Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Utilización de una interfaz cerebro-computador en videojuegos 3D

Using Brain-Computer Interface in 3D videogames

Jaime Pablo Pérez Moro

La Laguna, 14 de julio de 2023

D. **José Ignacio Estévez Damas**, con N.I.F. 43.786.097-P profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

”*Utilización de una interfaz cerebro-computador en videojuegos 3D*”

ha sido realizada bajo su dirección por D. **Jaime Pablo Pérez Moro**, con N.I.F. 43.839.770-E.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 14 de julio de 2023

Agradecimientos

Muchísimas gracias a mi tutor José Ignacio Estévez Damas. Gracias por darme la oportunidad de trabajar en un proyecto tan interesante e innovador.

A mis padres y amigos, que han estado ahí siempre que los he necesitado y siempre me han animado a seguir adelante.

Especialmente agradecer a mi pareja y compañera de grado, Anabel, quien ha sido mi apoyo incondicional estos años. Gracias por estar siempre a mi lado, por tus palabras y por creer en mí, nada de esto habría sido posible sin ti.

Licencia

Resumen

En este proyecto, se investiga el uso de interfaces cerebro-computador (BCI) en videojuegos 3D, específicamente en un videojuego en tercera persona. El objetivo principal es analizar el efecto del BCI en los videojuegos 3D y comparar su eficacia con los controles tradicionales. El proyecto consta de dos partes. En la primera etapa, se desarrolló una pequeña demostración que utiliza el BCI, junto con una versión de la demostración que emplea controles tradicionales. En la segunda etapa del proyecto, se llevaron a cabo pruebas con usuarios reales en ambas versiones de la demostración, con el objetivo de recopilar datos para determinar si el uso del BCI en videojuegos 3D proporciona una ventaja real.

Palabras clave: interfaces cerebro-computador (BCI), videojuegos 3D, tercera persona, eficacia, controles tradicionales, integración, industria de los videojuegos, demostración, usuarios reales, pruebas, datos, ventaja real.

Abstract

In this project, the use of brain-computer interfaces (BCI) in 3D video games is being investigated, specifically in a third-person video game. The main objective is to analyze the effect of BCI on 3D video games and compare its effectiveness with traditional controls. The project consists of two parts. In the first stage, a small demonstration was developed that utilizes BCI, along with a version of the demonstration that employs traditional controls. In the second stage of the project, tests were conducted with real users in both versions of the demonstration, with the aim of collecting data to determine if the use of BCI in 3D video games provides a real advantage.

Keywords: Brain-computer interfaces (BCI), 3D video games, third person, effectiveness, traditional controls, integration, video game industry, demonstration, real users, testing, data, real advantage.

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Contexto y justificación

En el contexto actual, los videojuegos se han convertido en una forma de entretenimiento ampliamente popular y de gran relevancia cultural y económica. Gracias a los avances tecnológicos, los juegos son cada vez más sofisticados y realistas, con gráficos impresionantes y narrativas envolventes. Además, la disponibilidad de consolas, computadoras y dispositivos móviles ha ampliado su audiencia. Los videojuegos también se consideran una forma de expresión artística y han dado lugar a nuevas formas de juego en línea, como los eSports. La interactividad y la inmersión en los mundos virtuales han contribuido a aumentar su capacidad de ofrecer experiencias gratificantes para una gran variedad de jugadores con diferentes grados de madurez e intereses. En resumen, los videojuegos son una forma de entretenimiento altamente atractiva y accesible, con un impacto significativo en la cultura actual.

La investigación del uso de la interfaz cerebro-computadora (BCI) en los videojuegos es relevante debido a sus numerosos beneficios potenciales. La BCI puede mejorar la inmersión al eliminar la necesidad de dispositivos de entrada tradicionales, lo que permite una conexión más profunda con el mundo virtual. Además, esta tecnología puede mejorar la accesibilidad al permitir que personas con discapacidades físicas interactúen directamente con los juegos a través de señales cerebrales. También ofrece la posibilidad de experiencias de juego personalizadas y nuevas formas de interacción, adaptándose a las respuestas cerebrales de los jugadores. Por último, la investigación en este área puede contribuir a la comprensión de la neurociencia y los procesos cognitivos involucrados en la interacción con los videojuegos. En conjunto, estos beneficios hacen que la investigación sobre BCI en los videojuegos sea emocionante y prometedora para la industria.

En la actualidad, aunque se han realizado avances significativos en el desarrollo de la interfaz cerebro-computadora, son escasos los ejemplos concretos de videojuegos que la incorporen. Esto se debe principalmente a que nos encontramos en una fase de mejora tanto del hardware como del software necesario para hacerlo práctico y efectivo. Por lo tanto, es crucial llevar a cabo investigaciones en torno a cómo integrar de manera efectiva este tipo de interfaz en los juegos. Estos avances podrían permitir el desarrollo de videojuegos más inmersivos, personalizados y accesibles para una amplia gama de jugadores. En consecuencia, los objetivos de esta investigación se centran en explorar las posibilidades de integración de la interfaz cerebro-computadora en los videojuegos y en proponer estrategias y enfoques que faciliten su implementación.

1.2. Objetivos

El objetivo principal de esta investigación se divide en dos partes fundamentales. En primer lugar, se busca implementar el uso de un interfaz cerebro-computadora (BCI) en un videojuego 3D, explorando diferentes alternativas para su integración efectiva. Esto implica investigar y desarrollar métodos y técnicas que permitan capturar y utilizar las señales cerebrales de los jugadores de manera precisa y en tiempo real dentro del contexto del videojuego. Además, se evaluará el rendimiento técnico de la implementación, considerando aspectos como la precisión de las lecturas cerebrales, la latencia en la respuesta del sistema y la compatibilidad con los controles tradicionales del juego.

En segundo lugar, se busca evaluar el resultado de la implementación del BCI en el videojuego a través de la experiencia de los usuarios. Se recopilarán datos para medir la experiencia subjetiva de los jugadores al utilizar tanto el BCI como los controles tradicionales del juego. Se evaluará la comodidad y la facilidad de uso del BCI, así como la satisfacción general del usuario al interactuar con el juego mediante esta interfaz. Además, se recogerán otros aspectos relevantes relacionados con la experiencia de juego, como la inmersión, el disfrute y la sensación de control.

1.3. Estado del arte

Las interfaces cerebro-computadora (BCI) han experimentado una evolución notable a lo largo de los años, pasando de ser un concepto teórico a convertirse en una realidad aplicada en diversos campos, desde la medicina hasta el entretenimiento. Esta tecnología, que permite la comunicación directa entre el cerebro y los dispositivos electrónicos, ha mostrado un enorme potencial para mejorar la vida de las personas, en particular de aquellas con limitaciones físicas. En los últimos años, se han realizado numerosos avances en el campo de las BCI. Un área de investigación clave ha sido el desarrollo de técnicas cada vez más precisas para la clasificación y el reconocimiento de patrones cerebrales. Por ejemplo, se ha utilizado el análisis de correlación canónica (CCA) para clasificar los potenciales evocados visuales de estado estable (SSVEP), permitiendo a los usuarios controlar dispositivos electrónicos con una precisión cada vez mayor [1].

Una de las técnicas más prometedoras en el campo de las interfaces cerebro-computadora (BCI) es la utilización de los potenciales evocados visuales de estado estable (SSVEP). Tradicionalmente, los SSVEP se han utilizado en neurociencia para caracterizar los procesos visuales dinámicos a lo largo de la vía aferente. Sin embargo, los avances recientes han permitido su aplicación en el campo de la neuroingeniería, concretamente en el desarrollo de BCI basadas en SSVEP.

Un ejemplo de esto es el uso de SSVEP en videojuegos controlados por la atención del usuario. Un estudio reciente ha demostrado cómo se puede controlar un videojuego mediante la atención a estímulos móviles parpadeantes que generan respuestas SSVEP. En este videojuego, los avatares enemigos lanzan ataques en forma de anillos parpadeantes que se mueven, y el jugador tiene que desviarlos prestando atención a estos estímulos [6].

Los resultados del estudio revelaron que los estímulos parpadeantes móviles pueden ser detectados de manera robusta en pocos segundos mediante un BCI basado en SSVEP,

con un nivel de precisión del 84 %. Además, los participantes describieron el juego como divertido y desafiante, lo que sugiere que este tipo de BCI puede proporcionar una experiencia de usuario satisfactoria además de ser eficaces.

Otra aplicación prometedora de las BCI en el campo del entretenimiento es el juego "Zombie Jumper". Este juego, desarrollado con la plataforma OpenViBE y el motor Unity, se controla mediante dos comandos cerebrales: el pensamiento de moverse hacia adelante y el parpadeo. Para grabar los datos brutos del EEG, se utiliza una diadema Muse 2, y el protocolo de capa de transmisión en laboratorio (LSL) sirve como enlace entre la diadema Muse 2, OpenViBE y Unity. En una prueba realizada con 37 sujetos, el juego demostró una impresionante precisión media de clasificación del 98.74 %, y se observó que jugar durante períodos más largos de tiempo resultaba en un mayor control del juego [2].

Además, se ha demostrado que los videojuegos pueden tener un efecto positivo en la cognición y en la dinámica cerebral subyacente. Un estudio reciente examinó las dinámicas cerebrales en estado de reposo, antes y después de cuatro semanas de jugar a un videojuego con una interfaz de BCI. Los participantes jugaron un juego de intercambio de gemas durante aproximadamente 90 minutos al día, cinco días a la semana, durante cuatro semanas consecutivas. El estudio encontró cambios significativos en el flujo de información y las medidas de conectividad en las redes fronto-central, fronto-parietal, y centro-parietal como resultado de la interacción prolongada con la BCI. Estos resultados sugieren que las BCIs pueden facilitar cambios en las redes cerebrales en estado de reposo, lo que podría ayudar a desarrollar nuevas estrategias para mejorar la cognición. Sin embargo, también es importante considerar los posibles efectos de la fatiga mental y la interrupción de la sensación de compromiso del jugador que podría resultar de este esfuerzo [8].

En el ámbito de los programas de entrenamiento cerebral, las BCI han mostrado un potencial prometedor, especialmente en el ámbito de los videojuegos. Los videojuegos proporcionan un ambiente que puede ayudar a aumentar la inteligencia y el rendimiento cerebral de una manera agradable y sin carga mental. Un ejemplo de esto es un videojuego controlado por BCI y realidad virtual que permite a los jugadores de diferentes dispositivos y ubicaciones jugar simultáneamente en un servidor online basado en el Internet de los cerebros (IoB). Un servidor en la nube de alto rendimiento recoge los datos de electromiografía (EMG) y electroencefalografía (EEG) de todos los sitios de los jugadores. Los jugadores controlan la velocidad del coche concentrándose y realizando tareas de aritmética mental (MA). Este nivel de concentración puede aumentarse o disminuirse, lo que ayuda a controlar los movimientos del coche durante el juego [9].

Además de su aplicación en videojuegos, esta técnica también tiene potencial en otros campos, como la educación para necesidades especiales o el tratamiento de trastornos de la atención. Esto se debe a que los videojuegos controlados por la atención pueden ayudar a entrenar la capacidad de atención y de seguimiento visual a estímulos móviles objetivo.

A pesar de los prometedores avances en las BCI basadas en SSVEP, todavía existen retos por superar. Por ejemplo, la variabilidad individual en la generación de SSVEP y la necesidad de reducir el tiempo necesario para detectar la atención del usuario son aspectos que requieren una investigación adicional.

1.4. Metodología de investigación

1.4.1. Descripción de la muestra

La muestra estuvo compuesta por 15 participantes reclutados de manera aleatoria entre las personas que pasaban por el laboratorio, sin restricciones de edad o perfil específico. La heterogeneidad de la muestra permitió una amplia representación de potenciales usuarios de la interfaz cerebro-computadora (BCI) empleada en este estudio.

1.4.2. Instrumentos y materiales

Para este estudio, se utilizó el dispositivo BCI NextMind para capturar las señales cerebrales de los participantes. Las pruebas se realizaron con un juego desarrollado en Unity 3D, que permitía a los participantes interactuar con el entorno de juego a través de diversas mecánicas de juego que debían aprender durante un tutorial inicial. Se desarrollaron dos versiones del juego: una que combinaba la BCI con los controles tradicionales de teclado y ratón, y otra que solo requería el uso del teclado y el ratón. La comunicación entre el BCI y el juego se realizó mediante el SDK de NextMind.

1.4.3. Procedimiento experimental

El procedimiento experimental comenzó con una calibración inicial del dispositivo BCI que duraba aproximadamente 55 segundos. A continuación, los participantes jugaban la versión BCI del juego durante un máximo de 10 minutos. Después de un breve descanso, jugaban la versión de teclado y ratón también durante 10 minutos. En total, cada sesión experimental duraba entre 25 y 30 minutos, dependiendo de la cantidad de tiempo que llevaba ajustar correctamente el dispositivo BCI.

1.4.4. Variables y medidas

Se recogieron diversos datos de cada participante para el análisis, incluyendo características personales (nombre, longitud y tipo de pelo, edad, experiencia previa con BCI), condiciones de la prueba (calificación de la calibración, luminosidad del entorno), desempeño en la prueba (número de fallos, finalización exitosa de la prueba, tiempo requerido para completar cada versión del juego) y feedback subjetivo (puntuación de comodidad).

Capítulo 2

Técnicas y herramientas utilizadas

2.1. Interfaz cerebro-computador (BCI)

2.1.1. Definición y concepto

Las interfaces cerebro-computadora (BCI) son sistemas que conectan el cerebro humano a dispositivos computacionales, permitiendo la comunicación y control sin utilizar métodos tradicionales de interacción. Se utilizan en medicina para ayudar a personas con discapacidades físicas o neurológicas, y en la investigación científica para estudiar la actividad cerebral. Además, las BCI tienen aplicaciones en realidad virtual y videojuegos, brindando una forma innovadora de control basada en señales cerebrales.

2.1.2. Tipos de BCI y Comparación con BCI basados en EEG

Introducción a los Tipos de BCI

Las interfaces cerebro-computadora pueden ser categorizadas en varios tipos, dependiendo del método de captura de señales cerebrales. Algunos de los tipos más comunes incluyen los BCI basados en electroencefalografía (EEG), magnetoencefalografía (MEG), resonancia magnética funcional (fMRI), y resonancia magnética funcional en estado de reposo (rs-fMRI). Cada uno de estos tipos tiene su propio conjunto de ventajas y desventajas, y se adaptan mejor a ciertas aplicaciones que otros.

BCI basados en EEG

Los BCI basados en EEG capturan las señales cerebrales mediante electrodos colocados en el cuero cabelludo del usuario. Estos electrodos miden las fluctuaciones en la actividad eléctrica del cerebro, que luego se procesan e interpretan utilizando algoritmos de aprendizaje automático para deducir la intención del usuario. Los BCI basados en EEG son no invasivos, relativamente baratos, portátiles, y pueden proporcionar una respuesta en tiempo real, lo que los hace adecuados para una amplia gama de aplicaciones, desde el control de dispositivos en la vida cotidiana hasta aplicaciones en medicina y neurociencia.

BCI basados en MEG

Los BCI basados en MEG, por otro lado, capturan las señales cerebrales a través de la detección de los campos magnéticos producidos por la actividad eléctrica del cerebro.

Aunque la MEG puede proporcionar una resolución espacial y temporal superior a la del EEG, los sistemas de MEG son significativamente más caros y menos portátiles que los sistemas de EEG. Además, la MEG requiere un entorno de operación muy controlado para reducir el ruido de fondo, lo que limita su aplicabilidad en configuraciones fuera del laboratorio.

BCI basados en fMRI y rs-fMRI

Los BCI basados en fMRI y rs-fMRI miden la actividad cerebral a través de cambios en el flujo sanguíneo y el oxígeno en diferentes regiones del cerebro. Aunque estos métodos pueden proporcionar una alta resolución espacial y pueden capturar información sobre la actividad cerebral en todo el cerebro, son menos adecuados para aplicaciones en tiempo real debido a su baja resolución temporal. Además, los sistemas de fMRI son muy costosos y no portátiles, y la interpretación de las señales de fMRI puede ser más compleja que en los sistemas basados en EEG o MEG.

Comparación de los Tipos de BCI

En general, la elección del tipo de BCI a utilizar depende en gran medida de la aplicación específica y de los requisitos de resolución espacial y temporal, costo, portabilidad y facilidad de uso. Mientras que los BCI basados en EEG son la opción más versátil para la mayoría de las aplicaciones, los BCI basados en MEG, fMRI, y rs-fMRI pueden ser más adecuados para aplicaciones de investigación y clínicas que requieren una mayor resolución espacial o la capacidad de mapear la actividad cerebral a nivel del cerebro entero.

2.1.3. Diferentes tipos de BCI basados en EEG

Dentro de los BCI basados en EEG, existen varias subcategorias dependiendo de la naturaleza de las señales cerebrales utilizadas para la comunicación. Algunas de las más comunes incluyen las basadas en ritmos sensoriomotores, potenciales evocados (como P300 y SSVEP), y la imaginación motora.

BCI basadas en ritmos sensoriomotores

Los BCI basadas en ritmos sensoriomotores utilizan las modulaciones de la amplitud de las oscilaciones alfa y beta en el cuero cabelludo causadas por el movimiento o la imaginación del movimiento. Este tipo de BCI ha sido ampliamente utilizado para controlar prótesis robóticas y otros dispositivos externos. [7]

BCI basadas en potenciales evocados

Los BCI basadas en potenciales evocados utilizan las respuestas cerebrales específicas evocadas por estímulos sensoriales. Los potenciales evocados visuales, como el P300 y el SSVEP, son los más comúnmente utilizados en BCI debido a su robustez y facilidad de detección. Estos BCI han sido utilizados para una variedad de aplicaciones, desde la comunicación hasta el control de dispositivos. [4]

BCI basadas en imaginación motora

Los BCI basadas en imaginación motora utilizan la actividad cerebral asociada con la imaginación del movimiento. Estos BCI suelen requerir un entrenamiento considerable y suelen utilizarse en aplicaciones de neuro-rehabilitación. [5]

2.2. NextMind BCI

Una vez presentado el principio de funcionamiento de cada tipo de BCI, se procede a explicar en detalle el dispositivo utilizado en este proyecto.

El BCI NextMind es un dispositivo que permite a los usuarios interactuar con la tecnología usando únicamente sus ondas cerebrales. Este dispositivo portátil y no invasivo se coloca en la parte posterior de la cabeza del usuario. El dispositivo cuenta con electrodos que entran en contacto con el cuero cabelludo. Como se puede observar en la Figura 2.1 Este dispositivo, a diferencia de otros cascos de EEG para funciones más genéricas, concentra sus 9 electrodos aproximadamente en la zona de los lóbulos occipitales de los hemisferios cerebrales, donde se concentra el cortex visual del cerebro, cuyas neuronas generan potenciales de acción ante estímulos visuales.

Figura 2.1: Dispositivo NextMind BCI

NextMind utiliza un algoritmo de aprendizaje automático para interpretar estas señales eléctricas. Cuando un usuario enfoca su atención en un estímulo visual en una pantalla, se producen cambios en las ondas cerebrales que NextMind puede detectar y decodificar. Esto significa que, cuando un usuario se concentra en estímulos visuales con características diferentes, NextMind puede diferenciarlos y producir comandos asociados a dichos estímulos.

2.2.1. Especificaciones del sensor

- Dimensiones: 135x66x55mm
- Peso: 60 gramos
- Tamaño de la diadema: de 54 cm a 62 cm
- Sistema de clip: se puede enganchar directamente en una diadema, una gorra o un auricular VR/AR
- Nº de electrodos: 9 electrodos de alta calidad
- Batería: Batería de polímero de litio (3,7 V 240 mAh) - 8H de uso continuo - Recargable en 2H
- Conectividad: USB-C Conectado a una computadora o cualquier cargador USB (5V DC)

2.3. Unity 3D

2.3.1. Elección de Unity 3D

Para el desarrollo de este proyecto, se optó por utilizar Unity 3D como motor de videojuegos. Esta decisión se tomó principalmente porque es el único motor de videojuegos compatible con el dispositivo BCI NextMind, dado que su Kit de Desarrollo de Software (SDK) solo se encuentra disponible para Unity. De esta manera, nuestra elección se vio en cierta medida condicionada por la compatibilidad técnica de las herramientas a utilizar.

2.3.2. Facilidad de uso y comunidad activa

Una de las ventajas clave de utilizar Unity 3D fue su facilidad de uso y la abundancia de recursos disponibles en línea. Dado que no se contaba con experiencia previa en la utilización de este motor de videojuegos antes de iniciar el proyecto, se hizo imprescindible el poder contar con una comunidad activa y una amplia variedad de tutoriales en línea.

Esto permitió una curva de aprendizaje más suave y la capacidad de resolver problemas rápidamente a medida que surgían. Algunas veces, los problemas que se encontraron ya habían sido resueltos y discutidos en los foros de Unity 3D o se encontraron tutoriales relevantes que trataban directamente con los problemas encontrados.

2.3.3. Implementación de la solución

Unity 3D facilitó la creación de las dos versiones del videojuego que forman parte de este proyecto. Su versatilidad y flexibilidad permitieron el desarrollo del videojuego según las especificaciones y necesidades del proyecto.

Además, se logró una integración fluida con el dispositivo NextMind BCI gracias al SDK específico de este dispositivo para Unity. Esto significó que se pudo centrar más en el diseño y desarrollo del videojuego en sí, en lugar de tener que lidiar con problemas de compatibilidad o de integración de tecnologías.

Capítulo 3

Desarrollo

3.1. Diseño y desarrollo de la demo BCI

3.1.1. Aprendizaje de Unity 3D y animación de personajes

El proceso de desarrollo comenzó con un enfoque profundo en Unity 3D, un motor de juegos 3D robusto y versátil, pero también complejo para los recién llegados. A través de una serie de tutoriales en línea y experimentación práctica, me familiaricé con Unity 3D y sus diversas características y sistemas. Inicialmente me centré en el desarrollo del movimiento y animación del personaje en 3D. Un elemento crucial fue el Input System[1], que permite a los desarrolladores manejar fácilmente la entrada del usuario desde diferentes tipos de dispositivos. En el desarrollo de mi proyecto, tomé la decisión de emplear el teclado y el ratón como los dispositivos primordiales para el control del personaje dentro del juego. Asigné diversas acciones a estos dispositivos, entre las cuales se incluye la utilización del ratón para manejar la rotación de la cámara en tercera persona. Como se observa en la Figura 3.1, en uno de los Inputs denominado “Look”, se ha implementado el binding “Delta” (Pointer). Este binding se encarga de capturar los movimientos del ratón y traducirlos en una acción dentro del entorno virtual.

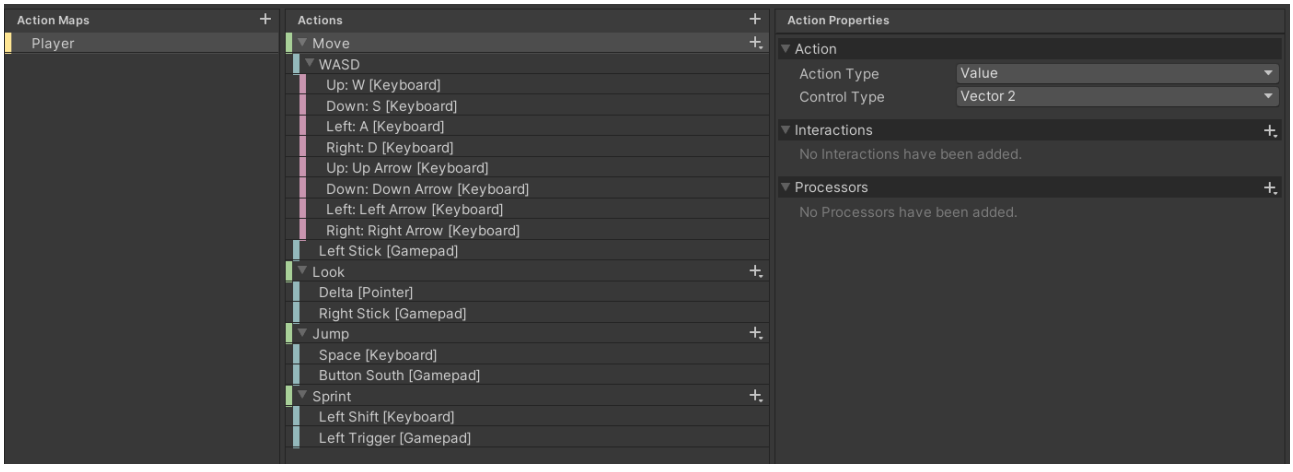

Figura 3.1: Unity Input Actions

[1] Unity Input System

Como guía y para facilitar el aprendizaje utilicé los “Starter Assets - Third Person Character Controller” [2].

Para el personaje principal, se utilizaron modelos y animaciones 3D para darle vida en el mundo del juego. Todas las animaciones y modelos de los personajes fueron adquiridos mediante la plataforma Mixamo[3] que ofrece una amplia variedad de animaciones y modelos de calidad gratuitos como el que se muestra en la Figura 3.2.

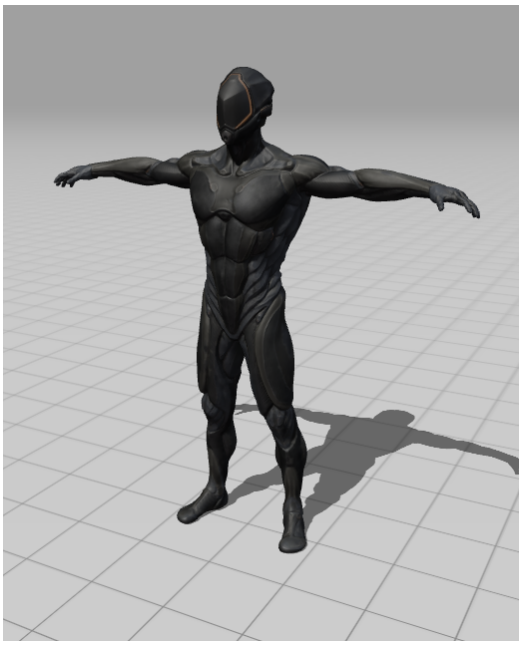

Figura 3.2: Modelo 3D del personaje principal del juego

Después de obtener el modelo y las animaciones del personaje, se procedió a implementarlas en Unity 3D utilizando el sistema Animator. El proceso comenzó con el aprendizaje de los conceptos básicos de la animación en Unity, que incluyen el uso del editor Animator para controlar y modificar las animaciones y el editor Animation para crear y editar secuencias de animación.

Una parte integral de este proceso fue aprender a utilizar los controladores de animación, que son esencialmente máquinas de estado finito que definen por selección y mezclado qué animación se reproduce en función de ciertas condiciones o parámetros. Esto implicó la creación de parámetros, que no son más que variables que se pueden modificar mediante **“scripts”** escritos en el lenguaje de programación C# ya que es el lenguaje que utiliza Unity 3D, y la definición de transiciones, que son las reglas que determinan cuándo se debe cambiar de un estado de animación a otro. Todo esto se puede apreciar en la Figura 3.3

Con el fin de entender el controlador de movimiento del personaje a través de la programación, desarrollé proyectos separados en Unity donde experimenté con técnicas para mejorar el movimiento del personaje. En la Figura 3.4 se presentan capturas de pantalla de este proyecto que ilustran una escena bastante sencilla donde probé diferentes animaciones y su manejo dentro de un script. La captura muestra mi exploración inicial

[2] Starter Assets - Third Person Character Controller
[3] Mixamo

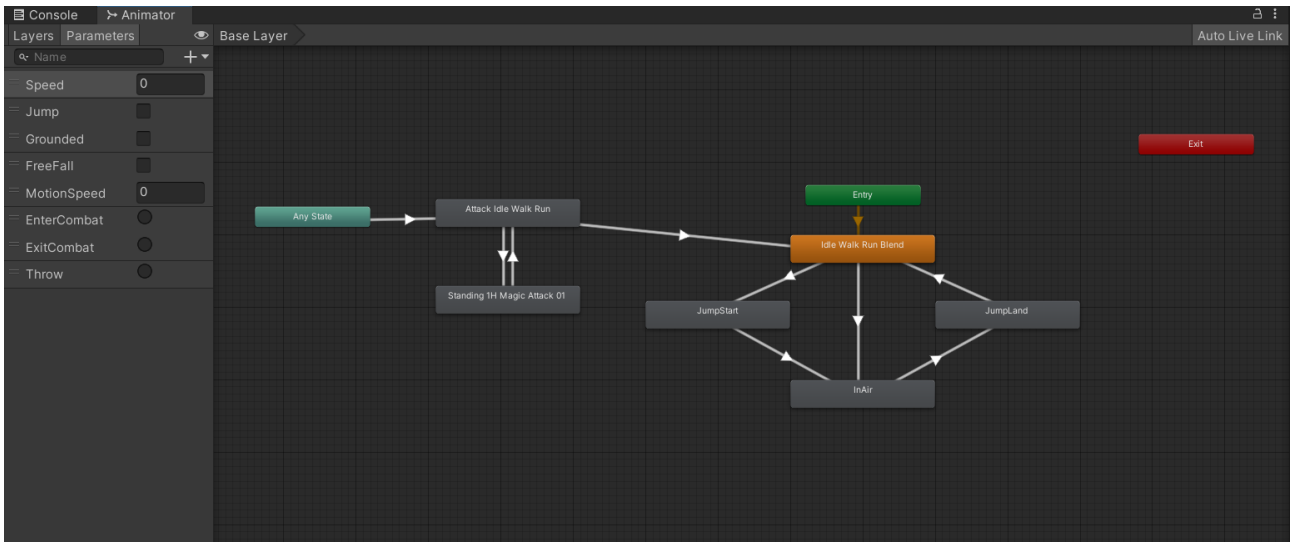

Figura 3.3: Ventana “Animator” de Unity

del uso de “**blend trees**” bidimensionales para enriquecer el movimiento del personaje y crear una apariencia más realista, dichos blend trees se explican más adelante en la página 14. Todo el conocimiento adquirido en estos proyectos iniciales viene de una serie de videos de Youtube dedicados al aprendizaje de animación de personajes 3D en Unity[3]

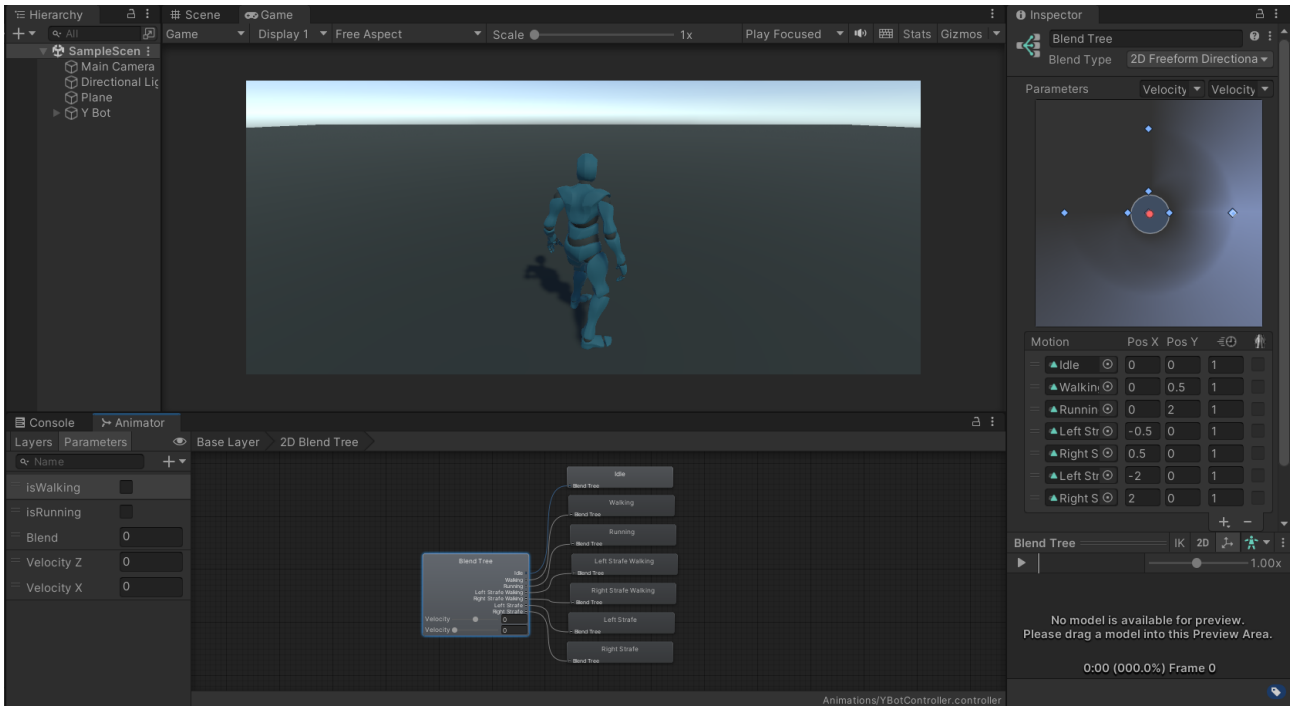

Figura 3.4: Proyecto de Aprendizaje de Animación

Gracias a los “**assets**” (Cualquier recurso que se vaya a utilizar para desarrollar el juego: scripts, animaciones, sonidos...) proporcionados por Unity en la “**Unity Asset Store**”[4], fue sencillo implementar el movimiento de los personajes. Aproveché los scripts de movimiento del paquete de controlador en tercera persona y los incorporé a mi proyecto. Este recurso fue fundamental para permitir interacciones fluidas y dinámicas

[4]Unity Asset Store

de los personajes dentro del entorno del juego. El código 3.1 muestra como se manejaba los parámetros que determinaban la velocidad del personaje en las distintas direcciones. El código completo se encuentra en el apéndice A.1

Listing 3.1: Código para controlar el personaje haciendo uso de un Blend Tree bidimensional

```
void changeVelocity(bool forwardPressed, bool leftPressed, bool
    rightPressed, bool runPressed, float currentMaxVelocity) {

    // if forward pressed increase the velocity in z direction
    if (forwardPressed && velocityZ < currentMaxVelocity) {
        velocityZ += Time.deltaTime * acceleration;
    }

    // if left pressed increase the velocity in x direction
    if (leftPressed && velocityX > -currentMaxVelocity) {
        velocityX -= Time.deltaTime * acceleration;
    }

    // if right pressed increase the velocity in x direction
    if (rightPressed && velocityX < currentMaxVelocity) {
        velocityX += Time.deltaTime * acceleration;
    }

    // decrease velocityZ
    if (!forwardPressed && velocityZ > 0.0f) {
        velocityZ -= Time.deltaTime * deceleration;
    }

    // increase velocityX if left is not pressed and velocityX < 0
    if (!leftPressed && velocityX < 0.0f) {
        velocityX += Time.deltaTime * deceleration;
    }

    // decrease velocityX if right is not pressed and velocityX > 0
    if (!rightPressed && velocityX > 0.0f) {
        velocityX -= Time.deltaTime * deceleration;
    }
}
```

A medida que avanzaba en mi comprensión de la animación en Unity, descubrí la utilidad de los Blend Trees. Un Blend Tree es una herramienta dentro del Animator que permite mezclar varias animaciones en diferentes proporciones para crear movimientos más fluidos y realistas. A través de un Blend Tree, pude hacer que el personaje principal se moviera suavemente entre las animaciones de caminar y correr dependiendo de la velocidad del personaje, la cual gestionaba desde un script (Véase en la Figura 3.5)

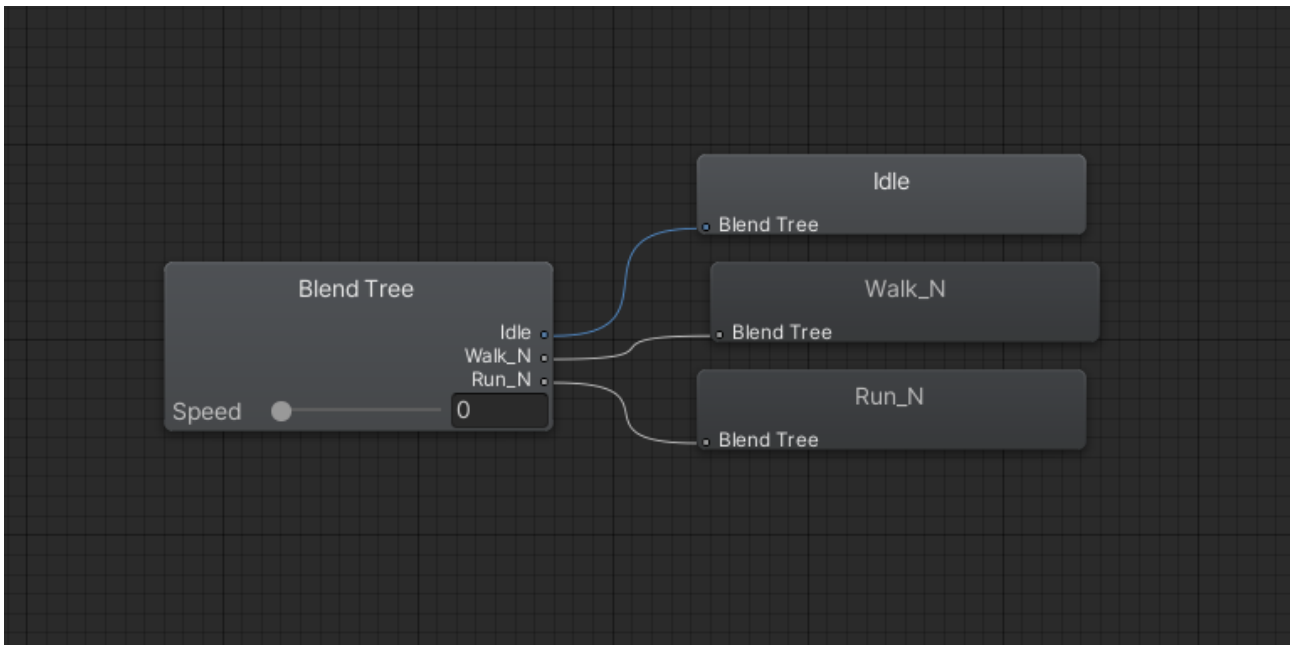

Figura 3.5: “Idle Walk Run” Blend Tree en Unity

Finalmente, a través de una serie de pruebas y ajustes, logré un sistema de animación que funcionaba bien con las interacciones y movimientos del personaje principal en el juego. Este proceso fue un reto, pero también una experiencia de aprendizaje muy gratificante. A través de él, gané una comprensión más profunda de cómo funcionan las animaciones en los videojuegos y cómo se pueden utilizar para mejorar la jugabilidad y la inmersión.

3.1.2. Selección y uso de assets para la creación de la escena

Con el personaje en movimiento, el siguiente paso fue la creación del entorno del juego. Se seleccionaron varios assets de la Unity Asset Store, una plataforma en línea que ofrece una variedad de assets gratuitos y premium que los desarrolladores pueden utilizar en sus proyectos. La selección del asset estuvo guiada por el estilo artístico deseado para el juego y la necesidad de estructuras altas para la mecánica de teletransportación que se implementaría más adelante. Los assets seleccionados permitieron la creación de un mapa que consistía en edificios altos separados por un vacío infinito como se puede apreciar en la Figura 3.6.

3.1.3. Exploración y uso del SDK de NextMind

A medida que avanzaba en el desarrollo del juego utilizando Unity 3D, también exploré en profundidad el SDK (Software Development Kit) de NextMind específicamente diseñado para Unity. Este SDK proporciona una serie de componentes y herramientas que facilitan enormemente la integración de la interfaz cerebro-computadora en el proyecto.

El SDK de NextMind se compone de dos tipos de componentes: los componentes centrales y los componentes de ejemplo. Los componentes centrales incluyen todos los archivos esenciales necesarios para construir una aplicación habilitada para NextMind. Aquí se encuentran las librerías centrales que exponen las principales clases, además

Figura 3.6: Diseño final del nivel usando assets de la unity asset store

de algunos componentes útiles, como "**prefab**"[5], "**shaders**"[6], etc. Por otro lado, los componentes de ejemplo ofrecen varias muestras de cómo utilizar el SDK, por ejemplo, cómo construir una aplicación de calibración personalizada o cómo aplicar un "NeuroTag" a un objeto.

Uno de las componentes principales en los que se basa el SDK de NextMind es el **NeuroTag**, que convierte cualquier objeto en un elemento interactivo a través de la mente. Al aplicar este componente a un objeto en la escena, se le aplica un shader que usa una textura que parpadea a una frecuencia especifica, esto es lo que detecta nuestro BCI. En la Figura 3.7 se muestra un NeuroTag con el patrón parpadeante.

El otro componente principal es el NeuroManager, que se encarga de gestionar la comunicación entre los NeuroTags en la escena y el núcleo del motor de NextMind.

Es por ello que siempre que se quiera utilizar un NeuroTag en un proyecto, es necesario que esté presente el prefabricado "NeuroManager" en alguna de las escenas, como se puede ver en la Figura 3.8 para este proyecto el NeuroManager se incluyó en la escena del menú principal. Este prefab gestiona eficazmente la frecuencia a la que parpadea cada NeuroTag, permitiendo así la integración exitosa del BCI en el juego. Solo es necesario una instancia del NeuroManager. Es más, tener varías instancias en diferentes escenas produjo fallos en el reconocimiento del dispositivo NextMind ya que el primer NeuroManager en conectar con el dispositivo evitaba que los demás pudieran comunicarse con el.

A través de varios proyectos de prueba, pude entender cómo se podrían emplear las capacidades de NextMind en el contexto del juego.

Utilizar un NeuroTag en algún objeto de la escena es tan fácil como añadir al objeto un componente "NeuroTag" en el inspector, el cual se verá como en la Figura 3.9. Este

[5]Prefabs: El prefab actúa como una plantilla a partir de la cual se pueden crear nuevas instancias del objeto en la escena

[6]Shaders: Los Shaders son Assets que contienen código e instrucciones para que la tarjeta gráfica ejecute

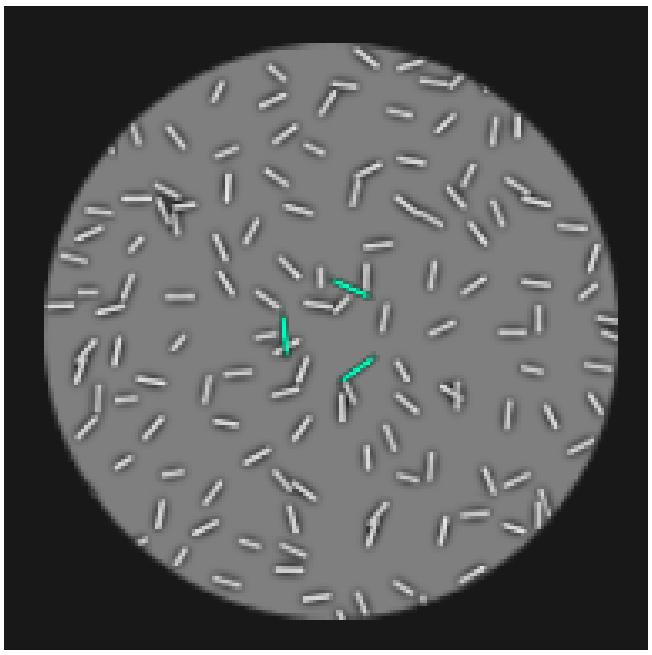

Figura 3.7: NeuroTag de NextMind

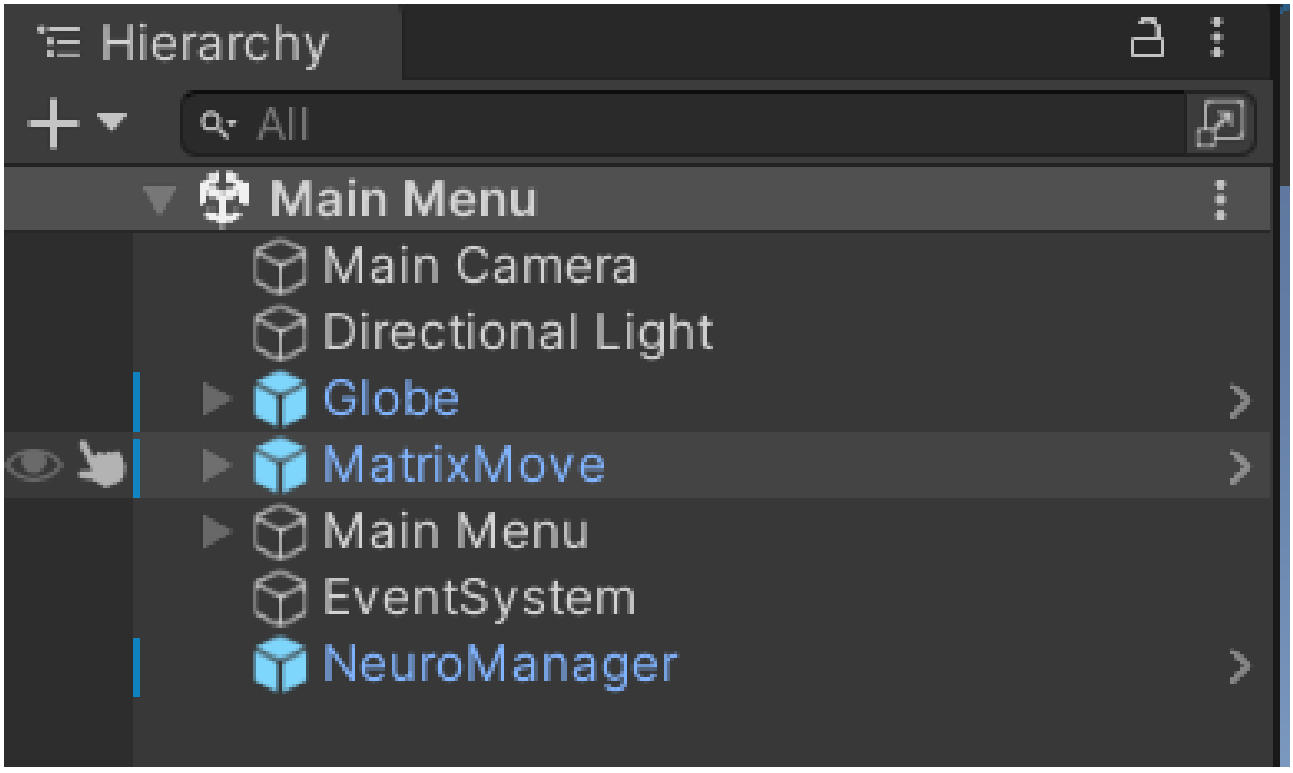

Figura 3.8: NeuroManager en la escena de Unity

componente permite llamar a una función de un script siempre que el NeuroTag en cuestión detecte que el jugador está mirándolo. De esta forma se puede implementar fácilmente algunas mecánicas como el teletransporte que se explica más en detalle en el apartado 3.1.4

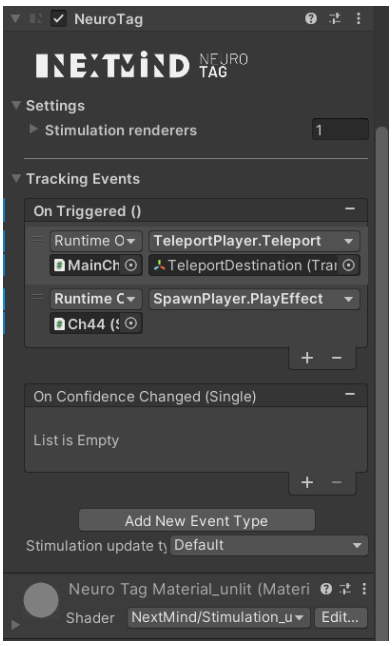

Figura 3.9: Componente NeuroTag en el inspector de Unity

3.1.4. Implementación de la mecánica del juego con BCI

Una vez que se entendió el funcionamiento del SDK de NextMind, se decidió que una mecánica interesante para el juego sería la teletransportación entre edificios, ya que controlar el movimiento del personaje por completo únicamente con el NextMind haría que el juego fuera algo tedioso de jugar tratándose de un juego 3D en tercera persona. Esto se debe a que por mucho que NextMind tenga un tiempo de respuesta bastante bueno (de unos 2 segundos aproximadamente) no es lo suficientemente rápido como para controlar el movimiento del personaje en 3D o deshacerse por completo de controles tradicionales como el teclado y el ratón.

Entonces, la mecánica de teletransporte sería controlada mediante el uso de “NeuroTags”, colocados estratégicamente en las diferentes zonas de la demo para permitir al jugador moverse libremente por el mapa y poder alcanzar lugares a los que no tendría acceso de no ser por esta mecánica. La implementación del teletransporte del personaje a un lugar asociado cuando se selecciona el neurotag, es una tarea fácil, ya que el componente NeuroTag permite configurar funciones “trigger” desde el inspector. Esta función se implementa en un script y es la que se ejecutará cuando el sistema detecta que el jugador está mirando el NeuroTag. En la Figura 3.9 se puede observar cómo se llama a la función “Teleport” del script “TeleportPlayer”, pasando como parámetro la posición a la que el jugador se teletransportará al mirar fijamente este NeuroTag. El código de dicha función se muestra en la Figura 3.10. El código es bastante sencillo, solo me aseguro de que el componente que controla el movimiento del personaje se desactive, cambio la posición del jugador a la pasada por parámetro y vuelvo a dar el control sobre el personaje al jugador, esto se hace para evitar errores del controlador al cambiar la posición del personaje[10]. Además de esto, y por añadirle algo más al diseño, cuando el personaje se teletransporta será rodeado por un efecto de partículas, el cual se llama también en el NeuroTag. El efecto de partículas en cuestión se puede ver en la Figura 3.11.

Como se puede apreciar en la Figura 3.12 los cubos de color gris colocados enfrente

```
public void Teleport(Transform destination)
{
    if (destination != null)
    {
        characterController.enabled = false;
        transform.position = destination.position;
        characterController.enabled = true;
    }
    else
    {
        Debug.LogError("Destination Transform not assigned. Please assign the destination Transform in the Inspector.");
    }
}
```

Figura 3.10: Función “Teleport” del script “TeleportPlayer”

Figura 3.11: Efecto de partículas para el teletransporte

del jugador le permiten moverse por el mapa con libertad.

Figura 3.12: Puntos de teletransporte en la demo

3.1.5. Incorporación de enemigos y combate

Con el mundo del juego y la mecánica de teletransportación establecidos, el siguiente paso fue añadir enemigos para que el jugador los combata. Esto implicó la selección y animación de un modelo de enemigo que se ajustara al estilo del juego, en la Figura 3.13 se puede ver el modelo 3D utilizado para los personajes enemigos.

Figura 3.13: Modelo 3D utilizado para los enemigos

Unity AI Navigation

Para el comportamiento de los enemigos en el juego, utilicé la funcionalidad de Navegación Artificial Inteligente (AI Navigation) de Unity[7]. Este sistema de navegación permite que los personajes se muevan de manera inteligente alrededor del entorno del juego, evitando obstáculos y siguiendo una serie de puntos de ruta definidos.

Cada enemigo en el juego tiene un Animator (Figura 3.14) parecido al del jugador, un componente Nav Mesh Agent, y un script personalizado que controla la IA del enemigo, denominado “EnemyAI”. Estos componentes permiten a los enemigos moverse y reaccionar de manera autónoma en el entorno del juego.

[7]Unity AI Navigation

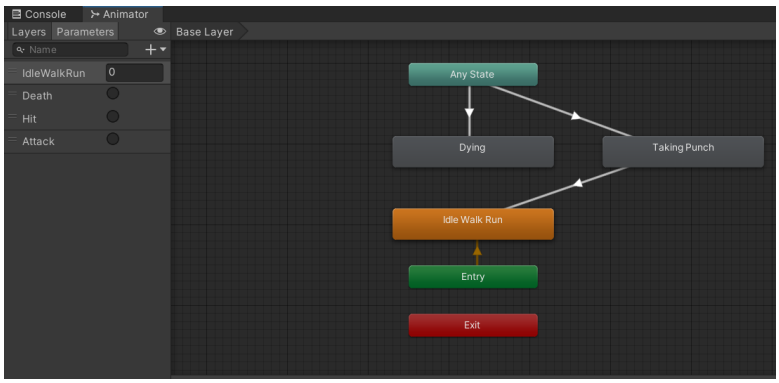

Figura 3.14: Animator Enemigo

El script EnemyAI

El script EnemyAI es la pieza central del comportamiento del enemigo. Se basa en la funcionalidad proporcionada por Unity para los agentes de malla de navegación (NavMeshAgent), permitiendo a los enemigos moverse por el entorno de juego de manera autónoma y eficaz.

El script define un array de transformaciones llamado Points, que representa a los puntos de ruta que el enemigo seguirá. En el método Start(), se obtienen referencias al NavMeshAgent y al Animator del enemigo. Se establece que el agente no se detenga automáticamente al alcanzar un objetivo, y se desactiva la actualización automática de la posición y rotación del agente para tener un mayor control sobre estas.

En el método GotoNextPoint(), se configura el destino del agente al siguiente punto en el array. Si el agente llega a su destino, lo cual se verifica en el método Update(), se detiene y se inicia una Coroutine que hace que el enemigo espere durante tres segundos antes de moverse hacia el próximo punto.

Listing 3.2: Método Update de la clase EnemyAI

```
void Update()
{
    if (!agent.pathPending && agent.remainingDistance < 0.5f)
    {
        // If the agent has reached the destination, stop and wait
        if (!agent.isStopped)
        {
            agent.isStopped = true;
            StartCoroutine(WaitAndGo());
        }
    }

    // Set the IdleWalkRun parameter based on the speed of the agent
    float speedPercent = agent.speed / maxAgentSpeed;
    animator.SetFloat("IdleWalkRun", speedPercent);

    // Apply position and rotation updates from the NavMeshAgent
    if (agent.enabled)
```

```
        {
            transform.position = agent.nextPosition;
            if (speedPercent > 0.1f)
            {
                Quaternion toRotation =
                  Quaternion.LookRotation(agent.desiredVelocity, Vector3.up);
                transform.rotation =
                  Quaternion.RotateTowards(transform.rotation,
                                           toRotation,
                                           agent.angularSpeed * Time.deltaTime
                                               );
            }
        }
    }
```

En el método Update(), también se actualiza el parámetro IdleWalkRun del animator en función de la velocidad del agente. Esto permite que la animación del enemigo se sincronice con su movimiento, dando la impresión de que el enemigo está caminando o corriendo hacia su destino.

Además, en cada frame se actualiza la posición y la rotación del objeto Transform del enemigo para coincidir con las del agente. De esta manera, incluso si el agente se detiene, su posición y rotación seguirán siendo las correctas.

Por último, en el método OnAnimatorMove(), se asegura que la siguiente posición del NavMeshAgent se actualice constantemente, incluso cuando no se está moviendo, copiándola de la rootPosition del animator. De igual manera, se actualiza la velocidad del agente para coincidir con el movimiento del Animator.

Listing 3.3: Método OnAnimatorMove() de la clase EnemyAI

```
// Ensure the NavMeshAgent's nextPosition is updated even when it's not
    moving
private void OnAnimatorMove()
{
    if (agent.enabled)
    {
        agent.nextPosition = animator.rootPosition;
        agent.velocity = animator.deltaPosition / Time.deltaTime;
    }
}
```

La combinación de este script con la funcionalidad de Navegación AI de Unity permite que los enemigos en el juego se muevan y se comporten de manera creíble, creando un desafío interesante para el jugador.

Véase el código completo en el Apéndice A.2.

Mecánica de combate

El combate en el juego se centra en el uso del BCI. El jugador utiliza su mente para seleccionar un enemigo y dispararle una bola de fuego. Esta bola no es más que un

prefab compuesto por varios efectos de partículas, extraídos de un paquete de assets suministrado por Unity en la Unity Asset Store[8] (Figura 3.15).

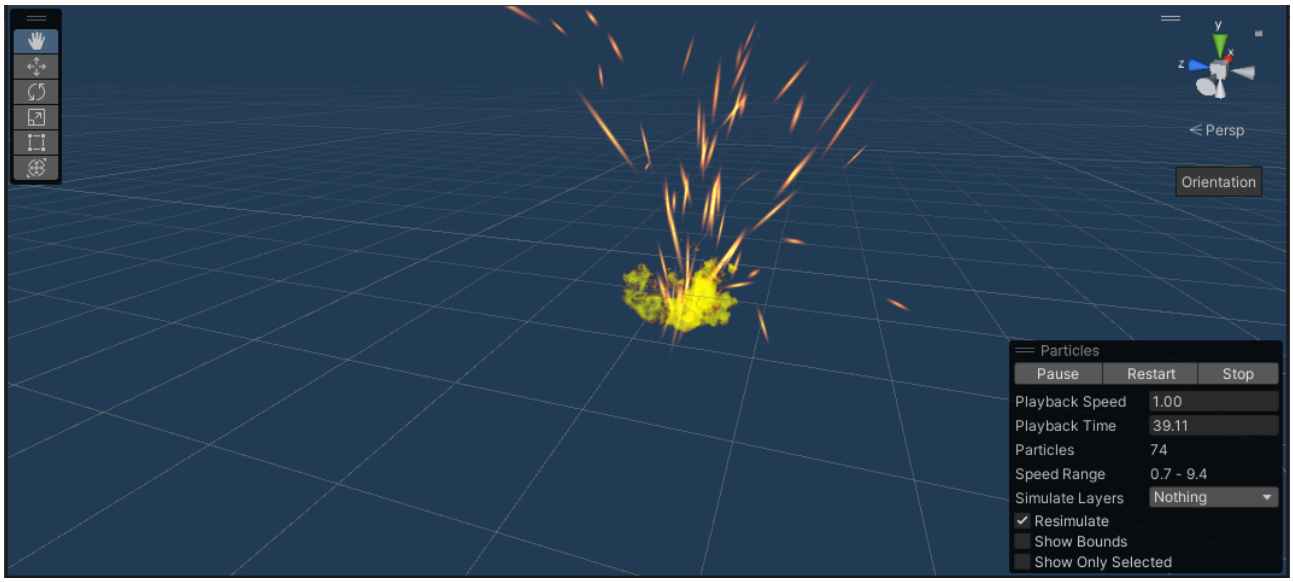

Figura 3.15: Prefab de la bola de fuego utilizado

Para una comprensión más clara, el proceso se divide en dos fases: la activación del modo de combate y el disparo de la bola de fuego. La activación del modo de combate se logra a través de un NeuroTag, incorporado en un Canvas que sirve como HUD para el jugador. Al concentrarse en este NeuroTag, el jugador entra en el modo de combate durante un tiempo limitado (5 segundos). Durante este periodo, el jugador debe concentrarse en otro NeuroTag, ubicado sobre el enemigo. Si el jugador tiene éxito en esta tarea, se dispara la bola de fuego, en caso de no ser capaz el jugador saldrá del modo de combate y tendrá que volver a entrar en el usando el NeuroTag del HUD. Cabe destacar que se añadió un reborde verde al NeuroTag del HUD que inicialmente es transparente y que al entrar en modo de combate se hace visible para dar al jugador feedback, si juntamos esto con la animación del modo de combate el jugador tiene varias indicaciones de que ha entrado en modo de combate con exito. Véase la Figura3.16

También se puede apreciar el código 3.4 utilizado para el funcionamiento de lo mencionado anteriormente. Para ver el código del script completo véase en el Apéndice A.3

[8]Unity Particle Pack

Listing 3.4: Código para lanzar una bola de fuego

```
public void shootFireball(Transform enemyTransform)
{
    enemyTRAN = enemyTransform;
    if (!isCharacterInCombatMode) { return; }
    characterANIM.SetTrigger("Throw");
    isChargingFireball = true;
    StartCoroutine(LaunchFireball());
}

IEnumerator LaunchFireball()
{
    yield return new WaitForSeconds(chargeTime + 1f);
    fireballOBJ = Instantiate(fireballPFAB,
                              spawnPointTRAN.position,
                              spawnPointTRAN.rotation);
    canMoveFireball = true;

    characterANIM.SetTrigger("ExitCombat");
    isCharacterInCombatMode = false;
    isChargingFireball = false;
}
```

Figura 3.16: Modo combate activado

Dado que el juego no proporciona ningún tipo de crosshair, decidí automatizar la trayectoria de la bola de fuego. En lugar de viajar en línea recta desde la posición del jugador, la bola de fuego sigue al enemigo seleccionado y colisiona con él. En el script

“AttackScript.cs” mencionado anteriormente se maneja el comportamiento de la bola de fuego de la siguiente manera:

Listing 3.5: Código para controlar la trayectoria de la bola de fuego

```
private void Update()
{
    if (canMoveFireball)
    {
        speed += fireballSpeed * Time.deltaTime;
        enemyLocationV3 = enemyTRAN.position;
        Vector3 fireballLocationV3 = fireballOBJ.transform.position;
        fireballLocationV3 = Vector3.MoveTowards(fireballLocationV3,
                                                 enemyLocationV3,
                                                 speed);

        fireballOBJ.transform.position = fireballLocationV3;

        if (fireballLocationV3 == enemyLocationV3)
        {
            Destroy(fireballOBJ);
            canMoveFireball = false;
            speed = 0;
        }
    }
}
```

En primer lugar, se actualiza la velocidad de la bola de fuego multiplicando la velocidad inicial (fireballSpeed) por el tiempo transcurrido desde el último fotograma (Time.deltaTime), incrementando así la velocidad de la bola de fuego de manera constante a medida que pasa el tiempo. Luego, se recupera la posición actual del enemigo (enemyTRAN.position) y se almacena en enemyLocationV3. La posición de la bola de fuego también se guarda en fireballLocationV3. Después, se utiliza la función Vector3.MoveTowards para mover la posición de la bola de fuego (fireballLocationV3) más cerca de la posición del enemigo (enemyLocationV3), a la velocidad calculada previamente. Posteriormente, la posición actualizada de la bola de fuego se aplica de vuelta a su objeto correspondiente en el juego (fireballOBJ.transform.position). Finalmente, si la posición de la bola de fuego coincide con la posición del enemigo (indicando que ha llegado a su destino y ha golpeado al enemigo), el objeto de la bola de fuego se destruye (Destroy(fireballOBJ)), se desactiva el interruptor canMoveFireball para detener el movimiento de la bola de fuego, y se reinicia la velocidad (speed) a 0.

La mecánica de salud de los enemigos se gestiona a través de un script separado. Cuando la bola de fuego golpea a un enemigo, detecta la colisión a través de un collider situado en la bola de fuego. Al detectarse la colisión, se resta la salud del enemigo y se refleja en un slider visible colocado sobre el enemigo. Este slider se actualiza con cada decremento en la salud del enemigo. En la Figura 3.17 se puede observar la barra de vida y el NeuroTag de los personajes enemigos dentro del juego.

Figura 3.17: Barra de salud de los enemigos y NeuroTag

Listing 3.6: Código para controlar el daño de la bola de fuego

```
public class Fireball : MonoBehaviour
{
    public int fireballDamage;

    void OnTriggerEnter (Collider collision)
    {
        Debug.Log("Fireball collided with " + collision.gameObject.name);
        if (collision.gameObject.name == "Ch15_nonPBR")
        {
            Debug.Log("Fireball hit an enemy!");
            EnemyHealth enemyHealth =
                collision.gameObject.GetComponent<EnemyHealth>();
            enemyHealth.TakeDamage(fireballDamage);
        }
    }
}
```

Listing 3.7: Código para gestionar la salud de los enemigos

```
public void TakeDamage(int damage)
{
    health -= damage;

    if (health <= 0)
    {
        animator.SetTrigger("Death");
        StartCoroutine(Die());
    }
    else
    {
        animator.SetTrigger("Hit");
    }
}
```

Además, como se puede apreciar en el código 3.7, he incorporado animaciones correspondientes a las acciones enemigas. Cuando la salud de un enemigo alcanza cero, se activa la animación de muerte. De igual manera, cada vez que un enemigo es golpeado por una bola de fuego, se dispara una animación de golpe. En los demás momentos, el enemigo se encuentra en un estado de movimiento o inactivo. Para ver el código del script que gestiona la salud de los enemigos véase en el Apéndice A.3

3.2. Diseño y desarrollo de la demostración con controles tradicionales

Después de completar la versión BCI del juego, se realizó una versión con controles tradicionales. Esta versión es esencialmente una copia de la versión BCI, pero el jugador utiliza el teclado y el ratón para controlar todas las mecánicas que antes se controlaban con el BCI. Esto implicó hacer ajustes en los scripts y controles para permitir la entrada tradicional del usuario.

Para entender las diferencias más profundas entre las dos versiones del juego, se detallan varios cambios clave en las secciones a continuación:

3.2.1. Interfaz de usuario y mecánicas de combate

Una de las principales diferencias se encuentra en la interfaz de usuario y en las mecánicas de combate. En la versión con controles tradicionales, se eliminó el NeuroTag del HUD que activaba el modo de combate en la versión BCI. En su lugar, se incorporó una mira (crosshair) en el centro de la pantalla para facilitar el apuntado del jugador. Este cambio permitió una mayor precisión y una sensación más tradicional en el juego.

La mecánica de combate también se alteró sustancialmente. En lugar de depender del BCI para entrar en modo de combate y apuntar al enemigo para lanzar una bola de fuego, se utilizan los botones del ratón. El click derecho activa el modo de combate y el izquierdo lanza la bola de fuego hacia el enemigo, apuntando con la mira. Se pueden ver estos cambios en la Figura 3.18.

Figura 3.18: Combate sin BCI con crosshair

3.2.2. Mecánica de teletransporte

La mecánica de teletransporte también cambió significativamente en la versión con controles tradicionales. En lugar de utilizar los cubos con NeuroTags para moverse por el mapa, como en la versión BCI, se implementó una mecánica diferente. Al presionar la tecla 'F', aparece una pequeña esfera en el centro de la mira que indica dónde puede teletransportarse el jugador.

Este cambio permite al jugador moverse libremente por el entorno del juego siempre que tenga visión directa del punto al que desea teletransportarse. Para seleccionar el destino, el jugador simplemente tiene que hacer click izquierdo con el ratón y se teletransportará (Figura 3.19).

Figura 3.19: Teletransporte sin BCI

3.3. Elementos adicionales del juego

3.3.1. Menú principal

Al iniciar el juego, los usuarios se encuentran con un menú principal que ofrece varias opciones: Jugar, Calibración NextMind y Salir. La opción “Jugar” inicia el juego y transporta al usuario a la escena principal. La opción “Calibración NextMind” permite a los usuarios calibrar el dispositivo BCI antes de empezar a jugar, un ajuste esencial para la versión que integra el BCI. Por último, la opción “Salir” permite al usuario cerrar el juego en cualquier momento. En la Figura 3.20 se puede ver el resultado final del menú principal.

Figura 3.20: Menú principal para la versión con BCI

3.3.2. Tutorial

El juego presenta un tutorial interactivo para guiar a los jugadores a través de las mecánicas y controles únicos del juego. Este tutorial se activa automáticamente al cargar la escena principal del juego. A continuación se muestran partes de cada script con los mensajes de tutorial para cada versión. El código completo que gestiona el avance del tutorial en el juego se puede encontrar en el Apéndice A.5.

Listing 3.8: Código con el contenido del tutorial para la versión BCI

```
private string[] tutorialSteps = new string[]
{
    "Usa las teclas A W S D para moverte",
    "Pulsa la barra espaciadora para saltar",
    "Pulsa la tecla Shift para correr",
    "Mira a los objetos que parpadean para interactuar con ellos",
    "Entra en modo de combate para atacar a los enemigos",
```

```
        "Dispara al enemigo mirando al recuadro que aparece en su cabeza",
        "Para moverte entre las distintas zonas mira a los cubos que parpadean
            "
};
```

Listing 3.9: Código con el contenido del tutorial para la versión sin BCI

```
private string[] tutorialSteps = new string[]
{
    "Usa las teclas A W S D para moverte",
    "Pulsa la barra espaciadora para saltar",
    "Pulsa la tecla Shift para correr",
    "Click derecho para entrar en modo de combate",
    "Apunta y dispara al barril con el click izquierdo",
    "Pulsa F para interactuar con el teletransportador",
    "Haz click para teletransoportarte",
    "Mata a todos los enemigos del nivel para ganar",
    "Usa los coches para subir a los edificios",
};
```

Para facilitar el aprendizaje de estas mecánicas, el tutorial está acompañado por un indicador de objetivos que se posiciona sobre el objetivo actual, orientando al jugador sobre qué hacer a continuación. Este indicador se mantiene una vez finalizado el tutorial, guiando a los jugadores hacia sus enemigos.

Figura 3.21: Ejemplo de indicador de objetivos en la versión BCI

Interacción durante el Tutorial

El tutorial es interactivo y permite a los jugadores avanzar a medida que prueban las diferentes mecánicas del juego. Por ejemplo, como se puede apreciar en la Figura 3.21, durante la lección sobre la interacción con objetos en la versión BCI, un indicador de objetivo se posiciona sobre un barril equipado con un NeuroTag. Al interactuar con el barril a través de la mirada, este explota y avanza el tutorial. Este enfoque práctico facilita la comprensión y el dominio de las mecánicas del juego por parte de los jugadores.

3.3.3. Escena de calibración del dispositivo BCI

Durante el proceso de calibración, el sistema supervisa el enfoque de atención del usuario en un NeuroTag específico. Esta operación, se repite 12 veces a lo largo de todo el proceso de calibración. Una calibración está compuesta por 12 pruebas (este número probablemente se reducirá en futuras versiones del SDK de NextMind), cada una de 3 segundos de duración. La única tarea del usuario es mantenerse enfocado en el NeuroTag supervisado durante cada prueba. La escena de ejemplo que proporciona NextMind se puede ver en la Figura 3.22.

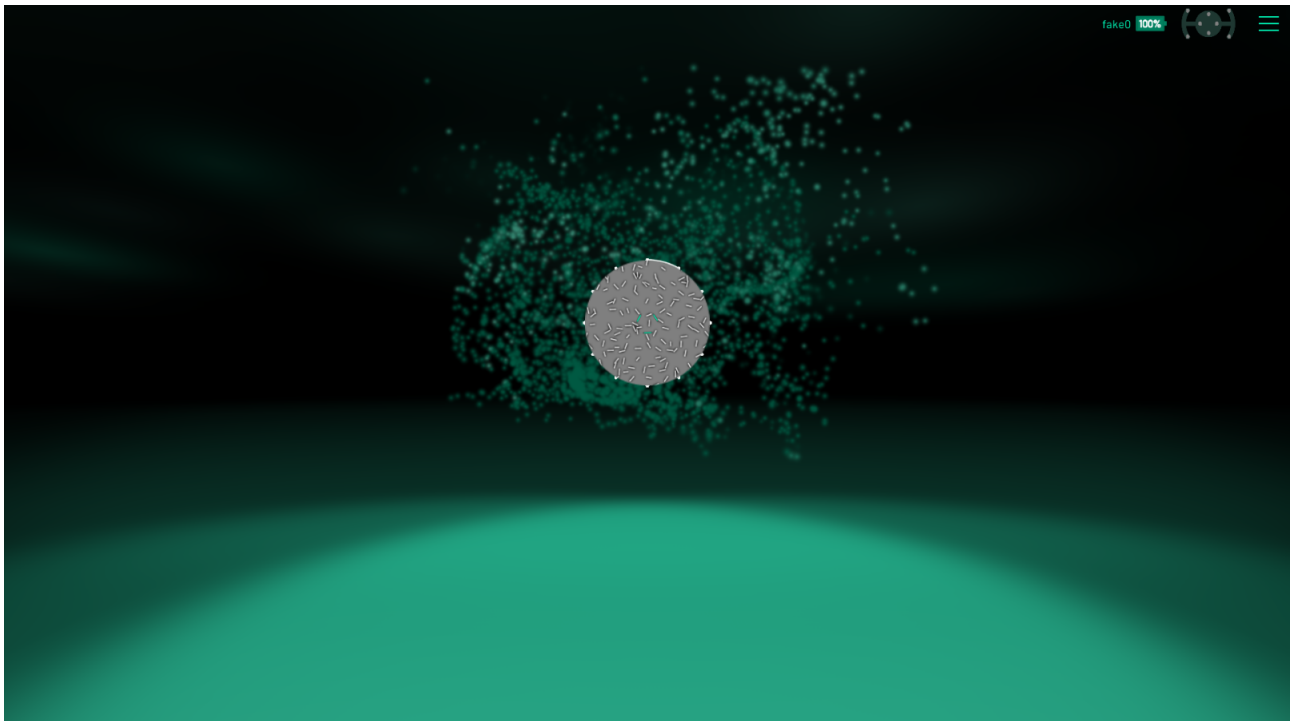

Figura 3.22: Escena de ejemplo de calibración de NextMind

Una vez que el usuario haya completado todas las pruebas, la calibración estará completa y el motor comenzará a procesar los datos en segundo plano. Una vez finalizada esta fase de procesamiento, el sistema enviará una puntuación a la aplicación que va desde 1 (peor) hasta 5 (mejor).

A continuación, se muestra la puntuación al usuario y se le permite al usuario reiniciar la calibración en caso de no estar satisfecho con la puntuación obtenida, en la Figura 3.23 se puede ver como se le muestra el resultado de la calibración al usuario en los ejemplos que proporciona NextMind. En el SDK, se proporciona un ejemplo de una escena de calibración que se ha usado en este proyecto.

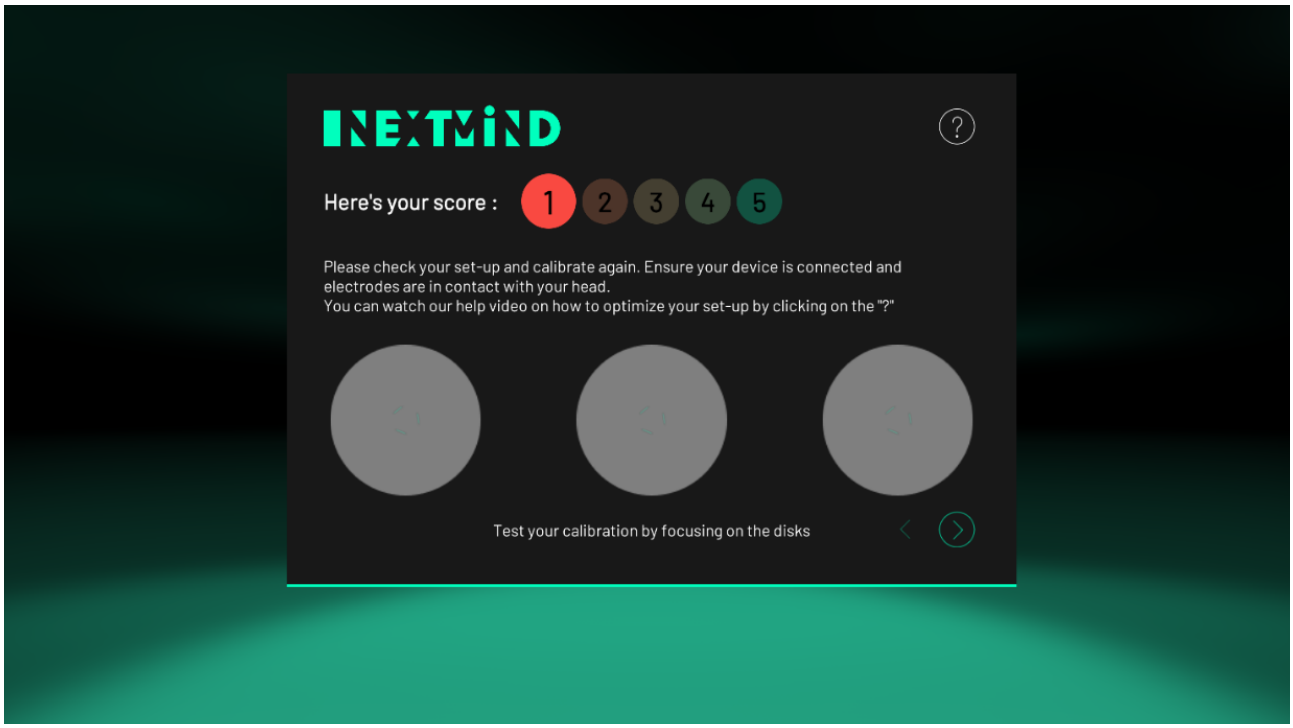

Figura 3.23: Puntuación de la calibración de NextMind

Capítulo 4

Análisis de las pruebas y resultados obtenidos

En esta sección, se van a detallar las pruebas realizadas, así como a mostrar y analizar los datos obtenidos para sacar conclusiones significativas.

4.1. Detalles de las pruebas realizadas

Las pruebas se realizaron siguiendo el procedimiento experimental descrito anteriormente en 1.4. Cada participante completó una sesión que incluía la calibración del dispositivo BCI, la prueba del juego en su versión BCI y su versión con controles tradicionales. Los datos de cada prueba se recogieron meticulosamente para su posterior análisis utilizando un formulario creado con Google Forms.

4.2. Presentación de los datos obtenidos

Tabla 4.1: Datos obtenidos en las pruebas para los 15 participantes

Longitud del pelo	Tipo de pelo	Nota de Calibración	Nº Errores	Luminosidad	Tiempo BCI	Tiempo sin BCI
Corto	Rizado	3	1	Media	03:06	02:21
Corto	Ondulado	2	2	Media	05:52	02:13
Corto	Lacio	3	1	Media	04:09	05:09
Corto	Rizado	3	1	Baja	03:48	05:25
Corto	Lacio	2	2	Media	12:06	02:38
Corto	Rizado	4	3	Media	02:22	02:55
Corto	Ondulado	2	2	Media	06:45	04:30
Corto	Lacio	3	1	Media	05:30	03:50
Corto	Lacio	3	1	Media	03:59	02:50
Corto	Lacio	4	0	Media	03:34	03:35
Calvo	-	2	0	Media	08:55	04:55
Largo	Ondulado	4	0	Baja	04:39	04:31
Largo	Rizado	2	1	Media	03:31	03:25
Largo	Rizado	4	1	Media	03:48	03:52
Largo	Rizado	1	3	Baja	08:10	04:30

Longitud del pelo	Nota de calibración media	Tiempo Medio BCI
Corto	3	05:07
Largo	3	05:02

Tabla 4.2: Media de Tiempos para la versión BCI por longitud de pelo

Nota de calibración	Tiempo medio BCI
2	07:25
3	04:06
4	03:35

Tabla 4.3: Media de Tiempos para la versión BCI por nota media de calibración

Nota de calibración	Nº de errores medio
1	3
2	2
3	1
4	1

Tabla 4.4: Media de errores por nota media de calibración

4.3. Interpretación y conclusiones

Basándonos en los datos obtenidos de nuestra muestra limitada de 15 participantes, es difícil establecer correlaciones claras debido a la baja variabilidad. Sin embargo, hemos extraído varias observaciones clave de estos resultados.

En primer lugar, no observamos una correlación aparente entre la longitud del cabello de los participantes y la puntuación obtenida en la calibración. Como se ilustra en la tabla 4.2, la longitud del cabello no parece influir en el tiempo que los participantes tardan en completar la versión del juego que utiliza BCI.

Un hallazgo notable es la aparente influencia de la puntuación de calibración en el rendimiento de los participantes. Los datos sugieren que una puntuación de calibración más baja puede dar lugar a tiempos de finalización más largos.

Si bien la mayoría de los participantes tardaron más en completar la versión del juego con BCI (con un tiempo medio de 05:10 minutos) en comparación con la versión sin BCI (03:47 minutos), es importante destacar ciertos casos atípicos. Algunos participantes lograron tiempos de finalización más cortos con la versión BCI que con la versión sin BCI. Un análisis más detenido de la tabla 4.1 revela que estos participantes lograron una puntuación de calibración máxima de 4.

Este hallazgo puede interpretarse de la siguiente manera: las mecánicas de la versión con BCI podrían haber sido más intuitivas para estos participantes, independientemente de su experiencia previa con videojuegos. Esto podría implicar que las mecánicas incorporadas en la versión BCI podrían ser más accesibles para un público más amplio, incluyendo a aquellos sin experiencia previa en juegos.

Además, la tabla 4.4 muestra un patrón interesante: el número medio de errores cometidos por los participantes aumenta a medida que disminuye su puntuación de

calibración. Teniendo en cuenta que el número mínimo de interacciones con el BCI necesarias para completar la demo es de 19, este hallazgo sugiere que la puntuación de calibración puede influir en la eficiencia del usuario al interactuar con el BCI.

4.3.1. Observaciones adicionales

Durante el estudio, también se recogieron datos valiosos relativos a la comodidad y satisfacción con la experiencia de utilizar el BCI. Aunque la muestra fue limitada, los resultados obtenidos indican una percepción generalmente favorable de la experiencia.

Se preguntó a los participantes que calificaran la comodidad del BCI en una escala de 1 a 5, siendo 1 muy incómodo y 5 muy cómodo. Los resultados, como se muestra en la Figura 4.1, fueron sorprendentemente positivos: solo el 6,7 % de los participantes calificaron la comodidad con una puntuación de 2, mientras que el resto consideró que el BCI era cómodo.

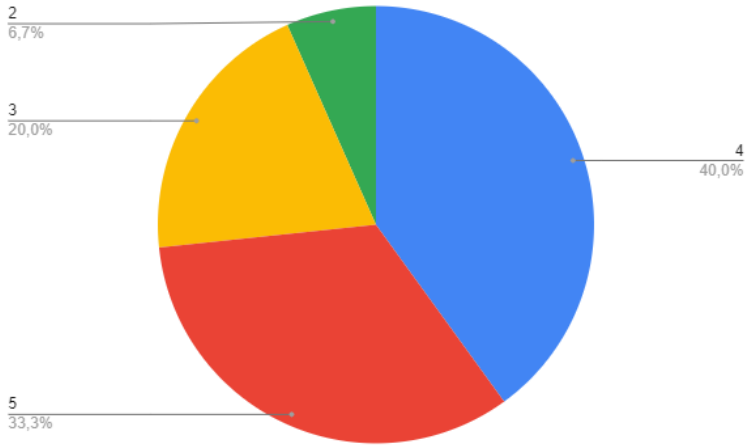

Figura 4.1: Gráfico circular de nivel de comodidad valorado del 1 al 5

Además, al preguntar a los participantes si recomendarían el uso del BCI para jugar a videojuegos, un destacable 83,3 % respondió afirmativamente, tal como se refleja en la Figura 4.2.

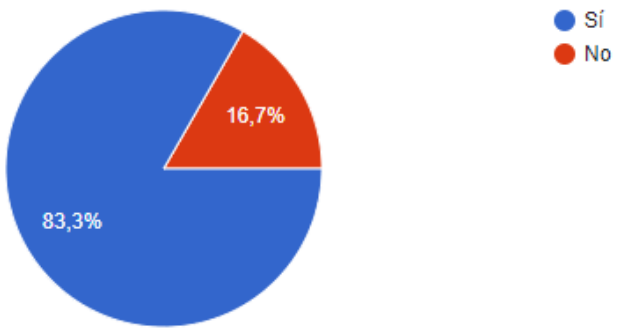

Figura 4.2: Recomendación del uso del BCI para jugar videojuegos

Se observaron algunas tendencias adicionales interesantes a lo largo del estudio. Por ejemplo, aunque los participantes con bajas puntuaciones de calibración inicialmente experimentaron una detección lenta de estímulos, el tiempo de respuesta pareció mejorar

a lo largo de la prueba y fue aún más notable cuando los participantes repetían la prueba varias veces. Además, en ambientes silenciosos, la concentración de los participantes y, por ende, el tiempo de respuesta, mejoraba notablemente.

También es importante resaltar que uno de los participantes, diagnosticado con TDAH, experimentó dificultades iniciales para calibrar el dispositivo. Aunque finalmente se adaptó al uso del BCI, mantener la concentración en los NeuroTags resultó ser una tarea exigente para él.

Además, se observó que los participantes con necesidad de corrección óptica experimentaban dificultades si no usaban sus gafas durante la prueba. Sin embargo, una vez que las gafas estaban puestas, su tiempo de respuesta mejoraba de manera significativa.

Capítulo 5

Presupuesto

5.1. Presupuesto

Concepto	Coste por horas	Horas	Coste total
Investigación	80	80	6400€
Documentación	80	50	4000€
Desarrollo	80	110	8000€
Licencia de Unity 3D para desarrolladores	80	-	370€
NextMind Dev Kit	80	-	399€
Análisis de las pruebas y resultados obtenidos	80	80	6400€
	Total	320	25.569€

Tabla 5.1: Presupuesto del proyecto

Capítulo 6

Conclusiones y líneas futuras

6.1. Desarrollo del prototipo

El proceso de desarrollo del prototipo presentó varios desafíos, siendo el mayor de ellos el aprendizaje de Unity 3D desde cero. Antes de este proyecto, no tenía experiencia previa en la creación de videojuegos, y tuve que considerar numerosos factores técnicos y de diseño que nunca antes había tenido en cuenta. En retrospectiva, esta experiencia ha cambiado fundamentalmente mi percepción y apreciación de los videojuegos.

El acceso a una gran cantidad de tutoriales y documentación en línea fue crucial para superar las barreras iniciales en el aprendizaje de Unity. Aunque este software tiene una curva de aprendizaje empinada para los novatos en el desarrollo de videojuegos, su rica funcionalidad y soporte de la comunidad lo convierten en una herramienta muy potente.

Uno de los aspectos más fluidos del proyecto fue la integración del BCI gracias al SDK de NextMind. A pesar de que el soporte de este SDK ha disminuido con el tiempo debido a la adquisición de NextMind por parte de Snapchat Inc, su integración en Unity fue relativamente sencilla. Sin embargo, es importante destacar que en ocasiones experimenté fallos del editor después de ejecutar el juego durante largos períodos de tiempo con múltiples NeuroTags activos en una escena.

Un desafío adicional fue la animación fluida y realista de los personajes enemigos controlados por IA. A pesar de numerosas iteraciones y ajustes en el script que controlaba su movimiento, la mejora en la fluidez de las animaciones fue mínima. Esto resaltó para mí la complejidad de animar personajes controlados por IA en comparación con los controlados por jugadores.

En cuanto a la optimización del proyecto, la gestión del tamaño del proyecto fue un desafío. A pesar de mis esfuerzos para reducir el tamaño del proyecto, la presencia de numerosos assets para el mapa hizo que el tamaño final del proyecto fuera considerable. Esto subraya la importancia de considerar la optimización y el tamaño del proyecto desde las primeras etapas de desarrollo.

Mi formación académica en programación y sistemas operativos avanzados fue esencial para superar estos desafíos. Las habilidades y conocimientos adquiridos en asignaturas como Algoritmos y Estructuras de Datos y Algoritmos y Estructuras de Datos Avanzados me permitieron entender y aplicar con éxito el lenguaje de programación C#. Además, los conceptos aprendidos en Sistemas Operativos Avanzados facilitaron mi comprensión de

las funciones IEnumerator en Unity, que son fundamentales para la ejecución de tareas en paralelo.

El desarrollo de este prototipo me permitió adquirir una profunda comprensión de Unity 3D y el desarrollo de videojuegos 3D, así como del campo emergente de la BCI. Estos conocimientos y habilidades serán de gran valor en mis futuros proyectos y en mi carrera profesional.

6.2. Desempeño del prototipo

En base a los datos recopilados, la investigación sugiere que el desempeño de los participantes al interactuar con la interfaz de BCI puede estar vinculado con la calificación obtenida durante la fase de calibración del dispositivo. La mayor cantidad de errores y tiempos de juego más largos asociados con una calificación de calibración más baja podrían interpretarse como indicativos de un ajuste menos eficiente entre el BCI y el usuario.

Además, nuestros datos indican que no hay una relación aparente entre las características físicas de los participantes (como la longitud del cabello) y el desempeño con el BCI. Esto sugiere que los dispositivos de BCI, al menos en el contexto de nuestro estudio, pueden ser versátiles y accesibles para una amplia variedad de usuarios.

Realizamos pruebas adicionales en distintos niveles de iluminación: baja y media en interiores, y en exteriores en condiciones desfavorables. Aunque los 15 participantes completaron las pruebas en interiores con niveles bajos y medios de luminosidad, el tamaño de la muestra nos impide extraer conclusiones sólidas sobre cómo estos niveles de luminosidad podrían afectar el rendimiento del BCI. Sin embargo, es relevante destacar que las pruebas en exteriores, a pesar de las condiciones adversas, mostraron que el BCI aún podía controlar la demo de manera efectiva. En la Figura 6.1 se muestra el entorno en el que se hicieron las pruebas en exterior y en la Figura 6.2 se muestra cómo se veía el juego en el exterior.

Figura 6.1: Setup de las pruebas en exterior

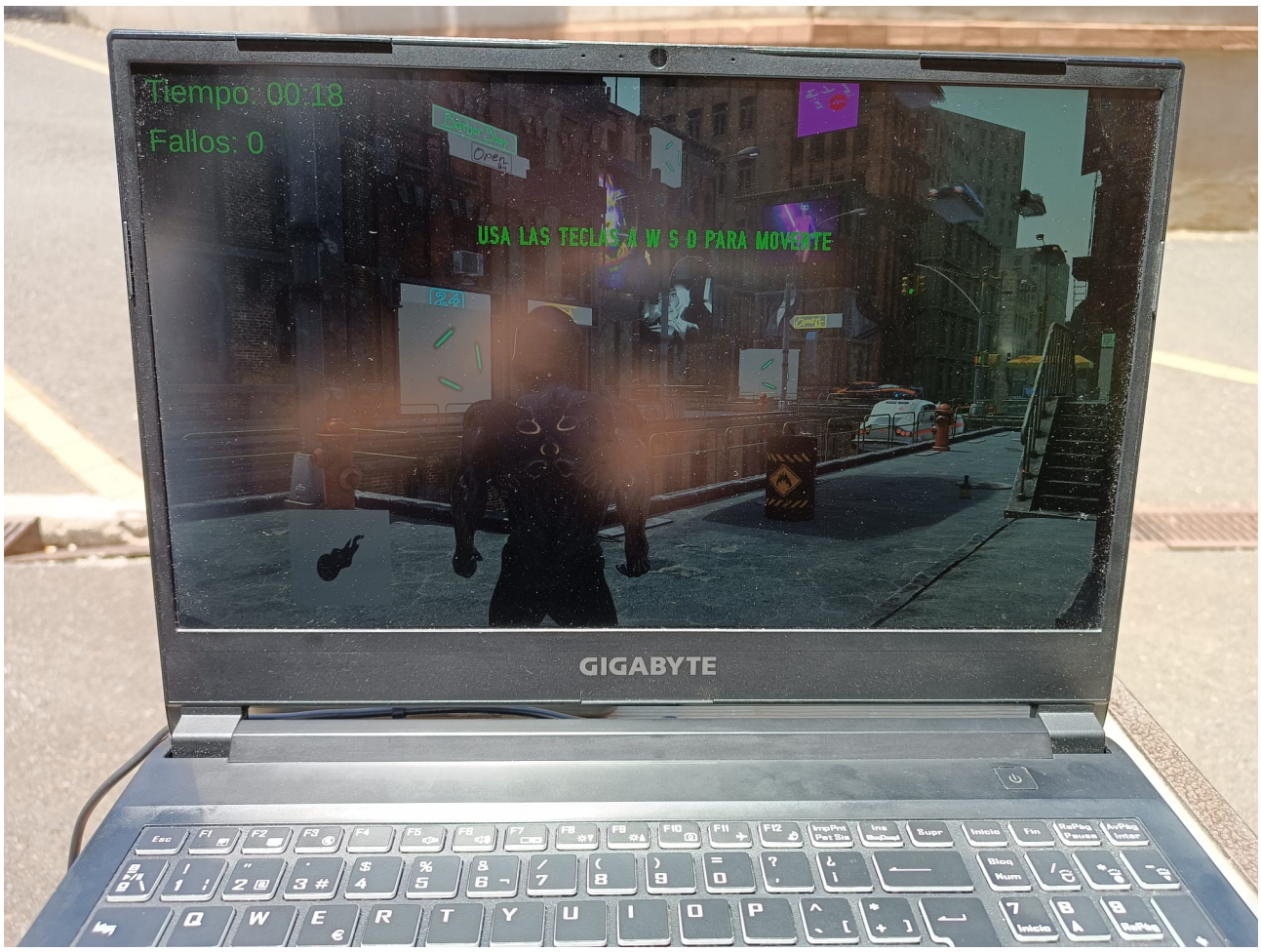

Figura 6.2: Prueba de la demo en exterior

Los resultados también muestran una preferencia significativa por la experiencia BCI entre los participantes, con una mayoría indicando que recomendarían su uso para jugar videojuegos. Esto sugiere que, a pesar de las dificultades potenciales durante la fase de calibración y los tiempos de juego más largos, la experiencia de jugar un videojuego con un BCI fue generalmente positiva para nuestros participantes.

No obstante, cabe destacar que los resultados de este estudio se basan en una muestra limitada y se requiere más investigación para confirmar y ampliar nuestros hallazgos.

6.3. Líneas futuras

6.3.1. Mejoras del prototipo

Existen numerosas oportunidades de mejora para el prototipo, las cuales se pueden dividir en dos categorías principales: mejoras relacionadas con Unity y aquellas relacionadas con el BCI.

En términos de Unity, algunos aspectos necesitan optimización. Como mencioné anteriormente, las animaciones de los personajes enemigos son un área que podría beneficiarse de una mejora sustancial. Otro aspecto a tener en cuenta son ciertas mecánicas de juego, en particular, el sistema de ataque en la versión sin BCI del juego. Este elemento resultó ser algo engorroso debido a que las “**hitboxes**” (áreas definidas para detectar

colisiones o interacciones) de los personajes enemigos eran bastante pequeñas, lo que dificultaba el disparo preciso.

En la versión con BCI, una mejora clave para facilitar una experiencia de juego más fluida se refiere a la mecánica del ataque. Cuando el jugador entra en modo de combate, debe fijar rápidamente su vista en el NeuroTag ubicado sobre el enemigo. Este NeuroTag, un elemento en 2D, estaba ubicado encima de la barra de salud del enemigo. Este diseño resultó ser problemático, dado que el movimiento constante del enemigo dificultaba que los participantes fijaran su vista en el NeuroTag ya que la posición de este dependía de la posición en la que se encontrara el enemigo y al moverse y rotar el NeuroTag en 2D rotaba con el enemigo haciendo que el jugador perdiera el enfoque en el patrón. Una posible solución a este problema sería convertir el NeuroTag de un objeto 2D a un objeto 3D, como una esfera.

En lo que respecta a las mejoras para el BCI, el tiempo de respuesta del dispositivo podría optimizarse aún más, especialmente si la intención es incorporar este tipo de tecnología en videojuegos 3D, como el de este proyecto. Además, sería recomendable mejorar la comodidad del dispositivo BCI. Tras períodos prolongados de uso, los participantes informaron que los electrodos del dispositivo se volvían incómodos. Mejorar el confort permitiría una interacción más duradera y placentera con el sistema.

6.3.2. Investigación

Aunque el presente estudio ha proporcionado hallazgos útiles, cuenta con varias limitaciones. En primer lugar, el tamaño de la muestra fue bastante pequeño, lo que puede afectar la generalización de los resultados. Los futuros estudios podrían beneficiarse de una muestra más grande y diversa para mejorar la validez y aplicabilidad de los hallazgos.

Además, la relación observada entre la puntuación de calibración y el rendimiento en el juego, aunque interesante, requiere una investigación más profunda. Podría ser útil investigar si esta relación se mantiene en diferentes tipos de juegos, o si se observa una mejora en el rendimiento con una práctica prolongada.

Asimismo, se observó una mejora en el rendimiento con el uso repetido del BCI y en entornos más silenciosos, lo que abre posibilidades para futuras investigaciones. Podría ser interesante explorar, por ejemplo, si existen métodos de entrenamiento o estrategias de juego que puedan optimizar el rendimiento del usuario al utilizar el BCI.

Además, nuestros hallazgos sobre las experiencias de los participantes con necesidades de corrección óptica y diagnósticos de TDAH podrían indicar direcciones para investigaciones futuras. Sería valioso explorar cómo las características individuales y las necesidades de los jugadores pueden influir en su experiencia con el BCI.

En términos generales, nuestra investigación ha arrojado luz sobre una serie de factores que pueden influir en la eficacia del BCI en los videojuegos. Aunque estos resultados son preliminares, ofrecen una base sólida para futuras investigaciones en este campo emergente.

Capítulo 7

Conclusions and future directions

7.1. Prototype Development

The prototype development process presented several challenges, with the biggest one being learning Unity 3D from scratch. Prior to this project, I had no previous experience in game development and had to consider numerous technical and design factors that I had never taken into account before. In retrospect, this experience has fundamentally changed my perception and appreciation of video games.

Access to a large number of online tutorials and documentation was crucial in overcoming the initial barriers in learning Unity. While this software has a steep learning curve for beginners in game development, its rich functionality and community support make it a very powerful tool.

One of the smoothest aspects of the project was the integration of BCI thanks to the NextMind SDK. Although the support for this SDK has decreased over time due to NextMind's acquisition by Snapchat Inc, its integration into Unity was relatively straightforward. However, it is important to note that I occasionally experienced editor crashes after running the game for long periods of time with multiple active NeuroTags in a scene.

An additional challenge was the smooth and realistic animation of AI-controlled enemy characters. Despite numerous iterations and adjustments to the script controlling their movement, the improvement in animation smoothness was minimal. This highlighted to me the complexity of animating AI-controlled characters compared to player-controlled ones.

Regarding project optimization, managing the project's size was a challenge. Despite my efforts to reduce the project's size, the presence of numerous assets for the map made the final project size considerable. This underscores the importance of considering optimization and project size from the early stages of development.

My academic background in programming and advanced operating systems was essential in overcoming these challenges. The skills and knowledge acquired in subjects such as Algorithms and Data Structures and Advanced Algorithms and Data Structures allowed me to understand and successfully apply the C# programming language. Additionally, the concepts learned in Advanced Operating Systems facilitated my understanding of IEnumerator functions in Unity, which are crucial for executing tasks in parallel.

The development of this prototype allowed me to gain a deep understanding of Unity 3D and 3D game development, as well as the emerging field of BCI. This knowledge and skills will be of great value in my future projects and professional career.

7.2. Prototype Performance

Based on the collected data, the research suggests that the performance of participants when interacting with the BCI interface may be linked to the rating obtained during the device calibration phase. The higher number of errors and longer gameplay times associated with a lower calibration rating could be interpreted as indicative of a less efficient adjustment between the BCI and the user.

Furthermore, our data indicates that there is no apparent relationship between participants' physical characteristics (such as hair length) and performance with the BCI. This suggests that BCI devices, at least in the context of our study, can be versatile and accessible to a wide variety of users.

We conducted additional tests under different lighting conditions: low and medium indoors, and outdoors under unfavorable conditions. Although all 15 participants completed the tests indoors with low and medium levels of luminosity, the sample size prevents us from drawing solid conclusions about how these levels of luminosity could affect BCI performance. However, it is relevant to highlight that the outdoor tests, despite adverse conditions, showed that the BCI could still effectively control the demo. In Figure 7.1, the environment in which the outdoor tests were conducted is shown, and in Figure 7.2, how the game looked outdoors is depicted.

Figura 7.1: Outdoor testing setup

The results also show a significant preference for the BCI experience among participants, with a majority indicating that they would recommend its use for playing video games. This suggests that, despite potential difficulties during the calibration phase and longer gameplay times, the experience of playing a video game with a BCI was generally positive for our participants.

However, it should be noted that the results of this study are based on a limited sample, and further research is needed to confirm and expand upon our findings.

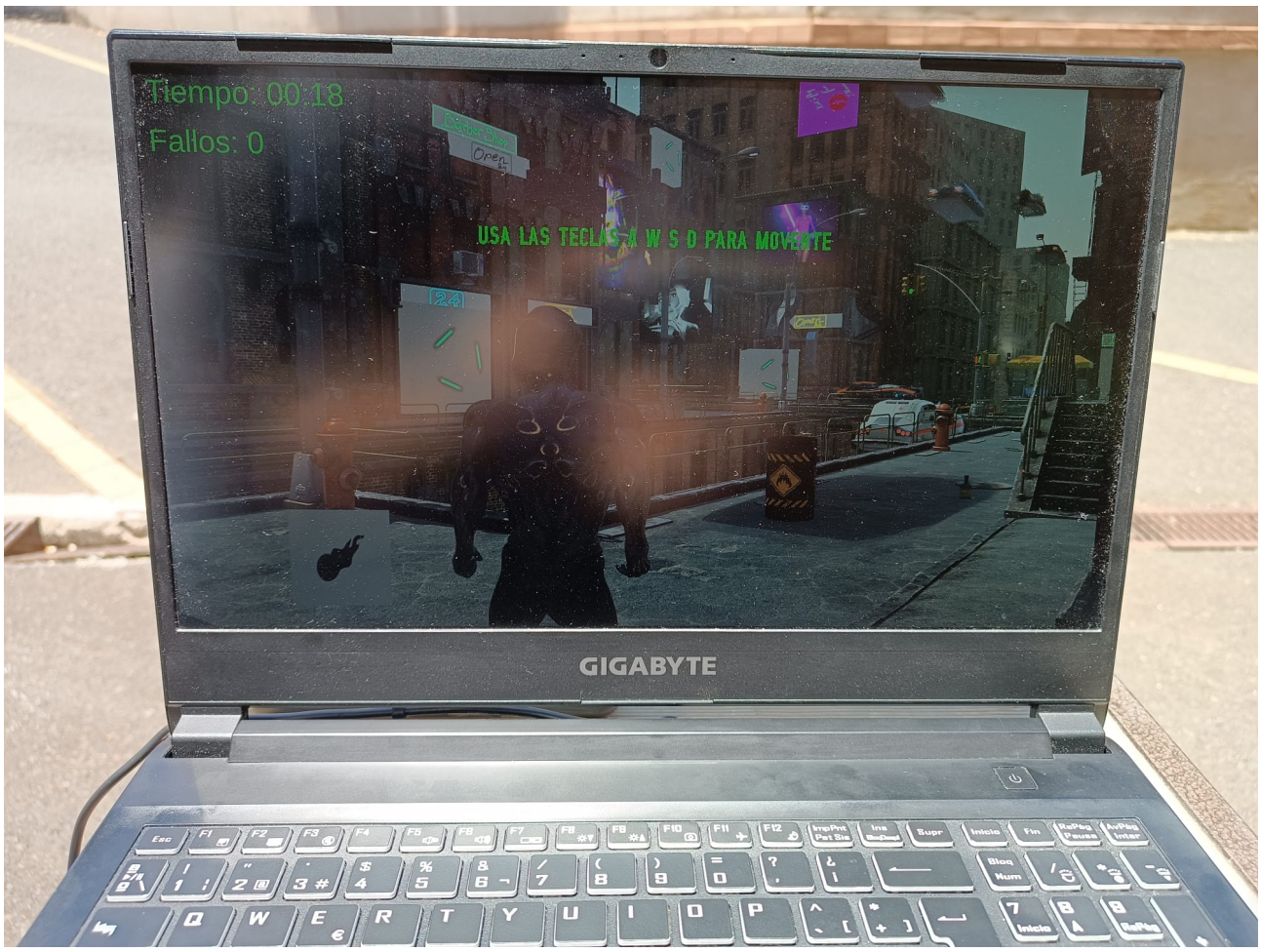

Figura 7.2: Demo testing outdoors

7.3. Future Directions

7.3.1. Prototype Improvements

There are numerous opportunities for improvement in the prototype, which can be divided into two main categories: improvements related to Unity and those related to the BCI.

In terms of Unity, certain aspects need optimization. As mentioned earlier, enemy character animations are an area that could benefit from substantial improvement. Another aspect to consider is certain gameplay mechanics, particularly the attack system in the non-BCI version of the game. This element proved to be somewhat cumbersome due to the enemy characters' "hitboxes"(areas defined for detecting collisions or interactions) being quite small, making precise shooting difficult.

In the BCI version, a key improvement to facilitate a smoother gameplay experience relates to the attack mechanic. When the player enters combat mode, they must quickly fix their gaze on the NeuroTag located above the enemy. This NeuroTag, a 2D element, was positioned above the enemy's health bar. This design proved to be problematic, as the enemy's constant movement made it difficult for participants to fix their gaze on the NeuroTag, as its position depended on the enemy's position, and when the NeuroTag rotated in 2D with the enemy, the player lost focus on the pattern. One possible solution to this problem would be to convert the NeuroTag from a 2D object to a 3D object, such

as a sphere.

Regarding improvements for the BCI, the device's response time could be further optimized, especially if the intention is to incorporate this type of technology into 3D video games, like the one in this project. Additionally, improving the comfort of the BCI device would be advisable. After prolonged periods of use, participants reported discomfort with the device's electrodes. Enhancing comfort would allow for longer and more enjoyable interaction with the system.

7.3.2. Research

Although the present study has provided valuable findings, it has several limitations. First, the sample size was quite small, which may affect the generalizability of the results. Future studies could benefit from a larger and more diverse sample to improve the validity and applicability of the findings.

Additionally, the observed relationship between calibration score and game performance, while intriguing, requires further investigation. It would be useful to explore whether this relationship holds true in different types of games, or if performance improves with prolonged practice.

Furthermore, an improvement in performance was observed with repeated use of the BCI and in quieter environments, opening up possibilities for future research. It could be interesting to explore, for example, whether there are training methods or gameplay strategies that can optimize user performance when using the BCI.

Additionally, our findings regarding the experiences of participants with vision correction needs and ADHD diagnoses may indicate directions for future research. It would be valuable to explore how individual characteristics and player needs can influence their experience with the BCI.

Overall, our research has shed light on a range of factors that can influence the effectiveness of BCI in video games. While these results are preliminary, they provide a solid foundation for future research in this emerging field.

Capítulo 8

Bibliografía

[1] Filiz, E. and Arslan, R. B. (2020). Design and implementation of steady state visual evoked potential based brain computer interface video game. pages 335–338.

[2] Glavas, K., Prapas, G., Tzimourta, K. D., Giannakeas, N., and Tsipouras, M. G. (2022). Evaluation of the user adaptation in a bci game environment. *Applied Sciences*, 12(24):12722.

[3] iHeartGameDev (2020). Unity's animation system.

[4] İşcan, Z. and Nikulin, V. V. (2018). Steady state visual evoked potential (ssvep) based brain-computer interface (bci) performance under different perturbations. *PloS one*, 13(1):e0191673.

[5] Neuper, C. and Pfurtscheller, G. (2001). Event-related dynamics of cortical rhythms: frequency-specific features and functional correlates. *International journal of psychophysiology*, 43(1):41–58.

[6] Perez-Valero, E., Lopez-Gordo, M. A., and Vaquero-Blasco, M. A. (2021). An attention-driven videogame based on steady-state motion visual evoked potentials. *Expert Systems*, 38(4):e12682.

[7] Pfurtscheller, G. and Neuper, C. (2001). Motor imagery and direct brain-computer communication. *Proceedings of the IEEE*, 89(7):1123–1134.

[8] Singh, A., Wang, Y.-K., King, J.-T., and Lin, C.-T. (2020). Extended interaction with a bci video game changes resting-state brain activity. *IEEE Transactions on Cognitive and Developmental Systems*, PP:1–1.

[9] Subramanian, R. R., Varma, K. Y., Balaji, K., Reddy, M. D., Akash, A., and Reddy, K.Ñ. (2021). Multiplayer online car racing with bci in vr. In *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 1835–1839.

[10] Vegas, J. (2019). How to write a teleport script in c# unity tutorial.

Apéndice A

Scripts de Unity

A.1. Código del script TwoDimensionalAnimationStateController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TwoDimensionalAnimationStateController : MonoBehaviour
{
    Animator animator;
    float velocityZ = 0.0f;
    float velocityX = 0.0f;
    public float acceleration = 2.0f;
    public float deceleration = 2.0f;
    public float maximumWalkVelocity = 0.5f;
    public float maximumRunVelocity = 2.0f;

    // increase performance
    int VelocityZHash;
    int VelocityXHash;

    // Start is called before the first frame update
    void Start()
    {
        // search the gameObject this script is attached to and get the
           animator component
        animator = GetComponent<Animator>();

        // increase performance
        VelocityZHash = Animator.StringToHash("Velocity Z");
        VelocityXHash = Animator.StringToHash("Velocity X");

    }

    // handles acceleration and deceleration
    void changeVelocity(bool forwardPressed, bool leftPressed, bool
       rightPressed, bool runPressed, float currentMaxVelocity) {
```

```
    // if forward pressed increase the velocity in z direction
    if (forwardPressed && velocityZ < currentMaxVelocity) {
        velocityZ += Time.deltaTime * acceleration;
    }

    // if left pressed increase the velocity in x direction
    if (leftPressed && velocityX > -currentMaxVelocity) {
        velocityX -= Time.deltaTime * acceleration;
    }

    // if right pressed increase the velocity in x direction
    if (rightPressed && velocityX < currentMaxVelocity) {
        velocityX += Time.deltaTime * acceleration;
    }

    // decrease velocityZ
    if (!forwardPressed && velocityZ > 0.0f) {
        velocityZ -= Time.deltaTime * deceleration;
    }

    // increase velocityX if left is not pressed and velocityX < 0
    if (!leftPressed && velocityX < 0.0f) {
        velocityX += Time.deltaTime * deceleration;
    }

    // decrease velocityX if right is not pressed and velocityX > 0
    if (!rightPressed && velocityX > 0.0f) {
        velocityX -= Time.deltaTime * deceleration;
    }
}

void lockOrResetVelocity(bool forwardPressed, bool leftPressed, bool
    rightPressed, bool runPressed, float currentMaxVelocity) {

    // reset velocityZ
    if (!forwardPressed && velocityZ < 0.0f) {
        velocityZ = 0.0f;
    }

    // reset velocityX
    if (!leftPressed && !rightPressed && velocityX != 0.0f && (velocityX >
        -currentMaxVelocity && velocityX < currentMaxVelocity)) {
        velocityX = 0.0f;
    }

    // lock forward
    if (forwardPressed && runPressed && velocityZ > currentMaxVelocity) {
        velocityZ = currentMaxVelocity;
    } else if (forwardPressed && velocityZ > currentMaxVelocity) {
        velocityZ -= Time.deltaTime * deceleration;
```

```
            if (velocityZ > currentMaxVelocity && velocityZ < (
                currentMaxVelocity + 0.05f)) {
                velocityZ = currentMaxVelocity;
            }
        } else if (forwardPressed && (velocityZ < currentMaxVelocity) &&
            velocityZ > (currentMaxVelocity - 0.05f)) {
            velocityZ = currentMaxVelocity;
        }

        // lock left
        if (leftPressed && runPressed && velocityX < -currentMaxVelocity) {
            velocityX = -currentMaxVelocity;
        } else if (leftPressed && velocityX < -currentMaxVelocity) {
            velocityX += Time.deltaTime * deceleration;
            if (velocityX < -currentMaxVelocity && velocityX < (-
                currentMaxVelocity + 0.05f)) {
                velocityX = -currentMaxVelocity;
            }
        } else if (leftPressed && (velocityX > -currentMaxVelocity) &&
            velocityX < (-currentMaxVelocity - 0.05f)) {
            velocityX = -currentMaxVelocity;
        }

        // lock right
        if (rightPressed && runPressed && velocityX > currentMaxVelocity) {
            velocityX = currentMaxVelocity;
        } else if (rightPressed && velocityX > currentMaxVelocity) {
            velocityX -= Time.deltaTime * deceleration;
            if (velocityX > currentMaxVelocity && velocityX < (
                currentMaxVelocity + 0.05f)) {
                velocityX = currentMaxVelocity;
            }
        } else if (rightPressed && (velocityX < currentMaxVelocity) &&
            velocityX > (currentMaxVelocity - 0.05f)) {
            velocityX = currentMaxVelocity;
        }
    }


    // Update is called once per frame
    void Update()
    {
        // input will be true if the player is pressing on the passed in key
            parameter
        // get key input from player
        bool forwardPressed = Input.GetKey(KeyCode.W);
        bool leftPressed = Input.GetKey(KeyCode.A);
        bool rightPressed = Input.GetKey(KeyCode.D);
        bool runPressed = Input.GetKey(KeyCode.LeftShift);

        // set current maxVelocity
```

```
            float currentMaxVelocity = runPressed ? maximumRunVelocity :
                maximumWalkVelocity;

            // handle changes in velocity
            changeVelocity(forwardPressed, leftPressed, rightPressed, runPressed,
                currentMaxVelocity);
            lockOrResetVelocity(forwardPressed, leftPressed, rightPressed,
                runPressed, currentMaxVelocity);

            // Set the parameters to our local variable values
            animator.SetFloat(VelocityZHash, velocityZ);
            animator.SetFloat(VelocityXHash, velocityX);
        }
    }
```

A.2. Código del script EnemyAI.cs

```
using System.Collections;
using UnityEngine;
using UnityEngine.AI;

public class EnemyAI : MonoBehaviour
{
    public Transform[] points;
    private int destPoint = 0;
    private NavMeshAgent agent;
    private Animator animator;

    public float maxAgentSpeed = 1f;


    void Start()
    {
        agent = GetComponent<NavMeshAgent>();
        animator = GetComponent<Animator>();

        agent.autoBraking = false;
        agent.updatePosition = false;
        agent.updateRotation = false;

        agent.speed = 0.4f * maxAgentSpeed;

        GotoNextPoint();
    }

    void GotoNextPoint()
    {
        if (points.Length == 0)
            return;
```

```
    // Start walking to the next point
    agent.destination = points[destPoint].position;
    destPoint = (destPoint + 1) % points.Length;
    agent.isStopped = false;
}

void Update()
{
    if (!agent.pathPending && agent.remainingDistance < 0.5f)
    {
        // If the agent has reached the destination, stop and wait
        if (!agent.isStopped)
        {
            agent.isStopped = true;
            StartCoroutine(WaitAndGo());
        }
    }

    // Set the IdleWalkRun parameter based on the speed of the agent
    float speedPercent = agent.speed / maxAgentSpeed;
    animator.SetFloat("IdleWalkRun", speedPercent);

    // Apply position and rotation updates from the NavMeshAgent
    if (agent.enabled)
    {
        transform.position = agent.nextPosition;
        if (speedPercent > 0.1f)
        {
            Quaternion toRotation =
              Quaternion.LookRotation(agent.desiredVelocity,
                                      Vector3.up);
            transform.rotation =
              Quaternion.RotateTowards(transform.rotation,
                                       toRotation,
                                       agent.angularSpeed *
                                       Time.deltaTime);
        }
    }
}

// Ensure the NavMeshAgent's nextPosition is
// updated even when it's not moving
private void OnAnimatorMove()
{
    if (agent.enabled)
    {
        agent.nextPosition = animator.rootPosition;
        agent.velocity = animator.deltaPosition / Time.deltaTime;
    }
}
```

```
    // Coroutine to make the enemy wait for a few seconds at each point
    IEnumerator WaitAndGo()
    {
        yield return new WaitForSeconds(3); // Wait for 3 seconds
        GotoNextPoint(); // Then go to the next point
    }
}
```

A.3. Código del script AttackScript.cs

```
using System.Collections;
using UnityEngine;

public class AttackScript : MonoBehaviour
{
    public GameObject characterOBJ;
    public GameObject fireballPFAB;

    public Transform spawnPointTRAN;
    private Transform enemyTRAN;

    public float chargeTime;
    public float fireballSpeed;
    public float combatModeDuration = 5f;

    Animator characterANIM;

    GameObject fireballOBJ;

    bool isCharacterInCombatMode;
    bool isChargingFireball;
    bool canMoveFireball;
    float speed;
    Vector3 enemyLocationV3;

    private void Start()
    {
        characterANIM = characterOBJ.GetComponent<Animator>();
    }

    private void Update()
    {
        if (canMoveFireball)
        {
            speed += fireballSpeed * Time.deltaTime;
            enemyLocationV3 = enemyTRAN.position;
            Vector3 fireballLocationV3 = fireballOBJ.transform.position;
            fireballLocationV3 = Vector3.MoveTowards(fireballLocationV3,
```

```
                                                    enemyLocationV3,
                                                    speed);

        fireballOBJ.transform.position = fireballLocationV3;

        if (fireballLocationV3 == enemyLocationV3)
        {
            Destroy(fireballOBJ);
            canMoveFireball = false;
            speed = 0;
        }
    }
}

public void enterCombatMode()
{
    if (isCharacterInCombatMode) { return;}
    isCharacterInCombatMode = true;
    characterANIM.SetTrigger("EnterCombat");
    StartCoroutine(ExitCombatMode());
}

public void shootFireball(Transform enemyTransform)
{
    enemyTRAN = enemyTransform;
    if (!isCharacterInCombatMode) { return; }
    characterANIM.SetTrigger("Throw");
    isChargingFireball = true;
    StartCoroutine(LaunchFireball());
}

IEnumerator LaunchFireball()
{
    yield return new WaitForSeconds(chargeTime + 1f);
    fireballOBJ = Instantiate(fireballPFAB,
                              spawnPointTRAN.position,
                              spawnPointTRAN.rotation);
    canMoveFireball = true;

    characterANIM.SetTrigger("ExitCombat");
    isCharacterInCombatMode = false;
    isChargingFireball = false;
}

IEnumerator ExitCombatMode()
{
    yield return new WaitForSeconds(combatModeDuration);

    // If no fireball was launched or
    // is being charged in the time frame
    if (!canMoveFireball && !isChargingFireball)
```

```
        {
            characterANIM.SetTrigger("ExitCombat");
            isCharacterInCombatMode = false;
        }
    }
}
```

A.4. Código del script EnemyHealth.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class EnemyHealth : MonoBehaviour
{
    public float health;
    public float maxHealth;

    public WaypointScript waypoints;

    public GameObject healthBarUI;
    public Slider slider;

    public Animator animator;

    void Start()
    {
        health = maxHealth;
        slider.value = CalculateHealth();
    }

    void Update()
    {
        slider.value = CalculateHealth();

        if (health < maxHealth)
        {
            healthBarUI.SetActive(true);
        }

        if (health > maxHealth)
        {
            health = maxHealth;
        }
    }

    float CalculateHealth()
    {
```

```
        return health / maxHealth;
    }

    public void TakeDamage(int damage)
    {
        health -= damage;

        if (health <= 0)
        {
            animator.SetTrigger("Death");
            StartCoroutine(Die());
        }
        else
        {
            animator.SetTrigger("Hit");
        }
    }

    IEnumerator Die()
    {
        yield return new WaitForSeconds(5f);
        Destroy(gameObject);
    }
}
```

A.5. Código del script TutorialManager.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class TutorialManager : MonoBehaviour
{
    public GameObject messagePanel;
    public TextMeshProUGUI messageText;
    public float transitionTime = 1f;
    public bool NextMindUsed = false;
    public bool LookedHUD = false;
    public bool LookedEnemy = false;
    public bool LookedTP = false;

    private string[] tutorialSteps = new string[]
    {
        "Usa las teclas A W S D para moverte",
        "Pulsa la barra espaciadora para saltar",
        "Pulsa la tecla Shift para correr",
        "Mira a los objetos que parpadean para interactuar con ellos",
        "Entra en modo de combate para atacar a los enemigos",
```

```
        "Dispara al enemigo mirando al recuadro que aparece en su cabeza",
        "Para moverte entre las distintas zonas mira a los cubos que parpadean
            "
    };

    private int currentStep = 0;
    private bool transitioning = false;

    void Start()
    {
        StartCoroutine(ShowMessage());
    }

    void Update()
    {
        if (!transitioning)
        {
            if (currentStep == 0 && (Input.GetAxisRaw("Horizontal") != 0 ||
                Input.GetAxisRaw("Vertical") != 0))
            {
                StartCoroutine(ChangeStep());
            }
            else if (currentStep == 1 && Input.GetKeyDown(KeyCode.Space))
            {
                StartCoroutine(ChangeStep());
            }
            else if (currentStep == 2 && Input.GetKeyDown(KeyCode.LeftShift))
            {
                StartCoroutine(ChangeStep());
            }
            else if (currentStep == 3 && NextMindUsed == true)
            {
                StartCoroutine(ChangeStep());
            }
            else if (currentStep == 4 && LookedHUD == true)
            {
                StartCoroutine(ChangeStep());
            }
            else if (currentStep == 5 && LookedEnemy == true)
            {
                StartCoroutine(ChangeStep());
            }
            else if (currentStep == 6 && LookedTP == true)
            {
                StartCoroutine(ChangeStep());
            }
        }
    }

    public int GetCurrentStep()
    {
```

```
    return currentStep;
}

public void UsedNextmind(int target)
{
    if (target == 0)
    {
        NextMindUsed = true;
    }
    else if (target == 1)
    {
        LookedHUD = true;
    }
    else if (target == 2)
    {
        LookedEnemy = true;
    }
    else if (target == 3)
    {
        LookedTP = true;
    }
}

IEnumerator ShowMessage()
{
    messageText.text = tutorialSteps[currentStep];
    messageText.color = new Color(messageText.color.r, messageText.color.g
        , messageText.color.b, 0);
    while (messageText.color.a < 1.0f)
    {
        messageText.color = new Color(messageText.color.r, messageText.
            color.g, messageText.color.b, messageText.color.a + (Time.
            deltaTime / transitionTime));
        yield return null;
    }
}

IEnumerator FadeOutText()
{
    while (messageText.color.a > 0.0f)
    {
        messageText.color = new Color(messageText.color.r, messageText.
            color.g, messageText.color.b, messageText.color.a - (Time.
            deltaTime / transitionTime));
        yield return null;
    }
}

IEnumerator ChangeStep()
{
    transitioning = true;
```

```
        yield return FadeOutText();
        currentStep++;
        if (currentStep < tutorialSteps.Length)
        {
            yield return ShowMessage();
        }
        else
        {
            messagePanel.SetActive(false);
        }
        transitioning = false;
    }
}
```