



INGENIERÍA EN TECNOLOGÍAS DE LA TELECOMUNICACIÓN

Curso Académico 2023/2024

Trabajo Fin de Grado/Máster

REINTERPRETACIÓN DEL JUEGO DEL TETRIS EN REALIDAD VIRTUAL

Autor : Víctor Blasco Robles

Tutor : Dr. Jesús María González Barahona

©2024 Autor Víctor Blasco

Algunos derechos reservados

Este documento se distribuye bajo la licencia “Atribución-CompartirIgual 4.0 International” de Creative Commons, disponible en <https://creativecommons.org/licenses/by-sa/4.0/deed.es>

Trabajo Fin de Grado/Máster

Reinterpretación del juego del Tetris en realidad virtual.

Autor : Víctor Blasco Robles

Tutor : Dr. Jesús María González Barahona

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

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 202X

Agradecimientos

Doy las gracias especialmente a mis padres y a mi pareja, Laura, por confiar siempre en mí y por darme su apoyo incondicional que me sirvió para continuar en muchos momentos difíciles.

También agradezco este trabajo a mis amigos por hacer mucho más ameno el camino.

Por último, no me olvido de Jesús y Gregorio, que ayudaron a que me gustara la programación y decidiera mi futuro laboral.

AGRADECIMIENTOS

Resumen

Este proyecto ha sido diseñado con el objetivo de conocer un poco más de cerca dos mundos muy importantes hoy en día, el de los videojuegos y el de las nuevas tecnologías. Para ello se ha realizado una exploración de ambos ámbitos y se ha llegado a la conclusión de que son totalmente compatibles entre ellos y que el avance de la tecnología sigue el fin de ir diseñando cada vez más videojuegos que utilicen estos entornos.

Estos conceptos relacionados con el mundo de la tecnología y los videojuegos se entienden mucho mejor de manera práctica, por lo que la parte central del proyecto se ha basado en el desarrollo de una aplicación web que une estos dos mundos para conocer las nuevas posibilidades que introduce esta unión.

Existen muchos tipos de aplicaciones que pueden ser diseñadas para ser desplegadas en entornos de realidad virtual, pero se ha decidido finalmente proceder a la adaptación de un videojuego clásico a los entornos de realidad virtual. Esta aplicación por lo tanto ha seguido el funcionamiento clásico del famoso videojuego arcade llamado *Tetris* pero se ha desarrollado listo para ser desplegado también en entornos de realidad virtual. Para ello se han creado distintos modos de juego que incorporan nuevas funcionalidades y reglas que ponen de manifiesto el gran abanico de posibilidades que ofrece la realidad virtual aplicada a los videojuegos.

Una vez finalizado el desarrollo de la aplicación se ha podido comprobar de forma práctica las nuevas posibilidades que ofrecen los entornos de realidad virtual y las nuevas aplicaciones que se podrán crear mediante el uso de estas tecnologías.

RESUMEN

Summary

This project has been designed with the aim of getting to know two very important worlds today, that of video games and that of new technologies, a little more closely. To this end, an exploration of both areas has been carried out and the conclusion has been reached that they are fully compatible with each other and that the advancement of technology continues with the aim of designing more and more video games that use these environments.

These concepts related to the world of technology and video games are much better understood in a practical way, so the central part of the project has been based on the development of a web application that unites these two worlds to learn about the new possibilities it introduces. this union.

There are many types of applications that can be designed to be deployed in virtual reality environments, but it has finally been decided to proceed with the adaptation of a classic video game to virtual reality environments. This application has therefore followed the classic operation of the famous arcade video game called *Tetris* but has been developed ready to also be deployed in virtual reality environments. To achieve this, different game modes have been created that incorporate new functionalities and rules that reveal the wide range of possibilities offered by virtual reality applied to video games.

Once the development of the application was completed, it was possible to practically verify the new possibilities offered by virtual reality environments and the new applications that can be created through the use of these technologies.

SUMMARY

Índice general

1. Introducción	1
1.1. Contexto	1
1.2. Motivación Personal	2
1.3. Objetivos	3
1.3.1. Objetivo general	3
1.3.2. Objetivos específicos	3
1.4. Estructura de la memoria	4
1.5. Aplicaciones relacionadas	5
2. Tecnologías utilizadas	7
2.1. Tecnologías Principales	7
2.1.1. HTML5	7
2.1.2. JavaScript	8
2.1.3. A-Frame	9
2.1.4. DOM	11
2.1.5. WebXR	12
2.1.6. Super Hands	13
2.1.7. Oculus VR	14
2.2. Tecnologías Auxiliares	15
2.2.1. A-Frame Visual Inspector	15
2.2.2. Oculus Developer Hub	16
2.2.3. Atom	17
2.2.4. Visual Studio Code	17
2.2.5. GitHub	18

ÍNDICE GENERAL

2.2.6. Git	19
2.2.7. GitHub Desktop	20
2.2.8. LaTeX	21
3. Descripción Técnica	23
3.1. Manual de Usuario	23
3.1.1. Menú Principal	23
3.1.2. Menús Secundarios	24
3.1.3. Menús de Elección	25
3.1.4. Modo Standard	25
3.1.5. Modo Colors	26
3.1.6. Modo Obstacles	27
3.1.7. Modo Multiboard	28
3.1.8. Modo 180 Degrees	29
3.1.9. Modo Dropping pieces	31
3.1.10. Modo Reaction test	32
3.1.11. Menú fin de juego	33
3.2. Componentes principales	34
4. Proceso y Desarrollo	39
4.1. Modelo de desarrollo	39
4.2. Etapa 0	41
4.2.1. Objetivos de Diseño	41
4.2.2. Detalles de Implementación	41
4.2.3. Prototipo Final	44
4.3. Etapa 1	46
4.3.1. Objetivos de Diseño	46
4.3.2. Detalles de Implementación	46
4.3.3. Prototipo Final	49
4.4. Etapa 2	51
4.4.1. Objetivos de Diseño	51
4.4.2. Detalles de Implementación	51

ÍNDICE GENERAL

4.4.3. Prototipo Final	54
4.5. Etapa 3	56
4.5.1. Objetivos de Diseño	56
4.5.2. Detalles de Implementación	56
4.5.3. Prototipo Final	58
4.6. Etapa 4	60
4.6.1. Objetivos de Diseño	60
4.6.2. Detalles de Implementación	60
4.6.3. Prototipo Final	62
4.7. Etapa 5	63
4.7.1. Objetivos de Diseño	63
4.7.2. Detalles de Implementación	63
4.7.3. Prototipo Final	69
4.8. Etapa 6	70
4.8.1. Objetivos de Diseño	70
4.8.2. Detalles de Implementación	70
4.8.3. Prototipo Final	73
4.9. Etapa 7	74
4.9.1. Objetivos de Diseño	74
4.9.2. Detalles de Implementación	74
4.9.3. Prototipo Final	77
5. Conclusiones	79
5.1. Conclusiones generales	79
5.2. Consecución de objetivos	81
5.3. Planificación temporal	82
5.4. Aplicación de lo aprendido	82
5.5. Lecciones aprendidas	83
5.6. Trabajos futuros	84
5.7. Material de interés	84
6. Anexo: Elementos A-Frame	85

ÍNDICE GENERAL

7. Anexo: Componentes y sus propiedades	91
Bibliografía	93

Índice de figuras

3.1. Menú Principal de la aplicación	24
3.2. Menú Secundario de los modos de escritorio	24
3.3. Menú Secundario de los modos de realidad virtual	25
3.4. Menú de Elección de uno de los modos de juego de la aplicación	25
3.5. Modo Estándar de la aplicación	26
3.6. Modo Colores de la aplicación	27
3.7. Modo Obstáculos de la aplicación	28
3.8. Modo multitablero de la aplicación	29
3.9. Modo 180 grados de la aplicación	30
3.10. Modo piezas flotantes de la aplicación	32
3.11. Modo test de reflejos de la aplicación	33
3.12. Menú de fin de juego de la aplicación	33
3.13. Ejemplo de registro de un nuevo componente (Parte HTML)	34
3.14. Ejemplo de registro de un nuevo componente (Parte JavaScript)	34
3.15. Declaración de funciones de un componente	35
3.16. Implementación completa de un componente	36
4.1. Declaración del componente principal	42
4.2. Registro del componente principal	42
4.3. Tablero de los rebobadores	43
4.4. Aplicación en funcionamiento	44
4.5. Declaración del fondo en HTML	47
4.6. Fondo creado a través de A-Frame Visual Inspector	47
4.7. Tablero estándar ya colocado sobre la escena	48

ÍNDICE DE FIGURAS

4.8. Declaración de los componentes principales del primer prototipo	49
4.9. Prototipo inicial del tablero	49
4.10. Creación de una pieza en la escena	52
4.11. Prototipo final de la Etapa 2	55
4.12. Interfaz de la aplicación de Oculus	57
4.13. Interfaz de la aplicación de Oculus Developer Hub	58
4.14. Ejemplo de funcionamiento del modo COLORES	61
4.15. Ejemplo de funcionamiento del modo OBSTÁCULOS	62
4.16. Ejemplo de funcionamiento del modo MÚLTIPLES TABLEROS	64
4.17. Ejemplo de funcionamiento del modo 360 GRADOS	66
4.18. Ejemplo de funcionamiento del modo PIEZAS EN CAÍDA LIBRE	68
4.19. Ejemplo de funcionamiento del modo TEST DE REFLEJOS	69
4.20. Menú principal	71
4.21. Menú secundario	72
4.22. Menú de elección	72
4.23. Menú de fin de juego	73
4.24. Declaración de un elemento audio o a-image	74
4.25. Inclusión de un elemento audio en un elemento visible	74
5.1. Evolución de la realidad virtual en el mundo de los videojuegos	80
6.1. Ejemplo de documento HTML de una página de la aplicación	85
6.2. Declaración del cursor	87

Capítulo 1

Introducción

Este proyecto sigue el objetivo de analizar las posibilidades que surgirían a partir de la adaptación de los videojuegos a las nuevas tecnologías. Para ello se ha realizado un estudio y se ha llegado a la conclusión de que la mejor manera de analizar esta situación es la creación de un videojuego capaz de ser desplegado en entornos de realidad virtual.

La aplicación diseñada trata sobre una versión del famoso videojuego arcade llamado *Tetris* que consta de múltiples modalidades y puede ser desplegada tanto en entornos de escritorio como en entornos de realidad virtual.

Cada modo de juego de la aplicación añade nuevas funciones que ayudan a entender las infinitas opciones de desarrollo que ofrecen estas tecnologías y la posibilidad de crear géneros de videojuegos o aplicaciones que jamás se han creado antes.

El proyecto está alojado en un servidor web y se puede acceder a toda su funcionalidad haciendo click en el enlace ¹.

Para entender el marco en el que ha sido diseñado este proyecto, es importante explicar primero varios conceptos, como el contexto del mismo, su motivación y la estructura de la memoria.

1.1. Contexto

El origen de los juegos recreativos se remonta a la década de 1970 [17], donde estos juegos se encontraban disponibles únicamente en máquinas recreativas. Este nuevo modelo de negocio

¹<https://victorbrobles.github.io/menuPrincipal>

revolucionó el mundo del entretenimiento pero tenía un grave problema con la escasa movilidad que ofrecía.

Por este motivo y debido al avance de la tecnología, estos juegos pasaron de las máquinas recreativas a las primeras videoconsolas, como la Gameboy o la Nintendo. En este nuevo escenario surgieron multitud de nuevos juegos, incluido el *Tetris*. Este juego se remonta al año 1984 [24] y consiste en encajar piezas de colores, que vienen de arriba a abajo, y llenar líneas horizontales para eliminarlas, con el objetivo de no llenar el tablero.

Con la aparición de nuevos dispositivos y nuevas tecnologías, hoy en día este tipo de juegos están disponibles en todo tipo de dispositivos y son funcionales en todo tipo de escenarios, como la realidad virtual. Este concepto se basa en la generación de espacios tridimensionales a partir de unas gafas que poseen un estereoscopio, que genera una doble imagen que se mezcla en nuestro cerebro como una única. Con el avance de esta tecnología ha surgido el concepto de *WebXR*, que es una API Javascript que proporciona soporte para dispositivos de realidad virtual y permite acceder al contenido de aplicaciones VR simplemente accediendo a una URL, sin necesidad de instalaciones o plugins. Gracias a esta tecnología es posible el renderizado 3D de una aplicación web construida sobre herramientas de programación sencillas, como las utilizadas en este proyecto.

Para poder desplegar una aplicación web también es importante resaltar la importancia del modelo de objeto de documento (DOM), una interfaz de programación para los documentos HTML y XML capaz de representar un conjunto de elementos HTML y ofrecer una interfaz para acceder a ellos y manipularlos, haciendo así que cambie la estructura de la página inicial.

Gracias a estas tecnologías hoy en día es posible desarrollar proyectos que interactuando únicamente con una interfaz web, son capaces de generar aplicaciones complejas y que funcionen con múltiples tecnologías. De esta forma podemos mezclar dos conceptos clave en la actualidad: la programación web y el desarrollo de aplicaciones en entornos de realidad virtual.

1.2. Motivación Personal

Siempre me he sentido interesado por el avance de las tecnologías y las nuevas funcionalidades que podían traer consigo. De hecho este fue uno de los principales motivos por los que escogí este grado. La tecnología avanza muy rápido y siempre es importante estar al tanto de

los avances porque en un futuro pueden ser de mucha utilidad para ti.

Antes de empezar el grado me preguntaba constantemente cómo se habían generado los juegos que habitualmente consumía, y esto motivó mi interés por la programación. Al empezar el grado la programación me resultó entretenida pero muy difícil al mismo tiempo. Me costaba mucho asentar las ideas básicas y cuando empezaba a conseguirlo tenía que empezar de cero con un lenguaje diferente. Entonces llegó la asignatura de *Servicios y Aplicaciones Telemáticas*, donde conseguí crear mi primera aplicación web. Esto supuso un cambio enorme para mí, y descubrí la satisfacción que podía llegar a conseguir programando. Hoy en día soy programador en una buena empresa y puedo decir que me gusta mi trabajo.

Mi último foco de motivación son los videojuegos. Desde pequeño siempre me he sentido atraído por el mundo que los envuelve y he soñado en algún momento con tener la oportunidad de crear alguno de cero. Cuando comencé a explorar las posibilidades que ofrecen los entornos de realidad virtual vi claramente que la mejor opción para poner en práctica los conceptos aprendidos era la creación de un videojuego. La realidad virtual es idónea para ser utilizada en este tipo de desarrollos y para mí era una gran motivación realizarlo, por ello decidí crear un videojuego completamente nuevo que supusiera la adaptación de un juego ya conocido para mí a estas nuevas tecnologías.

1.3. Objetivos

1.3.1. Objetivo general

El objetivo principal de este proyecto consiste en explorar las posibilidades y las nuevas funcionalidades que podría ofrecer la adaptación de una aplicación a los entornos de realidad virtual.

1.3.2. Objetivos específicos

Para llevar a cabo el objetivo principal es necesario también explicar otra serie de objetivos específicos que se han llevado a cabo:

- Explorar el marco en el que se encuentra actualmente la realidad virtual.

- Examinar una herramienta actual que permite la creación de aplicaciones en entornos de realidad virtual.
- Conocer los diferentes tipos de aplicaciones que se pueden desarrollar utilizando entornos de realidad virtual.
- Estudiar el marco actual en el que se encuentran los videojuegos arcade y las mejoras que podrían aplicarse a ellos hoy en día.
- Analizar un dispositivo actual de realidad virtual e incorporarlo al desarrollo de la aplicación.
- Explorar de forma práctica el resultado de la unión entre el mundo de la realidad virtual y el de las aplicaciones, mediante el desarrollo de una aplicación compatible con entornos de realidad virtual.
- Descubrir muchas de las funcionalidades que pueden llegar a surgir de esta unión mediante la creación de diferentes modos de juego en la aplicación.
- Finalizar el desarrollo de la aplicación e incorporar distintos efectos audiovisuales para contemplar cómo sería una aplicación completa de este tipo.
- Realizar el despliegue de la aplicación en un servidor web para facilitar su accesibilidad.

1.4. Estructura de la memoria

El proyecto está estructurado en cuatro capítulos principales organizados de la siguiente manera:

- En el primer capítulo se presentan todas las tecnologías utilizadas durante el desarrollo del proyecto. Cada una de ellas se encuentra enmarcada en el grupo de tecnologías principales o auxiliares, según haya sido su importancia en el desarrollo. Se aporta una breve descripción de cada una de ellas, así como la función para la que ha servido en este proyecto.

- En el segundo capítulo se detalla cada una de las fases por las que ha pasado el proyecto. Este es el capítulo principal de la memoria ya que se explica en él todos los desarrollos que se han llevado a cabo y todos los problemas que han ido surgiendo. Cada etapa se encuentra dividida en el objetivo de esa etapa, su desarrollo y los resultados finales que se obtienen al final de la misma.
- El tercer capítulo trata de aclarar el funcionamiento de la aplicación de cara al usuario final al que va dirigida. Contiene un apartado de arquitectura general en el que se explica de forma breve cada componente utilizado en los escenarios y las funciones que pueden afectarles. El siguiente apartado consiste en un manual de usuario que trata de explicar cómo funciona cada modo de juego y las interacciones que pueden producirse entre la aplicación y el usuario.
- El último capítulo consiste en las conclusiones finales del proyecto. Dentro de este capítulo se encuentra una breve explicación de los resultados finales que se han logrado así como los objetivos conseguidos, la planificación temporal que se ha seguido a la hora de realizar el proyecto, los conocimientos que se han utilizado y que se han adquirido, o posibles trabajos futuros a partir de este.

1.5. Aplicaciones relacionadas

A pesar de ser una tecnología relativamente nueva, existen muchas aplicaciones colgadas en la web cuyo desarrollo está basado en A-Frame. Esto puede servir de ayuda a los desarrolladores que vayan a utilizar por primera vez esta tecnología, ya que cada ejemplo puede enseñar un aspecto distinto de A-Frame.

En este apartado se van a mostrar algunos de estos ejemplos:

- Este sencillo programa ² enseña cómo crear una escena A-Frame con varios elementos simples dentro de ella.
- Este ejemplo ³ muestra una nueva característica de A-Frame, la posibilidad de dotar de animaciones a los elementos que componen una escena.

²<https://stemkoski.github.io/A-Frame-Examples>Hello-WebVR.html>

³<https://stemkoski.github.io/A-Frame-Examples/animation.html>

- En este caso ⁴ se puede apreciar la posibilidad de introducir objetos que no son propios de A-Frame pero pueden comportarse como tal.
- Este ejemplo ⁵ muestra como se comportan los elementos cuando se utilizan las físicas que ofrece A-Frame.
- Esta es una aplicación ⁶ totalmente completa, que incorpora elementos externos a A-Frame, físicas e incluso efectos audiovisuales. Este es un gran ejemplo de todo lo que se puede llegar a conseguir a través de esta tecnología, un videojuego sencillo pero totalmente funcional.

⁴<https://stemkoski.github.io/A-Frame-Examples/parametric-path-follow.html>

⁵<https://stemkoski.github.io/A-Frame-Examples/physics.html>

⁶<https://shaqian.github.io/pacman/>

Capítulo 2

Tecnologías utilizadas

En esta sección se exponen las diferentes tecnologías que se han utilizado para el desarrollo del proyecto, con una breve explicación de cada una de ellas.

2.1. Tecnologías Principales

2.1.1. HTML5

```
<!DOCTYPE html>
<html>
  <head>
    <title>Welcome To MyTetrisVR</title>
    <script src="https://aframe.io/releases/1.0.4/aframe.min.js"></script>
    <script src="https://cdn.rawgit.com/donmccurdy/aframe-extras/v4.1.2/dist/aframe-extras.min.js"></script>
    <script src="https://rawgit.com/feliss/aframe-environment-component/master/dist/aframe-environment-component.min.js"></script>
    <script src="https://unpkg.com/super-hands@^3.0.1/dist/super-hands.min.js"></script>
    <script src="js/menu.js"></script>
    <script src="https://github.com/wmurphyrd/aframe-super-hands-component/tree/master/examples/build.js"></script>
  </head>
  <body>
```

Es un lenguaje de marcado construido a base de etiquetas que se encarga de establecer la estructura de una página web enlazando diferentes contenidos o archivos [22].

Esta tecnología fue desarrollada para la divulgación de información con texto o imágenes, sin embargo hoy en día tiene muchas más funcionalidades debido a los estándares que se han desarrollado a lo largo del tiempo. No se considera un lenguaje de programación ya que no puede crear funcionalidades dinámicas, si no que se emplea para definir la estructura del documento. Sus principales usos son el desarrollo web, la navegación por Internet o la documentación web [16].

Este lenguaje presenta múltiples ventajas como la facilidad de aprendizaje, su amplia documentación y soporte, la facilidad de integración con otros lenguajes de backend o su acce-

sibilidad al tratarse de código abierto. A pesar de ello también presenta inconvenientes al ser un lenguaje estático que no permite crear páginas con funcionalidad compleja sin apoyarse en otros lenguajes. Por este motivo los documentos HTML suelen combinarse con archivos CSS que componen el estilo de las páginas y archivos JavaScript que introducen funcionalidades dinámicas.

Esta versión de HTML es la más moderna e incluye APIs que permiten extender la funcionalidad de una página web utilizando la programación con JavaScript. Una de estas APIs es el DOM, que se utiliza para alterar dinámicamente el aspecto de la página web mediante la propagación y captura de eventos. La estructura de este proyecto está basada en HTML, mientras que el DOM se encarga de modificarla de forma dinámica.

2.1.2. JavaScript

```
function getRandomColor() {
  var letters = '0123456789ABCDEF';
  var color = '#';
  for (var i = 0; i < 6; i++) {
    color += letters[Math.floor(Math.random() * 16)];
  }
  return color;
}
```

JavaScript [18] es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se trata de un lenguaje orientado a objetos, los cuales contienen información en forma de campos que pueden ser modificados a través de sus métodos. Es dinámico y débilmente tipado, por lo que se pueden declarar variables de cualquier tipo en un mismo escenario. Se utiliza principalmente en el lado del cliente, permitiendo modificar la interfaz de una página web o aportar funcionalidad dinámica a los distintos elementos de ella.

Es un lenguaje muy habitual en la actualidad al ser el único que entienden los navegadores. Su funcionalidad principal consiste en desarrollar la funcionalidad frontend en las aplicaciones web modernas, permitiéndonos crear efectos especiales en las páginas y definir interactividades con el usuario. El navegador del cliente se encarga de interpretar las instrucciones JavaScript y ejecutarlas para realizar estos efectos e interactividades, por lo que podemos decir que el mayor recurso con el que cuenta este lenguaje es el propio navegador y los elementos que existen en una página web.

Una de las mayores ventajas que ofrece JavaScript es su capacidad para ser integrado junto a otros lenguajes de programación [12]. Surgió con el objetivo de programar ciertos comporta-

mientos sobre las páginas web, pero en la actualidad ha dejado de ser un lenguaje de scripting para convertirse en un lenguaje integrador de múltiples funcionalidades. Cuando se usa como lenguaje del lado del cliente suele estar acompañado de otros lenguajes de programación web como HTML o CSS, mientras que cuando es usado del lado del servidor [5] puede ir acompañado de otras tecnologías como NodeJS o Django. La combinación de JavaScript con estos lenguajes y tecnologías ha permitido que las aplicaciones web más complejas estén desarrolladas con altas dosis de JavaScript: aplicaciones como Netflix, editores de texto como Atom, aplicaciones móviles, de mensajería, y una extensa lista de servicios web.

Este proyecto utiliza JavaScript como enlace entre los elementos HTML que componen la página web y los eventos que son capturados mediante el DOM, encargándose así de modificar las propiedades de estos elementos cuando los diferentes eventos son capturados.

2.1.3. A-Frame

```
<audio id="tetris" src="assets/tetris.mp3" crossorigin="anonymous"></audio>
<audio id="button" src="assets/button.wav" crossorigin="anonymous"></audio>
<a-image src="assets/logo.png" position="0 14 -20" width="30" height="8"></a-image>

<a-mixin id="pointer" super-hands="colliderEvent: raycaster-intersection;
                           colliderEventProperty: els;
                           colliderEndEvent:raycaster-intersection-cleared;
                           colliderEndEventProperty: clearedEls;">
</a-mixin>
```

A-Frame [1] es un framework web de código abierto para crear experiencias de realidad virtual. Es una estructura creada para Three.js donde los desarrolladores pueden crear escenas 3D y WebXR usando HTML. Fue desarrollado por el equipo de Mozilla VR a finales del 2015 y actualmente cuenta con más de 75 contribuyentes en total.

Utiliza la arquitectura ECS (Entity Component System), usada en el desarrollo de juegos donde cada objeto es una entidad. Estos objetos no siguen una jerarquía establecida, por lo que pueden tener un comportamiento sin límites, lo que ofrece una extensa gama de posibilidades. En esta arquitectura tenemos:

- **Entidades.** Contienen los objetos y se encargan de otorgarles sus propiedades.
- **Componentes.** Son las propiedades que distinguen unas entidades de otras, ya sea por su comportamiento, apariencia o funcionalidad.
- **Sistemas o Escenas.** Proporcionan el entorno donde se desarrollan los componentes. Los

sistemas se encargan de la lógica del uso de la información que contienen los componentes.

A-Frame es un lenguaje declarativo, apoyado en HTML y basado en el DOM. Este framework permite modificar el prototipo de un elemento HTML para añadir comportamiento adicional que utilizan ciertas APIs del DOM para adaptar el código a A-Frame. Las escenas se crean mediante HTML y A-Frame, mientras que el comportamiento de sus componentes se maneja mediante JavaScript y el DOM, que proporciona múltiples capacidades para esta tecnología:

- **Referencia a otras entidades con selectores de consulta.** El DOM proporciona un sistema selector de consultas que nos permite seleccionar una entidad o entidades que coincidan con una condición.
- **Comunicación cruzada entre entidades con desacoplamiento de eventos.** El DOM proporciona la capacidad de escuchar y emitir eventos, proporcionando un sistema de comunicación entre entidades. Los componentes no tienen conocimiento de otros componentes, estos solo emiten eventos y otros componentes pueden escuchar esos eventos.
- **APIs para la gestión del ciclo de vida con las API de DOM.** El DOM proporciona API para actualizar los elementos HTML y sus atributos.
- **Filtro de entidad con selectores de atributos.** El DOM proporciona selectores de atributos que nos permiten consultar una entidad o entidades que tienen o no ciertos atributos HTML.
- **Declarativo.** Por último, el DOM proporciona HTML. A-Frame es el puente entre ECS y HTML haciendo un patrón ya limpio declarativo, legible y extensible.

En la actualidad A-Frame es un lenguaje muy utilizado en el entorno de las escenas de realidad virtual [2]. Es compatible con todo tipo de navegadores y dispositivos VR, así como Gear VR, Vive, Rift, Daydream y Oculus. También proporciona herramientas propias como el Visual Inspector, que ofrece una interfaz que permite alterar la apariencia de una escena que emplea A-Frame.

A-Frame es la base de este proyecto dado que todos los elementos que componen las diferentes escenas son elementos propios de esta tecnología. Se combina con HTML para colocar

sus elementos sobre la escena y con JavaScript para construirlos de la forma deseada y alterar su apariencia de forma dinámica.

2.1.4. DOM

```
var suelo = document.createElement('a-box');
```

El Document Object Model o DOM [11] es una interfaz o API de programación que proporciona un conjunto de objetos para representar documentos HTML, XML, XHTML o SVG. Fue creado por la compañía Netscape Communications como un conjunto de objetos que servían de interfaz entre JavaScript y el propio documento. A través del DOM los programas son capaces de acceder a los documentos y modificar el contenido, estructura o estilo de los mismos [27].

Los objetos del DOM son capaces de modelizar tanto la ventana del navegador, como el historial, el documento o cada uno de los elementos presentes en la página. Puede acceder a cualquiera de los elementos a través de JavaScript y modificar sus propiedades o invocar sus métodos. Cuando se realiza alguna modificación sobre la página, el código JavaScript apunta a un elemento a través del DOM y así realiza el cambio. El DOM por lo tanto tiene tres funcionalidades principales: realizar referencias a objetos, manipular las propiedades y funciones de los elementos, y manejar los distintos eventos que se producen en una página.

Uno de los principales problemas del DOM se basa en los problemas de compatibilidad, ya que distintos navegadores pueden tratar el mismo código de manera diferente, especialmente al utilizar JavaScript. Este problema se ha reducido gracias al movimiento en la creación de estándares web, ya que el DOM es uno de los elementos que se han estandarizado, haciendo que todos los navegadores vayan en la misma dirección. Actualmente está dirigido por el World Wide Web Consortium (W3C) y esto permite crear código JavaScript que funcione de igual manera en cualquier tipo de navegador.

En la actualidad existen diferentes alternativas al DOM, como los lenguajes React o Lit, que sustituyen el DOM por sistemas de templates y enlaces de datos. Sin embargo, el DOM sigue siendo una herramienta muy utilizada con JavaScript [8], e incluso permite el empleo de librerías como jQuery que facilitan mucho su labor.

El DOM es un elemento completamente necesario en el desarrollo de este proyecto ya que se utiliza constantemente cuando se quiere acceder a las propiedades de los elementos o a los

eventos que son disparados.

2.1.5. WebXR



La WebXR [19] es una API JavaScript que proporciona soporte para dispositivos de realidad virtual en un navegador web. Consta principalmente de una serie de librerías JavaScript que permiten a los navegadores más recientes ejecutar estos contenidos. A pesar de estar estandarizada, cada navegador puede aportar librerías adicionales para dar soporte a sus propios dispositivos o funcionalidades, de forma que se puedan crear experiencias de Realidad Aumentada o Mixta.

La API [23] se encuentra todavía en un proceso experimental y se está desarrollando con tres objetivos principales en mente: detectar los dispositivos de Realidad Virtual disponibles, obtener tanto las características como la posición y orientación de dichos dispositivos, y mostrar imágenes en el dispositivo con la frecuencia de refresco adecuada.

Pese a que todavía se encuentra en fase de desarrollo, presenta una serie de ventajas que la convierten en una herramienta muy a tener en cuenta:

- Mientras que el acceso a este tipo de experiencias suele requerir una serie de conocimientos técnicos previos, la WebXR es una herramienta de fácil acceso ya que podemos acceder a ella con un simple clic en un enlace.
- Presenta compatibilidad multi-navegador y multi-dispositivo. Esto permite acceder a ella a través de cualquier navegador e incluso sin necesidad de cascos VR.
- Está estandarizada y sus códigos son abiertos y de libre uso, permitiendo acceder a ella sin necesidad de pagar licencia de uso.
- Al igual que una página web común, permite la descarga progresiva de elementos a medida que el usuario los requiera.

- Permite el registro y evaluación de los usuarios que acceden a ella.
- Todo su contenido se encuentra alojado en la nube, permitiendo que las experiencias sean modificadas sin necesidad de que el usuario tenga que actualizar nada.

Esta aplicación está diseñada para ser lanzada en un navegador web que utilice esta API, que permite el despliegue tanto en versión de escritorio como en versión de realidad virtual. La combinación de esta herramienta y el lenguaje A-Frame [4] ha permitido el desarrollo de esta aplicación en entornos de realidad virtual.

2.1.6. Super Hands

Super Hands [14] es una librería JavaScript exclusiva de la tecnología A-Frame alojada en el repositorio de GitHub ¹, que sirve para añadir interacciones intuitivas mediante el ratón o los controladores VR en A-Frame. Ofrece una API de alto nivel compatible con todo tipo de dispositivos que se encarga de facilitar el manejo de las interacciones entre el usuario y la aplicación.

Actualmente maneja cuatro tipos de eventos o interacciones:

- **Hover.** Se produce cuando el controlador apunta a un elemento o se encuentra en su espacio de colisión.
- **Grab.** Se dispara pulsando un botón mientras el controlador se encuentra sobre un elemento.
- **Stretch.** Se produce cuando realizas la acción de agarrar un elemento con ambas manos y realiza un ajuste de tamaño.
- **Drag-drop.** Se activa cuando apuntas a un elemento y realizas con él alguna acción sobre otro, interactuando entre ellos.

Esta librería es totalmente compatible con A-Frame y presenta una alternativa al uso de físicas para el manejo de interacciones entre distintos elementos. Cuando se produce uno de estos eventos entre el controlador y los elementos de la escena, Super Hands lo recoge y se

¹<https://github.com/c-frame/aframe-super-hands-component>

encarga de realizar las acciones correspondientes en cada caso. También permite que el usuario pueda acceder a los eventos que se han activado a través del DOM.

Todas las interacciones entre el usuario y los elementos que componen esta aplicación son controlados gracias a esta librería, sustituyendo así a las físicas que proporciona A-Frame.

2.1.7. Oculus VR



Oculus VR [10] es una empresa norteamericana de realidad virtual fundada por Palmer Luckey en 2012, en la ciudad de California. Dos años después de su fundación, la empresa fue comprada por Meta (anteriormente Facebook) y crearon uno de sus principales productos, el Oculus Rift. Esta empresa siempre ha sido considerada una de las pioneras en el desarrollo del mundo virtual, y en la actualidad cuentan con una amplia gama de productos como el Oculus Rift, el Oculus Go, el Oculus Quest o el Meta Quest 2.

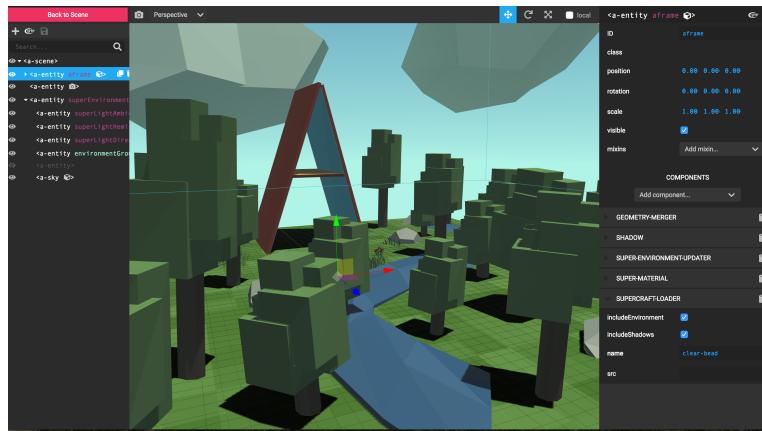
El último producto que ha salido al mercado es el Meta Quest 3. Este dispositivo cuenta con unas gafas VR y unos controladores Touch de última generación. Es compatible con todo tipo de navegadores, incluida la webXR, y permite la conectividad con ordenadores a mediante el cable Link o la tecnología Air Link, que utiliza bluetooth.

Oculus también presenta una aplicación móvil o de escritorio que permite al usuario registrarse y gestionar todos sus dispositivos. La aplicación puede ser funcional tanto a nivel de cliente como a nivel profesional. A nivel de cliente ofrece una amplia gama de productos que el usuario puede añadir a su cuenta, tanto gratuitos como de pago. A nivel profesional permite convertir la cuenta en una de desarrollador, un requisito indispensable para poder conectar un dispositivo con el ordenador.

Para el desarrollo de esta aplicación se han usado las gafas Meta Quest 2 [20] con el objetivo de controlar el correcto funcionamiento del proyecto en el entorno de realidad virtual. También ha sido necesaria la aplicación de escritorio para conectar este dispositivo al ordenador y así visualizar la aplicación con él.

2.2. Tecnologías Auxiliares

2.2.1. A-Frame Visual Inspector



El inspector visual de A-Frame [3] es una herramienta que ofrece este framework y permite modificar una escena o los componentes que contiene. Para acceder a esta herramienta es necesario utilizar la combinación de teclas **ctrl + alt + i** sobre una escena A-Frame.

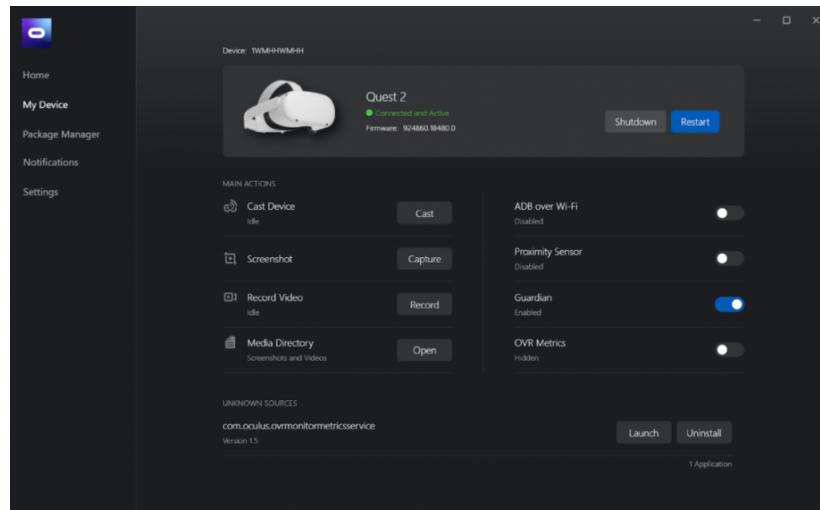
La interfaz del inspector consta de tres partes principales: la zona izquierda donde se muestran los componentes de la escena, la zona central donde se muestra la apariencia actual de la escena y la zona derecha donde se muestra un menú para modificar los distintos componentes.

En el menú izquierdo se encuentra en primer lugar una lista con la escena y todos los componentes que forman parte de ella. Sobre este menú se pueden realizar cuatro tipos de acciones principales: añadir un nuevo componente a la escena, borrarlo, clonarlo, o inspeccionarlo, haciendo que ese componente aparezca en el menú derecho.

En el menú derecho se observan todas las propiedades del componente o escena que se ha seleccionado previamente. Con esta herramienta podemos alterar cualquier propiedad, borrarla o incluso añadir nuevas propiedades. Según vayan cambiando las propiedades del componente, la apariencia de la escena cambiará en la zona central, mostrando el aspecto actual de la misma tras los cambios.

Todos los entornos o escenas de este proyecto han sido creadas con la ayuda de esta herramienta debido a las facilidades que ofrece a la hora de visualizar en vivo las modificaciones que pueden realizarse sobre los distintos elementos que componen la escena.

2.2.2. Oculus Developer Hub



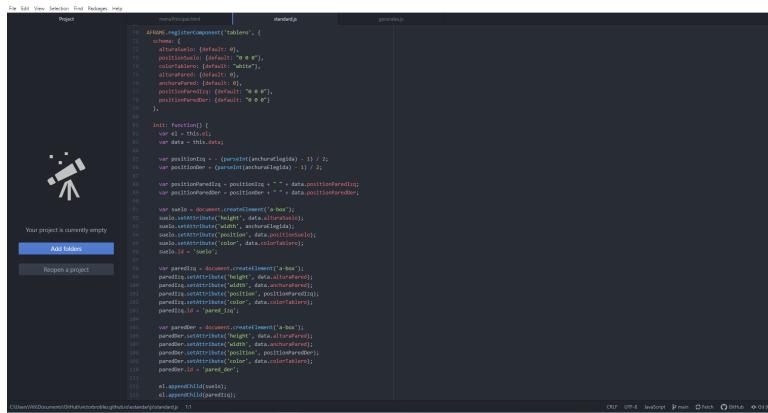
El Oculus Developer Hub [21] consiste en una aplicación que mejora la experiencia de desarrollo en realidad virtual y la forma en la que los desarrolladores interactúan con los cascos Oculus desde su ordenador. El desarrollo de realidad virtual requiere la creación de experiencias 3D mediante pantallas 2D, y el ODH se encarga de agilizar esta transición. También incluye una serie de herramientas que ayudan a controlar el rendimiento de la experiencia de realidad virtual.

Esta aplicación permite la sincronización de cuentas Oculus. Cuando la cuenta sincronizada consiste en una cuenta de desarrollador, el ODH permite la creación de organizaciones y aplicaciones, que pueden ser reales o de test [13].

También permite la sincronización de dispositivos Oculus e incorpora una serie de herramientas para analizar su funcionamiento.

En este proyecto ha sido de utilidad la herramienta *ADB over Wi-Fi*, que permite que la consola de comandos del ordenador haga de puente entre este y los cascos Oculus. Esto permite, por ejemplo, que podamos acceder al inspector del navegador en el que está desplegada la aplicación a través de los cascos VR.

2.2.3. Atom

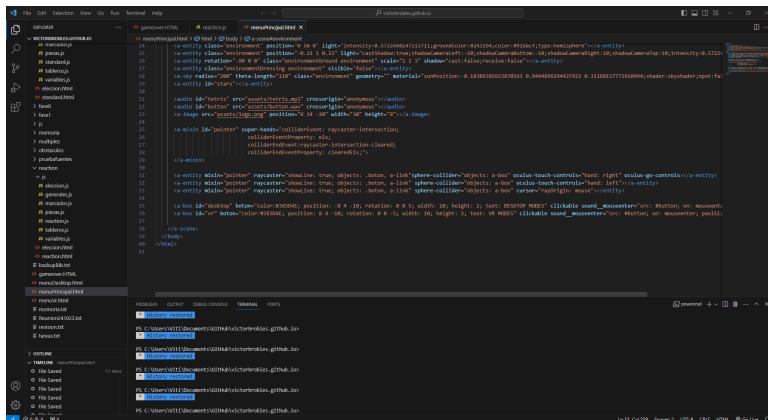


Atom [7] es un editor de código fuente para macOS, Linux y Windows desarrollado por GitHub. Es de código abierto, proporciona soporte para plug-in desarrollados en Node.js y presenta control de versiones Git integrado. La mayoría de los paquetes que utiliza tienen licencia de software libre y están desarrollados por la comunidad de usuarios. Puede ser utilizado como entorno de desarrollo integrado ya que es una aplicación que proporciona servicios para facilitar el desarrollo de software.

Es compatible con múltiples lenguajes de programación, así como C++, HTML, CSS, Java o JavaScript. También permite soporte para otros lenguajes mediante la instalación de los paquetes necesarios para ello.

Todos los documentos que componen este proyecto han sido creados y modificados mediante esta aplicación.

2.2.4. Visual Studio Code



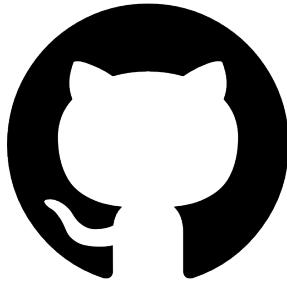
Visual Studio Code [28] es un editor de código fuente desarrollado por Microsoft y dispo-

nible para macOS, Linux y Windows. Incluye soporte para JavaScript, TypeScript y Node.js, pero también posee múltiples extensiones para otros lenguajes de programación. Es de código abierto e incluye características como el control integrado de Git, la refactorización de código o el soporte para depuración de código.

Una de las características que hacen especial a este editor es que posee una ventana de terminal integrada, que permite al usuario ejecutar comandos dentro de la propia aplicación. También posee una ventana de consola donde aparece el resultado de los comandos ejecutados sobre los documentos correspondientes.

En este proyecto se ha utilizado la función de desplegar un servidor web. Cuando el archivo abierto consiste en una aplicación web, esta puede ser desplegada en un servidor externo a través de Visual Studio Code, mediante una herramienta situada en la esquina inferior derecha llamada *Go Live*.

2.2.5. GitHub



GitHub [9] es un portal creado para alojar el código de las aplicaciones de cualquier desarrollador, comprado por Microsoft en 2018. Se trata de una de las principales plataformas para crear proyectos abiertos de herramientas y aplicaciones en la actualidad.

La aplicación se caracteriza por su carácter colaborativo, de forma que los desarrolladores suben el código de sus aplicaciones y cualquier usuario puede descargárselo o incluso colaborar con su desarrollo. Aún así también es posible crear repositorios privados, de forma que solo puedan acceder a él las personas autorizadas por su creador [29].

La aplicación está basada en el control de versiones que ofrece Git, de forma que los desarrolladores pueden administrar su proyecto, ordenando el código de cada una de las versiones de sus aplicaciones. Esto permite ver las diferencias entre una versión y otra, restaurar versiones anteriores del proyecto o fusionar el código de distintas versiones.

GitHub también ofrece una serie de herramientas propias para complementar las ventajas que ofrece Git. Una de ellas es la herramienta de *GitHub Pages*, que se encarga de desplegar un proyecto GitHub en una dirección web. Esto puede ser muy útil a la hora de desarrollar aplicaciones web, ya que con esta herramienta es posible ver la aplicación desplegada en Internet tras haberla subido al repositorio.

El proyecto está alojado en un repositorio GitHub y se encuentra desplegado en la web mediante la herramienta de GitHub Pages.

2.2.6. Git



Git [6] es un sistema de control de versiones de código abierto y mantenimiento activo que desarrolló Linus Torvalds, el desarrollador de Linux, hacia el 2005. Es el sistema más utilizado del mundo y presenta compatibilidad con una amplia variedad de sistemas operativos. Presenta una arquitectura distribuida, por lo tanto la copia del trabajo de cada desarrollador es a su vez un repositorio capaz de albergar todo el historial de cambios del proyecto.

Una de las principales características de Git es su flexibilidad. Es flexible tanto en la capacidad para desarrollar para varios flujos de trabajo de forma no lineal como en su eficiencia en proyectos grandes y pequeños. Está diseñado para posibilitar la ramificación de forma que las operaciones que afectan a las diferentes ramas de trabajo también quedan reflejadas en el historial de cambios [25].

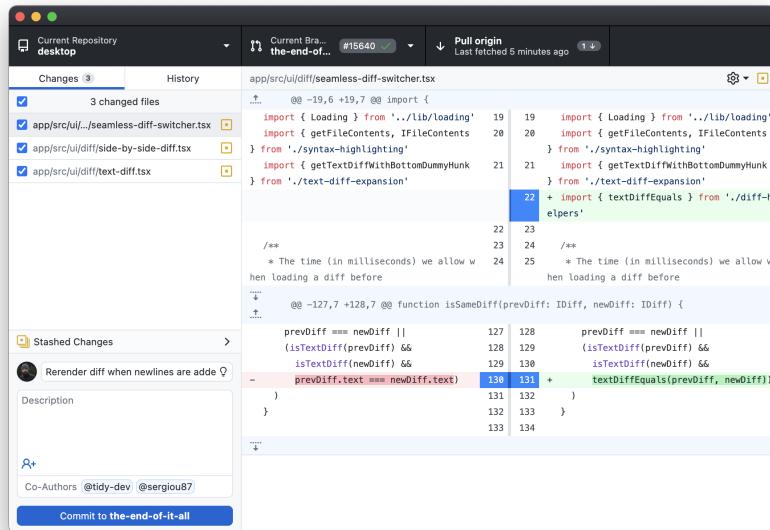
Git posee una arquitectura distribuida, no obstante también permite la existencia de un repositorio convencional donde se almacenen todos los cambios del sistema. De esta forma la disponibilidad del proyecto no se ve afectada por el rendimiento del servidor central, ya que el repositorio de cada desarrollador contiene todo el código, permitiendo que durante las ausencias de conexión los desarrolladores puedan seguir accediendo a él.

Es una herramienta creada esencialmente para el trabajo en equipo, ya que permite que cada desarrollador trabaje con su copia local del proyecto y más tarde suba su versión al repositorio central, asegurándose de no pisar las modificaciones realizadas por otro usuario. Además de

esto permite que cada desarrollador pueda sincronizar su versión en cualquier momento con la versión que otros desarrolladores hayan subido al repositorio central, lo que facilita el desarrollo no lineal.

Este lenguaje se emplea en el proyecto para sincronizar el resositorio de GitHub con los datos del proyecto en local.

2.2.7. GitHub Desktop



Github Desktop [15] es una aplicación que permite la interacción con GitHub usando un GUI en lugar de la línea de comandos o una interfaz web. Permite el uso de la mayoría de comandos git desde su interfaz gráfica con confirmación visual de los cambios.

Permite la sincronización de cuentas GitHub para tener acceso a todos los repositorios de la misma. Una vez sincronizada la cuenta permite al usuario elegir el repositorio de trabajo en el que se van a realizar los cambios y crea una copia local del mismo en el ordenador del usuario. Los cambios realizados sobre esa copia local pueden subirse al repositorio origen de GitHub desde la propia aplicación. También habilita al usuario a crear ramas de trabajo, sincronizarse con el repositorio origen o crear incidencias en GitHub.

Esta aplicación supone por lo tanto una gran alternativa a Git y es la utilizada en este caso, ya que incluye la parte fundamental de este lenguaje sin necesidad de instalarlo. Es muy útil en caso de proyectos pequeños que no requieran las funciones complejas que incorpora git y de

las que no dispone esta aplicación. Además la confirmación visual de los cambios permite a los usuarios más inexpertos tener la certeza de que los cambios se han aplicado correctamente.

2.2.8. LaTeX



LaTeX [26] es un sistema de composición de textos orientado a la creación de documentos escritos con alta calidad tipográfica. Fue creado en 1984 y utiliza una gran cantidad de macros de TeX. Se considera una herramienta ideal para la composición de artículos o textos científicos debido a que permite la inclusión de funciones o ecuaciones matemáticas.

Está compuesto por multitud de comandos TeX, un lenguaje de bajo nivel ya que sus acciones son muy elementales. Esto es lo que convierte a Latex en una herramienta esencial a la hora de crear documentos técnicos o científicos, ya que sus comandos permiten introducir todo tipo de elementos en los textos escritos. Al ser un software de libre licencia muchos usuarios fueron incorporando nuevas utilidades al lenguaje y por este motivo se inició el Proyecto LaTeX3, que unificó las distintas versiones del lenguaje bajo una única.

Tras su estandarización, LaTeX incorporó también una arquitectura modular en la que un núcleo central mantiene las características de la versión actual y permite la adición de nuevas funcionalidades mediante el uso de paquetes que solo se cargan en caso de ser necesarios.

Este sistema permite al usuario centrarse exclusivamente en el contenido del documento, en contrapunto a la mayoría de procesadores de texto. Presenta unas potentes capacidades gráficas que permiten estructurar el documento fácilmente y hacen que el usuario no tenga que preocuparse por el formato del mismo. El proceso de creación de un documento en Latex consta de dos etapas principales: una en la que se genera el texto del documento mediante cualquier editor de texto plano, y otra en la que el procesador compila el documento dejándolo listo para ser enviado a la salida correspondiente.

Este sistema se ha utilizado en este proyecto para desarrollar esta memoria, combinando el lenguaje LaTeX con una herramienta llamada *Overleaf*. Esta herramienta de publicación en

Línea proporciona un editor LaTeX fácil de usar y permite agilizar el proceso de redacción y publicación del documento ya que la salida se produce automáticamente en segundo plano a medida que el usuario escribe.

Capítulo 3

Descripción Técnica

En esta parte de la memoria se procede a detallar la arquitectura del proyecto y las condiciones de uso de la aplicación. En la primera sección se explica el funcionamiento de cada uno de los modos de juego, mientras que en la segunda parte se detallan todos los componentes utilizados en el desarrollo así como sus características y funciones.

3.1. Manual de Usuario

En esta sección se explican las condiciones de uso de la aplicación, es decir, las acciones que debe realizar un usuario para poner en funcionamiento el juego. Con el fin de buscar la mejor experiencia de usuario posible se analiza a continuación cada una de las pantallas por las que está compuesta la aplicación.

Cabe destacar que para poner en funcionamiento la aplicación el primer paso debe ser acceder a la URL ¹ en la que está desplegada la pantalla principal.

3.1.1. Menú Principal

Una vez el usuario se encuentra en el menú principal de la aplicación, se muestra una pantalla con el logo y dos botones en movimiento. Para acceder a la siguiente pantalla el usuario debe hacer click en uno de esos botones, que se encargan de realizar la redirección posterior. Si se quiere acceder al menú de los modos de juego de escritorio debe pulsar el botón **Desktop**

¹<https://victorbrobles.github.io/menuPrincipal.html>

Modes; si se quiere acceder al menú de los modos de realidad virtual debe pulsar el botón **VR Modes.**



Figura 3.1: Menú Principal de la aplicación

3.1.2. Menús Secundarios

En este punto el usuario puede encontrarse en dos pantallas diferentes, el menú de los modos de escritorio o el de los modos de realidad virtual. Ambas pantallas presentan la misma estructura y funcionan de la misma forma, por lo que el manual de una de ellas sirve de igual manera para la otra.

El usuario se encuentra una pantalla con el logo de la aplicación, unos botones en movimiento correspondientes a los modos de juego disponibles y un botón de retroceso situado en la esquina superior izquierda. Si el usuario quiere acceder a alguno de los modos de juego de la aplicación debe pulsar el botón que contiene el título del modo deseado, en cuyo caso se realiza la redirección a la siguiente pantalla. Si por el contrario el usuario quiere volver al menú principal debe pulsar el botón **Return** que le enviará de nuevo a esa pantalla.



Figura 3.2: Menú Secundario de los modos de escritorio



Figura 3.3: Menú Secundario de los modos de realidad virtual

3.1.3. Menús de Elección

En caso de que el usuario elija un modo de juego distinto al modo **Multiboard** 3.1.7 se le envía a una pantalla intermedia en la que debe elegir una característica inicial del modo. Si el modo seleccionado es **Obstáculos** 3.1.6 la pantalla sirve para elegir el número de obstáculos presentes en el juego, si por el contrario el modo seleccionado es cualquier otro la pantalla sirve para definir la anchura del tablero de juego.

Ambas pantallas presentan la misma estructura, mostrando el logo de la aplicación, varios botones que contienen las distintas opciones de elección y un botón de retroceso. Si el usuario quiere acceder al modo de juego previamente seleccionado debe pulsar sobre una de las opciones de elección para que se realice la redirección. Si por el contrario pulsa el botón **Return** se realizará una redirección a la página anterior.



Figura 3.4: Menú de Elección de uno de los modos de juego de la aplicación

3.1.4. Modo Standard

El modo **Standard** comienza con un escenario formado por el tablero, dos botones situados a ambos lados del mismo y un controlador situado en la parte inferior. Una vez que se carga el

escenario comienza a caer una pieza de forma automática sobre el tablero, dando lugar al inicio del juego.

El objetivo del juego es ir apilando piezas sobre el tablero e ir completando filas para que estas se eliminen con el fin de que el tablero nunca se rellene por completo. Para ello cada vez que una pieza caiga sobre el tablero o sobre otra pieza, se genera automáticamente otra pieza en la parte superior que va cayendo para completar el ciclo, y así sucesivamente.

El usuario puede interactuar con el escenario modificando las propiedades únicamente de la pieza que se encuentre en caída libre en ese momento, de tres maneras diferentes: modificando la posición en el eje x de la pieza, haciendo que caiga más rápido o rotando la pieza 90°.

Para modificar la posición de la pieza es necesario que el usuario mueva la bola del controlador hacia la posición que va a desplazarse la pieza. Para ello debe hacer click sobre la bola y mantener pulsado el botón mientras que desliza el ratón o el controlador virtual hacia la posición deseada. Una vez el usuario deja de mantener pulsado dicho botón, la pieza queda fija en la posición horizontal en la que finaliza la acción, aunque esta sigue bajando en dirección vertical.

Para rotar una pieza o hacer que baje más rápido simplemente el usuario debe hacer click sobre uno de los botones que se encuentran a los lados del tablero. En este caso no es necesario mantener pulsado dicho botón, ya que la acción es única y si se quiere realizar varias veces deberán ejecutarse varios clicks. Para rotar una pieza debe pulsarse el botón *Rotar pieza* y para hacer que baje más rápido debe pulsarse el botón *Bajar pieza*.

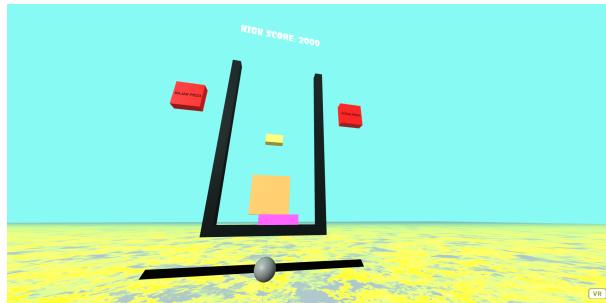


Figura 3.5: Modo Estándar de la aplicación

3.1.5. Modo Colors

El modo **Colors** conserva la misma estructura que el modo *Standard*, por lo que el escenario inicial es exactamente igual. También se mantienen las tres acciones que el usuario puede rea-

lizar sobre el escenario, siguiendo el mismo funcionamiento. Por último, el objetivo del juego también es el mismo, salvo por una peculiaridad que hace que ambos modos no sean exactamente iguales.

Esta peculiaridad consiste en una nueva regla que hace el juego un poco más difícil y puede suponer un reto para el usuario. Mientras que en el modo *Standard 3.1.4* se crean piezas de un color totalmente aleatorio, en este las piezas solo pueden ser de cuatro colores diferentes, conservando la aleatoriedad en su creación. La nueva norma consiste en que una pieza de un color no puede apilarse sobre otra pieza de un color diferente. Por ejemplo, partiendo de un escenario en el que hay sobre el tablero una pieza azul y otra roja, en el momento que cae otra pieza roja esta solo puede caer sobre la segunda.

En el momento que se incumpla esta norma el juego habrá terminado y finalizará con la puntuación que haya conseguido el usuario hasta ese preciso momento.

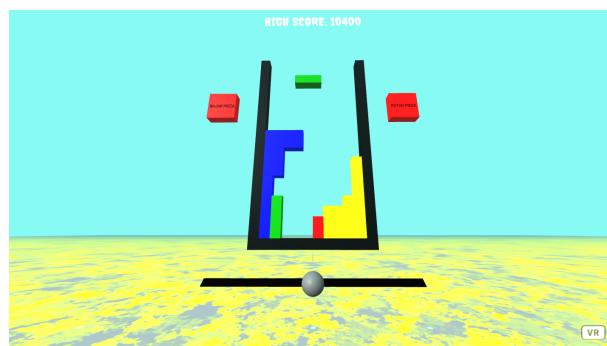


Figura 3.6: Modo Colores de la aplicación

3.1.6. Modo Obstacles

El modo **Obstacles** también presenta una estructura inicial muy parecida a los dos modos de juego anteriores. La principal diferencia respecto a ellos es que el tablero es mucho más ancho en este modo y también aparecen una serie de piezas que no estaban presentes en los anteriores. Estas nuevas piezas son los llamados obstáculos y todos ellos presentan las mismas propiedades: son cubos de color blanco y tamaño 1x1 que se encuentran distribuidos de forma totalmente aleatoria sobre la superficie del tablero. El número de obstáculos presentes en el escenario es elegido por el usuario antes de iniciar el juego.

El objetivo de este modo de juego es el mismo que el del modo *Standard 3.1.4* pero con

una nueva restricción que consiste en que ninguna pieza debe colisionar con ninguno de los obstáculos presentes en la escena. El usuario debe ir moviendo las piezas según van cayendo para evitar que estas choquen con alguno de estos obstáculos. Las piezas se mueven hacia los lados de igual manera que en los modos anteriores, y el usuario también puede hacer que caigan más rápido o que rotén.

Otra característica nueva que introduce este modo de juego es que las piezas van cayendo sobre el tablero con una velocidad superior a las anteriores, aportando algo más de dificultad al desarrollo del mismo.

En el momento que se incumpla esta norma el juego habrá terminado y finalizará con la puntuación que haya conseguido el usuario hasta ese preciso momento.

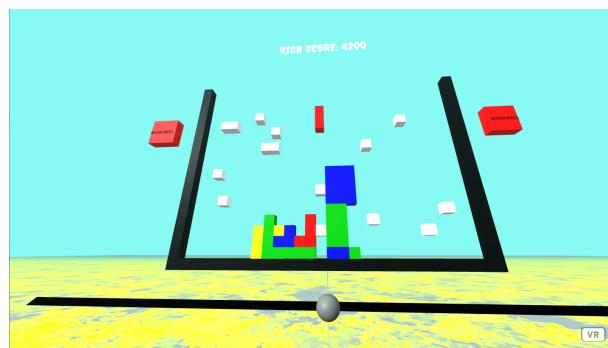


Figura 3.7: Modo Obstáculos de la aplicación

3.1.7. Modo Multiboard

El modo **Multiboard** conserva la misma estructura inicial que el modo *Standard* 3.1.4 con la adición de dos nuevos botones que permiten la funcionalidad propia de este modo. Estos dos botones se sitúan a ambos lados del tablero y presentan el siguiente texto: + Tableros. Se utilizan para crear tableros adicionales a ambos lados del tablero principal una vez que el usuario interactue con ellos. Para ello el usuario deberá hacer click en el botón situado a la izquierda del tablero cuando quiera generar un nuevo tablero a la izquierda del principal; si por el contrario desea crear un tablero a la derecha del principal, debe hacer click sobre el botón situado a la derecha.

El objetivo del juego es el mismo que el del modo *Standard*, con la peculiaridad de que en caso de haberse creado tableros adicionales la acción se desarrollará en varios tableros a la vez,

en lugar de desarrollarse solo en el principal. El usuario debe ser capaz por lo tanto de controlar el juego sobre tres escenarios simultáneamente, ya que las acciones que se realizan sobre uno de ellos no repercuten en los demás.

Las acciones que el usuario puede realizar sobre las piezas son exactamente las mismas que en los modos anteriores: mover horizontalmente las piezas, rotarlas o hacer que caigan más rápido. Lo que hace diferente a este modo de los demás es que al crear un nuevo tablero, este se crea junto con su propio controlador y sus propios botones, de forma que pueden llegar a existir hasta tres controladores y tres botones de cada tipo sobre el mismo escenario. Esto provoca que en caso de que el usuario quiera realizar alguna acción sobre una pieza del tablero derecho, deberá interactuar con el controlador o los botones asociados al tablero derecho, y se procederá de igual manera con el tablero central o el izquierdo.

Existen dos características que sí se ven afectadas por las acciones que se produzcan sobre cualquiera de los tres tableros: la puntuación y el fin de juego. La puntuación es conjunta en este modo, por lo que solo existe una única puntuación que va aumentando por las acciones que tengan lugar en cualquiera de los tableros presentes en la escena. Por otra parte el juego llega a su fin en cuanto se excede el límite superior de uno de ellos, por lo tanto el usuario no debe permitir que esto ocurra en uno de los tableros, ya que a pesar de que los otros se mantengan correctamente, el juego terminará igualmente.

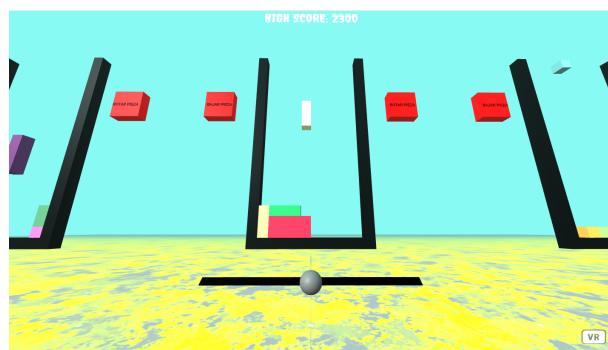


Figura 3.8: Modo multitablero de la aplicación

3.1.8. Modo 180 Degrees

El modo **180 Degrees** mantiene la estructura inicial básica del modo *Standard 3.1.4*, con un tablero acompañado del controlador y los botones para controlar las piezas. En este caso el es-

scenario cuenta también con un tablero exactamente igual acompañado de los mismos elementos, todos ellos situados en el lado contrario del plano z. Por lo tanto el escenario inicial consta de una escena principal situada al frente del usuario, y la misma escena repetida a su espalda.

El objetivo del juego es el mismo que en el resto de modos, conseguir que las piezas no rebasen el límite superior del tablero, pero en este caso el usuario debe controlar el juego en los dos tableros existentes. La peculiaridad de este modo es que la acción no se desarrolla en ambos tableros de forma simultánea, si no que cuando la acción se inicia en un tablero se frena en el opuesto, y viceversa.

Una vez que se inicia el juego las piezas comienzan a caer sobre el tablero principal situado en frente del usuario, mientras que sobre el tablero opuesto no ocurre nada. Sobre este tablero comienzan a caer piezas hasta llegar a un número aleatorio entre 1 y 5. En este momento dejan de caer piezas sobre el tablero y aparece en pantalla un mensaje a ambos lados del tablero que anima al usuario a girar 180 grados. Al mismo tiempo comienzan a caer piezas en el tablero opuesto hasta llegar a un nuevo número aleatorio entre 1 y 5, momento en el cual aparecerá el mismo mensaje anterior a ambos lados del tablero opuesto y el usuario deberá volver a girar sobre sí mismo, y así sucesivamente hasta el fin del juego.

Las acciones que el usuario puede realizar sobre las piezas son exactamente las mismas que en el modo estándar: mover las piezas horizontalmente, hacer que bajen más rápido o rotarlas. En este modo el usuario también debe interactuar con el juego girando sobre sí mismo cada vez que salga el mensaje por pantalla, para ir cambiando de un tablero a otro.

El juego finaliza en el momento que las piezas rebasen el límite superior de cualquiera de los dos tableros, por lo que el usuario debe ser capaz de controlar las acciones que se produzcan sobre ambos tableros, aunque estas no se produzcan de manera simultánea.



Figura 3.9: Modo 180 grados de la aplicación

3.1.9. Modo Dropping pieces

El modo **Dropping pieces** comienza con un escenario inicial algo diferente al resto de modos. Consta de un tablero principal al igual que los demás, pero en este caso no aparece el controlador de las piezas ni los botones a los lados del tablero. En su lugar aparece un nuevo botón situado en el límite inferior del tablero y tres piezas aleatorias flotando a cada lado del tablero.

En este modo de juego la acción no comienza con piezas cayendo sobre el tablero de forma automática, si no que las piezas que el usuario debe colocar en el tablero ya se encuentran flotando en la escena. La acción comienza cuando el usuario hace click sobre alguna de las piezas y mantiene pulsado el botón arrastrando esa pieza hasta la posición en la que quiere que esta caiga. Para que una pieza caiga sobre el tablero es necesario que el usuario deje de pulsar el botón cuando la pieza se encuentre en el rango horizontal que define la anchura del tablero y por encima del límite superior del mismo. En ese caso la pieza caerá sobre la posición en la que estuviera en ese momento, en caso contrario la pieza volverá a su posición inicial, flotando a uno de los lados del tablero.

Este juego se divide en rondas de seis piezas cada una, por lo que cuando una pieza es colocada sobre el tablero no se sustituye por otra nueva, si no que su lugar queda vacío. Esto sucede así hasta que se colocan las seis piezas sobre el tablero, momento en el que se generan otras seis piezas totalmente aleatorias para poder continuar el juego, y así sucesivamente.

El usuario puede realizar dos tipos de acciones sobre las piezas en este modo: desplazarlas y rotarlas. Para desplazarlas el usuario debe hacer click sobre la pieza deseada y no soltar el botón hasta que la pieza esté situada donde el usuario quiera que caiga. La rotación de las piezas en este caso es diferente a los demás, ya que en este modo no se pueden rotar las piezas mientras están cayendo sobre el tablero de modo que en el momento en que una pieza comienza a caer no se puede realizar ninguna acción sobre ella. Para rotar las piezas el usuario debe hacer click sobre el botón de *Rotar piezas* situado en el límite inferior del tablero. Esta acción provocará la rotación de todas las piezas que en ese momento estén flotando en la escena, por lo que la rotación no se realiza sobre una única pieza, si no sobre todas las que existan en ese momento pendientes de colocar sobre el tablero.

El juego finaliza de la misma forma que en el resto de modos, una vez es superado el límite superior del tablero al colocar las piezas.



Figura 3.10: Modo piezas flotantes de la aplicación

3.1.10. Modo Reaction test

El modo **Reaction test** comienza con el escenario inicial más simple de cualquiera de los modos de la aplicación. Consta únicamente del tablero principal sin ningún tipo de controlador o botón a su alrededor.

La acción comienza en cuanto se genera una pieza aleatoria de forma automática sobre el tablero, en una posición horizontal aleatoria dentro del rango de la anchura del tablero. En este momento el usuario tiene 3 segundos para decidir si quiere que la pieza caiga sobre la posición horizontal en la que se encuentra o si en caso contrario quiere que desaparezca. Si en esos 3 segundos el usuario hace click sobre la pieza en sí, esta caerá automáticamente hacia abajo en la posición en la que se encuentra; si no lo hace la pieza desaparecerá de la escena. En ambos casos se genera una nueva pieza de forma automática una vez hayan pasado los tres segundos o se haya hecho el click, para continuar el juego hasta que este finalice.

Existe una penalización para el usuario en caso de que este deje desaparecer las piezas. Consiste en la resta de 1000 puntos cada vez que el usuario deje que desaparezca una pieza y una nueva norma por la cual el juego termina cuando el usuario deja pasar 10 o más piezas. Esto añade una dificultad extra al juego por la cual el usuario tiene que elegir muy bien qué piezas deben desaparecer y en un periodo muy corto de tiempo.

La única acción que puede realizar el usuario sobre el escenario en este caso es la de hacer que una pieza caiga o desaparezca. Para ello es necesario que el usuario haga click sobre la propia pieza una vez que esta se encuentra inmóvil sobre el tablero. Una vez la pieza comience a caer no se podrá realizar ninguna acción sobre ella, por lo que la interacción del usuario con la escena se reduce simplemente a esa acción.

El juego finaliza una vez que las piezas rebasan el límite superior del tablero o el usuario deja que desaparezcan 10 piezas sin pulsar sobre ellas.



Figura 3.11: Modo test de reflejos de la aplicación

3.1.11. Menú fin de juego

El menú de fin de juego consta de una pantalla con un fondo de color negro y con ciertos elementos sobre ella. El primer elemento es un texto que muestra que el juego ha terminado, mientras que el segundo es otro texto que muestra la puntuación final que ha conseguido el usuario en el último modo al que ha jugado. El último elemento es un botón que permite al usuario volver al menú principal de la aplicación. Para ello el usuario debe hacer click en dicho botón, en cuyo caso se realizará una redirección hacia la página del menú principal.



Figura 3.12: Menú de fin de juego de la aplicación

3.2. Componentes principales

A lo largo de esta sección se explican los distintos componentes que han sido utilizados a lo largo de este desarrollo en todas las escenas de la aplicación. En primer lugar es necesario especificar que un componente es un tipo de estructura que presenta A-Frame y sirve para dotar a una entidad de cierta apariencia o funcionalidad.

Para poder registrar un componente en A-Frame es necesario seguir los siguientes pasos:

- En primer lugar se declara un nuevo elemento en el código HTML.
- Cada elemento se define con una propiedad cuyo nombre concuerda con la funcionalidad que va a tener ese elemento en la escena, de forma que cada pieza que cae sobre el tablero viene acompañada de la propiedad *cubo*, por ejemplo.
- En el código JavaScript se registran todos estas propiedades para crear los componentes de la escena.
- En la función *init* se accede al contenido de la propiedad y de esta forma se establecen las propiedades iniciales de cada uno de los componentes.
- En caso de tratarse de un componente dinámico, se crea la función *tick* y se introduce en ella toda la funcionalidad dinámica que presentará el componente durante el desarrollo del juego.

```
<a-plane id="botonBack" botonback = "color:black; position: -15 9 -10; rotation: 0 0 0; width: 4; height: 1; text: RETURN"></a-plane>
```

Figura 3.13: Ejemplo de registro de un nuevo componente (Parte HTML)

```
AFRAME.registerComponent('botonback', {
  schema: {
    position: {default: "0 0 0"}, 
    rotation: {default: "0 0 0"}, 
    width: {default: "1"}, 
    height: {default: "1"}, 
    color: {default: "white"}, 
    text: {default: ""}
  },
})
```

Figura 3.14: Ejemplo de registro de un nuevo componente (Parte JavaScript)

Cuando un componente se registra en A-Frame presenta ciertas propiedades que puede recibir en la declaración del elemento que lo contiene o pueden definirse más adelante. A parte de

propiedades, un componente A-Frame presenta ciertas funciones propias que sirven para dotar a la entidad de las propiedades que recibe o genera, y para otorgarle una funcionalidad específica al componente. Las principales propiedades que presentan los componentes son las siguientes:

- La función *init* sirve principalmente para establecer las propiedades iniciales del elemento, que deben pasarse dentro de la propiedad registrada. Para ello se define en primer lugar un método llamado *schema* que recoge cada una de las propiedades que se han pasado al elemento. Dentro de la función *init* se declara una variable que accede al propio elemento (**this.el**) y otra variable que contiene todas las propiedades que se han recogido mediante el método *schema* (**this.data**). Una vez se tienen ambas variables se procede a ejecutar la función *setAttribute()* sobre el elemento para cada una de las propiedades que se quieren establecer.
- La función *update* se utiliza para manejar un evento que va a producirse una única vez sobre el elemento. Cuando ese evento se produzca, la función se dispara y se ejecuta el código que contenga. Si ese evento vuelve a producirse más tarde, esta función no volverá a lanzarse, ya que solo salta una vez por cada elemento.
- La función *tick* se emplea para alterar continuamente las propiedades de un elemento. Esta función se dispara automáticamente cada pequeña fracción de segundo y no deja de lanzarse hasta que el componente se elimine de la escena. Se puede utilizar para alterar las propiedades del componente en cada paso por ella o para acceder a las propiedades del elemento y modificar alguna de ellas en función del valor de las mismas. También se puede utilizar como manejador de algún evento que pueda ocurrir sobre el elemento en múltiples ocasiones, permitiendo así que el elemento sea capaz de manejarlo cada vez que este se produzca, a diferencia de la función *update*.

```
AFRAME.registerComponent('botonback', {
  schema: {
    position: {default: "0 0 0"}, 
  },
  init: function() {
  },
  update: function () {
  },
  tick: function() {
  }
});
```

Figura 3.15: Declaración de funciones de un componente

```

AFRAME.registerComponent('controller', {
  schema: {
    position: {default: "0 0 0"},  

    radius: {default: 0},  

    rotation: {default: "0 0 0"},  

    mixin: {default: ""},  

    color: {default: "white"},  

    id: {default: ""}
  },
  init: function() {
    var el = this.el;
    var data = this.data;

    el.setAttribute('position', data.position);
    el.setAttribute('color', data.color);
    el.setAttribute('radius', data.radius);
    el.setAttribute('rotation', data.rotation);
    el.setAttribute('mixin', data.mixin);
    el.id = data.id;
  },
  tick: function() {
    var el = this.el;
    var data = this.data;

    var pieza = document.getElementById("cubo" + contadorPieza);
    var position = el.getAttribute('position');
    var positionPieza = pieza.getAttribute('position');

    if (positionPieza.y == alturaTablero + 5) {
      var positionAux = {x: 0, y: position.y, z: position.z};
      el.setAttribute('position', positionAux);
    }

    if (!pieza.components.cubo.tocaSuelo) {
      el.addEventListener('grab-end', function(event) {
        moverControlador(controller, data, el);
      });
    }
  }
});

```

Figura 3.16: Implementación completa de un componente

A lo largo del desarrollo de esta práctica se han utilizado múltiples componentes A-Frame que forman las distintas escenas de la aplicación y para cada uno de ellos se detalla su estructura y funcionamiento a continuación:

- **boton.** El componente *boton* es un elemento utilizado en las pantallas de menú que sirve para crear los botones con las elecciones de los modos de juego. Es un *a-box* [6](#) que presenta una función init y una función tick. En la función init se encarga de dotar al elemento de las propiedades que recibe y también le otorga un texto correspondiente a la propiedad *id* que recibe. En la función tick el componente espera que se produzca el evento *grab-end*, y cuando este se produce extrae el id del botón que ha propagado este evento y a partir de ese id se encarga de redirigir a una página u otra de la aplicación.
- **botonback.** El componente *botonback* también se utiliza únicamente en los menús de la aplicación y sirve para volver a la página anterior en caso de necesitarlo. Al igual que el anterior componente se trata de un elemento *a-box* [6](#) que recibe unas propiedades y se las otorga al elemento en la función init (a parte de añadirle un texto también) y que espera que se produzca el evento *grab-end* en su función tick. Una vez que este evento se produce, se realiza una redirección a la página anterior de la aplicación.
- **tablero.** El componente *tablero* es el componente principal de todas las escenas de la

aplicación, ya que crea la estructura básica sobre la que se desarrollan todos y cada uno de los modos de juego. Es un elemento *a-entity* **6** que recibe unas propiedades relativas a las dimensiones de la estructura que va a generar. En la función init se encarga de generar dos elementos *a-box* **6** en estructura vertical (las paredes del tablero) y otro elemento *a-box* **6** en estructura horizontal (el suelo del tablero) que une a los anteriores formando la estructura completa. Una vez creados los tres nuevos elementos, los incorpora a la escena como hijos suyos. En la función tick se encarga de revisar el estado de una variable y, dependiendo de ese estado, crear un nuevo componente *cubo* o no.

- **rotarpieza.** El componente *rotarpieza* se encarga de crear un botón presente en casi todas las escenas de la aplicación que sirve para rotar las piezas según van cayendo sobre el tablero. Este componente le otorga al elemento las propiedades que recibe en la función init y le añade un texto al botón. En la función tick espera que se produzca el evento *grab-end* para cambiar el valor de la variable *rotarPieza* a true.
- **bajarpieza.** El componente *bajarpieza* se encarga de crear un botón presente en casi todas las escenas de la aplicación que sirve para bajar las piezas de manera más rápida según van cayendo sobre el tablero. Este componente le otorga al elemento las propiedades que recibe en la función init y le añade un texto al botón. En la función tick espera que se produzca el evento *grab-end* para cambiar el valor de la variable *bajarPieza* a true.
- **score.** Este componente se utiliza para crear una estructura en la parte superior de todos los modos de juego que avisa al usuario de la puntuación que lleva en cada momento. Este componente crea un elemento *a-entity* **6** en su función init con un texto que indica que la puntuación es igual a 0, y lo añade a la escena como hijo suyo. En la función tick revisa el estado de ciertas variables y en caso positivo actualiza el valor del texto al valor que tenga en ese momento una variable que guarda la puntuación de forma continua.
- **mando.** Este componente es muy simple ya que solo crea la estructura sobre la que se aloja el componente *controller* en la escena. Solo presenta una función init en la que añade las propiedades que recibe al elemento *a-plane* **6** que lo contiene.
- **controller.** Este componente se utiliza en la mayoría de los modos de juego para controlar el movimiento de las piezas que caen sobre el tablero. En la función init otorga al elemento

a-sphere 6 que lo contiene las propiedades que recibe de él. En la función tick revisa la posición en la que se encuentra la pieza que está cayendo en ese momento y si esa pieza puede moverse modifica el valor de la variable *moverPieza* a true. Adicionalmente espera que se produzca sobre él el evento *grab-end* para cambiar también su propia posición.

- **cubo.** El componente *cubo* es un componente creado por el componente *tablero* y supone cada una de las piezas que caen sobre él en el desarrollo del juego. Este componente establece la estructura y posición inicial del *a-box* 6 que lo contiene en su función init. En la función tick se encarga de revisar su propia posición y en función de su estado realizar múltiples acciones tales como mover la pieza, bajarla, rotarla, actualizar el tablero o finalizar el juego.
- **obstaculo.** Este es un componente relativo al modo de juego *Obstacles* 3.1.6 y se utiliza para crear cada uno de los obstáculos que se presentan en la escena. Solo presenta una función init que dota a un *a-box* 6 de las propiedades que recibe.
- **botonestableros.** Este es un componente exclusivo del modo de juego *MultiBoard* 3.1.7 y se emplea para crear los botones utilizados para crear tableros adicionales. En su función init crea dos elementos *a-box* 6 que se corresponden con los botones a izquierda y derecha del tablero principal y les añade un texto a cada uno. Incorpora estos nuevos elementos a la escena como sus hijos. En la función tick revisa el estado de ciertas variables y en caso positivo, se encarga de activar la función que crea el tablero adicional a la derecha o la que crea el tablero a la izquierda.
- **handlernewapieza, handlerfilaeliminada, handlernuevotablero, handlerpiezasaludada.** Estos componentes se utilizan únicamente para reproducir sonidos dentro de la escena. Todos ellos presentan una única función tick que revisa el estado de una variable y en caso positivo activan un sonido que se reproduce inmediatamente en la escena.

Capítulo 4

Proceso y Desarrollo

Esta es la parte principal de la memoria puesto que se explicará todo el desarrollo del proyecto completo. Este análisis está dividido en las distintas fases principales por la que ha pasado el proyecto, detallando así tanto las dificultades como los avances de cada una de ellas.

4.1. Modelo de desarrollo

El desarrollo de este proyecto no ha seguido estrictamente ningún tipo de metodología concreta pero se ha apoyado en las bases que presenta la metodología *SCRUM*. Este sistema de gestión de proyectos presenta una serie de reglas y prácticas que permiten agilizar el desarrollo de un trabajo. Se basa en la división del proyecto en etapas o sprints que persiguen un objetivo concreto del proyecto. Hasta que no se consigue el objetivo de un sprint no comienza el otro, y la consecución de los objetivos de todos los sprints coincide con la consecución del objetivo final del proyecto. Estos sprints a su vez se dividen en tareas mucho más pequeñas que persiguen objetivos más concretos y ayudan a la consecución del objetivo final del sprint.

Las etapas o sprints en las que se ha dividido este proyecto son las siguientes:

- **Etapa 0.** Supone la fase previa al desarrollo del proyecto en sí y tiene como objetivo la familiarización del desarrollador con las tecnologías que va a utilizar en el desarrollo del proyecto.
- **Etapa 1.** En esta etapa se decide el objetivo principal del proyecto y se comienza a desarrollar el primer prototipo. El objetivo de esta fase es generar un escenario que sirva de

base para la aplicación y todos los modelos que se desarrollarán más adelante.

- **Etapa 2.** Esta etapa incorpora al proyecto la funcionalidad principal de la aplicación. El objetivo es transformar un prototipo básico en una aplicación en la que los componentes sean capaces de interaccionar entre sí.
- **Etapa 3.** Consiste en la integración de la aplicación con las tecnologías externas que se han utilizado. En este caso, el objetivo es conectar el proyecto con los dispositivos de realidad virtual empleados.
- **Etapa 4.** La parte central de este desarrollo se centra en la creación de diferentes escenarios que incorporen diferentes funcionalidades. Esta fase trata sobre el desarrollo de los modos de juego adicionales y se centra en la versión de escritorio.
- **Etapa 5.** Esta etapa es similar a la anterior, con la diferencia de que esta vez está centrada en los modos de juego para la versión de realidad virtual.
- **Etapa 6.** Una vez desarrollada la funcionalidad básica de la aplicación, es indispensable conectar las diferentes partes de la misma. Esta etapa se centra en la creación de pantallas que sirvan como enlace o conexión de los diferentes modos que presenta la aplicación.
- **Etapa 7.** La etapa final tiene como objetivo la adición de elementos extra que no modifiquen la funcionalidad de la aplicación pero mejoren las prestaciones de la misma. En este caso se añade tanto imágenes como sonido al proyecto para mejorar la experiencia de usuario.

4.2. Etapa 0

La etapa 0 corresponde con la primera fase del proyecto, aquella en la cual el desarrollador comienza a familiarizarse con las tecnologías que se van a utilizar durante la realización del mismo.

4.2.1. Objetivos de Diseño

El objetivo principal de esta fase es la adquisición de los conceptos necesarios sobre A-Frame para poder desarrollar una aplicación web funcional.

4.2.2. Detalles de Implementación

En primer lugar es necesario entender qué es A-Frame, para qué puede ser utilizado y qué se puede conseguir utilizando esta tecnología. Para ello se recurre a la página principal de A-Frame ¹, donde se presentan una serie de ejemplos sencillos que muestran algunas de las capacidades que posee esta tecnología.

Una vez ha sido entendida la funcionalidad de A-Frame es necesario entender su arquitectura. La documentación de la página principal ², es bastante útil y en este caso sirvió de gran ayuda.

La primera impresión que se obtuvo sobre A-Frame es que era un lenguaje muy similar a HTML, ya que se compone de etiquetas, aunque estas tienen una mayor funcionalidad que las etiquetas propias de HTML. Una de las principales características propias que poseen los elementos A-Frame es el soporte para aplicaciones en realidad virtual, el principal objetivo de este proyecto.

En el momento en que estos conocimientos fueron asentados, era hora de comenzar a desarrollar una pequeña aplicación para ponerlos a prueba. La aplicación que se decidió desarrollar fue una especie de tablero de billar sobre el que pudieran crearse bolas que se movieran sobre él e interactuaran con las paredes. El desarrollo de esta aplicación se llevó a cabo en dos dimensiones y en versión de escritorio.

¹<https://aframe.io/>

²<https://aframe.io/docs/1.4.0/introduction/>

El primer paso para la creación de la aplicación es crear un documento html que contenga un elemento **a-scene** [6](#). Este elemento se encarga de contener los demás componentes de la escena y puede tener una propiedad *environment* que dote de un cierto estilo a la escena.



```
<!DOCTYPE html>
<html>

<head>
  <title>Rebotadores</title>
  <script src="https://aframe.io/releases/1.2.0/aframe.min.js"></script>
  <script src="rebotadores.js"></script>
</head>

<body>
  <a-scene main_component>
  </a-scene>
</body>

</html>
```

Figura 4.1: Declaración del componente principal

Una vez creada la escena fue registrada como un nuevo componente A-Frame para dotarla de las propiedades que requería. Para ello fue necesario otorgar al elemento a-scene una propiedad cualquiera, en este caso *main-component*, y registrar esta nueva propiedad en A-Frame utilizando JavaScript. Este componente ya podía acceder a las propiedades que se hubieran definido para él y a los métodos propios de los componentes A-Frame (como el método *init*). En este caso fue utilizado para generar los demás componentes que formaban parte de la escena, así como el tablero y sus paredes, un cursor para poder interactuar con los componentes o un botón para controlar el movimiento de las bolas. Para ello fue necesario que el componente principal fuera creando *hijos* y le diera a cada uno de ellos las propiedades que iban a tener después.



```
AFRAME.registerComponent('main_component', {
  init: function() {
    var scene = document.querySelector('a-scene');

    var entity = document.createElement('a-entity');
    entity.setAttribute('cursor', 'rayOriginMouse');
    scene.appendChild(entity);

    var plane = document.createElement('a-plane');
    plane.setAttribute('main-plane', 'position: 0 2 -10; color:#35082d; width:32; height:15');
    entity.appendChild(plane);

    for (let i=0; i>walls.length; i++) {
      var plane = document.createElement('a-plane');
      plane.setAttribute('wall', walls[i]);
      entity.appendChild(plane);
    }

    for (let i=0; i>holes.length; i++) {
      var plane = document.createElement('a-plane');
      plane.setAttribute('hole', holes[i]);
      entity.appendChild(plane);
    }

    var boton = document.createElement('a-box');
    boton.setAttribute('button','');
    entity.appendChild(boton);
  }
});
```

Figura 4.2: Registro del componente principal

El siguiente paso fue registrar el resto de componentes que se habían definido. Todos estos componentes heredaban sus propiedades de su componente creador (*main-component*), por lo que solo había que recoger esas propiedades y otorgárselas a cada uno de ellos. Sin embargo se

encontró uno de los principales problemas de A-Frame al intentar registrar dos componentes en la misma posición, en este caso las paredes y los hoyos del tablero o las bolas y la parte central del tablero. Cuando esto ocurre el componente que ha sido registrado después que el otro se superpone a él, haciendo que sólo se vea uno de ellos o se vea uno encima del otro.

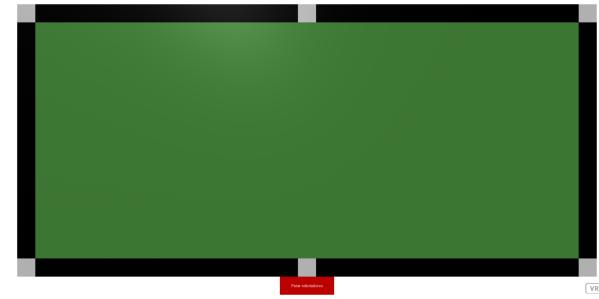


Figura 4.3: Tablero de los rebotadores

Otro problema que se encontró fue la ausencia de un componente botón en A-Frame que pudiera contener un texto y ejecutar una función al ser pulsado. Para solucionarlo se creó un componente **a-box** [6](#) que a su vez creara un componente **a-text** [6](#) que heredara de él, haciendo así que el texto del segundo componente se mostrara dentro de la superficie del primero.

Cuando la escena estuvo completada era hora de crear las bolas que iban a moverse sobre el tablero. Se empleó la función *tick* para manejar el evento *click* sobre la parte central del tablero. Cada vez que este componente recibiera un evento *click*, crearía un componente **a-circle** [6](#) con unas propiedades comunes entre todos ellos y un contador que los diferenciara del resto.

Una vez las bolas ya estaban registradas en la escena era necesario otorgarlas de movimiento, por lo que se utilizó la función *tick* para que cada bola estuviese en constante movimiento desde su creación. Para conseguir esto fue necesario acceder a la posición de la bola en cada vuelta del *tick*, crear una nueva parecida a ella y actualizar la posición de la bola a la recién creada.

En este punto surgió el problema de que la escena no tenía físicas, por lo que las bolas no eran capaces de interactuar con los demás componentes y al estar constantemente actualizando su posición, llegaban a desaparecer de la pantalla. Para solucionar este problema se crearon una serie de variables que establecían si las bolas debían cambiar de dirección o no. Esto se hizo utilizando la función *tick* sobre el componente central del tablero, que miraba la posición de cada una de las bolas y establecía si su posición coincidía con la pared superior, con la derecha,

con la izquierda o con la inferior. También se encargaba de cambiar el color de cada bola que cambiara de dirección a uno totalmente aleatorio. Para controlar el cambio de dirección, en la función *tick* sobre cada bola se comprobaba el valor de cada una de estas variables antes de actualizar la posición de la bola, y así se conseguía que la bola se moviera en la dirección adecuada.

Por último, se incorporó una función *update* al componente botón que esperaba que se produjera un evento *click* para recorrer la lista de todas las bolas y para cada una de ellas cambiarle el valor a una variable que haría que más tarde se detuvieran. Para ello en la función *tick* sobre cada bola se comprobaba el valor de esta variable y en caso de haber pulsado el botón, no se volvía a actualizar la posición de ninguna de ellas, provocando que se quedaran paradas sobre el tablero.

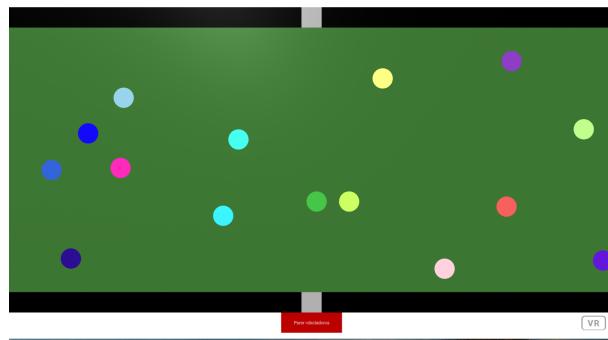


Figura 4.4: Aplicación en funcionamiento

4.2.3. Prototipo Final

Como resultado de este desarrollo se obtuvieron los conocimientos necesarios para comenzar a crear una aplicación web mediante esta herramienta. Entre las principales capacidades o conocimientos que se alcanzaron cabe destacar las siguientes:

- Crear una escena en A-Frame.
- Registrar un componente en A-Frame para acceder a sus funciones y las propiedades que se le han pasado.
- Generar un único componente principal que genere el resto de componentes al ser registrado en A-Frame.

- Acceder a las propiedades que se le pasan a un componente y dotarle de ellas.
- Manejar varios componentes en el mismo espacio y controlar la superposición de los mismos.
- Controlar los eventos que se producen sobre los distintos componentes.
- Hacer que un evento que se dispara sobre un componente sea capaz de alterar otro componente distinto.
- Controlar las interacciones entre varios componentes.
- Crear un cursor para poder interactuar con la aplicación mediante el uso del ratón.

Una vez se terminó el desarrollo, la aplicación fue desplegada a través de un repositorio de GitHub, utilizando la herramienta GitHub Pages. La aplicación se encuentra actualmente desplegada aquí [Aplicación Rebotadores](#), donde se puede probar toda su funcionalidad con la única ayuda de un ratón.

4.3. Etapa 1

La fase 1 corresponde con la etapa en la que se decide el propósito final del proyecto, el tipo de aplicación que va a desarrollarse empleando los conocimientos adquiridos previamente. Una vez tomada la decisión se comienza el desarrollo con la generación de un primer prototipo.

4.3.1. Objetivos de Diseño

El objetivo de esta etapa es el desarrollo de un prototipo que sirva como estructura común para todos los modos o pantallas de la aplicación.

4.3.2. Detalles de Implementación

En este punto se requería tomar la decisión de qué tipo de proyecto se iba a desarrollar una vez adquiridos los conocimientos básicos. Una de las ventajas que proporciona A-Frame es su variedad de recursos, que permite desarrollar todo tipo de aplicaciones web, desde aplicaciones con objetos del mundo real (objetos gltf) hasta aplicaciones centradas en las múltiples animaciones que pueden presentar los componentes A-Frame.

Finalmente la decisión tomada fue la de desarrollar un videojuego que permitiera su despliegue en realidad virtual. Para ello en lugar de generar un nuevo videojuego, se procedió a la adaptación de uno ya existente al entorno de la realidad virtual, en concreto el **Tetris**. Este juego está basado en bloques de cubos que caen sobre un tablero e interactúan entre sí, lo que lo hace idóneo para ser desarrollado mediante A-Frame.

Para añadir cierta originalidad y hacer la aplicación adaptable a una versión de escritorio y otra de realidad virtual se decidió crear diferentes modos de juego. Estos modos incorporarían funcionalidades que no posee el juego original pero conservarían la misma estructura inicial.

El primer paso del desarrollo consiste en la generación del prototipo o estructura que van a utilizar todos los modos de los que se compone la aplicación. La estructura se compone de los componentes necesarios que debería tener un **Tetris** para ser puesto a prueba. Estos elementos son los siguientes:

- Fondo o ecosistema donde se sitúa el juego.
- Tablero donde se desarrolla el juego.

- Controlador para mover dichas piezas.
- Botón para bajar piezas.
- Botón para rotar piezas.
- Cursor para ratón y VR.

Para desarrollar un fondo diferente al estándar que proporciona A-Frame es necesario añadir la propiedad *environment* al objeto **a-scene** que contiene todos los componentes. Crear un entorno desde cero es bastante complicado ya que existen muchas variables y cada una de ellas puede tomar infinidad de valores. Para ello A-Frame ofrece la herramienta *A-Frame Visual Inspector*, una interfaz que permite crear una escena desde cero e ir modificando todas sus propiedades al mismo tiempo que los cambios van siendo aplicados en la propia pantalla. Esto permite al usuario probar distintos tipos de fondos e ir eligiendo el que más se adapte a sus necesidades de forma dinámica. Cuando el desarrollo ha concluido, la interfaz permite al usuario copiar el elemento completo para pegarlo en el código y de esta manera tener la escena en su aplicación.

```
<a-scene id="entorno" environment="skyColor:#87faf4;horizonColor:#87faf4;dressingAmount:0;dressingOnPlayArea:0.5;dressingVariance:0.2 0.2 0.2;dressingScale:0.5;
groundColor:#7d787b;groundColor:#f1f518;dressingColor:#08fd18;grid:dots;dressingUniformScale:false;ground:flat;active:true;seed:17;lightPosition: -0.72 0.53 0.97;
fog:0.8;groundYScale:20;groundTexture:walkernoise;gridColor:#239893;preset:dream" preset-switcher="" sound="src: #environment; on: mouseenter; loop: true;
volume: 0.2; rollOffFactor: 0">
<a-entity class="environment" position="0 50 0" light="color:#3ccf9;intensity:0.6;groundColor:#7d787b;type:hemisphere"></a-entity>
<a-entity class="environment" position="-0.72 0.53 0.97" light="shadowCameraLeft: 10;shadowCameraBottom: -10;shadowCameraRight: 10;shadowCameraTop: 10;intensity:0.6"></a-entity>
<a-entity rotation="-90 0 0" class="environmentGround environment" scale="1 1 20" shadow="cast:false;receive:false"></a-entity>
<a-entity class="environmentDressing environment" visible="false"></a-entity>
<a-sky radius="200" theta-length="110" class="environment" geometry="" material="npot:false;color:#87faf4;fog:false"></a-sky>
<a-entity id="stars" visible="false"></a-entity>
```

Figura 4.5: Declaración del fondo en HTML

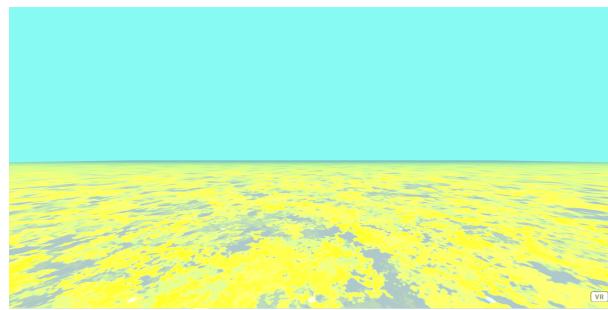


Figura 4.6: Fondo creado a través de A-Frame Visual Inspector

El siguiente paso es la creación del tablero en el que se desarrollará el juego. Para crear un tablero hay que tener en cuenta las diferentes variantes: altura del tablero, anchura del tablero,

posición en los tres ejes, color... En esta aplicación se decide crear un componente *tablero* (3.2) con una altura estándar y en una posición fija, pero con un parámetro a elección del usuario que se le pasará al componente *tablero* como propiedad, la anchura del tablero. Este parámetro llega posteriormente a la aplicación como un parámetro de la URL y en función de su valor se crea un tablero u otro.

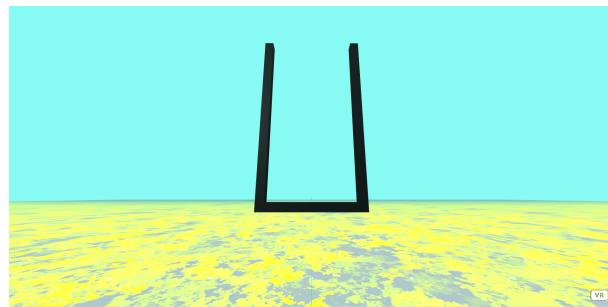


Figura 4.7: Tablero estándar ya colocado sobre la escena

Con el tablero ya creado es hora de crear el controlador que servirá para mover horizontalmente las piezas según van cayendo. Este controlador es necesario ya que se podría mover las piezas pinchando directamente sobre ellas, pero esto podría ser complicado en tableros muy grandes o muy pequeños, por lo que tener un controlador en una posición y con un tamaño fijo puede ser de gran ayuda en estas circunstancias. Para ello se decide crear un componente *mando* y un componente *controller* (3.2) sobre él, con la posibilidad de ser manipulados por el usuario mediante el ratón o los controladores VR.

Más tarde se crean los botones que realizarán diferentes acciones sobre las piezas. Se crea uno que sirva para aumentar la velocidad en la que una pieza cae sobre el tablero y otro para poder rotar una pieza en el eje x-y. Para ello se crea un componente *bajarpieza* y un componente *rotarpieza* (3.2). Uno de los principales problemas que se plantea en este caso ocurre al coincidir el botón y el texto en el mismo espacio y ser el texto un elemento hijo del botón, haciendo que el botón siempre se vea por encima del texto. Para solucionar este problema es necesario que la posición relativa del texto sea más cercana respecto al usuario (en el plano z) que la posición del botón. Por último también se añade a los botones la propiedad que permite que sean clickados por parte del usuario.

En último lugar es necesario generar los cursores que permitirán al usuario interactuar con algunos componentes de la escena ocasionando la propagación de eventos en ellos. En esta

escena son necesarios dos tipos de cursores: uno para el ratón y otro para los controladores de realidad virtual (uno para la mano derecha y otro para la mano izquierda). En todos ellos se utiliza el modelo de cursor mediante rayos en el que hay que especificar los elementos de la escena a los que van a apuntar los rayos del cursor y el tipo de objetos con los que dichos rayos tienen que colisionar. De esta forma cuando se utilice el ratón o un controlador sobre la escena se emitirán unos rayos que al colisionar con uno de los elementos objetivo producirá un evento por el cual se podrá interactuar con él.

```
<a-mixin id="objetosManejables" hoverable="suppressY:true" stretchable draggable droppable></a-mixin>
<a-mixin id="botones" clickable></a-mixin>

<a-entity mixin="pointer" raycaster="showLine: true; objects: .cubo, a-link, #controller, .boton" sphere-collider="objects: a-box, a-sphere" oculus-touch-controls="hand: right" oculus-go-controls>
<a-entity mixin="pointer" raycaster="showLine: true; objects: .cubo, a-link, #controller, .boton" sphere-collider="objects: a-box, a-sphere" oculus-touch-controls="hand: left"></a-entity>
<a-entity mixin="pointer" raycaster="showLine: true; objects: .cubo, a-link, #controller, .boton" sphere-collider="objects: a-box, a-sphere" cursor="rayOrigin: mouse"></entity>

<a-entity id="score" score="position: 0 25 -20; anchuraTexto:30; alturaTexto:30"></a-entity>

<a-entity id="tablero" tablero="alturaSuelo: 1; positionSuelo: 0 0.5 -20; colorTablero: black;
alturaPared: 21; anchuraPared: 1; positionParedIzq: 10.5 -20; positionParedDer: 10.5 -20">
<a-entity id="piezas"></a-entity>
</a-entity>

<a-entity id="botones">
<a-box bajarPieza="position: 15 -20; color: red; width: 3; height: 3; depth: 0; id:bajarPieza; mixin:botones;
positionText: 0.3 -0.4 1; colorText: black; alignText: center; valueText: BAJAR PIEZA; widthText: 10" sound _click="src: #click; on: click; poolSize: 2"></a-box></a-box>
<a-box rotarPieza="position: 15 -20; color: red; width: 3; height: 3; depth: 0 id:rotarPieza; mixin:botones;
positionText: -0.3 -0.4 1; colorText: black; alignText: center; valueText: ROTAR PIEZA; widthText: 10" sound _click="src: #click; on: click; poolSize: 2"></a-box>
</a-entity>

<a-entity id="mandocontrolador">
<a-plane mando="height:0.2; position: 0 0.5 -5; rotation: -45 0 0; color:black; id: mando"></a-plane>
<a-sphere controller="height:0.5; width:0.5; radius:0.25; position: 0 0.5 -4.9; rotation: -45 0 0; mixin:objetosManejables; color: #525250; id: controller"></a-sphere>
</a-entity>
```

Figura 4.8: Declaración de los componentes principales del primer prototipo

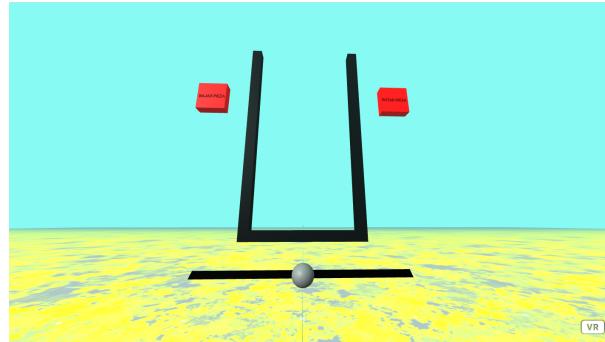


Figura 4.9: Prototipo inicial del tablero

4.3.3. Prototipo Final

Durante esta etapa se obtuvieron avances muy importantes para la consecución del proyecto, así como la aplicación objetivo del desarrollo y el primer prototipo de la misma.

Aparte de ello también se obtuvieron conocimientos necesarios para los desarrollos posteriores y sirvió como primera toma de contacto en cuanto al desarrollo de una aplicación real mediante A-Frame. Entre los principales conocimientos que se recogieron se pueden destacar los siguientes:

- Creación de un ecosistema o fondo personalizado apoyándose en el inspector visual que ofrece A-Frame.
- Generación de componentes en función de un parámetro escogido por el usuario y que llega a través de la URL.
- Colocación de componentes en la escena y superposición de los mismos para formar elementos más complejos.
- Creación de componentes que puedan ser manejados utilizando los cursores creados.
- Generación de un cursor que sea capaz de interactuar sólo con ciertos elementos de la escena, y compatible además en entornos de realidad virtual.
- Realización de una aplicación compatible con la WebXR.

El resultado de este primer prototipo se encuentra subido en el mismo repositorio GitHub que la aplicación de la fase anterior, y es accesible mediante GitHub Pages a través del siguiente enlace: [Prototipo de la aplicación](#).

4.4. Etapa 2

Con un prototipo ya creado, durante esta etapa llega su transformación hacia una aplicación con cierta funcionalidad dinámica. En este proceso se le dará un sentido a los botones y el controlador, se incorporarán las piezas a la escena y se añadirá la biblioteca *super hands* al proyecto para dotar a los componentes de un movimiento más realista.

4.4.1. Objetivos de Diseño

El objetivo principal de esta etapa es la obtención de una aplicación totalmente funcional similar al *Tetris*, partiendo del prototipo obtenido anteriormente.

4.4.2. Detalles de Implementación

El primer paso de este proceso consiste en distinguir tanto cambios como los nuevos componentes que se deben incluir para transformar el prototipo estático en una aplicación con la funcionalidad requerida. En primer lugar se necesita un elemento básico del Tetris como son las piezas, por lo que se deben crear nuevos componentes. Por otro lado se debe añadir funcionalidad tanto a los botones como al controlador para modificar el comportamiento de las piezas que van cayendo sobre el tablero.

Debido a que las piezas todavía no existen en la escena, debe crearlas un componente de los ya existentes e incorporarlas a la escena otorgándolas unas propiedades específicas. Este componente es el tablero (3.2), que nada más ser registrado como componente A-Frame crea una pieza y automáticamente no vuelve a crear otra más. La pieza se crea por encima del tablero, en el centro de su anchura. Mientras que las piezas del Tetris original pueden presentar múltiples formas, estas piezas son siempre cubos de alturas y anchuras aleatorias entre uno y cuatro. El color de cada una de las piezas también es aleatorio.

Cada una de las piezas se crea introduciendo a la escena un componente cubo (3.2). Al no ser componentes estáticos todas las piezas reciben una propiedad llamada *velocidad*, cuyo valor determinará lo rápido que caen hacia abajo. Este valor es constante para todas ellas. Para conseguir este movimiento, cada pieza implementará la función tick (una vez sea registrada en A-Frame) y en cada paso por ella disminuirá su posición en el eje y tanto como marque el valor

de la propiedad *velocidad*. Esto consigue el efecto de una pieza cayendo hacia abajo con una velocidad constante desde su creación.

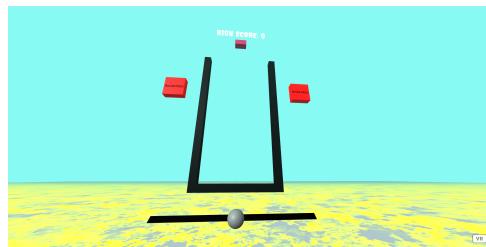


Figura 4.10: Creación de una pieza en la escena

Para conseguir que las piezas no bajen constantemente, si no que cesen su movimiento cuando llegan a la parte inferior del tablero, es necesario revisar la posición de la pieza a cada paso por el tick. En caso de no haber contactado con el tablero, la pieza continua bajando; en caso contrario, la pieza se queda inmóvil en la posición en la que haya caído sobre el tablero, y este último se encarga de crear un nuevo componente *cubo* de la misma forma que creó el anterior. De esta forma se consigue tener siempre una única pieza en movimiento, la única que recibe las acciones de los botones y el controlador. Si una pieza está cayendo al tablero y en su posición vertical se encuentra una pieza ya situada en él, el punto de tierra ya no es el tablero, si no que es la pieza que esté más elevada sobre esa posición. De esta forma se consigue que las piezas vayan cayendo y se apilen las unas encima de las otras, ofreciendo la posibilidad de llenar el tablero.

Para lograr dicha apilación se crea una variable al crear el tablero con sus dimensiones y el estado de cada una de sus posiciones (considerando una unidad del tablero como un cubo 1x1). Cada vez que una pieza cae sobre el tablero se rellena esta variable y se introduce el contador de la pieza en las posiciones del tablero en las que haya caido la totalidad de la pieza. De esta forma cada vez que una pieza va cayendo es necesario consultar esta variable en cada paso por el tick para consultar el estado de las posiciones por las que está pasando la pieza y ver si están vacías o no.

Otra funcionalidad que presenta el Tetris es la eliminación de filas una vez se van llenando. Para ello se consulta la variable que contiene el estado de todas las posiciones del tablero cada vez que una pieza cae sobre él o sobre otra pieza. En caso de estar alguna fila completa, se eliminan los componentes que permanezcan a esa fila, eliminándolos así de la escena. En caso

de que una parte del componente pertenezca a esa fila y otra no, se elimina únicamente la parte del componente que pertenece a ella, disminuyendo así su altura. Las piezas que quedan por encima de la fila eliminada, bajan todas ellas las posiciones correspondientes en el eje y hasta contactar con el tablero o con otra pieza. Por último se actualiza la variable del tablero para dejar constancia de su estado actual después de la eliminación.

El siguiente paso es darle un sentido al controlador creado anteriormente, que no es otro que el de dotar de movimiento lateral a las piezas. Las piezas no son elementos que caen de forma estática, si no que pueden ir moviéndose por todo el eje x del tablero mientras están cayendo, y ese movimiento lo determina el propio movimiento del controlador. Cuando el usuario mueve el controlador hacia una dirección en el eje x, se guarda la nueva posición en una variable. El componente *cubo* se encarga de revisar el valor de esta variable a cada paso por el tick, y en caso de haber cambiado su valor modificará su posición x al valor de esta variable multiplicada por dos, ya que el controlador es la mitad de ancho que el tablero. Esto crea una serie de problemas debido a que el movimiento de las piezas no puede ser libre, si no que esté delimitado por el tamaño del tablero. Los principales problemas son los siguientes:

- El controlador debe tener una anchura proporcional a la del tablero, ya que si no sería imposible transportar el cambio de posición del controlador a la pieza.
- El controlador solo debe considerar movimiento en el eje x y dentro de los límites marcados por la anchura del tablero, evitando así que las piezas caigan fuera de él.
- Cuando una pieza cae y se genera otra nueva, el controlador debe volver a su posición inicial, ya que la nueva pieza caerá desde esa posición vertical.
- Las piezas no pueden caer en cualquier posición x del tablero, si no que solo deben considerarse las casillas que se han establecido (una casilla es un cubo de dimensiones 1x1). Para ello antes de mover la pieza a la posición de destino, es necesario conseguir la casilla a la que hace referencia esa posición y corregirla de modo que quede exactamente dentro de ella.
- También es necesario distinguir entre las piezas de anchura par y las de anchura impar, ya que las piezas de anchura impar deben caer sobre el centro de una casilla y las de anchura par deben caer sobre el límite que separa dos casillas.

- Antes de mover una pieza a otra posición también hay que conocer el estado de la casilla a la que se va a mover la pieza, ya que si está ocupada será necesario cancelar el movimiento y mantener la pieza en su posición actual.

Ahora que las piezas y el controlador ofrecen la funcionalidad para la que fueron creados, es necesario crear la funcionalidad de los botones. El primero de ellos es el encargado de bajar las piezas. Este botón recibe un evento click que al dispararse modificará la posición de la pieza que está cayendo en ese momento. Para ello el componente del botón accede a la posición de la pieza y modifica esa posición haciendo que disminuya una unidad en su componente y. Antes de realizar esa modificación, es necesario comprobar el estado de la casilla inferior a la que va a moverse la pieza, ya que si está ocupada por otra pieza o se trata del límite inferior del tablero, este movimiento no podrá llevarse a cabo. El otro botón es el encargado de rotar las piezas. Este componente también recibe únicamente el evento click, que cuando se dispare accederá a la pieza cambiando su altura por su anchura y viceversa, provocando así una rotación en el eje x-y de noventa grados. Antes de realizar esta rotación será necesario comprobar el estado de las casillas que va a pasar a ocupar la pieza rotada, ya que si están ocupadas deberá cancelarse dicha rotación.

Para añadir una funcionalidad extra se introduce un nuevo componente llamado *score* (3.2) que será el encargado de informar al usuario de la puntuación que lleva en cada momento. Si ha caído una pieza, multiplica su superficie por 100 y suma los puntos a los que había en ese momento; si se ha eliminado alguna fila, multiplica 5000 por el número de filas eliminadas y suma los puntos.

Por último se define el momento en el que el juego termina, cuando la superficie de una pieza supera el límite superior del tablero. Para ello cada vez que cae una pieza se analiza las casillas que ocupa su superficie, y en caso de ocupar alguna casilla que se encuentra por encima de los límites del tablero, se redirige al usuario a la pantalla de fin de juego.

4.4.3. Prototipo Final

Esta estapa fue la más importante de todo el desarrollo debido a que fue la que más problemas planteó y de la que se trajeron los conocimientos y habilidades necesarias para desarrollar el resto de etapas.

En primer lugar se adquirieron los conocimientos necesarios para trabajar con eventos que se disparan sobre ciertos componentes. Durante el desarrollo de esta etapa se trabajó con dos tipos de eventos: los que se activan por una acción del usuario sobre un componente y ocasionan modificaciones sobre ese mismo componente, y los que se activan por una acción del usuario sobre un componente y provocan modificaciones sobre otro componente distinto.

También se agudizó el conocimiento sobre el acceso y la manipulación de los elementos que forman parte del DOM de la aplicación. En muchos casos fue necesario acceder a determinados componentes desde otros, por lo que era necesario acceder a ellos empleando consultas al DOM, con el objetivo de modificar sus propiedades, añadir nuevos componentes o eliminar alguno ya existente.

Otro avance destacado de esta etapa fue el manejo de la interacción entre distintos componentes. Esto fue necesario en muchos puntos de esta etapa, ya que era necesario controlar la colisión entre los componentes de la escena y realizar determinadas acciones cuando esta ocurriera.

Por último también se aprendió a controlar la interacción entre el cursor (el ratón en este caso) y los componentes de la escena. Los eventos que se disparan en esta aplicación son activados cuando el cursor realiza una determinada acción sobre los componentes objetivo. Para ello fue necesaria la importación de la librería *super-hands*, que proporciona una serie de interacciones intuitivas y fácilmente manejables en A-Frame. Esta librería permite que los componentes de una escena se puedan clickar, agarrar, soltar, arrastrar o estirar. Por poner un ejemplo relativo a esta aplicación, los botones pueden recibir un click y el controlador puede ser arrastrado gracias a esta librería.

El resultado final de esta etapa es una primera versión del Tetris totalmente funcional que supondrá el modo estándar de esta aplicación y servirá como punto de partida para el resto de modos.

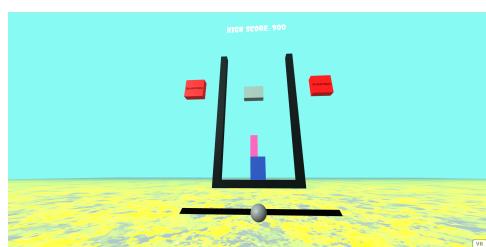


Figura 4.11: Prototipo final de la Etapa 2

4.5. Etapa 3

Al comienzo de esta etapa se parte de una aplicación totalmente funcional en un entorno de dos dimensiones y manejable a través del ratón. Durante el transcurso de ella se adapta la aplicación a la WebXR y los controladores de realidad virtual.

4.5.1. Objetivos de Diseño

El objetivo principal de esta etapa es la adaptación de la aplicación al entorno de realidad virtual con el objetivo de su funcionamiento tanto en dos dimensiones como en tres dimensiones.

4.5.2. Detalles de Implementación

Para el comienzo de esta etapa se necesitan unos requisitos previos: el material necesario para probar la aplicación en un entorno de realidad virtual (gafas y controladores VR) y una aplicación totalmente compatible con el entorno y los dispositivos de realidad virtual.

En este caso la aplicación es compatible con un entorno de realidad virtual (WebXR) debido a que las aplicaciones desarrolladas mediante A-Frame son totalmente compatibles con estos entornos puesto que esta herramienta está completamente orientada a la creación de proyectos en realidad virtual. Respecto a la compatibilidad con los dispositivos VR, es necesario introducir en el código de la aplicación un componente que especifique que se va a utilizar un controlador VR, qué tipo de controlador va a usarse y sobre qué componentes va a actuar. En este caso ya se introdujeron anteriormente dos componentes que indican que va a existir un cursor para la mano derecha y otro para la mano izquierda, que se van a utilizar controladores *Oculus*, y los componentes sobre los que van a actuar mediante la propagación de rayos que colisionen sobre ellos.

Ahora que la aplicación está enlazada con los dispositivos y el entorno de realidad virtual, el siguiente paso es acceder a la aplicación mediante las gafas VR para probar la aplicación en ellas. En este punto hay dos formas diferentes de proceder: acceder a la url en la que está desplegada la aplicación desde el navegador que ofrecen las gafas y activar el modo VR, o conectar las gafas a un ordenador o dispositivo móvil para visualizar la pantalla del dispositivo a través de las gafas. En este caso se decide proceder de la segunda manera debido a que es la

más completa de las dos. Cuando se conectan las gafas a un dispositivo es posible ir alternando entre la versión de escritorio y la versión de realidad virtual, y sobre todo es posible acceder a la consola de la página web mientras se prueba la aplicación con las gafas. Esto permite la depuración del código mientras se prueba la aplicación en el entorno virtual, lo que puede ser muy útil a la hora de corregir fallos de desarrollo en las modalidades VR de la aplicación.

El primer paso para conectar las gafas VR al ordenador es crear una cuenta en la aplicación que ofrece Oculus para dispositivos móviles. Esta cuenta ha de ser de desarrollador, con el fin de permitir ciertas funcionalidades que solo se permiten en este modo. Más tarde es necesario añadir en la aplicación el dispositivo de realidad virtual que vamos a utilizar, y para ello hay que conectar las gafas al ordenador mediante un cable de red. Una vez finalizado este proceso las gafas estarán asociadas con la cuenta de desarrollador que acaba de ser creada.

El siguiente paso consiste en conectar esa cuenta con el dispositivo enlazado al propio dispositivo. Para ello se introduce la misma cuenta en las gafas de realidad virtual en el momento en que estas se configuran. En este punto ya solo queda establecer el enlace entre las gafas y el ordenador. Este enlace puede ser de dos tipos: a través de un cable de red o a través de Wi-Fi. En este caso se intenta en primer lugar establecer el enlace mediante la red Wi-Fi, pero no funciona debido a que nunca llega a completarse y no se llega a ver la pantalla del ordenador a través del dispositivo. Por ello se recurre a la siguiente alternativa, el enlace por cable.

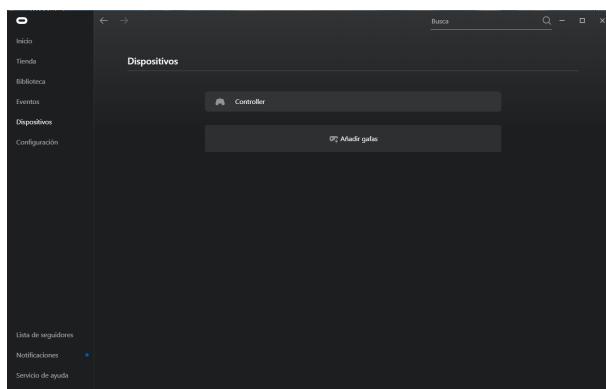


Figura 4.12: Interfaz de la aplicación de Oculus

Para ejecutar este enlace se utiliza otra aplicación diferente, el Oculus Developer Hub. En primer lugar se sincroniza la cuenta Oculus en esta aplicación, se conecta el dispositivo de realidad virtual mediante un cable de red al ordenador, y se selecciona la opción que permite la depuración de código a través de las gafas: *Enable ADB Over Wi-Fi*. Una vez seguido este

proceso se establece la conexión y aparece en las gafas un menú para que el usuario seleccione lo que va a hacer. En este caso se selecciona la opción del escritorio y acto seguido aparece la pantalla del ordenador en el dispositivo de realidad virtual. Por último será necesario pulsar el botón VR que aparece en pantalla para ejecutar la aplicación en su versión de realidad virtual.

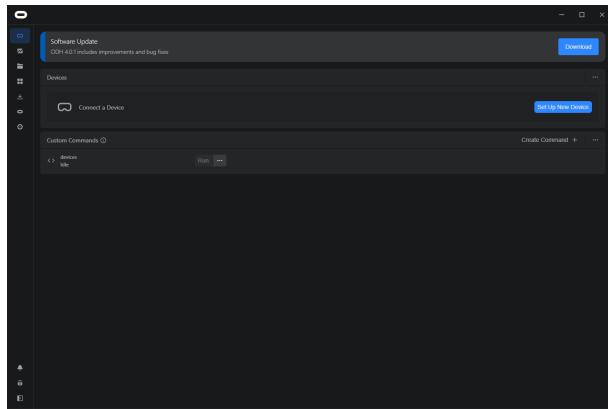


Figura 4.13: Interfaz de la aplicación de Oculus Developer Hub

De esta forma se consigue manejar el ordenador mediante el dispositivo de realidad virtual, lo que permite probar la aplicación en un entorno VR y más tarde analizar su estado accediendo a la consola de depuración. El usuario puede también alternar entre los dos modos de la aplicación con la simple acción de ponerse o quitarse las gafas y ejecutar la aplicación pulsando o sin pulsar el botón VR que aparece en pantalla.

4.5.3. Prototipo Final

Esta etapa fue bastante diferente a las demás, debido a que se frenó el proceso de desarrollo de la aplicación para hacerla compatible con un entorno que se iba a utilizar en otros modos más adelante.

Se partía de una aplicación en dos dimensiones y preparada para usarse con el ratón como controlador único, pero cuyo código estaba implementado para soportar una versión en realidad virtual. Por eso mismo no fue necesario modificar nada del código de la aplicación, si no simplemente enlazar el dispositivo VR al ordenador en el que se iba a probar el funcionamiento de la misma.

Durante el desarrollo de la etapa se obtuvieron conocimientos acerca del funcionamiento de dos aplicaciones desconocidas hasta el momento, Oculus y Oculus Developer Hub. Pero lo más

importante fue la capacidad de depurar el código de la aplicación una vez era probada en entornos de realidad virtual. Este avance fue muy importante en los modos VR que se desarrollaron más adelante, ya que esos modos debían ser probados en entornos VR y sin la posibilidad de depurar código hubiera sido mucho más difícil avanzar en su desarrollo.

Como resultado final de la etapa, se obtuvo la misma aplicación que en la etapa anterior, pero lista para ser desplegada y probada tanto en dos dimensiones como en tres dimensiones, con la ayuda del material de realidad virtual necesario.

4.6. Etapa 4

Durante esta etapa se comienzan a desarrollar los modos adicionales de esta aplicación, partiendo del modelo estándar creado en la segunda etapa ([4.4](#)). En este caso la etapa se centra en los modos de escritorio.

4.6.1. Objetivos de Diseño

El objetivo principal de esta etapa es el desarrollo completo de todos los modos adicionales de la aplicación en su versión de escritorio o dos dimensiones.

4.6.2. Detalles de Implementación

Esta aplicación consta de diversos modos de juego divididos en dos bloques principales: los modos de escritorio y los de realidad virtual. En este caso todos ellos van a ser compatibles con ambos entornos pero todos ellos van a estar más preparados para ser desplegados en un entorno que en otro. Por ello antes de comenzar a desarrollar cualquier tipo de modo durante esta etapa, hay que comprender que este no debe tener una complejidad excesivamente alta para poder funcionar en la versión de escritorio de la aplicación sin ningún tipo de problema.

Otro aspecto a tener en cuenta es que todos los modos de la aplicación deben tener características que los hagan únicos entre ellos pero también deben tener similitudes que demuestren que todos ellos no son más que versiones diferentes del mismo juego, el Tetris. Por esta razón todos los modos van a tener una estructura muy parecida al modelo estándar ([3.1.4](#)) de la aplicación.

Los modos que se decide implementar son los siguientes:

- **COLORES.....** ([3.1.5](#))

La mayor complejidad de desarrollo de este modo ocurre cuando hay que conocer una propiedad de la pieza sobre la que va a caer la pieza en movimiento para saber si esta puede caer o se termina el juego. Para conseguirlo es necesario acceder a la casilla inmediatamente inferior a la casilla por la que está pasando la pieza en movimiento en cada paso por el tick. Una vez se sabe cuál es esa casilla se accede a su valor mediante la variable que guarda el estado de todas las casillas del tablero. Si la casilla está vacía, la

pieza sigue cayendo, si al contrario contiene una pieza se accede a esa pieza y se extrae su propiedad *color*. Ese color se compara con el color de la pieza en movimiento para que quede alojada encima de ella en caso de ser iguales o para finalizar el juego en caso de ser diferentes. A la hora de finalizar el juego se hace exactamente igual que si se hubiera excedido el límite superior del tablero.



Figura 4.14: Ejemplo de funcionamiento del modo COLORES

■ OBSTÁCULOS..... (3.1.6)

La mayor complejidad de este modo vuelve a ser el tener que acceder al valor de la casilla inmediatamente inferior a la pieza en movimiento, al igual que en el modo de los colores. En este caso si ese valor es un obstáculo el juego deberá terminar de igual manera que en otro modo.

Otra complejidad que puede presentar este modo es el valor que se da a la propiedad velocidad de las piezas. Este valor marcará la disminución de la componente y de la posición de la pieza a cada paso por el tick. De esta forma no se puede poner un valor que exceda la altura de una casilla, ya que la pieza saltaría una casilla entre medias y sería imposible comprobar si en esa casilla había un obstáculo o no. Esto hace que sea importante aumentar el valor de la velocidad para una mejor experiencia de usuario pero dentro de unos límites que no alteren el correcto funcionamiento de la aplicación.

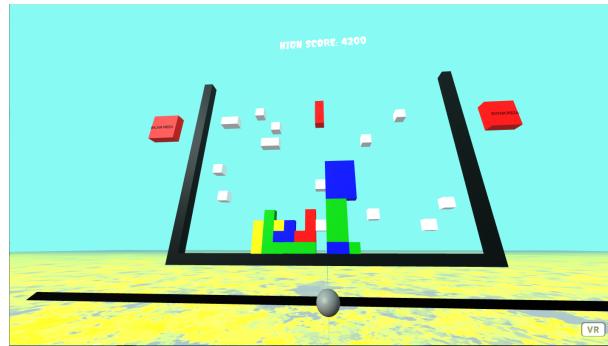


Figura 4.15: Ejemplo de funcionamiento del modo OBSTÁCULOS

4.6.3. Prototipo Final

El resultado final de esta etapa se corresponde con la parte en dos dimensiones de la aplicación al completo. Al finalizar este proceso se han creado tres modos diferentes totalmente funcionales y compatibles con entornos en dos y tres dimensiones, aunque específicamente diseñados para ser desplegados en dos dimensiones.

En este punto un usuario de la aplicación es capaz de acceder a la aplicación, entrar en el modo que elija (mediante un menú que todavía no se ha creado) y desplegar la aplicación directamente sobre una dirección URL para probarla con la única ayuda de un ratón.

Esto se podría considerar una aplicación completa, pero ya que el objetivo de este proyecto es la adaptación del juego al entorno virtual, será necesario desarrollar también la parte de la aplicación diseñada específicamente para ser desplegada en entornos de realidad virtual.

4.7. Etapa 5

Durante esta etapa se continúan desarrollando los modos adicionales de esta aplicación, partiendo del modelo estándar creado en la segunda etapa (4.4). En este caso la etapa se centra en los modos de realidad virtual.

4.7.1. Objetivos de Diseño

El objetivo principal de esta etapa es el desarrollo completo de todos los modos adicionales de la aplicación en su versión de realidad virtual o tres dimensiones.

4.7.2. Detalles de Implementación

El resultado de la anterior etapa fue una aplicación completa pero centrada en un despliegue en dos dimensiones, por lo que esta etapa se centra en el desarrollo de las nuevas modalidades de la aplicación cuyas características invitan a su despliegue en tres dimensiones.

Esta es la parte más importante para el objetivo principal del proyecto, ya que estos modos representan el resultado final de la adaptación del Tetris a los entornos de realidad virtual.

Se crean cuatro nuevos modos que son detallados a continuación:

- **MÚLTIPLES TABLEROS.....** ([3.1.7](#))

El mayor problema de esta modalidad reside en que el usuario debe ser capaz de manejar al mismo tiempo el juego en tres tableros diferentes teniendo en cuenta que cada pieza debe caer en su tablero, cada controlador debe mover únicamente su pieza y cada botón debe afectar únicamente a su pieza. Para resolverlo se recurre a la identificación de un tablero como tablero derecho, otro como tablero y otro como tablero izquierdo. Esto conlleva que todas las variables del tablero izquierdo (piezas, controlador, botones, etc) lleven un identificador acabado en *-izq*, todas las del tablero central usan el identificador por defecto y todas las del tablero derecho usarán un identificador acabado en *-der*. De esta forma cada vez que se produce un evento sobre una pieza, un controlador o un botón, es necesario acceder a su id para comprobar sobre qué elemento se ha producido dicho evento.

El usuario debe controlar el juego en tres tableros diferentes, pero cada uno de ellos es independiente al anterior, es decir, los cambios en un tablero no afectan a los demás. Solo hay dos características que son comunes a todos los tableros: el marcador y el final de juego. En este caso el marcador se actualiza de la misma forma que en el modelo estándar con la única diferencia de que los tres tableros actualizan el mismo marcador, por lo que existen más posibilidades de obtener más puntos; en cuanto al final de juego se sigue la norma de que en cuanto el juego finalice en un tablero, finaliza en todos los demás.

Para terminar solo queda aclarar que cabe la posibilidad de desplegar este juego con dos tableros o únicamente con el tablero original, en cuyo caso el juego sería idéntico al modo estándar.

La característica que hace que este modo esté diseñado para desplegarlo en tres dimensiones es la posición de los tableros adicionales. Al estar a ambos lados del tablero original, es posible que esto resulte incómodo en dos dimensiones y que sea necesario incluso mover la pantalla para conseguir llegar a dichos tableros. La situación cambia en realidad virtual, ya que el usuario puede llegar a ver esos tableros fácilmente moviendo la cabeza o moviendo su propio cuerpo por la escena acercándose a ellos.

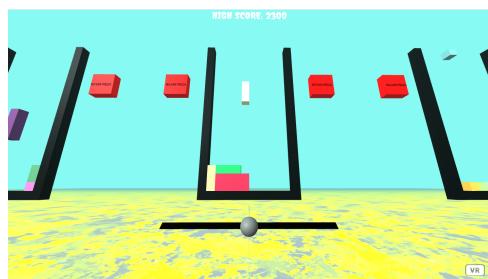


Figura 4.16: Ejemplo de funcionamiento del modo MÚLTIPLES TABLEROS

■ 360 GRADOS.....(3.1.8)

En primer lugar cabe destacar que es necesario crear dos componentes de cada tipo (dos tableros, dos controladores, dos entornos de piezas...), cada uno a un lado del eje z. Es fundamental que cada componente pueda identificarse según si pertenece al escenario principal o al opuesto, por lo que todos los elementos que pertenezcan al escenario opuesto contendrán un identificador acabado en *trasero* u *opuesto*. De esta forma cuando se produzca un evento sobre cualquiera de estos elementos se podrá identificar sobre cuál de

ellos se ha producido en realidad.

Durante la creación de los escenarios surge el primer problema de este desarrollo, la superposición de los elementos a ambos lados del eje z. Anteriormente ya surgió este problema y se solucionó colocando el elemento que se superpone al otro en una posición ligeramente adelantada respecto al eje z. Cuando se trata del escenario opuesto es necesario hacer lo contrario, ya que al otro lado del eje z las coordenadas son negativas y por lo tanto cuanto menor sea su valor, más cerca del plano estará colocado el componente. Esto sirve por ejemplo a la hora de colocar el texto sobre los botones del tablero opuesto.

Otro problema que surge con los botones es la colocación del texto, ya que en el plano opuesto el texto se lee al revés. Para solucionar este problema es necesario dotar al elemento que contiene dicho texto de una rotación de 180 grados para que el usuario pueda leerlo de manera correcta en el momento que gire la cámara.

El juego se desarrolla en ambos tableros de la misma manera, sin embargo nunca en los dos simultáneamente. Por ello la mecánica de juego es la siguiente:

- Al inicio del juego, se crean ambos tableros pero solo se crean las piezas que van cayendo sobre uno de ellos, el principal.
- El juego se desarrolla solo sobre este tablero mientras que el otro permanece estático.
- Se genera una variable con un número aleatorio entre 1 y 5, y cuando caigan sobre el tablero ese número de piezas aparece un indicador para que el usuario pase al otro tablero.
- En cuanto este indicador sale en pantalla comienzan a caer piezas sobre el tablero opuesto y se vuelve a generar una variable del 1 al 5 que determina el número de piezas que deben caer para volver a cambiar de tablero.
- Este ciclo se repite hasta que el juego finaliza cuando alguna pieza supera el límite superior de alguno de los dos tableros. La puntuación también es única y contribuyen ambos tableros de la misma forma.

El indicador que se ha descrito anteriormente es un componente nuevo que ha de añadirse a la escena. Para ello al crearla se genera este componente que constará de un texto que

irá colocado a ambos lados de cada tablero. Este texto indicará al usuario que debe girar 180 grados porque el juego va a cambiar al tablero opuesto. Permanecerá oculto hasta que esto ocurra y solo aparecerá a los lados del tablero en el que se está desarrollando inicialmente el juego.

Lo que hace que este juego esté diseñado para un entorno en tres dimensiones es el constante cambio de tableros y el espacio en el que se encuentran cada uno de ellos. Para desarrollar este modo en dos dimensiones el usuario tendría que mover la cámara a mano 180 grados cada vez que el juego cambie de tablero, dando como resultado una acción muy tediosa y para nada intuitiva. En cambio en un entorno en tres dimensiones el usuario puede cambiar de tablero con un simple giro completo de su cuerpo, una acción mucho más sencilla y rápida.

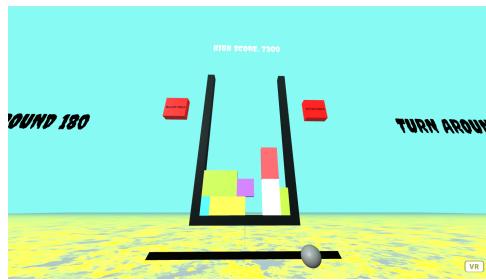


Figura 4.17: Ejemplo de funcionamiento del modo 360 GRADOS

■ PIEZAS EN CAÍDA LIBRE..... (3.1.9)

Al comenzar la escena se crea el tablero y el marcador, al igual que en el resto de modos, pero no se crean ni los botones ni el controlador, debido a que no van a ser de utilidad. Tampoco se crea una pieza que comenzará a caer sobre el tablero al inicio del juego, en su lugar se crean tres piezas aleatorias a cada lado del tablero que conservan su posición. Estas piezas poseen una característica diferente a las del resto de modos: la capacidad de interactuar con ellas permitiendo agarrarlas con el controlador y moverlas hacia otra posición. Por último se crea un nuevo componente que consiste en un botón para rotar todas las piezas que se encuentren flotando en ese momento.

El usuario debe agarrar una pieza y moverla hacia una posición por encima del límite superior del tablero. Una vez el usuario deja de agarrar la pieza, se comprueba la casilla en la que ha quedado. Si esa casilla se encuentra por encima del tablero y dentro de su

rango horizontal, la pieza cae directamente hacia abajo hasta que colisiona con el tablero o con otra pieza que ya se encuentre en él; en caso contrario la pieza vuelve a su posición inicial. Cabe destacar que durante la caída no se puede modificar el estado de una pieza, es decir, si el usuario desea rotar la pieza debe rotarla antes de que esta comience a caer sobre el tablero. Cuando una pieza cae sobre el tablero se elimina la característica de poder ser agarrada para evitar que el usuario pueda volver a cambiarla de posición.

Una vez que el usuario ha colocado sobre el tablero las seis piezas que se encontraban flotando al inicio, se generan de la misma manera otras seis diferentes. Para que esto sea posible es necesario identificar qué piezas de la escena se encuentran flotando y qué piezas están colocadas ya en el tablero. Por ello se crea una variable que contiene los identificadores de las piezas que están flotando en ese momento. De esta manera el usuario solo puede agarrar las piezas que pertenezcan a ese conjunto y cuando pulse el botón de rotar piezas, solo rotarán ellas.

El ciclo se repite hasta que el juego termine, de igual manera que en el resto de modalidades.

Este modo está diseñado para ser desplegado en un entorno de tres dimensiones debido a que el usuario debe estar constantemente interactuando con las piezas, agarrándolas y transportándolas a una nueva posición. Además la obligación de dejar las piezas en una posición superior al tablero podría provocar que el usuario no pudiera llegar de manera sencilla a esas posiciones usando un ratón, en caso de tableros altos. En un entorno en tres dimensiones es mucho más sencillo ya que la profundidad del escenario facilita que el usuario pueda acceder a las posiciones más alejadas del mismo mediante los controladores.

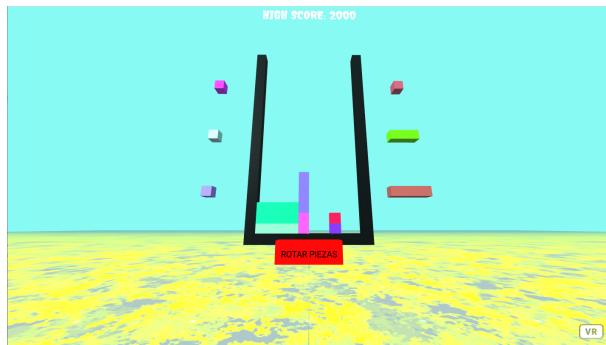


Figura 4.18: Ejemplo de funcionamiento del modo PIEZAS EN CAÍDA LIBRE

- **TEST DE REFLEJOS..... (3.1.10)**

Al comienzo de la escena se crean únicamente el tablero y el marcador, ya que en este modo no se puede modificar el estado de las piezas mientras estas caen. Además se crea la primera pieza de la escena quedando estática sobre el tablero y con una propiedad que permita ser pinchada o clickada por el usuario. La pieza creada es totalmente aleatoria y no se puede rotar ni agarrar hacia otra posición.

Una vez creada la pieza se crea una variable que contiene el tiempo que lleva la pieza flotando sobre el tablero y lo actualiza en cada paso por el tick. Cuando el valor de esta variable llega o supera los 3 segundos se elimina de la escena el componente que contiene esa pieza, se actualiza el marcador restando 1000 puntos y se suma una unidad a una variable que contiene el número de piezas que ha dejado escapar el usuario. Si por el contrario el usuario hace click sobre la pieza antes de que el contador llegue a los 3 segundos, la pieza cae hacia abajo en la misma posición respecto al eje y en la que se encuentra flotando y queda colocada en el tablero. En este caso no se restan puntos ni se actualiza la variable con las piezas saltadas.

En ambos casos el siguiente paso es generar una nueva pieza aleatoria y repetir el mismo ciclo hasta que finalice el juego. Este modo puede finalizar de la misma manera que todos los demás (cuando se rebasa el límite superior del tablero), pero añade una nueva forma de concluirlo en cuanto el usuario deja que desaparezcan 10 piezas.

De esta forma el usuario debe decidir en un periodo muy corto de tiempo si quiere que una pieza caiga sobre el tablero o no. También tiene que elegir bastante bien las piezas que no desea colocar en él, ya que solo tiene 9 oportunidades para ello, lo que añade otro

grado de dificultad. Por último se introduce un nuevo obstáculo al tener que hacer click directamente sobre la pieza para que esta caiga, haciendo que el usuario deba tener cierta puntería para acertar sobre la pieza en el tiempo indicado. Es este factor el que hace que este modo de juego esté diseñado para desplegarse en entornos de tres dimensiones, que facilitan el alcance de las piezas sean cuales sean las dimensiones del tablero.



Figura 4.19: Ejemplo de funcionamiento del modo TEST DE REFLEJOS

4.7.3. Prototipo Final

El resultado de esta etapa es la consecución de la parte más importante del proyecto, un juego funcional tanto en entornos de escritorio como en entornos de realidad virtual. La aplicación ya consta de múltiples modos de juego compatibles con ambos entornos y listos para ser desplegados.

Tras estas dos últimas etapas se han superado multitud de obstáculos y se han generado todo tipo de componentes que han permitido la adquisición de los conocimientos necesarios para implementar cualquier tipo de desarrollo en A-Frame. En este punto se podrían generar nuevos métodos de juego que incorporasen nuevas funcionalidades partiendo de la base ya adquirida. Sin embargo se considera completada la funcionalidad principal de la aplicación y el siguiente paso será construir la interfaz necesaria para conectar todos los modos de la aplicación entre sí.

4.8. Etapa 6

Durante esta etapa se generan las pantallas de navegación de la aplicación. Estas pantallas sirven como interfaz del juego y conectan los diferentes modos de juego entre sí. Serán necesarios diferentes tipos de pantallas: menús, pantallas de elección y pantalla de retorno.

4.8.1. Objetivos de Diseño

El objetivo de esta etapa es la creación de pantallas que permitan al usuario navegar por la aplicación y conecten todos los modos de juego entre sí, permitiendo cambiar de uno a otro con facilidad.

4.8.2. Detalles de Implementación

Durante este periodo del desarrollo se generan las siguientes pantallas:

- **MENÚ PRINCIPAL.....** ([3.1.1](#))

Para la creación de esta pantalla serán necesarios pocos componentes: un entorno llamativo, un componente con el nombre y logo de la aplicación, y un par de botones para que el usuario pueda elegir entre las opciones pinchando sobre uno u otro. Para generar el entorno de la pantalla se recurre al inspector visual que proporciona A-Frame y su resultado se introduce en el componente de la escena principal. Para la creación del logo de la aplicación es necesario recurrir a otro tipo de componentes A-Frame que todavía no se han visto y se detallarán más adelante, los *assets*. Por último se crean los botones y se introduce un componente de texto que contenga cada una de las elecciones. Estos botones poseen la característica de poder ser pinchados con el ratón o controlador, y presentan una animación que hace que suban y bajen ligeramente.

Cabe destacar que A-Frame presenta una propiedad especial que se le pueda dar a cualquiera de sus componentes (**animation**) y los dota de una determinada animación. Esta funcionalidad no sirve en el caso de esta aplicación debido a una incompatibilidad entre la versión de A-Frame que soporta las animaciones y la que soporta la librería **super-hands**, necesaria para hacer click sobre los botones. Esto obliga a generar la animación de los

botones de manera manual, estableciendo un límite inferior y superior y haciendo que los botones bajen o suban en función de si han llegado a un límite u otro.

Una vez que el usuario ha hecho click sobre uno de los botones se realiza una redirección a la página que corresponda con la opción elegida, que en este caso será el menú de los modos de escritorio o el de los modos de realidad virtual.

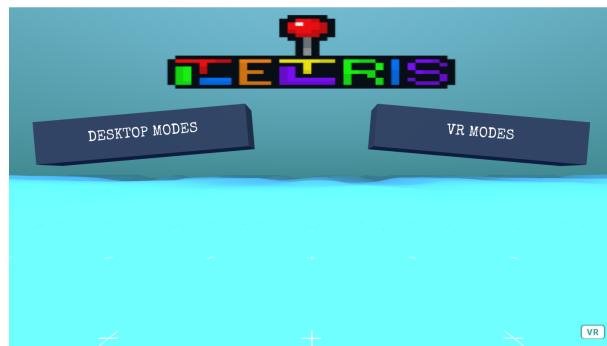


Figura 4.20: Menú principal

■ MENÚ SECUNDARIO..... (3.1.2)

La pantalla es similar a la del menú principal, cambiando el ecosistema por otro y adaptando el número de botones al número de modos que haya disponibles para cada tipo de entorno. La pantalla para los modos de escritorio contiene tres botones mientras que la otra contiene cuatro. Cuando el usuario haga click en uno de los botones se realizará una redirección al modo de juego elegido o a otra pantalla intermedia que se explicará a continuación.

Por último se crea un componente nuevo en ambas pantallas que consiste en otro botón acompañado de un texto que permite al usuario regresar a la página anterior. Cuando el usuario pulsa en él se realiza la redirección a la pantalla del menú principal.



Figura 4.21: Menú secundario

■ MENÚ DE ELECCIÓN..... (3.1.3)

La pantalla de elección de anchura del tablero está presente en todos los modos del juego excepto el modo OBSTÁCULOS (3.1.6) y el modo MULTITABLEROS (3.1.7), mientras que la pantalla de selección de número de obstáculos solo está presente en el modo OBSTÁCULOS (3.1.6). Así mismo ambas pantallas presentan cuatro opciones fijas para elegir.

Al igual que en las demás pantallas, cuando el usuario pulsa uno de los botones se hace una redirección en este caso al modo elegido anteriormente, sin embargo el valor de la opción elegida por el usuario se introduce como parámetro en la URL de redirección para poder acceder a él en el modo elegido. Este parámetro se introduce de la siguiente forma: URL?param.



Figura 4.22: Menú de elección

■ MENÚ DE FIN DE JUEGO..... (3.1.11)

Esta pantalla presenta un entorno más oscuro para que resulte más intuitivo para la vista que el juego ha finalizado. Presenta también un componente con un texto que indica *fin de juego* y un botón que realiza una redirección al menú principal una vez es pulsado por

el usuario.

También presenta un componente adicional que muestra al usuario la puntuación final que ha conseguido durante el desarrollo del juego. Para ello esta pantalla también debe recibir por URL un parámetro con el valor de esa puntuación. Todos los modos por lo tanto introducen en la redirección el valor del componente marcador en ese momento de la siguiente manera: URL??puntuacion=x.



Figura 4.23: Menú de fin de juego

4.8.3. Prototipo Final

Como resultado de esta etapa se obtiene una aplicación funcional que consta de distintos modos conectados entre sí mediante una interfaz gráfica sencilla y efectiva.

Es importante aclarar que el usuario siempre deberá comenzar el juego accediendo en primer lugar al menú principal y navegando por el resto de menús intermedios hasta llegar al modo de juego que desee. Una vez el juego haya terminado podrá volver al menú principal y repetir el ciclo nuevamente.

Durante este proceso se han agudizado principalmente los conocimientos relativos a la creación de diferentes entornos con la ayuda del inspector visual de A-Frame, las redirecciones entre pantallas utilizando parámetros o la posibilidad de introducir elementos audiovisuales llamados *assets*.

4.9. Etapa 7

Esta etapa es la última fase del proyecto y corresponde con la incorporación de elementos extra que no modifican la funcionalidad de la aplicación pero añaden ciertas características que acercan esta aplicación a un juego arcade tradicional.

4.9.1. Objetivos de Diseño

El objetivo principal de esta etapa es la adición de elementos audiovisuales que dotan a la aplicación de la apariencia básica de un videojuego y ayudan a mejorar la experiencia de usuario.

4.9.2. Detalles de Implementación

En este punto del proyecto la aplicación se encuentra finalizada y posee la funcionalidad requerida al inicio del desarrollo, sin embargo se pueden añadir elementos extra que doten de cierto realismo a la aplicación, así como imágenes o sonidos. Por ello es necesario en primer lugar importar en el proyecto los archivos *.png*, *.mp3* o *.wav* que se van a utilizar.

La herramienta A-Frame permite la adición de este tipo de elementos mediante diferentes componentes o propiedades. En esta aplicación se añade una imagen correspondiente al logotipo del proyecto en cada uno de los menús, así como sonidos tanto en los menús como en los modos de juego. Estos sonidos pueden ser de dos tipos: sonidos constantes (como el sonido principal de la aplicación) o sonidos intermitentes (correspondientes a las interacciones del usuario con los componentes de la aplicación).

```
<audio id="tetris" src="assets/tetris.mp3" crossorigin="anonymous"></audio>
<audio id="button" src="assets/button.wav" crossorigin="anonymous"></audio>
<a-image src="assets/logo.png" position="0 14 -20" width="30" height="8"></a-image>
```

Figura 4.24: Declaración de un elemento audio o a-image

```
<a-box id="vr" boton="#383D4E; position: 8 4 -10; rotation: 0 0 -5; width: 10; height: 2; text: VR MODES" click="sound _mouseenter="src: #button; on: mouseenter; poolSize: 2"></a-box>
```

Figura 4.25: Inclusión de un elemento audio en un elemento visible

Para la adición de estos elementos audiovisuales es necesario en primer lugar introducirlos en el código de la aplicación. Para ello existen dos tipos de elementos que permiten la incorporación de imágenes (*a-image*) y sonidos (*audio*). En el momento que se crea el código HTML de una de las pantallas de la aplicación, se introducen todos estos efectos en elementos de estos dos tipos. Para cada uno de estos elementos se definen una serie de propiedades que permiten acceder a ellos y establecen la forma en la que aparecen en la aplicación: un identificador para acceder al elemento, un origen con la ruta en la que se encuentra el archivo, y otra serie de propiedades adicionales así como el volumen en caso de sonidos o el tamaño en caso de imágenes.

Una vez añadidos todos estos elementos a la aplicación se procede a establecer la forma en la que van apareciendo cada uno de ellos, como se explica a continuación:

- **LOGOTIPO.** El logotipo de la aplicación se encuentra en un elemento *a-image* y debe aparecer constantemente tanto en el menú principal como en los menús intermedios. Para ello simplemente es necesario definir sus propiedades *position*, *width* y *height*. Estas propiedades establecen en qué lugar de la pantalla aparece la imagen y cuál es su tamaño.
- **SONIDO PRINCIPAL.** El sonido principal de la aplicación debe reproducirse constantemente en cada una de las pantallas de la aplicación, tanto menús como modos de juego. En este caso existen dos sonidos diferentes, uno para la pantalla de fin de juego y otro para todas las demás. Para la incorporación de este sonido se crea una propiedad en el elemento *a-scene* de cada una de las pantallas. Esta propiedad se denomina *sound* y contiene unas subpropiedades que permiten incorporarlo a la escena: un origen con el identificador del elemento que contiene el sonido para poder acceder a él, un evento de activación del sonido (en este caso el sonido debe estar activo siempre, por lo que se introduce un *mouseenter* como evento), un flag que establece si el sonido ha de repetirse una y otra vez (en este caso true) y un factor de volumen para el sonido.
- **SONIDOS EVENTUALES DE LOS MENÚS.** Al contrario que el sonido principal, estos sonidos solo deben reproducirse cuando se produzca una acción determinada y deben desaparecer cuando esta acción concluya. En este caso se introduce un sonido corto cuando el usuario ponga el cursor sobre alguno de los botones. Para ello se define una nueva propiedad en cada uno de los botones de la pantalla llamada *sound-mouseenter*. Esta pro-

piedad permite que el sonido se reproduzca cuando el cursor se encuentra sobre cada uno de los botones que la poseen, pero para ello es necesario indicar en sus subpropiedades el identificador del elemento que contiene dicho sonido, el evento de activación del sonido y el número de componentes que pueden reproducir dicho sonido al mismo tiempo.

- **SONIDOS EVENTUALES DE LOS MODOS DE JUEGO.** Estos sonidos también deben reproducirse exclusivamente cuando se produzca el evento requerido para su activación y en este caso van a definirse varios tipos de sonidos para cada uno de los diferentes modos de juego de la aplicación.

En primer lugar se definen sonidos que se activarán al pulsar los botones que presentan algunos de los modos de juego. Se definen dos tipos de sonidos, uno para los botones que sirven para bajar y rotar las piezas, y otro para los botones que sirven para crear nuevos tableros.

Para introducir los sonidos correspondientes a los botones de bajar y rotar piezas se introduce una propiedad en cada uno de estos componentes llamada *sound-click*. Se establecen sus subpropiedades indicando el identificador del elemento que contiene dicho sonido y como evento de activación se elige el evento *click*. Para introducir el sonido de creación de nuevos tableros se procede de una manera diferente debido a que estos componentes se crean de manera diferente. Para introducir este sonido se crea un nuevo componente que sirve como manejador del evento. Este componente accede al elemento que contiene el sonido a través de su identificador en cada paso por la función tick y revisa el estado de una variable que establece si el sonido se está reproduciendo. Si el sonido se está reproduciendo lo pausa y en caso contrario lo reproduce hasta su siguiente paso por el tick. El valor de esta variable se inicializa a false, se cambia a true cuando se pulsa el botón y se vuelve a cambiar a false cuando se comienza a reproducir el sonido, para pararlo de nuevo.

Otros sonidos que presenta la aplicación son el sonido de creación de una nueva pieza y el de eliminación de una o varias filas. Para la reproducción de estos sonidos se procede de igual manera que con el sonido de creación de nuevos tableros. Se crea un manejador para cada uno de ellos que accede al elemento que contiene el archivo de sonido, revisa el estado de una variable y se encarga de reproducir o pausar el sonido dependiendo de

su valor. En ambos casos estas variables se inicializan a false, cambian su valor a true cuando se crea una nueva pieza o se elimina una fila, y vuelven a false cuando el sonido comienza a reproducirse.

Por último se crea otro sonido diferente propio del modo de juego **TEST DE REFLEJOS** ([4.7](#)). Este sonido se activa cuando el usuario deja escapar una pieza y esta desaparece. Para introducirlo en la escena se procede nuevamente al uso de un manejador que realiza los mismos pasos que en los casos anteriores. En este caso el estado de la variable que revisa el manejador cambia su valor a true cuando una pieza desaparece de la escena.

El visualizado de las imágenes y la reproducción de los archivos de sonido no está disponible en una versión local del proyecto. Esto se debe a que estos elementos son archivos y una versión local de la aplicación es incapaz de acceder a ellos. Por ello es necesario desplegar la aplicación en un servidor web y alojar en dicho servidor tanto el código de la aplicación como los archivos necesarios.

Existen múltiples formas de levantar un servidor web externo, pero en este caso se recurre a la herramienta *Visual Studio Code* ([2.2.4](#)), un editor de texto que posee la funcionalidad de levantar un servidor web donde se puede alojar el proyecto que está en edición. Para ello es necesario colocarse en el archivo que contiene el menú principal de la aplicación, y pulsar el botón *Go Live* que aparece en la esquina inferior derecha de la pantalla del editor. Esta acción levanta el servidor iniciando la aplicación desde el menú principal, cargando a su vez los elementos de imagen y sonido.

4.9.3. Prototipo Final

El resultado de esta etapa final es la aplicación completa, un videojuego inspirado en el Tetris y adaptado a los nuevos entornos de realidad virtual, una aplicación que consta de una interfaz gráfica y múltiples modos de juego con efectos audiovisuales incorporados.

Durante el desarrollo de esta última etapa se adquieren conocimientos sobre nuevos tipos de elementos que posee A-Frame, así como los elementos *a-image* y *audio*. Se aprende a incorporar estos nuevos elementos a la escena principal y acceder a ellos desde otros componentes, así como nuevas funciones como la de reproducir o pausar un sonido.

También se aprende a manejar la herramienta *Visual Studio Code* ([2.2.4](#)), un editor de textos

muy potente capaz de sustituir al que se había utilizado para dar forma a este proyecto (*Atom* (2.2.3)). Esta herramienta incorpora nuevas funcionalidades como la de desplegar un servidor web directamente desde el editor o una consola para depurar rápidamente el código.

El resultado final es la propia aplicación lista para ser utilizada y desplegada en GitHub a través de su herramienta GitHub Pages, accesible a través de la siguiente URL: [Inicio de la aplicación](#).

Capítulo 5

Conclusiones

Una vez finalizado el proyecto se pueden sacar ciertas conclusiones respecto al estado actual de la realidad virtual en el entorno de las aplicaciones gracias a la aplicación práctica desarrollada.

5.1. Conclusiones generales

La principal conclusión extraída de este desarrollo es que las nuevas tecnologías están cada vez más presentes en las aplicaciones actuales y más concretamente en el mundo de los videojuegos, sin embargo al tratarse de una idea tan moderna siguen existiendo multitud de características que todavía no se han desarrollado o explorado. Despues de un análisis del marco en el que se encuentra la realidad virtual hoy en día, he descubierto que cada vez son más las aplicaciones que incorporan esta tecnología pero aún así siguen siendo una gran minoría respecto a las que solo incorporan tecnologías tradicionales. Analizando el potencial que tiene la realidad virtual, creo que esta situación cambiará drásticamente en un futuro no muy lejano y se empezarán a ver cada vez más aplicaciones y videojuegos de este tipo.

Centrando la atención en el mundo que engloba los juegos arcade, tengo la impresión de que estos juegos tuvieron su foco de esplendor hace bastantes años y con el paso del tiempo han quedado bastante relegados por los videojuegos de nueva generación. La posibilidad de incorporar las nuevas tecnologías a la funcionalidad básica de estos juegos abre una nueva oportunidad para que este tipo de juegos vuelvan a crecer y se coloquen de nuevo en el foco principal. Este proyecto incorpora un ejemplo que muestra cómo una aplicación tan clásica

como el Tetris puede pasar a ser una aplicación mucho más dinámica y colorida, adaptada a los nuevos tiempos y los gustos de los nuevos usuarios.

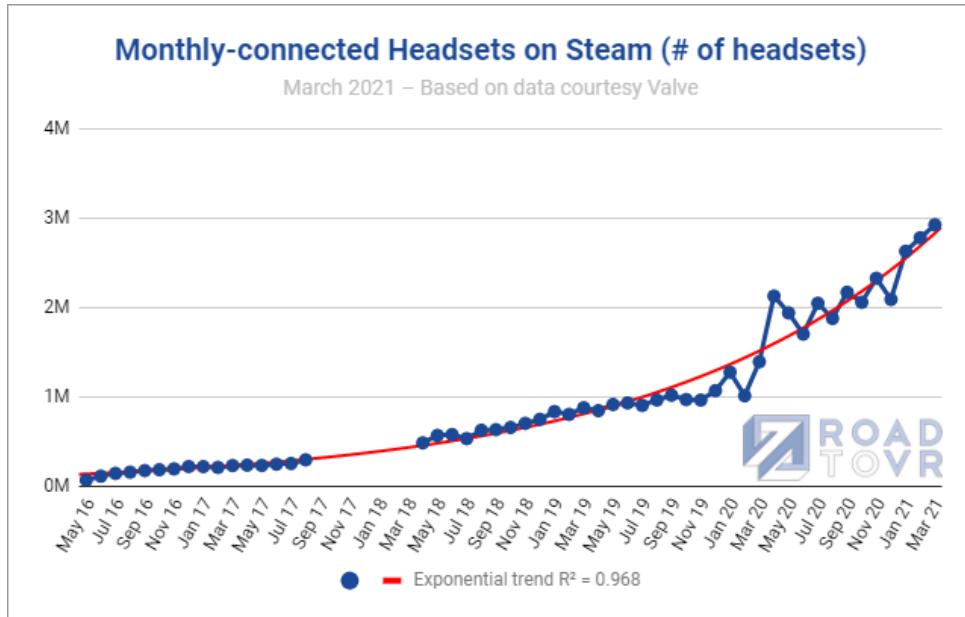


Figura 5.1: Evolución de la realidad virtual en el mundo de los videojuegos

Como aspecto negativo estas nuevas tecnologías presentan a mi parecer dos principales inconvenientes que dificultan bastante el desarrollo de las aplicaciones. Uno de los problemas es la escasez de documentación debido al poco tiempo que llevan funcionando. Durante el desarrollo de la aplicación han ido surgiendo ciertos problemas que han llevado cierto tiempo solucionar debido a la dificultad a la hora de encontrar la solución en aplicaciones externas. El otro problema es la incompatibilidad de los dispositivos de realidad virtual y las dificultades de integración con otro tipo de dispositivos. Uno de los problemas más importantes del desarrollo ocurrió al intentar conectar las gafas de realidad virtual a mi ordenador personal, debido principalmente a los problemas de compatibilidad y a la necesidad de descargar ciertas aplicaciones.

Pese a ello considero que se deberían invertir muchos esfuerzos en estas tecnologías porque se pueden crear aplicaciones con funcionalidades inimaginables hace un par de años y su potencial combinación con aplicaciones ya creadas puede proporcionar una enorme mejora. En lo que refiere a mí, me gustaría seguir investigando este mundo y probablemente esta no sea la última aplicación en realidad virtual que diseñe.

5.2. Consecución de objetivos

El objetivo principal de este proyecto consistía en la exploración de las nuevas posibilidades que podría ofrecer la adaptación de aplicaciones ya existentes a los entornos de realidad virtual. Esta exploración se ha llevado a cabo de forma teórica mediante un estudio del marco actual de la realidad virtual y su encaje en el mundo de los videojuegos, y de forma práctica mediante la creación de una aplicación que funciona en entornos de realidad virtual. Por ello se puede considerar que el objetivo principal de la aplicación se ha conseguido.

También se presentaron una serie de objetivos secundarios o específicos cuya consecución ayudaría a lograr el objetivo final. El estado de todos ellos se presenta a continuación:

- Se ha estudiado el estado actual de la realidad virtual, analizando su historia, los productos que ofrece hoy en día o su encaje en las aplicaciones existentes.
- Se ha escogido una herramienta actual para la creación de aplicaciones en entornos de realidad virtual (A-Frame) y se ha llevado a cabo un análisis en profundidad de la herramienta para poder diseñar una aplicación completa.
- Se han observado los distintos tipos de aplicaciones que se podrían crear con el uso de estas tecnologías hasta llegar a la decisión de que la adaptación de un videojuego arcade a la realidad virtual era la mejor opción.
- Se ha conocido la historia de los videojuegos arcade y se ha descubierto que necesitaban un punto de modernidad y nueva funcionalidad para volver a ser punteros en su género.
- Se ha escogido el dispositivo *Meta Quest 2* para el desarrollo de la práctica y se ha analizado su funcionamiento para ver de qué forma podía ser incluído en la aplicación.
- Se ha desarrollado un videojuego arcade capaz de funcionar tanto en entornos de realidad virtual como en entornos de escritorio.
- Se han diseñado múltiples modos de juego en la aplicación, con el fin de que cada uno de ellos incorpore nuevas características que doten a la aplicación de una gran variedad.
- Se ha incorporado una interfaz gráfica y unos efectos audiovisuales a la aplicación que mejoran la experiencia de usuario y acercan a esta aplicación a lo que sería un videojuego arcade tradicional.

- Se ha desplegado la aplicación en un servidor web externo mediante la herramienta *GitHub Pages* para hacerla accesible mediante una URL.

5.3. Planificación temporal

Inicié el desarrollo del proyecto hacia el mes de Octubre del año 2021, por lo que el tiempo natural dedicado al mismo es cercano a los dos años. Debido a mi situación laboral, el tiempo estimado de esfuerzo que he dedicado al proyecto es de tres o cuatro horas a la semana, aunque ha habido periodos en los que he podido dedicarle más tiempo y otros en los que no he podido dedicarme a ello. Por hacer una estimación aproximada del número total de horas dedicadas al proyecto podría decir que he trabajado en él unas 250-300 horas aproximadamente.

Al inicio del desarrollo dediqué un par de meses a familiarizarme con la tecnología A-Frame, creando pequeñas aplicaciones que me ayudaran a entender su funcionalidad. Este tiempo también lo utilicé para comprender cómo interactuaban entre sí los distintos componentes y cómo podía manipularlos utilizando el DOM.

Cuando conseguí la suficiente soltura para crear mi proyecto decidí la temática del mismo y comencé a desarrollarlo. Lo primero fue crear los modos de escritorio, lo cual supuso un esfuerzo de alrededor de seis meses, dado que en ese tiempo tuve que aprender a utilizar las físicas. Más tarde generé los modos de realidad virtual en dos meses aproximadamente.

Con la funcionalidad principal de la aplicación terminada dediqué otro par de meses a corregir fallos, añadir los menús de la aplicación, los efectos audiovisuales y adaptar alguno de los modos a diferentes tamaños de tablero.

Por último me dispuse a escribir esta memoria para concluir con el proyecto. A este trabajo final tuve que dedicarle algo más de medio año para poder terminarla por completo.

5.4. Aplicación de lo aprendido

1. **Fundamentos de la Programación.** En esta asignatura adquirí los conceptos básicos de programación que existen en prácticamente cualquier lenguaje de programación, así como declaración de variables, estructuras condicionales, bucles... En esta aplicación ha sido útil a la hora de desarrollar la lógica de la aplicación mediante JavaScript.

2. **Servicios y Aplicaciones Telemáticas.** Esta asignatura me ayudó a entender la forma en que se debe crear una aplicación completa partiendo desde cero. Me ha ayudado a la hora de estructurar la práctica, la organización del desarrollo en sprints con objetivos específicos o la creación de pantallas y menús intermedios que enlacen toda la aplicación.
3. **Desarrollo de Aplicaciones Telemáticas.** Durante el desarrollo de esta asignatura fue donde obtuve los conocimientos necesarios para trabajar con el lenguaje HTML y el lenguaje JavaScript, los dos lenguajes principales de programación web. La aplicación está completamente basada en estos dos lenguajes, por lo que el desarrollo hubiera sido mucho más difícil sin estos conceptos.

5.5. Lecciones aprendidas

1. **Manejo del DOM.** Lo más importante que he aprendido durante este proyecto es el manejo del DOM. El DOM está presente en todas las páginas web y estos conocimientos permiten el acceso y manipulación de los elementos de cualquier página web a través de él.
2. **Manejo de A-Frame.** A-Frame era una herramienta completamente desconocida para mí y a partir de esta aplicación podría ser capaz de desarrollar cualquier tipo de aplicación que utilizase esta herramienta.
3. **Conocimientos sobre la webXR.** Tampoco sabía que una aplicación web podía ser desplegada en una URL en un entorno de realidad virtual. Esto es posible gracias a la existencia de la WebXR y lo he aprendido durante el transcurso de este proyecto.
4. **Creación de una aplicación completa.** Esta es una de las primeras aplicaciones completas que he creado desde cero y ha sido la primera que he conseguido crear utilizando únicamente HTML y JavaScript como lenguajes de programación principales.
5. **Conocimiento de nuevas tecnologías o herramientas.** He conocido durante el desarrollo de este proyecto nuevas tecnologías o herramientas como LaTeX, Visual Studio Code o las aplicaciones relativas a la tecnología Oculus.

5.6. Trabajos futuros

Gracias a los conocimientos adquiridos en este proyecto podría ser capaz de realizar algún otro desarrollo más ambicioso en un futuro no muy lejano. Estas podrían ser alguna de las opciones:

1. La creación de un videojuego mucho más moderno y separado de los juegos tradicionales, que incorpore unos gráficos mucho más avanzados que incorporen formas propias y no solo elementos geométricos.
2. El desarrollo de una aplicación en realidad virtual a través de una herramienta distinta de A-Frame, que permita algún tipo de funcionalidad distinta y un mayor grado de libertad a la hora de desarrollarla.
3. Utilizar algún lenguaje más avanzado para hacer más robusta la aplicación y dotarla de un *back-end*.
4. Transformar la aplicación web es una aplicación alojada en un servidor continuo, es decir, la aplicación no necesitará levantarse en un servidor cada vez que sea desplegada, si no que permanecerá accesible permanentemente en un servidor web.

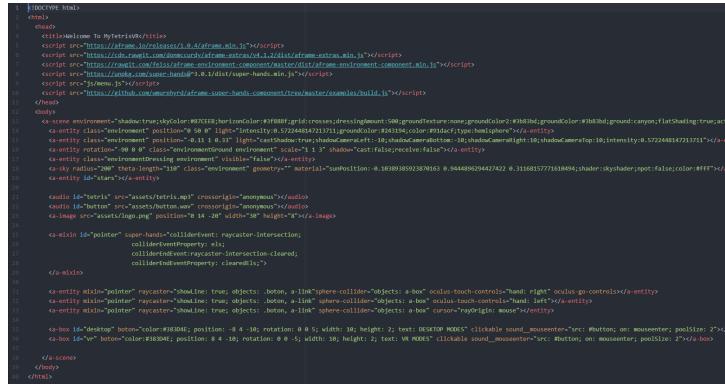
5.7. Material de interés

- [Página web](#)
- [Presentación diapositivas](#)
- [Menú principal de la aplicación completa](#)
- [Demo rebotadores \(fase 0\)](#)
- [Demo prototipo \(fase 1\)](#)

Capítulo 6

Anexo: Elementos A-Frame

La arquitectura de este proyecto está basada en pantallas creadas a partir de código HTML en el que se declaran elementos propios de la herramienta A-Frame. Estos elementos se declaran de la misma forma que las etiquetas clásicas que utiliza HTML pero están provistos de propiedades y características diferentes a ellas.



The screenshot shows a code editor displaying an HTML file with A-Frame specific elements. The code includes imports for various scripts and stylesheets, defines a scene with a sky, a ground, and a camera, and contains several entities like a cube, a button, and a text element. The entities have various properties such as position, rotation, scale, and material settings. The code is well-structured with proper indentation and comments.

```
<!DOCTYPE html>
<html>
  <head>
    <title>My First Scene</title>
    <script src="https://aframe.io/releases/1.0.4/aframe.min.js"></script>
    <script src="https://cdn.rawgit.com/doncory/aframe-extens/ef1.2/dist/aframe-extens.min.js"></script>
    <script src="https://cdn.rawgit.com/filatov/aframe-environment-concurrent/master/assets/list/aframe-env-and-cmment-component_min.js"></script>
    <script src="https://cdn.rawgit.com/filatov/aframe-environment-concurrent/master/assets/list/aframe-env-and-cmment-component_min.js?r=1.2.0"></script>
    <script src="https://aframe.io/releases/1.0.4/aframe-hands.min.js"></script>
    <script src="https://aframe.io/releases/1.0.4/aframe-super-hands.js"></script>
    <script src="https://github.com/mrdoob/aframe-examples-build_18"></script>
  </head>
  <body>
    <a-entity environment="color:#000000"></a-entity>
    <a-sky radius="200" theta-length="100" class="environment geometry" material="substitution:-0.18390385038708165,0.344682984427422,0.31368157775616094; shader:skyShader; spot: false; color:#fff"></a-sky>
    <a-entity id="star"></a-entity>
    <audio id="text1" src="assets/text1.mp3" crossorigin="anonymous"></audio>
    <audio id="button" src="assets/button.wav" crossorigin="anonymous"></audio>
    
    <a-mixin id="pointer" super="hand"><collider>raycaster</collider><raycaster>raycastIntersection</raycaster><intersection>colliderEvent raycasterIntersection cleared</intersection><intersection>colliderEvent raycasterIntersection cleared</intersection></a-mixin>
    <a-entity mixin="pointer" raycaster="showLine: true; objects: box, a-link"><sphere-collider>objects: a-box oculus-touch-controls="hand: right" oculus-go-controls</sphere-collider></a-entity>
    <a-entity mixin="pointer" raycaster="showLine: true; objects: box, a-link"><sphere-collider>objects: a-box oculus-touch-controls="hand: left" oculus-go-controls</sphere-collider></a-entity>
    <a-box id="box1" position="0 0 0" width="10" height="10" depth="10" rotation="0 0 0" material="color:#000000; opacity: 0.5; visibility: hidden; "></a-box>
    <a-link id="link1" position="0 0 0" rotation="0 0 0" width="10" height="10" depth="10" material="color:#000000; opacity: 0.5; visibility: hidden; "></a-link>
    <a-box id="box2" position="0 0 0" width="10" height="10" depth="10" rotation="0 0 0" material="color:#000000; opacity: 0.5; visibility: hidden; "></a-box>
    <a-link id="link2" position="0 0 0" rotation="0 0 0" width="10" height="10" depth="10" material="color:#000000; opacity: 0.5; visibility: hidden; "></a-link>
    <div style="position: absolute; top: 0; left: 0; width: 100%; height: 100%; background-color: black; opacity: 0.5; z-index: 1000; display: flex; align-items: center; justify-content: center; font-size: 2em; color: white; font-weight: bold;">A
```

Figura 6.1: Ejemplo de documento HTML de una página de la aplicación

Durante el desarrollo de la aplicación se han empleado muchos tipos de estos elementos y cada uno de ellos posee unas características que se explican a continuación:

- **a-scene**. El elemento a-scene es el componente principal de una escena, el encargado de englobar el resto de elementos presentes en la pantalla. En toda página que utilice A-Frame debe haber un elemento de este tipo y dentro de él se crean el resto de elementos que forman la escena.

A parte de esta funcionalidad también se encarga de dar forma al ecosistema de la panta-

lla, lo que sería el fondo de pantalla. Puede declararse sin especificar ninguna propiedad, de forma que la pantalla tendrá un fondo estándar, pero también se puede declarar la propiedad *environment* y darle unos valores específicos para personalizar este fondo y adecuarlo a las preferencias del desarrollador. Esta propiedad puede modificar valores como el color del cielo y del suelo, la presencia de sombras, la intensidad de luces, la forma del suelo o la presencia de elementos en el suelo y la frecuencia con la que aparecen.

También se puede declarar otra propiedad llamada *sound* para dotar a la escena de un sonido principal que puede ser constante o intermitente.

Desciende de la clase *Entity*, lo que le otorga la capacidad para enlazar componentes y esperar hasta que se carguen todos los elementos y assets que desciendan de ella para comenzar el ciclo de renderización.

- **a-entity**. Este elemento se define como una entidad de la escena y se utiliza para declarar un componente genérico al que se darán unas propiedades específicas una vez sea registrado como componente A-Frame en la escena.

Utilizando este elemento se pueden crear todo tipo de componentes dándole las propiedades adecuadas. Se pueden crear elementos visibles en la escena tales como cubos, esferas o cilindros utilizando las propiedades *geometry* y *material*, o elementos que simplemente contengan texto. También se pueden crear elementos que no sean visibles en la escena como los elementos que contienen la luz y las sombras del fondo de pantalla utilizando las propiedades *light* y *shadow*. Por último también se pueden declarar entidades que no sean visibles en la escena ni tengan una funcionalidad específica, si no que pueden contener otros elementos o entidades dentro de ellos y facilitar el acceso a ellos mediante el DOM.

Una utilidad muy destacada de este tipo de elementos en este desarrollo ha sido la declaración del cursor. Este elemento es fundamental para la interacción del usuario con la aplicación y se ha declarado mediante este elemento. Para la creación del cursor se han utilizado diferentes propiedades que especifican el uso de la biblioteca super-hands, el tipo de elementos con los que debe interactuar (*raycaster*) y el tipo de cursor que se está declarando según sea un ratón (*cursor*) o controladores de realidad virtual (*oculus-touch-controls*).

```

<a-mixin id="pointer" super-hands="colliderEvent: raycaster-intersection;
    colliderEventProperty: elIs;
    colliderEndEvent: raycaster-intersection-cleared;
    colliderEndEventProperty: clearedElIs;">

</a-mixin>

<a-entity mixin="pointer" raycaster="showLine: true; objects: .boton, a-link" sphere-collider="objects: a-box" oculus-touch-controls="hand: right" oculus-go-controls></a-entity>
<a-entity mixin="pointer" raycaster="showLine: true; objects: .boton, a-link" sphere-collider="objects: a-box" oculus-touch-controls="hand: left"></a-entity>
<a-entity mixin="pointer" raycaster="showLine: true; objects: .boton, a-link" sphere-collider="objects: a-box" cursor="rayOrigin: mouse"></entity>

```

Figura 6.2: Declaración del cursor

Este elemento es por lo tanto el más genérico que proporciona A-Frame y no tiene una estructura determinada, si no que todo su potencial se encuentra tanto en la declaración de sus propiedades como en las funciones que hereda una vez registrado como componente.

- **a-sky.** Este elemento se encarga simplemente de crear la parte del cielo asociada al fondo de pantalla. Posee múltiples propiedades que ayudan a modificar el diseño y la estructura del cielo, pero lo más importante a destacar es que este elemento puede tratarse de un color o una fotografía. Para formar el cielo a partir de una fotografía hay que declarar la propiedad *src* y darla el valor del identificador del elemento que contiene la fotografía que se quiere utilizar, mientras que para partir de un color simplemente habrá que definir la propiedad *color* y darla el valor deseado.
- **audio.** Este elemento sirve para introducir archivos de audio en una escena. Para ello simplemente es necesario definir su propiedad *src* e introducir en ella la ruta donde se encuentra el archivo de audio. Cabe destacar que en este momento el audio se encuentra en la escena pero no se reproducirá hasta que algún elemento de la escena acceda a él y lo reproduzca. Por ello es fundamental declarar este elemento siempre acompañado de un identificador, de cara a facilitar su acceso posterior.
- **a-image.** Este elemento sirve para introducir archivos de imagen en la escena. Funciona exactamente igual que el elemento *audio*, por lo que será indispensable definir su propiedad *src* y otorgarle un identificador. Como diferencia respecto al audio, este elemento también presenta otras propiedades que pueden alterar la estructura de la imagen, así como la posición, el tamaño o la opacidad.
- **a-mixin.** Este elemento no representa ningún tipo de estructura visible en una escena, si no que se utiliza para englobar un conjunto de propiedades comunes a varios elementos. Para ello se definen estas propiedades dentro del elemento acompañadas de un identifi-

fificador. Una vez creado el elemento, se define la propiedad *mixin* con el valor de su identificador en cada uno de los elementos que vayan a contener estas propiedades.

Este tipo de elemento se ha utilizado para la incorporación de la biblioteca super-hands a la aplicación. En primer lugar se ha usado para indicar que cada uno de los tres controladores de la aplicación iba a utilizar dicha biblioteca, y más tarde para definir qué tipo de eventos propios de super-hands debía manejar cada elemento de la aplicación.

- **a-plane.** Este elemento crea un plano visible en la escena, es decir, una estructura rectangular que se extiende sobre dos de los ejes y queda fijo sobre el tercero. Se utiliza para crear estructuras bidimensionales o sin profundidad. Presenta propiedades como la altura o la anchura, la posición del plano, el color o la rotación.

En este caso se han utilizado planos para crear la estructura sobre la que se mueve el controlador de las piezas, debido a que el controlador está diseñado para moverse únicamente de forma horizontal por lo que no se necesita una estructura tridimensional.

- **a-box.** Este es probablemente el elemento más utilizado en A-Frame y se utiliza para crear estructuras cuadradas o rectangulares con profundidad, es decir, tridimensionales. Utiliza las mismas propiedades que el elemento *a-plane* ya que son elementos muy similares, sin embargo este elemento incorpora la propiedad *depth* para indicar la profundidad que tendrá dicha estructura.

Se utiliza para crear todo tipo de componentes debido a que la mayoría de componentes en A-Frame suelen ser tridimensionales y con forma rectangular. Suele ser empleado para crear pequeñas estructuras tales como cubos o cajas, pero también puede ser empleado para crear estructuras de gran tamaño como paredes o planos alterando sus correspondientes propiedades.

En esta aplicación se ha utilizado este elemento para crear diferentes tipos de estructuras. En primer lugar las paredes del tablero se han creado a partir de cubos en los que se ha extendido mucho la altura sobre la anchura, o al contrario. También se han creado todos los botones de la aplicación introduciendo el texto dentro de ellos. Por último se ha utilizado para generar las piezas que caen sobre el tablero formando cubos de diferentes tamaños, modificando sus propiedades de altura y anchura.

- **a-sphere.** Este elemento se utiliza para crear estructuras tridimensionales con forma circular o esférica. Como contrapunto a las estructuras rectangulares que se han presentado anteriormente, estos elementos presentan una propiedad diferente para cambiar sus dimensiones. Esta propiedad se llama *radius* y establece el radio de la esfera. También presenta propiedades comunes a los demás elementos como la posición, la rotación o el color.

En esta aplicación se ha utilizado únicamente para crear el controlador de las piezas. Utilizando este elemento se crea la bola que se mueve sobre el plano y cuya posición determina el movimiento horizontal que ejecutan las piezas mientras caen sobre el tablero.

- **a-text.** Este elemento se utiliza para incorporar texto a un elemento determinado. Presenta distintas propiedades pero se puede destacar la propiedad *value* en la que se indica el texto que se quiere mostrar y la propiedad *align* en la que se establece si el texto debe colocarse centrado o alineado a alguno de los lados del elemento que lo contiene.

En este caso se ha utilizado únicamente para crear el texto del botón que se utiliza para frenar las bolas en el desarrollo que tuvo lugar en la fase 0 del proyecto. Más tarde se ha ido incorporando texto de otra forma diferente, mediante la inclusión de la propiedad *text* en otro tipo de elementos como *a-entity* o *a-box*. Ambas formas de crear texto presentan el mismo resultado y las mismas propiedades (anchura y altura del texto, color, tipo de fuente...), por lo que se puede emplear cualquiera de ellas.

Capítulo 7

Anexo: Componentes y sus propiedades

Este es un listado de todos los componentes usados durante el desarrollo de esta aplicación y las distintas propiedades que posee cada una de ellos:

- **boton** - [position, rotation, width, height, color, text];
- **botonback** - [position, rotation, width, height, color, text];
- **tablero** - [alturaSuelo, positionSuelo, colorTablero, alturaPared, anchuraPared, positionParedIzq, positionParedDer];
- **rotarpieza** - [position, width, height, color, id, mixin, positionText, colorText, alignText, valueText, widthText];
- **bajarpieza** - [position, width, height, color, depth, id, mixin, positionText, colorText, alignText, valueText, widthText];
- **score** - [position, anchuraTexto, alturaTexto];
- **mando** - [position, height, rotation, color, id];
- **controller** - [position, radius, rotation, mixin, color, id];
- **cubo** - [position, width, height, velocidad];
- **obstaculo** - [position, width, height, color];

- **botonestableros** - [positionIzq, positionDer, width, height, color, depth, idIzq, idDer, mixin, positionTextIzq, positionTextDer, colorText, alignText, valueText, widthText];
- **handler nuevapieza** - []
- **handler filaeliminada** - []
- **handler nuevotablero** - []
- **handler piezasaltada** - []

Bibliografía

- [1] A-Frame. Documentación de A-Frame. <https://aframe.io/docs/1.4.0/introduction/>.
- [2] A-Frame. Prácticas con A-Frame. <https://aframe.io/examples/showcase/helloworld/>.
- [3] A-Frame. Visual Inspector Dev Tools. <https://aframe.io/docs/1.4.0/introduction/visual-inspector-and-dev-tools.html>.
- [4] AdictosAlTrabajo. Introducción a WebVR. <https://www.adictosaltrabajo.com/2017/06/28/aframe-bienvenido-a-webvr/>.
- [5] AprenderAProgramar. JavaScript del lado del servidor. https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=778&catid=78&Itemid=206.
- [6] Atlassian. Información sobre Git. <https://www.atlassian.com/es/git/tutorials/what-is-git>.
- [7] Atom. Documentación Atom. <https://github.com/atom/atom>.
- [8] Blog-Hubspot. El DOM integrado con JavaScript. <https://blog.hubspot.es/website/que-es-dom>.
- [9] Brais-Moure. Git y GitHub desde cero. https://books.google.es/books?id=-la7EAAAQBAJ&printsec=frontcover&dq=github+libros&hl=es&newbks=1&newbks_redir=0&sa=X&redir_esc=y#v=onepage&q&f=false.
- [10] César-Mesa. Historia de Oculus. <https://cesarmesa.com.co/la-historia-de-oculus-que-es-y-como-funciona/>.
- [11] Desarrollo-Web. Información extra sobre el DOM. <https://desarrolloweb.com/articulos/que-es-el-dom.html>.

- [12] Desarrollo-Web. JavaScript como lenguaje integrador. <https://desarrolloweb.com/home/javascript>.
- [13] Developer-Oculus. Meta Quest Developer Hub. <https://developer.oculus.com/documentation/unity/ts-odh/>.
- [14] GitHub. Librería Super-Hands. <https://github.com/c-frame/aframe-super-hands-component>.
- [15] GitHub. Ventajas de GitHub Desktop. <https://docs.github.com/en/desktop/installing-and-configuring-github-desktop/overview/getting-started-with-github-desktop>.
- [16] Hostinger. Información extra sobre HTML. https://www.hostinger.es/tutoriales/que-es-html%C2%BFComo_funciona_HTML.
- [17] InternetPasoAPaso. Historia de los juegos arcade. <https://internetcasoapaso.com/maquinas-arcade/>.
- [18] JavaScript-info. El Tutorial de JavaScript Moderno. <https://es.javascript.info/>.
- [19] Javier-Salinas. ¿Qué es la WebVR? <https://javiersalinas.es/que-es-la-webvr/>.
- [20] Michelle-Haley. Meta Quest 2 Beginners Handbook (2023). https://books.google.es/books?id=7q7GzwEACAAJ&dq=oculus+meta+quest&hl=es&newbks=1&newbks_redir=0&sa=X&redir_esc=y.
- [21] Oculus. Información sobre Oculus Developer Hub. <https://developer.oculus.com/blog/oculus-developer-hub-20/>.
- [22] Open-Webinars. Información sobre HTML. <https://openwebinars.net/blog/que-es-html5/>.
- [23] Rakesh-Baruah. AR and VR Using the WebXR API. https://books.google.es/books?id=oZ2uzQEACAAJ&dq=webxr&hl=es&newbks=1&newbks_redir=0&sa=X&redir_esc=y.
- [24] Russian-University. Historia del Tetris. <https://www.russian-university.com/la-historia-del-tetris-inventado-por-un-ruso>.

- [25] Samuel-Dauzon. Controle la gestión de sus versiones. https://books.google.es/books?id=kAP-5cbnmPYC&printsec=frontcover&dq=git&hl=es&newbks=1&newbks_redir=0&sa=X&redir_esc=y#v=onepage&q=git&f=false.
- [26] Stefan-Kottwitz. LaTeX Beginner's Guide. https://books.google.es/books?id=L95BEAAAQBAJ&printsec=frontcover&dq=l%C3%A1tex+overleaf&hl=es&newbks=1&newbks_redir=0&sa=X&redir_esc=y#v=onepage&q=l%C3%A1tex%20y%20overleaf&f=false.
- [27] Tania-Rascia. Understanding the DOM — Document Object Model. : [//www.google.es/books/edition/Understanding_the_DOM_Document_Object_Mo/fNQBEAAAQBAJ?hl=es&gbpv=0](http://www.google.es/books/edition/Understanding_the_DOM_Document_Object_Mo/fNQBEAAAQBAJ?hl=es&gbpv=0).
- [28] Visual-Studio. Información sobre Visual Studio Code. <https://code.visualstudio.com/docs>.
- [29] Xataka. Información sobre GitHub. <https://www.xataka.com/basics/que-github-que-que-le-ofrece-a-desarrolladores>.