Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

Posibilidades de la Realidad Virtual para mejorar la accesibilidad en desarrollos realizados con Unity

Autor: Sergio Sánchez López

Tutores: José María Gutiérrez Martínez
Juan Aguado Delgado

2018

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior
Grado en Ingeniería Informática

Trabajo de Fin de Grado

"Posibilidades de la Realidad Virtual para mejorar la accesibilidad en desarrollos realizados con Unity"

Autor: Sergio Sánchez López

Tutor: José María Gutiérrez Martínez

Cotutor: Juan Aguado Delgado

Tribunal:

Presidente: .....

Vocal 1º: .....

Vocal 2º: .....

Calificación: .....

Fecha: ..... de ..... de .....

Palabras clave

Realidad virtual, Unity, Aplicaciones móviles, Accesibilidad en videojuegos.

Keywords

Virtual reality, Unity, Mobile apps, Accessibility in videogames

Resumen corto

El proyecto consiste en realizar un análisis de las librerías y tecnologías de realidad virtual aplicables en desarrollos realizados con el motor Unity. Mediante el uso de la librería escogida, GoogleVR para Cardboard, se ha desarrollado un videojuego móvil mediante el motor Unity que aplica técnicas de realidad virtual siguiendo directrices de accesibilidad para que pueda ser utilizada por personas con diversos perfiles de diversidad funcional.

Short summary

The proyect consists in the analysis of the available virtual reality libraries and technologies that can be applied to developments done using the Unity engine. Through the use of the chosen technology, which is GoogleVR for Cardboard, a mobile videogame is developed using Unity, which makes use of VR techniques following accessibility guidelines to ensure the ability of people of different functional diversity profiles to use the application.

Agradecimientos

A todos mis compañeros de N31 por los buenos momentos que hemos pasado y por el apoyo que hoy en día sigo recibiendo; y sobre todo a José María, que me dio la oportunidad de tenerlos como compañeros.

A mis amigos Adri, Tony, Alberto y María por haberme apoyado en lo bueno y en lo malo y por ayudarme a desconectar cuando lo necesitaba.

A Sergio por haberme dado la motivación necesaria para seguir adelante con todo lo que me he propuesto, incluyendo este trabajo.

Y a mi *yo* de hace algo más de 4 años, que tuvo la decisión de empezar este camino que es la Ingeniería Informática, y a mi familia por hacerlo posible.

Índice resumido

Índice

1 Introducción

La realidad virtual proporciona unas oportunidades fantásticas para personas con discapacidad, tales como nuevas experiencias, beneficios terapéuticos, e incluso mejora la interacción con los videojuegos y aplicaciones para gente que tiene un mejor control con la cabeza que con las manos.

Sin embargo, la realidad virtual trae consigo misma nuevas barreras, con bastante potencial para evitar que ciertos grupos de personas puedan aprovecharse de estos beneficios.

Algunas de estas barreras que trae la realidad virtual son inevitables, pues hay gente que sencillamente no podrán tomar parte de la RV tal y como es en el estado actual, siendo por ejemplo personas que no pueden adaptarse a la realidad virtual y sufren mareos constantes, o personas que no pueden sostener de forma física un dispositivo pesado sobre su cabeza.

A pesar de esto, existen muchas otras barreras que se pueden evitar, a través de las consideraciones de diseño apropiadas: a través de la accesibilidad.

El último censo oficial sobre personas con discapacidad en España realizado por el Instituto Nacional de Estadística, publicado en 2008 y titulado ‘Discapacidad, autonomía personal y situaciones de dependencia’, estima que hay 3,48 millones de personas dentro de distintos perfiles de diversidad funcional, lo que representa aproximadamente un 8,5% de la población [1].

A pesar de ser una cifra tan elevada, en el desarrollo de videojuegos y otras aplicaciones interactivas no se ofrecen soluciones para que estas personas puedan disfrutarlos. De hecho, muchas de las veces las dificultades que le suponen a ciertos grupos de personas manejar una aplicación suelen ser muy fáciles de solventar siempre que sea en la fase de análisis y diseño.

Este trabajo intenta establecer un enlace entre el diseño accesible y las características ofrecidas por la realidad virtual actualmente, además de las cuestiones intrínsecas del desarrollo de un videojuego.

Se espera que este proyecto sirva como una experiencia de aprendizaje personal para poder tener en cuenta en todos los proyectos futuros las diferentes barreras que personas con diversidad funcional pueden encontrar, para así poder mejorar en la medida de lo posible la experiencia del usuario, tanto en el disfrute como en la utilidad de aplicaciones, videojuegos y otros sistemas.

2 Objetivo

El objetivo principal del trabajo a desarrollar es la creación de un videojuego basado en tecnologías de realidad virtual móvil mediante el motor Unity, centrándose en las características de accesibilidad.

Para ello, durante la realización de este trabajo se busca aplicar las guías de accesibilidad existentes y otras recomendaciones sobre accesibilidad, siendo el desarrollo marcado por el seguimiento de directrices de accesibilidad para que así la aplicación pueda ser utilizada por el mayor número de perfiles de personas con diversidad funcional.

Además, se quiere conocer el grado de cumplimiento posible de estas directrices, y descubrir los límites de adaptación de la aplicación sin minimizar el disfrute y/o funcionalidad ofrecido al usuario; es decir, se busca que la aplicación sea usable por el máximo espectro de usuarios con diversidad funcional y que la experiencia sea igual de satisfactoria para todos los perfiles de usuarios, en la medida de lo posible.

En lo que a diseño de videojuegos se refiere, se busca aprender a realizar un diseño de un videojuego que sea entretenido, con objetivos marcados, y que se pueda aplicar conceptos de rejugabilidad, teniendo siempre en cuenta las diferentes barreras que sufren las personas con diferentes perfiles de discapacidad, para así conocer qué ideas de diseño deben descartarse o ser alteradas a favor de la accesibilidad para sortear esas barreras presentes en otras aplicaciones del mercado.

Más centrado a nivel técnico, se marca como objetivo específico el aprendizaje del uso de la herramienta de desarrollo de videojuegos y aplicaciones Unity, aprendiendo sobre sus características, estructura, flujo de trabajo, y programación en el lenguaje C# y ShaderLab; y el estudio de las diferentes tecnologías soportadas por Unity para la creación de aplicaciones basadas en realidad virtual, centrándose en la herramienta RV seleccionada para el desarrollo.

3 Estado del arte

En este capítulo se va a profundizar en los dos pilares de este trabajo: el campo de la realidad virtual y el de la accesibilidad. Primero se explicarán las tecnologías existentes en el campo de la realidad virtual, tanto en el caso de la realidad virtual no móvil como la móvil. Esta es la más vinculante para este proyecto, pues comenta las características de Google VR, el framework utilizado a lo largo del mismo para realizar la aplicación móvil de VR mediante el uso de Cardboard. Después, se explican conceptos relacionados con la accesibilidad, los diversos perfiles de diversidad funcional, las guías de accesibilidad, la accesibilidad en videojuegos, y la accesibilidad aplicada en el caso de aplicaciones VR.

3.1 Realidad virtual

La realidad virtual es una experiencia interactiva generada por ordenador que toma lugar en un entorno simulado, que incorpora múltiples tipos de respuesta sensorial, como puede ser visual, auditiva o háptica (sentido del tacto). El objetivo de la realidad virtual es ofrecer un entorno inmersivo para el usuario.

La tecnología actual de realidad virtual se basa en el uso de headsets de VR o entornos proyectados para generar imágenes realistas, sonidos y otras sensaciones que simulan la presencia física del usuario en el entorno virtual. Mediante el uso de los headset VR, el usuario puede moverse para mirar alrededor del entorno virtual, moverse por él, e interactuar con los elementos de ese entorno. Este efecto suele estar creado por los cascos VR, que consisten en un dispositivo montado en la cabeza con una pequeña pantalla delante de los ojos con un sistema de lentes que simulan la profundidad del entorno y la visión periférica.

La realidad virtual comparte algunos elementos con la realidad aumentada (AR). La AR es un tipo de tecnología de realidad virtual que combina lo que ve el usuario en su entorno real con contenido digital generado por software. Las similitudes residen sobre todo en el tipo de sensores utilizados para detectar la posición del usuario a través del dispositivo.

Estos sensores suelen ser giroscopios y sensores de movimiento para realizar el seguimiento de la cabeza, manos, o posiciones del cuerpo. En los cascos de realidad virtual, se usan pequeñas pantallas para formar una visión estereoscópica, y en algunos casos se incluyen procesadores para manejar la información de los sensores y procesar las imágenes que se deben mostrar al usuario. En el caso de la realidad virtual móvil, todo esto suele suceder en un sólo dispositivo: el teléfono móvil.

En la actualidad, la realidad virtual ha experimentado un aumento en su visibilidad debido a la madurez que está alcanzando, debido a la apuesta de varias grandes compañías por esta tecnología, tales como Google, Samsung, HTC, Valve y Facebook (Oculus, adquirida por Facebook) [2].

Actualmente sus aplicaciones se centran sobre todo en el ámbito de los videojuegos, aunque la VR también se está aplicando en el campo militar (como puede ser en simuladores de vuelo [3]), en el campo de la

salud (terapias a personas con fobias y síndrome postraumático, permitiendo que el paciente se enfrente a ciertos desencadenantes emocionales y fisiológicos de una forma controlada [4]), en la ingeniería (herramientas de entrenamiento para formación [5]), entre otros.

3.1.1 Realidad Virtual no móvil.

Los principales competidores en lo que respecta a tecnologías de realidad virtual son los proyectos HTC Vive [6] y Oculus Rift [7], disponibles para PC, con la relativamente reciente incorporación de PlayStation VR [8], disponible para PlayStation 4.

HTC Vive

HTC Vive son las gafas de realidad virtual desarrolladas por HTC y Valve Corporation. El casco utiliza tecnologías de seguimiento en escala de habitación, lo que significa que es capaz de seguir el movimiento del usuario en un área relativamente amplia. Esto es posible gracias a las Vive Base Station, conocidas también como el sistema de seguimiento Lighthouse, que consiste en dos cajas negras que crean un espacio virtual de 360º en un radio de 4,5 metros aproximadamente. Incluye dos mandos que representan dentro de las simulaciones la mano izquierda y la mano derecha del jugador.

El visor tiene una tasa de refresco de 90 Hz y un campo de visión de 110 grados. Utiliza dos paneles OLED, uno para cada ojo, con una resolución de 1080x1200. Una característica interesante es la presencia de una cámara que apunta hacia delante del visor, que permite al usuario visualizar su entorno real sin tener que quitarse el visor. Esto también es aprovechado por el software para identificar objetos que se mueven o estáticos en la habitación, lo cual puede ser usado para avisar al usuario de un posible choque. Respecto a sensores, incluye un conjunto de sensores infrarrojos que interactúan con el sistema de posicionamiento Lighthouse, giroscopios, sensores de proximidad, y acelerómetros.

Oculus Rift

El visor usa un panel OLED para cada ojo, cada uno con una resolución de 1080x1200, al igual que las HTC Vive. Estos panales tienen una tasa de refresco de 90 Hz. Unas de las peculiaridades del visor es su refresco de pantalla global (que consiste en cambiar toda la imagen a la vez, en vez de hacerlo línea a línea), y la baja persistencia de imágenes (que evita que una misma imagen se vea durante más de 2 milisegundos), mejorando así la fluidez de la experiencia.

Para reducir los requisitos necesarios para utilizar Oculus Rift en el PC, se hace uso de la técnica de interpolación de imágenes para que, en caso de ordenadores que no puedan procesar imágenes a una tasa de 90 imágenes por segundo, simplemente tengan que producir 45 imágenes por segundo al aplicar una interpolación intermedia entre una imagen y la siguiente en el casco.

Oculus Rift tiene un seguimiento posicional y rotacional de 6 grados de libertad, gestionado a través del sistema de seguimiento llamado Constellation. Este consiste en un conjunto de sensores de seguimiento infrarrojo, de forma similar a Vive; pues el casco y los mandos poseen LEDs infrarrojos que emiten luz infrarroja en ciertos patrones que facilitan el seguimiento preciso. Posee unos auriculares integrados, que incorporan efectos de audio 3D en tiempo real.

PlayStation VR

Conocido anteriormente como Project Morpheus, es el visor de realidad virtual desarrollado y manufacturado por Sony. Fue diseñado para ser completamente funcional con su consola PlayStation 4.

El casco tiene un panel de 5.7 pulgadas OLED, con una resolución de 1080p. Tiene un conjunto de procesadores con varias funcionalidades, como las que pueden ser el tratamiento de audio 3D. Además, tiene nueve LEDs de posición en su superficie, los cuales son usados para hacer el seguimiento de 360 grados de la cabeza mediante el uso de la PlayStation Camera, que es el sensor de movimiento colocado en la consola, necesario para usar las características de realidad virtual.

Además del casco, se complementa mediante los mandos PlayStation Move que están provistos de sensores de posición y de rotación, además de usar un LED incorporado que funciona del mismo modo que los que aparecen en el casco.

3.1.2 Realidad Virtual móvil.

En lo referente a la realidad virtual móvil, son dos los principales protagonistas: Google VR y Gear VR; permitiendo el acceso a mundos de realidad virtual de forma más reducida pero más económica. Esto ha podido ser posible debido a la mejora del hardware de los dispositivos móviles, mejoras como la inclusión de sensores de posicionamiento (giroscopio, acelerómetro…), la mejora de la densidad de píxeles en las pantallas, y la renderización de imágenes a una tasa de refresco aceptable para esta tecnología.

Gear VR

Gear VR es el visor de realidad virtual desarrollado por Samsung con la colaboración de Oculus [9]. Este visor fue puesto en venta el 27 de noviembre de 2015. Sin embargo, la compatibilidad con los dispositivos móviles de otras marcas es más bien limitado, pues está diseñado para funcionar con los buques insignia de Samsung: la familia de los Samsung S6, Samsung S7, y Samsung S8.

Google VR: Daydream y Cardboard

Google VR es el kit de desarrollo ofrecido por Google, el cual ofrece dos plataformas de realidad virtual móvil: Daydream y Cardboard.

Gracias a la plataforma Cardboard, se ha acercado el concepto de la realidad virtual a una gran cantidad de usuarios debido a la poca inversión que deben realizar para disfrutar de ella: simplemente se requiere un teléfono móvil y un visor compatible, que en muchos de los casos únicamente requiere que el visor pueda sujetar el teléfono de forma correcta. Cardboard es compatible tanto en Android (versiones 4.1 o superiores) como en iOS (versiones 8.0 o superiores) [10].

Fig. 1. Visor básico de Google Cardboard.

En el lado izquierdo del visor se observa un interruptor constituido por un disco imantado. En las primeras versiones de Cardboard, la interacción con los elementos de las simulaciones se realizaba mediante la detección del cambio del campo magnético en los sensores magnéticos del dispositivo, provocado al empujar el disco hacia abajo. En versiones más nuevas, el activado por disco magnético se ha sustituido por botones capacitivos que pulsan la pantalla.

La plataforma Daydream ofrece una mejor calidad de simulación, con una latencia reducida para aumentar la sensación de inmersión. Daydream usa técnicas avanzadas para predecir la orientación de la cabeza del usuario, para mejorar la nitidez de la imagen evitando así la 'motion blur', lo que se traduce en un mejorado confort del usuario [11]. Daydream incorpora soporte para punteros físicos, usados para simular el movimiento de la mano dentro de las aplicaciones, utilizado de forma estándar para la interacción con elementos, en vez del uso del apuntado basado en mirada de Cardboard.

Sin embargo, en la actualidad sólo es compatible con el visor Daydream y con una lista de dispositivos exclusivamente Android certificados 'Daydream-ready', que suelen ser aquellos de alta gama, tales como los Pixel, los Galaxy S8, y el Moto Z, entre otros [12].

Fig. 2. Visor oficial de Google Daydream, acompañado del puntero usado en sus aplicaciones.

3.2 Accesibilidad

El concepto de accesibilidad se refiere al diseño inclusivo de productos, dispositivos, servicios, o entornos para todo tipo de usuarios (incluyendo personas con alguna discapacidad). La accesibilidad puede verse como la habilidad de acceso a algún recurso por cualquier usuario, sin importar sus capacidades ni los medios de los que dispone. El diseño accesible asegura tanto el acceso directo como el acceso indirecto, entendiendo por acceso directo el hecho de poder acceder al recurso sin asistencia, y el acceso indirecto, mediante el uso de tecnología de asistencia.

Si bien el concepto de accesibilidad se centra en permitir el acceso a personas con discapacidad o necesidades especiales, los desarrollos accesibles proporcionan un aumento de la calidad del producto para todos los perfiles, tengan alguna discapacidad o no.

Debido a la cantidad creciente de personas interesadas en jugar a videojuegos, y el uso cada vez mayor de los videojuegos con enfoques más allá del entretenimiento, tales como la educación y la salud, la accesibilidad en videojuegos se ha convertido en un campo de investigación emergente.

Existe una variedad de condiciones que pueden convertirse en barreras cuando una persona intenta jugar a un videojuego. Las categorías principales que se encuentran en el mundo de los videojuegos son limitaciones sensoriales, motrices y cognitivas. [13]

3.2.1 Discapacidad sensorial.

Dentro de la discapacidad sensorial, se incluyen aquellos usuarios que presentan dificultades o imposibilidad para percibir la información de salida emitida por el dispositivo usado, a través de uno o varios canales o sentidos [14]. Dentro de los sentidos que más afectan a esta dificultad o imposibilidad de obtener la información, se encuentra el sentido de la vista y el sentido del oído.

Discapacidad visual

Es aquella que afecta al sentido de la vista: los usuarios con discapacidad visual no pueden depender del canal visual como fuente exclusiva de información. Existen tres grandes tipos dentro de la discapacidad visual: la ceguera total, la ceguera parcial y el daltonismo.

- La ceguera total es definida como la pérdida total de la visión que no es corregible con lentes. Las personas que son invidentes no pueden jugar a videojuegos que dependen del uso de señales visuales para interactuar con el jugador: dependen de los sonidos o hardware especial, como pueden ser dispositivos de respuesta háptica, para obtener información.
- La ceguera parcial está relacionada con la invidencia: una persona con ceguera parcial puede detectar luz, pero de una forma muy limitada. Existe un término conocido como la 'ceguera legal' [15], acuñado por la Organización Mundial de la Salud, que indica que esta ceguera se da cuando la agudeza visual de ambos ojos, mediante la aplicación de medidas correctivas como pueden ser lentes, tenga un valor igual o inferior a 6/18 en la Escala de Wecker o una agudeza visual en el test de Snellen de 20/200 [16] en el mejor de los ojos. En caso de superar estos valores de agudeza visual, pero tener un campo visual en el mejor de los ojos inferior a 20 grados, también se trataría de ceguera legal. En cualquiera de esos casos, los usuarios podrían jugar a juegos siempre que haya un grado suficiente de magnificación en la pantalla, permitiendo que puedan reaccionar tanto a señales auditivas como señales visuales, aunque la incapacidad de ver un área extensa del juego debido a la magnificación de la pantalla supone una reducción de la calidad de la experiencia para el jugador.
- El daltonismo consiste en la imposibilidad de distinguir ciertos colores. Engloba diferentes perfiles, oscilando entre la falta de capacidad de distinguir cualquier color, conocido como acromatopsia (muy poco común), en el que la persona percibe el mundo en escala de grises, hasta perfiles más comunes como personas que no perciben la diferencia entre el rojo y el verde (deuteranopia), los colores rojizos (protanopia), o entre el amarillo y el azul (tritanopia).

Discapacidad auditiva

Dentro de la discapacidad auditiva, existen dos grandes afecciones: la sordera y la dificultad auditiva o pérdida de audición.

- Se dice que alguien sufre pérdida de audición cuando su capacidad de audición es reducida, es decir, cuando el umbral de audición en ambos oídos es inferior a 25 decibelios [17]. Ésta puede afectar a uno o ambos oídos y entraña dificultades para oír una conversación o sonidos leves. En casos leves o moderados, la pérdida de audición se puede corregir mediante el uso de dispositivos como audífonos o implantes cocleares.
- La sordera es la imposibilidad de entender el habla o reconocer sonidos: las personas sordas son aquellas que padecen una pérdida de audición profunda, lo que significa que oyen o muy poco o nada.

Por ello, requieren alternativas visuales o hápticas para recibir información de emisores naturalmente auditivos. El mejor ejemplo de ello es el lenguaje de signos, usado para comunicarse de forma exclusivamente visual.

Aunque las personas con dificultades auditivas pueden tener una capacidad auditiva mejorada con dispositivos, esto no significa que se encuentren completamente cómodas al procesar información exclusivamente auditiva. Por ello se deben ofrecer siempre alternativas visuales o hápticas.
También es importante el hecho de que una gran parte de los casos de problemas auditivos están relacionados con problemas del habla, por lo que el uso de la voz como medio de entrada de información no es adecuado para personas de este perfil.

3.2.2 Discapacidad motora

Dentro de la discapacidad motora se incluyen aquellas personas que se encuentran barreras a la hora de realizar tareas que requieren realizar determinados movimientos, aplicar fuerza o el uso de diferentes gestos de forma simultánea. Al igual que el resto de tipos de discapacidad, en ella se incluye una gran variedad de casos, ya sean los más severos, como pueden ser la completa incapacidad de mover extremidades debido a parálisis o ausencia de ellas, o aquellos casos en los que el usuario no puede aplicar fuerza o sus movimientos están limitados.

En el caso de la imposibilidad de mover, por ejemplo, los brazos, un videojuego que requiera una buena coordinación mano/ojo o la habilidad de poder presionar de forma rápida un botón, no se puede considerar accesible, ya que el usuario no podrá realizar esas acciones.

Aquellos usuarios que sufran de tendiditis, síndrome de túnel carpiano u otras lesiones por esfuerzo repetitivo puede ser que no disfruten del videojuego si este requiere que se realicen movimientos continuos. Estas enfermedades son tratables, pero pueden volver a aparecer si se realizan los mismos movimientos de forma repetitiva. Esto podría ser solucionado mediante el uso de hardware ergonómico.

3.2.3 Discapacidad cognitiva

Dentro de lo que engloba la discapacidad cognitiva se incluye una gran cantidad de perfiles diferentes, por lo que se puede considerar el grupo más heterogéneo, el cual tiene como principal diferenciador el grado de severidad.

En general, los usuarios dentro de este grupo presentan dificultades para comprender o aprender el funcionamiento de un sistema, aplicación o videojuego.

Como enfoque para mejorar la accesibilidad y superar las barreras que le suponen a los usuarios, se debe dar especial importancia a la simplicidad en el manejo: cuanto más intuitiva, simple, consistente y uniforme sea la interacción con el sistema, será más sencillo de utilizar para aquellos usuarios con este tipo de perfiles. Por tanto, se deben evitar los términos enrevesados, las navegaciones no intuitivas e innecesariamente largas o mal clasificadas y las tareas complejas.

Dentro de la discapacidad cognitiva se encuentran usuarios con patologías tales como pueden ser la pérdida de memoria, trastorno de déficit de atención, y dislexia, entre muchos otros.

3.3 El estado de la accesibilidad en Unity nativo

De forma nativa, Unity apenas ofrece características y herramientas para mejorar la accesibilidad de los desarrollos realizados en cualquiera de las plataformas que soporta. La única característica encontrada en la documentación de Unity es la clase 'VisionUtility', dentro del paquete UnityEngine.Accessibility, la cual consta exclusivamente de un método estático 'GetColorBlindSafePalette' [18].

Este método recibe como parámetros un array de colores a rellenar, y dos números de coma flotante: uno para especificar la luminancia mínima de la paleta y otro para especificar la luminancia máxima de la paleta.

Como valor de retorno, devuelve el número de colores que no son ambiguos. Es decir, en caso de que se repitan los colores en la paleta, se devolverá un número menor que la longitud del array de colores rellenado.

La función obtiene una paleta de colores cuyos colores deberían ser distinguibles tanto para personas con visión normal, como para aquellos con deuteranopia, protanopia y tritanopia.

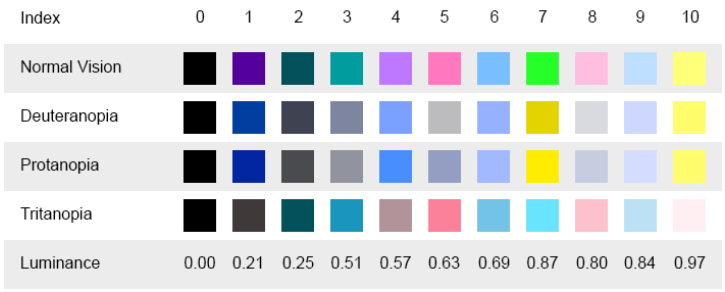


Fig. 3. Ejemplo de paleta de colores de devuelta por la función GetColorBlindSafePalette.

En la figura se muestra un ejemplo de llamada al método. En este caso, al pasar como parámetro un array de longitud 11 y una luminancia mínima de 0 y máxima de 0.97, devuelve los colores que se ven en la fila 'Normal Vision'. Las filas de abajo muestran cómo ven las personas con deuteranopía, protanopia y tritanopia los colores devueltos.

3.4 Directrices de accesibilidad

En la actualidad existen un conjunto de pautas de accesibilidad de distintos tipos de aplicaciones, ya sean web (tales como la Web Content Accesibility Guidelines (WCAG) 2.1, que consta desde el día 30 de enero de 2018 como W3C Candidate Recommendation [19]), de escritorio tales como la Microsoft Human Interface Guidance [20], la guía de Accesibilidad de Apple [21]; o en aplicaciones móviles, como puede ser la guía creada por Juan Aguado Delgado y Francisco Javier Estrada Martínez, ofrecida por la Secretaría General de Administración Digital [14].

Todas estas guías tienen elementos en común que pueden extrapolarse para otro tipo de desarrollos: a pesar de que no existen unas pautas específicas para la mejora de la accesibilidad en aplicaciones de realidad virtual, se pueden aplicar tantas pautas como sea posible de las guías mencionadas anteriormente.

3.5 Aplicabilidad de las directrices sobre videojuegos y VR

Si bien es cierto que las guías poseen algunos elementos más enfocados a cierto tipo de plataformas, como puede ser las WCAG, que mencionan tecnologías ya implementadas dentro de sistemas web para soportar las tareas de la accesibilidad, en el caso de Unity no existen tecnologías creadas de forma nativa que permitan cumplimentar de forma directa aquellas pautas de accesibilidad establecidas por este tipo de guías.

En el desarrollo de videojuegos no existen pautas de accesibilidad certificadas como tal, pero sí que existen guías que se han sintetizado a partir de la combinación de guías de accesibilidad para sistemas multimedia, como puede ser vídeo y audio, junto a pautas de accesibilidad de sistemas software. El ejemplo por excelencia de este tipo de guías es la guía *Game Accessibility Guidelines [22]*.

La guía *Game Accessibility Guidelines* consiste en un conjunto de pautas de diseño clasificadas según tres niveles estimados, basados en el valor (coste de la implementación), el alcance (número de personas que se pueden beneficiar en la implementación), y el impacto (cuán grande es la diferencia que notan las personas alcanzadas tras su implementación). Los tres tipos de niveles se denominan básico, intermedio y avanzado.

Además de la distinción entre los tres tipos de niveles, también se distinguen entre el tipo de accesibilidad al que están destinados: motora, cognitiva, de visión, de escucha, de habla, y general. Algunas de las pautas están repetidas en diferentes secciones de tipos de accesibilidad, debido a sus aportaciones a más de un tipo.

La guía sigue un funcionamiento de libre colaboración, pero el grupo de contribuyentes principales está constituido por diseñadores, consultores y profesionales de la industria de los videojuegos y de la accesibilidad.

Por ello, se ha realizado una tarea de extrapolación mediante el uso de algunas de las pautas de accesibilidad WCAG 2.0 junto a las pautas proporcionadas en la guía *Game Accessibility Guidelines* en el diseño del videojuego creado en este trabajo. También se hará uso del checklist ofrecido por *Game Accessibility Guidelines*, que permite marcar las diferentes pautas aplicadas en el desarrollo según si su aplicación es relevante para las mecánicas de la aplicación y si han sido finalmente implementadas.

3.6 Accesibilidad en la Realidad Virtual

En esta sección se hace hincapié en las diferentes barreras de accesibilidad presentes en el medio de la realidad virtual: aquellas específicas de las formas de input de la realidad virtual, la interacción con un mundo virtual y otras relacionadas con el hardware utilizado. Las cuestiones tratadas en esta sección se refieren a la accesibilidad en aplicaciones de realidad virtual de cualquier tipo y sobre los diferentes tipos de visores VR. Muchas de las cuestiones tratadas aquí no son aplicables al diseño de aplicaciones VR mediante Cardboard dada su simplicidad y su carencia de sensores avanzados y punteros manuales, pero es interesante conocer todos los detalles en caso de una extensión futura del trabajo en otras plataformas [23].

3.6.1 Cinetosis provocada por simulación

La cinetosis, véase, los mareos producidos por el movimiento, es provocada debido a un desacuerdo entre el movimiento percibido de forma visual y el sentido de movimiento del sistema vestibular, es decir, cuando alguno de los sentidos está enviando información al cerebro de una persona sobre algo que está sucediendo, que en este caso sería que la persona está en movimiento, mientras que otro de los sentidos le está informando de lo contrario, que está estacionaria. La cinetosis habitual, como la que puede suceder cuando una persona va en tren, avión, o coche, se debe a que el sistema visual está informando a la persona de que no se está desplazando, y que el sistema vestibular del oído le está indicando que sí que se está desplazando, lo que provoca mareos de diferente magnitud dependiendo de la persona.

En el caso de la cinetosis provocada por simulación, el concepto es el mismo pero las causas son justo las contrarias: cuando el jugador está en una simulación de realidad virtual ya sea de pie o sentado, siempre que sea en una posición relativamente estacionaria, se pueden producir mareos de este tipo cuando el jugador se está moviendo dentro de la simulación, como puede ser utilizando un mando, un joystick, un teclado, o un ratón. Es decir, está sucediendo un desajuste sensorial: el aparato vestibular está detectando que la persona no se está moviendo, mientras que el sentido de la vista está viendo cómo se está desplazando.

Este es un problema importante, el cual es conocido desde los orígenes de la realidad virtual [24]. De hecho, hasta la actualidad, sigue siendo un problema. Por ejemplo, en el caso del juego *Resident Evil 7*, fue la principal razón por la que tuvo una recepción negativa al mostrar la demo al público [25]: debido a que el personaje debe de moverse de forma independiente usando el mando, y puede apuntar la linterna moviendo la cabeza, el jugador pierde de una forma fácil la sensación de equilibrio. Además, se cree que la

sensación se incrementaba debido a la insuficiente frecuencia de refresco para un juego con un sistema de movimiento de tal magnitud: se recomienda que los juegos VR con ese tipo de movimiento se ejecuten a aproximadamente 90 imágenes por segundo, mientras que este juego se ejecutaba a 60 imágenes por segundo no constantes, que es el mínimo admitido en los videojuegos de PlayStation VR [26].

Sin embargo, este efecto se puede evitar si se aplica el diseño apropiado en la simulación: el objetivo es reducir esa sensación de movimiento que está recibiendo la vista para que no sea dispar con lo que se está recibiendo en el sistema vestibular del oído. Hay numerosas técnicas para evitarlo:

- Hacer que la cámara sea siempre controlada por el movimiento de la cabeza del jugador, y nunca por el juego en sí. También puede hacerse que la cámara pueda ser controlada por el mando, de forma extra, para mejorar la accesibilidad motora, aunque esto puede provocar cinetosis, aunque en menor grado que un movimiento total del punto de vista (POV) del personaje.
- Reducir de forma dinámica la visión periférica durante el movimiento: cuando gran parte del campo visual se mueve, el espectador siente que él ha sido el que se ha movido y que el mundo se ha mantenido estacionario. Este fenómeno se conoce como vección, del inglés *vection*. Hay tres tipos: la vección circular, en la que el espectador ve un escenario que rota a su alrededor y cree que es él el que está rotando mientras que el escenario no se mueve; la vección lineal, en la que el observador ve un escenario que se acerca y se aleja hacia él y cree que es él el que se está acercando y alejando del escenario, y la vección de guiñada (la guiñada es el eje vertical de giro de un avión), en la que el espectador ve un escenario que rota alrededor de su línea de visión, pero siente que ha rotado alrededor de la línea de visión [27].

  Una solución para esto puede ser el uso del tunneling, en el que se reduce el campo de visión que se mueve mientras que el jugador está realizando un movimiento del personaje, el cual vuelve a reestablecerse cuando el jugador está quieto [28]; entendiendo como movimiento la transición del personaje dentro del mundo, no hacia dónde está mirando.

  También se puede aplicar esta técnica cuando el jugador está moviendo la cámara mediante medios no naturales, véase con un ratón o mando. Dar al jugador un marco de referencia, ya sea para el movimiento o para la cámara. En el caso de una referencia para el movimiento, puede ser útil en casos en el que el juego haga un movimiento de translación sobre la cámara de forma que no sea controlada por el jugador: si se muestra una visualización de la trayectoria del movimiento para que así el jugador sea capaz cual es el camino que va a tomar su visión, se puede mejorar la comprensión del usuario sobre a dónde se dirige, reduciendo la sensación de cinetosis. Un marco de referencia para la cámara puede ser un elemento que aparezca en la pantalla que sea fijo, como puede ser algo similar a una cabina de piloto o elementos de interfaz que no roten con el usuario, para que así sirvan como puntos de referencia de la rotación que ha hecho, favoreciendo la sensación de posicionamiento sobre el mundo. Los elementos estáticos que rodean al usuario durante los movimientos ayudan a reducir sensaciones incómodas ya que todo pasa a ser más predecible para el usuario.

Fig. 4. En la parte superior, visión sin movimiento, en la cual no se aplica el tunneling. En la parte inferior, el jugador se está desplazando, por lo que se aplica el efecto de tunneling. Se puede observar cómo el recuadro mostrado en el tunneling es la posición actual del jugador, mientras que la imagen mostrada detrás del recuadro es la posición inicial del jugador. Fuente: "Tunneling" VR Locomotion, disponible en YouTube.

- Reducir las opciones de locomoción del juego puede evitar la sensación de movimiento que no se corresponde con el movimiento del jugador. Esto se ha visto en juegos como Skyrim VR [29], en el que el movimiento de translación puede ser libre o mediante teletransportación: el movimiento libre es el movimiento estándar con el mando, y el de teletransportación permite al usuario apuntar a una zona y mover la cámara, y a su vez al personaje, al instante en esa posición. De este modo, aunque parezca menos inmersivo, se reduce en gran medida la sensación de movimiento. Del mismo modo, también ofrece un modo reducido de rotación de la cámara: posee el modo natural, en el que la rotación no tiene restricciones y sigue el movimiento de la cabeza, o el modo fijo, en el que la cámara solo se puede mover en incrementos/decrementos de un número especificado de grados.

Fig. 5. Imagen del videojuego *Skyrim VR* que muestra el movimiento por teletransporte que ofrece. Se aprecia un marcador en el suelo que aparece desde la mano izquierda del personaje, que representa el punto al que se teletransportará el jugador para moverse.

- La representación del cuerpo del personaje puede afectar no tanto la cinetosis, aunque puede empeorar la inmersión en el videojuego. Esta puede ser debido a la altura de la cámara y la altura del jugador si está de pie, disparidad entre el usuario sentado y el personaje de pie o viceversa, u otras posiciones; o directamente que el juego muestre una representación del cuerpo del personaje, pues puede provocar conflicto con el esquema corporal percibido (véase, el fenómeno por el cual conocemos donde se encuentra nuestro cuerpo, por ejemplo, sabemos dónde se sitúan nuestras extremidades): quizá el jugador tenga un brazo extendido pero el personaje muestre ese brazo no extendido, por lo que hay una disparidad sensorial relacionada con la posición del cuerpo. Por ello, no se suele representar el cuerpo del personaje, ya que, aunque se haga de forma precisa, nunca será lo suficientemente precisa como para relacionarse con cada uno de los jugadores; o se representa de forma sencilla, únicamente mostrando las manos. Respecto a si el jugador está sentado o de pie y el personaje está en otra posición diferente, es ya cuestión del jugador cambiar a la postura que tenga el personaje, siempre que sea posible.
- Otras cuestiones pueden ser los cambios del horizonte para el jugador, con la solución directa de no hacerlo. Juegos que juegan con la gravedad del mundo y giran los puntos de referencia suelen ser desencadenantes de la cinetosis, pues el jugador pierde el punto de referencia básico para los humanos: el suelo; por lo que se debe mantener un horizonte constante para evitarla.
- Dentro de las cuestiones más técnicas, es importante mantener una frecuencia de refresco de la imagen constante, ya sean 60fps, 90fps, 120fps, y una latencia de visualización de la imagen inferior a 20 milisegundos, para evitar el *input lag*. Los cambios irregulares en la frecuencia de refresco pueden dar la

sensación de latencia entre movimientos, que rompe la fluidez de la simulación y por ello, una pérdida de la inmersión.

3.6.2 Dificultad motora

La accesibilidad motora, de forma tradicional, se puede traducir a la posibilidad de manejar el mando o el ratón y teclado; asumiendo como única dificultad motora aquella que se manifiesta en brazos y manos. Sin embargo, con la realidad virtual la complejidad de habilidad motora ha aumentado, ya que no se pueden realizar acciones usando únicamente las manos, sino también mediante el movimiento de la cabeza, el hecho de mirar fijamente un punto para realizar alguna acción, o incluso tener que darse la vuelta, en el caso de tener que realizar una acción que requiera la atención del jugador a sus espaldas.

En estos casos, se puede decir que las limitaciones en la movilidad son sobre todo visibles en el caso de la realidad virtual. Por ejemplo, una persona en silla de ruedas no puede, o al menos no le es cómodo, interactuar en un medio de realidad virtual en 360º. En estos casos, la solución puede llegar en forma de software, ya sea diseñando la experiencia para que no se requiera la visión en 360º de forma constante (como puede ser que aparezca un enemigo o algo que requiera la atención del jugador detrás suya), o sistemas de rotación de la cámara controlada por el usuario, sin que requiera girarse hacia atrás.

También existen videojuegos en VR que asumen que el jugador es capaz de levantarse, saltar y agacharse, siendo estas acciones necesarias para avanzar en el juego; o aquellos que permiten asignar la altura de la cámara, lo cual es positivo para personas en silla de ruedas, pero que al establecer esa altura asumen que al ser una altura baja el avatar debe ser el de un niño, en lugar de tomar como posibilidad que el jugador esté sentado [30].

Existen sistemas de seguimiento ocular que permiten extender los controles de los sistemas de realidad virtual, habilitando el movimiento según el movimiento de los ojos, aunque algunos de ellos todavía no están disponibles en el mercado de forma extendida. Varios de estos sistemas pueden ser los siguientes:

- La solución de Pupil Labs [31], que ofrecen add-ons de hardware que se pueden acoplar a diferentes tipos de headset de VR y AR, tales como el headset HTC Vive, las Oculus Rift, o las Microsoft Hololens. Estos accesorios se ajustan al headset permitiendo enviar información sobre el movimiento ocular al ordenador a través de un cable USB 2.0. Proporciona una API que permite trabajar con esta información en Unity [32].

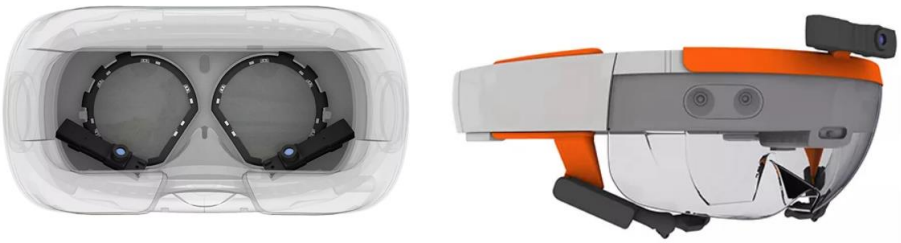


Fig. 6. Add-ons de seguimiento ocular en dispositivos XR de Pupil Labs. A la izquierda, add-on binocular para HTC Vive; a la derecha, add-on binocular para Microsoft Hololens. Fuente: página oficial de Pupil Labs.

- Tobii Tech [33] aporta al mismo modo que Pupil Labs una solución que se integra con el headset HTC Vive, siguiendo un funcionamiento similar; aunque su objetivo no es el del seguimiento ocular para VR exclusivamente, sino también tienen proyectos para la implantación de este tipo de sensores en dispositivos como portátiles y monitores.
- FOVE Inc. [34] está desarrollando directamente un headset VR que ya incorpora en sus funcionalidades básicas el seguimiento ocular, aunque a día de hoy (marzo de 2018) únicamente está disponible el headset en formato devkit para desarrolladores e investigadores. Sus sensores permiten tomar 120 imágenes por segundo de ambos ojos, y mide el movimiento de los ojos con una precisión inferior a 1 grado.

Aunque no esté relacionado con la accesibilidad motora, también es importante indicar que el hecho de permitir el seguimiento ocular dentro de un videojuego puede incrementar su rendimiento gráfico, pues abre la posibilidad de detectar dónde está mirando el jugador y así reducir la calidad gráfica del resto de elementos o incluso mejorar la calidad gráfica del elemento al que el usuario está mirando, proporcionando así un equilibrio entre calidad y rendimiento con menos penalización gráfica. Por lo tanto, se puede considerar la compatibilidad con estos sistemas en un desarrollo VR no solo para mejorar la accesibilidad, sino también para mejorar el rendimiento de las aplicaciones.

Los tipos de habilidad motora que pueden entrar en conflicto con barreras relacionadas con lo motor en la VR incluyen:

- Fuerza y fatiga: la posibilidad de llevar el peso añadido del headset VR en la cabeza sin que esté soportado por algo externo o los controles en las manos, o no poder realizar movimientos de forma repetida ya sea con la cabeza o con las manos.
- Rango de movimientos: cuánto puede mover el usuario la cabeza o las manos en cualquier dirección, en lo que a distancia se refiere; o las posibles limitaciones que el usuario tenga en el movimiento de sus dedos para manejar los controles; algo que es sobre todo importante en el caso de headsets VR que tengan los botones directamente en el headset, como puede ser en el caso de algunos visores Cardboard.
- Precisión: la capacidad del usuario de poder realizar movimientos pequeños ya sea al manejar un dispositivo con joystick como puede ser un mando de consola, un ratón, o un controlador que dependa de la

información del giroscopio como pueden ser los mandos el headset de PlayStation VR; o juegos que requieran dirigir la cabeza hacia un punto de forma precisa.

- Altura: la altura del usuario, que puede tomar un amplio rango de valores y deberá tenerse en cuenta dentro de la simulación, incluyendo si el usuario está sentado en una silla de ruedas.
- La presencia de extremidades y/o dígitos: debido a que no todo el mundo tiene dos manos funcionales con los diez dedos funcionales, es posible que no puedan manejar los controladores estándar para el sistema de VR.
- Velocidad: el poder realizar una determinada tarea en un marco temporal reducido.
- Locomoción: la posibilidad de que el usuario ande, se incline o se agache.
- Equilibrio: el hecho de poder mantener el equilibrio al mantener ciertas posturas, sobre todo al estar de pie.

Si bien es cierto que puede haber algunas barreras que puedan ser superadas por limitaciones de hardware, la mayoría de ellas pueden ser superadas mediante un buen diseño a nivel de software, como puede ser permitiendo tener más de un método de input y que este sea configurable al gusto y necesidad del jugador, permitiendo el ajuste por el jugador de la altura de la cabeza, la implementación de un sistema de asistencia a la puntería, que reduzca la precisión necesaria para hacer determinadas acciones; entre muchas otras soluciones.

3.6.3 Epilepsia fotosensible

Si bien los casos de epilepsia fotosensible son raros, no significa que no sea una parte crítica de la accesibilidad en VR. Generalmente, la accesibilidad se centra en evitar malas experiencias o exclusiones de forma innecesaria, o en el caso de la cinetosis provocada por simulación, mareos. Sin embargo, en el caso de la epilepsia, pueden producirse daños bastante severos sobre el jugador en caso de que sufra un ataque de epilepsia debido al videojuego.

Por este motivo, los fabricantes de hardware desaconsejan a los usuarios con epilepsia el uso de dispositivos de VR, debido a los riesgos que puede tener, además de que, al ser una tecnología más reciente, no se ha entrado a mejorar las condiciones del hardware para evitar el parpadeo u otros desencadenantes de ataques epilépticos. Sin embargo, la única forma de saber si estos dispositivos VR provocan epilepsia por sí solos es probarlos sobre personas que padecen epilepsia y conocer si sufren ataques epilépticos en su uso o no, lo cual no se ha llevado a cabo.

Todavía no se han hecho investigaciones sobre el impacto de la realidad virtual sobre la probabilidad de sufrir ataques epilépticos, aunque uno de los factores para sufrir ataques epilépticos es la proporción del campo de visión del usuario que está siendo cubierto por un efecto visual. En el caso de una pantalla, ya sea un monitor, televisión o dispositivo móvil, la imagen cubre solo una pequeña parte del campo de visión del usuario, pues está únicamente limitada al tamaño de la pantalla y la distancia a la que se sitúe el jugador

del dispositivo. Sin embargo, en el caso de la VR, la proporción del campo de visión del jugador respecto a una imagen parpadeante puede ser completa: en VR el campo de visión puede ser un 100%, mientras que en un monitor estándar suele ser de entre un 25% y un 40% del campo de visión.

Además, otro factor que no permite asegurar al completo la seguridad para las personas con epilepsia fotosensible es la propia naturaleza de las pantallas, ya que estas funcionan mediante el parpadeo de diferentes imágenes a una gran velocidad: hay monitores de 60Hz, lo que se traduce a 60 imágenes parpadeando sobre el monitor en tan sólo un segundo, o incluso 144Hz, siendo 144 imágenes por segundo.

Si bien es cierto que es difícil que un juego sea completamente seguro para personas que sufran epilepsia fotosensible, por la dificultad de probarlo para comprobar que no hay riesgo de ataques epilépticos y el hecho de que no se puede asegurar que todas las partes de la experiencia sean libres de desencadenantes, pues estos pueden existir dentro del diseño del videojuego, dentro de la renderización de la imagen, o incluso en la misma pantalla; se pueden tomar medidas generales para reducir el riesgo de ataque epiléptico.

Existen un conjunto de desencadenantes comunes a evitar, los cuales están relacionados con parpadeos, luces muy intensas, o patrones repetidos con alto contraste. Algunas compañías evitan totalmente la inclusión de funcionalidades que pueden tener alguna de esas características, otras permiten deshabilitar aquellas porciones del programa que encajan con esas características generales de desencadenante.

Los desencadenantes a evitar están definidos de la siguiente forma [35]:

- Cualquier secuencia de imágenes parpadeantes que duran más de 5 segundos. Por parpadeo se refiere a un gran cambio en el brillo o contraste de forma instantánea, incluyendo cambios de escena, incluyendo además aquellos parpadeos que, sin tener en cuenta la luminancia, hacen que la imagen vaya al color rojo o venga desde el color rojo.
- Más de tres parpadeos en un solo segundo si cubre más de un veinticinco por ciento de la pantalla.
- Patrones repetidos o texto uniforme que se mueve, cubriendo más de un 25% de la pantalla. Por patrones repetidos se entiende más de cinco líneas repetidas con alto contraste, filas o columnas tales como pueden ser rejillas o tableros.
- Patrones repetidos o texto uniforme estático que cubren más del cuarenta por ciento de la pantalla.

El texto puede ser un problema si son más de cinco líneas formateadas con sólo letras mayúsculas, sin demasiado espaciado entre letras, y el espaciado de línea siendo tan alto como lo que ocupan las letras. Esto hace que el texto sea uniforme y siga un patrón, convirtiéndolo de forma efectiva en un conjunto de filas con alto contraste colocadas de forma alternada con el entrelineado.

Existen fórmulas específicas para calcular qué diferencia en el contraste tiene algún efecto y qué porción del campo de visión de una persona se tiene que ver afectada para provocar el efecto, aunque estos cálculos son complicados y suelen ser estimaciones.

Existe un servicio de test de epilepsia, desarrollado inicialmente para contenido para televisión pero que también es aplicable para videos de juegos, recomendado por la industria y usado tanto por estudios como por editoras de videojuegos; aunque no asegura al completo que el videojuego no pueda provocar ataques epilépticos [36].

Por lo tanto, es mejor ser precavido y evitar en mayor medida los patrones uniformes y los colores parpadeantes, y evitar al completo el término "libre de epilepsia" o similares, pues no solo hay riesgo de dañar a los jugadores, sino que también puede tener repercusiones legales.

3.6.4 Discapacidad visual

El reducido tamaño de los textos y de la interfaz de usuario en general son una queja bastante homogénea, pues se puede encontrar en muchos de los videojuegos presentes en el mercado.

Existen un conjunto de recomendaciones para el tamaño mínimo que deben de tener los elementos de la interfaz de usuario y los subtítulos de cualquier medio visual que muestre imágenes a una resolución de 1080p: 28 píxeles en el caso de textos de la interfaz, y 46 píxeles en el caso de los subtítulos [37].

Estas recomendaciones están indicadas para personas con una visión de 20/20 según el Test de Snellen, lo cual sería equivalente a una visión con 0 dioptrías. Teniendo en cuenta que estas indicaciones son rara vez seguidas, lo cual provoca dificultades para la lectura de la interfaz, sobre todo para personas con una agudeza visual inferior, es especialmente grave para personas que tengan una visión reducida.

La solución tomada por las personas que no consiguen leer los textos desde lejos es bastante sencilla: sentarse más cerca de la pantalla, como forma de ver los textos más grandes. En el caso de la realidad virtual, es imposible sentarse más cerca de la pantalla, y la solución de hacer zoom sobre todos los elementos de la pantalla no es válida, al menos sin incrementar bastante el riesgo de cinetosis.

Por ello, en el desarrollo de una aplicación VR hay que tener en cuenta dos factores:

1. Prestar atención al tamaño y al contraste, tanto de los elementos de la interfaz de usuario como de elementos importantes de la jugabilidad, pudiendo ofrecer además una forma de controlar el tamaño de los textos de la aplicación.
2. Introducir la funcionalidad que permita realizar lo equivalente a acercarse a la pantalla: en lugar de mantener los elementos de la interfaz a una distancia siempre fija respecto al jugador, permitir que el usuario pueda inclinarse hacia los elementos para verlos más de cerca. Si bien es cierto que inclinarse demasiado puede provocar problemas con la distancia de los sensores, se puede solucionar al ofrecer una opción para permitir la recalibración de los sensores en tiempo real. Hay que tener en cuenta que esta opción no es posible en sistemas de VR más sencillos como Cardboard, pues sólo es posible capturar los movimientos de inclinación hacia delante o hacia atrás con el uso de sensores más allá del giroscopio,

que es el único sistema de posicionamiento usado por Cardboard. Sin embargo, es posible emular estos movimientos mediante algún otro tipo de input, como puede ser el mando.

El uso de crosshairs, o retículas de apuntado, es bastante recomendado para aquellos usuarios que tengan una visión dispar entre los dos ojos como puede ser en el caso del estrabismo, o para aquellos que tengan una pérdida de visión mucho más reducida en un ojo que en otro, como puede ser en el caso de la ambliopía. Los usuarios con estas condiciones o similares suelen fijar su mirada a un punto ligeramente alejado del centro, lo cual en ausencia de un punto que indique el centro puede provocar que interactúe con elementos indeseados. Teniendo en cuenta el apartado anterior sobre tamaño de fuente, se puede considerar la implementación de una retícula personalizable tanto en tamaño como en color.

En el caso de la ceguera completa, se puede pensar que la realidad virtual es un medio a evitar por la asunción de ser un medio primordialmente visual. Sin embargo, esto es algo equivocado, pues las posibilidades que ofrece la realidad virtual en temas de audio son muy interesantes gracias a la espacialización del audio según el movimiento de la cabeza del jugador. Esto permite la simulación de entornos únicamente mediante el uso de la espacialización auditiva. De hecho, existen aplicaciones de realidad virtual basadas exclusivamente en esta característica, como el videojuego *Blind Swordsman*, que consiste en atacar a los enemigos mirando hacia ellos únicamente obteniendo como información los sonidos que emiten, como pueden sus pasos y voces [38].

Fuera del ámbito de la realidad virtual, existen videojuegos los cuales son jugados perfectamente por usuarios con ceguera, como *Grand Theft Auto V* y *Resident Evil 6*, debido al uso de una combinación de tecnologías asistivas, como la que permite reducir la precisión necesaria para realizar ciertas tareas, como presionar un botón e interactuar con un elemento cercano en vez de tener que apuntar directamente a ese elemento, o tecnologías de apuntado asistido; combinado con un excelente y detallado diseño de audio y entornos simples. Estos conceptos son aplicables también a aplicaciones de realidad virtual, pues simplemente cambian la forma en la que se controla la cámara.

Si bien es cierto que existen limitaciones bastante fuertes para personas con ceguera, como la navegación de entornos complejos y la interacción con la interfaz, mediante un buen diseño teniendo en cuenta usuarios con estos perfiles se pueden crear experiencias completamente disfrutables sin requerir el uso de feedback visual.

Otro concepto a tener en cuenta es la falta de visión binocular, como puede ser en casos de usuarios que padezcan de ambliopía (ojo vago) o diplopía (visión doble). En muchos de los casos el no disponer de visión binocular no es un problema, sobre todo en aplicaciones en las que se utiliza la retícula de apuntado, pero es importante tenerlo en cuenta si hay elementos que requieren de forma innecesaria percibir diferencias en profundidad, lo cual puede ser dificultoso en algunos de los casos. También es importante asegurarse que siempre se muestran los mismos elementos como pueden ser subtítulos o elementos de la interfaz en las pantallas para ambos ojos.

3.6.5 Discapacidad auditiva

El sonido es una parte importante de la realidad virtual, aunque está limitada en la actualidad a una única forma de transmisión: mediante auriculares estéreo. A pesar de la introducción de sistemas de procesamiento de audio binaural y espacializado 3D, no se llega a alcanzar la fidelidad de audio que puede proporcionar un sistema de más de dos canales como puede ser un sistema surround no digital, es decir, con canales reales, no simulados.

El estricto uso de sistemas estéreo es problemático en el caso de usuarios con pérdida de audición unilateral, es decir, aquellos usuarios que poseen un nivel de audición dispar entre ambos oídos, pudiendo ser de escaso impacto o la completa pérdida de audición por uno de los oídos. En estos casos, es posible que mediante el uso de una aplicación VR se pueda perder por completo información que esté sucediendo en el mundo virtual en una posición que esté directamente a la derecha o a la izquierda de la cámara en la aplicación.

Una de las soluciones empleadas para la corrección de este problema es la posibilidad de alternar el sonido del juego a modo monoaural, el cual consiste en la reproducción de los mismos sonidos tanto en el oído izquierdo como en el derecho, sin importar desde qué dirección se ha emitido ese sonido. Si bien es cierto que de este modo no se pierde información sobre los sonidos que se han emitido en el entorno, se pierde un fragmento de información bastante importante en el caso de la realidad virtual: la dirección del sonido.

En la vida real, la pérdida de audición unilateral es mitigada en cierto modo por el eco, pues es posible que un sonido emitido a la izquierda de la persona se escuche, de forma reducida, en su oído derecho, gracias al eco y la reverberación. De hecho, este sistema es uno de los objetivos en la mejora del audio en sistemas de realidad virtual, mediante sistemas avanzados de espacialización de audio que empleen herramientas que permitan simular efectos como el eco, la reverberación, entre otros. Uno de los sistemas que ofrecen estas características es Resonance Audio, implementado por Google, o paquetes de audio ofrecidos por Oculus, como pueden ser el Oculus Ambisonics [39] u Oculus Spatializer [40].

Sin embargo, uno de los problemas más destacados en la accesibilidad de aplicaciones de VR es la falta de subtítulos en muchas de ellas. A pesar de que los sistemas de subtítulos son una característica incluida en la gran mayoría de títulos de la industria de los videojuegos clásica, en el caso de títulos VR es una característica que apenas se ve implementada.

En el caso de los videojuegos y otros medios audiovisuales clásicos, la inclusión de subtítulos tenía un coste reducido en lo que respecta al diseño: se suelen colocar ya sea en la parte superior de la pantalla o en la parte inferior. En el caso de la realidad virtual, no existe esa noción de posición en la pantalla, pues no se puede colocar algo en la parte inferior o superior de la pantalla de forma estática sin más, por lo que se deben de buscar alternativas.

Una de las alternativas es colocar los subtítulos en el mundo virtual de forma que siempre estén situados en la parte inferior del campo visual del usuario. Una vez colocados en el mundo, hay que ocuparse de que los subtítulos no colisionen con los elementos del mundo, para que no puedan ser tapados por elementos del mundo que estén más cerca que los subtítulos. Esto requiere que los subtítulos estén colocados a una distancia muy reducida respecto a la cámara principal de la aplicación.

Si bien esto soluciona el problema de la oclusión de los subtítulos, está provocando otro problema diferente por su cuenta: el fenómeno conocido como conflicto vergencia-acomodación, o *vergence-accomodation conflict* [41]. El conflicto vergencia-acomodación, presente en todos los productos de headsets VR, consiste en que la forma en la que las lentes de los ojos de las personas enfocan un objeto es totalmente diferente a la forma en la que los ojos apuntan de forma física al objeto a el cual se quiere enfocar, cuando lo normal es que estas distancias sean proporcionales.

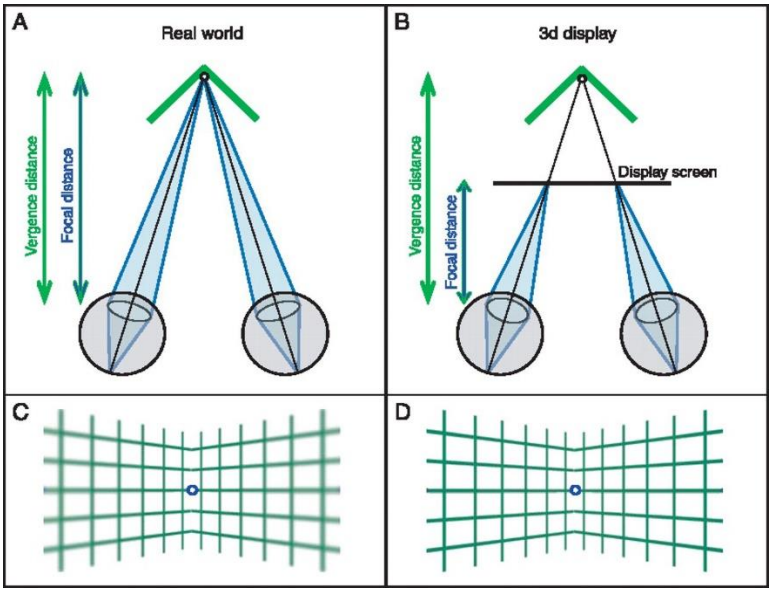


Fig. 7. Comparativa entre las distancias de vergencia y focales entre el mundo real y una pantalla, siendo en este caso un visor VR.

De forma resumida, consiste en que los ojos se dirigen a una posición para mirar al objeto en el que se quiere enfocar asumiendo que está a una distancia determinada, pero la distancia focal es en realidad uniforme a lo largo de todos los objetos de la pantalla, pues no se puede enfocar más allá de la pantalla.

Este fenómeno, imposible de evitar en las soluciones de realidad virtual actuales, puede provocar mareos y malestar cuando se intenta enfocar sobre objetos que están cerca y objetos que están lejos de forma intermitente dentro de una simulación. Esto está estrechamente relacionado con la forma de situar los subtítulos a una corta distancia para evitar la oclusión con otros elementos.

Para querer utilizar esta solución, es recomendable mantener los subtítulos en una posición no tan cercana a pesar de la oclusión, e intentar que haya un tiempo adecuado entre los cambios de profundidad de los objetos cuando se leen subtítulos, por ejemplo, intentando mantener los subtítulos durante más tiempo.

Una alternativa para evitar el conflicto vergencia-acomodación es la introducción de subtítulos de forma contextual, haciendo que aparezcan junto a los elementos que estén emitiendo el sonido o el diálogo. Sin embargo, esto expone otro problema: los subtítulos no pueden ser leídos si no se está mirando a la fuente del sonido, por lo que no resulta práctico.

4 Desarrollo del proyecto

En los siguientes puntos se expone el proceso seguido durante el transcurso del desarrollo del proyecto, que incluye el análisis de las herramientas utilizadas, el diseño del videojuego, y detalles sobre la implementación de este. Respecto a la implementación, se divide en dos secciones: implementación de las mecánicas del juego, e implementación de los elementos dedicados a mejorar la accesibilidad de éste.

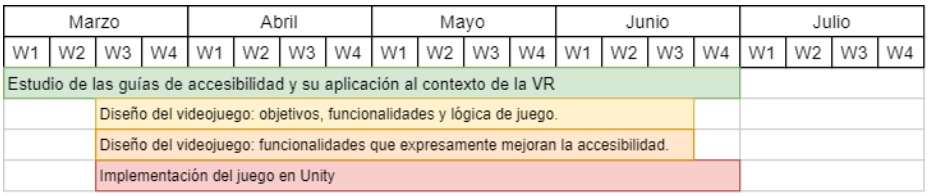


Fig. 8. Diagrama simplificado de Gantt que representa los principales bloques de trabajo realizado en este proyecto. Las tareas que están solapadas representan el proceso de mejora continua.

La metodología de trabajo elegida ha sido una metodología ágil mediante la creación inicial de un prototipo, que representa la forma más primitiva del videojuego, sobre el que se ha seguido un proceso de mejora continua hasta alcanzar el mayor punto de refinamiento a nivel técnico y a nivel de accesibilidad posibles por el alumno. El objetivo de este proceso continuo es obtener un conocimiento mejorado sobre las herramientas inherentes al desarrollo en Unity y las capacidades de Google VR, herramienta escogida para el desarrollo de VR móvil.

De forma más específica, la metodología aplicada se ha basada en la metodología Scrum, mediante la realización semanal de planificaciones para periodos breves de tiempo basadas en objetivos definidos a partir del diseño del videojuego y de los componentes de accesibilidad. Por ello, se han realizado *sprints* semanales para concluir las tareas establecidas. A pesar de que la metodología Scrum es aplicada en grupos de trabajo, se ha adaptado a un grupo unipersonal en este proyecto.

4.1 Análisis: Unity

Unity es un motor de videojuegos multiplataforma desarrollado por Unity Technologies, usado para desarrollar videojuegos y simulaciones tanto 3D como 2D para ordenadores, consolas y dispositivos móviles. En sus orígenes, únicamente estaba destinado a crear videojuegos para Mac OS. El motor fue anunciado por primera vez en la Apple's Worldwide Developers Conference en 2005 [42].

Unity soporta tanto gráficos 2D como 3D, ofrece funcionalidad drag-and-drop y scripting mediante el lenguaje de programación C#. Anteriormente soportaba tanto Boo, el cual quedó obsoleto en el lanzamiento de Unity 5 [43], como una variante de JavaScript llamada UnityScript, para la cual empezó el proceso de

ser obsoleta hace relativamente poco tras el lanzamiento de la versión de Unity 2017.1 en agosto de 2017, siendo un lenguaje marcado como obsoleto ya en la versión 2017.2 [44].

El entorno de desarrollo corre sobre Mono, una versión open-source del framework .NET [45], mientras que el motor Unity en sí corre sobre C++.

Unity posee un sistema de objetos de juego basados en componentes: a cada objeto de juego, o game object en inglés, se le pueden asociar componentes de código (scripts), derivados de alguna clase y comportamiento, y cuando se arrastra el componente de código sobre un objeto, ese objeto de juego ejecutará el código; permitiendo así una gran modularidad de las entidades de juego.

4.1.1 Editor de Unity

El editor de Unity contiene una interfaz basada en paneles modulares, siendo los más importantes el panel de escena, el panel de jerarquía, el panel del inspector y el panel del proyecto.

Fig. 9. Interfaz del editor de Unity. Se pueden observar los paneles mencionados: Scene, Hierarchy, Project e Inspector, además de una barra superior marcada como Toolbar. Fuente: Documentación de Unity.

- Dentro del panel de escena aparecen las entidades del juego de forma visual, tales como el jugador, la cámara, entre otros. Tiene su propia cámara que permite al programador navegar el escenario, ya sea 3D o 2D.

- Dentro del panel de jerarquía se encuentran todos los GameObject (entidades, objetos de juego) en la Scene (escena) actual, que siguen un orden jerárquico, pues pueden existir entidades que son hijas de otras entidades, compartiendo así cierto comportamiento.
- Dentro del panel del inspector se puede ver toda la información sobre un asset o un GameObject que esté dentro de la escena. En este panel se muestra la información detallada sobre el GameObject en cuestión y los Components (componentes) que tiene asociados, permitiendo cambiar los valores modificables de los componentes. Permite además realizar cambios sobre estas propiedades mientras que el juego se está ejecutando, para así poder ver de forma instantánea cómo cambia el comportamiento del objeto. Además, dentro de los campos habilitados, permite que se arrastren assets para que tomen como valor el asset en cuestión, útil en el caso de referencias a otros objetos dentro de scripts, texturas, materiales, etcétera. En el caso de components de script, aparecerán como propiedades modificables todos los atributos del script C# que tengan la palabra clave 'public'.
- Dentro del panel del proyecto se puede acceder y administrar los assets del proyecto. Estos pueden ser scripts, texturas, materiales, modelos y otros archivos, organizados mediante carpetas dentro del proyecto. Esta vista permite ver la estructura del proyecto en forma de lista jerárquica de carpetas, e incluye un motor de búsqueda para la rápida localización de assets del juego.
- La barra superior contiene controles básicos de manejo de los assets en la escena, tales como movimiento, posición, rotación y ajuste de escala; los controles de la vista de juego, que permiten ejecutar y detener el juego; y otras opciones.

Existen otros paneles o vistas, tales como el panel de consola, que muestra un log de los mensajes, warnings y errores emitidos por la ejecución de los scripts, la vista de juego, que muestra lo que se ve en el juego a través de la cámara principal, la vista de animación, que permite animar los objetos en la escena; entre otras.

4.1.2 GameObject

Los GameObject son los objetos fundamentales en Unity que representan personajes, escenario y otras entidades que actúan como contenedores de los Components, que implementan la funcionalidad. Un GameObject siempre tiene el componente Transform adjunto, sin posibilidad de eliminarlo. El resto de componentes que se pueden añadir al objeto pueden ser tanto componentes dados por Unity, como puede ser un Collider, un Renderer, o un Light, o desarrollados mediante scripts.

El componente Transform es obligatorio debido a que éste define la posición, rotación y escala en el juego. Además, el componente Transform es el que permite que un GameObject tenga otros GameObject hijos o padres, por lo que es un componente crítico para la creación de la jerarquía de objetos.

4.1.3 Tags

Una etiqueta o Tag es un valor de texto que se puede asignar a uno más o de un GameObject, el cual es usado para poder marcar a los GameObject con cierto valor para poder realizar diferentes tareas que involucran a ese GameObject según el valor: por ejemplo, se puede asignar el tag “Player” al jugador, “Collectible” a los objetos que se pueden recoger, y “Enemy” a los enemigos del juego.

El uso de etiquetas es práctico para scripts en los que se debe tener una referencia a un conjunto de GameObjects, pero no se quiere asignar esa referencia mediante drag & drop desde el editor. Es posible buscar los GameObject con un Tag determinado mediante el uso del método GameObject.FindWithTag(valor) [46].

4.1.4 Layers

Las capas o layers son usadas para reunir a GameObjects dentro de una misma categoría, de forma similar a las etiquetas. Son utilizadas, por lo general, para determinar objetos que se van a mostrar en la cámara, en el filtrado de colisiones, o en iluminación selectiva.

En el caso del tratamiento de colisiones, Unity ofrece una matriz de colisión por capas que permite indicar qué capas producirán colisiones con otras capas en específico [47].

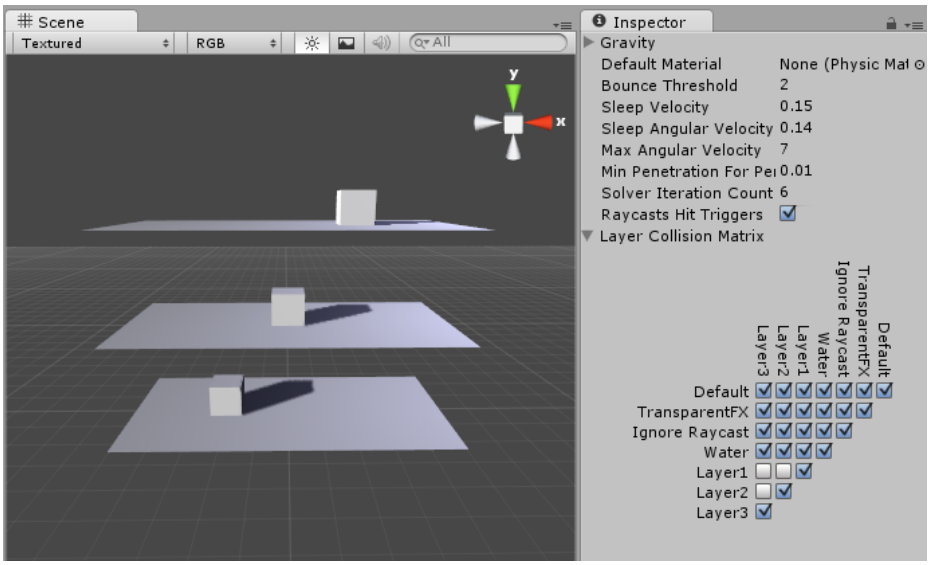


Fig. 10. Layer Collision Matrix, o matriz de colisión de capas. En ella se ve como se ha asignado que la capa “Layer1” solo colisionará con objetos de la misma capa. Fuente: documentación de Unity.

4.1.5 Prefabs

Un prefab es una colección almacenada que contiene uno o varios GameObject con componentes que tienen unos valores de propiedades marcados. Estos se crean desde GameObject obtenidos de la escena arrastrando un GameObject desde la jerarquía hasta la vista del proyecto, o se pueden crear directamente dentro de esa vista. Posteriormente, estos prefab pueden ser instanciados en la escena ya sea de forma estática instanciándolos en la escena usando el editor, o de forma dinámica, instanciándolos en tiempo de ejecución. Todas las instancias de un prefab comparten los valores en sus propiedades que se han indicado en el editor, aunque estas pueden cambiar individualmente para cada instancia en tiempo de ejecución.

4.1.6 Sistema de colisiones en Unity

El elemento principal que permite que las colisiones funcionen en Unity son los collider, o colisionadores, que definen la forma de un objeto para poder emitir colisiones físicas.

Un collider es invisible, y no tiene por qué encajar con la forma de la forma de la mesh (modelo) del objeto, lo cual se ve bastante debido a la posibilidad de usar un collider más simplificado que el modelo, aumentando el rendimiento sin afectar demasiado a la fidelidad del cálculo de las colisiones.

Los collider es uno de los elementos principales en las físicas de un juego en Unity, junto a los Rigidbody, CharacterControllers y Joints. Dado que los Collider son los más usados en este videojuego, se va a pasar a explicar su funcionamiento.

Colliders

Los collider conocidos como collider de tipo primitivo son aquellos con forma de caja (Box Collider), de esfera (Sphere Collider), y de cápsula (Capsule Collider), una especie de cilindro en la que las partes inferior y superior son semiesferas. Estos collider son los menos exigentes en lo referente a la computación de la información de las colisiones, y permiten obtener una aproximación representativa sobre un objeto más complejo. También es posible incluir más colliders como hijos del GameObject, aumentando así la precisión de la aproximación.

En caso de que este tipo de colliders no sean lo suficientemente precisos para lo que se quiere realizar, es posible usar los Mesh Collider, que permiten crear un collider con exactamente la misma forma que el mesh introducido, que puede ser el mismo que el modelo en cuestión. Sin embargo, estos collider son más exigentes en el procesador que los tipos primitivos, aunque se pueden usar en número reducido para mantener un equilibrio entre el rendimiento y funcionalidad.

Una buena regla para el uso de colliders es la de utilizar Mesh Colliders para objetos estáticos de la escena y colliders primitivos para aquellos objetos que se mueven en la escena, tal y como pueden ser proyectiles, un personaje, etcétera. De esta forma, se mantiene la calidad del escenario sin afectar mucho

al rendimiento, cosa que no es tan sencilla si se pretende usar Mesh Colliders tanto para los objetos estáticos como los objetos dinámicos.

Los collider pueden ser añadidos a un objeto sin que tenga un Rigidbody: estos son conocidos como colisionadores estáticos, o static colliders. Aquellos que sí que están enlazados a un objeto con Rigidbody se conocen como colisionadores dinámicos.

Para poder entender bien este concepto, hay que conocer el concepto de Rigidbody en Unity.

Rigidbodies

Un Rigidbody, o "Cuerpo Rígido", es el componente principal que habilita el comportamiento de físicas de un GameObject. Al añadir un Rigidbody a un objeto, este responderá a la gravedad y se moverá según su empuje y las colisiones con elementos que tengan un collider, ya sean dinámicos o estáticos.

Debido a que un Rigidbody controla el movimiento de un GameObject, no es recomendable cambiar su posición directamente desde su Transform, sino que se deben aplicar fuerzas sobre ese objeto, dejando que el motor de físicas sea el que calcula su siguiente posición.

Hay casos en los que se necesita que un GameObject tenga un Rigidbody pero su movimiento no esté controlado por el sistema de físicas. Este movimiento se considera un movimiento cinemático, o kinematic. Se puede indicar al Rigidbody que los movimientos van a ser realizados a través de scripts y no por el sistema de físicas mediante su valor Is Kinematic.

Una de las posibles utilidades de un Rigidbody marcado como kinematic es que un GameObject pueda interactuar con Triggers: los Triggers son aquellos collider que no se comportan como objetos sólidos, permitiendo pasar a través del collider otros objetos con collider sin que choquen, pero emitiendo información sobre el objeto que ha pasado a través. Esto es útil para implementar funcionalidades como detectar cuando un personaje ha pasado por una zona para activar una cinemática, por ejemplo.

Colisiones: colliders y triggers

Cuando ocurre una colisión, el motor de físicas llama a funciones con unos nombres en específico en los scripts que son componentes al GameObject que tiene como componente el collider. Estas funciones se pueden implementar en cualquiera de los scripts para realizar alguna acción como respuesta al evento emitido por la colisión.

En el momento que se detecta una colisión, se llama al método OnCollisionEnter. En las siguientes actualizaciones de físicas en las que se sigue detectando contacto, se llama al método OnCollisionStay. Al romperse el contacto entre los colisionadores, se llama al método OnCollisionExit. De forma análoga

para los Trigger, se llaman a los métodos OnTriggerEnter, OnTriggerStay y OnTriggerExit. La diferencia es que a los métodos OnCollision se le pasa por parámetro una referencia a los datos de la colisión, mientras que a los métodos OnTrigger se le pasa por parámetro una referencia a los datos del collider del objeto que ha entrado al trigger. De ambas formas se puede obtener información sobre el objeto que ha provocado la colisión.

Existe otra forma de detectar objetos más allá de las colisiones provocadas con el choque de dos collider y del uso de triggers: el raycasting.

Raycasting

La técnica de raycasting permite trazar una línea invisible que sigue un vector de dirección y una longitud a partir de un punto, conocida como 'ray'. Estos rays son emitidos desde un punto del espacio y pueden definirse como un vector direccional y una longitud, o definirse mediante el segmento entre el punto inicial y un punto de destino, que define implícitamente un vector dirección y una longitud.

En el caso de que se defina el raycast como el ray entre un punto y otro, se conoce como linecasting, aunque funciona del mismo modo.

La utilidad del raycasting es que el ray emitido es capaz de detectar GameObjects que tengan algún Collider como componente, lo cual es especialmente útil cuando se quiere obtener información sobre objetos que tenemos cerca pero sin la necesidad de que el objeto que emite el ray tenga que colisionar; o incluso para obtener colisiones a partir de un objeto que no tiene collider. Un ejemplo de esto último es el hecho de poder obtener información sobre algún objeto del entorno simplemente con apuntar la cámara hacia ese objeto: esto se realiza mediante el uso de raycast desde la cámara hasta que encuentra un objeto delante.

En Unity, los raycast se aplican mediante el método Physics.Raycast que recibe como parámetro el punto de origen, el vector de dirección, la distancia máxima, la máscara de capa (qué capas definidas en las físicas del juego son detectadas o ignoradas por este ray) y la información del impacto del ray en un objeto de clase RaycastHit que es definido como un parámetro de output (out en C#) por lo que será rellenado por la función [48].

En el caso de ser un ray definido entre dos puntos, el funcionamiento es similar, pero llamando al método Physics.Linecast, que recibe los parámetros del punto de inicio, punto de final, información del impacto (un objeto RaycastHit, al igual que Physics.Raycast) e información sobre la máscara de capa.

El objeto RaycastHit es una estructura usada para obtener información de un raycast. Esta estructura contiene bastante información sobre el punto en el que ha impactado, pues contiene información sobre

el collider impactado, a qué distancia se ha impactado, las coordenadas del punto en el que se ha impactado (en coordenadas globales), el rigidbody del objeto que se ha impactado (siempre que tenga alguno, si no, devuelve null); y otra información más avanzada sobre la geometría del collider, como puede ser las coordenadas baricéntricas del triángulo que ha sido impactado, el vector normal de la superficie impactada, el índice del triángulo impactado del collider, e incluso las coordenadas UV del lightmap y de la textura.

4.1.7 Interfaces de usuario en Unity

El elemento Canvas es un GameObject que contiene el componente Canvas. El Canvas define el área en el que todos los elementos de la interfaz van a ser dibujados, por lo que todo elemento de UI debe incluirse como hijo de un elemento Canvas.

De hecho, al crear cualquier elemento UI como puede ser Text o Image, en caso de no existir un Canvas en la escena se añade uno automáticamente, convirtiendo este elemento Canvas de forma automática en el padre del nuevo elemento.

Los elementos se dibujan en el mismo orden que en el que aparecen en la jerarquía de objetos de la escena. En el caso de querer añadir elementos de forma dinámica, se pueden ajustar en qué orden se quieren dibujar los elementos ordenándolos en la jerarquía mediante SetAsFirstSibling, que coloca el elemento al inicio de entre sus hermanos, SetAsLastSibling, que coloca el elemento al final de entre sus hermanos, y SetSiblingIndex, que coloca el elemento en el índice indicado.

El elemento Canvas tiene una propiedad Render Mode que permite cambiar el modo con el cual se renderizan los elementos de UI:

- Screen Space – Overlay: los elementos de UI se renderizan siempre encima de la escena, colocando los elementos en la pantalla.
- Screen Space – Camera: similar a Screen Space – Overlay, pero en este modo el Canvas se sitúa a una distancia especificada de la cámara. En este modo, los elementos de la UI se ven afectados por las propiedades de la cámara.
- World Space: en este modo, el Canvas se comporta como un elemento más de la escena. Representa a las interfaces diegéticas, pues están integradas dentro del mundo de juego.

En los dos primeros modos, en caso de que cambie el tamaño de la pantalla o la resolución, el Canvas se ajustará automáticamente al nuevo tamaño de la pantalla.

En los GameObject relacionados con la UI, no existe el componente Transform: este se ve sustituido por el componente Rect Transform. Los Rect Transform tienen posición, rotación y escala, pero también tiene anchura, altura, e información sobre los anclas y pivotes [49].

Los pivotes de un Rect Transform afectan a las modificaciones de la rotación, tamaño y escala, realizando los cambios en forma relativa a la posición del pivote.

Las anclas, o anchors, de un Rect Transform determinan la posición del Rect Transform de forma relativa a una posición, la posición del ancla, en el elemento padre. Por ello, si un elemento tiene el ancla en el centro del elemento padre, todo cambio de tamaño del elemento padre hará que el elemento hijo se mantenga siempre en el centro, a la misma distancia del pivote que como se indicó.

También es posible indicar el ancla en más de una posición, lo cual afecta al tamaño del elemento. Ya que el ancla determina la distancia de los cuatro puntos del Rect Transform (pues es un rectángulo) a él, por ejemplo, en el caso de dividir el ancla en dos puntos, se asegura que dos de los puntos estén a la misma distancia de un punto de anclaje y que los otros dos puntos del rectángulo estén a la misma distancia del otro punto de anclaje, forzando un cambio de tamaño del elemento para cumplir esa condición.

La posición de los puntos de anclaje se determina en forma de porcentajes respecto al tamaño del elemento padre.

4.2 Puesta en marcha: preparación de proyectos para GoogleVR

Para la realización de este proyecto, se ha decidido utilizar el framework para desarrollo en realidad virtual GoogleVR, dada su compatibilidad con Unity, sencillez, y la compatibilidad con los dispositivos usados durante el desarrollo.

Para poder integrar GoogleVR con un proyecto de Unity, es necesario descargar la librería completa desde su repositorio de GitHub, la cual contiene todo tipo de assets, como scripts, prefabs, e incluso otros recursos como una escena de demostración de GoogleVR que es compatible tanto con Cardboard como con Daydream. El SDK de GoogleVR para Unity se encuentra en el repositorio público googlevr/gvr-unity-sdk [50].

Tras integrar el paquete de Unity incluido en el SDK, ya se pueden incorporar todos los assets del SDK dentro de cualquier proyecto, ya sea uno recién creado o uno ya existente que se quiere adaptar para realidad virtual.

Si bien es cierto que el SDK incluye bastantes prefabs y scripts, la gran mayoría es para mejorar la experiencia de usuario con la VR de Daydream. Para Cardboard, el número de assets necesarios es reducido, pues no se deben mantener las funcionalidades como son el controlador de tipo puntero que dispone Daydream, las áreas de seguridad que indican si el usuario está moviendo el controlador hacia una zona en la que puede haber obstáculos, entre otras muchas otras funcionalidades.

Por ello, para que se pueda empezar a desarrollar un proyecto, o simplemente para adaptarlo para Cardboard, se han de realizar los siguientes pasos, sin un orden en específico, excepto en aquellos casos que dependan de un prefab ya añadido:

- Desde la carpeta Prefabs del SDK, añadir el prefab GvrEditorEmulator a la escena. Este prefab sirve para facilitar las cuestiones de debugging, pues permite rotar la cámara e inclinarla con el ratón simulando así el movimiento de la cabeza, además de añadir la posibilidad de clicar, simulando así una pulsación al visor de Cardboard.
- Desde la carpeta Prefabs/EventSystem, añadir el prefab GvrEventSystem a la escena. Este es el sistema de eventos por defecto de GoogleVR, que reemplaza al prefab de Unity de Event System por defecto. En vez de utilizar el componente StandaloneInputModule de Unity, utiliza el componente GvrPointerInputModule, que permite que se puedan interactuar con los elementos de interfaz y objetos 3D dentro de una aplicación de GoogleVR: este script está diseñado para que funcione tanto con punteros 3D con el controlador Daydream, o punteros basados en mirada "Gaze-based-Pointer", para Cardboard. Este prefab permite que se pueda interactuar con GameObjects que tengan como componente GvrPointerPhysicsRaycaster en caso de objetos 3D o GvrPointerGraphicRaycaster en caso de elementos UI, o con aquellos GameObjects que incluyan el componente Event Trigger.
- Desde la carpeta Scripts/EventSystem, añadir el script GvrPointerPhysicsRaycaster como componente al GvrEventSystem que se ha añadido a la escena. Este script permite que los punteros (de ambos tipos) puedan interactuar con objetos 3D que existen en la escena.
- Desde la carpeta Scripts/EventSystem, añadir el script GvrPointerGraphicRaycaster a todo elemento Canvas, que permite que se pueda interactuar con los elementos de la interfaz dentro de ese Canvas a través de los punteros de VR. Es necesario eliminar el componente GraphicRaycaster que es añadido por defecto en los elementos Canvas.
- Desde la carpeta Prefabs/Cardboard, añadir el prefab GvrReticlePointer a la cámara principal de la escena. Este prefab añade la retícula del visor de mirada de Cardboard, es decir, el punto central que permite saber al usuario hacia dónde está apuntando. Este incluye el script de GvrReticlePointer, que permite configurar varios parámetros sobre cómo se visualiza la retícula y cuánto de largo es el Raycast que se va a trazar desde el centro de la cámara hacia delante. Ajusta de forma automática el tamaño de la retícula cuando se está mirando hacia un elemento con el que se puede interactuar.
- Para mejorar las tareas de debugging, es recomendable añadir a la escena del prefab GvrInstantPreview que se encuentra en Prefabs/InstantPreview, el cual permite hacer el debug desde el dispositivo móvil conectado por USB al ordenador. La característica principal que esto ofrece es la posibilidad de ver en el dispositivo móvil una previsualización de cómo se vería la escena si se estuviese ejecutando dentro del dispositivo, aunque en realidad lo que hace es enviar por USB la imagen simulada desde el ordenador, mientras que el dispositivo envía al ordenador los datos sobre rotación para así desplazar la cámara dentro de la escena. Esto es posible gracias a la aplicación Instant Preview, la cual viene incluida dentro del SDK de Google. También es posible utilizar streaming Wi-Fi en vez de conexión USB [51].

Una vez que se han añadido estos prefabs y scripts a la escena, hay que configurar el proyecto de Unity para que sea compatible con VR. Para ello, hay que abrir en el Inspector las PlayerSettings, accesibles desde el menú Edit -> Project Settings -> Player. Dentro de la pestaña de Android, en la sección XR Settings, 'Virtual Reality Supported' debe estar marcado. Dentro de Virtual Reality SDKs hay que añadir la SDK de Cardboard. Esto se realiza pulsando el icono del signo de suma y seleccionando Cardboard en el desplegable.

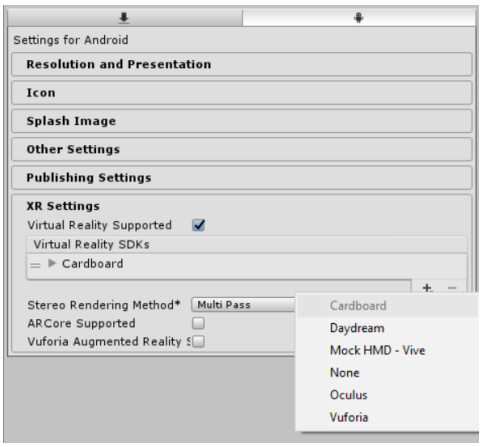


Fig. 11. Pestaña de ajustes de Android dentro del inspector sobre el elemento de configuración PlayerSettings. Se observa cómo la opción Virtual Reality Supported dentro de XR Settings está marcada, cómo Cardboard aparece dentro de Virtual Reality SDKs, y las SDK que Unity permite seleccionar para el proyecto.

4.3 Diseño del videojuego

En este apartado se van a detallar las decisiones de diseño tomadas respecto a la ambientación y las mecánicas del juego, explicando a nivel de diseño el objetivo del juego, el escenario en el que se lleva a cabo, y las diferentes mecánicas desarrolladas.

4.3.1 Objetivo del juego

El videojuego desarrollado consiste en un juego de disparos, en el que los objetivos, que son dianas, aparecen alrededor del jugador y éste puede disparar moviendo la cabeza para apuntarles, disparando al accionar el input del controlador que usa el usuario, ya sea un mando de Xbox, un mando bluetooth genérico, un ratón, o el botón de Cardboard.

Fig. 12. Primera página del panel de "Cómo jugar".

El objetivo del juego consiste en conseguir la mayor cantidad de puntos posible. El aliciente del mismo es aguantar el máximo tiempo consiguiendo la mayor cantidad de puntos. Teniendo en cuenta que la dificultad del juego va aumentando según el tiempo que haya transcurrido en la ronda, se proporciona un reto incremental mientras que se mantiene la misma fórmula de jugabilidad. Debido a que el tiempo máximo se extiende mediante mejoras obtenidas a partir de los objetivos bonus, se añade un factor de aleatoriedad y azar a la duración de las partidas.

4.3.2 Escenario del juego

El juego transcurre en una isla situada en medio del océano, lugar predilecto de retiro para el avatar que se maneja. El personaje llega a la isla en su bote, donde lleva los recursos necesarios para acampar allí durante varios días. Uno de sus pasatiempos favoritos es practicar su puntería al lanzar hachas hacia dianas, las cuales aparecen gracias a un sistema de lanzamiento de discos que ingenió.

Al comenzar el juego, la escena de menú transcurre en el bote en el que llega a la isla, y al empezar la partida, el jugador está sobre la plataforma del centro de la isla, estando recién activada la máquina, preparado para disparar a las dianas.

Fig. 13. Menú principal del juego, en el que se muestra el nombre del juego, y botones para jugar, ver las instrucciones, ir al menú de opciones, y salir del juego.

El estilo elegido para los objetos y escenarios es el diseño low poly, o diseño poligonal. Este se caracteriza por modelos con bajo número de polígonos, dándole una estética simplista y minimalista. Debido a su fácil texturizado, pues se aplican técnicas de *flat shading*, que consisten en texturas que sólo contienen colores planos, se facilita la tarea de crear nuevos modelos o modificarlos.

Fig. 14. Detalle del bote inicial, donde aparecería el menú principal del juego. Desde él, se escucha el sonido del mar y la música emitida por el reproductor de música que trae consigo el personaje.

4.3.3 Mecánicas del juego

En este apartado se van a explicar las decisiones de diseño tomadas en lo que se refiere al videojuego en cuestión, sin entrar en detalles técnicos, respecto a las mecánicas existentes en el juego. Los detalles técnicos se explicarán en el apartado de implementación.

Sistema de aparición de dianas

Los objetivos irán apareciendo alrededor del jugador en un arco de 180 grados, es decir, en un campo determinado por una semicircunferencia cuyo centro está situado en la posición del jugador. La posición del jugador es estática, por lo que no se permite más movimiento que el de la cámara, para apuntar.

El ángulo inicial de aparición de los objetivos ha sido elegido de tal forma para facilitar la jugabilidad a usuarios que no puedan voltear la cabeza mientras juegan. Este puede ser el caso de usuarios con movilidad reducida, por lo que es una decisión de diseño tomada para mejorar la accesibilidad motora del juego. También mejora la jugabilidad para jugadores que simplemente están sentados en una silla o no quieren tener que realizar movimientos demasiado bruscos para dar la vuelta.

Este ángulo marcará los puntos de aparición (spawnpoints) de los objetivos respecto al punto inicial del jugador al iniciar el juego. En caso de ángulos demasiado amplios, podría provocar que el jugador se maree con facilidad o, en caso de jugadores con movilidad reducida, que sea injugable.

Fig. 15. Dianas dentro del juego. Se puede observar cómo aparecen en el arco definido por la isla, siempre colocadas a la misma distancia respecto al jugador, situado en la plataforma de madera central. El arco está señalizado mediante piedras y arbustos.

También es importante ajustar el ángulo vertical de aparición, para que los objetivos puedan aparecer algo arriba o algo abajo del punto de inicio, pero que el jugador no tenga que apuntar directamente hacia arriba o hacia el suelo, por el mismo motivo que se ha expuesto en el párrafo anterior. Para ello, se han aplicado limitaciones relacionadas con la dificultad del juego que interactúan con la altura tomada por los objetivos cuando aparecen.

Según se incrementa la dificultad del juego, más probable es que las dianas que aparecen se desplacen a lo largo del arco en un ángulo determinado al azar, a una velocidad constante que se irá incrementando según la dificultad.

Tipos de dianas

Existen cuatro tipos diferentes de dianas, las cuales otorgan diferentes resultados tras impactar sobre ellas:

- Diana -1 pt.: al impactar sobre ella, penaliza al jugador restando un punto de su marcador.
- Diana 1 pt.: otorga al jugador un punto al ser impactada.
- Diana 2 pts.: otorga dos puntos al jugador tras ser impactada.
- Diana de bonus: otorga al jugador una bonificación al azar dentro de las seis bonificaciones disponibles. Estas son: sumar quince puntos al marcador, aumentar el tiempo restante de ronda treinta segundos, duplicar los puntos obtenidos durante un tiempo determinado, aumentar la velocidad de recarga y por tanto la cadencia de tiro de forma temporal, otorgar mejora de arma láser de forma temporal, y otorgar mejora de arma explosiva durante un tiempo. Estas bonificaciones serán explicadas en uno de los próximos apartados.

Desde el primer diseño se ha descartado la opción de otorgar más puntos o más bonificación según la precisión al impactar sobre las dianas. Esto se debe a la poca puntería posible mediante el uso de un headset de Google Cardboard y a la poca fidelidad del apuntado respecto al punto indicado por la retícula de apuntado proporcionada por defecto, además de la intrínseca pérdida de precisión al usar proyectiles erráticos como es el caso de las hachas arrojadizas.

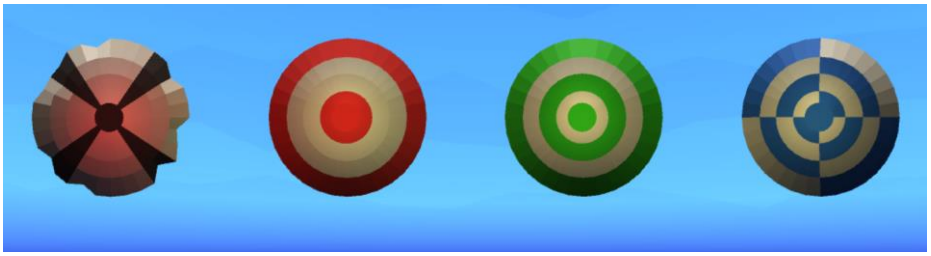


Fig. 16. Modelos de las cuatro dianas diferentes. De izquierda a derecha: diana de menos un punto, diana de un punto, diana de dos puntos, diana de bonus. Las dianas tienen diferente formas y patrones para su fácil distinción en casos de daltonismo.

Modo de disparo

La forma principal de interacción con las dianas es mediante el impacto de proyectiles sobre ellas. En este caso, los proyectiles diseñados son hachas arrojadizas. Durante la partida, el jugador puede ver a su derecha el hacha en cuestión.

Estos proyectiles se mueven en línea recta, a pesar de estar siguiendo una rotación constante cuya utilidad es simular la trayectoria del hacha según su filo y parte por la cual se ha lanzado. Por simplicidad y cohesión con el sistema de la retícula de apuntado y feedback auditivo, no se ha implementado que su trayectoria se vea afectada por la gravedad.

Sin embargo, estos proyectiles tienen una velocidad de movimiento determinada no modificable. Esto significa que tardan cierto tiempo en llegar al objetivo: esto aumenta la dificultad ya que se requiere estimar cuánto tiempo necesita el hacha para llegar al objetivo e impactar antes de que este desaparezca. En el caso de dianas que se desplacen a lo largo del arco una vez hayan aparecido, se requiere aún más destreza para determinar hacia qué dirección y cuándo se debe lanzar el proyectil para acertar.

Fig. 17. Proyectiles de hacha arrojadiza dirigiéndose a una diana.

Para evitar un lanzamiento indiscriminado de proyectiles, lo cual va en contra del principio del videojuego de tener la precisión necesaria para acertar en las dianas, pues permitiría ignorar por completo esa precisión, se ha incluido un sistema de recarga entre disparo y disparo, lo cual establece un valor de cadencia de tiro. Una vez que se realiza un disparo, no se puede realizar otro hasta que transcurra un tiempo determinado. Este tiempo se considera tiempo de recarga y puede ser reducido mediante uno de los bonus temporales disponibles en el juego.

Aun así, se han incluido dos modificaciones en forma de mejora temporal que reducen la precisión necesaria para impactar sobre un diana: arma láser y arma explosiva, las cuales reducen el tiempo de recorrido del proyectil y amplían la zona de eficacia del proyectil, respectivamente. Se entrará en más profundidad respecto a estas mejoras en la sección de sistemas de bonus.

Fig. 18. Arma láser siendo utilizada.

Sistema de puntuación, tiempo y final de partida

El objetivo del juego, como ya se ha mencionado, consiste en conseguir el mayor número de puntos posible dentro del tiempo concedido. Este objetivo se ha diseñado según el modelo de récords personales, por lo que siempre que terminen las rondas se almacenará la puntuación obtenida, comparándola con la mejor obtenida anteriormente. En caso de ser una puntuación mejor a la mejor puntuación obtenida anteriormente, se considerará como nuevo récord en el juego y se almacenará junto al tiempo que ha durado la partida.

Las partidas tienen un límite de tiempo, el cual, como ya se ha mencionado, se puede extender mediante una de las mejoras puntuales que permiten aumentar el tiempo restante.

Fig. 19. Indicador de tiempo extra: se muestra como se ha añadido 30 segundos al marcador. Este texto se desplaza hacia arriba, desvaneciéndose.

Cuando termina el tiempo, se concluye la ronda: en ese momento es cuando se muestra al usuario su puntuación final y el tiempo que ha durado la ronda, además de indicar si es un nuevo récord o no.

Fig. 20. Pantalla de final del juego, donde se indica que el jugador ha conseguido 65 puntos en un tiempo de 2 minutos y 16 segundos. Indica, además, que no ha batido el récord, pues el récord anterior es de 178 puntos, que es superior a 65.

Sistema de bonus: mejoras puntuales y mejoras temporales

Dentro del sistema de bonus podemos distinguir entre dos tipos de bonus: las mejoras temporales y las mejoras puntuales.

Las mejoras temporales consisten en mejoras que se aplican durante un número determinado de segundos. En este caso, todas duran quince segundos. Existen cuatro mejoras temporales diferentes:

- Velocidad de recarga mejorada: durante el tiempo establecido, el tiempo entre disparos se ve reducido a un cuarto del tiempo por defecto. En el caso de haber realizado un disparo el cual todavía se está recargando, el nuevo tiempo se aplicará para esa recarga, realizándose al instante si se calcula que con el nuevo tiempo debe de haber concluido ya.
- Puntuación doble: durante su duración, todos los puntos obtenidos serán multiplicados por dos. Esto también afecta a la mejora puntual de quince puntos, y sobre las dianas de puntos negativos, que restarán el doble.
- Arma láser: el arma de hachas arrojadizas se reemplaza por un cristal que dispara proyectiles láser que llegan al instante al objetivo. Esto reduce por completo el tiempo entre disparo e impacto, haciendo más fácil la tarea de acertar sobre dianas que están a punto de desaparecer o que están en movimiento.
- Arma explosiva: los proyectiles del arma ahora son explosivos y detonan cuando están a la misma distancia que el jugador de las dianas, por lo que es capaz de acertar en varias dianas a la vez en un área dado por el radio de la explosión. Esto reduce, además, la necesidad de apuntar directamente a las dianas para impactar sobre ellas.

Todas estas mejoras son compatibles entre sí. En el caso de las dos últimas mejoras, esto se convierte en algo peculiar, pues pueden existir cuatro combinaciones distintas de armas según qué mejoras estén activadas:

- Hacha arrojadiza normal: cuando ninguna de las mejoras de láser ni explosivos están activas, el arma es el hacha arrojadiza por defecto, que tiene tiempo de desplazamiento y sólo acierta a la diana sobre la que impacta directamente.
- Hacha arrojadiza con detonante: activo cuando sólo la mejora de explosivos está activada. Posee tiempo de desplazamiento, al igual que el hacha normal, pero no requiere que impacte directamente con una diana, pues explota en área.
- Cristal láser: activado cuando sólo la mejora de arma láser está activada. Los proyectiles no tienen tiempo de recorrido, y sólo pueden impactar sobre una diana.

- Cristal láser explosivo: posee las ventajas tanto de las armas láser, por no tener tiempo de recorrido, como de las armas explosivas, pues puede atacar en área. Se considera la mejor combinación posible junto a la mejora de recarga rápida, pues permite barrer una zona entera en cuestión de instantes.

Toda mejora puntual es aquella que no tiene duración, sino que marca mejoras permanentes. En el juego se han incluido dos: extensión de tiempo restante y quince puntos extra. Cabe decir que en el caso de tener activada la mejora temporal de obtención doble de puntos, en vez de obtener quince puntos se obtendrán treinta.

La bonificación de extensión de tiempo se considera una de las más importantes pues permite seguir ganando puntos, ya que extiende el tiempo que va a durar la ronda. Esto se traduce en aumentar la dificultad de la ronda, pues cuanto más dura una ronda más sube la dificultad, lo que a su vez significa que las dianas van a aparecer más a menudo que en dificultades menores.

4.3.4 Diseño de la interfaz de usuario

En proyectos no relacionados con la VR, la UI (user interface) suele aparecer sobrepuesta al resto de elementos de la pantalla, para mostrar cosas tales como la vida, la puntuación, entre otros, en lo que se denomina HUD (Heads Up Display). Este tipo de interfaz de usuario se denomina no diegética (non-diegetic): no existe en el mundo, pero tiene sentido para el jugador en el contexto del juego.

En Unity, añadir una interfaz no diegética se suele hacer marcando en el Canvas de la interfaz el Render Mode a Screen Space – Camera o Screen Space – Overlay.

Sin embargo, esta forma de hacer interfaces no funciona en la VR. Al situar los objetos tan cerca de nuestra visión, siempre superpuesto a todo, nuestros ojos no son capaces de enfocar la visión a algo que se sitúa tan cerca. Además, el modo de renderizado del Canvas 'Screen Space – Overlay' no está soportado en Unity VR [52].

Por lo tanto, tenemos dos alternativas para crear la interfaz de usuario: UI espacial y UI diegética.

Interfaz de usuario espacial

En el caso de la UI espacial, el Render Mode del Canvas es 'World Space', lo que significa que los elementos de la interfaz van a ser colocados dentro del entorno de juego, es decir, situados en el mundo de la escena.

Es importante tener en cuenta dónde se va a situar esta interfaz, pues si está demasiado cerca al usuario puede provocar fatiga visual (por la misma razón que no se usa la interfaz no diegética), y si está demasiado alejada puede dar la sensación de que el usuario tiene que enfocar la vista al horizonte, algo que

puede ser molesto al pasar a mirar hacia un elemento cercano. Por ello, es conveniente colocar la interfaz a una distancia en la que la lectura sea cómoda y con un tamaño adecuado; incluso de forma dinámica, dependiendo de cómo se quiera interactuar con ella.

Si se coloca la interfaz a una distancia determinada del usuario, puede suceder que la interfaz se esconda detrás de otros objetos que están más cercanos. Esto se puede solucionar mediante la aplicación de shaders sobre los elementos de la interfaz que fuercen su dibujado a pesar de estar detrás de elementos, evitando así el efecto de clipping [53].

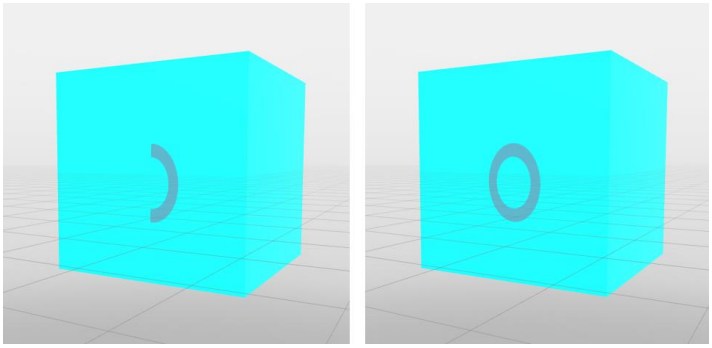


Fig. 21. Representación de un elemento de UI espacial que representa la retícula que indica dónde está apuntando el usuario. A la izquierda, la retícula está siendo renderizada con el shader estándar de UI. Se puede apreciar que parte de la retícula está oculta debido al efecto de clipping con el cubo de la escena. A la derecha, la misma retícula es ahora visible a pesar de hacer clipping con el cubo debido a que está usando un shader que evita ese efecto. Fuente: documentación de Unity.

Es posible enlazar esta interfaz con la cámara, para que al desplazar la cámara también se mueva la interfaz con ella, dando la impresión de que la interfaz está fija en la pantalla. Esto funciona bien para elementos pequeños, como puede ser textos pequeños o una retícula para apuntar, pero en el caso de elementos grandes que cubran gran parte de la pantalla puede provocar nauseas. Una posible solución para poder usar elementos grandes que sigan a la cámara es añadir un pequeño retardo entre el movimiento de la cámara y el movimiento de la interfaz, para que así el usuario pueda familiarizarse con el entorno sin que la interfaz esté bloqueando su campo de visión, manteniendo la funcionalidad de poder colocar la interfaz delante del usuario.

En el caso de este proyecto, se han utilizado interfaces de usuario espaciales en el caso de menús, como pueden ser los menús de opciones y el menú principal, o la ventana de información mostrada al terminar la ronda.

Interfaz de usuario diegética

La alternativa a las interfaces de usuario espaciales son aquellas interfaces que están integradas en el entorno, es decir, que son los mismos objetos los que contienen la información que se quiere mostrar. Ejemplos sencillos de este tipo de interfaz pueden ser relojes de pared que muestren la hora, una televisión, un visor holográfico en un arma futurista...

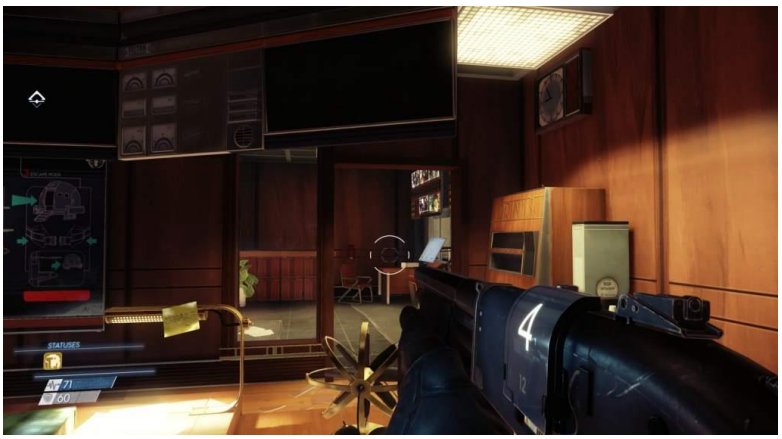


Fig. 22. El videojuego '*Prey*' muestra el ejemplo perfecto de interfaz de usuario diegética combinada con no-diegética. En la imagen se puede ver cómo la información sobre la munición del arma aparece en la misma arma, y no en una HUD separada del mundo. Sin embargo, la información sobre la vida del jugador se sitúa en la pantalla, fuera del mundo.

La diferencia principal entre las interfaces diegéticas y las espaciales es que las espaciales siguen estando dentro de la categoría de los elementos de interfaz que están fuera del contexto del juego, a pesar de que estén en el mundo del juego. Las interfaces diegéticas, por otra parte, también están en el mundo del juego, pero pertenecen al contexto del juego en cuestión.

En este caso, se ha empleado la interfaz diegética en el indicador de puntos, tiempo y bonus. Cerca del jugador, en la ronda, se encuentran dos carteles con forma de lienzo en el que se ve escrita la información actual.

4.4 Implementación del videojuego

En esta sección, se van a tratar las cuestiones sobre la implementación de los elementos expuestos en el diseño del videojuego, ya sean las cuestiones sobre la interfaz de usuario, las mecánicas del juego y aquellos componentes diseñados para mejorar la accesibilidad del juego.

4.4.1 Implementación de mecánicas del juego

A continuación, se explican ciertos detalles sobre la implementación de los distintos sistemas que trabajan para ofrecer las mecánicas del juego. Esto incluye el sistema de aparición de dianas, la implementación de las dianas, el sistema de puntos, el sistema de bonus, cómo funcionan las diferentes armas y cómo interactúan con las dianas; el sistema de tiempo de juego, y final de partida.

Sistema de aparición de dianas

El componente Target Spawner es el encargado de gestionar la aparición de las dianas alrededor del jugador. Target Spawner tiene referencias al GameObject del jugador, al componente ScoreManager y al BonusManager, las cuales son usadas para asignarlas a las dianas que se instancian, ya que, en el caso de las dianas de puntuación, estas requieren una referencia al ScoreManager y en el caso de dianas de bonificación, estas requieren una referencia al BonusManager. En lugar de hacer que las dianas sean las que busquen la referencia dentro de la escena, se ha hecho de tal modo para aumentar el rendimiento.

Existen una gran cantidad de parámetros que permiten ajustar la dificultad del juego:

- ChanceOfMovingTargets: probabilidad de que las dianas instanciadas se muevan a lo largo del arco una vez instanciadas.
- MaximumMovingAngle: la máxima amplitud del arco que recorrerán las dianas instanciadas que se muevan a lo largo del arco.
- MovingTargetSpeed: velocidad que tomarán las dianas que se muevan a lo largo del arco.
- TimeBetweenSpawns: tiempo en segundos que determina cuántos segundos deben pasar entre cada instanciación de diana.
- TimeUntilTargetDisappears: tiempo en segundos que determina cuántos segundos deben pasar hasta que una diana con la que no se ha interactuado se desactive.

Para la computación de la altura a la que deben aparecer las dianas, se utilizan los siguientes parámetros:
- BaseHeight: altura base. Esta altura es tomada como referencia para instanciar las dianas a una altura en un intervalo de [MinRandomHeight + BaseHeight, BaseHeight + MaxRandomHeight].
- MaxRandomHeight: altura máxima que tomarán las dianas. Tendrá valor positivo, pues es relativo a la altura base.
- MinRandomHeight: altura mínima que tomarán las dianas. Tendrá valor negativo, pues es relativo a la altura base.

Tanto las variables de movimiento, tiempo de desaparición y alturas tienen cada una otra variable terminada en “step” que determina cuánto debe variar esos valores cuando se aumenta la dificultad. El valor de la dificultad es un número entero que va desde 1 hasta 8. Cada vez que pasa el tiempo asignado para incrementar la dificultad, se aumenta en 1 ese valor, sumando o restando a los parámetros antes

mencionados el valor de step que tienen asignados. En caso de que el factor de dificultad ya sea 8, no se harán cambios.
Para que el Target Spawner sea capaz de instanciar dianas, se ha establecido un parámetro de tipo array que permite introducir los diferentes tipos de dianas y las probabilidades que hay de que se instancie cada una de ellas, mediante un algoritmo de probabilidad acumulada. En este caso, hay 4 dianas: las dianas de -1 punto, que tienen un 15% de probabilidad de aparecer, las dianas de +1 punto, con un 30%, dianas de +2 puntos, con 25%, y dianas de bonus, con un 30%.

La posición de las dianas queda definida por un arco con una amplitud determinada. Esta amplitud es de 180º grados por defecto o 90º grados en el caso de la zona de juego reducida, pero se ha preparado el algoritmo de tal forma que se permita cualquier valor. La bisectriz del ángulo del arco siempre estará en la misma dirección que la dirección del jugador al iniciar la partida. Se escoge un ángulo aleatorio entre la amplitud del arco, se transforma en radianes y se asigna su posición en el mundo en una circunferencia de 1 unidad de radio, teniendo en cuenta que X es el coseno del ángulo, Y es cero, y Z es el seno del ángulo. Tras obtener esa posición, utiliza el método Transform.TransformPoint que permite desplazar un punto desde el centro del mundo mediante una magnitud escalar, siendo el radio de la circunferencia deseada esa magnitud.

Tras conocer la posición, se instancia uno de los GameObject al azar teniendo en cuenta sus probabilidades. Dado que todos los tipos de dianas tienen el scripts que heredan de TargetScript, como en el caso del script ScoreTarget y BonusTarget, se obtiene, asignando los valores específicos de cada subcaso de script: referencia al Score Manager en el caso de ScoreTarget y referencia al BonusManager en el caso de BonusTarget; y los comunes a ambos: altura, tiempo para ocultarse y referencia al jugador.

Además, en caso de que se tenga que según la probabilidad se deba permitir a las dianas que se muevan a lo largo del arco, se les añade el componente MoveTarget, indicando el ángulo en el que se deben de mover y a qué velocidad. Este ángulo se ajusta de tal modo que las dianas nunca se desplacen fuera el arco total definido.

Implementación de las dianas y su jerarquía

Los componentes asociados a las dianas se encuentran dentro de una jerarquía de clases dado a que las dianas comparten muchas características como las de movimiento, aparición, desaparición, daño recibido, entre otras; pero la funcionalidad que realizan cuando son impactadas las diferencia entre ellas.

Generalizando, hay dos tipos de dianas: aquellas que manipulan la puntuación (dianas de -1, +1 y +2 puntos) y aquellas que otorgan bonus (diana de bonus). Por ello, se ha diseñado la clase abstracta TargetScript que contiene las funcionalidades comunes de ambas dianas, y es heredada por ScoreTarget y BonusTarget.

TargetScript contiene los métodos necesarios para vincularse con el componente de brújula, reproducir sonidos, desplazarse hasta el punto indicado por la altura que se le ha ajustado en el instanciador de dianas, desactivarse tras el tiempo ajustado, realizar las tareas de desactivación en lo que se refiere a animación; y además incluye el método abstracto DealDamage que es definido por la interfaz IDamageable.

Esta interfaz es empleada para determinar que un objeto puede ser atacado, siendo en este caso las dianas. El método será llamado por las armas cuando impacten sobre cualquier elemento que implemente la interfaz, lo cual se explicará en la parte del sistema de armas y proyectiles.

Las clases BonusTarget y ScoreTarget se ocupan de implementar el método DealDamage mediante override. En el caso de ScoreTarget, que contiene un parámetro que indica cuantos puntos debe otorgar, se llama al método AddPoints de su referencia a ScoreManager, pasándole el número de puntos, posteriormente llamando al método AnimateDestruction implementado por la clase abstracta. En el caso de BonusTarget, se llama al método ExecuteRandomBonus de su referencia al BonusManager, y se llama al método AnimateDestruction. El método AnimateDestruction es el que realiza las tareas de destrucción, mencionado anteriormente.

Sistema de puntos

El sistema de puntos es gestionado por el componente ScoreManager, el cual está asignado a un único GameObject en la escena. Éste se encarga de mantener la cuenta de los puntos, modificar la cantidad de puntos, e indicar el estado y los cambios tanto por la interfaz de usuario (medio visual) como por síntesis de voz en caso de que los modos de navegación con TTS estén activados (medio auditivo).

Sus atributos principales son la cantidad de puntos almacenados y el multiplicador de puntos. Los puntos se añaden mediante AddPoints, que actualiza los elementos de la interfaz y voz para indicar la nueva puntuación. A este cambio de puntos se le aplica el valor del multiplicador de puntos, que en situaciones normales equivale a 1. Mediante la mejora temporal de bonus que duplica los puntos adquiridos, este multiplicador será 2, por lo que se ganarán y se perderán el doble de puntos durante el tiempo determinado.

Sistema de tiempo de ronda y final de partida

El sistema de tiempo se gestiona mediante el componente TimerManager, que contiene la información sobre los segundos que han pasado en la ronda, los segundos totales, los segundos que quedan, y cuántos segundos deben de pasar para aumentar la dificultad. Mediante una corrutina ejecutada cada segundo, se aumenta el número de segundos transcurridos y se actualiza en la interfaz. En caso de que el número de segundos transcurridos sea múltiplo de los segundos necesarios para aumentar la dificultad, se llamará

al método IncreaseDifficulty del TargetSpawner. También incluye un método que permite aumentar el tiempo restante, para así extender la duración de la ronda.

Cuando el tiempo se acaba, se ejecutan las acciones de final de ronda. Para ello, se llama al método TriggerMatchEnd del componente GameEndManager cuando termina la corrutina de tiempo (esta corrutina termina cuando el tiempo restante es igual a 0).

El componente GameEndManager es el encargado de recopilar la información final sobre la ronda y deshabilitar componentes de la partida, como pueden ser las el instanciador de dianas y el sistema de armas. En su método TriggerMatchEnd se deshabilitan todos esos elementos y se obtienen los puntos obtenidos por el jugador desde ScoreManager y el tiempo transcurrido desde TimerManager. Una vez obtenidos estos datos, se intenta acceder los valores registrados del récord de puntuación hasta el momento, almacenados en PlayerPrefs, estando la puntuación del récord en la clave "record_score" y el tiempo en "record_time". Al comparar la puntuación actual con la puntuación récord, si la puntuación actual es superior se sobrescribe en las PlayerPrefs. Si no, se deja tal y como está. En todos los casos se muestra al usuario una pantalla en la que se menciona su resultado actual, el resultado del récord, y si se ha superado este record. Los resultados se escriben sobre elementos de la interfaz de tipo texto, resaltando el contenido importante del texto mediante negrita, ya que soporta texto enriquecido.

Sistema de armas

En el juego existen principalmente dos armas diferenciadas: el hacha y el cristal. Cada una de ellas es un GameObject diferente, y solo puede estar una de ellas activada en el mismo momento. Ambos GameObject están gestionados por el componente WeaponManager, que permite intercambiar qué armas están activas a través de la propiedad UsingLaserWeapon, que oculta el arma de hachas arrojadizas y muestra el arma laser si está activada, y al revés en caso contrario. Además, permite indicar si se debe usar munición explosiva a través de la propiedad Explosive, y permite asignar el multiplicador de tiempo de recarga a ambas armas con la propiedad FireRateMultiplier.

Cada GameObject de arma tiene un script diferente. En el caso del hacha, es el script ThrowingAxeHand, y en el caso del cristal, es LaserWeapon. Dado a que ambas comparten muchas características y funcionalidades, como la forma de comprobar si se puede disparar, el tiempo de recarga, y la acción de disparar, se ha creado la clase abstracta Weapon que contiene todo lo común entre las dos armas, y se ha hecho que los dos scripts hereden de esa clase.

Weapon ofrece los atributos fireRate, explosive, fireRateMultiplier y previouslyFiredTime, y se ocupa de implementar el método Update, CanShoot y parcialmente el método Shoot. Update comprueba cuándo se ha realizado input. En caso de haber realizado input, se comprueba si se puede disparar, información dada por el método CanShoot. En caso positivo, se llama al método Shoot.

CanShoot se ocupa de comprar el tiempo actual en el que se ha intentado atacar con el tiempo anterior de tiro. En caso de que la diferencia entre el tiempo actual y el tiempo anterior de tiro sea mayor que la frecuencia de disparo a la que se le aplica el multiplicador de frecuencia de tiro, se permite disparar.

La implementación parcial de Shoot simplemente cambia el valor del tiempo anterior de tiro. Por lo tanto, esta implementación debe ser completada por las clases hijas.

En el caso de ThrowingAxeHand se completa el método Shoot mediante la instanciación del proyectil de hacha y la llamada a la corrutina que oculta el hacha hasta que se pueda volver a disparar. El proyectil instanciado es diferente dependiendo de si el modo de arma explosiva está activado o no: si está activado, se usa el prefab de las hachas explosivas que poseen el script ExplosiveAxeProjectile; y si está desactivado, se instancia el prefab de hachas arrojadizas que usan el script ThrowingAxeProjectile. Ambos scripts heredan de AxeProjectile, clase abstracta que únicamente implementa el método Update, que se dedica a animar el movimiento del hacha.

ThrowingAxeProjectile utiliza el collider que tiene para detectar si ha chocado contra un elemento que implemente la interfaz IDamageable cuando se detecta cualquier colisión. En caso afirmativo, se añade esta hacha como hija del objeto en el que ha impactado para así quedarse clavada, y llama al método DealDamage del script que implemente la interfaz.

Por otro lado, ExplosiveAxeProjectile sobrescribe el método Update, analizando en cada frame si el proyectil está a la distancia a la que se le ha indicado que debe detonar. En caso afirmativo, se llama al método Explode, que instancia el objeto que produce el efecto visual de la explosión, y llama al método Physics.OverlapSphere que permite obtener los colliders de los objetos que estén situados a un radio determinado desde este objeto. El método es llamado de tal forma que devuelve únicamente aquellos colliders cuyo GameObject esté en la capa “Targets”. Tras obtener los colliders, se recorren, obteniendo los GameObject que poseen esos colliders, y se intenta obtener cualquier script que implemente la interfaz IDamageable. En caso de que se encuentre, significa que ese GameObject puede ser dañado, por lo que se llama a su método DealDamage.

Volviendo a las armas disponibles, LaserWeapon implementa el método Shoot de forma que no se instancia ningún proyectil, sino que calcula los impactos directamente. Dependiendo de si el modo explosivo está activado, se calcula de una u otra forma:

- Si está desactivado, se hace un Raycast hacia donde se está apuntando, con una distancia de 100 unidades. Se obtiene la información del impacto del raycast y se obtiene el GameObject que ha impactado, del que se intenta obtener el script que implemente IDamageable. Si no existe, no sucede nada, y si existe, se llama a su método DealDamage. Finalmente, se anima el láser mostrándolo usando un LineRenderer, se haya impactado sobre algo o no.

- Si está activado, se calcula el punto situado hacia donde se está apuntando a una distancia determinada, que toma el mismo valor que la distancia entre las dianas y el jugador, mediante operaciones vectoriales. Tras haberlo calculado, se usa ese punto para instanciar el elemento que muestra el efecto de la explosión, y del mismo modo que en el proyectil de hacha explosiva, se utiliza Physics.OverlapSphere para obtener todos los GameObjects en la capa "Targets" con collider en el radio de explosión, aplicando el método DealDamage de IDamageable a todos los GameObjects que tengan el script que implemente esa interfaz.

Sistema de bonus

El sistema de bonus está gestionado desde el componente BonusManager. Éste contiene el método ExecuteRandomBonus, llamado cuando se destruye una diana de bonus, este método genera un número entero aleatorio entre 1 y 7, y se ejecutan diferentes tareas dependiendo del número obtenido:

- Número 1: mejora temporal de tiempo de recarga reducido. Primero intenta detener la corrutina de tiempo de recarga reducido en caso de que se esté ejecutando, y ejecuta una nueva instancia de esa corrutina, QuickReloadCoroutine. Esta se encarga de asignar un multiplicador de cadencia de tiro de 4 para que así se reduzca al 25% de su velocidad normal. La corrutina actualiza cada segundo el temporizador para la mejora en la interfaz, que dura 15 segundos, indicado por el atributo quickReloadDuration. Después, se reestablece el multiplicador a su valor normal: 1.
- Número 2: mejora temporal de obtención de puntos. Del mismo modo que en el anterior, se intenta detener su corrutina si ya existe y se ejecuta. La corrutina cambia el multiplicador de puntos de scoreManager a 2, mantiene el reloj hasta que se acaban los segundos para esta mejora, y se reestablece.
- Número 3: mejora puntual de puntos extra. Se añaden 15 puntos al marcador desde el método AddPoints del ScoreManager. Estos puntos son afectados por el multiplicador de puntos actual.
- Números 4 y 7: mejora puntual de tiempo extra. Se añaden 30 segundos al reloj. Toma los números 4 y 7 para que haya más probabilidad de que se aplique este bonus.
- Número 5: mejora temporal de arma explosiva. Se intenta detener su corrutina y se ejecuta, la cual consiste en marcar el arma como explosiva (propiedad Explosive de WeaponManager, booleana) a true, se espera el tiempo determinado y se reestablece a false.
- Número 6: mejora temporal de arma láser. Se detiene su corrutina si existe, y se lanza, la cual marca el arma desde WeaponManager como arma láser, asignando su propiedad UsingLaserWeapon a true. Tras trascurrir el tiempo asignado para esta mejora, se deshabilita la propiedad.

4.4.2 Elementos diseñados para mejorar la accesibilidad

En este apartado se van a tratar uno por uno los diferentes elementos que se han implementado exclusivamente para mejorar la accesibilidad del videojuego, explicando la razón de ser de cada uno y cómo se han construido.

Tratamiento y espacialización de audio

Se han usado herramientas de espacialización sonora, que nos permiten incrementar la calidad de la experiencia del usuario mediante la simulación 3D de audio dentro del juego, de forma mejorada respecto a la ofrecida por el motor Unity por defecto. Esto ayudará a aquellos jugadores que dependan sobre todo del sentido del oído para reconocer un entorno e interactuar con los elementos de éste.

Dentro de la espacialización se recurren a múltiples técnicas: el cálculo de la velocidad de propagación del sonido para así marcar una diferencia temporal entre lo que recibe cada uno de los oídos, el cálculo de la diferencia de intensidad y amplitud que hay entre lo oído por un oído u otro, el uso del efecto Doppler para simular el movimiento entre la fuente y el oyente de un sonido, y la reverberación, la cual modificará el sonido según el entorno en el que se esté emitiendo.

Estas técnicas de espacialización sonora sirven como medio para crear experiencias auditivas similares a lo que vivimos en la vida real, haciendo que los sonidos se oigan de un modo u otro en consecuencia a nuestra posición (dónde estamos) y hacia donde estamos mirando.

La herramienta seleccionada es la recomendada por Google VR, diseñada por Google: Resonance Audio. Resonance Audio es un SDK de audio espacial publicada el día 6 de noviembre de 2017, basado en la tecnología del Audio SDK de Google VR, el cual es compatible tanto en plataformas de escritorio como en plataformas móviles. Es compatible con varios motores y plataformas, ya sea Unity, Unreal, FMOD, Wwise, en la Web, en Android y en iOS. Ofrece características tales como:

- Personalización de la directividad de la fuente de sonido.
- Efectos de campo cercano.
- Propagación
- Reverberación basada en geometría
- Oclusión del sonido
- Grabación de archivos de audio para Ambisonic.

Este SDK permite proporcionar a los entornos VR una mejoría en la inmersión debido a la mejora de percepción auditiva. La decisión de usar este SDK para la mejora del audio en el videojuego no es por simple capricho, permitiendo una mejora considerable en la calidad del audio en lo que a la sensación de espacio se refiere, pues permite el uso de una cantidad mayor de fuentes de sonido 3D que el motor de audio estándar de Unity, ya que este está optimizado para usar los recursos de CPU de una forma más escueta mediante el uso de algoritmos de procesamiento de señales digitales basados en tecnologías de audio ambisónico.

El audio ambisónico no se debe confundir con audio surround, pues ésta es una tecnología isotrópica: esto significa que los sonidos de cualquier dirección se tratan de la misma forma, no como en el surround, en el que la mayoría de las veces se asume que las fuentes principales de sonido sólo se escuchan en los

canales frontales y traseros y las fuentes secundarias de sonido, como pueden ser sonidos ambientales o de efectos especiales, al resto de canales [54].
Mediante la aplicación de las tecnologías inspiradas en el audio ambisónico, se mejora considerablemente la calidad de la espacialización del audio, a pesar de no estar usando pistas de audio grabadas en formato Ambisonic.

Es importante asignar dentro de Unity el plugin Resonance Audio como el plugin de espacializado (Spatializer plugin) dentro de las opciones de audio del proyecto. Esto se asigna en el menú Edit -> Project Settings -> Audio en el editor de Unity, pudiendo así cambiar el valor del Spatializer plugin a Resonance Audio, como se ve en la figura.

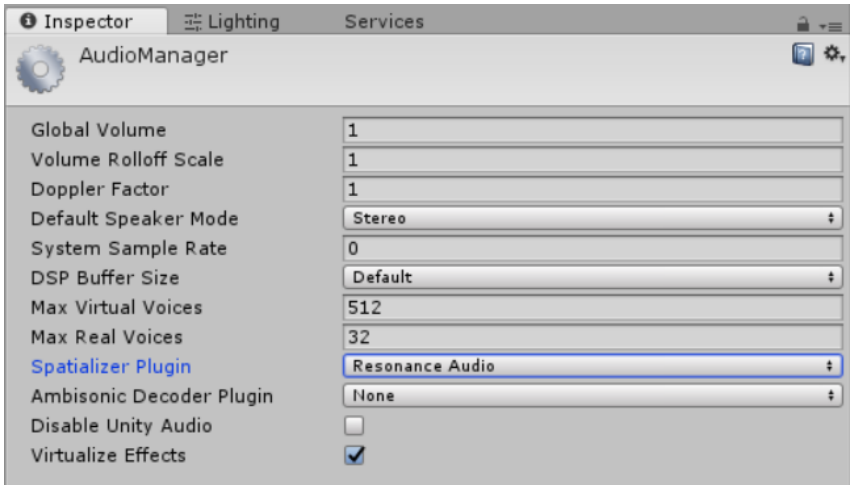


Fig. 23. Opciones de audio de la escena en el editor de Unity. Dentro de las múltiples acciones, se puede apreciar como el valor del desplegable de Spatializer Plugin es Resonance Audio, que es la configuración correcta si se quiere usar la espacialización de Resonance Audio y no la predeterminada de Unity (valor: None).

Para su uso en los componentes de la escena que emiten sonido, se debe añadir el componente Resonance Audio Source junto al Audio Source ya existente. Este permite configurar diferentes parámetros, tales como la ganancia, la directividad tanto del oyente como del emisor, que determina hacia dónde tiene que mirar el oyente para escuchar el sonido y hacia dónde se emite el sonido, respectivamente; y otras opciones sobre oclusión y efecto de campo cercano.

Síntesis de voz TTS (Text-to-speech)

Debido a que Unity no ofrece funcionalidades que sean compatibles con sistemas de accesibilidad basados en lectores de pantalla para personas invidentes, tales como Talkback en dispositivos Android, se ha de buscar una alternativa que nos permita sintetizar voz a partir de texto escrito, para indicar al jugador la puntuación actual o para navegar por las interfaces de la aplicación.

En la Unity Asset Store hay bastantes assets que implementan esta funcionalidad. La más avanzada es UAP, o UI Accessibility Plugin, [55] que permite hacer que toda la interfaz del videojuego sea accesible para usuarios con discapacidad visual, imitando la forma en la que funcionan sistemas como TalkBack en Android o VoiceOver en iOS; siempre que sean interfaces basadas en elementos 2D. Sin embargo, no se ha podido utilizar este asset debido a su coste de 75 dólares [56], por lo que se ha seguido buscando alternativas más económicas.

Centrándonos en herramientas de síntesis de voz, existe el asset RT-Voice [57], la cual ofrece una gran cantidad de opciones de text-to-speech, compatible en todas las plataformas de Unity: Escritorio, Móvil y Web, y soporta los lenguajes de marcado SSML (Speech Synthesis Markup Language [58]) y EmotionML (Emotion Markup Language [59]). Este asset tiene dos versiones, la estándar, que cuesta 35 dólares, y la pro, que contiene todo el código fuente en C# para poder realizar modificaciones.

De forma mucho más económica encontramos bastantes assets que usan las características nativas de iOS y Android para la síntesis de voz. Estos assets son Easy TTS [60], Android Speech TTS, que sólo funciona para Android, pero ofrece otras características como el reconocimiento de voz a texto (speech-to-text) [61], entre otras.

Para este desarrollo, debido a que la plataforma objetivo es Android, se usará un plugin open-source encontrado en GitHub (HoseinPorazar/Android-Native-TTS-plugin-for-Unity-3d) [62], que ofrece una simple implementación del TTS nativo de Android con la posibilidad de realizar cambios sobre la velocidad de la voz, tono de la voz, y localización.

Por desgracia, al usar el TTS nativo de Android no se puede virtualizar el audio de la voz emitida, sino que se escucha en el dispositivo sin más, cosa que no sucede en el asset RT-Voice, que es capaz de situar la voz en el escenario; aunque esto no afecta al desarrollo de este videojuego pues no hay intención de virtualizar la voz, ya que será usada para narrar elementos de la interfaz de usuario.

Durante el desarrollo, se ha observado que el plugin obtenido en GitHub para el uso de la tecnología TTS tiene una interfaz de acceso al sistema nativo algo capada, pues en el caso del locale, únicamente soporta las localizaciones en_UK (inglés de Reino Unido) y en_US (inglés de Estados Unidos). Por ello, se ha realizado una modificación del plugin a partir del código fuente del repositorio para que se pueda acceder a todas las localizaciones que proporciona Android, para poder así establecer que la voz sintetizada hable en español, que sería la localización es_ES.

Para ello, se ha realizado la modificación mediante Android Studio (versión 3.0.1). Se ha añadido una función nueva al archivo TTS.java, la cual recibe como parámetro el lenguaje y el país, que son los dos componentes necesarios para crear una nueva locale en Java.

```
public void SetLocale(String language, String country) {
```

```
        if(t1!=null)
            t1.setLanguage(new Locale(language,country));
    }
```

El atributo t1 es una instancia de android.speech.tts.TextToSpeech, que es la clase nativa del TTS de Android. Posteriormente, se ha ejecutado el script de build de Gradle proporcionado por el creador del plugin, que permite crear un archivo .aar en lugar de un archivo .apk: necesitamos el .aar ya que es el formato [63] que acepta Unity para plugins a ejecutar en Android. Este tipo de fichero es, en esencia, una biblioteca de Android compilada en un archivo de tipo Android Archive (AAR), que puede ser usada como dependencia para un módulo de app de Android. Ya que Unity en realidad está compilando una app de Android al compilarla para esa plataforma, se puede beneficiar de esta dependencia, la cual es solamente usada en la plataforma Android.

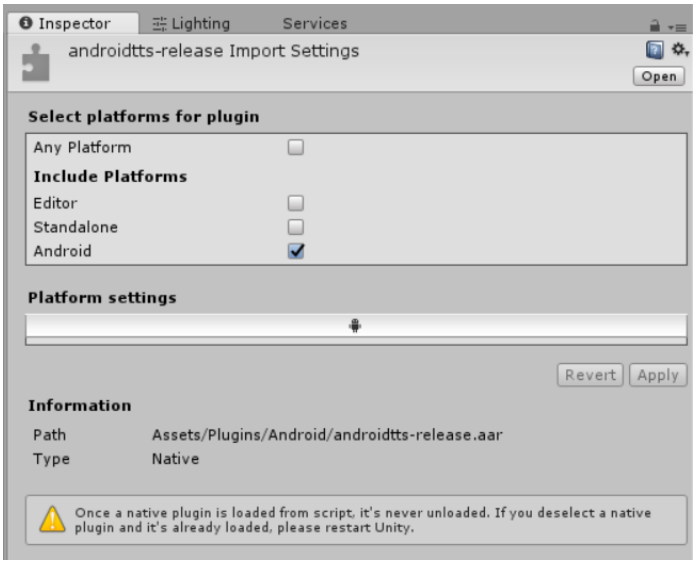


Fig. 24. Información mostrada en el inspector de Unity al pulsar sobre el plugin de la librería usada de TTS para Android y su configuración de importación. Se observa que permite indicar las plataformas que usarán este plugin, que en este caso sólo está marcada la opción de Android, y los ajustes específicos de la plataforma, que en este caso no hay ninguno.

Tras haber realizado este cambio, ahora se puede realizar la llamada a la nueva función dentro del script que carga el TTS en Unity:

```
TTSExample.Call("SetLocale","es","ES");
```

Donde TTSExample es un objeto de tipo AndroidJavaObject creado con el parámetro “ir.hoseinpo-razar.androidtts.TTS”, que indica la ruta de paquetes para acceder a la clase de la librería Android del plugin; el método Call es el método para realizar llamadas genéricas a funciones Java, “SetLocale” es el nombre de la función, y “es” “ES” son los argumentos pasados a esa función, donde “es” es el lenguaje y “ES” el país, lo que se traduciría a que el sistema de TTS de Android usará la localización de español de España.

Sistema de navegación de interfaz por foco

Se ha diseñado un sistema de interfaz dentro del juego para facilitar la navegación por menús a usuarios con visión reducida, que ofrece una alternativa al tener que mirar hacia un componente de la interfaz para interactuar con él.

Este sistema se ha diseñado para que se pueda navegar a través de cada elemento marcado como ‘interactivo’ dentro de un panel que contiene elementos de UI, ya sean etiquetas de texto o botones. La forma de navegar es similar al uso de la tecla tabulación dentro de un ordenador, pues se basa en la obtención de foco por parte de los elementos, foco el cual se puede ir desplazando según se pulsa la tecla especificada para realizar el cambio de foco al siguiente elemento.

En Cardboard, ya que se basa en plataforma móvil, se tiene en cuenta como acción para cambiar el foco el pulsar la pantalla. Esta pulsación de pantalla, si bien no se puede realizar directamente con el dedo sobre la pantalla, pues el dispositivo está montado en el visor VR, se realiza a través del botón que incluye los visores VR certificados por Cardboard. También se ha permitido que se puedan usar otras formas de input, como un mando o un ratón.

Si bien en un ordenador es directo el hecho de interactuar y desplazarse entre elementos, en este caso hay que tener en cuenta que solo existe un tipo de input para dispositivos móviles (en realidad hay más si se usan controladores externos como puede ser un mando con más de un botón, pero para mantener la compatibilidad y no obligar al usuario a disponer de otro dispositivo adicional, se asume que únicamente hay una forma de hacer input: los toques sobre la pantalla), así que también hay que diseñar una forma de, usando ese mismo input, se puedan realizar dos acciones: una para cambiar el foco, y otra para interactuar con el elemento que tiene el foco actualmente.

Para ello, se ha diseñado el sistema de tal forma que un toque breve en la pantalla, es decir, que entre la acción de pulsar la pantalla y despulsar la pantalla haya tomado un periodo inferior a un tiempo que determina si un toque ha sido largo, o un toque largo, en el que ese periodo es igual o superior al tiempo usado como referencia, realicen acciones diferentes. En este caso, el toque breve sirve para cambiar el foco, y el toque largo sirve para interactuar con el elemento.

Actualmente se ha soportado dos tipos de elementos: etiquetas y botones. Al cambiar el foco a una etiqueta, se lee usando la tecnología TTS el mensaje “Etiqueta” y el valor del texto que contiene esa etiqueta, y al cambiar el foco a un botón, se lee “Botón” y el valor del texto que contiene ese botón.

Al interactuar con un elemento de tipo etiqueta, se vuelve a leer el mensaje. Al interactuar con un elemento de tipo botón, se activa el botón, lanzando exactamente el mismo evento que se lanzaría si pulsamos el botón desde la interfaz con el puntero, por lo que se realiza la acción la cual ese botón tiene enlazada en el tratamiento del evento ‘OnClick’.

Como feedback visual, se ha implementado un sistema que resalte mediante un borde colorido el elemento sobre el que se tiene el foco. Esto ha facilitado las tareas de debugging para conocer sobre qué elemento se tiene el foco actualmente, además de servir como utilidad para usuarios que pueden utilizar la información visual y que desean usar el modo de navegación por foco por la interfaz.

Este modo de foco puede ser cambiado en el menú de opciones, ya sea para deshabilitarlo como para habilitarlo. Esta selección es persistente, es decir, se mantiene entre sesiones de juego incluso si cerramos la aplicación. Esto se ha realizado mediante el uso de la clase PlayerPrefs, que permite almacenar valores mediante un sistema de clave-valor gestionado por Unity, el cual almacena ese diccionario de datos dentro del dispositivo, ya sea ordenador o móvil, ofreciendo una capa de abstracción para la cual el programador no debe preocuparse dónde se han almacenado esos datos.

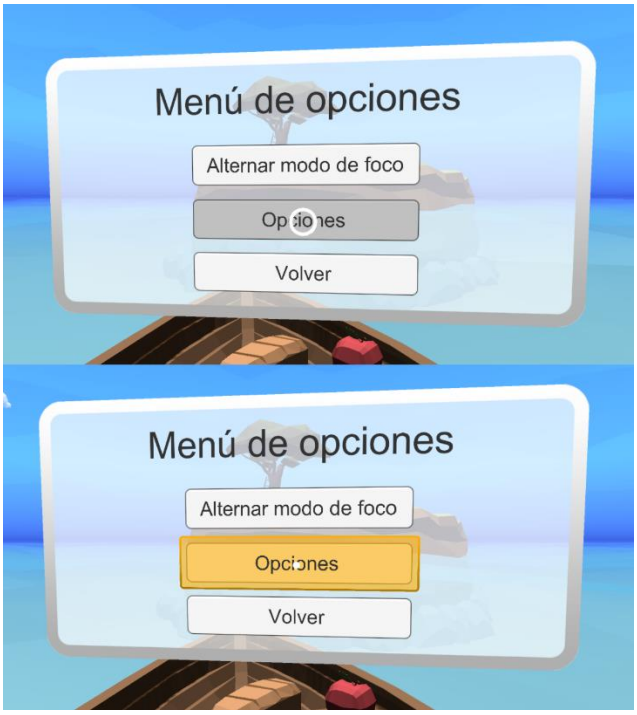


Fig. 25. Representación de la interacción con el mismo elemento de la interfaz, pero de dos formas diferentes. En la imagen de arriba, sin el modo de foco activado: se aprecia como la retícula de Cardboard se expande al apuntar hacia al botón 'Opciones' y se colorea el botón. En la imagen de abajo, con el modo de foco activado: la retícula de Cardboard no está expandida, pues no se puede interactuar con la interfaz más que con la tecla asignada para interactuar. Se ve cómo el botón 'Opciones' está rodeado por un borde naranja y coloreado de ese mismo color para indicar el foco.

Para evitar que el comportamiento por defecto de Cardboard interactúe con la interfaz mientras que el modo de foco está activado, se desactiva en el Canvas de la interfaz su componente GvrPointerGraphicRaycaster, para que así el raycast emitido por la retícula de Cardboard ignore por completo la interfaz.

Se han diseñado dos scripts por separado para el sistema de foco. El primero es PanelNavigationAccessibility, el cual debe ser añadido a los elementos de interfaz de tipo Panel, y el segundo es MenuManager, el cual debe ser añadido al Canvas de la interfaz.

PanelNavigationAccessibility se ocupa de desplazar el foco, interactuar con los elementos, obtener la información sobre los elementos, y dibujar la visualización gráfica del foco sobre los elementos.

Para obtener la información de los elementos, recibe un array de GameObject. Estos GameObject son los elementos con los que se quiere que se pueda interactuar dentro de este Panel cuando el modo de foco esté activado. Como ya se ha comentado, estos elementos son tanto etiquetas como botones:

- Las etiquetas son consideradas como todo GameObject que tenga como Tag el valor “Label” y tengan un GameObject hijo con el componente Text, que indica el texto de la etiqueta.
- Los botones son considerados como todo GameObject que tenga como Tag el valor “Button” y el componente Button, además de tener como hijo un GameObject con el componente Text, que indica el texto del botón.

Al iniciar el script del Panel, se recorren estos GameObject, obteniendo otros arrays como resultado:
- Textos a leer: array de String que indican el texto que se va a leer cuando se cambie el foco a cada elemento. Su valor para cada elemento se obtiene a partir del componente Text de cada elemento, añadiendo a ese valor el texto “Etiqueta:” o “Botón:” si es una etiqueta o si es un botón, respectivamente. En caso de que el GameObject tenga un componente de tipo AlternativeText, se sustituirá el texto a leer por el indicado por ese componente, sirviendo así como forma de leer un texto diferente al que se está mostrando por pantalla.
- Highlights (resaltados): array de GameObjects que representan el rectángulo de color que se representa cuando el elemento tiene el foco. Empiezan deshabilitados, y se habilita únicamente el del elemento que tiene el foco. Este elemento se añade como hijo a cada elemento con el que se puede interactuar, y se añade como primer hijo para que el texto aparezca delante del rectángulo y no detrás del rectángulo.

El componente PanelNavigationAccessibility tiene dos métodos importantes: ShiftIndex(bool forward) y InteractWithCurrent():

- ShiftIndex desplaza el índice del foco hacia delante, deshabilitado el rectángulo de highlight del elemento anteriormente seleccionado, habilitando el rectángulo de highlight del elemento nuevamente seleccionado, y leyendo por pantalla del valor del elemento actual. Es posible pasar como parámetro un booleano ‘forward’ que determina si el índice debe moverse hacia delante o hacia atrás (hacia delante si es true, hacia atrás si es false).
- InteractWithCurrent interactúa con el elemento actual. Primero determina qué tipo de elemento es, mirando el valor de su Tag, el cual puede ser “Label” o “Button”, como ya se ha indicado anteriormente. En caso de ser “Label”, lee el contenido del texto, y en caso de ser “Button”, invoca el manejador de evento del botón para el evento “onClick”.

En el momento que el componente PanelNavigationAccessibility de un Panel se desactiva, antes de desactivarlo se lanza el método OnDisable que deshabilita toda representación visual del foco.

El componente MenuManager se ocupa de determinar si el modo de foco está activado o desactivado, realizando las acciones pertinentes para cada estado; además de indicar qué elemento Panel debe ser mostrado en la interfaz, ofreciendo así una forma de navegar entre menús.

Al arrancar, intenta obtener el valor de si el modo de foco está activado o no desde las PlayerPrefs. En caso de no encontrar el valor, deshabilita el modo de foco por defecto.

Este componente mantiene como referencias todos los paneles de la interfaz por los que se quiere navegar, y sus scripts de PanelNavigationAccessibility. Al tener la referencia de los scripts de cada Panel, puede desactivarlos o activarlos según conveniencia: incluye el método SetFocusMode(bool value), el cual cambia el valor del modo de foco. En caso de cambiar a false, esto se actualiza en PlayerPrefs, y se deshabilitan todos los scripts PanelNavigationAccessibility y se activa el GvrPointerGraphicRaycaster, que permite el uso del puntero de Cardboard por defecto. En caso de ser true, los scripts se habilitan y se desactiva el Raycaster, pasando así al modo de foco, sobrescribiendo el comportamiento del puntero de Cardboard para la interfaz.

Sistema de navegación por barrido

Los sistemas de navegación por barrido permiten a personas con discapacidad motora severa utilizar interfaces de usuario mediante un sistema de foco que va desplazándose de forma automática hasta que el usuario interactúa con el sistema, que desencadena la activación del elemento sobre el que se está situando el foco en ese momento.

Existen dos tipos de modos de barrido: el barrido automático y el barrido dirigido. El barrido automático está asociado al control del barrido mediante un único pulsador, moviendo el indicador de barrido de forma secuencial y de forma automática. En el caso del barrido dirigido, se pueden usar varios pulsadores, que permiten cambiar la dirección del barrido y su velocidad [64]. En el caso de esta implementación, se ha optado por la solución de barrido automático ya que solo se dispone de un tipo de entrada.

A pesar de ser una navegación lenta, pues obliga a esperar para que el foco llegue al sitio deseado, puede significar una diferencia abismal en caso de que persona con estos perfiles utilicen el programa.

El indicador de barrido se mueve a través de los elementos de la pantalla, marcando con un relieve cada elemento de la pantalla y anunciando cada elemento a través de la salida de voz, mediante el sistema desarrollado de síntesis de voz. El usuario activa el pulsador, que en este caso es cualquier forma de input recibido al dispositivo (botón Cardboard, mando, ratón, etc.). A diferencia de la navegación por foco, se ha establecido que una pulsación de cualquier duración sirva para interactuar con el elemento marcado, en vez de requerir que se haga una pulsación larga.

La implementación se ha realizado a partir del sistema de la navegación por foco, por lo que se puede considerar un caso especial de ese sistema. La única diferencia entre el modo de navegación por foco y el modo de navegación por barrido es cómo se interactúa con el elemento marcado por el foco, y como se desplaza este foco.

Para ello, se ha modificado el script MenuManager, añadiendo un valor booleano que indica si el modo de barrido está activado, sweepModeEnabled; y un método que permite activar o desactivar el modo de barrido. A pesar de ser, a nivel de script, un subcaso del modo de foco, no pueden estar activados a la vez para el usuario. Por ello, en el momento que se activa el modo de foco, se desactiva el modo de barrido en caso de estar activado, y se habilitan los scripts de la navegación por foco. En el caso de activar el modo de navegación, se desactiva el modo de foco, pero se siguen manteniendo sus scripts activos, pero se les añade la funcionalidad de modo de navegación. En caso de no estar activo el modo de foco anteriormente, se activarán los scripts de PanelNavigationAccessibility de todos modos.

Dentro de PanelNavigationAccessibility se ha añadido un parámetro que indica si el modo de barrido está activado en ese panel. Cuando el modo de barrido está activado, se lanza una corrutina que se ocupa de llamar a ShiftIndex cada X segundos (indicados por una variable), que desplaza el foco hacia delante, leyendo el contenido del nuevo elemento sobre el que está el foco. Además, sobrescribe la forma de input del modo de foco: en el método Update, en lugar de hacer los cálculos para comprobar si se ha hecho una pulsación larga o no, determina que toda pulsación, sin importar la duración, llamará al método InteractWithCurrent, lo cual permite interactuar con el elemento que esté bajo el foco del barrido en ese momento con cualquier tipo de pulsación.

El valor persistente del modo de barrido se almacena bajo la clave “sweep_mode” en PlayerPrefs, que es leída por MenuManager al iniciar su script.

Retícula reactiva a elementos interactivos

Por defecto, Google VR permite mostrar un indicador especial en elementos de la interfaz de usuario que son interactivos, como es el caso de los botones. El indicador consiste en la retícula de la cámara, que indica en qué punto se está apuntando: en la mayoría de los casos, es un punto blanco sin más, mientras que cuando se apunta hacia un elemento interactivo, este punto se expande y se convierte en un círculo, indicativo que es un elemento interactivo.

Fig. 26. A la izquierda, visualización de la retícula de apuntado cuando no se está apuntando hacia un elemento interactivo. A la derecha, retícula expandida al estar apuntando hacia un elemento interactivo.

El objetivo en este caso es permitir marcar elementos como interactivos para que así la retícula cambie en consecuencia, facilitando la comprensión sobre qué elementos son interactivos y cuáles no. En este caso, se quiere marcar como interactivos las dianas del juego, para diferenciarlos del resto de elementos decorativos de la escena.

Para ello, se ha analizado el funcionamiento del sistema de eventos utilizado por Google VR, que se debe incorporar a la escena mediante el Prefab aportado por el SDK de Google VR: GvrEventSystem. Los EventSystem son los responsables de procesar y manejar eventos dentro de una escena de Unity. Por ello, una escena sólo debe de tener un EventSystem, para evitar un funcionamiento errático o un procesamiento de eventos repetido.

Para poder realizar una tarea en específico al emitir un evento, primero hay que provocar la emisión del evento en cuestión, y asignar un manejador.

La emisión de los eventos es trabajo de los Input Modules. Un Input Module es cualquier componente que herede de la clase BaseInputModule. En el arranque del Event System, se registran los Input Module, que son los encargados de lanzar los eventos que son detectados por aquellos GameObject que posean el componente EventTrigger.

El componente EventTrigger recibe los eventos emitidos, y llama a las funciones registradas para cada evento en ese GameObject. En el inspector de Unity, el componente Event Trigger permite indicar qué eventos se quieren capturar y qué instrucciones se van a tomar en la ejecución de cada uno de ellos.

El prefab anteriormente mencionado, GvrEventSystem, contiene un Input Module personalizado, diseñado especialmente para el input de Gvr: Gvr Pointer Input Module. El script GvrPointerInputModule implementa la clase BaseInputModule, permitiendo que se pueda interactuar con los elementos de la interfaz de usuario basadas en elementos Canvas y objetos 3D en una aplicación de Google VR. Este Input Module está diseñado para el uso ya sea con un puntero 3D con el Daydream Controller, o con un puntero basado en mirada, como en el caso de Cardboard, que es el interesante para este proyecto.

Para que sea funcional, hay que asegurarse que todos los Canvas tengan asignado como Render Mode el modo World Space, y que incluyan el componente GvrPointerGraphicRaycaster. Para que funcione con objetos 3D, que es lo que se busca en el caso de las dianas, la cámara principal debe de tener el componente GvrPointerPhysicsRaycaster.

Tras haber realizado estos ajustes, se pasa a añadir un EventTrigger a los elementos de las dianas que incluye el Collider principal y el modelo. Se añade como eventos a manejar el evento de Pointer Exit y Pointer Enter, sin necesidad de asignar ningún método a llamar cuando se manejen esos eventos.

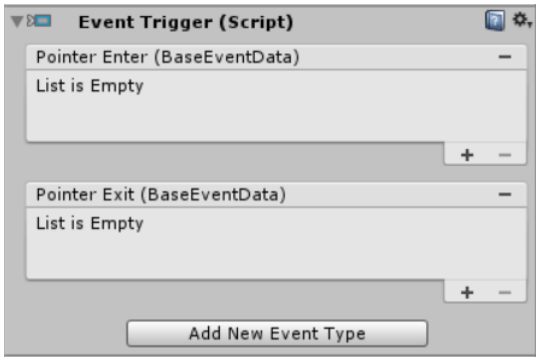


Fig. 27. Fragmento del inspector de Unity en el caso del elemento de las dianas que posee el collider principal. Se observa cómo se ha añadido las entradas para Pointer Enter y Pointer Exit, pero no se les ha asignado ninguna instrucción de llamada a métodos en cada caso.

La retícula de Google VR está programada para cambiar su aspecto cuando se apunta a un elemento, mediante raycasting, que contiene un Event Trigger que maneje los eventos de Pointer Enter y Pointer Exit, por lo que ya tienen el comportamiento deseado.

Sonido monoaural

Se ha trabajado sobre la mejora de la espacialización del audio, y sobre todo sobre su posibilidad de mejora del efecto estéreo que tiene el audio dentro de Unity. Sin embargo, para personas que no dispongan de una audición aceptable en ambos oídos, puede ser que haya información que se pierda debido a que está siendo emitida exclusivamente a través del canal de audio que llega al oído que menos capacidad auditiva presenta del usuario. Esto puede ser, por ejemplo, cuando en la escena del videojuego un objeto que se sitúa a la izquierda del personaje emite un sonido: este será emitido a través del canal estéreo que corresponda con el auricular izquierdo. Si el jugador no tiene capacidad auditiva por ese oído, es muy posible que no haya sido capaz de detectarlo, perdiéndose esa información que puede ser un sonido sin importancia, o una señal auditiva que indica la existencia de un objetivo, por ejemplo.

Por ello, se ha buscado un modo de forzar, de forma completamente opcional y ajustable por el usuario, la combinación de los datos de sonido de todos los canales de audio, reproduciéndolos por todos los canales; es decir, convertir lo que puede ser audio en estéreo o surround 5.1 o 7.1 en mono, es decir, un solo canal auditivo.

De este modo, aunque el usuario reciba, por ejemplo, información auditiva de algo que está situado a su izquierda a través de su oído derecho, junto a la información auditiva de algo que está situado a la derecha, es posible que no sea capaz de identificar de qué dirección proviene el sonido, pero sí es capaz de detectar que ese sonido ha sido emitido.

También cabe la posibilidad de que junto a esta solución se pueda mostrar a través de una señal visual la dirección desde la que ha venido un sonido, o sencillamente mostrar la dirección en la que se encuentra una entidad dentro de la escena que requiera la atención del usuario, pues es posible que para sonidos que son menos significativos, como pueden ser los cantos de pájaros que simplemente añaden inmersión en la escena, el usuario no requiera información de posición sobre ellos.

Unity no ofrece una forma directa de decidir si un emisor de audio (un GameObject con componente Audio Source) debe de emitir sonido espacializado pero por todos los canales del mismo modo: mientras que es posible indicar que una fuente de audio emite sonido 2D, el cual se emite al mismo volumen por todos los canales de audio, el sonido 2D perderá la espacialización a la que una persona con capacidad auditiva reducida puede seguir aportándole información: la distancia a la que se encuentra el objeto que emite el sonido, pues el sonido 2D en Unity va a emitir el sonido al mismo volumen sin importar la distancia respecto a la cámara, que es el GameObject que contiene el AudioListener.

Sin embargo, Unity ofrece una funcionalidad a partir de scripts que permite ajustar las opciones globales de audio del juego: la clase AudioSettings [65]. Dentro de AudioSettings, se puede acceder a un struct de tipo AudioConfiguration a través del método GetConfiguration, el cual puede ser modificado y aplicado al AudioSettings mediante el método Reset(AudioConfiguration config). AudioConfiguration [66] es un enum que contiene varias propiedades para el sistema del audio:

- dpsBufferSize, que permite cambiar el tamaño del buffer de audio para cambiar la latencia de sonidos.
- numRealVoices, que indica el número máximo de sonidos que se pueden oír concurrentemente en el juego.
- numVirtualVoices, el número máximo de voces que se pueden emitir concurrentemente en el juego.
- sampleRate, que indica la frecuencia de muestreo del dispositivo de audio.
- speakerMode, que indica el modo de altavoces usado por el dispositivo de output de audio.

La propiedad speakerMode es la que permitirá ajustar el número de canales usados en el videojuego. Es una variable de enum AudioSpeakerMode la cual puede tomar los valores Raw, Mono, Stereo, Quad, Surround, Mode5point1, Mode7point1 y Prologic. En este caso, es interesante poder asignar su valor al modo Mono.

Es posible cambiar directamente los parámetros de la configuración de audio directamente al asignar valores a los campos de AudioSettings, como puede ser AudioSettings.speakerMode. Sin embargo, esto se ha marcado como deprecated y será eliminado en futuras versiones de Unity, por lo que no es recomendable.

Cabe mencionar que los atributos de las opciones de audio pueden ser ajustados en el menú Edit -> Project Settings -> Audio dentro del editor de Unity, como se ha explicado en la sección anterior sobre espacialización de audio. Sin embargo, es interesante comprobar cómo se puede modificar mediante un

script en tiempo de ejecución para que sea elección del usuario si se va a usar el modo Mono o el estándar, que para dispositivos móviles es Stereo.

Al asignar este valor a Mono, se ha podido comprobar que el audio efectivamente se oye igual por todos los canales de audio. Además, permite el uso de audio espacializado sin tener que hacer ningún ajuste en especial: no existe panning, es decir, fluctuación del volumen de un sonido dependiendo de la posición respecto si está a la derecha o izquierda del personaje, pues el modo es Mono, pero sí que existe atenuación del audio provocado por la distancia que existe entre la fuente de sonido y el receptor de sonido, por lo que cumple con la funcionalidad deseada para soportar el sonido monoaural.

Por ello, se ha añadido una opción en el menú de opciones del juego que permita cambiar el valor del canal auditivo: Mono o Estéreo. El valor se asigna desde el componente MonoStereoSwitcher, que manipula la configuración de audio cambiando el speakerMode mediante los métodos indicados anteriormente. Esta configuración es persistente mediante el almacenamiento de la propiedad "stereo_mode" en PlayerPrefs, que indica si el modo estéreo debe estar activo si toma el valor 1 o el mono si es 0.

Modo de alto contraste

Puede haber personas con una capacidad visual reducida que, a pesar de su bajo nivel de visión, quieran utilizar las señales visuales dentro de sus capacidades, para no depender exclusivamente de las señales auditivas. Para ello, y con el fin de que las señales visuales que proporciona el juego puedan ser más visibles, se ha desarrollado un modo de alto contraste el cual oscurezca los objetos que no son necesarios para jugar y que haga más claros aquellos elementos que estén involucrados en la jugabilidad y no formen parte exclusivamente del decorado.

Es decir, en una escena normal, los objetivos, los proyectiles del jugador y el arma del jugador sobresaldrán de la escena, mientras que los objetos de la escena no imprescindibles se oscurecerán.

Para no romper la estética del juego y permitir una inmersión mayor, en lugar de directamente pintar los objetivos, arma y proyectiles de un color fuerte al que no le afecten las sombras y que el fondo y escenario sean negros sin más, se ha desarrollado un modo 'nocturno' que simule que es de noche en la escena. Los objetivos, al portar una luz, se verán mucho más brillantes que los elementos de la escena, dando el efecto de alto contraste.

Este modo de alto contraste también estará presente para elementos de la interfaz del juego, sobre todo en los menús. Debido a que en el menú de inicio el fondo es un escenario dinámico y colorido, se puede establecer que el modo de alto contraste oscurezca el fondo del menú, haciendo que así destaquen aún más las letras de la interfaz sobre el fondo, facilitando la lectura.

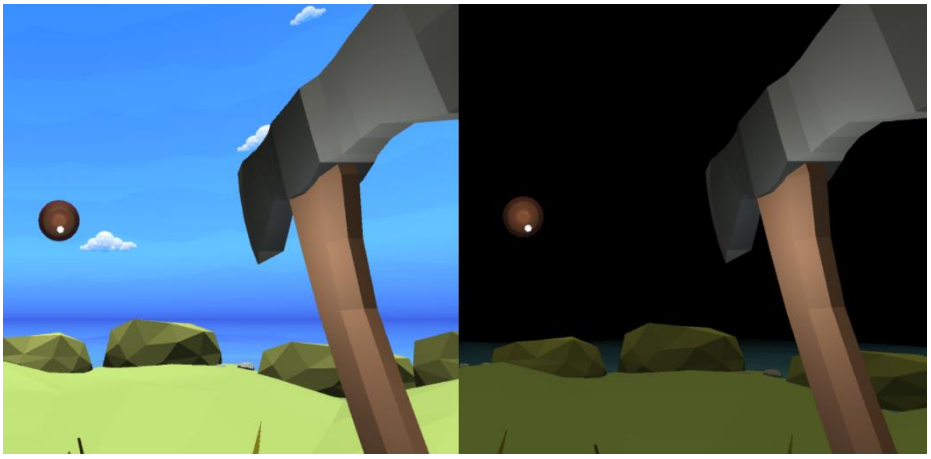


Fig. 28. A la izquierda, modo de juego sin haber activado el modo de alto contraste. A la derecha, modo de juego habiendo activado el modo de alto contraste: tanto el arma como el objetivo destacan más sobre el fondo, que es totalmente negro.

Se ha implementado mediante un GameObject llamado HighContrastModeManager, que tiene adjunto el componente HighContrastModeManager. Este componente contiene el valor booleano que indica si el modo de alto contraste está habilitado. Para cambiar ese valor, hay que llamar al método ChangeHighContrastMode, que recibe como parámetro el valor que se quiere asignar: true o false, siendo true para activar el modo de alto contraste, y false para desactivarlo.

El método, además de cambiar el valor booleano, hace las acciones pertinentes para realizar los cambios en la iluminación del escenario. Para ello, se han de realizar tres acciones principales: oscurecer el color de la niebla, que en el modo normal es azulada, oscurecer el color de la luz direccional de la escena, que en el modo normal es blanca, y modificar los parámetros del material de la Skybox para oscurecerlo.

Para cambiar el color de la niebla, se obtiene el campo de la clase estática RenderSettings, que contiene un conjunto de valores que determinan cómo se renderizan ciertas entidades en el juego, sobre todo relacionadas con la iluminación. Para ello, se modifica el campo RenderSettings.fogColor, asignándole el color deseado.

Para cambiar el color de la luz direccional, el script necesita una referencia al GameObject que contiene la luz direccional de la escena. Tras ser asignado, se puede cambiar su campo color modificar la luz. Al aplicar el color negro, la luz deja de ser emitida, por lo que se puede considerar equivalente a desactivar la luz.

Para cambiar los parámetros de la Skybox, se obtiene la referencia al Material de la Skybox a partir de RenderSettings.skybox. Los Material son instancias de un Shader, e incluyen métodos que permiten

asignar qué valores de input están siendo enviados al Shader en el renderizado de la Skybox. En este caso, hay que cambiar el parámetro de tipo float de nombre "_Exposure" por el valor 0 cuando se activa el modo de alto contraste. Cabe destacar que este parámetro es exclusivo de este Shader, pues un Shader personalizado obtenido a partir de un asset de la Asset Store. En caso de utilizar la Skybox por defecto de Unity, habría que ajustar otro parámetro.

Para asegurar la persistencia del valor del modo de alto contraste, se incluye a PlayerPrefs una nueva clave "high_contrast_mode" la cual se intentará cargar en el Start() del script, y se actualizará siempre que se llame al método ChangeHighContrastMode.

Para facilitar el cambio del valor en la interfaz, se ha creado un método ToggleHighContrastMode cuya funcionalidad consiste en llamar a ChangeHighContrastMode con el valor contrario del existente, permitiendo así alternar el modo según conveniencia.

Personalización de color de elementos principales

A pesar de la introducción de patrones en las texturas de las dianas, es posible que el usuario use el color para identificar los diferentes tipos, ya que es una forma más rápida que reconociendo los patrones utilizados. Sin embargo, esto puede ser un problema en el caso de usuarios con los diferentes perfiles del daltonismo: protanopia, deuteranopía, tritanopia o acromatopsia.

Por ello, se ha diseñado un sistema que permita personalizar el color de las dianas para que se ajusten a las preferencias de cada jugador, ya sea por cuestión de diferenciarlas correctamente o porque simplemente les guste más un color que otro.

Para poder cambiar el color de las dianas, se ha desarrollado un shader basado en el shader por defecto que recibe luz (*lit shader*) el cual permite asignar el color de parte de los modelos a los que se le asigna un material con este shader.

La asignación de los colores se realiza a través del análisis de la superficie (*surface* shader): para todos los píxeles de la superficie se comprueban las coordenadas UV correspondientes al modelo. En caso de que la coordenada U (horizontal) sea menor que 0.5, es decir, la primera mitad de la superficie, se colorea con el color primario. En caso contrario, se colorea con el color secundario.

Para que los colores encajen con los diferentes patrones diseñados para cada diana, se tienen que preparar los modelos de tal forma que en el UV *unwrap* las caras de la geometría del modelo que se quieren pintar del color primario estén al lado izquierdo y las que se quieren pintar del color secundario estén al lado derecho.

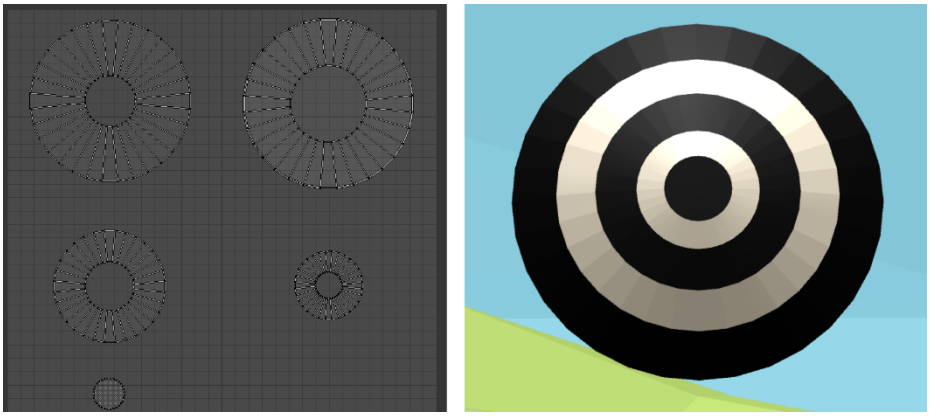


Fig. 29. A la izquierda, representación del *UV unwrap* del modelo de diana de 2 puntos, obtenido desde la interfaz de Blender. De izquierda a derecha y de arriba a abajo, se muestran los diferentes anillos que forman el patrón a dibujar sobre la diana. En este caso, el color entre anillo y anillo se alterna, visible en la imagen de la derecha.

El shader, llamado ReplaceColorShader recibe los siguientes campos como parámetros:

- _PrimaryColor: color primario a asignar en el modelo.
- _SecondaryColor: color secundario a asignar en el modelo.
- _MainTex: textura que debe tomar el modelo, aunque es reemplazada por los colores primarios y secundarios. Proporciona la información sobre UV. Facilita las tareas de diseño si se inserta una textura que esté formada por una franja vertical blanca que ocupe la mitad de la textura y una franja vertical negra que ocupe la otra mitad, pues así se muestran las zonas en las que va a afectar el color primario y el color secundario, respectivamente.
- _Glossiness: float que indica el valor de brillo que va a tener el modelo en el tratamiento de luz.
- _Metallic: float que indica cuánto de metálico va a ser el modelo en el tratamiento de luz.
- _Alpha: float que indica cuánto de transparente va a ser el modelo. Utilizado para desvanecer las dianas que no están activas.

Todos estos parámetros son ajustables desde el código, siempre que se tenga una referencia al material que está usando este shader. Teniendo esto en cuenta, se ha desarrollado el componente Colorizer, el cual se encarga de obtener el material del GameObject al que se asigna, y cambiar sus parámetros según el color deseado.

A pesar de que el shader permite cambiar ambos colores, se ha establecido que sólo sea intercambiable el color principal y no el secundario desde el script Colorizer, pero sigue el mismo procedimiento, por lo que se puede añadir en cualquier momento.

El componente Colorizer, al iniciarse, obtiene el valor del color primario que está almacenado en las PlayerPrefs. Debido a que PlayerPrefs sólo permite almacenar números enteros (int), números decimales (float) y valores de cadena (strings), y no estructuras más complejas, se ha codificado el color en forma de cadena en formato "RGBA(r,g,b,a)", siendo 'r', 'g', 'b', y 'a' valores de 0 a 1 que representan el valor de los canales rojo, verde, azul y alfa, respectivamente. Al leer este valor, se transforma al objeto Color de Unity, que puede ser aplicado sobre el parámetro _PrimaryColor del material, mediante el método Material.SetColor(key, value), siendo key "_PrimaryColor" y value el objeto Color en cuestión.

La personalización de colores se realiza desde el menú de opciones, la cual permite asignar los cuatro colores principales para los cuatro tipos de dianas. Esta configuración se realiza a través de la clase ColorizerManager.

ColorizerManager contiene la paleta de colores entre la que se van a poder elegir los colores para aplicarlos en cada diana. Además, contiene como atributos cuatro instancias de la clase ColorInfo, que contiene la clave bajo la cual se guarda el color para la diana que representa en PlayerPrefs, el color que tiene, y el índice dentro de la paleta de colores que se corresponde con ese color.

En el menú de opciones se puede rotar por los distintos colores de la paleta, aumentando el índice del ColorInfo de la diana en cuestión para así pasar al siguiente color. Siempre que se cambia el color de alguna de las dianas se actualiza su valor de color y su valor de índice en PlayerPrefs. Esto se realiza obteniendo el valor de la clave en su ColorInfo, que puede tomar por ejemplo el valor "target_minus_1_point_primary_color", y almacenando en esa clave el color codificado en forma de texto y en la clave de índice el índice el cual representa ese color en la paleta.

El formato del nombre de las claves para almacenar el color sigue el siguiente: "target_X_primary_color", pudiendo ser X "minus_1_point", "1_point", "2_points" o "bonus", para las dianas de menos un punto, un punto, dos puntos, o bonus, respectivamente. En el caso de la clave para almacenar el índice de la paleta para esa diana, se añade "_index" al final de la clave de color.

Se ha inicializado la paleta con 10 colores, asignados desde el inspector de Unity. El motivo por el cual se han elegido 10 colores es ofrecer un número razonable de alternativas diferentes que además de favorecer a la personalización, permitan en el caso de usuarios daltónicos una amplia selección para reducir el esfuerzo para diferenciar las diferentes dianas según el tipo de daltonismo presente ayudado además por la diferenciación de patrones de cada tipo de dianas.

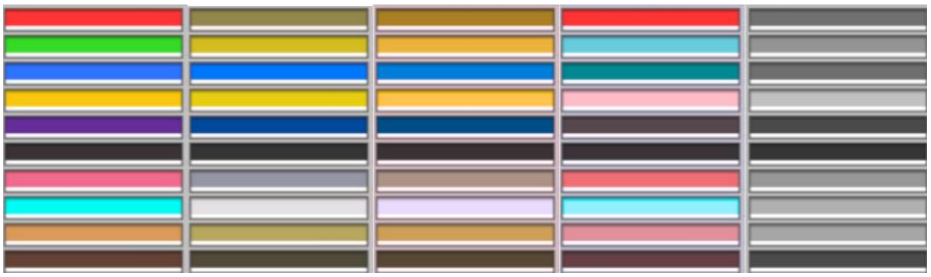


Fig. 30. Paleta de diez colores escogida para la personalización de dianas. De izquierda a derecha: paleta original, paleta con filtro de protanopia simulada, paleta con filtro de deuteranopia simulada, paleta con filtro de tritanopia, paleta con filtro de acromatopsia simulada. Si bien es cierto que en algunos casos los colores son similares, la variedad permite en todos los casos escoger como mínimo 4 colores diferenciables.

Personalización del tamaño de la fuente

El tamaño pequeño de la fuente es una queja bastante común entre las personas con visión reducida, ya sea médica, como en el caso de la miopía, la hipermetropía, o la ceguera parcial; o situacional, como puede ser en el caso de dispositivos con pantallas pequeñas o pantallas que están alejadas del jugador a una distancia que no sea cómoda para leer fuentes de tamaño reducido [67].

La solución ideal para ello es permitir que el usuario escoja el tamaño de fuente que le sea más cómodo. En caso de que esto no sea posible, es preferible establecer una fuente razonablemente grande por defecto.

Hay que tener en cuenta que a pesar de que permitamos ajustar la fuente por parte del usuario, si la fuente por defecto es pequeña para el jugador, es posible que éste sea incapaz de entender dónde está situada la opción del ajuste de tamaño de fuente, lo que reafirma la utilidad de establecer como fuente por defecto una con un tamaño que sea grande y pueda leerse con claridad.

Esto no sólo afecta a los usuarios con visión reducida, también a los usuarios que padecen de dislexia. El hecho de usar fuentes pequeñas puede ocultar detalles en los caracteres de la fuente, haciendo que algunos caracteres se parezcan entre sí, dificultando así el hecho de poder leer un mensaje con claridad. Al poder manipular el tamaño de la fuente, se pueden observar mejor las diferencias entre los distintos caracteres de la fuente, minimizando así las posibles confusiones.

El sistema implementado para la personalización del tamaño de fuente se basa en un cambio porcentual relativo al tamaño de la fuente tal y como se ha diseñado, siendo este el tamaño base. A partir de este tamaño base, la fuente se puede incrementar o decrementar según se configure: se ha diseñado para que tome un valor mínimo de un 80% y un valor máximo de 150%, aumentando o decrementando un 10% cada vez que se ajuste el tamaño.

El sistema de escalado de fuente sigue un patrón observador con dos piezas principales: el componente Font Scale Manager, que toma el papel del observado, y el componente Font Scaler, que toma el papel de los observadores.

El componente Font Scale Manager debe estar presente en la escena en todo momento, dentro de un GameObject preferiblemente vacío e invariable, que tenga como etiqueta "FontScaleManager". Este dispone de los métodos IncrementScale e DecrementScale, que llaman al método ChangeScale que recibe como parámetro el cambio relativo porcentual que se quiere realizar. IncrementScale llama a ChangeScale con el parámetro stepScale, que es un atributo de la clase que indica el salto porcentual que debe realizarse, mientras que DecrementScale llama a ChangeScale con el parámetro -stepScale, para que el cambio sea negativo. ChangeScale obtiene la escala actual y le aplica el cambio, y llama al método SetScale, que recibe el valor absoluto porcentual que debe asignar y cambia la escala. Al ser la escala de la fuente el valor observado por los observadores, se les notificará.

Todos los valores mencionados son configurables desde el editor de Unity, tanto la escala mínima, la escala máxima, y el cambio que se debe realizar en cada llamada a los métodos que incrementan y decrementan la fuente. El valor de la escala es persistente, pues siempre que se modifica se almacena en PlayerPrefs bajo la clave "font_scale". Este valor siempre es cargado cuando se inicia el script FontScaleManager.

Debido a que Font Scale Manager representa la entidad observada en el patrón, incluye dos métodos: AddObserver, que recibe como parámetro un objeto de la clase FontScaler, que se encarga de añadir ese observador a la lista de observadores; y NotifyAll, que recorre la lista de observadores y llama a sus métodos Notify.

El componente Font Scaler debe añadirse a todo elemento que tenga el componente Text que se quiera permitir reescalar. Cuando inicia el script, se obtiene el tamaño de la fuente asignado en el diseño de la interfaz, almacenándolo, para así tomarlo como referencia siempre que se quiera realizar un cambio porcentual de tamaño al variar la escala. Otra de las tareas realizadas al iniciar el script consiste en obtener la referencia al Font Scale Manager de la escena, que se obtiene mediante la etiqueta "FontScaleManager". Una vez que se tiene la referencia, se llama al método de Font Scale Manager AddObserver, pasando el script actual como observador, para que así se le notifique de todo cambio en la escala. Siempre que se cambie la escala, este componente será notificado. En el método Notify, que es el llamado al notificar a este componente, se llama a ChangeTextSize, que es el encargado de obtener el valor de escala de Font Scale Manager, y aplicarlo sobre el tamaño fuente recogido al inicio del script. Para evitar usar fuentes de tamaños no enteros, el tamaño computado se redondea, para así evitar posibles impurezas visuales.

Fig. 31. Diferencia entre el valor de escalado 100% y el 150%. Se ha asegurado en todos los elementos de la interfaz de usuario que el tamaño máximo no exceda los límites del espacio asignado para elemento de la interfaz, de modo que no requiera ampliar el tamaño de la UI. Esto podría provocar problemas de comodidad debido a una interfaz demasiado grande demasiado cerca.

Para el caso de otro tipo de interfaces, las cuales están integradas en elementos del mundo, como en el caso del tablero que indica la puntuación actual y el tiempo restante, se ha diseñado otro componente para ajustar la escala de la interfaz. En lugar de cambiar el tamaño de la fuente, lo cual no sería lo apropiado ya que el tamaño de la fuente por defecto ya está ajustado a todo el tamaño del contenedor, el componente acerca o aleja el objeto en el mundo entre dos puntos, acercándolo si la escala es mayor, o alejándolo si es menor. Estos puntos son asignados desde el punto inicial donde se ha diseñado que se sitúe el tablero de forma original, hasta un punto más cercano al jugador, de forma que esté más cerca, pero teniendo en cuenta su posición para que no oculte otros elementos del juego. Este componente, denominado Score Holder Position Calculator, se debe añadir al GameObject que recoge tanto el modelo sobre el que se coloca la UI, como el canvas sobre el que se pinta la UI deseada. Su funcionamiento es sencillo, pues simplemente, al tener una referencia al Font Scale Manager de la escena, obtiene su valor de escala e interpola la posición entre los dos puntos indicados según ese valor mediante interpolación lineal.

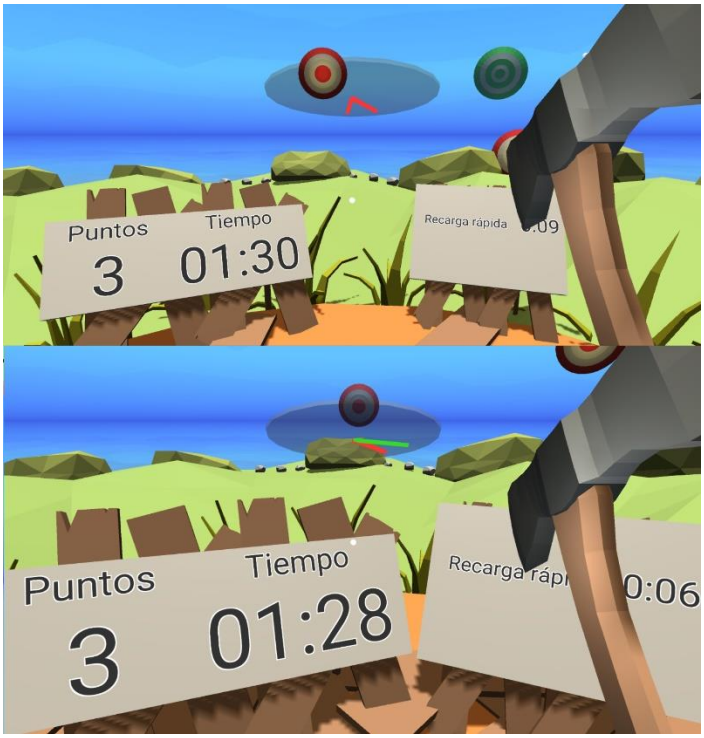


Fig. 32. Diferencia entre el modo de escalado a 100% y a 150%, aparente en la posición de indicadores de puntos y bonus en la escena. En el caso de una escala del 100%, los elementos están en el lugar que se diseñó. En el caso de 150%, están a aproximadamente 2 metros más cerca del jugador. El jugador puede seguir disparando a las dianas sin que los tableros bloqueen la visión.

Sistema de subtítulos

A pesar de que en este juego no haya diálogo más allá del sintetizado en el modo de navegación por foco, sí que se emiten algunos sonidos que son importantes para la jugabilidad, como aquellos que avisan de la aparición de dianas.

Por lo tanto, se ha diseñado un sistema de subtítulos, aplicados a tareas de *close-captioning* (descripción de sonidos), para así poder ofrecer una alternativa visual a la información auditiva de una forma descriptiva.

Se ha implementado el componente AudioController, el cual deberá ser aplicado sobre los elementos que reproduzcan sonidos y requieran mostrar una entrada en la interfaz en el espacio reservado a los subtítulos. Este componente toma el papel de un proxy para interactuar con AudioSource, pues en los GameObject que lo usen deberán aplicar los métodos Play y Stop de AudioController en lugar de los

nativos de AudioSource. En caso de no llamar a los métodos de AudioController, no se mostrarán subtítulos, pues se ocupan de hacer las tareas necesarias para instanciarlos además de reproducir el audio.

Por lo tanto, el componente AudioController tiene varios atributos necesarios para su funcionamiento: el componente AudioSource desde el que se quiere emitir el sonido, el texto a mostrar, la duración del texto, y la referencia al gestor de subtítulos, que es el componente SubtitleManager.

El AudioSource debe ser asignado desde el inspector para evitar ambigüedad al obtener la referencia desde el código, ya que puede ser que haya más de un AudioSource por cada objeto. Para casos en los que haya más de un AudioSource y se quieran mostrar subtítulos para todos, se deberá añadir un elemento AudioController por cada uno de ellos que tengan sus referencias.

El componente SubtitleManager se encarga de instanciar las líneas de subtítulos sobre el espacio reservado para ellas. Debe existir un GameObject vacío con la etiqueta "SubtitleManager" en la escena en la que se quieren utilizar subtítulos, además de un Canvas con un Panel reservado para el sistema de subtítulos.

El componente tiene una referencia al Prefab que se instanciará para cada una de las líneas, el cual consiste en un GameObject con el elemento Text de un determinado tamaño. Posee un atributo booleano que indica si los subtítulos están habilitados. Primero se obtiene su valor desde las PlayerPrefs, para saber si se han activado o no desde las opciones del juego.

Los componentes AudioController se ocupan de llamar a el método DisplaySubtitle del SubtitleManager, el cual recibe el texto del subtítulo y su duración. Siempre que estén activados los subtítulos, se instancia una línea de subtítulo mediante el Prefab mencionado, y se coloca dentro del Panel, cambiando el contenido de su texto al texto indicado en el parámetro. El tiempo indicado es utilizado para llamar a una corrutina que se encarga de eliminar esa línea realizando una animación: esta se encarga de esperar el tiempo indicado, y después realiza la animación para finalmente eliminarla.

Fig. 33. Ejemplo de una línea de subtítulo, colocada en la parte inferior de la pantalla. El tamaño establecido es el tamaño por defecto, el cual puede cambiar según las opciones de escalado de fuente. Se observa el efecto de bordeado negro alrededor del texto blanco.

El texto de los subtítulos se ha diseñado de tal modo que haya un gran contraste entre el texto y el fondo. Si bien muchos juegos sencillamente oscurecen un poco el fondo sobre subtítulos claros, puede

seguir habiendo problemas relacionados con el contraste cuando se muestran por encima de fondos complejos. Otra de las técnicas es el *letterboxing*, que consiste en dibujar una caja opaca o semi-opaca detrás de los subtítulos, definiendo así un fondo que siempre haga contraste con los subtítulos.

La solución aplicada en este desarrollo ha sido añadir un efecto de bordeado en el texto, de forma similar al que se ha aplicado en la interfaz del juego. En este caso, se ha aplicado un borde negro sobre letras blancas, con una anchura de trazo lo suficientemente grande como para aplicar un contraste suficiente para la legibilidad del texto.

En el diseño de los textos a mostrar por cada sonido, se ha procurado mantener el contenido lo más escueto pero descriptivo posible, para permitir que se pueda leer con rapidez. Además, se han vinculado los subtítulos con la funcionalidad de asignar tamaño de fuente, por lo que el tamaño de los subtítulos es personalizable según los gustos y necesidades del jugador [37].

Sistema de feedback auditivo

En conjunto con el sistema de espacialización de audio, el sistema de feedback auditivo permite mejorar la experiencia de los usuarios que dependan de las señales auditivas para interactuar con el entorno.

En el caso de este juego, se ha desarrollado un sistema que permite identificar los diferentes objetivos mediante un sonido diferente para que el jugador sea capaz de reconocer los diferentes tipos de dianas que aparecen en la escena, ya que cada uno reproduce un sonido diferente cuando aparece, cuando se le apunta y cuando se destruyen.

La reproducción de sonidos cuando aparece una diana y cuando se destruye es un sistema sencillo, el cual se basa en el reproductor de audio que está enlazado con el componente de la diana. Los sonidos son reproducidos desde la posición de la diana, y se ven afectados por los cálculos de la espacialización del audio 3D, por lo que se ven afectados por la posición de la cámara.

Sin embargo, para facilitar la tarea de apuntado para usuarios con visión reducida, se ha diseñado un sistema inspirado en la tecnología de los sonar: se reproduce un sonido no espacializado (audio 2D) al usuario cuando éste está apuntando a una diana. El volumen y el tono del sonido dependerá de la cercanía al centro de la diana, pero también empieza a sonar cuando se acerca a una diana, sin estar apuntando directamente a ella, por lo que ayuda al jugador a orientar su mirada hacia el punto donde el disparo dará en la diana.

El sonido reproducido es diferente dependiendo de cada diana. Este sonido es manejado por el componente SonarElement, componente asignado a los cuatro tipos de dianas. Este elemento se encarga de ofrecer métodos para reproducir, detener y cambiar volumen y tono del sonido de sonar. Posee referencias tanto al AudioSource que emitirá el sonido, y a AudioClip que determina qué sonido se reproduce.

El funcionamiento del sonar es posible gracias a un componente situado en la cámara que comprueba los elementos de sonar que se sitúan en el lugar donde el jugador está apuntando: el componente SonarRaycaster es el encargado de realizar un Raycast hacia delante para comprobar las intersecciones que ha tenido con los triggers especializados de sonar de las dianas. Para ello, primero se construye el raycast y se ejecuta, obteniendo todos los GameObjects que estén en la capa de colisiones Sonar que hayan intersecado con el rayo, que se dirige hacia delante a una distancia de 100 unidades.

Después, para cada uno de esos elementos, se obtienen dos puntos: el punto en el que se sitúa el centro del trigger, que se corresponde con la posición de ese GameObject, y el punto en el que ha impactado el raycast en cilindro del trigger. Mediante esos dos puntos, y la obtención del radio del trigger, que se obtiene tomando el valor de la escala local de ese GameObject, se calcula un valor relativo que irá de 1 a 0.25, siendo 1 en el caso de que el punto de colisión esté a una distancia inferior a 1 unidad respecto al punto central, hasta 0.25 en el caso de que el punto de colisión esté a una distancia respecto al centro igual al radio del trigger, es decir, en el borde. Este valor se calcula mediante la combinación de un valor interpolante mediante interpolación inversa entre 1 y el radio del trigger para el valor de la distancia entre los dos puntos, pasado a una función de interpolación lineal que interpola ese valor entre 1 y 0.25.

Tras haber obtenido este valor, se almacena junto al GameObject que contiene el trigger en una estructura de datos definida por la clase GameObjectAndDistance, y es añadida a una lista que almacena los GameObjects con los que se ha colisionado en este frame.

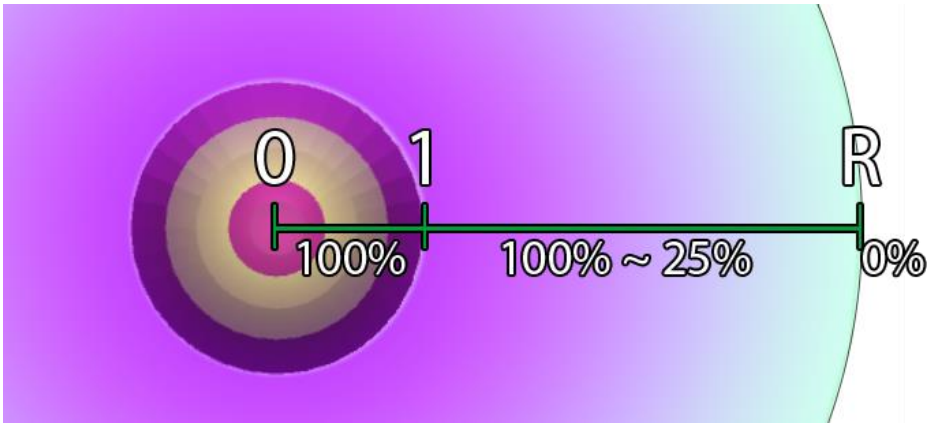


Fig. 34. Figura explicativa del cálculo del valor relativo dependiendo de la distancia del impacto. En la parte superior, se indican los valores de distancia, mientras que en la parte inferior se muestran los valores relativos en formato porcentual. Fuera del círculo, el valor es 0%, pues no está impactando sobre el trigger. En este caso, no habrá sonido. Al entrar al trigger, es decir, estando a una distancia R siendo R el radio desde el centro, se aplica el valor 25%. Según se va a acercando la distancia a 1, que es la distancia a la que está el borde del modelo de la diana, se incrementa el valor hasta 100% de forma gradual. A una distancia inferior a 1, el valor es siempre 100%.

Una vez almacenados en esa lista, se recorre, obteniendo el componente SonarElement de la diana en cuestión. Si este componente existe dentro de una lista que contiene los GameObjects con los que se ha colisionado en el frame anterior, se llama a ChangePitchAndVolume de SonarElement para cambiar su tono y su volumen según el valor relativo obtenido anteriormente respecto a la distancia de impacto, y se elimina de la lista de objetos colisionados en el anterior frame. En caso contrario, significa que este objeto no se había encontrado en el frame anterior, por lo que se reproduce con el valor relativo indicado.

Debido a que se eliminan los objetos de la lista de objetos colisionados en el frame anterior con los cuales se ha colisionado en este frame también, en esa lista sólo quedan aquellos objetos con los que se había colisionado anteriormente pero no en este frame. Por lo tanto, se recorre esa lista y se detiene el sonido del sonar para esos elementos.

Finalmente, se limpia la lista de objetos colisionados en el frame anterior y se rellena con los elementos colisionados en este frame.

Dado que los elementos de sonar de la diana no incluyen el componente SonarElement, estos tienen el componente NearbySonarElement que contienen la referencia SonarElement del padre, para que así sea accesible por SonarRaycaster.

Indicador visual de eventos que ocurren fuera del campo visual

Debido a la naturaleza de los juegos VR, que simulan un entorno completo y envolvente, se suelen tomar como única referencia de sucesos que están sucediendo a nuestro alrededor los sonidos que emiten: esto provoca una reacción por parte del jugador de mirar hacia el punto de emisión de ese sonido.

Sin embargo, para personas que no pueden depender del sentido del oído como medio de obtención de información, ya sea porque son personas sordas, con pérdida auditiva de diferentes grados, o sencillamente no puedan escuchar el sonido del juego por problemas técnicos, de entorno, o porque tengan el sonido desactivado, es imposible que reciban la información sobre ese elemento que está fuera del campo visual si no se utiliza más de un medio sensorial para notificarlo.

Si bien se ha tenido en cuenta el uso de notificación háptica para este tipo de eventos, se ha considerado que las notificaciones visuales son mucho más representativas ya que pueden ser más complejas y por lo tanto representar más información sobre lo que está sucediendo.

Por lo tanto, se ha desarrollado un indicador visual, presente en todo momento a lo largo del juego, que muestra la dirección de las dianas activas que hay en la escena, tomando la forma de una brújula con diferentes manillas. Estas manillas están apuntando en todo momento a las dianas que existen en la escena. En el momento que aparece una diana, se añade una manilla nueva a la brújula. En el caso de dianas que son desactivadas, ya sea por inactividad o porque han sido impactadas, la manilla correspondiente debe desaparecer de la brújula.

Para reconocer el tipo de diana que se muestra, cada manilla tiene el color correspondiente con el color que tiene la diana.

Fig. 35. Muestra de la brújula colocada en la parte superior. Se puede observar cómo se representan las dos dianas de 1 punto de color rojo en las flechas que apuntan hacia delante en la brújula. Fuera de la pantalla se puede ver cómo hay otras tres dianas: azul, verde y roja; situadas a la derecha del campo visual.

El sistema de brújula está implementado en las clases Compass y CompassArrow. El componente Compass está presente en la escena en el GameObject que incluye el modelo circular que representa a la brújula vacía. El GameObject tiene la etiqueta “Compass”, la cual es usada por las dianas de ambos tipos en el script TargetScript para tener la referencia a la brújula. Compass tiene dos métodos que serán llamados por las dianas en dos momentos diferentes: cuando aparecen, y cuando se desactivan.

El método llamado al activarse es AddTargetToBeTracked, que recibe como parámetro el GameObject desde el cual se está realizando la llamada. La función de este método es añadir la referencia al GameObject a una lista contenida en Compass, e instanciar un GameObject que haga de manilla para la brújula que esté vinculado con el seguimiento de la posición de ese objeto. En este método se configura

el GameObject que debe de seguir la manilla que se acaba de instanciar, y se añade como hijo en la jerarquía para que así el movimiento de la brújula sea junto a las manillas.

El método llamado al desactivarse es DeleteTarget, que recibe como parámetro el GameObject que se quiere desvincular de la brújula. Para ello, se elimina la referencia a este GameObject, además del GameObject que representaba su manilla en la brújula. En el caso de la manilla, se destruye directamente.

Las manillas ejecutan el script CompassArrow. Al iniciarse, obtienen el script Colorizer de su diana vinculada, que es la que contiene la información sobre su color primario: una vez obtenido, se modifica el atributo _Color del material de la manilla para cambiar su color consecuentemente. El apuntado de la flecha hacia la diana se realiza en su método Update, en el que se recalcula la rotación utilizando el método LookAt y bloqueando la rotación en los ejes de Euler X e Z para que únicamente rote en el eje Y local de la brújula.

Personalización de amplitud de campo de juego

A pesar de que 180 grados es un ángulo aceptable, también se ha tenido en consideración la posibilidad de cambiar la amplitud del arco para reducirla a un ángulo de 90 grados. Esto puede ser interesante para usuarios con limitaciones motoras más severas, relacionadas sobre todo con el movimiento de la cabeza. Al reducir la amplitud de la zona de juego, se reduce en consecuencia la amplitud de los movimientos de cabeza que debe de realizar el jugador.

Para ello, se ha modificado el script TargetSpawner para que en su método Start lea el valor de PlayerPrefs “reduced_game_area”, puede tomar el valor 0 en caso de tener que aplicar el área normal, y 1 en caso de tener que aplicar el área reducida. En caso de aplicar el área reducida, se asigna el ángulo de 90 grados. En caso contrario, se asigna el valor definido en el componente TargetSpawner desde el inspector, que es 180 grados. El valor de “reduced_game_area” se asigna desde el componente GameAreaManager, configurable desde el menú de opciones.

Reducción de la velocidad del juego

La velocidad del juego podrá ser ajustada: los problemas relacionados con la agilidad y la necesidad de tener una sincronía precisa con lo que está sucediendo en la pantalla pueden verse reducidos de una forma efectiva con esta opción, permitiendo que a lo mejor usuarios con movilidad reducida puedan tener una agilidad relativa a la velocidad del juego que les permita reaccionar con el suficiente margen de tiempo. Además, esto mejora la accesibilidad cognitiva ya que puede servir para dar más tiempo al jugador para procesar lo que está sucediendo en cada momento en el videojuego: las dificultades con el rápido procesamiento de acciones pueden verse mejoradas al dar más holgura en el tiempo de reacción.

Un ejemplo de esta característica puede verse en el videojuego *Celeste*, que ofrece un conjunto de opciones para reducir la dificultad del juego, entre las cuales se encuentra la posibilidad de reducir la velocidad del juego.

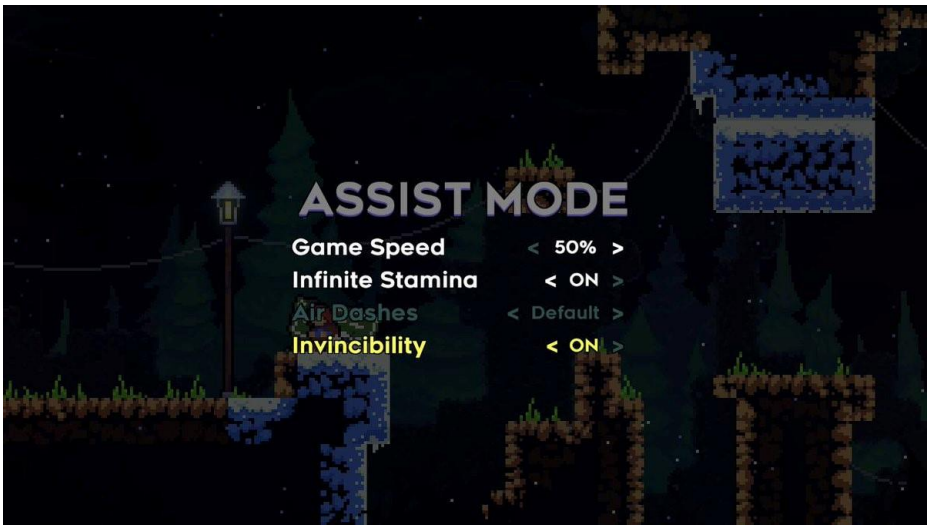


Fig. 36. Imagen del menú del juego *Celeste*, donde se permite modificar los parámetros para el *Assist Mode*. En él, se puede ajustar la velocidad del juego y facilitar las acciones de movimiento del personaje, como puede ser otorgándole invencibilidad o la posibilidad de correr de forma infinita.

La velocidad del juego determina cuánto de rápido va a avanzar el tiempo en el caso del tiempo restante de la ronda, la duración de las mejoras temporales, y la velocidad de recarga del arma. También afecta a la cadencia de aparición de objetivos, al tiempo que debe transcurrir hasta que desaparecen si no se les ha atacado, y su velocidad en caso de tener movimiento una vez hayan aparecido.

Para que esta configuración no sea simplemente un modo ralentizado del juego, la velocidad de los proyectiles sigue siendo la misma. Si los proyectiles fuesen a la misma velocidad que el resto de elementos ralentizados, el modo de velocidad de juego reducida carecería de sentido pues requeriría una velocidad de reacción equivalente a la velocidad de reacción necesaria en el modo de velocidad estándar, impidiendo así que jugadores que hayan seleccionado fehacientemente este modo disfruten de la experiencia de juego.

Para la implementación se ha desarrollado el componente GameSpeedModifier, que contiene como atributos dos números float: uno determina la velocidad de juego normal, y otro la velocidad del juego en el modo reducido. Estos números toman el valor de 1.0 en el caso de la velocidad normal, y 0.5 en el caso de la velocidad reducida. El valor 1.0 representa una escala temporal del 100%, mientras que 0.5

representa una escala temporal del 50%, por lo que en el juego cada segundo durará dos segundos en la vida real.

Cuando se pone en marcha el componente, realizan dos tareas en su método Awake. Primero, se obtiene el booleano que determina si el modo de velocidad reducida está activada, consultando las PlayerPrefs con la clave "reduced_speed_mode". Cabe decir que este valor de PlayerPrefs se asigna desde el componente GameSpeedManager, configurable desde el menú de opciones. Después, se asigna el valor de la escala apropiado al atributo Time.timeScale.

El atributo timeScale de la clase Time determina la escala a la que pasa el tiempo, afectando todas las variables de medida temporal como pueden ser Time.time o Time.deltaTime, que son utilizadas dentro del motor además de dentro de los scripts desarrollados en el proyecto.

Existen varias funcionalidades las cuales no se desea que se vean afectadas por el cambio de escala temporal. Por lo tanto, se han realizado modificaciones en algunas secciones del código para que no se vean afectadas características como el tiempo de animación de la expansión y contracción de la retícula de apuntado, y el movimiento de los proyectiles.

El primer enfoque era obtener la escala del tiempo en todas estas porciones de código y multiplicar los valores referidos al tiempo, como puede ser los segundos de animación en una corrutina. Por ejemplo, en el caso de que una animación dure 4 segundos y se aplique una escala de tiempo de 0.5, al multiplicar 4 entre 0.5 se obtiene 2 segundos. Cuando Unity aplique la escala de tiempo sobre ese tiempo, volverá a ser 4 segundos.

Sin embargo, esta es una solución subóptima, pues obliga realizar más operaciones y en algunos casos, como en el uso de los tiempos delta (diferenciales de tiempo, la diferencia de tiempo entre el frame actual y el anterior en Update), da resultados algo erráticos. Por lo tanto, la solución tomada es hacer uso de las medidas de tiempo no escaladas: unscaledTime en vez de time, unscaledDeltaTime en lugar de deltaTime, y, en el caso de corrutinas, WaitForSecondsRealtime en vez de WaitForSeconds.

Esta solución ha sido aplicada al movimiento de las hachas en el script AxeProjectile, en el comportamiento de la retícula dentro del código de Google VR en GvrReticlePointer, y en PanelNavigationAccessibility para evitar que la escala de tiempo afecte al detector de pulsación larga.

Ajuste de duración de pulsado largo

En el modo de navegación por foco, se distinguen dos tipos de input: las pulsaciones cortas y las pulsaciones largas, las cuales se diferencian en la diferencia de tiempo entre el momento en el que se presiona la tecla o la pantalla, y el momento en el que se deja de presionar. Por defecto, el tiempo de una pulsación larga está establecido en 500 milisegundos.

Sin embargo, existen personas con perfiles relacionados a dificultades motoras que se pueden beneficiar de un tiempo de aceptación más largo en el caso de las pulsaciones largas, como puede ser en el caso de que realizar pulsaciones cortas sea un inconveniente para ellas y accidentalmente estas pulsaciones cortas sean recibidas como pulsaciones largas.

Para ello se podría haber alargado a 1000 milisegundos el tiempo para tomar una pulsación como larga. Sin embargo, esto podría ser también problemático para personas que no puedan mantener el input durante tiempos largos, por lo que fijarlo a este valor no es recomendado.

Ya que la mejor solución depende del jugador, se ha permitido que este valor sea personalizable entre los valores 500ms y 1000ms, siendo así el jugador el que decida qué duración de pulsación le conviene más. Para ello, se ha modificado el método Update de PanelNavigationAccessibility para que el límite de tiempo para que una pulsación se considere larga sea parametrizado, siendo este parámetro ajustable desde MenuManager. MenuManager, al cargar, carga el booleano de PlayerPrefs "longer_long_press_duration", que, en caso de ser true, marca que la pulsación sea de 1000ms y en caso de ser false, 500ms.

El valor de PlayerPrefs "longer_long_press_duration" se asigna desde el componente LongPressDurationManager, configurable desde el menú de opciones.

4.5 Conformidad del proyecto con las Game Accessibility Guidelines

Tras finalizar el diseño y la implementación del videojuego, se ha pasado a rellenar el checklist ofrecido por la página Game Accessibility Guidelines [22] para comprobar el nivel de conformidad del seguimiento de las pautas a lo largo del proyecto.

4.5.1 Pautas sobre accesibilidad motora

- Asegurarse de que los controles sean tan simples como sea posible.
  - Los controles se basan simplemente en pulsaciones del único botón disponible en Google Cardboard, por lo que se han mantenido sencillos.

- Asegurarse de que todas las áreas de la interfaz de usuario sean accesibles con el mismo método de input que la jugabilidad.
  - Sí, pues se usa tanto el movimiento de la cabeza como el botón de Cardboard tanto para la navegación como para el juego.

- Incluir una opción para ajustar la sensibilidad de controles.
  - Se puede personalizar la sensibilidad de la pulsación larga.

- Asegurarse de que los elementos y controles virtuales sean grandes y estén bien espaciados, sobre todo en pantallas pequeñas o en pantallas táctiles.
  - Los elementos de la interfaz son lo suficientemente amplios.

- Permitir más de un tipo de dispositivo de input.
  - Se puede utilizar un ratón, el botón de Cardboard (ya sea capacitivo o magnético), o cualquier controlador Bluetooth similar al mando de Xbox.

- Hacer que los elementos interactivos que requieren puntería (en el caso de opciones de menú controladas por toque/cursor) sean estacionarios.
  - En la interfaz de usuario no hay elementos que se muevan.

- Asegurarse que todas las acciones clave puedan ser llevadas a cabo por controles digitales (pulsaciones, teclas) sin requerir el uso de input más complejo (voz, gestos), o que este tipo de input complejo sea opcional y complementario.
  - No existe input complejo para realizar acciones. Se había intentado implementar reconocimiento de voz para llevar a cabo acciones, pero siempre de forma complementaria.

- Incluir una opción para ajustar la velocidad del juego.
  - Se ha implementado un sistema para ajustar la velocidad del juego, reduciéndola a la mitad.

- Evitar o aportar alternativas para acciones que requieran mantener pulsado botones.
  - En el caso de navegación por barrido, se sustituye la acción de hacer una pulsación larga para interactuar con el elemento actual por hacer una pulsación corta.

- Permitir tipos de cuerpo variados para VR.
  - Los controles sencillos de Cardboard y su sistema de sensores no tienen en cuenta las cuestiones relacionadas con los tipos de cuerpo ni condiciones físicas de los usuarios.

- Incluir un periodo de cadencia de pulsaciones de 0.5 segundos.
  - Se ha implementado en parte en el caso de la cadencia de tiro de las armas.

- Utilizar sistemas muy simples de input que sean compatibles con dispositivos de tecnología asistiva, como puede ser los switches.
  - Mediante el modo de navegación por barrido se ha implementado un sistema similar a la navegación por switches, pues solo se necesita como input el botón para interactuar con el elemento actual, ya que el movimiento entre elementos se realiza de forma automática.

4.5.2 Pautas sobre accesibilidad cognitiva

- Permitir que el juego pueda ser empezado sin tener que navegar a través de múltiples niveles de menús.

- Usar un tamaño de fuente por defecto que sea leíble con facilidad.

- Usar un lenguaje claro y simple.

- Usar un formateado de texto simple.

- Permitir que los usuarios se muevan a través de mensajes de texto (text prompts) a su ritmo.
  — Sí. En el caso del mensaje final de texto que aparece al acabar una ronda, que es el único caso de aviso de texto, no desaparece hasta que no se pulsa el botón.

- Evitar imágenes parpadeantes y patrones repetitivos.
  — Se ha evitado el uso de ese tipo de imágenes.

- Incluir tutoriales e instrucciones de juego.
  — Sí, se ha incluido un panel indicativo del funcionamiento del juego y sus objetivos.

- Dar clara indicación de qué elementos son interactivos.
  — La retícula cambia según se esté apuntando a un elemento interactivo o no interactivo.

- Dar una opción para ocultar movimiento detrás de objetos interactivos.
  — El modo de alto contraste oculta el cielo, que contiene elementos que se desplazan, como nubes.

- Asegurarse que los sonidos para elementos, objetos o eventos importantes sean distintos entre sí.
  — Cada elemento emite sonidos diferentes. En caso de que varios elementos usen el mismo sonido, se diferencian mediante el tono.

- Incluir una opción para ajustar la velocidad del juego.
  — Se ha implementado un sistema para ajustar la velocidad del juego, reduciéndola a la mitad.

- Resaltar palabras importantes.
  — Se ha usado texto enriquecido en algunos diálogos para remarcar en negrita la información importante, como en el caso del mensaje del final de ronda, en el que se marca la información sobre los puntos, tiempo, y si se ha superado el récord o no.

- Ofrecer voz para todos los textos, incluyendo menús.

  - El sistema de síntesis de voz puede leer todos los elementos.

- Permitir que las instrucciones se puedan ver de nuevo.
  - Las instrucciones de tutorial pueden verse en todo momento desde el menú principal.

- Ofrecer una opción para ocultar todos los elementos no interactivos.
  - El modo de alto contraste oculta muchos de los elementos al oscurecer la escena, excepto elementos importantes como la interfaz, dianas y armas, que están resaltados sobre el fondo oscuro.

4.5.3 Pautas sobre accesibilidad visual

- Asegurarse de que no haya información esencial que se transmita únicamente utilizando color.
  - En el caso de las dianas, se indica su tipo también por el patrón utilizado en su decoración; y en el caso del contador de puntos, a pesar de que al recibir puntos los puntos positivos se muestren en verde y los negativos en rojo, siempre vienen acompañados del símbolo + y – dependiendo de si son positivos o negativos.

- Si el juego usa *field of view*, se debe asignar un valor por defecto apropiado.
  - Sí. Este valor está controlado por Cardboard.

- Evitar desencadenantes de cinetosis en VR.
  - Sí. Se han evitado algunos desencadenantes como el movimiento del personaje. Se evitan ya que el personaje está estático en el centro del escenario, únicamente variando el punto al que está mirando. Además, no hay apenas movimientos de elementos en la escena que puedan provocar situaciones de mareo al perder el punto de referencia del suelo.

- Utilizar un tamaño de fuente por defecto fácil de leer.
  - Sí. Se ha utilizado un tamaño moderadamente grande para que no haya dificultades. Este tamaño puede ser configurado.

- Usar un formateado de texto simple.
  - No se ha recurrido a diseños complejos en el caso del formateado de texto.

- Ofrecer alto contraste entre el texto de la interfaz y el fondo.
  - En menús, se ha usado un fondo blanco bajo texto negro. En otros elementos de interfaz, se han aplicado bordes en los caracteres: borde blanco en caso de texto negro, y borde negro en caso de texto blanco.

- Asegurarse de que los elementos y controles virtuales sean grandes y estén bien espaciados, sobre todo en pantallas pequeñas o en pantallas táctiles.
  — Los elementos de la interfaz son lo suficientemente amplios.

- Uso de sonido envolvente.
  — Sí. Además del ofrecido por Unity, se ha mejorado por sistemas de espacialización de audio 3D ofrecidos por Google (Resonance Audio).

- Dar una opción para deshabilitar la animación de elementos del fondo.
  — El modo de alto contraste oculta el cielo, que contiene elementos que se desplazan, como nubes.

- Ofrecer soporte para sistemas de lectura de pantalla.
  — No, pues el soporte de sistemas nativos de lectura de pantalla no es posible en aplicaciones de Unity. Para ello, se ha diseñado un sistema de síntesis de voz usado junto al modo de navegación por foco.

- Permitir ajustar el contraste.
  — En parte, pues se ha desarrollado un sistema de alto contraste, pero no un modo para ajustarlo de forma gradual.

- Asegurarse que los sonidos para elementos, objetos o eventos importantes sean distintos entre sí.
  — Cada elemento emite sonidos diferentes. En caso de que varios elementos usen el mismo sonido, se diferencian mediante el tono.

- Dar clara indicación de qué elementos son interactivos.
  — La retícula cambia según se esté apuntando a un elemento interactivo o no interactivo.

- Evitar colocar información esencial temporal fuera del campo de visión del jugador.
  — Toda información temporal que esté fuera del campo de visión se notifica a través de la interfaz, como en el caso de subtítulos y la brújula de objetivos.

- Permitir que el tamaño de fuente sea ajustable.
  — Sí. El tamaño de la fuente puede ser ajustado desde un 80% hasta un 150% de su tamaño original.

- Ofrecer un mapa auditivo de sonar.
  — Sí, se ha implementado un sistema de sonar para localizar los objetivos.

- Ofrecer voz para todos los textos, incluyendo menús.
  — El sistema de síntesis de voz puede leer todos los elementos.

- Simular grabaciones binaurales.
  - Sí, son ofrecidas por el plugin de espacialización de audio Resonance Audio.

4.5.4 Pautas sobre accesibilidad auditiva

- Proporcionar subtítulos para todos los diálogos importantes.
  - No hay diálogos, pero todos los sonidos importantes se muestran en los subtítulos.

- Asegurarse de que no haya información esencial que únicamente se indique a través de sonidos.
  - Toda la información esencial que se reproduce por sonidos también se indica en la interfaz, como es el caso de la obtención de puntos o la aparición de dianas.

- Si hay subtítulos, mostrarlos de forma clara y sencilla de leer.
  - El tamaño de los subtítulos se puede ajustar y cumplen estándares de contraste.

- Ofrecer subtítulos para habla complementaria.
  - El sistema de subtítulos describe algunos sonidos no esenciales.

- Asegurarse de que se pueda saber si los subtítulos están habilitados sin tener que reproducir un sonido.
  - Sí, esta información es visible desde el menú de opciones.

- Permitir que la forma de los subtítulos sea modificable.
  - Se puede configurar el tamaño de fuente de los subtítulos.

- Asegurarse de que toda la información suplementaria importante, como puede ser la dirección en la que está sucediendo algo, representada por audio se muestre también de forma visual.
  - El componente de la brújula informa sobre la dirección en la que se encuentran las dianas que han aparecido.

- Permitir cambiar entre sonido estéreo y mono.
  - Sí, se ha añadido un sistema que permite intercambiar los modos de audio, disponibles desde el menú de opciones.

- Asegurarse de que los subtítulos sean lo suficiente breves y se muestren a una velocidad apropiada para el público del juego.
  - Sí. Se ha mantenido el tamaño del texto de los subtítulos de forma escueta y a una velocidad razonable.

4.5.5 Pautas sobre accesibilidad vocal

- Asegurarse que el input por voz no sea requerido, y en caso de incluirlo, sólo esté incluido de forma complementaria o haya alternativas.
  - No hay ninguna forma de input por voz.

4.5.6 Pautas sobre accesibilidad general

- Asegurarse de que todos los ajustes se almacenen y sean recordados.
  - Sí, todos los ajustes del menú de opciones son persistentes.

- Proporcionar un sistema de guardado automático.
  - Al terminar la ronda, se almacena el resultado.

- Permitir que la jugabilidad sea ajustable al exponer sus parámetros.
  - En parte, pues se puede ajustar la velocidad del juego y la amplitud del área de juego, que afecta en la generación de dianas y sus tiempos de aparición y desactivación, además de la duración de mejoras y de la ronda.

5 Coste del proyecto

En esta sección, se pretende calcular una cifra que represente el presupuesto total necesario para llevar a cabo este proyecto, mediante un cálculo de costes materiales y el coste del trabajo de la persona que desarrolla el proyecto.

5.1 Presupuesto de ejecución material

Se determina coste total de la ejecución material a la suma tanto del coste de los equipos como del coste por el tiempo de trabajo. Por lo tanto, se procede a calcular el coste de equipos y el coste por tiempo de trabajo.

5.1.1 Coste de equipos

A lo largo del proyecto, se ha utilizado un ordenador de gama media con conexión a Internet, un dispositivo móvil compatible con Cardboard, el cual es el BQ Aquaris X Pro, y un headset compatible con Google Cardboard. La estimación de la duración del proyecto es de 3 meses por lo que se asume un uso de 3 meses. Se considera que la vida útil de los productos adquiridos es de 3 años.

EQUIPO	PRECIO	VIDA ÚTIL	USO	TOTAL
Ordenador de gama media	650,00€	3 años	3 meses	54,16 €
Conexión a Internet	29,00€	-	3 meses	87 €
BQ Aquaris X Pro	320,00 €	3 años	3 meses	26,66 €
Headset compatible con Google Cardboard	20,00 €	3 años	3 meses	1,66 €

Coste total de equipos	**169,48 €**

5.1.2 Coste por tiempo de trabajo

En la realización de este proyecto, se ha estimado que las horas dedicadas han sido 483. Redondeando a una jornada de empleo de 8 horas/día, las horas dedicadas son 488 horas. Tras conocer esta estimación, se puede obtener el coste por tiempo de trabajo de un ingeniero informático en la realización de este proyecto, asumiendo que el salario es de 30 euros la hora:

FUNCIÓN	Nº HORAS	€/HORA	TOTAL
Ingeniería	488	30	14.640,00 €

Coste por tiempo de trabajo	**14.640,00 €**

5.1.3 Coste total de ejecución material

Como se ha mencionado anteriormente, es el resultado de la suma del coste de equipos y el coste por tiempo de trabajo.

CONCEPTO	COSTE
Coste de equipos	169,48 €
Coste por tiempo de trabajo	14.640,00 €
Coste de ejecución material	14.809,48 €

5.2 Gastos generales y beneficio industrial

Los gastos generales incluyen los gastos necesarios que cubren las instalaciones en las que se desempeña el trabajo, junto a otros gastos adicionales. Para su cálculo, se aplicará un cargo con una cantidad con valor del 20% del coste total de ejecución material. Así:

Gastos generales y beneficio industrial	2.961,89 €

5.3 Presupuesto de ejecución por contrata

El valor del presupuesto de ejecución por contrata es igual a la suma del coste por ejecución material y el valor de los gastos generales y beneficio industrial.

Presupuesto de ejecución por contrata	17.771,37 €

5.4 Importe total

Al coste total de ejecución material se le aplica un 21% del I.V.A., obteniendo así el importe total.

	VALOR
Presupuesto de ejecución por contrata	17.771,37 €
21% de I.V.A.	3.731,98 €
IMPORTE TOTAL	**21.503,35 €**

Por lo tanto, el importe total de proyecto asciende a la cantidad de **21.503,35 euros**.

6 Resumen, conclusiones y trabajos futuros

Tras la finalización de este proyecto, se ofrece un resumen de las tareas realizadas, las conclusiones alcanzadas, y los posibles trabajos futuros sobre este proyecto y sobre la misma línea de trabajo.

6.1 Resumen

Al comienzo del proyecto, se han investigado las diferentes librerías, frameworks y sistemas de realidad virtual disponibles para realizar aplicaciones móviles mediante el motor Unity, tomando la decisión de utilizar el GoogleVR, que proporciona soporte tanto para el sistema Daydream como Cardboard, siendo Cardboard el utilizado para este proyecto dada su amplia compatibilidad con una gran cantidad de dispositivos Android, por lo que tiene un mayor impacto en el mercado.

Se ha investigado sobre el concepto de la accesibilidad en aplicaciones y sistemas, y se ha buscado información sobre los distintos tipos de perfiles de usuarios con diversidad funcional, teniendo en cuenta diferentes tipos de discapacidad como es la discapacidad motora, sensorial y cognitiva; incluyendo dentro de la sensorial la visual y auditiva.

Tras haber examinado la información tanto para sistemas de realidad virtual como para accesibilidad, se han definido un conjunto de soluciones generales para las múltiples barreras de accesibilidad existentes en la realidad virtual. A lo largo del desarrollo del proyecto, se han aplicado los conocimientos obtenidos en estas secciones junto a las pautas de desarrollo de videojuegos accesibles ofrecida por la guía *Game Accessibility Guidelines*.

El desarrollo ha consistido en un videojuego sencillo de disparos, para el cual se han tomado decisiones de diseño en favor de la accesibilidad, demostrando que, si se tiene en cuenta la accesibilidad desde el primer momento en el desarrollo de un sistema, aplicación o videojuego, el esfuerzo es mucho menor que una vez ya desarrollado, alcanzando un mayor espectro de usuarios.

Se han explicado los diferentes sistemas de las mecánicas del juego, tanto su diseño como su implementación, y se ha entrado en profundidad en los diferentes mecanismos diseñados específicamente para mejorar la accesibilidad. Entre estos sistemas se incluye un sistema de síntesis de voz, modos de navegación de interfaces por foco y por barrido, personalización de colores, incorporación de un sistema de subtítulos, modo de alto contraste, sistema de feedback auditivo en formato sonar, personalización de algunos parámetros del juego como puede ser la velocidad, entre otros.

Para finalizar, se ha rellenado el checklist ofrecido por *Game Accessibility Guidelines* para comprobar qué pautas se han seguido en el desarrollo y se ha calculado el coste aproximado de la realización del proyecto.

6.2 Conclusiones

Una de las conclusiones más importantes consiste en la facilidad que tiene la mejora de la accesibilidad en algunos aspectos, pero que en la mayoría de desarrollos de videojuegos se ignoran por completo, perdiendo la oportunidad de ampliar el público que puede disfrutar del videojuego. Cierto número de estos aspectos no suponen demasiado esfuerzo, y en caso de no tenerlos en cuenta pueden suponer una barrera innecesaria a un grupo de personas.

Si bien es cierto que algunas cuestiones son fáciles de implementar, hay muchas otras que se hacen exponencialmente más complicadas según lo avanzado que está el proyecto en cuestiones de desarrollo. Por lo tanto, otra conclusión muy importante es la presencia de la accesibilidad desde la primera fase del proyecto, para así poder evitar algunas decisiones de diseño que van en contra de algunos principios de accesibilidad.

Además, en el momento que se diseñan las mecánicas del juego teniendo en mente la accesibilidad, no sólo se mejora la experiencia para que personas con diferentes perfiles de diversidad funcional puedan disfrutar del videojuego, sino que también mejora la calidad del producto al ofrecer en muchos de los casos alternativas para la forma de juego de las cuales todos los usuarios pueden beneficiarse.

En el desarrollo de aplicaciones de realidad virtual se pueden aplicar muchas de las prácticas para mejorar la accesibilidad en videojuegos, pues ambas comparten el hecho de estar creando un mundo virtual y una forma de interactuar con el medio en tiempo real.

Respecto a las herramientas utilizadas, Unity ha demostrado ser un motor de videojuegos y aplicaciones 3D bastante robusto y completo. Mediante la realización de este proyecto, se han podido explorar las opciones que ofrece sobre interacción entre entidades, estructura interna de clases, integración de plugins que permitan la comunicación con el sistema nativo sobre el cual se está ejecutando el programa, la creación de shaders personalizados, gestión de audio, incorporación de assets externos, y mucho más.

Las herramientas proporcionadas por Google para Google VR son bastante completas y flexibles, ya que todo el código sobre el que funcionan es visible y por lo tanto modificable para algunos casos especiales, como ha sucedido en este proyecto al tener que editar el código fuente de la retícula para no ser afectado por la escala de tiempo modificada al cambiar la velocidad de juego. Entre eso y la buena documentación y la creciente incorporación de dispositivos disponibles con Daydream, existe un aliciente robusto para seguir trabajando con esta plataforma.

6.3 Trabajos futuros

Como trabajos futuros a raíz de este proyecto, se pueden distinguir dos líneas de trabajo: la mejora de la jugabilidad y la mejora de la accesibilidad.

Como futuras tareas a realizar sobre la jugabilidad, se tienen en mente las siguientes:

- Interacción con el escenario mejorada, permitiendo cambios sobre el escenario según al nivel de dificultad al que se va alcanzando. Una característica factible para esto es simular el paso del día mediante cambios en la iluminación, incorporación de sonidos diferentes según el paso del tiempo para así también indicar la fase de dificultad en la que se encuentra el jugador, entre otras características.
- La introducción de nuevos bonus, como puede ser un bonus que ralentice el movimiento de las dianas, lo cual facilitaría bastante las fases más avanzadas, ya que en ellas el movimiento de las dianas es más amplio y rápido.
- La mejora de la precisión del arma para así poder introducir la mecánica que otorgue más puntos en las dianas dependiendo del punto impactado.
- Creación de escenarios nuevos para así disponer de modos diferentes de juego, como puede ser un escenario en el que el jugador esté situado sobre una plataforma que se desplaza. Para ello, habría que tener en cuenta las soluciones proporcionadas en el análisis de la accesibilidad aplicadas a realidad virtual en lo que respecta a la cinetosis y a los movimientos.
- Introducción de algún tipo de objetivo que ataque al jugador. Este ataque reduciría los puntos obtenidos

Respecto a las mejoras de accesibilidad, se incluyen las siguientes, dentro de todas las posibles:

- Por cuestiones de carga de trabajo, no se ha podido implementar la funcionalidad de análisis de voz para utilizar formas input del habla para interactuar con el juego. Este podría ser usado para navegar por los menús, ya sea diciendo las palabras “abajo”, “derecha”, “izquierda”, o “abajo” para navegar por los botones sin tener que apuntar con la cabeza e “interactuar” o “ok” para interactuar con el elemento marcado; o simplemente permitir decir “ok” para interactuar con los elementos a los que el jugador está apuntando con la cabeza. También sería práctico para disparar dentro del juego, pero habría que hacer estudios sobre el rendimiento de esta característica para evitar el retardo que puede provocarse entre la fase de reconocimiento de voz y el momento en el que se termina de reconocer, para así reducir el tiempo entre input y disparo.
- La mejora de la personalización de las opciones desarrolladas en este trabajo, como puede ser el cambio de colores y fuente de los subtítulos, la velocidad del modo de navegación por barrido, opciones sobre el volumen de los diferentes sonidos, entre otros parámetros.

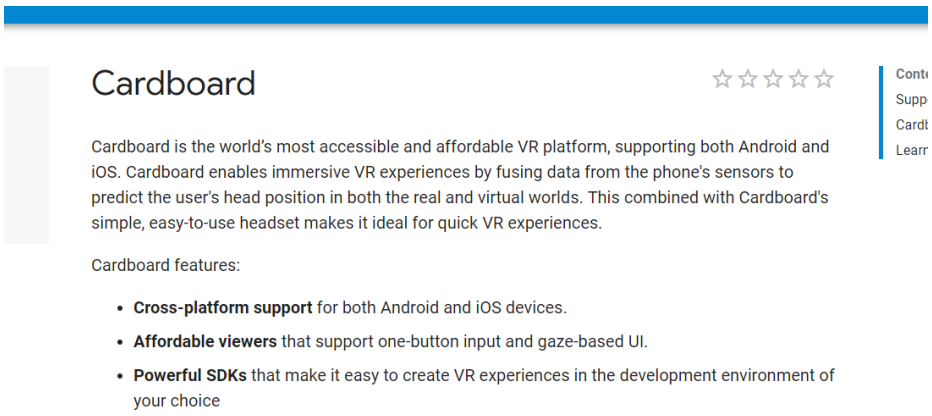
Sería muy interesante disponer de un dispositivo compatible con la tecnología Daydream para así introducir el puntero que ofrece como una nueva forma de input. Esto abriría un nuevo campo de estudio de la accesibilidad y de la jugabilidad al incluir diferentes formas de input simultáneas. Con el puntero, se podría implementar que las hachas sean lanzadas con el movimiento del brazo, ofreciendo además una alternativa de disparo mediante el botón del puntero para personas que no pueden realizar esos movimientos.

La inclusión de Daydream también serviría para acercarse más a las soluciones de realidad virtual no móviles, pues todas ellas hacen uso de punteros y otro tipo de sensores que detectan el movimiento de los

brazos, o incluso de todo el cuerpo. Mediante la aplicación de los mismos principios, sería posible aplicar conocimientos de esas plataformas a la plataforma móvil Daydream, y viceversa.

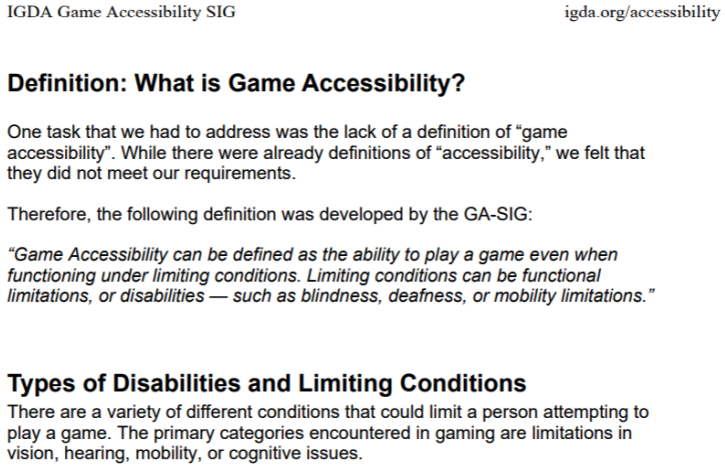
7 Bibliografía

1. Instituto Nacional de Estadística. 'Encuesta sobre discapacidades, autonomía personal y situaciones de dependencia'. [Online] Disponible: http://www.ine.es/dyngs/INEbase/es/operacion.htm?c=Estadistica_C&cid=1254736176782&menu=resultados&secc=1254736194716&idp=1254735573175
2. The Verge. 'The Ultimate VR Headset Buyer's Guide'. [Online] Disponible: https://www.theverge.com/a/best-vr-headset-oculus-rift-samsung-gear-htc-vive-virtual-reality
3. Virtual Reality Society. 'Virtual reality air force training' [Online] Disponible: https://www.vrs.org.uk/virtual-reality-military/air-force-training.html
4. Adam Popescu. 'These VR Systems Help Treat Veterans Recovering From PTSD' [Online] Disponible: https://www.bloomberg.com/news/articles/2017-03-16/these-vr-systems-help-treat-veterans-recovering-from-ptsd
5. Sixense. Página de presentación de aplicaciones VR de entrenamiento. [Online] Disponible: https://www.sixense.com/solutions/training/
6. HTC Vive. Página principal. [Online] Disponible: https://www.vive.com/eu/
7. Oculus VR. Página principal. [Online] Disponible: https://www.oculus.com/
8. PlayStation. Página principal de PlayStation VR. [Online] Disponible: https://www.playstation.com/es-es/explore/playstation-vr/
9. Samsung. Página principal de Gear VR. [Online] Disponible: http://www.samsung.com/global/galaxy/gear-vr/
10. Google Developers. 'Discover Cardboard' [Online] Disponible: https://developers.google.com/vr/discover/cardboard



11. Google Developers. 'Discover Daydream' [Online] Disponible: https://developers.google.com/vr/discover/daydream
12. Google VR. 'Teléfonos compatibles con Daydream' [Online] Disponible: https://vr.google.com/daydream/smartphonevr/phones/

13. IGDA (International Game Developers Association) ‘Accessibility in Games: Motivations and Approaches’. [Online] Disponible: http://g3ict.org/download/p/fileId_776/productId_50



IGDA Game Accessibility SIG igda.org/accessibility

Definition: What is Game Accessibility?

One task that we had to address was the lack of a definition of “game accessibility”. While there were already definitions of “accessibility,” we felt that they did not meet our requirements.

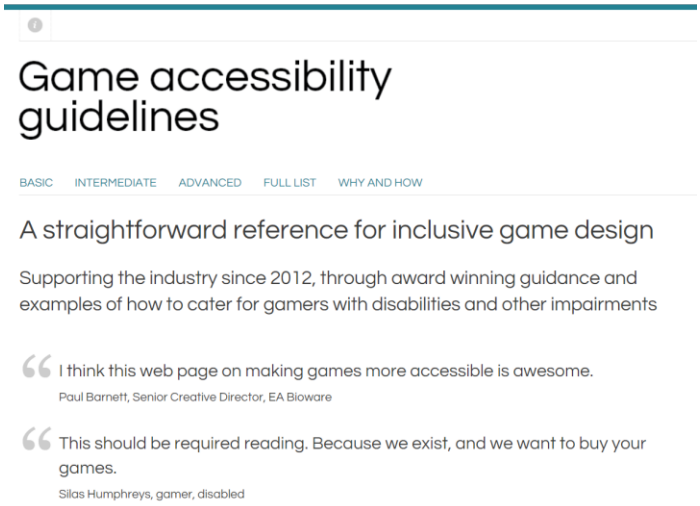
Therefore, the following definition was developed by the GA-SIG:

“Game Accessibility can be defined as the ability to play a game even when functioning under limiting conditions. Limiting conditions can be functional limitations, or disabilities — such as blindness, deafness, or mobility limitations.”

Types of Disabilities and Limiting Conditions

There are a variety of different conditions that could limit a person attempting to play a game. The primary categories encountered in gaming are limitations in vision, hearing, mobility, or cognitive issues.

14. Aguado Delgado, J.; Estrada Martínez, F.J. ‘Guía de accesibilidad de aplicaciones móviles’ [Online] Disponible: https://administracionelectronica.gob.es/pae_Home/pae_Estrategias/pae_Accesibilidad/pae_documentacion/pae_eInclusion_Accesibilidad_de_apps.html
15. Asociación D.O.C.E. ‘Qué es la ceguera legal’. [Online] Disponible: https://asociaciondoce.com/que-es-la-ceguera-legal
16. Wikipedia. Información sobre Test de Snellen. [Online] Disponible: https://es.wikipedia.org/wiki/Test_de_Snellen
17. Organización Mundial de la Salud. ‘Sordera y pérdida de la audición’. [Online] Disponible: http://www.who.int/mediacentre/factsheets/fs300/es/
18. Documentación de Unity. Información sobre el método VisionUtility.GetColorBlindSafePalette. [Online] Disponible: https://docs.unity3d.com/ScriptReference/Accessibility.VisionUtility.GetColorBlindSafePalette.html
19. Kirkpatrick, A.; O’Connor, J.; Cooper, M. 2018 Enero. Web Content Accessibility Guidelines (WCAG) 2.1 [Online] Disponible: https://www.w3.org/TR/WCAG21/
20. Microsoft. Microsoft Human Interface Guidance. [Online] Disponible: https://msdn.microsoft.com/en-us/library/windows/desktop/dn688964(v=vs.85).aspx
21. Apple. Accesibility Programming Guide. [Online] Disponible: https://developer.apple.com/library/content/documentation/Accessibility/Conceptual/AccessibilityMacOSX/index.html#//apple_ref/doc/uid/TP40001078
22. Game Accessibility Guidelines. Página principal de la guía. [Online] Disponible: http://gameaccessibilityguidelines.com/



23. Ian Hamilton, para Gamasutra. 'VR & Accessibility'. [Online] Disponible: https://www.gamasutra.com/blogs/IanHamilton/20161031/284491/VR__accessibility.php
24. PCAuthority. 'Sega and VR: ahead of its time'. [Online] Disponible: https://www.pcauthority.com.au/feature/sega-and-vr-ahead-of-its-time-417722#ixzz4CK8E4UUn
25. Patrick Klepek, para Kotaku. 'Resident Evil 7 Is A Nauseating Mess'. [Online] Disponible: https://kotaku.com/resident-evil-7-in-vr-is-a-nauseating-mess-1782288909
26. Charlie Hall, para Polygon. 'Sony to devs: If you drop below 60 fps in VR we will not certify your game'. [Online] Disponible: https://www.polygon.com/2016/3/17/11256142/sony-framerate-60fps-vr-certification
27. Jerrold Prothero. 'Vection measures', explicación del fenómeno de vección. [Online] Disponible: https://web.archive.org/web/20081121042049/http://www.hitl.washington.edu/publications/r-98-11/node17.html
28. YouTube. 'Tunneling VR Locomotion', muestra del efecto de tunneling en locomoción en RV. [Online] Disponible: https://www.youtube.com/watch?v=lKnM5gC-XpY
29. David Jagneaux, para UploadVR. 'Hands-On: Skyrim VR Without Teleportation Is Much More Immersive', controles de movimiento para el videojuego *The Elder Scrolls V: Skyrim VR*. [Online] Disponible: https://uploadvr.com/preview-skyrim-vr-without-teleportation/
30. Voices of VR Podcast. '#490: Making VR Experiences Wheelchair Accessible'. [Online] Disponible: http://voicesofvr.com/490-making-vr-experiences-wheelchair-accessible/
31. Pupil Labs. Página principal. [Online] Disponible: https://pupil-labs.com/vr-ar/
32. Pupil Labs en GitHub. Repositorio de librería para eye-tracking. [Online] Disponible: https://github.com/pupil-labs/hmd-eyes
33. Tobii Tech. Página de productos para RV. [Online] Disponible: https://www.tobii.com/tech/products/vr/
34. FOVE Inc. Página principal de FOVE. [Online] Disponible: https://www.getfove.com/
35. Independent Television Commision. 'ITC GUIDANCE NOTE FOR LICENSEES ON FLASHING IMAGES AND REGULAR PATTERNS IN TELEVISION'. [Online] Disponible: https://www.ofcom.org.uk/__data/assets/pdf_file/0021/16248/gn_flash.pdf

36. Game Accessibility Guidelines. 'Avoid flickering images and repetitive patterns'. [Online] Disponible: http://gameaccessibilityguidelines.com/avoid-flickering-images-and-repetitive-patterns/
37. Ian Hamilton, para Gamasutra. 'How to do subtitles well – basics and good practices'. [Online] Disponible: https://www.gamasutra.com/blogs/IanHamilton/20150715/248571/How_to_do_subtitles_well__basics_and_good_practices.php
38. Jon Fingas, para Engadget. 'Virtual reality game asks you to fight blind'. [Online] Disponible: https://www.engadget.com/2015/05/13/blind-swordsman/
39. Google Developers. Página principal de Resonance Audio. [Online] Disponible: https://developers.google.com/resonance-audio/develop/overview
40. Oculus. Página de descarga de paquetes de Audio. [Online] Disponible: https://developer.oculus.com/downloads/audio/
41. Adrienne Hunter, para Medium – VRInflux 'Vergence-accommodation conflict is a *****—here's how to design around it.'. [Online] Disponible: https://medium.com/vrinflux-dot-com/vergence-accommodation-conflict-is-a-bitch-here-s-how-to-design-around-it-87dab1a7d9ba
42. Unity. 'Unity Technologies Celebrates 5 Years of Unity'. Datos sobre la historia de Unity. [Online] Disponible: https://unity3d.com/es/http%3A//unity3d.com/company/public-relations/news/unity-5year-press
43. Unity. Características de la versión 5.0 de Unity. [Online] Disponible: https://unity3d.com/es/unity/whats-new/unity-5.0
44. Richard Fine, para Unity Blog. 'UnityScript's long ride off into the sunset'. [Online] Disponible: https://blogs.unity3d.com/es/2017/08/11/unityscripts-long-ride-off-into-the-sunset/
45. Wikipedia. Página para Mono, compilador de C# open-source. [Online] Disponible: https://en.wikipedia.org/wiki/Mono_(software)
46. Documentación de Unity. Documentación sobre las Tags en Unity. [Online] Disponible: https://docs.unity3d.com/560/Documentation/Manual/Tags.html
47. Documentación de Unity. Documentación sobre Layers en Unity. [Online] Disponible: https://docs.unity3d.com/560/Documentation/Manual/LayerBasedCollision.html
48. Documentación de Unity. Documentación sobre Raycasting. [Online] Disponible: https://docs.unity3d.com/ScriptReference/Physics.Raycast.html
49. Documentación de Unity. Documentación sobre el sistema de UI de Unity. [Online] Disponible: https://docs.unity3d.com/Manual/UIBasicLayout.html
50. Google en GitHub. Repositorio de GoogleVR. [Online] Disponible: https://github.com/googlevr/gvr-unity-sdk/releases
51. Google VR. 'Instant Preview'. [Online] Disponible: https://developers.google.com/vr/develop/unity/instant-preview
52. Unity. 'User Interfaces for VR'. [Online] Disponible: https://unity3d.com/es/learn/tutorials/topics/virtual-reality/user-interfaces-vr
53. Unity. 'Interaction in VR'. [Online] Disponible: https://unity3d.com/es/learn/tutorials/topics/virtual-reality/interaction-vr#anchor
54. Wikipedia. Comparación de audio ambisónico con otros formatos de sonido envolvente. [Online] Disponible: https://en.wikipedia.org/wiki/Ambisonics#Comparison_to_other_surround_formats
55. Unity Forums. '[RELEASED] UI Accessibility Plugin - UAP V1.0'. Información sobre el plugin UAP. [Online] Disponible: https://forum.unity.com/threads/released-ui-accessibility-plugin-uap-v1-0.469298/

56. Unity Asset Store. ‘UI Accessibility Plugin (UAP)’. [Online] Disponible: https://assetstore.unity.com/packages/tools/gui/ui-accessibility-plugin-uap-87935?aid=1011lNGT&utm_source=aff
57. Unity Asset Store. ‘RT-Voice’. [Online] Disponible: https://assetstore.unity.com/packages/tools/audio/rt-voice-48394
58. W3C. ‘Speech Synthesis Markup Language (SSML) Version 1.1’. [Online] Disponible: https://www.w3.org/TR/speech-synthesis/
59. W3C. ‘Emotion Markup Language (EmotionML) 1.0’. [Online] Disponible: https://www.w3.org/TR/emotionml/
60. Unity Asset Store. ‘Easy TTS (Text-to-Speech) for iOS and Android’. [Online] Disponible: https://assetstore.unity.com/packages/tools/input-management/easy-tts-text-to-speech-for-ios-and-android-22315
61. Unity Asset Store. ‘Android Speech TTS’. [Online] Disponible: https://assetstore.unity.com/packages/tools/integration/android-speech-tts-45168
62. GitHub. Repositorio de plugin para TTS Nativo en Android para Unity. [Online] Disponible: https://github.com/HoseinPorazar/Android-Native-TTS-plugin-for-Unity-3d
63. Android Developers. ‘Crear una biblioteca de Android’. [Online] Disponible: https://developer.android.com/studio/projects/android-library.html?hl=es-419#aar-contents

64. TecnoAccesible. ‘Método de acceso por barrido’. [Online] Disponible: https://tecnoaccesible.net/content/barrido
65. Documentación de Unity. Información sobre las opciones de audio AudioSettings desde scripting. [Online] Disponible: https://docs.unity3d.com/ScriptReference/AudioSettings.html
66. Documentación de Unity. Información sobre la clase AudioConfiguration. [Online] Disponible: https://docs.unity3d.com/ScriptReference/AudioConfiguration.html
67. Game Accessibility Guidelines. ‘Use an easily readable default font size’. [Online] Disponible: http://gameaccessibilityguidelines.com/use-an-easily-readable-default-font-size/

8 Anexos

Tabla 1. Interacciones entre diferentes colliders/triggers y cuándo ocurre la detección de colisión, la cual provoca la emisión de mensajes (llamadas a funciones OnCollision...) sobre la colisión.

	Static Collider	Rigidbody Collider	Kinematic Rigidbody Collider	Static Trigger Collider	Rigidbody Trigger Collider	Kinematic Rigidbody Trigger Collider
Static Collider		x				
Rigidbody Collider	x	x	x			
Kinematic Rigidbody Collider		x				
Static Trigger Collider						
Rigidbody Trigger Collider						
Kinematic Rigidbody Trigger Collider						

Tabla 2. Interacciones entre diferentes colliders/triggers y cuándo se emiten mensajes de trigger (OnTrigger...) al colisionar.

	Static Collider	Rigidbody Collider	Kinematic Rigidbody Collider	Static Trigger Collider	Rigidbody Trigger Collider	Kinematic Rigidbody Trigger Collider
Static Collider					x	x
Rigidbody Collider				x	x	x
Kinematic Rigidbody Collider				x	x	x
Static Trigger Collider		x	x		x	x
Rigidbody Trigger Collider	x	x	x	x	x	x
Kinematic Rigidbody Trigger Collider	x	x	x	x	x	x

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR