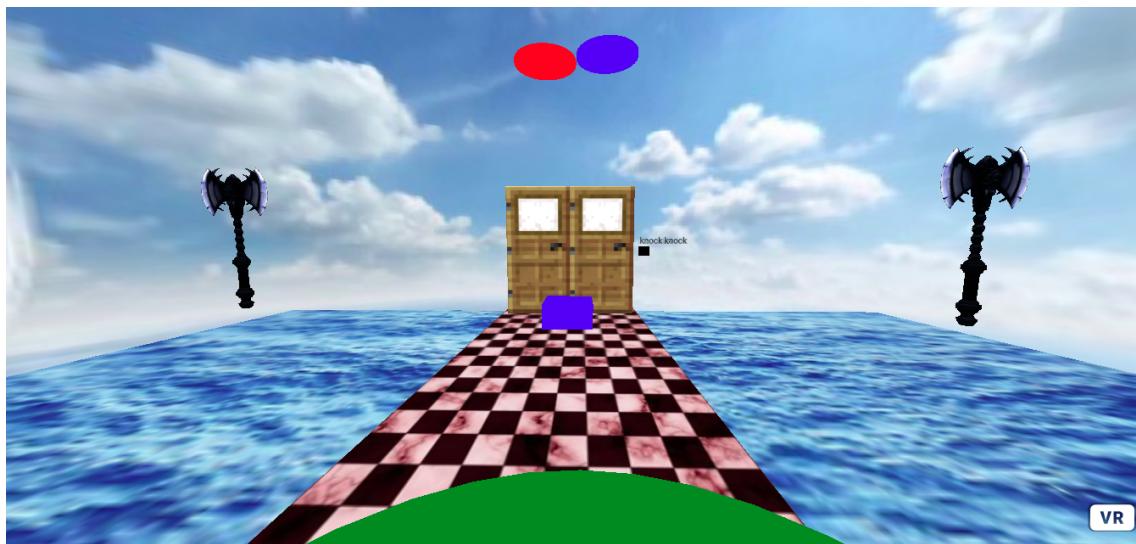


Figure 3.2: Punto de partida Minijuego

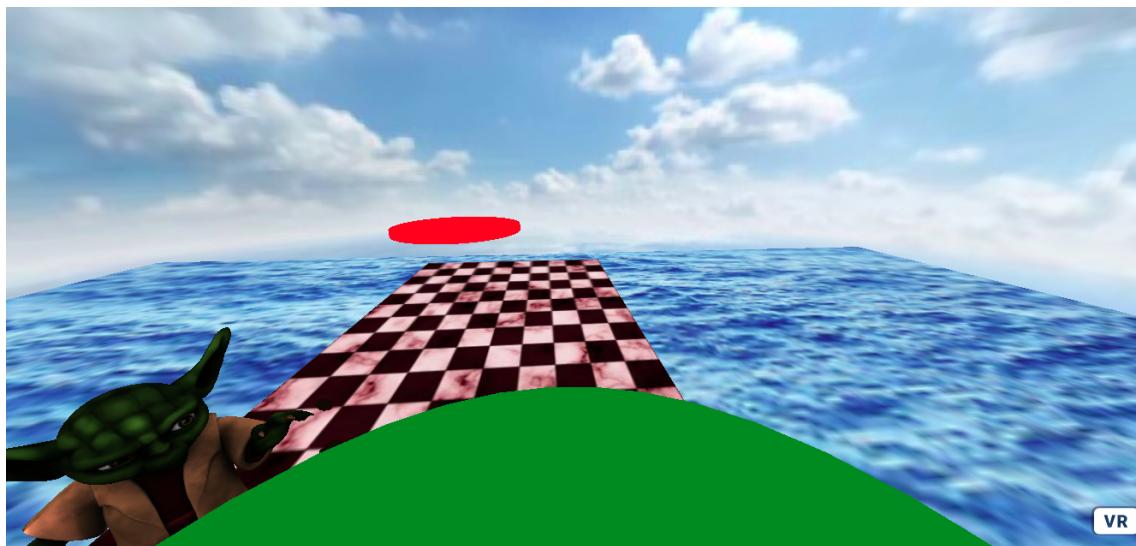


Figure 3.3: Plataformas para subir de nivel

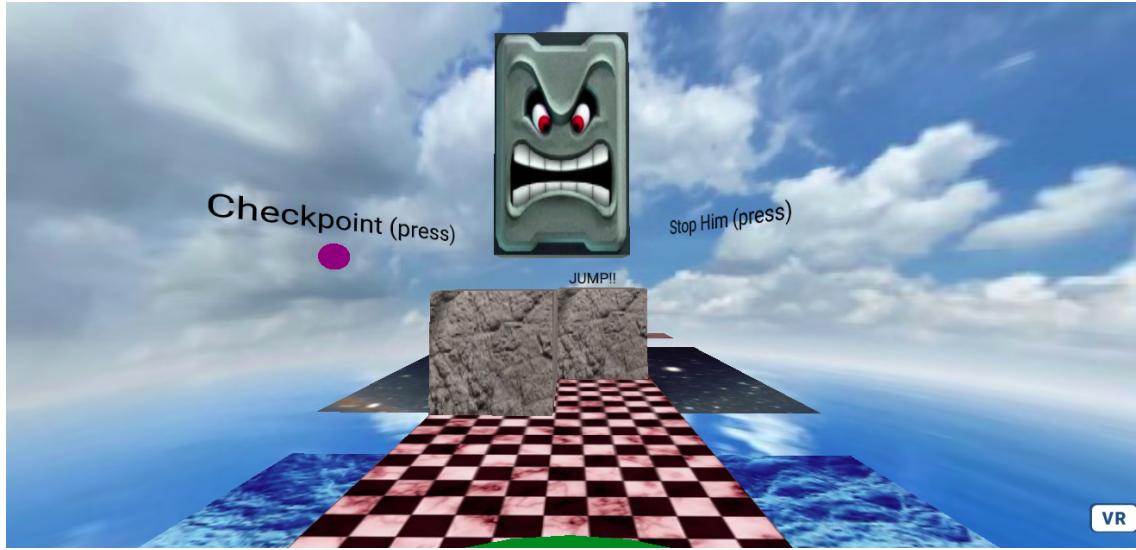


Figure 3.4: Piedras que bloquean el camino y bloque gigante

3. Ya subidos en la plataforma giraremos 180 grados para encontrarnos un nuevo obstáculo en movimiento, la piedra gigante (Figura 3.4). Pero para llegar a ella primero debemos empujar las dos piedras que nos cortan el camino, haciendo así que caigan al vacío.
4. Para conseguir cruzar debemos parar el bloque gigante a nuestros pies, lo que nos dejará ver el siguiente y último reto, las baldosas móviles (Figura 3.5). Cruzando con cuidado llegaremos a la última plataforma gigante.
5. En el centro de esta plataforma nos espera un botón que invoca un evento de felicitación junto con las palabras de fin del juego.

Son muchos obstáculos pero esto permite que utilice el mayor número de entidades y cree los eventos necesarios. Por último, si caes en cualquier momento, volverás al inicio para además añadirle un nivel de dificultad, al fin y al cabo, es un juego.

Para crear este pequeño juego se comienza con un archivo HTML en el que se importan los scripts necesarios de Aframe para poder comenzar a crear entidades, estos son algunos de ellos:

```
<script src="../../assets/js/aframe.min.js"></script>
<script src="../../assets/js/aframe-physics-system.min.js"></script>
```

Gracias a esto podemos empezar a crear nuestras entidades. Las entidades son los distintos objetos que podemos crear como pueden ser un plano, un cilindro o un cubo, entre otros. Para este caso, el pasillo será un plano, las puertas serán cubos y las plataformas cilindros. Todas las entidades tienen propiedades que pueden ser modificadas dinámicamente, mediante eventos

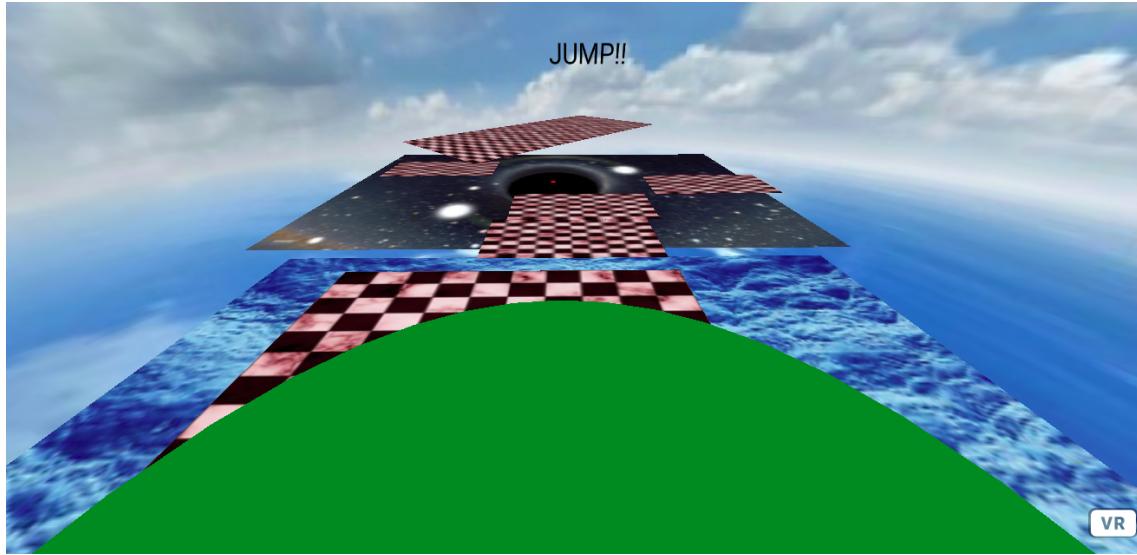


Figure 3.5: Obstáculo de baldosas con movimiento

por ejemplo. Modificando las propiedades rotación, ancho y largo del plano, podemos crear nuestro pasillo:

```
<a-plane position="0 0 0" rotation="-90 0 0" width="3.25" height="20"
src="#floor" repeat="1 5" static-body></a-plane>
```

En el apartado ”src” podemos asignar una textura a nuestras entidades, mientras que con ”repeat” podemos hacer que se repita la textura para que no pierda calidad, o crear incluso un pasillo de baldosas como el visto anteriormente. Siguiendo estas reglas, podemos crear todas las entidades necesarias para nuestra escena. Ahora sólo queda añadir las dinámicas y eventos a estas, al igual que animaciones.

A-Frame no nos aporta figuras más complejas que cubos, cilindros, etc. Por eso, podemos crearlos nosotros con distintas aplicaciones de edición 3D en formato gltf³ e importarlas a nuestra escena. Podemos usar distintas páginas web con diseños completamente gratuitos de modelos 3D para nuestras aplicaciones, una de ellas es Sketchfab⁴. Por ejemplo, los diseños del obstáculo de las hachas los encontré en esta página.

Para hablar de los eventos, tomaré como ejemplo el botón para abrir las puertas. El botón es una caja que tiene como propiedad el componente ”clickable”. Esto permite acceder a el gracias a JavaScript y que cada vez que se posicione el ratón encima de la entidad y se haga

³<https://es.wikipedia.org/wiki/GlTF>

⁴<https://sketchfab.com/feed>

click en él, se dispare un nuevo evento. Al capturar ese evento, podemos modificar propiedades de una entidad en tiempo real. El siguiente código muestra un ejemplo de ello:

```
buttonEl.addEventListener('mouseenter', function() {
  buttonEl.setAttribute('color', "gray");
  buttonEl.setAttribute('scale', "0.22 0.22 0.22");
});
```

Así podemos hacer que al pulsar el timbre (que es un cubo), la rotación de las puertas cambien 90 grados (lo que hace que la puerta se abra). Los mismos principios se siguen para hacer que el bloque gigante se pare para que podamos pasar o el botón que activa la escena final del juego.

El movimiento de las hachas, las plataformas que nos suben de piso o las baldosas que nos impiden cruzar es a lo que se llama animación. Una animación nos permite establecer un patrón de movimiento en una entidad (las entidades son estáticas). La animación es otra propiedad de la entidad como puede ser el ancho, se declara como sigue:

```
<a-cylinder class="platform" position="1.5 22 -12"
animation="property: position; dur: 10000; to: 1.5 -1 -12;
dir: alternate; loop: true" static-body></a-cylinder>
```

En este trozo de código declaramos la duración de la animación, la posición final donde acaba, si es infinita y si se alterna o no. Este es el código que usan las plataformas para subir, empiezan en una posición, acaban en otra y gracias al atributo "alternate", cuando llegan a la posición final vuelven a la inicial como si de un ascensor se tratara.

Para completar este apartado, resta explicar los usos de los componentes. Un componente es un comportamiento que se asocia a una entidad, por ejemplo el antes mencionado "clickable". Para conseguir la función de reaparición del jugador cada vez que caía del escenario, creé el siguiente componente, en el que se detallan estas condiciones:

```
AFRAME.registerComponent('restart', {
// ...
schema: {
  checkpoint: {type: 'string', default: '0 0 9'},
},
tick: function () {
  var audio = document.querySelector('#deathsound');
  var data = this.data;
  if(this.el.getAttribute('position').y < -6){
    this.el.setAttribute('position', data.checkpoint);
```

```

        audio.play();
    }
}
));

```

En este componente aparece la función "tick", la cual se ejecuta cada milisegundo, comprobando si la posición "y" del jugador es menor (para así saber si ha caído) y haciendo que reaparezca en el inicio. Los componentes son herramientas muy potentes, ya que nos permiten crear funciones complejas como esta y simplifica el código. Para asociar un componente a una entidad debemos incluirlo como, ya hemos visto anteriormente, un atributo cualquiera.

3.1.3 Resultado

Como iniciación a las nuevas tecnologías, el resultado fue notablemente bueno. Pude practicar con muchas de las posibilidades que me ofrecía Aframe mientras aprendía JavaScript. Además, tuve mi primer contacto con las gafas de realidad virtual en las que pude depurar el final de mi juego (Figura 3.6). El minijuego se encuentra alojado en la página de GitHub del proyecto y puede ser jugado⁵. El código JavaScript⁶ se encuentra a su vez alojado en la carpeta correspondiente a este "Sprint" junto a su HTML⁷. Para terminar, se mantuvo una reunión en la que se comentó lo aprendido gracias a este "Sprint" y dió comienzo al siguiente apartado.

3.2 Sprint 1

Debido al avance y conocimientos obtenidos gracias a la anterior iteración, en esta se expusieron las bases del proyecto. Como consecuencia a todos los objetivos que se verán a continuación, se llevaron a cabo un par más de reuniones para conseguir un asentamiento adecuado del trabajo.

3.2.1 Objetivo

El objetivo de esta fase es empezar a construir un editor simple que genere entidades a elección. Se establecerá un panel de selección que nos deje elegir entre una mesa de entidades u otro

⁵<https://javierbravodonaire.github.io/A-frame/Sprint%201/my-examples/minigame.html>

⁶<https://github.com/JavierBravoDonaire/A-frame/blob/master/Sprint%201/my-examples/minijuego.js>

⁷<https://github.com/JavierBravoDonaire/A-frame/blob/master/Sprint%201/my-examples/minigame.html>



Figure 3.6: Escena victoria

panel de entidades. Al lado de ese panel tendremos un podio, en el cual aparecerán entidades que elijamos en el panel o en la mesa infinitamente. Para ello, se hará uso de distintos eventos para generarlas y se definirán los elementos básicos de la escena en el archivo HTML.

3.2.2 Desarrollo

Al tratarse de los cimientos del programa, se trabajarán las fases una encima de otra (esto significa, en los mismo archivos). Con el propósito de explicar todo el proceso ocurrido en esta iteración y siguiendo el método Scrum, podemos dividir los Sprints en distintas tareas donde cada una trabaja con el proyecto que finaliza la anterior. A esto se le llama bolsa de tareas y en este caso, se dividirá en cuatro.

Tarea 1:

En esta primera tarea se trabajó en una escena que ofreciese un panel con distintos botones, que permitiese la elección de diferentes figuras. Para empezar, se debe crear un entorno en el que el usuario se sienta cómodo, y para ello, se crea un plano en horizontal (el cual estará dotado de una rotación de 90 grados) que simule el suelo y se dota a la escena del elemento "sky" (con una textura que se asemeje al cielo) para que parezca que nos encontramos en un plano real. Una vez tenemos el suelo, ya podemos empezar a situar nuestros elementos.

El panel será entonces un plano, de un tamaño razonable, en el cual se colocarán los botones.

Para crear los botones, usaremos un cubo al que le modificaremos su propiedad "depth" (profundidad), para dar la sensación de apretar un botón (Figura 3.7). Cada botón tendrá un texto donde se indica que objeto crea. Este texto se añade creando una entidad "a-text" y enlazándola con el botón.

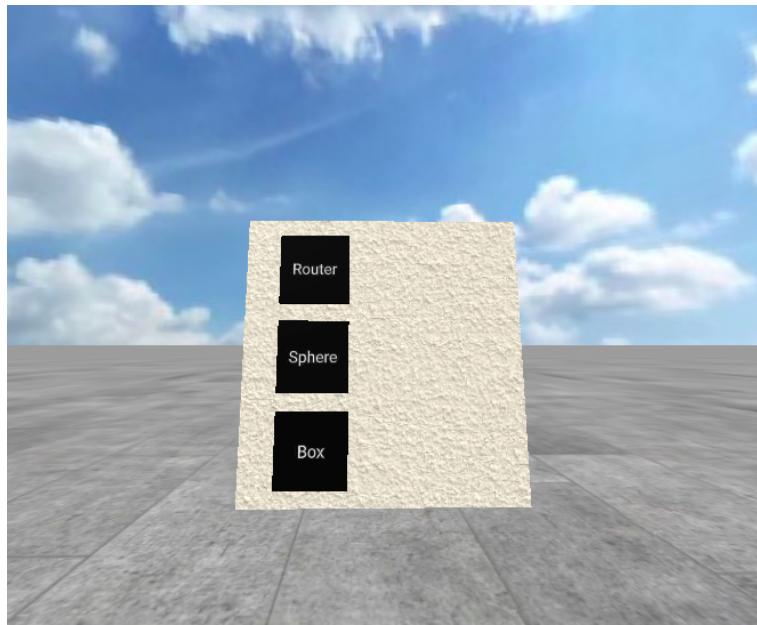


Figure 3.7: Panel con botones para generar entidades

Para crear las entidades necesitamos hacer uso de eventos mediante JavaScript. Le asignaremos a cada botón un "id" distinto para poder acceder a todos ellos como un elemento cualquiera del DOM, y le añadiremos un evento. Este evento se encargará de detectar si en algún momento se ha pulsado sobre un botón, y cuando esto suceda se creará la entidad a la que se refiere el mismo. Para conseguir capturar este evento, debemos dotar a los botones con el atributo "clickable" definido en el script de A-Frame, el cual le da la cualidad de ser pulsado. Si pulsamos todos los botones nos encontraremos una escena como la que se puede observar en la Figura 3.8.

Como queremos mantener el código lo más limpio posible y evitar la repetición de líneas, se crea un apartado en el HTML llamado "assets" donde podemos crear los llamados "mixin". Un mixin es una especie de entidad genérica en la que declaras su material, tamaño, etc, y a partir de ella puedes crear clones. Un ejemplo para comprobar su utilidad podría ser el siguiente: imaginemos que queremos crear tres cubos pulsando un botón, entonces desde el JavaScript debemos manejar el evento y crear desde el mismo la entidad, sentencia a sentencia, declarando

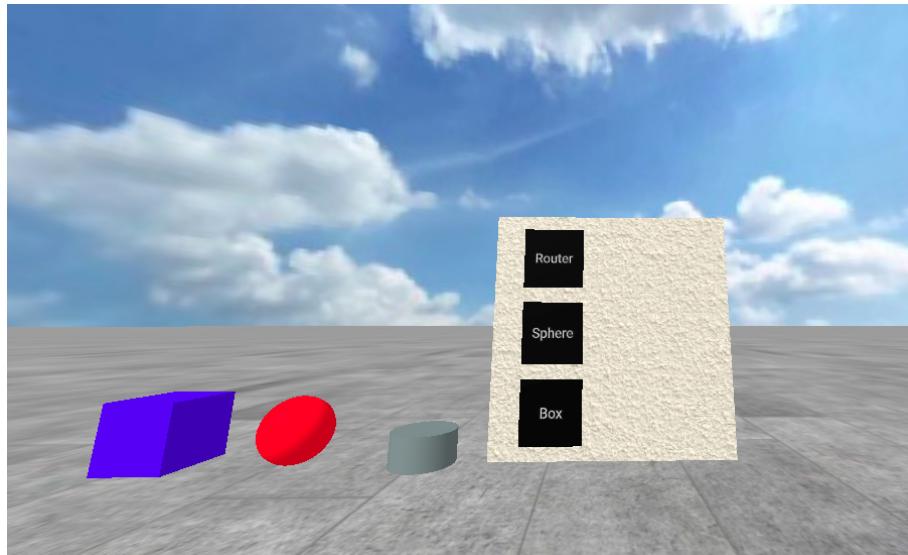


Figure 3.8: Entidades generadas con el panel

el color, el tamaño y la posición. Con el mixin ya hemos declarado todos estos atributos, por lo que solo necesitaríamos declarar que es un "mixin-cubo" y nos ahorraríamos todas las sentencias de sus propiedades en el JavaScript. Un mixin para un cubo es como el siguiente:

```
<a-mixin id="box" hoverable grabbable stretchable draggable
geometry="primitive: box; width: 0.5; height: 0.5; depth: 0.5"
droppable</a-mixin>
```

Tarea 2:

Fue entonces cuando surgió la idea de que sería más adecuado tener dos modos de creación de objetos. Una de ellas sería el panel que ya tenemos, y la otra sería una mesa de entidades, donde las encontrases todas y pudieses agarrarlas, dotando al programa de una visión más intuitiva.

Para ello, me apoyé nuevamente en los modelos 3D ofrecidos por la página Sketchfab⁸. Ya que A-Frame no ofrece objetos tan complejos como puede ser una mesa, buscar un diseño agradable y gratuito era la mejor opción. Para agregar a la escena estos objetos, debemos declararlos en el mismo apartado que declaramos los mixin, en "assets". Debemos prestar atención a la escala de los modelos 3D, ya que no están diseñados para añadirlos a las escenas directamente y lo más probable es que sean gigantescos.

⁸<https://sketchfab.com>

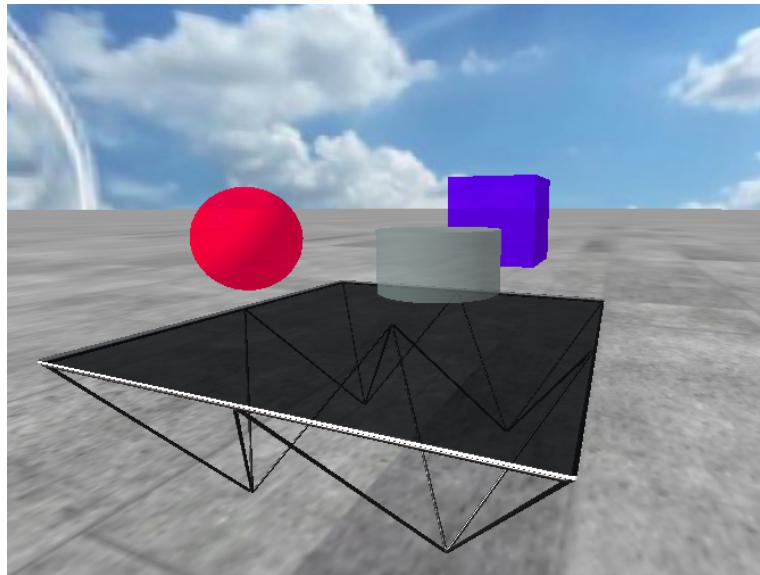


Figure 3.9: Mesa con entidades holográficas

Una vez tenemos la mesa en nuestra escena, es hora de colocar las entidades encima de ella para poder elegir. Con el objetivo de dotar a la escena de más movilidad, se añadieron animaciones y se modificó la opacidad y transparencia del color para dar la sensación de unos hologramas que flotan por encima de la mesa (Figura 3.9).

Por último, se debe añadir la funcionalidad de elección para crear un clon de la entidad con la que podamos trabajar (igual que creábamos los objetos al pulsar el botón). En este caso, era el propio holograma el cual, al pulsar sobre él, nos creaba una copia. La copia se generaba delante del usuario y para lograr esto se tomaba como punto de referencia la posición de la cámara, consiguiendo así que la entidad se generase en frente, dando igual desde qué posición la cogieses.

Tarea 3:

Como ya tenemos nuestros modos de selección, ahora debemos añadir una funcionalidad que implique cambios en las entidades. La idea principal es crear una paleta de colores y texturas que nos permita en cualquier momento cambiar el estilo de esta. Esta aparecerá cuando pulsemos sobre la entidad, por lo que debemos tener cuidado con el evento que lance esta acción.

La paleta nos aporta un buen ejemplo de lo que son los nodos. La entidad tendrá un nodo hijo que será el plano que representa la paleta, que a su vez tendrá x hijos que serán los botones



Figure 3.10: Paleta de colores perteneciente a una entidad

que representan los diferentes colores y texturas (Figura 3.10). Es aquí, cuando me enfrenté a un gran problema, los eventos escalan hacia las entidades padre, por lo que se debe detener el evento para que no afecte y genere comportamientos inadecuados. En esta ocasión, el evento de cambiar el color, afectaba tanto a la entidad como al plano que representa la paleta, por lo que se debía controlar.

La paleta se compone así de un total de ocho opciones divididas en dos filas. La primera fila se corresponde con cuatro colores distintos, y la segunda contiene tres texturas diferentes y un último botón que devuelve a la entidad el color original con el que fue creada.

Tarea 4:

Terminadas todas las funcionalidades propuestas, solo restaba unificar los dos modos de selección. Para ello, se optó por un panel con dos opciones a elegir: la mesa o el panel de entidades. Esta tarea es fácil, ya que hemos hecho un panel con botones anteriormente, solo debemos controlar los eventos para generar la mesa (ya que ahora no se encuentra nada más iniciar la escena) y el otro panel, pero esto nos conduce al siguiente dilema, los dos modos tienen una forma diferente de generar las entidades.

Esto da lugar a la idea de crear un podio, en el cual, aparecerán las entidades según las seleccionemos tanto en la mesa como en el panel (con el panel aparecían en un lugar predeterminado declarado en el código para cada entidad, y con la mesa aparecían delante de nosotros). Se crearán así todas las entidades que necesitemos y por último, las agregaremos el atributo "grabable", el cual dotará a estas de movilidad para poder arrastrarlas por toda la escena. Quedará entonces una escena como la que podemos ver en la Figura 3.11, donde observamos el podio

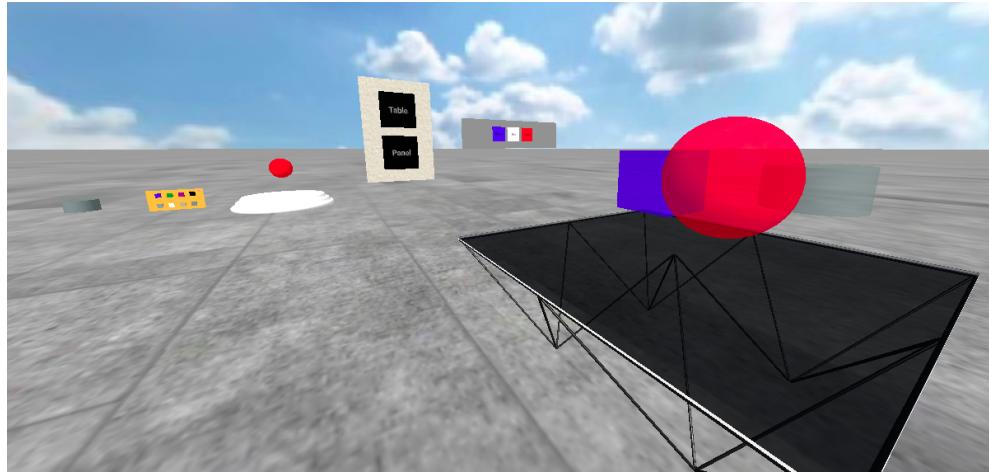


Figure 3.11: Versión funcional conseguida al final del Sprint 1

con una entidad sobre él, una entidad aparte con su paleta de edición, la mesa de entidades, el panel donde elegimos el modo de selección y el panel de elección de entidad.

3.2.3 Resultado

Gracias a esta iteración hemos asentado las bases y varias funcionalidades que darán paso a las siguientes etapas. Todos los eventos y funciones empleados se recogen en el código JavaScript⁹ alojado en el repositorio de GitHub, así como el código HTML¹⁰ donde se definen los elementos iniciales de la escena. Como en el anterior apartado de resultado y en los siguientes, se adjunta la dirección para probar esta versión en el navegador¹¹.

3.3 Sprint 2

Una vez establecidas las bases de la escena, se deben implementar nuevas funcionalidades que aporten dinamismo y utilidad. La reunión que finalizó la anterior iteración, fue la misma que generó los objetivos de esta. Como ya me había adecuado al estilo de A-Frame y al lenguaje JavaScript, era hora de añadir opciones más complejas.

⁹<https://github.com/JavierBravoDonaire/A-frame/blob/master/Sprint%202/improve.js>

¹⁰<https://github.com/JavierBravoDonaire/A-frame/blob/master/Sprint%202/improve.html>

¹¹<https://javierbravodonaire.github.io/A-frame/Sprint%202/improve.html>

3.3.1 Objetivo

Hasta ahora, podemos generar entidades infinitamente encima del podio y cada una de ellas tiene su propia paleta de colores, pero esta opción no es la más intuitiva. Por ello, la mejor opción es añadir en el propio panel, donde creamos las entidades, otra paleta de colores y texturas. Pero existe un problema, la entidad no se crea hasta que no la eliges, y una vez seleccionada ya se ha creado con su color por defecto, por eso se creará la principal funcionalidad propuesta en esta iteración, el podio con entidad dinámica.

El podio con entidad dinámica trata del mismo podio, pero en este caso ya existe una figura flotando por encima de este, y pulsando sobre los distintos botones de color, textura o incluso entidad, es la propia figura la que cambia. Entonces, existirá una sola figura flotante que cambiará según nuestras indicaciones, y cuando la arrastremos fuera del podio se creará una copia para poder seguir editando entidades infinitamente.

Por último, se añadirá una función para, al igual que elegimos el estilo de las figuras, podemos elegir su tamaño.

3.3.2 Desarrollo

Como el fundamento de esta iteración es la entidad cambiante que se encuentra encima del podio, lo primero es crearla. Para ello, debemos entender como funcionan las relaciones padre-hijo. El podio es una entidad hija de la escena y a su vez, la entidad cambiante será una hija del podio. Con esto, conseguiremos realizar correctamente la generación de una nueva entidad cuando saquemos la que se encuentra en el podio. A continuación, se explica paso a paso como se realiza esta funcionalidad, para que quede de forma clara:

1. Como en la iteración anterior ya importamos el modelo del podio, no necesitamos crearlo de nuevo. Se crea un entidad aleatoria (en mi caso elegí un cubo), que será la que aparezca al inicio de la escena y sobre la que editaremos.
2. Debemos crear esa entidad como hija del podio, de forma que la posición, el tamaño y la función de arrastras se vean afectadas por este. Una entidad que sea hija de otra dependerá en sus atributos de su nodo padre, entonces, si queremos que la entidad este flotando, solo se necesitará modificar la posición en el eje "y" (el padre es la posición de referencia).

3. Siguiendo esos pasos, ya tenemos el podio y la entidad colocada perfectamente. Como vamos a necesitar acceder a los atributos de la entidad para realizarle cambios, debemos añadirle un "id" único, el cual nos permitirá hacerle referencia desde nuestro código JavaScript (donde se encuentra toda la funcionalidad).
4. Añadimos al panel distintos botones (a los que ya había), con texturas y colores diferentes. Ahora, debemos modificar el código anterior, ya que no vamos a crear entidades directamente, vamos a modificar una existente. Se divide así el código en dos partes: la referente a los colores y texturas, y la referente a las entidades.
5. En el caso de los botones de selección de entidad, deben modificar la estructura de la figura, por eso lo más sencillo es realizar lo visto en la anterior iteración, crear un mixin por entidad. Así, cada vez que pulsemos un botón distinto, solo será necesario acceder a la entidad (gracias al "id") y modificar su atributo mixin.
6. El caso para colores y texturas es más sencillo. No necesitamos ningún mixin, solo accedemos a su atributo color o source, y lo modificamos.
7. Por último, resta crear una nueva entidad cuando saquemos del podio la editada. Se podría repetir el primer paso, crear una entidad aleatoria y vuelta a lo mismo, pero si queremos crear dos entidades iguales, ¿tenemos que editarla dos veces?. Es aquí cuando surge la función "cloneEntity", la cual una vez hemos retirado la entidad del podio, realiza una copia idéntica a la ya creada. La entidad separada, pasará entonces a ser una hija de la escena, y no del podio, y delegará su "id" al clon (el cual será el nuevo hijo del podio), generando un bucle infinito que permite extraer todas las figuras necesarias.

De todos estos pasos, podemos generar una nueva funcionalidad de los botones de color y textura. Como el color y la textura se corresponden con distintos atributos de una entidad, podemos combinarlos. Esto es, si pulsamos sobre una textura y luego un color, cambiará la tonalidad de esa textura ofreciendo más combinaciones de estilo.

Al igual que podemos colorear las figuras, quizás necesitemos eliminar su color. La mejor idea es usar el color blanco debido a que si estableces una textura, no vas a querer pulsar el blanco encima de ella y además no surtiría efecto. Esto hace que el color blanco del panel se convierta en el botón de reinicio de estilo.

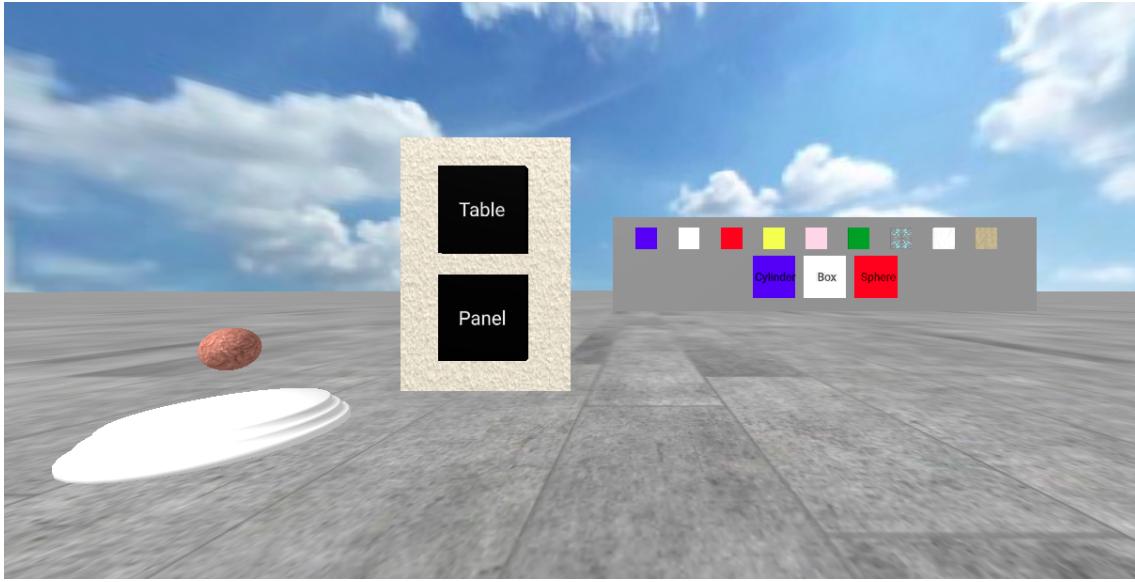


Figure 3.12: Panel con botones para colores y texturas

Podemos observar un ejemplo de estas funcionalidades (podio con entidad variable y textura combinada con color) en la Figura 3.12.

Lo último que quedaba para terminar esta iteración, era el control de tamaño. Para esta modificación, la idea inicial fue implementarlo como el color y texturas, a través de botones, pero además de no ser intuitivo, solo permitiría editar el tamaño en las mismas cantidades que el número de botones establecidos. La idea sucesora, y la finalmente implementada, fue una barra deslizable.

A-Frame no proporciona ninguna entidad que represente una barra deslizante, pero gracias a GitHub podemos encontrar muchos scripts que definen varios componentes que nos interesaría importar directamente a nuestro proyecto, entre ellos la barra deslizable. En este caso, la biblioteca aportada por el usuario **rdub80**¹², proporcionaba diversos elementos como barras de progreso, etiquetas y la empleada, barra deslizante.

Este elemento se divide en: la barra y un punto de referencia. La movilidad del elemento ya está creada, pero falta enlazarla con nuestra entidad del podio. La referencia para determinar el tamaño se encuentra en la posición del punto de referencia. Mediante JavaScript, obtenemos la situación del punto de referencia (que por defecto se encuentra en el centro de la barra, para permitirnos elegir aumentar o disminuir) en la escena, y siguiendo un sencillo algoritmo mod-

¹²<https://github.com/rdub80/aframe-gui>

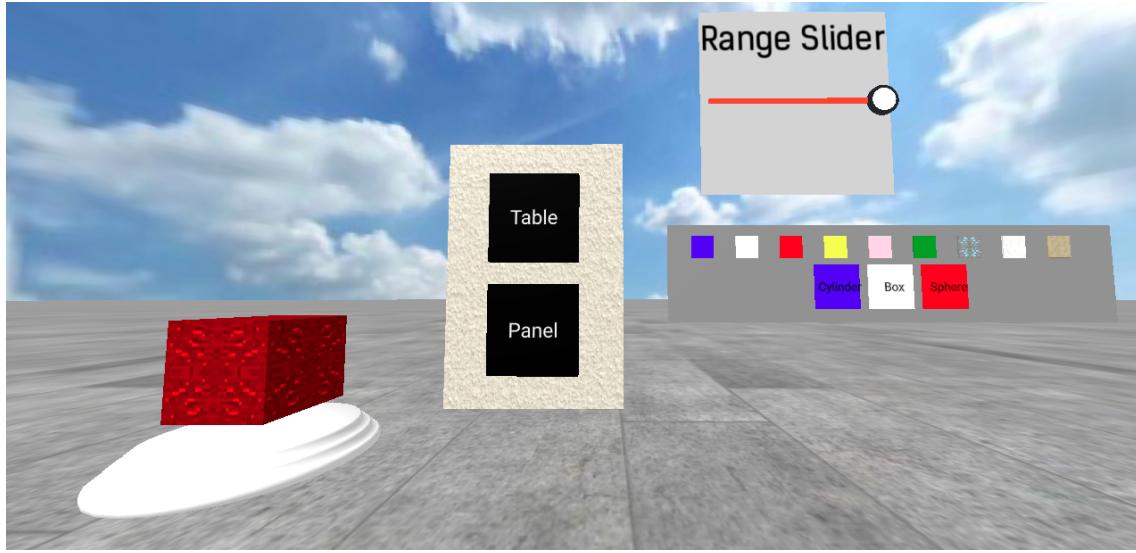


Figure 3.13: Escena con panel de edición y barra deslizable

ificamos el atributo escala de la entidad. Quedaría entonces una escena como la que podemos ver en la Figura 3.13, en la que hemos decidido hacer crecer al máximo un cubo.

3.3.3 Resultado

Hemos conseguido así, una gran funcionalidad para nuestro entorno de realidad virtual. No se encontraron dificultades añadiendo la barra deslizable ni mejorando el panel, pero lo más costoso (y más importante), fue realizar la clonación de entidades y la edición de estas. Como siempre, el código HTML¹³ y JavaScript¹⁴, están almacenados en el repositorio referente al proyecto en GitHub, y se encuentra disponible el enlace para probar la versión funcional de esta iteración¹⁵.

3.4 Sprint 3

Esta iteración se corresponde con la recta final del proyecto, y como todas las anteriores, comienza con una reunión con el tutor del proyecto. En ella se establecieron los objetivos que se exponen en el siguiente apartado y se inició la redacción de este escrito.

¹³<https://github.com/JavierBravoDonaire/A-frame/blob/master/Sprint%203/improve.html>

¹⁴<https://github.com/JavierBravoDonaire/A-frame/blob/master/Sprint%203/improve.js>

¹⁵<https://javierbravodonaire.github.io/A-frame/Sprint%203/improve.html>

3.4.1 Objetivo

Es hora de adaptar el proyecto a su uso mediante unas gafas de realidad virtual, lo que implica dividir este en dos ramas, la versión de escritorio (navegador) y la versión con gafas de realidad virtual. El objetivo es entonces generar una escena funcional para ambos modos. Cada una tendrá su propio modo de uso y componentes. Para finalizar, se agrega una última funcionalidad, que consta de un modo de edición, donde se generan los ejes correspondientes a la escena y a las entidades para permitir una visión más clara de los tamaños y posiciones.

3.4.2 Desarrollo

Por simplicidad, se empezará añadiendo las funcionalidades nuevas a la versión de escritorio, para más tarde, solo adaptar esta a las gafas VR.

Se trabajará entonces en dos archivos:

- desktop.html¹⁶
- glasses.html¹⁷

Versión de Escritorio:

Al dividir el trabajo, en este apartado solo debemos implementar la nueva funcionalidad (ya que hemos estado trabajando siempre en modo escritorio). El objetivo del modo edición consiste en mostrar una cuadrícula en la escena con el eje de esta y mostrar el esqueleto de las entidades con su propio eje.

Todas las entidades poseen un atributo llamado "wireframe" (estructura metálica), que cuando se activa (se añade como una propiedad cualquiera y se le indica el valor "true"), muestra el esqueleto que sigue la figura. Se puede observar en la Figura 3.14 que la estructura sigue la unión de pequeños triángulos, ya que es así como se consigue una renderización de figuras más veloz.

Para dotar de ejes a las figuras, debemos agregar como nodos hijos tres entidades "line" (línea). Al ser hijas de la entidad, se moverán con ella y su posición central (0,0,0) será igual a la de la entidad y no a la de la escena, lo que nos permite generar los ejes modificando solo

¹⁶<https://github.com/JavierBravoDonaire/A-frame/blob/master/Sprint%204/desktop.html>

¹⁷<https://github.com/JavierBravoDonaire/A-frame/blob/master/Sprint%204/glasses.html>

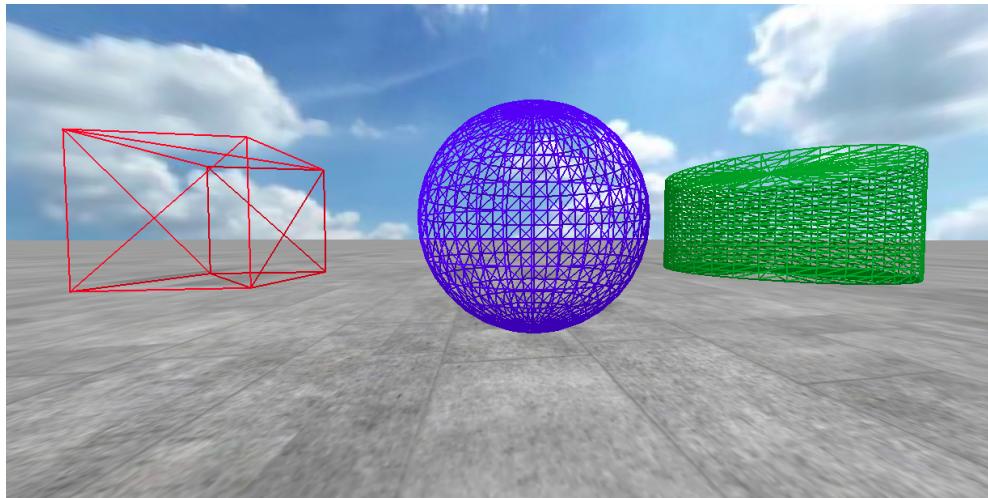


Figure 3.14: Wireframe de distintas entidades

un valor de posición de cada línea (eje 'x', eje 'y' y eje 'z'). La entidad linea contiene las propiedades inicio y fin, si queremos representar el eje 'x' entonces debemos crear una línea como sigue:

```
<a-entity line="start: 0, 0, 0; end: 2 0 0; color: black"></a-entity>
```

Una vez tenemos las figuras con su esqueleto y sus ejes, resta generar el eje de la escena, que es el que marca los tamaños y posiciones. Para ello, la idea es generar en el plano 'xz' una cuadrícula de tamaño 1, que nos permita observar la profundidad y desplazamiento, y un plano 'y' que tratará de un eje con marcas cada distancia 1 (estas marcas serán líneas más pequeñas que simulan un eje como el que podemos ver en una gráfica).

Para esta tarea, se crea una función que genera líneas, en los distintos planos hasta una distancia que se le pasa como parámetro, por si queremos hacerlo más grande o más pequeño, creando así la cuadrícula y el eje 'y'. Además, para que este modo se diferencie y deje apreciar bien los ejes, se cambia la textura del suelo por un color verde que resalte el negro de los ejes.

Para alternar entre la escena en su estado normal y la de su modo edición, se establecen dos teclas del teclado:

- Tecla G: entra en modo edición. Cambia la textura del suelo por el verde y añade la cuadrícula, los ejes y el esqueleto de las figuras (Figura 3.15).
- Tecla H: sale del modo edición. Las figuras vuelven a su estructura inicial y pierden los ejes. El suelo recupera su textura y desaparece la cuadrícula.

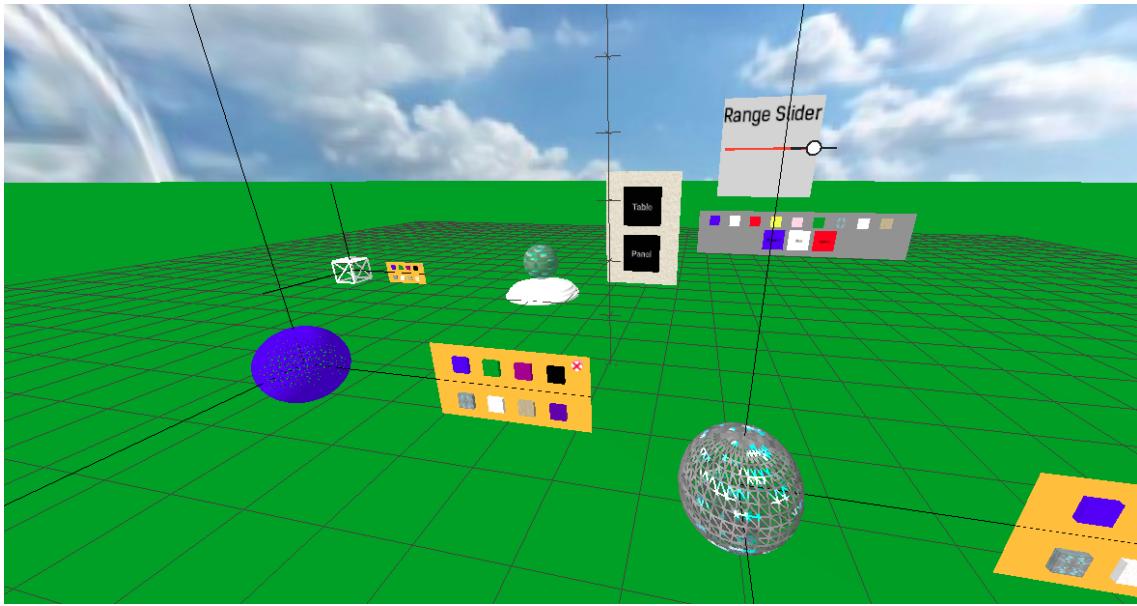


Figure 3.15: Escena en modo edición

Versión con Gafas VR:

Para empezar, se debe modificar el elemento que simula al usuario, ya que por el momento solo se movía con comandos del teclado. Para esto, se le añaden atributos que permita movernos con los mandos de las gafas y, dentro de este, se añaden como dos hijos la mano derecha y la mano izquierda.

Como las dos manos son iguales pero cambiando un atributo, es más factible crear un mixin que las represente, en el que añadimos los atributos generales. Entre estas propiedades nos encontramos "raycaster" y controles varios (vive-controls, oculus-touch-controls, windows-motion-controls, gearvr-controls, daydream-controls y oculus-go-controls) para los diversos dispositivos en el mercado. "Raycaster" es la propiedad más importante, ya que representa un rayo que interseca con los objetos permitiéndonos capturar los eventos. Antes era el ratón el que hacía click y con el que apuntábamos a los botones, de eso se encarga "raycaster" cuando usamos los mandos y apuntamos con ellos. Dentro de la propiedad "raycaster" debemos indicar las clases de todos los elementos HTML con los que podemos interaccionar (los botones por ejemplo).

Una vez creado los mixin, solo necesitamos crear las dos manos como hijas del usuario en una posición que las simulen correctamente. En este caso, decidí emplear la función láser de los mandos, lo que no representa unas manos en la escena, sino que vemos los mandos con un

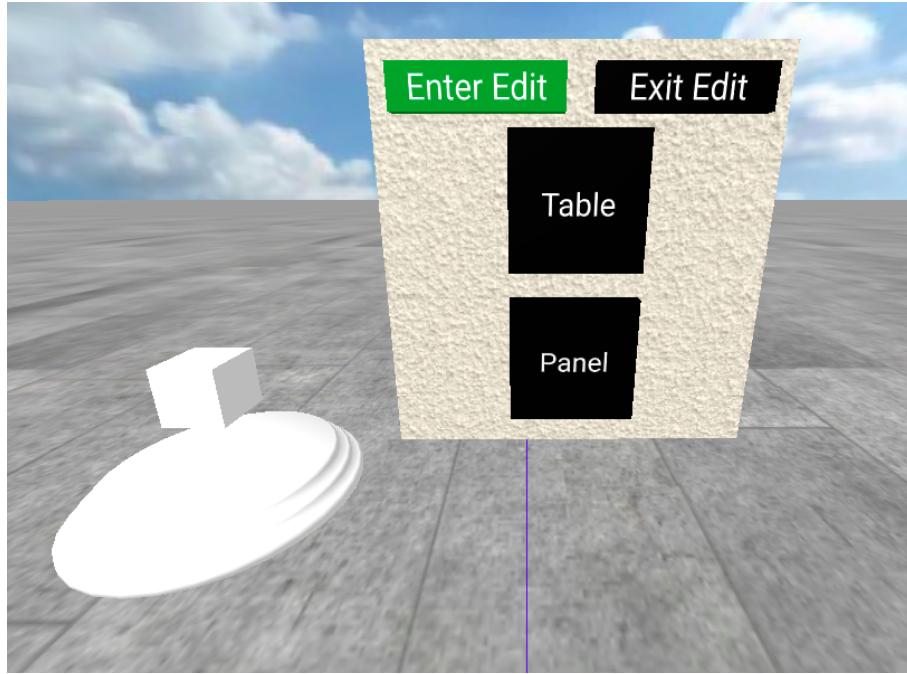


Figure 3.16: Panel con botones de modo edición versión VR

rayo láser con el que podemos apuntar y seleccionar todos los elementos. Esta decisión se tomó para garantizar una movilidad mayor, fusionando los controles del usuario con los "joysticks" de los mandos y el apuntado con láser.

Debido a estas modificaciones, se tuvieron que rediseñar dos funcionalidades para su uso con el láser:

- La barra deslizante se elimina para añadir un gesto que hace aumentar y disminuir a la entidad, este es, apuntar con los dos láseres a la figura y hacer el gesto de estirar o encoger.
- Como el uso del teclado mientras usamos las gafas y los mandos no es muy intuitivo, se añaden en el panel inicial dos botones para entrar y salir del modo edición, Figura 3.16.

Con todos estos cambios, se logra adaptar la versión de escritorio a cualquier dispositivo declarado anteriormente.

3.4.3 Resultado

Se consigue así la versión final de este proyecto, logrando todas las funcionalidades propuestas con éxito. El código HTML de las dos versiones se adjunto anteriormente, pero el código

JavaScript¹⁸ se adjunta en este apartado. Tanto la versión de escritorio como la de las gafas, usan el mismo JavaScript con el objetivo de lograr una unificación de ambas, en caso de que sea necesario. Por último, la versión final del proyecto se encuentra alojada en el repositorio de GitHub para su libre uso, en versión de escritorio¹⁹ y versión VR²⁰.

¹⁸<https://github.com/JavierBravoDonaire/A-frame/blob/master/Sprint%204/improve.js>

¹⁹<https://javierbravodonaire.github.io/A-frame/Sprint%204/desktop.html>

²⁰<https://javierbravodonaire.github.io/A-frame/Sprint%204/glasses.html>

Capítulo 4

Resultado Final

En este capítulo se incluyen el manual de usuario y la arquitectura resultante del proyecto. En el manual se explica el funcionamiento desde un punto de vista sencillo para usuarios novatos, mientras que en la arquitectura resultante se exponen los componentes y estructuración del código.

4.1 Manual de Usuario

Existen dos modos de empleo de la interfaz presente:

- **Modo escritorio.** Para mover la cámara debemos pulsar con el botón y arrastrar hacia la dirección deseada. Para moverse por la escena se pueden usar tanto las flechas de dirección como las teclas "WASD". Para interactuar con los distintos elementos de la escena se usa el ratón, haciendo click sobre estos.
- **Gafas VR.** Con los mandos de las gafas nos movemos, y el movimiento viene marcado por la dirección en la que miramos. Existe la posibilidad de desplazarnos en la escena moviéndonos nosotros. Aparecerán dos mandos en la escena con un láser que funcionarán como el ratón e interactuaremos con el apuntando a los elementos (Figura 4.1).

La escena comienza con un podio y un panel en el que podemos elegir el método de creación de entidades: mesa o panel, como se puede observar en la Figura 4.2. Son dos botones, por lo que, siguiendo las instrucciones anteriores, interactuaremos con ellos. Pulsando sobre las entidades de la mesa o los botones del panel se editarán la entidad que aparece en el podio.

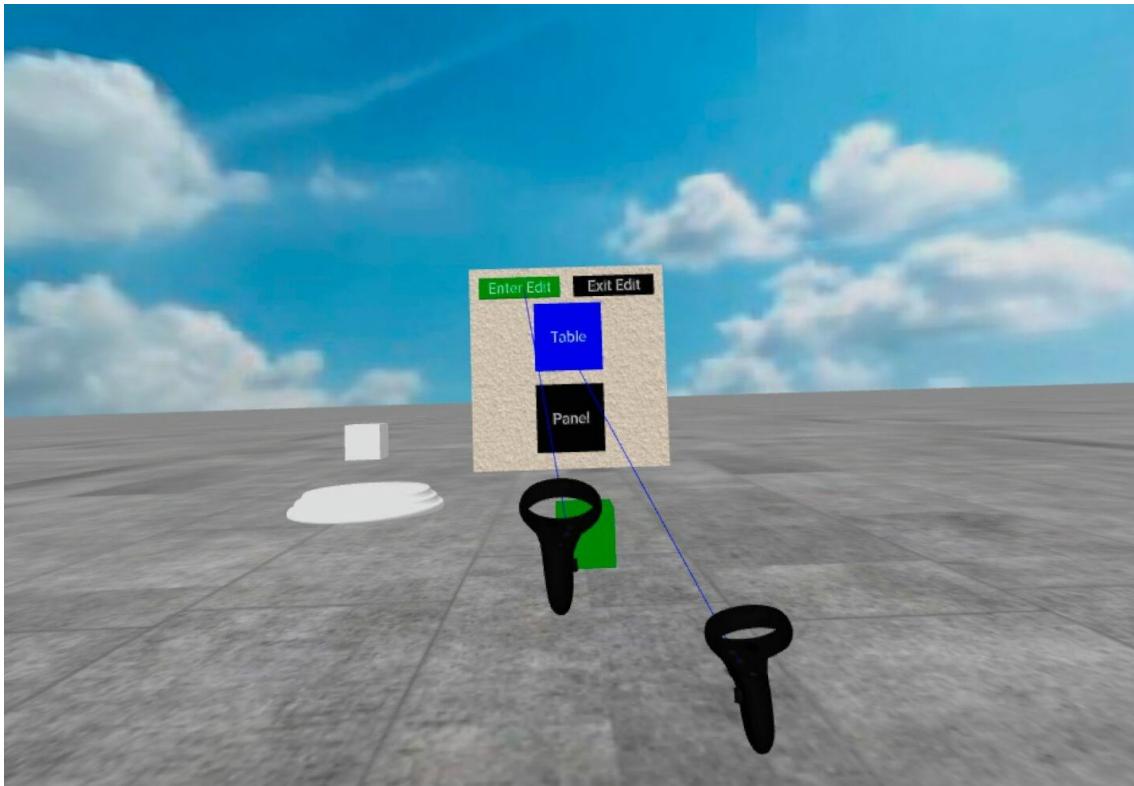


Figure 4.1: Escena inicial versión con gafas de realidad virtual

Además, se pueden editar la textura, color y tamaño de la figura mediante los botones y la barra deslizante. Se pueden combinar colores y texturas, esto es, eligiendo una textura y después un color para llenar esta del color indicado. El botón que representa el color blanco servirá para reiniciar la figura.

Siguiendo todas las indicaciones anteriores, ya se puede editar la figura a tu gusto. Una vez terminada la edición, podemos arrastrar la figura fuera del podio mediante el ratón (o apuntando con el láser y arrastrando), para incluirla en nuestra escena. Al arrastrala, se creará en el podio una figura totalmente idéntica por si queremos crear más de una con el mismo diseño (evitando así la pérdida de tiempo).

Al sacar una figura del podio, esta obtendrá una paleta más pequeña que se moverá con ella permitiendo editar su textura y color después de haber sido creada. El último de los botones de esta paleta, será el color con el que ha sido creada por si queremos volver a este.

Atajos mediante teclas:

Adicionalmente, existen distintas teclas que activan funciones en la escena:

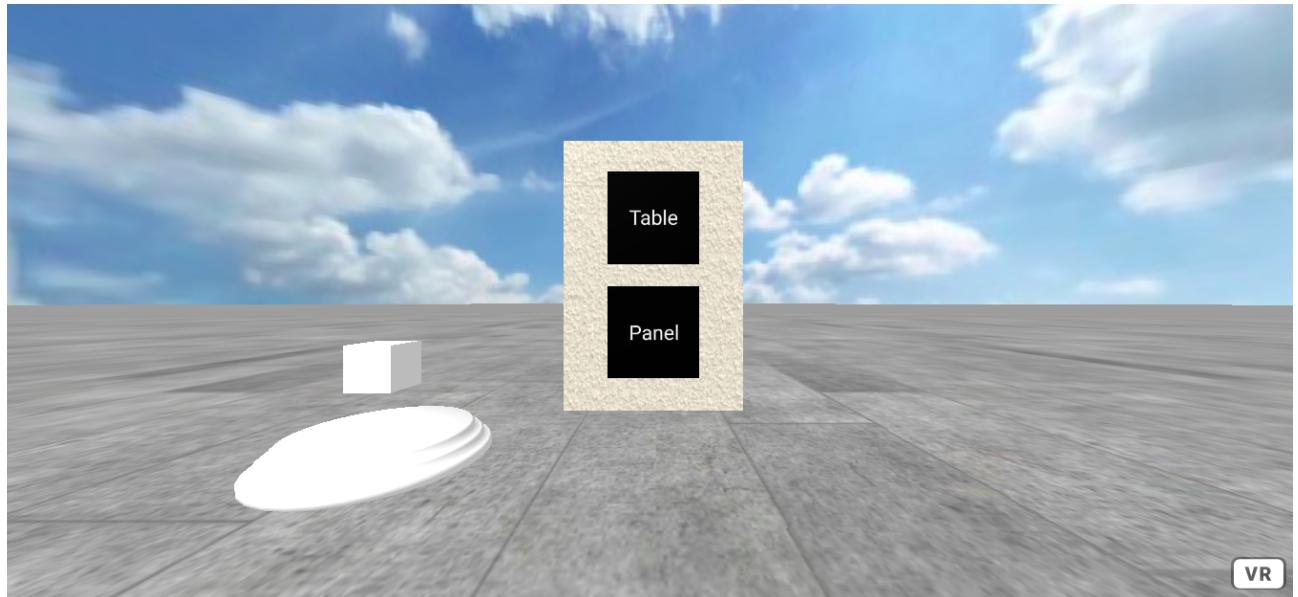


Figure 4.2: Escena inicial versión de Escritorio

- Mediante la tecla '**F**' se nos permite entrar tanto en el modo pantalla completa, como en el modo VR cuando estemos usando nuestro dispositivo VR. Para salir de este modo, está habilitada la tecla '**Esc**'.
- Con la tecla '**G**', accedemos al modo edición. En este modo la escena cambia, el suelo se sustituye por uno verde y aparece una cuadrícula en el suelo y un eje 'y' para obtener una visión más profesional, que permite centrar las figuras y colocarlas. Además, aparecerán ejes para cada figura creada y se mostrará su wireframe (estructura metálica o esqueleto).
- Para volver al modo de creación y eliminar los ejes de la escena se puede utilizar la tecla '**H**'.

Estas dos últimas teclas se sustituyen por dos botones en el panel principal en el modo de gafas VR, para un uso más intuitivo. Además, se añade un componente para poder agrandar las figuras sin hacer uso de la barra deslizante con los mandos mediante un gesto (se apunta con los dos mandos a la figura y se separan). Siguiendo este manual se puede llegar a una versión compleja como la que se muestra en la Figura 4.3.

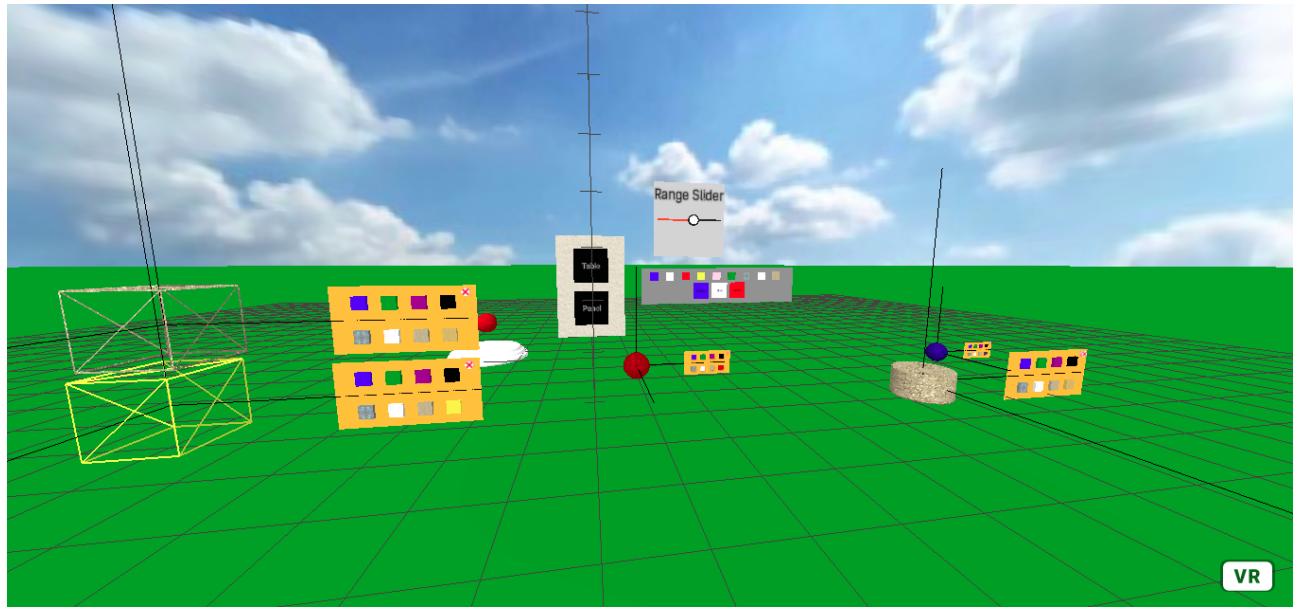


Figure 4.3: Escena compleja con todas las funcionalidades

4.2 Arquitectura resultante

Esta sección seguirá la siguiente estructura: primero se muestran los componentes (tanto creados como los que nos aporta A-Frame) con una pequeña explicación, a esto le seguirá la estructura que sigue la escena inicial implementada y por último, se entrará mas en detalle respecto a los eventos manejados por cada componente.

Componentes:

- *Editable*. Es el encargado de la paleta de generar la paleta de colores de cada entidad.
- *Button*. Maneja los eventos de los botones que se encuentran en la paleta.
- *Create-table*. Crea la mesa de entidades y sus respectivas entidades.
- *Create-panel*. Al igual que se crea la mesa, se crea el panel de colores y la barra deslizante a través de este componente.
- *Edit-mode*. Es el encargado de ayudarnos a acceder al modo edición en la versión de escritorio.
- *Enter-edit-mode*. Nos ayuda a entrar en el modo edición, pero este componente se encuentra en el modo para gafas VR.

- *Exit-edit-mode*. Análogo al anterior, pero su función es salir del modo edición.

Cabe mencionar aquellos componentes usados que nos aporta A-Frame y se han usado en la escena:

- *Clickable*. Este componente nos aporta el evento de click que se activa al pulsar sobre las entidades que lo poseen. Junto con el siguiente componente, nos permite manejar la gran parte de los eventos que suceden en la escena (botones).
- *Grabbable*. Dota a las entidades de la capacidad de agarrarlas y moverlas por toda la escena.
- *Stretchable*. Gracias a este componente podemos realizar el gesto con los mandos para modificar el tamaño de las figuras en nuestra escena.

Una vez presentados los componentes que nos encontraremos, podemos observar que la estructura HTML que se genera cuando se inicia la escena es la siguiente:

```
<a-scene editor>
  <a-assets>...</a-assets>
  <!-- Ground -->
  <a-plane id="ground" rotation="-90 0 0" static-body></a-plane>
  <!-- Panel -->
  <a-plane id="panel" position="0 2 -4" static-body>
    <a-box id="buttonTable" clickable create-table></a-box>
    <a-box id="buttonPanel" clickable create-panel></a-box>
  </a-plane>
  <!-- Podium -->
  <a-entity id="baseEntity" gltf-model="#base" static-body>
    <a-entity id="podiumEntity" mixin="box" editable></a-entity>
  </a-entity>
  <!-- User Camera -->
  <a-entity id="player" movement-controls="fly: true" edit-mode>
    <a-entity camera position="0 1.7 0"></a-entity>
  </a-entity>
  <!-- Sky -->
  <a-sky src="#sky"></a-sky>
</a-scene>
```

En ella se declaran aquellos elementos que necesitamos que aparezcan en el inicio y aquellos que no van a cambiar (Suelo, cielo o panel principal, por ejemplo). El código que hemos visto es

una versión simplificada con los componentes, obviando datos como posiciones y texturas para desarrollar este apartado de una forma más técnica. Dentro de la sección "assets" se encuentran todas las imágenes, modelos 3D y mixin empleados.

Como se observa, varios elementos tienen nodos hijos: panel, podium y user camera. El panel contendrá los botones para generar la mesa y el otro panel, el usuario contiene la cámara y el podio mantiene la entidad. Cuando la entidad abandone el podio, esta pasará a ser un elemento de la escena como podría ser el cielo, y el clon la sustituirá modificando de esta forma el DOM.

Esta escena está compuesta de la manera explicada en los dos párrafos anteriores dando lugar a la Figura 4.2, pero existen muchas combinaciones que darían lugar a la misma, por ejemplo, la escena podría empezar con solo el podio y el panel y la mesa se generaría mediante dos botones distintos.

Con la estructura de la escena vista y una pequeña introducción a los componentes, solo resta explicar la función de cada uno de estos de una forma más detallada:

- *Editable*. Todas las entidades generadas en nuestra escena contendrán este componente. En él, se encuentra el código que genera la paleta de colores y texturas de cada entidad. Para ello, se hace uso de un evento, el cual se ocupa de escuchar si la entidad es pulsada o arrastrada. Una vez ocurre esto, el componente es el encargado de crear la paleta (haciendo uso de un plano) y añadirle a esta los colores y texturas (que serán botones). Para evitar que se generen paletas infinitas cada vez que toques o arrastres la entidad, este mismo componente tiene una variable para asegurar que esto no ocurra. La paleta será entonces, hija de la entidad y los botones serán así vez hijos de la paleta. En el ejemplo que hemos visto antes, podemos ver que este componente se encuentra en la figura del podio, y se irá clonando para que todas las entidades creadas lo tengan.
- *Button*. Funciona de forma conjunta al anterior, ya que cada botón de la paleta recibirá este componente como atributo (esto se hace desde el anterior componente), y será este el que se encargue de modificar la textura o el color de la entidad padre. Para ello hace uso del evento que surge cuando se pulsa sobre cualquier botón de la paleta. Accederá al nodo padre (la entidad) y cambiará su color o textura (esto dependerá del botón que se pulse). Este componente no se encuentra en la escena inicial, ya que hasta que no se cree

la paleta no se crearán los botones, que como se ha comentado, son los que adoptan este componente.

- *Create-table*. Este componente lo encontramos en el botón del panel que se encarga de generar la mesa. Para ello, hace uso del evento que surge al pulsar sobre el botón de mesa. Una vez se pulsa, el componente será el encargado de generar la mesa y las entidades que flotan por encima de ellas y añadirlas a la escena. En este caso la mesa será hija de la escena en nuestro árbol de nodos.
- *Create-panel*. Al igual que se crea la mesa, se crea el panel de colores y la barra deslizante a través de este componente. Maneja un evento igual al que el anterior componente y se encuentra también al inicio, pero enlazado al botón de panel.
- *Edit-mode*. Este componente se enlaza con el elemento que representa al usuario y es el encargado de detectar si se pulsa la tecla necesaria para entrar y salir del modo edición. Para esto, escucha al evento encargado de la pulsación de cualquier teclas, y una vez se activa, diferencia entre las teclas que se han pulsado y realiza una acción u otra. Si se pulsa 'G' entra en el modo edición, si se pulsa 'H' sale. Dentro de este están declarados todos los parámetros y funciones para dotar tanto a la escena como a las entidades de ejes. Todos los ejes (los de la escena y los de las entidades) reciben una clase y un parámetro para que se encuentren invisibles para el usuario. Cuando se pulse la tecla para activar el modo edición, gracias a este componente accedemos a todas las entidades que tienen la clase de los ejes y se hacen visibles creando el modo edición. Al contrario, se vuelven invisibles cuando se quiera volver al modo normal.
- *Enter-edit-mode*. Como pulsar teclas no es intuitivo, se crean dos botones para entrar y salir del modo edición cuando se usen las gafas VR. Este componente y el siguiente se encargan de ello. Al igual que el anterior, se encarga de los ejes y de detectar en este caso solo el botón de entrada al modo edición.
- *Exit-edit-mode*. Se enlaza con el botón de salida de modo edición para cuando se usen las gafas VR. Vuelve a dejar la escena como se encontraba antes del modo edición y elimina los ejes.

Gracias a todos los eventos que manejan y todas las entidades implicadas, se logra llegar a tener una versión compleja donde podemos generar, mover y editar entidades como la vista anteriormente en la Figura 4.3, pero esta vez desde un punto de vista más técnico y no como un manual para usuarios.

Por último, añadir la siguiente imagen (Figura 4.4), donde podemos observar una escena compleja, pero desde un dispositivo de gafas de realidad virtual. En este caso, la estructura cambiará añadiendo dos botones más y cada botón tendrá los dos últimos componentes respectivamente.

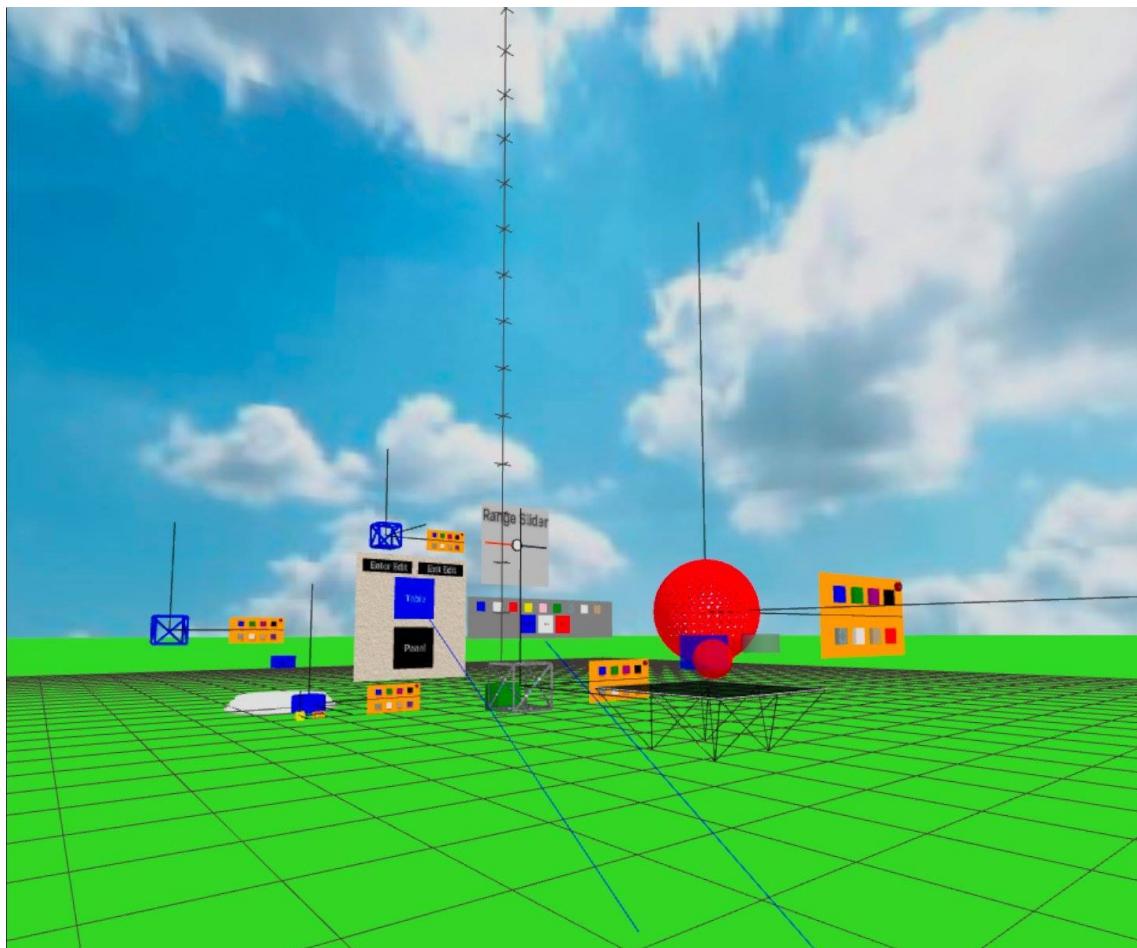


Figure 4.4: Escena compleja mediante dispositivo de realidad virtual

Capítulo 5

Conclusiones

5.1 Consecución de objetivos

En este apartado, echaremos la vista atrás al capítulo de objetivos (Capítulo 1) y se debatirá si se logró la consecución y los problemas que surgieron.

En general, todos los objetivos propuestos se han solventado con éxito con el paso de las iteraciones. Se ha logrado desarrollar un entorno en realidad virtual didáctico y de fácil uso en el navegador. Trata de una escena en tres dimensiones, en la cual, encontramos un espacio infinito con un suelo a los pies del usuario y, de frente, un panel y un podio para poder empezar con la tarea deseada.

Se aportan dos modos de creación de figuras, mesa de entidades y panel con botones. Cada entidad tiene su propia paleta de edición de color y textura, por si se cambia de opinión sobre su estilo una vez creada. Añadiendo, la capacidad de elección de tamaño a través de una barra deslizable.

Como añadido, se realizaron las modificaciones pertinentes para permitir al usuario cambiar entre la versión de escritorio y la versión con gafas de realidad virtual, agregando botones para hacer más intuitivo su acceso mediante los mandos u acciones como agrandar una entidad con los dos mandos.

Se consiguió una correcta movilidad del usuario por la escena, permitiéndole volar y girar en cualquier dirección, además de agregar su versión con gafas, en la que el sentido de movimiento viene dado por la dirección de nuestra vista.

El modo edición de la escena, que nos permite observar los ejes de las entidades, así como su

esqueleto, se implementó con éxito permitiendo al usuario obtener una visión más esquemática de la escena.

Por último, se logró correctamente alojar todos los ejemplos funcionales y la versión final del proyecto en GitHub para el uso de cualquier usuario, como se ha podido observar en los enlaces que se adjuntaban a lo largo del capítulo 3.

5.2 Aplicación de lo aprendido

Durante mi grado he recibido muchas lecciones en diferentes ramas de las tecnologías, entre ellas el software, el cual es la base de este proyecto. Algunas de las asignaturas que me han hecho más llevadero el avance son las siguientes:

1. Asignaturas como *Servicios y Aplicaciones Telemáticas* me han permitido obtener una base en HTML, además de conocer la estructura del DOM y el diagnóstico de errores en la consola del navegador.
2. Gracias a la asignatura *Desarrollo de Aplicaciones Telemáticas*, pude cumplimentar mi aprendizaje en JavaScript ya que coincidió el lapso de tiempo de la asignatura con el trabajo.
3. *Ingeniería en Sistemas de Información* fue la encargada de enseñarme todos los conocimientos necesarios para el control de versiones y el uso de GitHub.
4. *Sistemas Operativos*, además de aportarme los conocimientos sobre un lenguaje de programación, me proporcionó unas nociones básicas del manejo de la consola de Linux, con la cual pude manejar mis archivos, crear un servidor donde probar mi trabajo e instalar diversas aplicaciones como Atom.
5. Todas las materias que consistían en fundamentos de programación han sido las responsables de aportarme los cimientos de distintos lenguajes de programación, así como la algoritmia y pensamiento que hay detrás de ello.

5.3 Lecciones aprendidas

A continuación, se encuentran todos los conocimientos adquiridos gracias a este Trabajo de Fin de Grado.

1. Aprendizaje del framework para creación de experiencias A-Frame. En esto se incluye: creación de escenas, desarrollo de componentes, agregación y edición de entidades, control de la cámara e iluminación. Para esto, se profundizó en la estructura ECS y en la estructura de elementos HTML, etiquetas y atributos.
2. Manejo de eventos y modificación de elementos a tiempo real mediante el uso de JavaScript.
3. Competencia en la resolución de errores, como la propagación de eventos padre-hijo vista en el apartado 3.3.
4. Capacidad de crear un servidor de forma local donde poder depurar y probar el trabajo realizado.
5. Habilidad para desenvolverme con el uso de las gafas de realidad virtual, así como la adaptación del trabajo a estas y la conexión con el ordenador para permitir una depuración más veloz.
6. Importación de modelos 3D a la escena y correcta inyección, permitiendo usar las animaciones vinculadas a estos.
7. Manejo de la plataforma de control de versiones GitHub.
8. Uso de LaTeX para la edición de textos científico-técnicos.

5.4 Trabajos futuros

Todos los proyectos basados en ingeniería de software siguen una máxima: un trabajo nunca se termina, siempre podemos mejorarlo y añadir funcionalidades. Por lo que a continuación se incluyen algunas ideas que serían factibles incluir en un futuro:

- Añadir una función para cambiar entre el láser que se usa actualmente por unas manos, con la misión de poder interactuar con las entidades y los botones con ellas.

- Incluir dinámicas en las entidades para poder apilarlas, tirarlas y dotarlas de movimiento en general.
- Opción de guardar una entidad específica con su color, tamaño y posición al dejarla sobre un podio o la propia mesa, con el objetivo de usarla más tarde o incluso clonarla.
- Opción de moldear las entidades a nuestro gusto, por ejemplo quitándoles los vértices o creando agujeros en ellas.
- Agregar acciones mediante movimientos de las manos.

Bibliography

- [1] Atom flight manual.
[https://flight-manual.atom.io/.](https://flight-manual.atom.io/)
- [2] Git manual.
[https://git-scm.com/docs.](https://git-scm.com/docs)
- [3] Introduction to scrum.
[https://www.scrum.org/resources/scrum-guide.](https://www.scrum.org/resources/scrum-guide)
- [4] JavaScript documentation.
[https://developer.mozilla.org/es/docs/Web/JavaScript.](https://developer.mozilla.org/es/docs/Web/JavaScript)
- [5] Node.js documentation.
[https://nodejs.org/es/docs/.](https://nodejs.org/es/docs/)
- [6] Three.js Documentation.
[https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene.](https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene)
- [7] A-Frame. Getting started.
[https://aframe.io/docs/1.0.0/introduction.](https://aframe.io/docs/1.0.0/introduction)
- [8] R. Baruah. *AR and VR Using the WebXR API: Learn to Create Immersive Content with WebGL, Three.js, and A-Frame*. Apress, 2021.
- [9] P. Cozzi. *WebGL Insights*. CRC Press, July 2015.
- [10] A. Freeman. *The Definitive Guide to HTML5*. Apress, USA, 1st edition, 2011.
- [11] L. Lamport. *LATEX: a document preparation system: user's guide and reference manual*. Addison-wesley, 1994.

- [12] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan. Curating github for engineered software projects. *Empirical Software Engineering*, 22(6):3219–3253, 2017.