



POLITÉCNICA
"Ingeniamos el futuro"

CAMPUS
DE EXCELENCIA
INTERNACIONAL



Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

TRABAJO FIN DE GRADO

Desarrollo de técnicas de visualización colaborativa
sobre escenas neurocientíficas utilizando el motor de
juegos UDK 4

Autor: Aitor Garcia Alvarez

Director: Vicente Martín Ayuso

MADRID, JUNIO 2017

“Miracles are illusions caused by insufficient observation and understanding. They’re just... glorious misunderstandings”

Tanya Degurechaff

“People who can’t throw something important away can never hope to change anything.”

Armin Arelet

Resumen

En este proyecto se realiza un estudio de técnicas de visualización científica colaborativa y métodos de comunicación interactiva, además del estudio del entorno de trabajo.

Así mismo se desarrolla una infraestructura compuesta de una herramienta de visualización científica implementada sobre el motor de juegos Unreal Engine, un sistema de persistencia y un servicio web RESTful para la gestión de la persistencia, que además proporciona una interfaz gráfica para la extracción de información producida dentro de la herramienta de análisis.

Finalmente se obtiene como resultado un sistema que proporcionará a investigadores la capacidad de colaborar en estudios de análisis de escenas neurocientíficas dentro de la iniciativa Human Brain Project (HBP).

Abstract

This project bring out a study of collaborative scientific visualization techniques and interactive communication methods, as well as the study of the work environment.

Also it has been developed is an infrastructure composed of a scientific visualization tool implemented on the Unreal Engine game engine, a persistence system and a RESTful web service for the management of persistence, which also provides a graphical interface for the extraction of information produced within the analysis tool.

Finally, the result is a system that will provide researchers with the ability to collaborate in neuroscientific scene analysis studies within the Human Brain Project (HBP).

Tabla de Contenidos

Introducción	7
Estado del Arte	8
Visualización	8
Técnicas de visualización colaborativa	9
El motor de juegos Unreal Engine 4	10
Investigación Anterior	11
Infraestructura adicional	13
Evaluación sobre el uso de BBDD SQL o NoSQL	14
Servicio Web RESTful como Middleware	15
Diseño de la Herramienta	16
Especificación de Requisitos	16
Especificación de requisitos en la herramienta de visualización colaborativa	18
Especificación de requisitos para el sistema de persistencia	20
Especificación de requisitos para el servicio web dedicado al sistema de persistencia	21
Especificación de requisitos para servicio web de visualización de datos almacenados en el sistema de persistencia	21
Arquitectura del Software del Framework	22
Implementación de la herramienta de visualización colaborativa	27
Arquitectura del Motor	28
Diagrama de Clases	31
Detalles de Implementación	34
Sistema Multiusuario	37
Replicación de Datos	38
Interfaz	41

Visualización de bosques neuronales	47
Comunicación con el Sistema de Persistencia	48
Usuarios	49
Métodos de Comunicación entre usuarios	50
Comunicación a través de Chat	50
Comunicación a través de Voz	51
Mecanismo You See What I See	52
Selección de Neuronas	53
Información Contextual	54
Captura de Imagenes	55
Sistema de Repetición	56
Realidad Virtual	56
Implementación del servicio Web y sistema de persistencia	58
Diagrama de Clases	58
Identificación de usuario	59
Sandbox	60
Visualización de Capturas	61
Visor de Chats	62
Resultados	64
Conclusiones	64
Bibliografía	66

1. Introducción

El mundo del videojuego ha estado creciendo desde sus inicios en los años 70, llegando a un nivel de negocio superior al de música o películas^[1]. Existe mercadotecnia y campañas publicitarias para posicionar a un videojuego por encima de otros. La influencia de este sector en el mercado global ha marcado un notable esfuerzo en la evolución tecnológica tanto de hardware como software enfocados en la industria del videojuego.

Los avances tecnológicos en el desarrollo de aplicaciones gráficas permiten que otros campos científicos/profesionales como la simulación, visualización arquitectónica o análisis gráfico, se vean beneficiados en gran medida^[2].

Unreal Engine 4 está considerado uno de los motores gráficos más importantes del mercado^[3]. Se compone de conjunto de herramientas integradas para que los desarrolladores de juegos diseñen y construyan juegos, simulaciones y visualizaciones. Se encuentra disponible de forma gratuita y utiliza el lenguaje C++ para el desarrollo.

Como parte de la iniciativa Human Brain Project (HBP), se ha investigado el uso de este motor como herramienta de visualización científica en un proyecto anterior, demostrando su utilidad en este campo.

Los sistemas de visualización científica permiten que la emergente cantidad de datos procedentes de diferentes tareas tales como resultados de complejas simulaciones físicas, stacks de imágenes de microscopía de alta calidad o mallados de muy alta resolución, sean analizados proporcionando la capacidad de extraer conocimiento útil.

Dentro de las técnicas de análisis científico, la visualización colaborativa permite que grupos de usuarios analicen datos de manera conjunta, compartiendo la propia experiencia de los usuarios además de acelerar la rapidez y fiabilidad de los análisis.

El objetivo de este proyecto consiste en el desarrollo de una infraestructura que pretende proporcionar un entorno colaborativo donde diferentes usuarios participen en tareas de análisis científico.

Esta infraestructura se compone de una herramienta de visualización implementada con el motor de juegos Unreal Engine 4. Permitirá cargar, navegar y estudiar una escena neuronal de forma cooperativa, facilitando que investigadores geográficamente distantes puedan analizar la misma escena en tiempo real. Utilizando como apoyo el estudio del motor de juegos UDK 4 como herramienta de visualización científica, el cual desarrolla un prototipo capaz de representar modelos de neuronas.

Asimismo incluye un sistema de persistencia gestionado a través de un servicio web RESTful que permite almacenar los datos generados de dichas colaboraciones, tales como anotaciones sobre los datos bajo análisis, chats entre los usuarios, logs de las sesiones de usuario y reproducción de los análisis realizados por otros usuarios.

Y finalmente un servicio web para la recuperación de datos procedentes del sistema de persistencia a través de una interfaz accesible desde el navegador.

Con este objetivo de desarrollo se realiza, en primer lugar, un análisis sobre las técnicas de visualización colaborativa y métodos de comunicación aptos para su aplicación dentro de la iniciativa HBP. En segundo lugar, se realizará un estudio sobre el entorno de desarrollo y el prototipo existente para comprender su funcionamiento. Por último y con la información obtenida de desarrollar la infraestructura colaborativa.

2. Estado del Arte

2.1. Visualización

La visualización científica se ha convertido en una herramienta imprescindible para el manejo y distribución de la información en la actualidad. Diariamente se generan datos en el orden de Terabytes, en consecuencia para su análisis las técnicas de visualización son adecuadas aumentando las capacidades humanas en lugar de reemplazar a las personas con métodos computacionales de toma de decisiones, ya que de otra forma sería imposible para las personas estudiar tal cantidad de datos.^[4]

Visualizar en términos científicos es transformar y representar una serie de datos junto con una serie de conceptos abstractos, en imágenes o un conjunto de ellas.

La búsqueda de información valiosa entre tal cantidad de datos es realmente compleja con las técnicas comunes, por ejemplo la representación de los datos en texto, donde solo es posible visualizar pequeñas cantidades de información, eliminando la posibilidad de analizar correctamente la mayor parte de información trascendente que se ha generado. Mediante la visualización se facilita el entendimiento de las características de los datos, ayudándonos a la comprensión de estos, de una manera mucho más eficiente desde el punto de vista de esfuerzo y tiempo.

Es importante diferenciar entre visualización científica y gráficos de presentación. Los gráficos de presentación se ocupan principalmente de la comunicación de la información y los resultados en formas que se entienden fácilmente. En la visualización científica, se busca entender los datos. Sin embargo, a menudo los dos métodos están entrelazados.^[5]

A través de la disponibilidad de equipos computacionales cada vez más potentes con cantidades cada vez mayores de memoria interna y externa, es posible investigar dinámicas increíblemente complejas mediante simulaciones cada vez más realistas. Sin embargo, esto trae consigo grandes cantidades de datos. Para analizar estos datos es imprescindible contar con herramientas de software que puedan visualizar estos conjuntos de datos multidimensionales. Para simulaciones complicadas y dependientes del tiempo, la ejecución de la simulación puede implicar el cálculo de muchos pasos de tiempo, lo que requiere una cantidad sustancial de tiempo de CPU y los recursos de memoria son aún limitados, por lo que no se pueden guardar los resultados de cada paso de tiempo. Por lo tanto, será necesario visualizar y almacenar los resultados de forma selectiva en tiempo real para que no tengamos que recalcular la dinámica si queremos ver la misma escena de nuevo.^{[5][6]}

El proyecto Human Brain Project es un ejemplo de aplicación en el campo de la visualización científica. Un supercomputador genera un gran volumen de datos imposible de almacenar, relacionados con el cerebro y su funcionamiento, en consecuencia, sólo queda la opción de analizar visualmente representaciones gráficas, y en tiempo real.

2.2. Técnicas de visualización colaborativa

La colaboración es uno de los grandes desafíos de la visualización y la analítica visual por una buena razón: los problemas que enfrentan los analistas en el mundo real son cada vez más grandes y complejos, con gran incertidumbre, mal definidos y ampliamente delimitados. Ya no es factible para un único analista abordar los inmensos conjuntos de datos que se generan gracias a los avances tecnológicos, a menudo se requiere una amplia experiencia, diversas perspectivas y un número de personas dedicadas a su resolución.

La visualización tradicional y las herramientas de análisis visual se diseñan normalmente para que un solo usuario interactúe con una aplicación de visualización. Ampliar estas herramientas para incluir el apoyo a la colaboración es una meta clara hacia el aumento del alcance y la aplicabilidad de la visualización. Con este fin se busca reunir a muchos colaboradores para que cada uno pueda contribuir al objetivo común.

Los colaboradores a menudo no necesitan ser expertos. Los no expertos pueden participar en análisis colaborativos y aprender de los procesos de análisis y puntos de vista de otros en un conjunto de datos.^[7] A partir de este modelo nacen las comunidades virtuales de aprendizaje, como un movimiento hacia concepciones de aprendizaje más socialmente situadas y una visión de la inteligencia distribuida más que como una propiedad de las mentes individuales.^[8]

La colaboración permite también a los usuarios geográficamente separados acceder a un entorno virtual compartido para visualizar y manipular conjuntos de datos para la resolución de problemas sin necesidad de un encuentro físico.^[7]

Cuando grupos de personas comparten el mismo software de visualización interactiva, ya sea en ubicaciones distribuidas o en ubicaciones compartidas, pueden elegir y seleccionar vistas alternativas de los datos para su exploración, análisis, discusión e interpretación. En los entornos distribuidos los resultados suelen intercambiarse a través de chat, comentarios, vídeo/audio, de modo que se puedan discutir y analizar las opiniones y las representaciones alternativas de los datos. Esta discusión también puede ocurrir cara a cara en ubicaciones co-localizadas. El objetivo de un grupo con este nivel de compromiso es poder cubrir y explorar mayor cantidad de diferentes aspectos de los datos, considerar interpretaciones alternativas y discutir los datos en un contexto visual más amplio.^[7]

2.3. El motor de juegos Unreal Engine 4

Mientras que muchos estudios de desarrollo de juegos utilizan su propio motor de videojuegos, todavía queda un enorme mercado para desarrolladores independientes e incluso estudios más grandes que necesitan un gran motor de juego para ayudarles a crear su juego.

Unreal Engine 4 es un conjunto completo de herramientas de desarrollo hechas para que cualquier persona que trabaje con la tecnología en tiempo real. Desde aplicaciones empresariales y experiencias cinematográficas hasta juegos de alta calidad a través de PC, consola, móvil, realidad virtual y realidad aumentada.

Dispone de flujos de trabajo accesibles que capacitan a los desarrolladores para iterar rápidamente ideas y ver resultados inmediatos sin tocar una línea de código, mientras que el acceso completo al código fuente da total libertad de modificar y extender las características del motor.

El lenguaje usado para el desarrollo de las funcionalidades en código fuente es C++, el cual se considera como el mejor lenguaje para la programación de motores de videojuegos ya que está orientado a objetos, tiene control manual completo sobre la memoria y se compila directamente sobre cualquier plataforma. En resumen, se puede exprimir hasta la última gota de rendimiento si sabe cómo. Las capacidades gráficas y de procesamiento son, por tanto, muy elevadas y dispone de diferentes herramientas para trabajar en grupos multidisciplinares (artistas, desarrolladores, producción de video, etc).

Para el desarrollar aplicaciones usando el sistema Unreal Engine, se dispone del SDK UDK4. El UDK4 proporciona una extensa librería de clases en forma de interfaz de programación de aplicaciones o API (del inglés application programming interface). También incluye códigos de ejemplo, notas técnicas de soporte y otra documentación de soporte para ayudar a clarificar ciertos puntos del material de referencia.

Unreal Engine se enfoca en videojuegos y por lo tanto la mayor parte de utilidades proporcionadas por la API son las requeridas por estas aplicaciones, desde escenarios y cámaras, a modos de juego y personajes. Cabe destacar que Unreal Engine es un apoyo para el desarrollo y por lo tanto no limita cualquier otro desarrollo complejo fuera de los límites de Unreal Engine.

Para utilizar esta herramienta en un sector como la visualización científica, será necesario adaptar las estructuras de juego y desarrollar características personalizadas con el fin de poder utilizar todo el potencial de este motor.

2.4. Investigación Anterior

Dentro de la iniciativa Human Brain Project (HBP) se ha realizado un análisis de viabilidad sobre de la utilización del motor de videojuegos Unreal Engine 4 en el diseño e implementación de herramientas de visualización científica, orientadas principalmente hacia esta iniciativa.

Implementando el diseño de un prototipo de pruebas y realizando una valoración del motor de juegos Unreal Engine 4 como herramienta para el desarrollo de técnicas de visualización científica desde distintos puntos de vista, teniendo en cuenta la eficiencia en el tiempo de implementación.

En el proyecto actual, se ha realizado un análisis arquitectural y de los módulos de visualización presentes en el prototipo, de cara a aprovechar lo máximo posible todo lo desarrollado con anterioridad.

Este prototipo se desarrolla en la versión 4.10 de Unreal Engine, por lo que es necesario tener en cuenta la retrocompatibilidad entre versiones del motor. Tiene implementadas la capacidad de cargar mallas neuronales y representarlas de forma individual o en forma de árbol neuronal, como se aprecia en la Figura 1 y Figura 2, usando un sistema de pivotación para visualizar desde todos los ángulos, manteniendo la neurona o el árbol en el centro de la pantalla y pivotando sobre el centro como si los elementos se encontraran en el interior de una esfera.

Entre sus características dispone además de diferentes test de sobrecarga para comprobar cuáles son los límites del motor y hardware donde se ejecuta, los cuales son útiles para el estudio y entendimiento del motor pero no aplicables en la herramienta final a desarrollar en este proyecto.

Con el objetivo de aprovechar la máxima cantidad posible de trabajo realizado en el prototipo y teniendo en cuenta que el proyecto actual se desarrolla sobre la versión 4.15.3 de Unreal Engine, sólo será posible reutilizar el código de carga de mallas neuronales y árboles neuronales, dada la mala retrocompatibilidad entre versiones de este software y la necesidad de crear un entorno colaborativo.

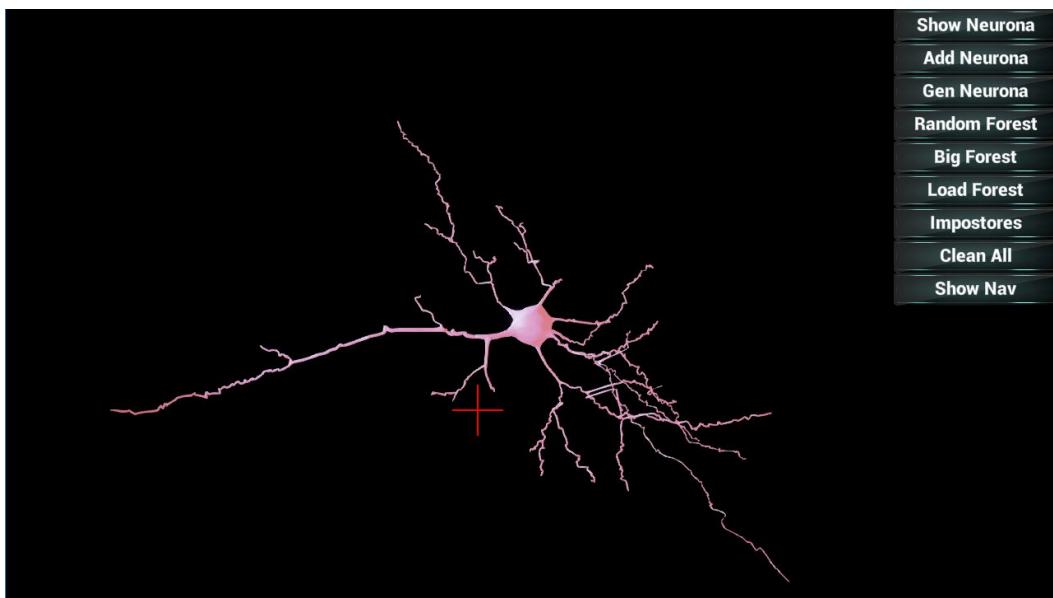


Figura 1: Malla de neurona representada

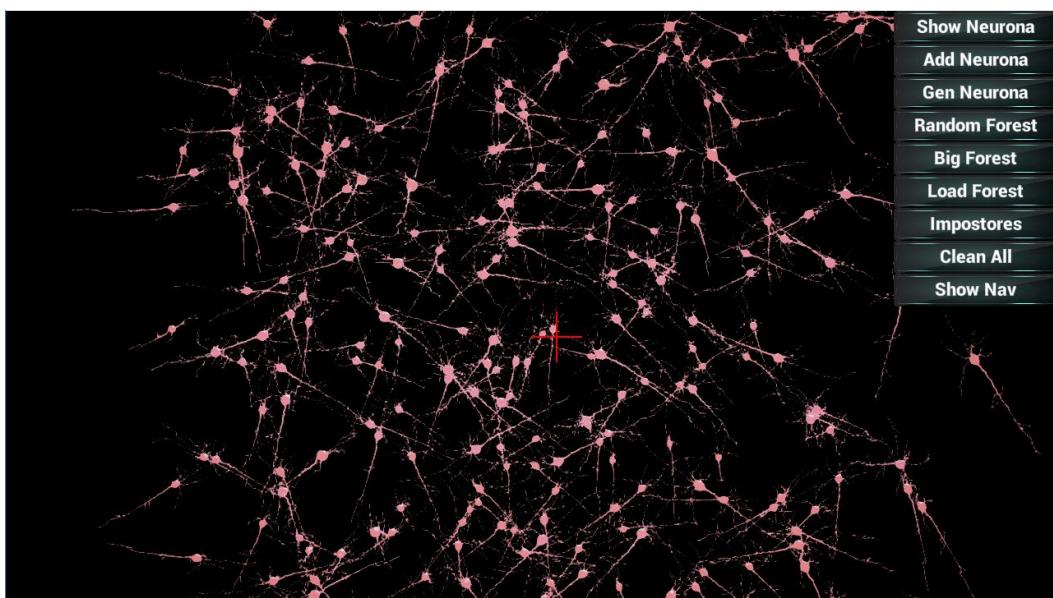


Figura 2: Pivoteación sobre árbol neuronal

No es posible mantener el sistema de visualización del prototipo y es necesario desarrollar uno que se adapte a las necesidades de este proyecto por lo mencionado anteriormente.

Las características anteriores serán optimizadas y modificadas para adaptarlas a la versión actual del motor y mejorar las capacidades de visualización en la medida de lo posible.

2.5. Infraestructura adicional

De forma suplementaria pero crítica para el desarrollo del proyecto es necesario realizar un análisis de los sistemas disponibles para la gestión de la información generada por una herramienta de visualización en su ámbito colaborativo.

Recientemente, el término análisis de datos sociales se ha acuñado para describir la interacción social que es una parte central de la visualización colaborativa: "El análisis de datos sociales es una versión del análisis exploratorio de datos que se basa en la interacción social como fuente de inspiración y motivación".^[7] Este término enfatiza la posibilidad de interacciones humanas tales como discusiones, negociaciones o argumentos alrededor de las visualizaciones como factores impulsores de la exploración de datos.

En la mayoría de los casos, queremos averiguar las relaciones entre estos datos sociales y otro evento, o queremos obtener resultados interesantes de los análisis de datos sociales para predecir algunos eventos. Para lograr estos objetivos normalmente podríamos usar métodos estadísticos, métodos de aprendizaje automático o métodos de minería de datos para realizar los análisis.

Considerando la importancia de esta información es necesario disponer de un sistema de persistencia para almacenar estos datos, distribuirlos y poder realizar un análisis posterior.

Es necesario establecer un método de comunicación entre las diferentes herramientas que generan e interpretan datos y el sistema de persistencia en sí. Teniendo en cuenta el aspecto colaborativo de la infraestructura es necesario asegurar que aunque los datos cambien de lugar o formato, las herramientas de análisis no se vean afectadas en absoluto. Ello permite también a otros desarrolladores poder comunicar sus aplicaciones con el sistema de persistencia y acceder a los datos de forma sencilla y rápida.

2.5.1. Evaluación sobre el uso de BBDD SQL o NoSQL

Los datos que se almacenarán en el sistema de persistencia son datos en texto muy estructurados procedentes de chats, comentarios, descripciones, información de usuarios, etc y ficheros de audio y video.

Una base de datos NoSQL nos proporciona un modelo de datos de estructura muy flexible y no relacional, o lo que es lo mismo, permite almacenar datos en cualquier registro que no tenga ningún esquema fijo. En un entorno distribuido, utilizamos NoSQL sin ninguna inconsistencia, si existen fallos en cualquier máquina, no se interrumpirá ningún trabajo. NoSQL es escalable horizontalmente conduciendo a un alto rendimiento de forma lineal.^[9] Visto de otra manera, son bases de datos orientadas a documentos. En lugar de almacenar sus datos en tablas hechas de filas individuales, como una base de datos relacional, almacena sus datos en colecciones hechas de documentos individuales. Documentos JSON sin ningún formato o esquema particular. Es básicamente una enorme estructura de datos organizados como un fractal.

Por otro lado una base de datos SQL tradicional, conlleva una organización relacional y sigue el concepto ACID (Atomicity, Consistency, Isolation and Durability) por tanto los datos deben cumplir requisitos de integridad tanto en tipo de dato como en compatibilidad. La operación se hace entera y si falla se utiliza la técnica de rollback.^[10] Su uso mayoritario y debido al largo tiempo que llevan en el mercado, estas herramientas tienen un mayor soporte y mejores suites de productos y add-ons para gestionar estas bases de datos.

Explicado de forma menos técnica, las bases de datos SQL se asemejan a la transmisión automática en los vehículos, y las NoSQL, a la manual. Con NoSQL, el usuario es el responsable de organizar todo el trabajo mientras que en SQL, el sistema se encargaría de forma automática. Asimismo, las bases de datos NoSQL permiten ganar más rendimiento del sistema mediante la eliminación de una gran cantidad de comprobaciones de integridad. De forma similar al empleo de la transmisión manual, donde el usuario puede sacar más rendimiento del vehículo.

Los datos que generará la aplicación serán estructurados. La flexibilidad de esquema en NoSQL es una gran idea, pero sólo será realmente útil cuando la estructura de sus datos no tiene ningún valor. Los datos generados se podrán usar en análisis posteriores por lo que todos los sistemas de integridad de SQL serán necesarios, y por tanto para este proyecto se puede sacar una mayor ventaja haciendo uso de SQL frente a NoSQL, teniendo en cuenta también la rapidez de puesta en marcha del sistema y su facilidad de uso.

Reforzando la posición de elegir SQL, Google Adwords se implementa con MySQL, un ejemplo de Big Data a gran nivel de código crítico, donde se ha visto que las ventajas de rendimiento no eran suficientes como para perder integridad de datos.^[11]

2.5.2. Servicio Web RESTful cómo Middleware

El middleware de servicios Web desempeña el papel de "intermediario" en la arquitectura general del software. La capa de middleware puede abordar cuestiones como la seguridad o las comunicaciones entre plataformas. También puede permitir la mensajería entre diferentes componentes de software.

La lógica de usar un servicio web como middleware recae sobre la necesidad de una interfaz uniforme que no se vea afectada por la modificación de las estructuras de datos y asu vez proporcione estos recursos de forma sencilla.

Con REST, el sistema de persistencia obtiene total libertad de cambiar los recursos expuestos a voluntad. No hay una API fija por encima y más allá de lo que define REST, por lo cual, el cliente sólo necesita conocer una URI inicial y, posteriormente elegir entre las opciones suministradas por el servidor para navegar o realizar acciones.

Este sistema contrasta con los distintos esquemas de llamada de procedimiento remoto (RPC), en los que el cliente y el servidor deben ponerse de acuerdo sobre un protocolo detallado que normalmente necesita ser compilado en ambos extremos. Este enfoque no es óptimo, ya que cualquier cambio debe implementarse en ambos lados, servidor y cliente al mismo tiempo y se pierde en gran medida la escalabilidad.

La separación de las preocupaciones es el principio detrás de las restricciones cliente-servidor en un servicio REST. Al separar las preocupaciones de la interfaz de usuario de los problemas de almacenamiento de datos, mejoramos la portabilidad de la interfaz de usuario a través de múltiples plataformas y mejoramos la escalabilidad simplificando los componentes del servidor. Sin embargo, lo más significativo para la Web es que la separación permite que los componentes evolucionen de forma independiente, apoyando así el requisito de escala de Internet de múltiples dominios de organización.^[12]

Por tanto se utilizarán estos principios, aprovechando que se trata de un protocolo cliente/servidor sin estado, cada mensaje HTTP contiene toda la información necesaria para comprender la petición, las operaciones están bien definidas y se aplican a todos los recursos de información y se usa una sintaxis universal donde cada recurso es direccionable únicamente a través de su URI. Los sistemas que siguen los principios REST se llaman con frecuencia RESTful.

3. Diseño de la Herramienta

Una herramienta de visualización científica cooperativa debe acotarse a un área de trabajo dentro del gran número de áreas que tiene la ciencia de forma que sea posible determinar el tipo de datos se van a representar visualmente.

En este ámbito, se analizarán los datos obtenidos del proyecto Human Brain Project en consecuencia el área de trabajo queda acotada a la neurociencia.

Human Brain Project es un proyecto que pretende acercar el funcionamiento del cerebro humano mediante el desarrollo de un simulador que lo asemeje, intentando adaptarse lo máximo posible a la realidad y brindar a los científicos la capacidad de emular sus funcionalidades, estudiar su comportamiento o incluso estudiar el desarrollo de enfermedades del cerebro para poder así mejorar su diagnóstico y posibles tratamientos de las mismas.

Dentro de las grandes dimensiones de este área, el proyecto actual se centra en el diseño de una herramienta de visualización para el análisis de la morfología de las neuronas y navegación a través de diversas escenas neuronales de manera colaborativa, de tal modo que distintos neurocientíficos sin importar su localización puedan analizar la misma escena neuronal y su funcionamiento conjuntamente.

La arquitectura del software de la herramienta de visualización científica colaborativa buscará ofrecer una serie de funcionalidades que permitan a la herramienta usarse en un entorno de colaboración distribuido y que ofrezca a los usuarios un sistema de comunicación haciendo uso de las tecnologías de Unreal Engine y cumpliendo con una serie de requisitos esenciales.

3.1. Especificación de Requisitos

Previo a listar la especificación de requisitos se tiene en cuenta la facilidad de uso. Es probable que muchos de los usuarios que utilicen la herramienta no tengan conocimientos de computación, es por ello que se ha de facilitar su uso de manera completamente interactiva sin tener que recurrir a técnicas más complicadas para su control:

- Usabilidad, de forma que la herramienta desarrolle una interfaz sencilla, intuitiva y rápida de manejar, permitiendo a su vez abarcar todas las funcionalidades a implementar en la herramienta.

- Colaboración, teniendo en cuenta que muchas de las escenas generadas son demasiado grandes para ser analizadas por una única persona siendo usual que varios científicos operen sobre los mismos resultados, extrayendo información relevante y realizando puestas en común a través de distintos comentarios. Asimismo, surge la necesidad de interconectar usuarios permitiéndoles interactuar en la misma escena a través de la red.
- Escalabilidad de la herramienta, ya que es necesario que se pueda ampliar en un futuro con nuevas funcionalidades.

El conjunto de requisitos tanto funcionales como no funcionales están divididos en cuatro módulos, la herramienta de visualización colaborativa, el sistema de persistencia y como apoyo, un módulo para servicio web RESTful dedicado al sistema de persistencia y un módulo final como servicio web dedicado a visualizar los datos extraídos por la herramienta de visualización colaborativa, de forma que no sea necesario ejecutar la aplicación y poder acceder a los datos con mayor rapidez y desde cualquier lugar.

En la Figura 3 se muestra la relación que existe entre los módulos sobre los que se especifican los diferentes requisitos.

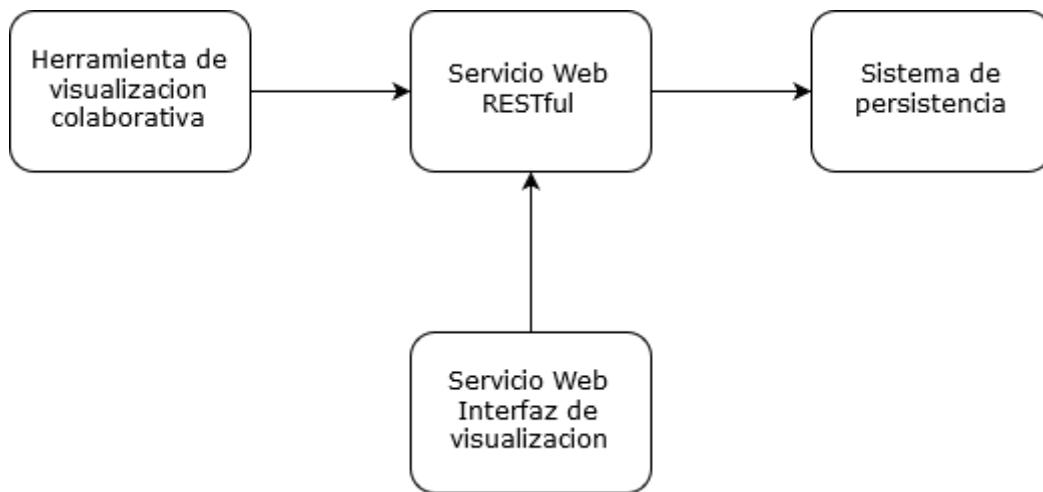


Figura 3: Diagrama de relaciones entre módulos

3.1.1. Especificación de requisitos en la herramienta de visualización colaborativa

1. **Acceso mediante identificación de usuarios y registro de los mismos**, en los que se pidan los datos necesarios con el fin de identificarlos de manera inequívoca. Con el registro se creará un nuevo usuario con su correspondiente usuario y contraseña. No puede existir más de un usuario con el mismo identificador y la herramienta nunca almacenará la contraseña de los usuarios.
2. **Acceso de usuarios en modo invitado**, en el que no es necesario introducir datos para obtener acceso a la aplicación. Esto fomenta la aplicación didáctica de la herramienta.
3. **Representar resultados de simulaciones masivas en 3D**. Se dispondrá de un mecanismo que permita cargar distintas escenas neuronales y crear escenas sintetizadas automáticamente.
4. **Representar modelos de estructuras microelectrónicas**. Se emulará el funcionamiento microelectrónico de las neuronas presentes en la escena a analizar.
5. **Sesión de análisis local**, donde un usuario puede visualizar una escena neuronal de forma individual. Este tipo de sesión puede comunicarse con el sistema de persistencia para almacenar la información que genere el usuario. Opción disponible solo para usuarios registrados.
6. **Creación de una sesión de análisis colaborativo**, un usuario registrado tendrá la capacidad de crear una sesión en red, con el fin de que otros usuarios puedan unirse al análisis, compartiendo la misma escena neuronal y percibiendo de forma sincronizada. Podrá especificar un nombre identificativo de session, el cual servirá a otros usuarios para reconocer de qué sesión se trata en concreto, también podrá especificar el límite de usuarios que tendrán acceso simultáneo a la escena en análisis. Se podrá seleccionar si la sesión se crea en red local o vía internet, permitiendo rebajar la carga de ancho de banda en sesiones en las cuales no sea necesario conectarse desde distintos puntos topográficos.
7. **Búsqueda de sesiones de análisis colaborativas**. Se podrán realizar búsquedas de sesiones activas en el sistema de forma automática por parte de tanto usuarios registrados como usuarios invitados, permitiendo listar todas las sesiones activas sin necesidad de que el usuario necesite introducir parámetros.

8. **Chat en tiempo real.** Los usuarios podrán enviarse mensajes a través de un sistema de chat grupal. Estos mensajes llegarán al resto de usuarios de forma inmediata y se almacenarán en el sistema de persistencia. Si un usuario se une a una sesión activa, todo el chat almacenado en persistencia será recuperado de forma que esté usuario pueda ver toda la conversación anterior a su llegada. Un usuario invitado podrá ver toda la conversación pero no escribir ella, siendo este sistema persistente. Se utilizará el estándar @nombredeusuario, como método para enviar mensajes privado a uno o más usuarios a la vez, de forma que si un usuario menciona a otro con este método dentro de un mensaje, el mensaje sólo será recibido por el usuario o los usuarios mencionados.
9. **Sistema de VoIP.** Todo usuario que se una a una sesión colaborativa podrá comunicarse por medio de voz. El usuario deberá disponer de altavoces y micrófono para usar esta característica. Este sistema está disponible tanto para usuarios registrados como invitados, dando la capacidad a estos últimos de interactuar con el resto de usuarios. Por tanto permitirá realizar comentarios de forma fluida y agilizar el desarrollo del análisis. Además este sistema permitirá al usuario poder comunicarse si no dispone de teclado o utiliza otro periférico como un mando para navegar por la escena neuronal.
10. **Navegación dentro de la escena,** dando la capacidad de movimiento total en la escena 3D y de movimiento de la cámara del usuario activo. Los usuarios estarán representados por una esfera y un identificador de usuario flotante encima de ésta. El sistema deberá ser capaz de reconocer diferentes entradas desde el teclado, un mando de consola o un equipo de realidad virtual. El movimiento de los usuarios se replicará de forma sincronizada al resto de usuarios permitiendo así que unos usuarios se encuentren a otros y se simplifique la labor de análisis de una neurona en concreto facilitando su localización
11. **Sistema You See What I See,** es decir, un usuario podrá visualizar lo que está viendo otro usuario de forma sincronizada, fomentando la formación didáctica haciendo uso de la herramienta.
12. **Selección de neuronas,** permitiendo marcar diferentes neuronas y replicar esta selección al resto de usuarios dentro de la sesión de forma sincronizada. El sistema ofrecerá al usuario la opción de ocultar neuronas según éstas estén seleccionadas o no. Las neuronas que se oculten lo harán para todos los usuarios dentro de la sesión, con el fin de poder analizar de forma focalizada una selección o el resto no seleccionado.

- 13. Información localizada dentro de la neurona.** Es necesario que el sistema reconozca el lugar donde un usuario quiera introducir información en el cuerpo de una neurona. Esta información se replicará a todos los usuarios. Debe marcarse de forma visual la zona con información relevante y que la marca se sincronice con el resto de usuarios de la sesión. Toda información introducida será almacenada en el sistema de persistencia.
- 14. Captura de imágenes en alta resolución y 360 grados.** El usuario podrá realizar una captura desde su posición en la escena en 360 grados y en alta resolución (4K). El sistema almacenará estas imágenes en el sistema de persistencia.
- 15. Interfaz de usuario,** que permita interactuar con la herramienta y acceder a todas sus funciones directamente. Abarca sistema de inicio y registro de usuario el menú de sesiones donde el usuario podrá elegir entre sesión local, colaborativa o buscar una sesión activa según los privilegios que tenga el usuario en cuestión, y dentro de la escena neuronal diferentes menús para introducir información neuronal o poder ocultar neuronas, entre otras opciones, de forma que el usuario pueda interactuar con todas las características que tiene disponible la herramienta de forma intuitiva.
- 16. Repetición de los pasos seguidos.** Un sistema de replay que permita a todos los usuarios repetir una secuencia de pasos ya realizada con anterioridad de forma sincronizada. El usuario que active este sistema obtendrá el control de la repetición y por tanto podrá moverse hacia delante y detrás en el tiempo, pausarlo y pararlo.
- 17. Realidad Virtual.** La herramienta deberá soportar el uso de equipos de realidad virtual, realizando los cambios necesarios en la configuración de forma automática si reconoce un equipo instalado y activo en el sistema.

3.1.2. Especificación de requisitos para el sistema de persistencia

- 1. Almacén de información de usuarios registrados.** Conjunto de datos necesarios para identificar usuarios de manera inequívoca.
- 2. Registro de mensajes,** procedentes del chat dedicado a la comunicación dentro de la herramienta de visualización colaborativa.
- 3. Directorio de imágenes y videos,** almacenados de forma local en el sistema de persistencia, de forma que la base de datos implementada no se vea sobrecargada por la enorme cantidad de datos que generan este tipo de ficheros.

3.1.3. Especificación de requisitos para el servicio web dedicado al sistema de persistencia

1. **Amigable para el desarrollador** y explorable a través de una barra de direcciones del navegador. La API debe ser simple, intuitiva y consistente para hacer la adopción no sólo fácil sino agradable.
2. **Los recursos individuales se identifican en las solicitudes**, utilizando URLs en sistemas REST basados en la Web. Los propios recursos están conceptualmente separados de las representaciones que se devuelven al cliente. El servidor debe enviar datos de su base de datos en formato JSON, el cual no es una representación interna del servidor.
3. **Cada mensaje incluye suficiente información para describir cómo procesar el mensaje**. El formato de la información enviada debe ser especificado por un tipo de medio de Internet (tipo MIME).^[12]

3.1.4. Especificación de requisitos para servicio web de visualización de datos almacenados en el sistema de persistencia

1. **Evitar distorsionar la información**. Los datos extraídos del sistema de persistencia no deberán ser modificados.
2. **Acceso de usuarios mediante identificación y registro**. Estos datos que serán usados con el fin de identificar a los usuarios de manera inequívoca. Con cada registro se creará un nuevo usuario con su correspondiente identificador de usuario y contraseña. No puede existir más de un usuario con el mismo identificador y la herramienta nunca almacenará la contraseña de los usuarios. Estos datos son compartidos con la herramienta de visualización para realizar la misma función.
3. **Implementación de Sandboxing o “cajón de arena”**. Un aislamiento de procesos, es decir, un mecanismo que aísla aplicaciones del resto del sistema dentro de una especie de contenedor virtual desde el cual se controlan los distintos recursos que pertenecen a la aplicación.
4. **Visor de chats antiguos**. Interfaz gráfica capaz de extraer la información generada de las conversaciones de chat dentro de la herramienta de visualización científica.
5. **Visor de imágenes en 360 grados**, capaz de visualizar la captura de una escena guardada como si de la herramienta de visualización científica se tratara.

3.2. Arquitectura del Software del Framework

La arquitectura de software define de manera abstracta los componentes que llevan a cabo la tarea de computación, sus interfaces y la comunicación entre ellos. Esta arquitectura se selecciona y diseña con base en los objetivos (requisitos) y también otros fines como la mantenibilidad, auditabilidad, flexibilidad e interacción con otros sistemas de información.

La arquitectura software de la herramienta estará definida del siguiente modo: los usuarios cliente de la herramienta de análisis se conectan a la aplicación servidor, la cual se establece dentro del cliente que genera la escena para el análisis.

El sistema servidor, por tanto carga una escena y además administra todos los componentes tales como objetos de usuario, neuronas e información, que se encuentran repartidos en el espacio.

Las acciones de cada usuario sobre los elementos desde las herramientas cliente se controlan directamente en el servidor, debido a que no se puede confiar en que los clientes realicen cambios sobre datos relevantes de la aplicación. Por ello, es necesario probar las acciones de los clientes antes de ejecutarlas. Esto aporta al sistema un grado muy alto de seguridad.

Solo la herramienta que actúe como servidor de la escena será la que tenga permitido realizar las tareas de envío de información al sistema de persistencia. El resto de clientes tendrán acceso al sistema de persistencia pero sólo para obtener información, nunca para intentar almacenar o modificar algún tipo de información.

Cada una de las herramientas dispone de una interfaz para interactuar con el entorno, y están capacitadas para renderizar todos los componentes de la visualización en el render del motor gráfico, el cual se encarga de pintar los componentes con la información que recibe del sistema servidor.

Toda comunicación con el sistema de persistencia se hace a través de una API REST que proporciona todos los recursos de forma transparente a los servicios que quieran intercambiar información con este sistema.

Una interfaz web permite a los usuarios extraer información del sistema de persistencia sin necesidad de utilizar la herramienta de visualización para ver estos resultados.

La arquitectura software de la herramienta se puede ver representada gráficamente en la Figura 4.

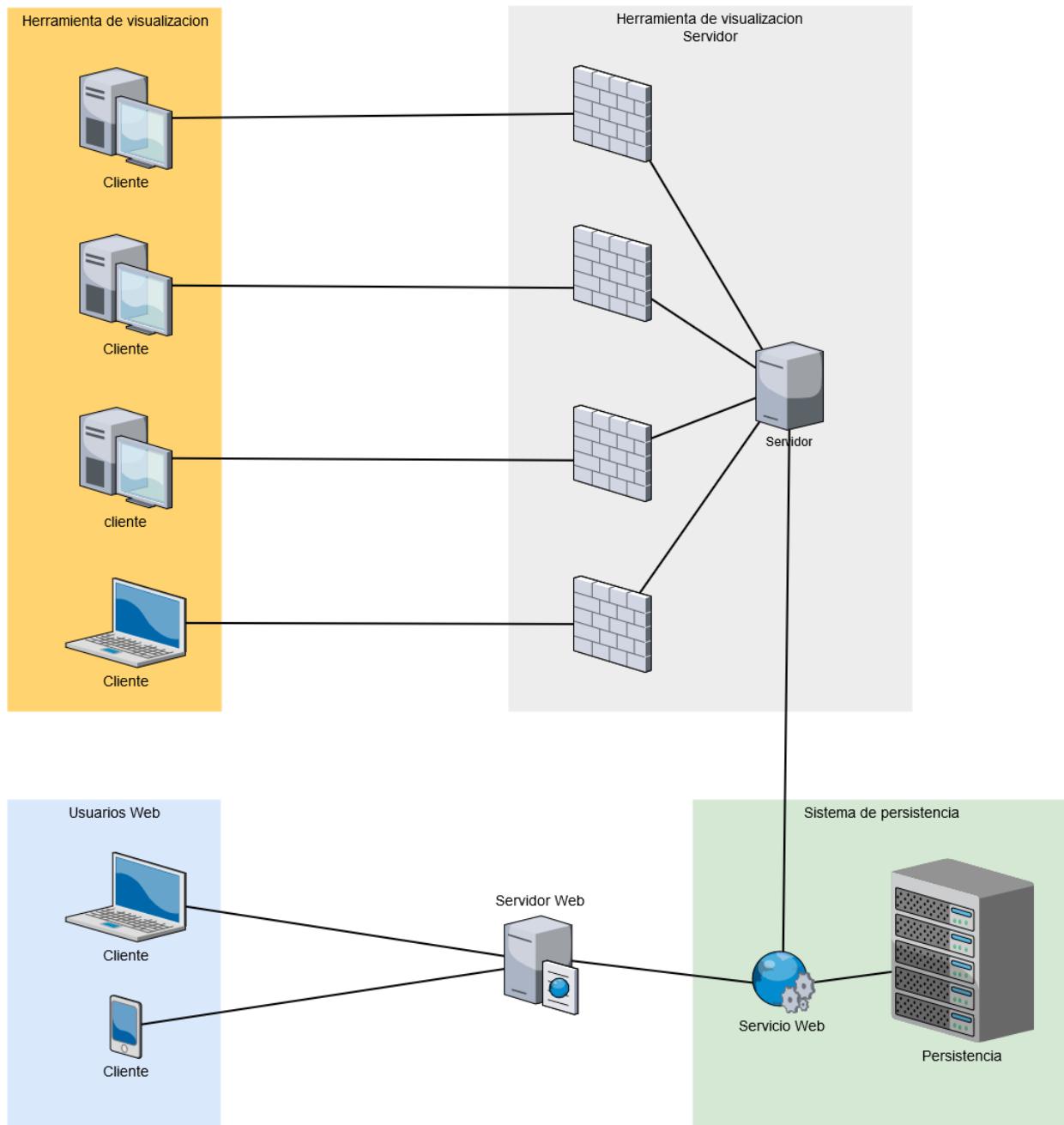


Figura 4: Arquitectura del Software de la Herramienta

El diseño de la interfaz gráfica para la herramienta de visualización colaborativa se puede separar en dos módulos principales, uno relacionado con los menús y el otro relacionado con la interfaz de usuario.

El primer módulo incluye la configuración de opciones, creación, o unión a una sesión de trabajo ya existente, así como la posibilidad de salir de la herramienta. El funcionamiento del menú principal se implementará siguiendo el diagrama de estados de la Figura 5.

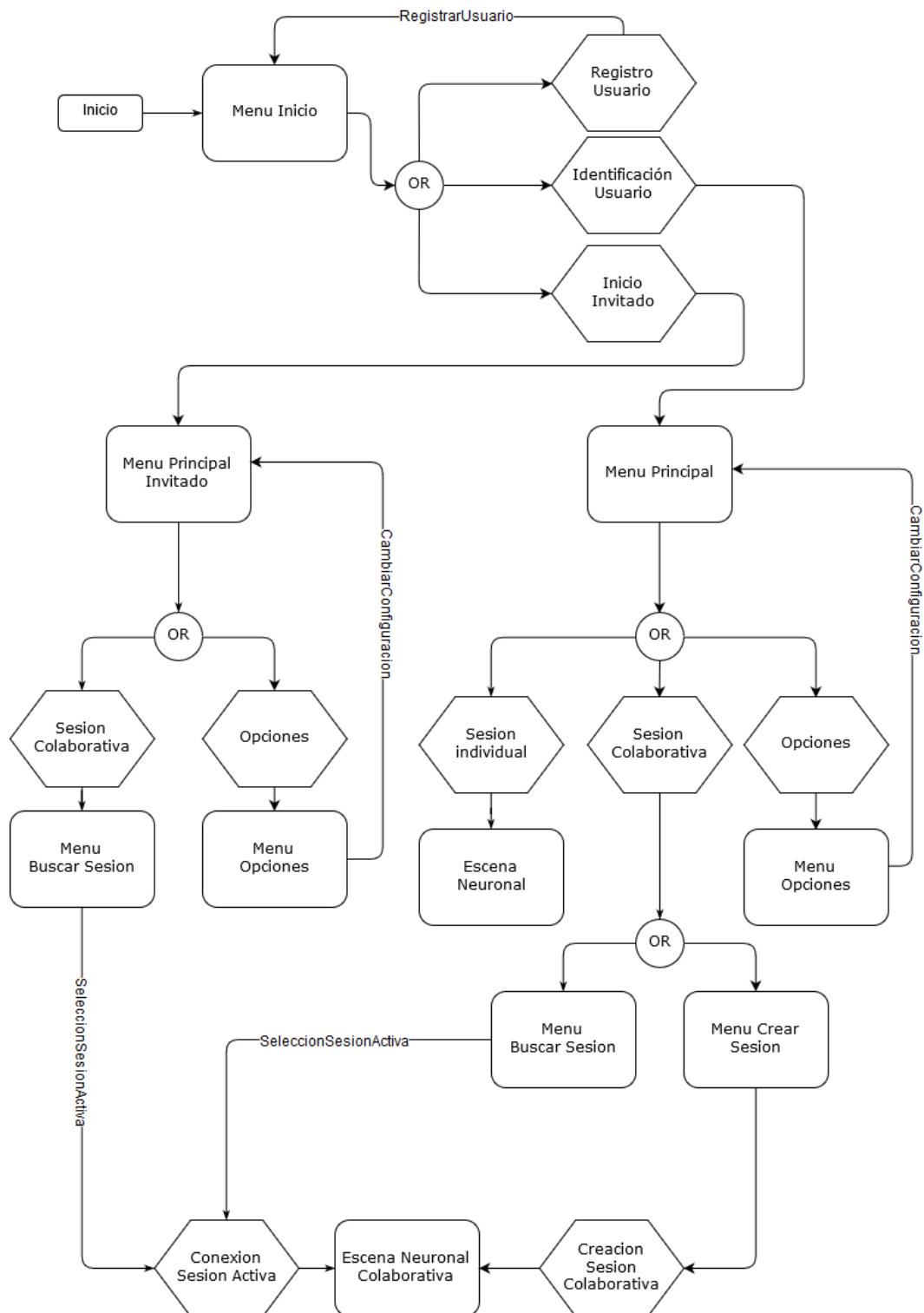


Figura 5: Diagrama de estado del Menú

Un usuario iniciado la aplicación dispondrá de un menú de inicio en el que introducir su identificación de usuario con el fin de acceder a los servicios de la herramienta. El menú de inicio dispondrá de las opciones de registrar un nuevo usuario o acceder como invitado. Si un usuario se registra dentro de la herramienta, volverá automáticamente al inicio.

El usuario identificado accederá al menú principal. Menú mediante el cual puede configurar las opciones de ventana, crear una sesión individual, ir al menú de crear una sesión o realizar la búsqueda automática de sesiones activas y unirse a ellas. En el menú de configuración se podrá seleccionar entre distintas resoluciones de pantalla, mientras que el menú de sesión colaborativa tendrá dos opciones. La primera es poder crear una sesión de trabajo, lo cual convierte a la herramienta en servidora creando un host disponible para el resto de clientes. Y la segunda opción es realizar una búsqueda automática de sesiones activas, donde se podrá acceder a una sesión creada por otro usuario en modo cliente.

Un usuario que inicie sesión como invitado, accederá al menú principal de invitado. Éste dispone de acceso al menú de configuración y acceso al sistema de búsqueda automática de sesiones activas donde podrá unirse a una sesión ya activa pero no podrá crearla.

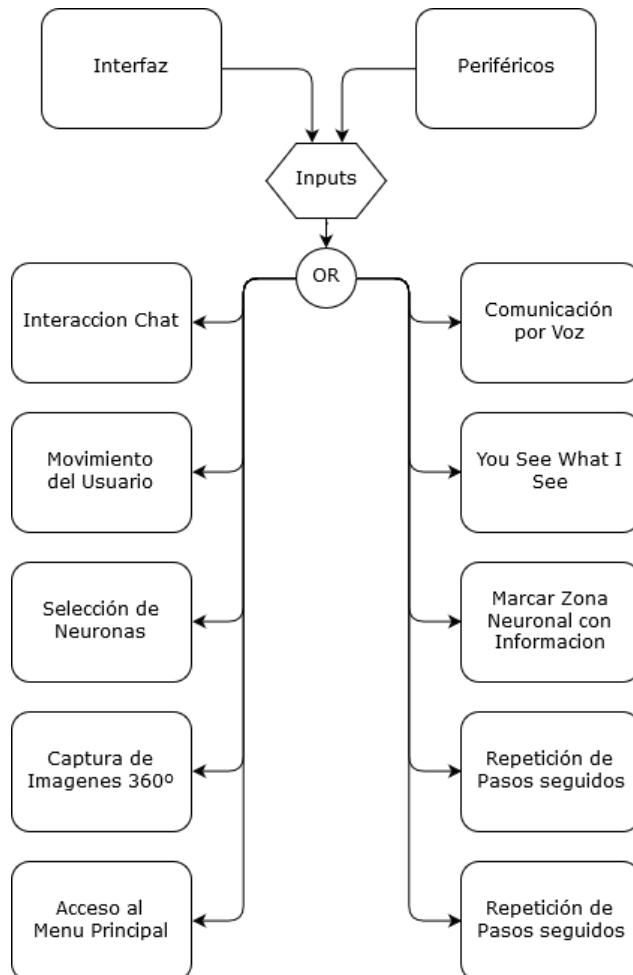


Figura 6: Diagrama de interfaz de usuario

El segundo módulo se corresponde a la interfaz de usuario, a través de la cual un usuario podrá realizar una serie de acciones para interactuar con los datos que representa y poder obtener información más detallada de estos.

En el diagrama de la Figura 6, se reflejan las opciones de interacción que tiene el usuario a través de la interfaz en conjunto a inputs generados por los periféricos conectados, dando al usuario acceso a todas las características de la herramienta y, por tanto, una mejor experiencia a la hora de evaluar los resultados.

Por último, el diseño de la interfaz de la aplicación web para la visualizar la información contenida en el sistema de persistencia dispondrá de una pantalla de identificación o registro de usuario, la cual controlará el acceso de los usuarios a un sandbox o “caja de arena” que contendrá las aplicaciones de visor de chat y galería de imágenes en 360 grados. El visor de chat dispondrá de un histórico de chats por sesiones y un buscador por mensajes. La galería de imágenes en 360 grados presentará todas las imágenes que estén presentes en el sistema de persistencia de forma organizada. Este diseño se puede comprobar en el diagrama de la Figura 7.

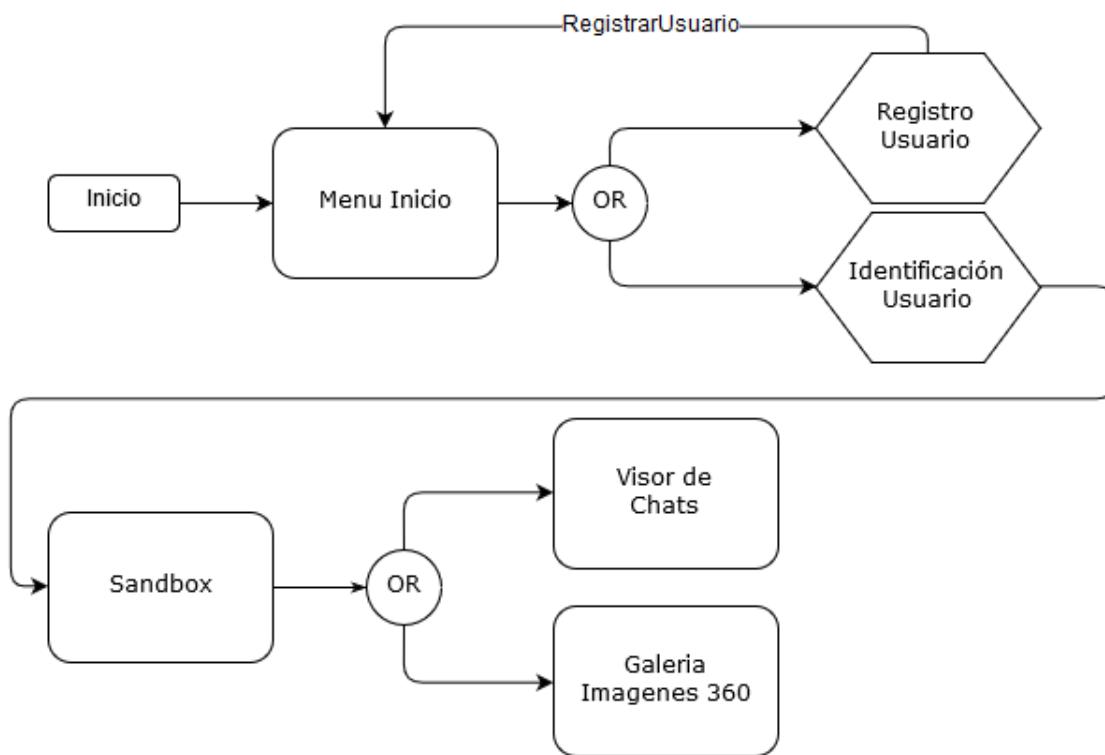


Figura 7: Diagrama de interfaz web

4. Implementación de la herramienta de visualización colaborativa

La implementación de los requisitos presentes en este proyecto será dividida en varias secciones ya que la herramienta se compone de diferentes módulos.

El primer módulo, la aplicación de visualización científica colaborativa, se desarrolla en el lenguaje C++, utilizando diferentes características del motor Unreal Engine, y la interfaz es basada en nodos para crear elementos en la elaboración de los diferentes menús de esta aplicación.

Es necesario crear un proyecto desde el sistema de Unreal Engine, el cual genera un código base en C++ con acceso a todo el motor y código de auto-compilación para facilitar la integración del motor al desarrollador.

Sobre este código base se agregan las clases que forman toda la aplicación y que heredarán distintas clases del motor si fuera necesario. Las nuevas clases se pueden crear desde el editor y el motor generará los ficheros de cabecera (.h) y ficheros de fuente (.cpp) para cada clase derivada, automatizando el proceso. También se podrán incluir estos ficheros directamente en el directorio "src" donde se encuentra todo el código de la aplicación si son creados por el desarrollador. El proceso automático de creación de nuevas clases abre una ventana asistente de creación de clase, permitiendo elegir al usuario la clase padre del nuevo objeto y su visibilidad pública, privada o protegida, añadiendo esta visibilidad a la nueva clase tanto al directorio de desarrollo "src" como dentro del editor permitiendo automatizar los procesos de creación de nuevos objetos que hereden de la clase padre implementada por el desarrollador.

La implementación de la herramienta se realiza de forma escalonada, haciendo un estudio detallado de cada sección y creando pequeños prototipos que se centran en el uso de características concretas de la herramienta de análisis. De esta forma se facilita la comprensión del motor permitiendo tomar ciertas decisiones que faciliten la implementación de los requisitos dentro de la aplicación de visualización colaborativa.

Es necesario tener en cuenta que tanto el código base como las clases heredables del motor se enfocan en el desarrollo de un sistema de juego, por tanto es necesario modelar estas herramientas que se proporcionan de manera que en el resultado final los usuarios que trabajen con la aplicación no descubran que debajo de todo se esconden mecanismos de juego para realizar análisis sobre escenas neurocientíficas.

4.1. Arquitectura del Motor

Las clases principales que proporciona el motor son las siguientes:

- Actor: Los actores son instancias de clases que derivan de esta interfaz. Es la clase base de todos los objetos de juego que se pueden colocar en el mundo 3D.^[13]
- UObject: Los objetos son instancias de clases que heredan de la clase UObject. La clase base de todos los objetos en Unreal Engine, incluidos los actores. Por ello, todas las instancias en Unreal Engine son objetos.^[14]

El término actores se utiliza comúnmente para referirse a instancias de clases que derivan de Actor, mientras que el término objetos se utiliza para referirse a instancias de clases que no heredan de la clase Actor.

En general, los actores pueden ser considerados como elementos o entidades completos mientras que los objetos son partes más especializadas. Los actores pueden hacer uso de componentes, que son objetos especializados para definir ciertos aspectos de su funcionalidad o mantener valores para una colección de propiedades.

- Pawn: Se trata de un actor que puede ser "agente" dentro del mundo 3D. Un pawn puede ser poseído por un controlador. Además, se configuran para aceptar fácilmente inputs del usuario y pueden incluir funcionalidades dedicadas al jugador.^[15]
- Character: Un Character hereda de la clase Pawn y tiene un estilo humanoide, que implementa un CapsuleComponent para colisión y un CharacterMovementComponent de forma predeterminada. Puede hacer movimientos humanos básicos, puede replicar el movimiento sin problemas a través de la red y tiene funcionalidades relacionadas con la animación.^[16]
- Controller: Actor responsable de dirigir un pawn. Un controlador puede "poseer" un pawn para tomar el control de él.
- PlayerController: Se trata de la interfaz entre el Pawn y el jugador que lo controla. Hereda de la clase Controller y toda funcionalidad realizada por el jugador debe desarrollarse en el PlayerController. Es posible manejar todos inputs en el Pawn, especialmente para casos menos complejos. Sin embargo, es una buena práctica desarrollarlo en esta clase, ya que permite cubrir necesidades más complejas, como múltiples jugadores en un cliente de juego o la capacidad de cambiar caracteres dinámicamente en tiempo de ejecución. En este caso, el PlayerController decide qué hacer y, a continuación, emite los comandos al Pawn.^[17]

- PlayerState: Interfaz que representa el estado de un jugador o un bot que está simulando a un jugador. Los datos que almacena está clase incluyen el nombre del jugador, la puntuación, el nivel de juego, o cualquier información relevante que sea necesaria distribuir entre todos los jugadores ya que puede replicarse libremente para mantener los datos sincronizados.^[18]
- AIController: Un AIController es una simulación de "voluntad" que puede controlar un pawn. Al igual que PlayerController, hereda de la clase Controller.
- GameMode: Incluso el juego más abierto tiene una base de reglas, y estas reglas forman un modo de juego.^[19] En el nivel más básico, estas reglas incluyen:
 - El número de jugadores y espectadores presentes, así como el número máximo de jugadores y espectadores permitidos.
 - Cómo entran los jugadores en el juego. Puede incluir reglas para seleccionar lugares de comienzo y otros comportamientos de inicio de jugador.
 - Si el juego se puede pausar, y cómo detener el juego.
 - Transiciones entre niveles.
- GameState: Todo acontecimiento relacionado con las reglas definidas en el juego que necesita ser compartido con todos los jugadores y se almacena y sincroniza a través del Estado del juego.^[20] Esta información incluye:
 - Cuánto tiempo ha estado funcionando el juego (incluyendo el tiempo de ejecución antes de que el jugador local se uniera).
 - El momento en el que cada jugador individual se unió al juego, y el estado actual de ese jugador.
 - La clase base del modo de juego actual.
 - Ya sea que el juego haya comenzado o no.
- PlayerCameraManager: Administra cómo se comporta la cámara asociada a un jugador. Cada PlayerController normalmente implementa esta clase como parte de sus componentes.^[21]
- HUD: La clase HUD “Head Up Display” o visualización en pantalla 2D es la clase que permite presentar información directamente en pantalla sin interferir con ningún elemento del entorno. Se trata de una capa que se coloca en el primer plano, como si se tratase de una transparencia colocada entre la cámara y el mundo 3D. Cada jugador tiene un HUD independiente.^[22]

El diagrama de flujo de la Figura 8 ilustra cómo se relacionan entre sí estas clases básicas de juego. Un juego se compone de un GameMode y un GameState. Los jugadores que se unen al juego se asocian a un PlayerController. Estos controladores permiten a los jugadores poseer pawns en el juego para que puedan tener representación física, además también proporciona a los jugadores controles de entrada, un HUD, y un PlayerCameraManager para manejar las vistas de la cámara.

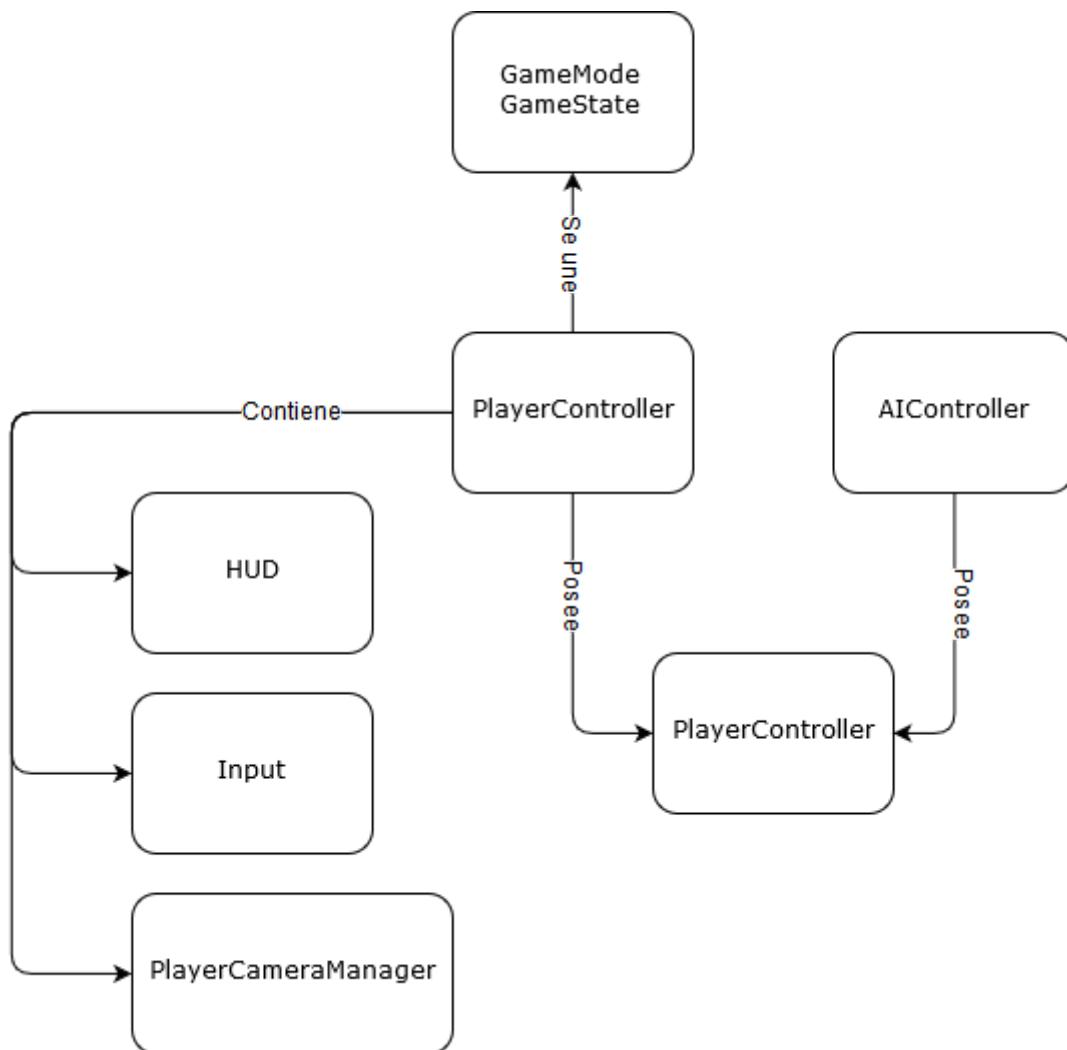


Figura 8: Arquitectura de juego

4.2. Diagrama de Clases

Haciendo uso de la arquitectura propia del motor Unreal Engine se muestra un diseño de alto nivel en la Figura 9 perteneciente a la herramienta de visualización colaborativa.

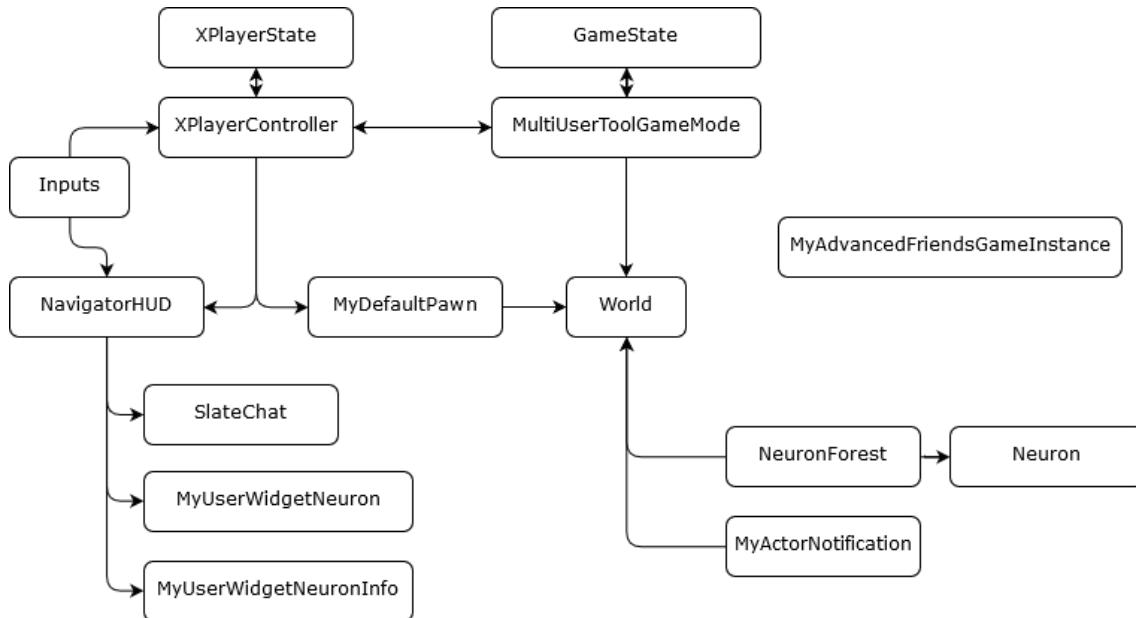


Figura 9: Diseño de alto nivel

Se definen las siguientes clases mostradas en el diagrama de alto nivel de la herramienta:

- **MultiUserToolGameMode**: Heredada de la clase **GameMode**, establece qué controlador utilizan los usuarios al unirse a la sesión y define el Pawn y HUD que se aplica por defecto. Además, en esta clase se administra la carga automática de objetos en el mapa de forma dinámica con la información de una escena neuronal.
- **XPlayerController**: El **PlayerController** existe en el servidor para cada pawn controlado por el usuario y también en la máquina del cliente que controla. Los controladores de otros usuarios no existen en la máquina de un cliente. Por tanto es en esta clase donde se deben implementar gran parte de los mecanismos de comunicación entre clientes y servidor, dado a que el servidor tiene acceso a todos los controladores y, por tanto, acceso a toda la información que estos contengan como posición y movimiento o manejo de los inputs enviados por los clientes.

- XPlayerState: Dada la capacidad de replicación de la clase PlayerState y la sincronización con el servidor de forma automática, es en esta clase en la que se implementan todos los métodos de comunicación a través de mensajes y voz.
- MyDefaultPawn: Clase que hereda de Pawn. Los métodos que implementa la clase Character de base dificultan el funcionamiento correcto de la aplicación dado a que las escenas neurocientíficas se generan como mundos 3D totalmente personalizados y no comprenden variables como el suelo o la gravedad, elementos que están presentes en la clase Character. El pawn que se genera en esta clase está formado por una esfera y un texto flotante encima de ella, que hará referencia al nombre de usuario del usuario que controle el pawn en cuestión. Al ser una clase básica es necesario incorporar distintos componentes dentro del pawn para conseguir el funcionamiento esperado, los cuales se instancian a partir de las siguientes clases:
 - UCameraComponent: Referente a la cámara asociada. Proporciona una visión personalizada al usuario.
 - USpringArmComponent: “Brazo” que se encarga de colocar la cámara en una posición respecto del Navigator (gestiona el seguimiento de la cámara al Pawn).
 - USpotLightComponent: Proporciona un foco de luz asociado a la cámara, el cual permite obtener iluminación direccional hacia el frente de la cámara.
 - UStaticMeshComponent: Malla simple que permite representar al Pawn en la escena, haciéndolo visible al resto de usuarios.
- Neuron: Deriva de la clase Actor. Una neurona es una malla estática generada mediante un programa externo, concretamente Neuronize, e importada como un fichero de extensión .obj. A este fichero se le ha de insertar antes de la definición de las caras el parámetro ‘s 1’ y al finalizar ‘s off’ con el objetivo de que UE4 suavice las neuronas al importar su malla, evitando que la superficie de ésta se vea formada por triángulos. Por otro lado, mediante el mismo programa podemos generar externamente la misma neurona con menos vértices, perdiendo detalle. A esta nueva malla se la denomina LOD y podemos asignarle a la neurona principal (LOD0) estas mallas secundarias como LOD1, LOD2, etc. De esta forma el motor automáticamente renderizará una malla más compleja o simple teniendo en cuenta la distancia del usuario a cada una de las neuronas, reduciendo considerablemente el consumo de recursos en la máquina del cliente.

- NeuronForest: Derivado de la clase Actor permite cargar la información de una escena y representarla a través de la generación de objetos Neuron en las posiciones exactas que forman la escena neurocientífica. Esta escena será renderizada en el cliente pero gestionada desde la parte servidor, siendo éste último el que controle todo el sistema de microelectrónica neuronal.
- MyActorNotification: Se trata de un actor representado por una pequeña esfera que actúa como marca en las neuronas para proporcionar a los usuarios un objeto visual que refleje un punto de información introducido por un usuario. Almacenará la información introducida por el usuario y se replicará a todos los clientes.
- NavigatorHUD: Clase que representa visualmente en primer plano de la ventana del cliente diferente tipo de información como el bloque de chat o diferentes menús y elementos de información qué son relevantes para el usuario. Además se encarga del manejo de los inputs que conlleven apertura de menús o acciones que se basen en reconocer un input de un periférico para ejecutar algún tipo de acción preparada. Algunos de los elementos más complejos se incluyen como componentes del HUD y se desarrollan en su propia clase. Estos componentes se constituyen por:
 - SlateChat: Heredada de la clase SCompoundWidget, implementa el sistema de chat, tanto métodos como aspecto visual de forma que puede usarse como componente del HUD. Recibe los inputs de la escritura y los maneja de forma que se pueda escribir texto dentro del componente para enviarlo en forma de mensaje al resto de usuarios.
 - MyUserWidgetNeuron: clase derivada de UUserWidget que proporciona los métodos de ocultamiento y desocultamiento de neuronas seleccionadas o no seleccionadas. El usuario interactuará con este componente por medio de una ventana con las cuatro opciones. Esta ventana se mostrará por medio de un input enviado por el usuario y que el HUD deberá reconocer.
 - MyUserWidgetNeuronInfo: Hereda de la misma clase que la del componente anterior. Cubre el requisito de introducir información en una posición de la neurona. Reconoce la escritura del usuario a través de diferentes inputs e introduce esta información en el actor MyActorNotification para ser replicada al resto de usuarios.

- World: El mundo es un objeto que representa un mapa donde actores y componentes son representados y procesados. Un mundo puede ser un solo nivel persistente con una lista opcional de niveles de flujo que se cargan y se descargan a través de volúmenes y funciones de plano o puede ser una colección de niveles organizados con una Composición Mundial.^[19] En la herramienta se desarrollarán los elementos en base a dos mundos, un primero dedicado al inicio de la aplicación y configuraciones, y un segundo, dedicado a la escena neuronal.
- MyAdvancedFriendsGameInstance: Hereda de la clase GameInstance. Se trata de un UObject instanciado globalmente, por lo tanto es accesible por todas las clases. Puede almacenar cualquier dato que desee transportarse entre distintos mundos. A través de esta clase se evita tener que escribir datos en un archivo de configuración o en un archivo binario para compartir información relevante entre niveles. Esta clase almacenará la configuración de pantalla del usuario y los datos de inicio de sesión.

4.3. Detalles de Implementación

El sistema Unreal Engine compila directamente a cualquier plataforma, pero hay que tener en cuenta que la libertad de decisión queda limitada a no utilizar accesos al sistema operativo ni funciones propias de éstos, evitando tener que realizar varias versiones de una misma acción enfocada a cada uno de los sistemas objetivo en los que se pretenda ejecutar la herramienta de visualización. De ésta forma, la escalabilidad aumenta considerablemente debido a que la herramienta podrá portarse fácilmente a una plataforma que podría no considerarse en un principio.

El lenguaje de desarrollo de la herramienta podría verse como “C++ asistido”, ya que Unreal Engine posee muchas características que facilitan notablemente el desarrollo en este lenguaje.

El asistente de clases genera el objeto con los métodos BeginPlay() y Tick() especificados como sobrecargas. BeginPlay() es un evento que permite saber cuándo el Actor ha entrado en el juego en un estado jugable. Este es un buen lugar para iniciar la lógica de la clase. Tick() se llama una vez por fotograma con la cantidad de tiempo transcurrido desde la última llamada pasada. En este punto se puede hacer cualquier lógica recurrente. Sin embargo, si no se necesita esta funcionalidad es mejor eliminarla para ahorrar rendimiento. Si se elimina, es necesario eliminar la línea en el constructor que indica que Tick está activo.

Unreal Engine proporciona también propiedades que pueden ser establecidas por los diseñadores en el editor de Unreal Engine. Para ello es necesario utilizar el macro especial, UPROPERTY(). Este macro recibe parámetros de visibilidad y edición, además las propiedades pueden ordenarse por categorías.

En la Figura 10, el código presenta propiedades configurables por el editor.

```
UCLASS()
class AMyActor : public AActor {
    GENERATED_BODY()
public:
    UPROPERTY(EditAnywhere, Category="Damage")
    int32 TotalDamage;

    UPROPERTY(EditAnywhere, Category="Damage")
    float DamageTimeInSeconds;

    UPROPERTY(VisibleAnywhere, Transient, Category="Damage")
    float DamagePerSecond;
    ...
};
```

Figura 10: Constructor básico de objeto Actor con propiedades

El indicador VisibleAnywhere marca una propiedad como visible, pero no modificable en el Editor de Unreal, mientras que el indicador EditAnywhere lo permite. El indicador Transient, implica que no se guardará ni cargará desde el disco. Se pretende que sea un valor derivado, no persistente. La Figura 11 muestra las propiedades como parte de los valores por defecto de la clase.

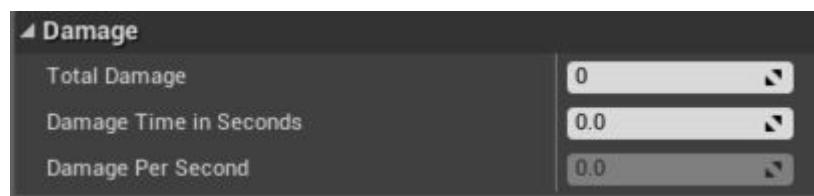


Figura 11: Propiedades gestionadas por el editor

Este tipo de propiedades son accesibles también desde un sistema de secuencias de comandos visuales disponible en el editor de Unreal Engine utilizado para definir clases orientadas a objetos en el motor. Los objetos definidos usando este sistema se denominan coloquialmente como "Blueprints".

Generalmente el uso de Blueprints permite integrar métodos y valores desarrollados en diferentes clases, con aspectos del diseño de interfaces y gráficos en el motor.

Las clases que implementan características de Unreal Engine tienen acceso a otros muchos macros especiales. UE4 utiliza su propia implementación de reflexión que permite características dinámicas como recolección de basura, serialización, replicación en red y comunicación Blueprint/C++. Estas características son “opt-in”, lo que significa que el desarrollador debe agregar el macro correcto a cada tipo, de lo contrario Unreal Engine los ignorará y no generará datos de reflexión para ellos. El conjunto de macros disponibles son los siguientes:

- **UCLASS():** Indica a Unreal Engine que genere datos de reflexión para una clase. La clase debe derivar de UObject.
- **USTRUCT():** Indica a Unreal que genere datos de reflexión para una estructura.
- **GENERATED_BODY():** UE4 reemplaza este macro con todo un código “boilerplate” que incluye todos los datos básicos del objeto necesarios para la construcción del objeto.
- **UPROPERTY():** Permite que una variable miembro de UCLASS o USTRUCT se utilice como propiedad accesible desde el editor. Puede permitir que la variable sea replicada, serializada o modificada desde el editor. También son utilizados por el recolector de basura para realizar un seguimiento de cuántas referencias hay para un UObject.
- **UFUNCTION():** Permite que un método de clase de UCLASS o USTRUCT pueda ser llamado desde funciones del editor o utilizado como RPC, entre otras cosas.

UE4 utiliza el sistema de reflexión para implementar un sistema de recolección de basura. Con la recolección de basura no es necesario gestionar manualmente la eliminación de los UObjects, sólo es necesario mantener referencias válidas a ellos. Las clases deben derivarse de UObject para poder utilizar el sistema de recolección de basura automática. En el recolector de basura existe un concepto llamado conjunto de raíces. Este conjunto de raíces es básicamente una lista de objetos que el recopilador establece que nunca serán recolectados como basura.

Un objeto no será recolectado como basura siempre y cuando haya una ruta de referencias desde un objeto en el conjunto raíz al objeto en cuestión. Si no existe tal ruta, se considerará inaccesible y se recopilará (eliminará) la próxima vez que se ejecute el recolector de basura. El motor ejecuta el recolector de basura en ciertos intervalos.^[23]

Con objetos derivados de la clase Actor, la recolección de basura no se realiza automáticamente. Es necesario llamar manualmente a Destroy(). No se eliminarán inmediatamente sino que desaparecerán durante la próxima fase de recolección de basura.^[23]

La gestión de memoria manual se puede realizar de forma nativa en C++, aunque Unreal Engine permite utilizar el operador “new” inicializando correctamente el vtable (tabla de funciones virtuales) para cualquier función virtual en su tipo de datos, llamando también al constructor correspondiente. Unreal Engine proporciona también funciones de memoria optimizadas para su uso con el motor en la clase FMemory, evitando tener que realizar funciones de memoria de nivel C++. La Figura 15 resume el uso de estas funciones.

4.4. Sistema Multiusuario

Unreal Engine 4 utiliza una arquitectura Cliente-Servidor estándar. Esto significa que el servidor es autoritario y que todos los datos deben enviarse primero desde el cliente al servidor. A continuación, el servidor valida los datos y reacciona dependiendo de la lógica dentro del código.

Tanto servidor como cliente ejecutan el mismo código de simulación neuronal, incluso el mismo ejecutable con la misma configuración, decidiendo si la instancia será cliente o servidor o ambos. En la herramienta de visualización neurocientífica un usuario actúa como el servidor y es a su vez cliente, el resto actúa únicamente como clientes.

En este modelo, el servidor sigue siendo autoritario sobre la evolución del estado de la aplicación. Sin embargo, el cliente mantiene un subconjunto de este estado localmente y por lo tanto, puede predecir el flujo de la acción ejecutando el mismo código que el servidor minimizando así la cantidad de datos que deben ser intercambiados entre los usuarios. El servidor envía información sobre el mundo a los clientes replicando actores relevantes y propiedades. Los clientes y el servidor también se comunican a través de funciones replicadas, las cuales se replican sólo entre el servidor y el cliente propietario del Actor desde el que se llama a la función.

La definición de un servidor usando Unreal Engine se realiza a través del uso de un subsistema online y sus Interfaces, los cuales proporcionan una abstracción a todo el funcionamiento en línea a través de distintas plataformas. Las plataformas en este contexto se refieren a Steam, Xbox Live, Facebook, etc. La portabilidad es uno de los principales objetivos.

De forma predeterminada, se utilizará la plataforma "SubsystemNULL", que permite alojar Sesiones LAN (la sesiones se encuentran a través de una lista de servidores y es posible unirse a ellas accediendo a esta lista) o unirse directamente a través de IP o un grupo de IPs.

El usuario servidor actuará como tal creando una sesión a la que otros usuarios podrán unirse como clientes. Una sesión es básicamente una instancia del juego que se ejecuta en el servidor con un conjunto dado de propiedades. Ésta deberá “anunciarse”, para que pueda ser encontrado y por los usuarios que quieran unirse al estudio de la escena generada en una sesión en concreto.

La interfaz de sesión, `IOnlineSession`, proporciona funcionalidades específicas de la plataforma usada, las cuales permiten a los jugadores encontrar y unirse a la sesión en línea. Esto incluye la gestión de sesiones, permitiendo encontrar sesiones a través de búsqueda así como unirse y abandonar esas sesiones. La interfaz de sesión se crea y es propiedad del subsistema.^[24] Esto significa que sólo existe en el servidor.

Mientras que la Interfaz de sesión realiza todo el manejo de la sesión, la aplicación no interactúa directamente con ella. En su lugar, la sesión de juego `AGameSession` actúa como un contenedor específico de la aplicación alrededor de la Interfaz de sesión.^[24] El código de la aplicación llamará directamente a la sesión de juego cuando sea necesario interactuar con la sesión. La interfaz `GameSession` se crea y es propiedad del `GameMode`.

4.5. Replicación de Datos

El motor dispone de una propiedad denominada `Replication`. La replicación es el acto del servidor que pasa información/datos a los clientes de forma automática. Esto puede limitarse a entidades y grupos específicos. La primera clase que es capaz de replicar propiedades es el `Actor`. Todas las clases que heredan de `Actor` reciben la capacidad de replicar propiedades si es necesario, aunque no todos ellos lo hacen de la misma manera. La clase `GameMode`, por ejemplo, no se replica en absoluto y solo existe en el servidor.

Un `Actor` con '`bReplicates`' establecido en “`true`” dentro de su constructor (Figura 12) se generará y se replicará en todos los clientes cuando se cree por el servidor. Y solamente cuando es generado por el servidor. Si un cliente genera este `Actor`, el `Actor` sólo existirá en este mismo Cliente.

```
AMyActor::AMyActor()
{
    // Replication
    bReplicates = true;
    bReplicateMovement = true;

}
```

Figura 12: Variables de replicación

Por todo ello no se puede permitir que el cliente genere nuevos actores, modifique los ya existentes o cualquier otra acción que deberá compartirse con el resto de usuarios.

La solución pasa por utilizar mecanismos RPC (Remote Procedure Call) incorporado en el motor. Este tipo de llamadas permite llamar funciones del cliente al servidor, del servidor al cliente o del servidor a un grupo específico. Estas llamadas no pueden retornar ningún valor, por tanto es necesario una segunda llamada RPC en la dirección contraria con la información deseada.

Las llamadas RPC se pueden realizar en base a tres reglas:

- Run on Server: La llamada se ejecuta en la instancia del Actor localizada en el servidor
- Run on owning Client: La llamada es ejecutada en el mismo cliente que realiza la llamada.
- NetMulticast: La llamada se ejecuta en todas las instancias del Actor, tanto en clientes como en servidor.

Existen algunos requisitos que deben cumplirse para que las llamadas RPC sean completamente funcionales:

1. Deben ser llamados desde Actores.
2. El Actor debe estar replicado.
3. Si el RPC está siendo llamado desde el Servidor para ser ejecutado en un cliente, sólo el cliente que realmente posee ese Actor ejecutará la función.
4. Si el RPC se está llamando desde el cliente para ser ejecutado en el servidor, el cliente debe ser el propietario del Actor en el que se está invocando el RPC.
5. Los RPC multicast son una excepción:
 - Si son llamados desde el servidor, el servidor lo ejecutará localmente, así como lo ejecutará en todos los clientes conectados.
 - Si son llamados desde los clientes, sólo se ejecutará localmente y no se ejecutará en el servidor.

Un Actor que necesite realizar llamadas RPC deberá incluir “UnrealNetwork.h” en el header. Será necesario especificar el macro UFUNCTION() en el header del Actor como se representa en la Figura 13.

```
UFUNCTION(Server, Reliable, WithValidation)
void Server_Action();
```

Figura 13: Ejemplo header RPC

Mientras que en el fichero .cpp del Actor se deben implementar dos funciones diferentes, la primera con el sufijo “_Validate” y la segunda con el sufijo “_Implementation”, representadas en la Figura 14.

```
bool AMyActor::Server_Action_Validate() {
    return true;
}

void AMyActor::Server_Action_Implementation() {
    // actions
}
```

Figura 14: Ejemplo cuerpo RPC

La idea de validación es que si la función de validación de un RPC detecta que alguno de los parámetros es incorrecto, puede notificar al sistema que desconecte al cliente/servidor que inició la llamada RPC o realizar cualquier otra labor de seguridad.

En la replicación de Actores se pueden establecer dos modos de simulación:

- ROLE_SimulatedProxy: Éste es el modo de simulación estándar. Generalmente se basa en la extrapolación del movimiento basado en la última velocidad conocida. Cuando el servidor envía una actualización para un actor determinado, el cliente ajustará su posición hacia la nueva ubicación y a continuación, entre actualizaciones, el cliente continuará moviendo el actor en función de la velocidad más reciente enviada desde el servidor.
- ROLE_AutonomousProxy: Generalmente sólo se usa en actores que poseen un PlayerController. Esto significa que el actor recibe inputs de un usuario por lo que en el proceso de extrapolación se envía más información y es posible utilizar los inputs del usuario para llenar la información que falta (en lugar de extrapolación basada en la última velocidad conocida).

Dentro de una escena colaborativa donde se pueden encontrar multitud de actores es necesario tener un control de relevancia. Esto permite regular la información que reciben los clientes desde el servidor en base a la importancia del actor en un momento dado, donde su localización puede ser lo suficientemente lejana como para no necesitar su información hasta que se sitúe en un lugar más cercano. Esto beneficia notablemente el ancho de banda consumido por la aplicación.

A través de establecer un valor a distintas variables como 'bAlwaysRelevant', 'bNetUserOwnerRelevancy', 'bOnlyRelevantToOwner', se definen valores de relevancia dependiendo del tipo de Actor. Además es posible controlar la relevancia de los actores en base a cuánta importancia tienen en una escena con la variable 'NetPriority' cuyo valor por defecto es la unidad y específica el ratio de carga.

4.6. Interfaz

La interfaz de la herramienta de visualización se separa en dos módulos, el primero comprende el menu de inicio y el menú principal, el segundo cubre los aspectos del HUD del usuario.

El primer módulo, la interfaz, se ha implementado a través del UI Designer de Unreal Motion Graphics (UMG). Se trata de una herramienta de creación de UI visual que se puede utilizar para crear elementos de interfaz de usuario. Facilita una rápida implementación al disponer de la capacidad de ejecutar funciones establecidas como 'VisibleAnywhere' en forma de programación por bloques desde el propio editor.

Los gráficos UMG utilizan un mecanismo Drag & Drop de componentes o widgets, los cuales se pueden combinar entre ellos para formar estructuras complejas.

En la Figura 15 se captura el diseñador con el menú de inicio.

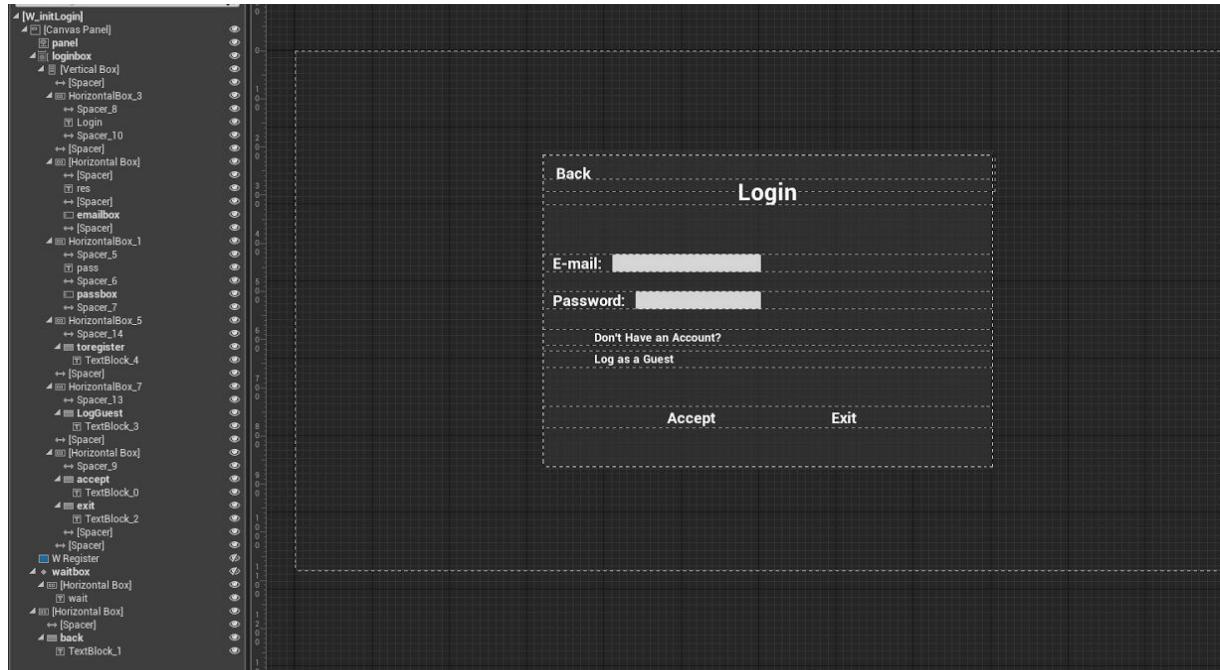


Figura 15: Diseñador del Menú Inicial

Se establece el control de la visibilidad entre el apartado de inicio de sesión y el de registro a través del uso de Blueprints disponibles desde el editor, así como otras funciones relativas a eventos de clics de botones. Ello facilita notablemente la velocidad con la que se implementan funciones dentro de la interfaz.

En la Figura 16 se representa en un Blueprint, las funciones de visibilidad en función del botón clicado.

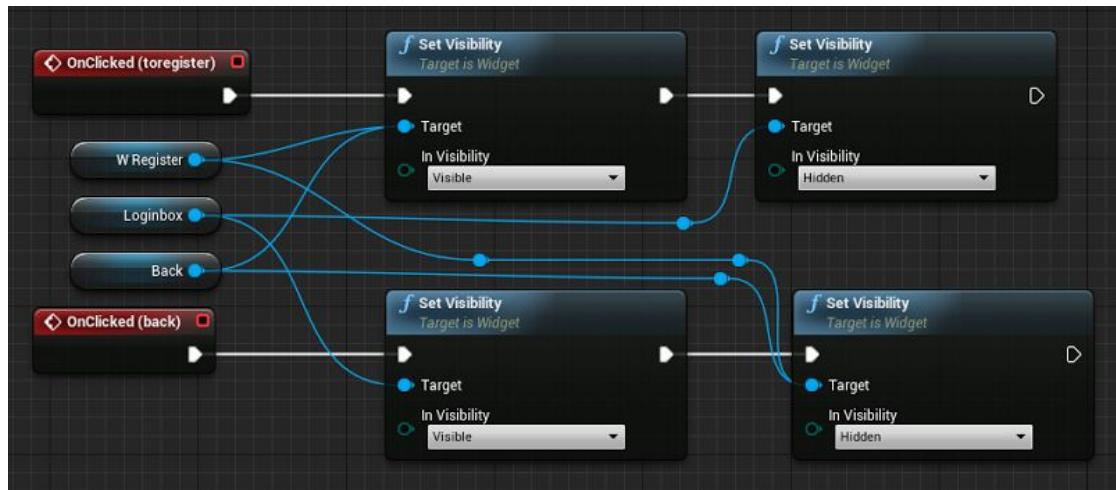


Figura 16: Control de visibilidad en el menú inicial

El sistema de identificación de usuario y registro se realiza también utilizando este modelo de programación por bloques, llamando a métodos desarrollados en C++.

En la Figura 17 se captura un fragmento de la función de Identificación de usuario realizada un Blueprint, y en la Figura 18 el método al que hace referencia el bloque en el código.

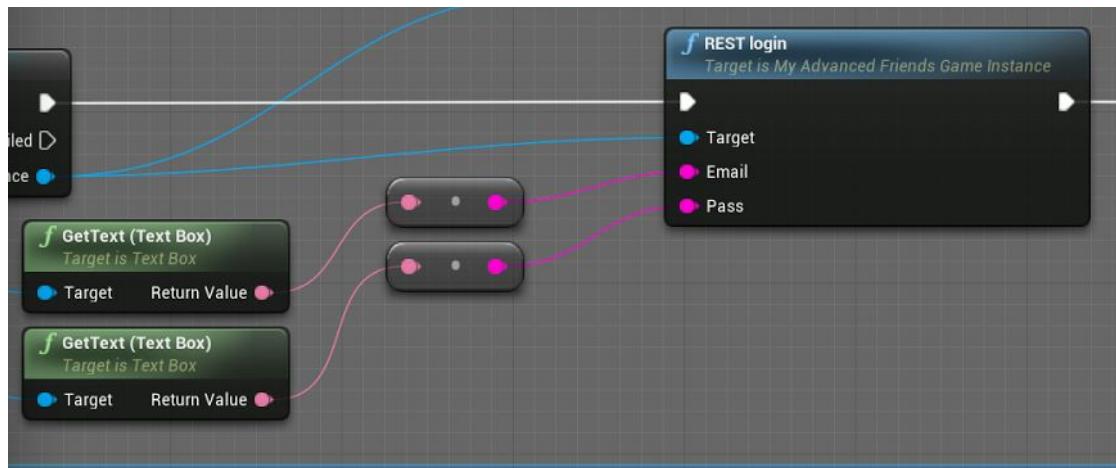


Figura 17: Comunicación Blueprint/C++ en identificación de usuario

```

/** REST Login Request */
UFUNCTION(BlueprintCallable, meta = (DisplayName = "REST login",
Keywords = "REST login"), Category = "REST")
void Login(FString email, FString pass);
  
```

Figura 18: Fragmento del header de la clase 'MyAdvancedFriendsGameInstance'

Haciendo seguimiento de esta metodología de desarrollo se implementan todas las funciones del menú inicial, cuyo aspecto final se representa en las figuras Figura 19 y Figura 20, permitiendo al usuario identificarse en la aplicación usando un sistema intuitivo y estandarizado.

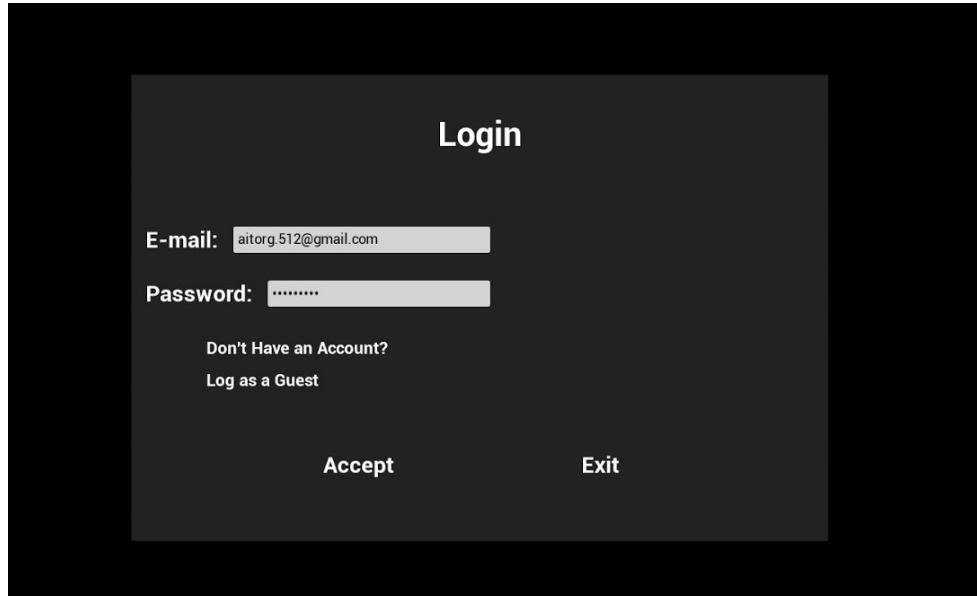


Figura 19: Ventana de identificación de usuario

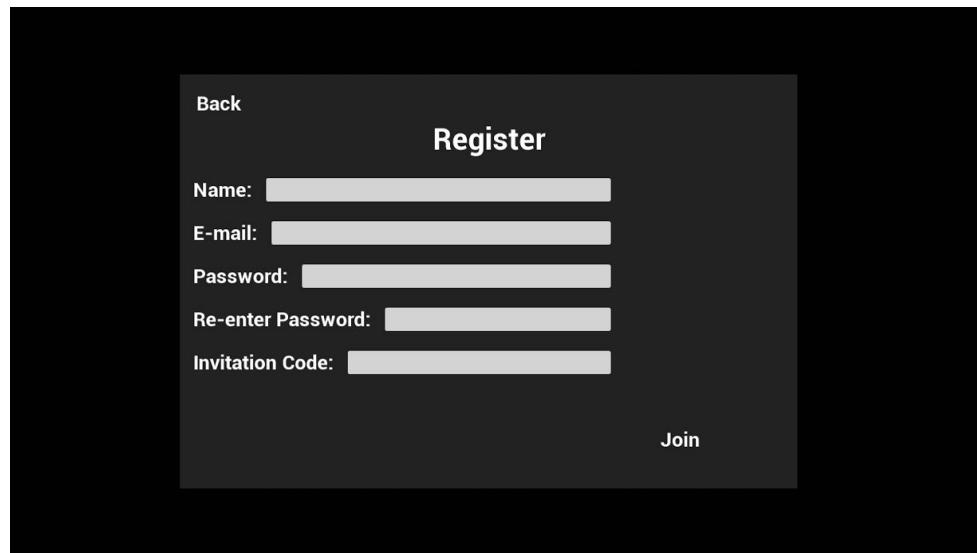


Figura 20: Ventana de registro de usuario

Para conseguir un control de registro se usa un sistema de registro por invitación, permitiendo acceder al sistema como usuario registrado sólo a expertos o usuarios relacionados directamente con el proyecto Human Brain Project. Separando a usuarios con intención didáctica o similares que deberán acceder como usuarios invitados dentro de la herramienta de visualización científica.

El menú principal, por tanto, se implementa utilizando los mismos métodos que el menú inicial. Debe reconocer si el usuario se identificó como invitado o como usuario registrado, limitando las opciones del usuario invitado a únicamente poder cambiar la configuración de pantalla y unirse a una sesión de trabajo activa, realizando una búsqueda de sesiones activas. Cabe destacar que este menú realiza llamadas a los métodos utilizados para crear una nueva sesión (Figura 21), buscar sesiones disponibles y unirse a una sesión activa.

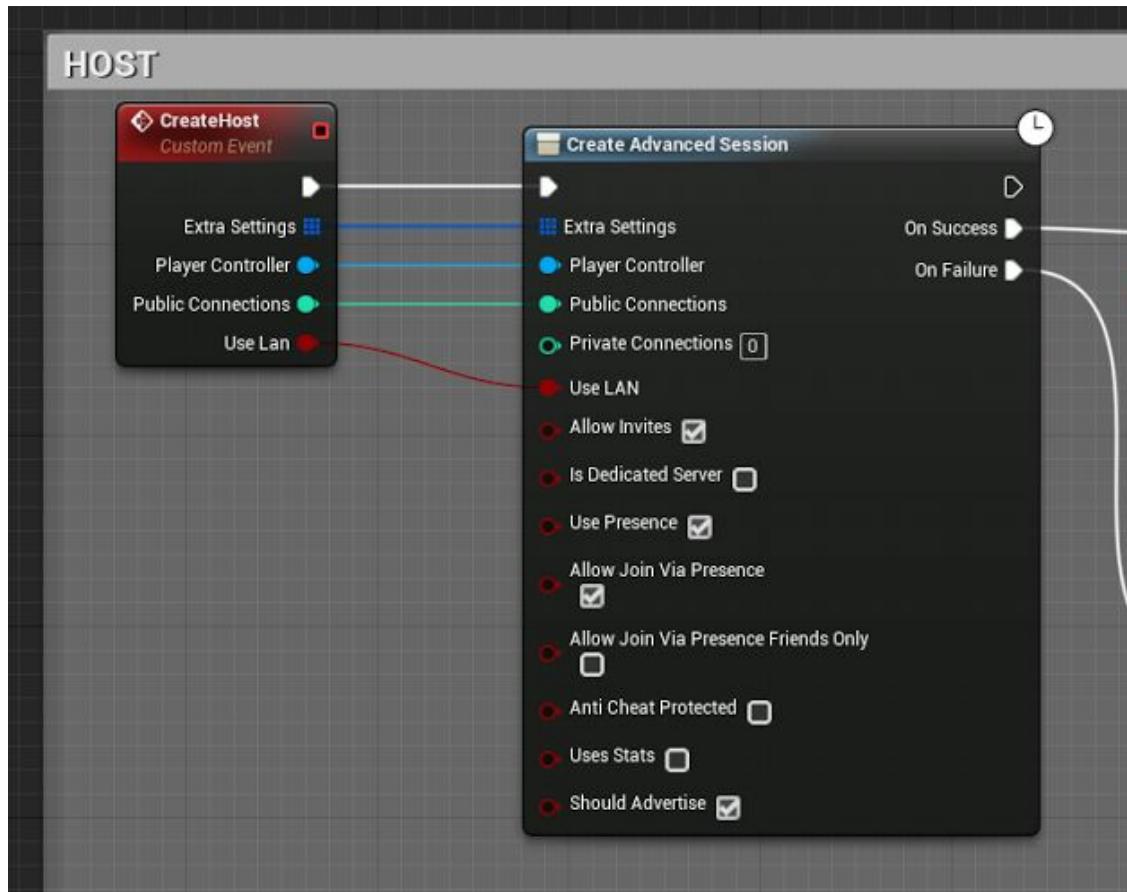


Figura 21: Creación de nueva sesión

En las figuras Figura 22 y Figura 23 se pueden apreciar los diferentes aspectos visuales que toma el menú principal en función de si el usuario es invitado o normal.

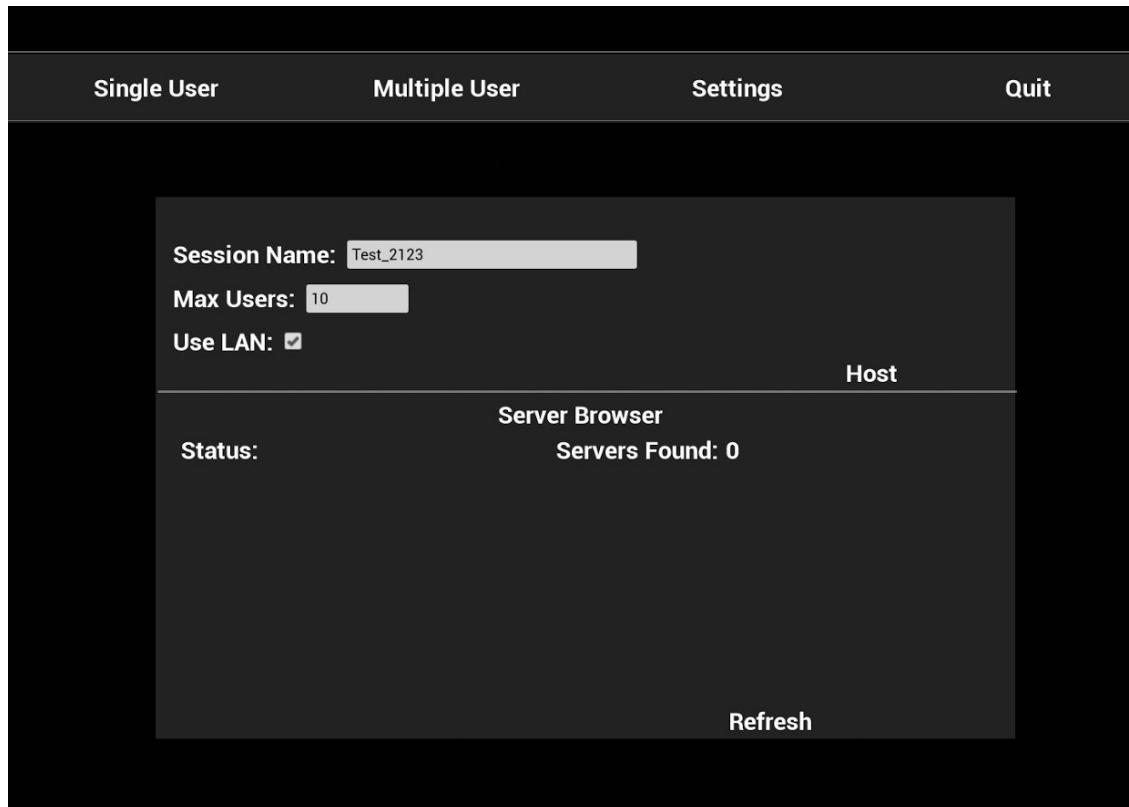


Figura 22: Creación de nueva sesión en el Menú principal de usuario identificado

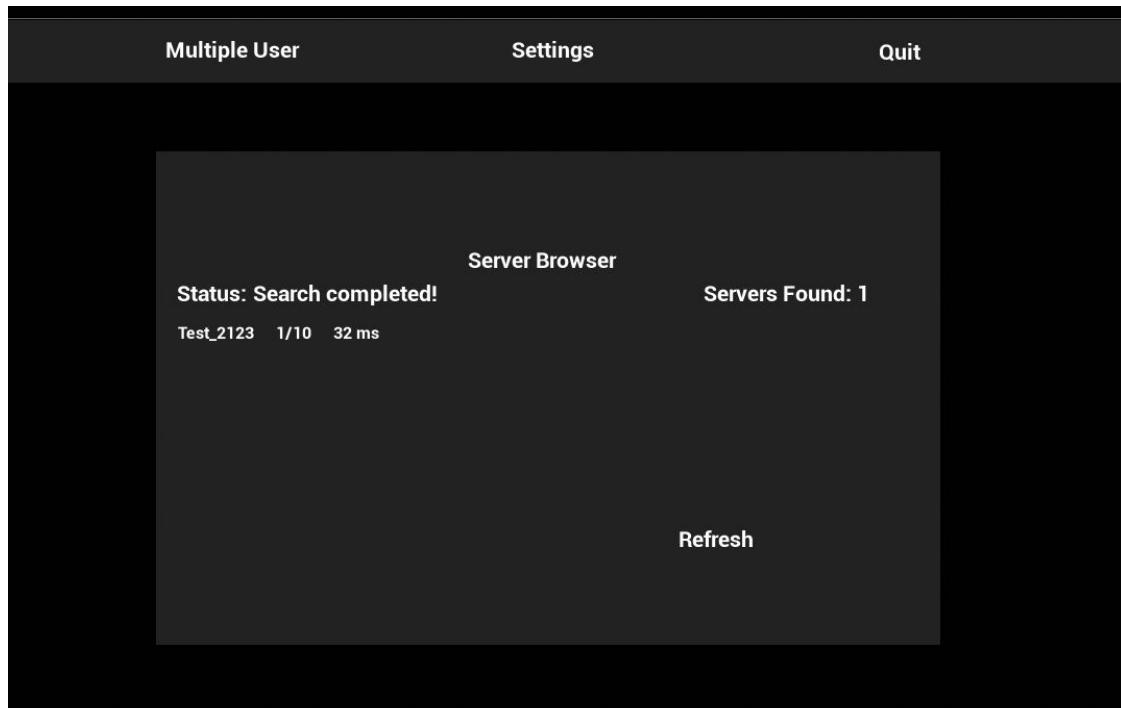


Figura 23: Búsqueda de sesiones activas en el Menú principal de usuario invitado

El segundo módulo que cubre lo referente al HUD de los usuarios una vez se encuentren en el interior de una escena. El HUD se implementa en la clase NavigatorHUD, siendo su principal labor el reconocimiento de diferentes inputs producidos por el usuario. NavigatorHUD representa visualmente diferente tipo de información como menús o el bloque de chat a través del uso de componentes.

El HUD reconocerá diferentes inputs procedentes del controlador asociado a cada usuario, ejecutando las tareas asociadas al input en cuestión.

La construcción del bloque de chat se realiza sin hacer ningún uso del diseñador de Unreal Engine, utilizando una sintaxis especial declarativa para el desarrollo de interfaces visuales con el framework de interfaz Slate UI disponible dentro del UDK.

La representación visual resultado de la construcción del componente para mostrar el bloque de chat se representa en la Figura 24.

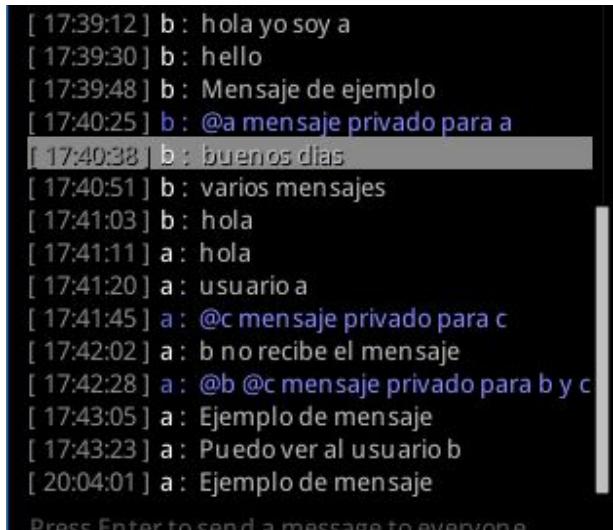


Figura 24: Representación del bloque de chat dentro del HUD

Además del bloque de chat, NavigatorHUD dispondrá de los componentes UMyUserWidgetNeuron y UMyUserWidgetNeuronInfo usados como menus para el control de ocultamiento de neuronas y añadir información sobre neuronas.

4.7. Visualización de bosques neuronales

La representación de las morfologías neuronales se extrae del prototipo de una investigación anterior (Véase punto 2.4). Esta metodología está basada en la importación de mallas neuronales generadas por una aplicación externa, Neuronize. Una vez se obtienen las mallas de cada una de las neuronas de la escena a generar, éstas se distribuyen en diferentes localizaciones con una rotación determinada obtenida de un fichero de escena.

En el proyecto actual se implementa la representación de estructuras microelectrónicas, emulando el funcionamiento microelectrónico de las neuronas presentes en la escena a analizar.

Un impulso nervioso es propagado hacia la terminación nerviosa, provocando un cambio de permeabilidad en los canales de sodio, iniciándose la despolarización y permitiendo el influjo de iones de calcio^[25]. Por tanto es posible representar diferentes impulsos eléctricos en una neurona teniendo en cuenta la cantidad de calcio y cómo se distribuye en la neurona a lo largo del tiempo, para ello es necesario recibir esta información desde el fichero de escena.

Es posible realizar la emulación de los impulsos nerviosos a través de la propiedad ‘Emissive Glow’ de los materiales en Unreal Engine. Sobre la malla neuronal se aplica un material emisivo, el cual da la ilusión de que un objeto está emitiendo luz, consumiendo muy pocos recursos ya que no se trata de un foco de luz que deba generar sombras e interactuar con el resto de actores del mundo 3D.

En la Figura 25 se reproduce una escena neuronal en la que se aplica un material emisivo sobre las neuronas, usando los niveles de calcio y una variable de tiempo para representar el impulso neuronal.

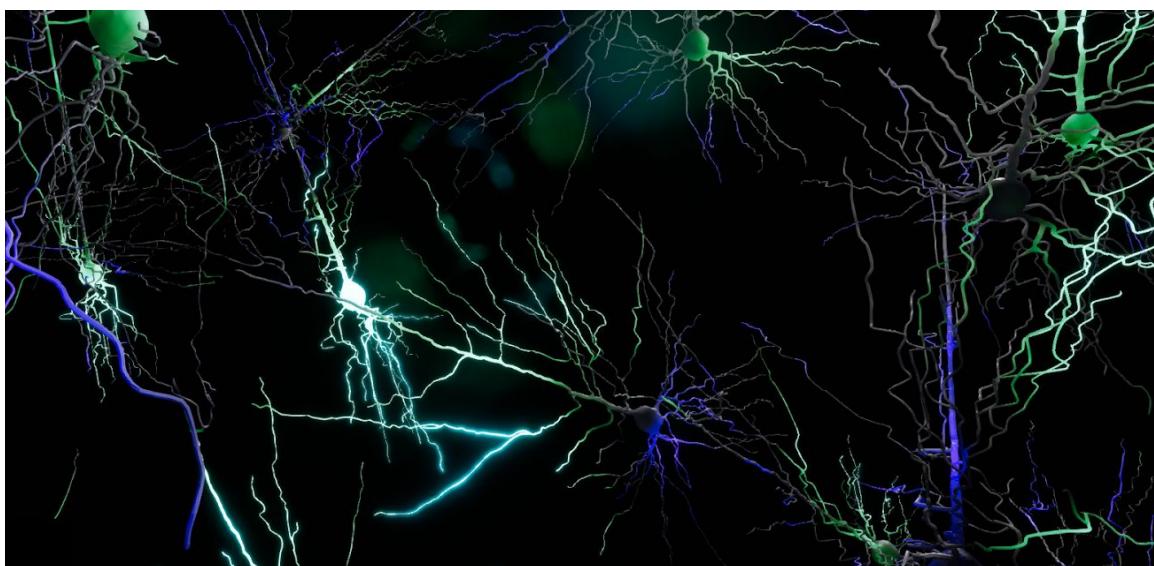


Figura 25: Material emisivo en las neuronas de una escena

4.8. Comunicación con el Sistema de Persistencia

La comunicación con el sistema de persistencia se realiza a través de llamadas Http al recurso correspondiente establecido por el servicio web dedicado a la persistencia.

Unreal Engine no dispone de métodos que ejecuten verbos Http de forma nativa, por lo que se implementan estos métodos dentro de la clase del controlador usando un módulo Http el cual permite personalizar muchas de las variables disponibles para realizar una llamada.

Usando este módulo se establecen cabeceras como “Content-Type” y “Accepts” en Json ya que este formato de datos es el que utiliza el servicio web de persistencia. Así mismo se puede establecer un verbo para realizar la llamada, pero únicamente acepta “GET” y “POST”, detalle importante con el que el sistema RESTful sólo podrá usar estos verbos para disponer sus recursos.

Finalmente es posible establecer un contenido como una cadena de caracteres y procesar la llamada hacia la dirección deseada.

El procesamiento tanto de datos recibidos como enviados, se realiza a través de la clase FJsonObjectConverter y sus métodos JsonObjectToString y UStructToJsonObject String, los cuales transforman una cadena de caracteres con formato Json, a una estructura de datos fácilmente accesible, y viceversa.

4.9. Usuarios

Los usuarios dentro de una escena se verán representados a través de un pawn generado por MyDefaultPawn, como se aprecia en la Figura 26. El pawn personalizado de usuario está compuesto por una esfera con el identificador de usuario encima de ella a modo de texto flotante.

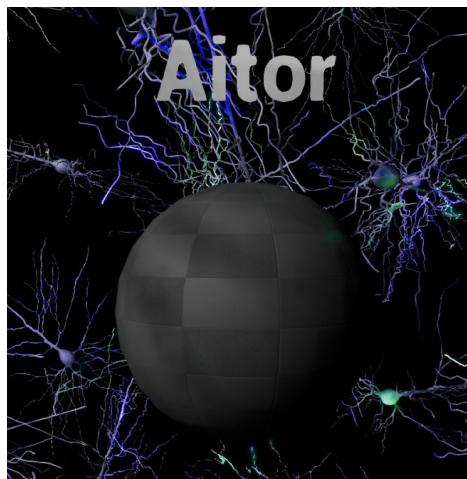


Figura 26: Pawn de un usuario en el interior de una escena

Información tal como el nombre de usuario u otra información de identificación del usuario se almacenan en el GameInstance gracias a su propiedad de persistencia a través de los distintos niveles, permitiendo almacenar la información recibida en la operación de identificación de usuario para ser extraída más tarde por el pawn de usuario y, entre otras funciones, establecer su texto flotante como el identificador de usuario.

Esta información de identificación de usuario, la cual es únicamente conocida por el cliente en cuestión, debe ser enviada al servidor para que almacene la información en la referencia del pawn del cliente que él dispone, permitiendo que la información se replique al resto de usuarios presentes en una escena neuronal.

Usando el mismo método de replicación usada para la replicación de la identificación se realiza una replicación del movimiento en el servidor, ya que de otra manera los usuarios sólo verán un pawn fijo y sin movimiento para un pawn de cualquier otro usuario.

Una vez el servidor procese la información recibida la información y mueva el pawn de un cliente, el resto de clientes verán todos estos cambios gracias a las propiedades de replicación que proceden del servidor.

Respecto al identificador flotante de usuario, éste deberá adaptar su rotación para que el resto de usuarios puedan verlo siempre en la posición correcta. Para ello se implementa una función que no se replica, creando un efecto distinto en cada uno de los clientes. Este método extrae la rotación actual del usuario en un cliente y se le aplica a la rotación del texto flotante en cada uno de los frames de la aplicación a través del método Tick().

4.10. Métodos de Comunicación entre usuarios

Los usuarios tienen la necesidad de comunicarse entre ellos para colaborar en el análisis de diferentes escenas neurocientíficas, por tanto es necesario disponer de un sistema de comunicación permita dejar un registro de la comunicación con información relevante del análisis en el sistema de persistencia y a su vez mantener la agilidad de una comunicación fluida en el intercambio de opiniones.

La calidad de la comunicación depende en parte del diseño de las interfaces de usuario que se desarrollen, de la usabilidad del producto final que se alcance y del grado de sociabilidad y socialización que los grupos practiquen en un sistema colaborativo.

Es necesario implementar dos sistemas de comunicación que funcionen en paralelo, el primero un sistema de chat que favorece el mantenimiento de la información al poder registrar los mensajes intercambiados en un servicio de persistencia, y el segundo un sistema de comunicación a través de voz que posibilita una comunicación fluida y ágil.

4.10.1. Comunicación a través de Chat

La comunicación usando un chat permite el envío de mensajes a través de un sistema de chat grupal. Estos mensajes llegan al resto de usuarios de forma inmediata y se almacenarán en el sistema de persistencia. Se implementa el envío de mensajes a través de llamadas RPC, ya que los mensajes no tienen propiedades que permitan su replicación de forma automática como ocurre con los actores, de forma que cuando un usuario escribe un mensaje, éste se almacena en una estructura y se envía al servidor a través usando RCP con la propiedad 'Server'. El servidor a través de esta función almacenará la información en el sistema de persistencia y a su vez realizará una llamada 'NetMulticast' con la que todos los usuarios recibirán el mensaje enviado.

Estos métodos RPC pertenecen a la clase PlayerState, la razón, todos los usuarios presentes en una sesión tienen una referencia al PlayerState de los demás usuarios por tanto es posible ejecutar sin problemas la llamada ‘NetMulticast’. Estos métodos se pueden encontrar en el header de la clase XPlayerState.

Los mensajes, tanto enviados como recibidos se incluyen en una lista de mensajes actuales dentro del bloque de chat, la cual tiene asociada un ‘listener’ que se activa al recibir cambios, añadiendo los nuevos mensajes a la interfaz.

Si un usuario se une a una sesión activa, todo el chat almacenado en persistencia será recuperado de forma que este usuario pueda ver toda la conversación anterior a su llegada. Para ello se realiza una llamada Rest con el identificador de sesión como parámetro. Los mensajes se incluirán en la lista de mensajes actuales y automáticamente el bloque de chat los introducirá como parte de su interfaz.

Un usuario invitado podrá ver toda la conversación pero no escribir ella, ya que sólo deben poder afectar al sistema de persistencia los usuarios identificados permitiendo un control de los cambios. En consecuencia, la escritura de mensajes se deshabilita para este tipo de usuarios, sin embargo no se limitará su visibilidad a los mensajes enviados por el resto de usuarios.

Se utilizará el estándar @nombredeusuario, como método para enviar mensajes privados a uno o más usuarios a la vez de forma que si un usuario menciona a otro con este método dentro de un mensaje, el mensaje sólo será recibido por el usuario o los usuarios mencionados. El elemento ChatInput del bloque de chat reconoce la entrada del carácter '@' junto con el texto que lo acompaña asociándolo a un usuario e introduciéndose como parámetro en el envío del mensaje, limitando la replicación de este mensaje únicamente al usuario o usuarios mencionados utilizando este método.

4.10.2. Comunicación a través de Voz

Un usuario que se une a una sesión colaborativa podrá comunicarse por medio de voz. Este sistema está disponible tanto para usuarios registrados como invitados, dando la capacidad a estos últimos de interactuar con el resto de usuarios. Permite por tanto realizar comentarios de forma fluida y agilizar el desarrollo del análisis, además este sistema facilita al usuario poder comunicarse si no dispone de teclado o utiliza otro periférico como un mando u otro periférico alternativo para navegar por la escena neuronal.

El usuario deberá disponer de altavoces y micrófono para usar esta característica. Por esta razón, la herramienta debe reconocer si el usuario tiene activados estos dispositivos previo al inicio del sistema de comunicación por voz.

Actualmente solo OnlineSubsystemSteam y OnlineSubsystemNull proporcionan soporte de chat de voz utilizando sus respectivas interfaces. En virtud de ello, es posible recuperar los datos de voz locales a través del método ProcessLocalVoicePackets() y agregarlos a una matriz que se enviará a OnlineVoiceImpl. Además se dispone de los métodos:

- NetDriver::ProcessLocalClientPackets(), el cual, envía paquetes al servidor.
- NetDriver::ProcessLocalServerPackets() envía paquetes a otros clientes.

Los paquetes se abren y se envían a IOnlineVoice para su procesamiento, IOnlineVoice descomprime el audio y lo reproduce a través de IVoiceEngine en ProcessRemoteVoicePackets().

Este sistema de comunicación se activa en el método PostLogin() de la clase GameMode. Este método es llamado por la sesión cuando un usuario se une a ella. En éste se encuentran las llamadas a los métodos de la clase OnlineVoiceImpl.

4.11. Mecanismo You See What I See

Este mecanismo pretende que un usuario pueda visualizar lo que está viendo otro usuario de forma sincronizada fomentando tanto la formación didáctica como el análisis científico.

Es necesario que el usuario realice la llamada a este método a través de un input. La clase encargada de reconocer este input es el propio HUD del usuario.

El método está basado en iterar por cada uno de los Pawns asociados a cada usuario presente en la escena y obtener una referencia a su “cámara”, la cual se asigna como “ViewTarget” en el controlador asociado al usuario. Pasando por cada una de las “cámaras” disponibles hasta llegar a la del usuario deseado y mostrando en el proceso un input visual cómo se puede apreciar en la Figura 27.



Figura 27: El usuario del cliente en la derecha puede ver la pantalla del usuario en la derecha.

4.12. Selección de Neuronas

Con el fin de poder analizar de forma focalizada una selección se implementa dentro de la herramienta de visualización la posibilidad marcar diferentes neuronas y replicar esta selección al resto de usuarios dentro de la sesión de forma sincronizada. Para ello se modifica el material actual de la neurona por otro representativo de selección.

Teniendo en cuenta que la escena neuronal se genera en el GameMode y éste se encuentra presente únicamente en el servidor, los usuarios no tienen ninguna referencia hacia estas neuronas. Esto genera el problema de no poder utilizar llamadas RPC siguiendo una metodología estándar ya que el cliente no tiene acceso a los métodos del objeto neurona.

La solución recae en dejar el control de marcar una neurona en la clase del controlador. Entonces, cuando se realiza la llamada al método de selección en el cliente, se ejecuta la llamarada RPC al servidor (ya que el servidor tiene referencias de todos los controladores de los usuarios presentes en una escena), llamando al método de selección del Actor Neuron pero esta vez la llamada estará en el servidor.

El sistema ofrecerá al usuario la opción de ocultar neuronas según éstas estén seleccionadas o no. Las neuronas que se oculten lo harán para todos los usuarios dentro de la sesión.

A través de un input, el HUD presentará un menú como el de la Figura 28.

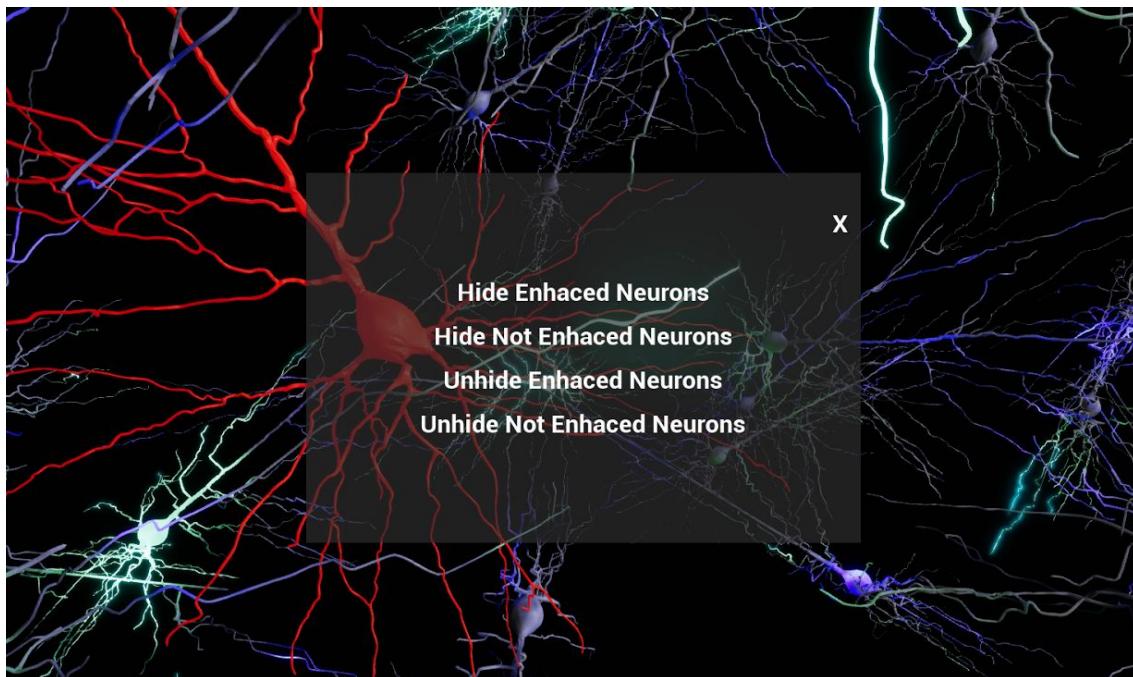


Figura 28: Selección de neuronas y menú asociado

4.13. Información Contextual

Una característica clave en la herramienta de colaboración es la capacidad de establecer distintos puntos de información en posiciones concretas del entorno 3D de una escena, de modo que es necesario que el sistema reconozca el lugar exacto donde un usuario quiere introducir la información.

El usuario generará un vector frontal en dirección a la posición en la que se encuentra el puntero del usuario, haciendo uso del método LineTraceByChannel del framework UDK. Este método se ejecuta al recibir un input del usuario, retornando en cada una de sus ejecuciones el actor y la localización de impacto del vector.

Gracias a la información que se produce es posible generar un nuevo actor MyActorNotification. Su creación se debe realizar a través de un método en el servidor para que pueda ser replicado, por tanto deberá llamarse a este método desde el cliente usando RPC.

El actor que se genera para marcar un punto de información contextual está formado únicamente por una esfera de color llamativo como se observa en la Figura 29. Además almacena la información que el usuario desee introducir a través de un menú asociado al HUD del usuario, el cual se muestra al realizar esta acción.

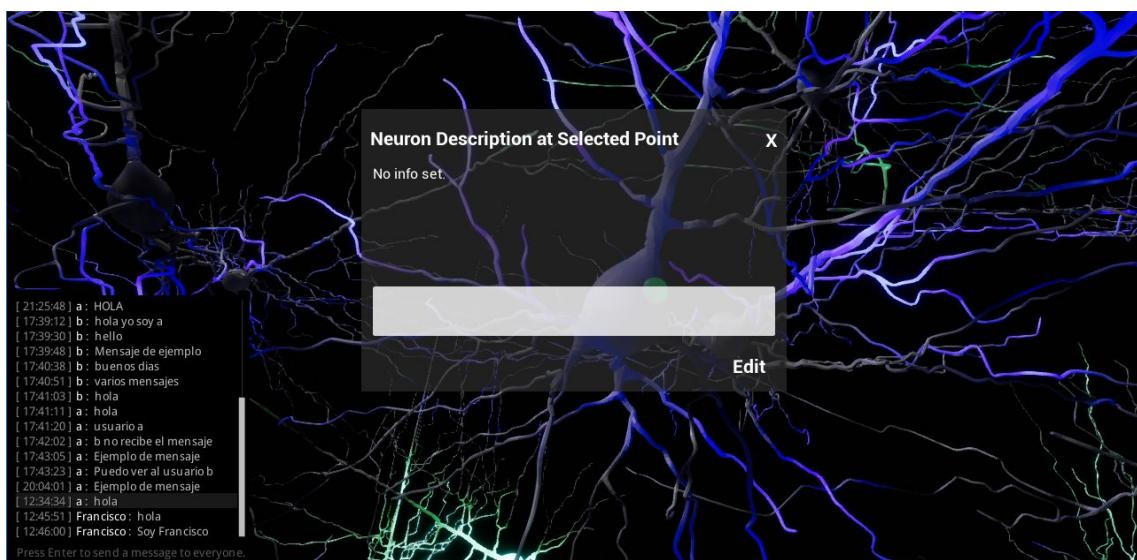


Figura 29: Usuario estableciendo información sobre una neurona

El resto de usuarios realizar doble-clic sobre el objeto recién creado, serán capaces de leer la información establecida previamente. En la Figura 30 se puede ver un cómo un usuario accede a la información del punto creado en la Figura 29.

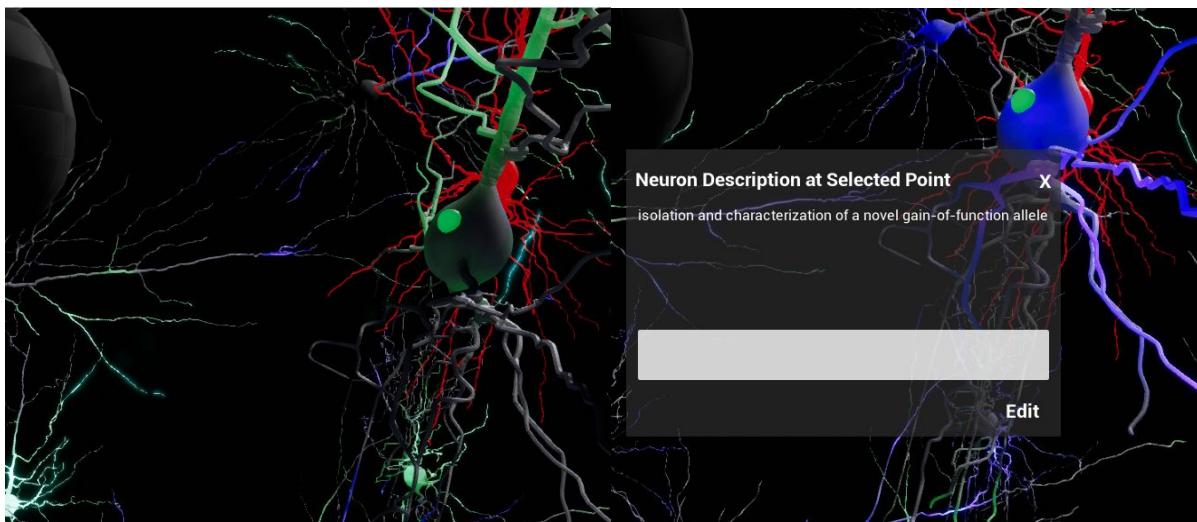


Figura 30: Usuario accediendo a información sobre una neurona

4.14. Captura de Imágenes

El usuario podrá realizar una captura de toda la escena que le rodea en 360 grados desde su posición, a través de una de las funcionalidades disponibles de Unreal Engine, SP.PanoramicScreenshot al cual se pueden establecer diferentes parámetros asociados a la calidad de renderización de la imagen^[26].

Al generarse una nueva imagen el sistema realizará la llamada correspondiente al sistema Rest para almacenar este fichero en el sistema de persistencia.

En la Figura 46 se presenta una captura realizada en alta resolución desde el interior de una escena utilizando esta funcionalidad. Debido a su alta resolución la Figura 31 no escala correctamente y perdiendo calidad en la representación.

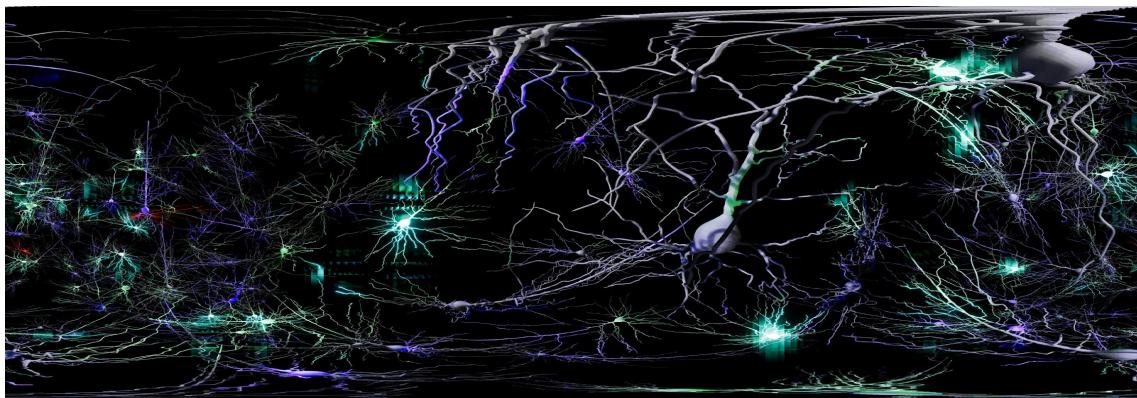


Figura 31: Captura de una escena en 360 grados

4.15. Sistema de Repetición

Un sistema de repetición permite que los usuarios pueden volver a ver los pasos recorridos hasta llegar al punto en el que se encuentran usando algunos controles básicos de visualización como botones de play/pause y controles de progreso de la reproducción. Este sistema se puede activar de forma sincronizada en todos los usuarios de una escena a la vez de forma nativa haciendo uso de las propiedades del motor gráfico de Unreal Engine, siendo el cliente que active esta funcionalidad el controlador del curso de la repetición.

Esta funcionalidad no ha podido implementarse correctamente en la herramienta de visualización colaborativa, ya que cabe destacar que el sistema de replay interno de Unreal Engine se encuentra actualmente en construcción y no se recomienda su uso, más que para realizar diversos tipos de pruebas^[27].

El sistema de replay en su fase actual no es capaz de reconocer elementos personalizados de una escena como pueden ser las Neuronas o los Pawn de usuario, de forma que al implementar este sistema dentro de la herramienta e intentar ejecutar una repetición el sistema de Unreal Engine para su ejecución.

Por todo ello, es necesario recibir futuras actualizaciones de Unreal Engine con el objetivo de obtener esta característica como parte de las aplicaciones disponibles dentro de la herramienta de visualización colaborativa.

4.16. Realidad Virtual

Utilizar la inmersión en entornos virtuales a través de equipos de realidad virtual puede mejorar en nivel de concentración de los usuarios y permitir a su vez que realicen mejores análisis al tener una mejor constancia de la visualización al encontrarse en el interior de ésta. presentando una nueva forma de investigación alternativa.

La escena neuronal es totalmente compatible con la realidad virtual de forma nativa al desarrollarse sobre un entorno 3D, sin embargo la interfaz se desarrolla en dos dimensiones por lo tanto no es compatible con esta tecnología. A cambio es posible crear actores que tengan como malla componentes de interfaz, los cuales permiten la interacción con el usuario.

En la Figura 32 se muestra el menú inicial de identificación de usuario establecido dentro de un actor y representado por tanto en 3D.

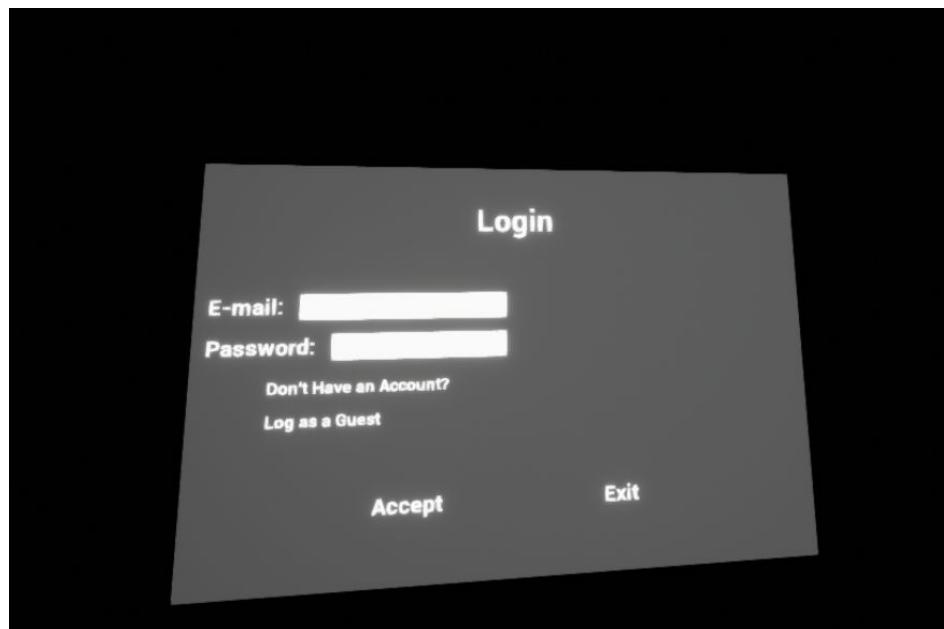


Figura 32: Interfaz UMG sobre un actor en el mundo 3D

El sistema reconoce si un usuario dispone de un equipo de realidad virtual instalado y activo en el sistema, a través de los métodos IsHeadMountedDisplayEnabled() e IsHeadMountedDisplayConnected() de la clase UHeadMountedDisplayFunctionLibrary de Unreal Engine, lo cual permite intercambiar entre el uso de interfaz en dos dimensiones o 3D.

La Figura 33 captura una escena neurocientífica en la que se usa un equipo de realidad virtual.

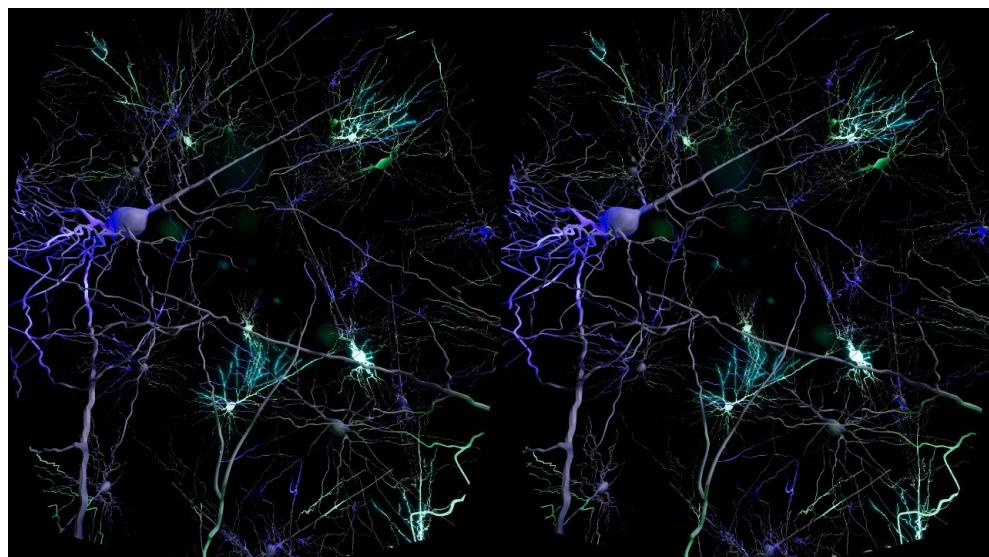


Figura 33: Escena neurocientífica en realidad virtual

5. Implementación del servicio Web y sistema de persistencia

El siguiente módulo se compone los servicios web para el sistema de persistencia y la aplicación de recuperación de información del sistema de persistencia, los cuales se desarrollan sobre los lenguajes PHP y Javascript en conjunto con Html y Css dedicados a la elaboración de páginas web. PHP permite que el código sea interpretado por un servidor web con un módulo procesador de PHP que genera los resultados necesarios y Javascript, lenguaje interpretado por parte del navegador web del cliente. En conjunto forman un servicio rápido y sencillo de implementar sin perder ninguna funcionalidad compleja propia de lenguajes compilados.

Estos lenguajes permiten la implementación de un servicio web RESTful ya que existe el soporte de verbos HTTP GET, POST, UPDATE y DELETE, los datos se preparan correctamente para evitar la inyección SQL, y es posible manejar valores nulos.

5.1. Diagrama de Clases

En la Figura 34 se muestra un diagrama Entidad-Relación que proporciona una idea de cómo se almacenan los datos en el interior del sistema de persistencia.

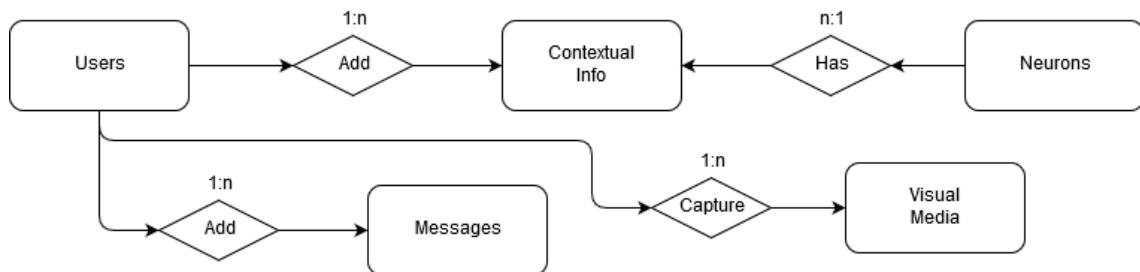


Figura 34: Diagrama E/R del sistema de persistencia

El diagrama del servicio web REST se muestra a través de la Figura 35, en la cual, se muestra cómo acceder a cada uno de los recursos del servicio.

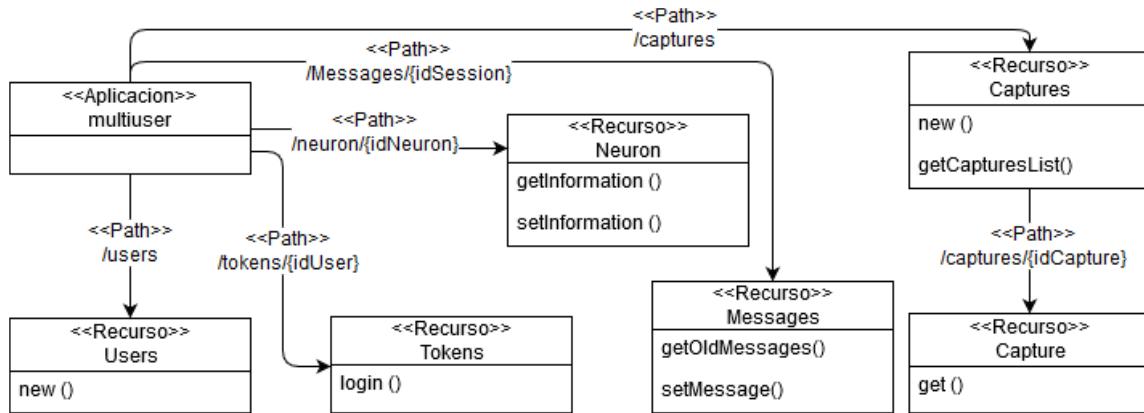


Figura 35: Diagrama UML del servicio REST

A su vez este servicio pone a disposición del usuario un acceso vía web a través del cual es posible recuperar información del sistema de persistencia y visualizarla de forma interactiva. En la Figura 6 se dispone el diagrama de estados de la aplicación web.

5.2. Identificación de usuario

La interfaz del servicio web se desarrolla intentando mantener cierta similitud con el estilo de la herramienta, estableciendo un sistema de identificación a través de usuario y contraseña, y un sistema de registro usando código de invitación, usando llamadas al servicio REST a través de funciones Ajax de Javascript.

En las figuras Figura 36 y Figura 37 se muestran las pantallas de identificación y registro generadas por un navegador

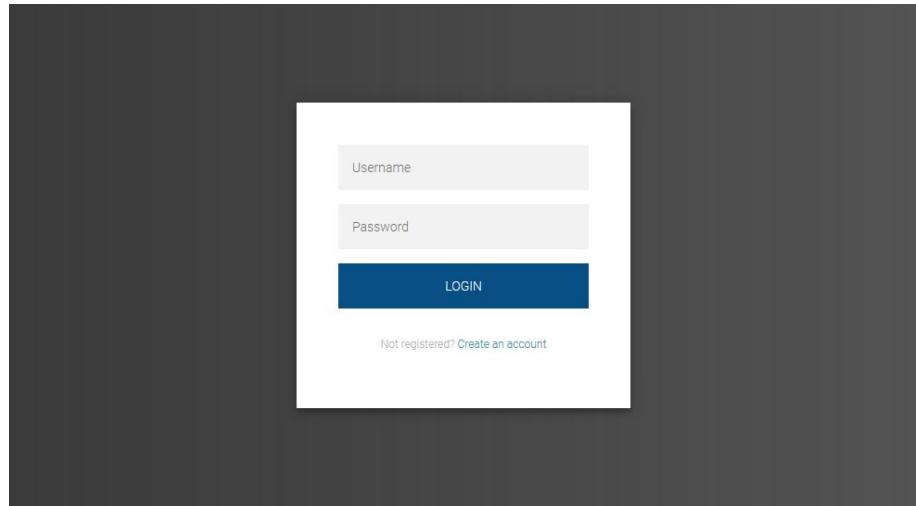


Figura 35: Ventana de identificación de usuario

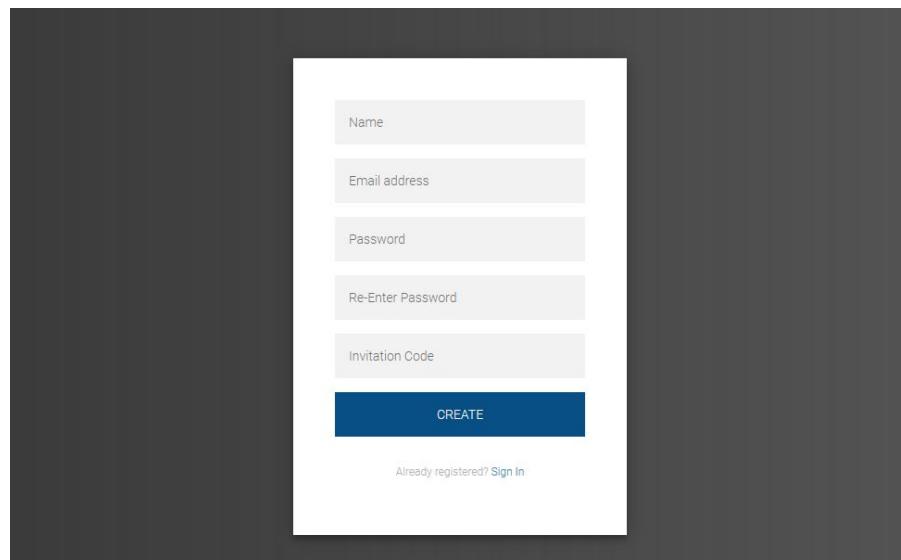


Figura 36: Ventana de registro de nuevo usuario

5.3. Sandbox

La función del sandbox es mantener separadas las herramientas de visualización web del servicio web, permitiendo ejecutar estas herramientas en una nueva instancia del navegador.

Además el sandbox muestra al usuario las herramientas de las que dispone el usuario una vez accede al sistema, tal y como se muestra en la Figura 37.

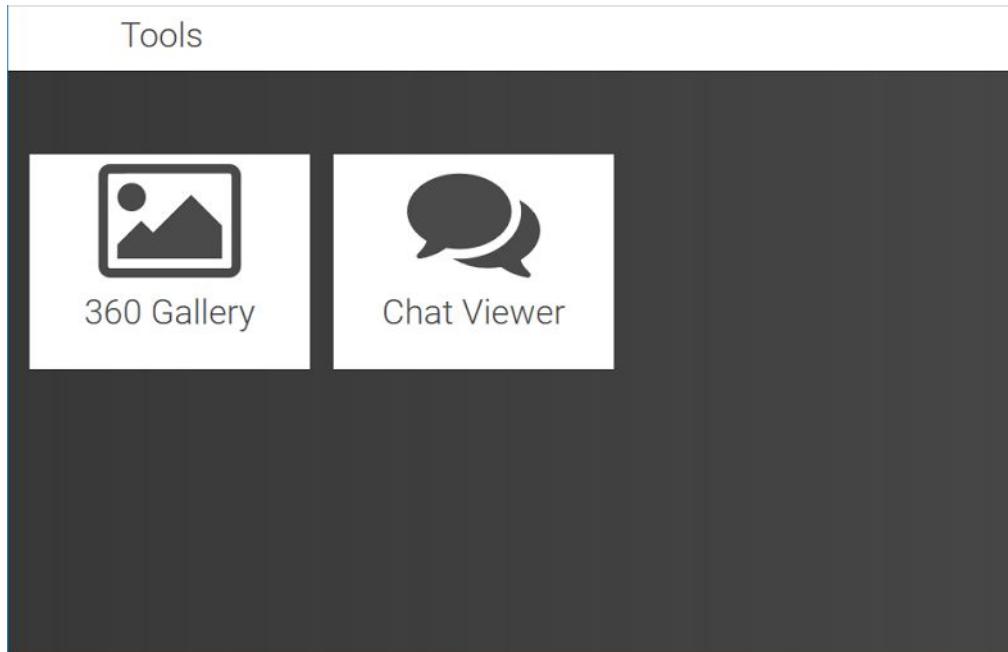


Figura 37: Ventana de Sandbox

5.4. Visualización de Capturas

El sistema de visualización de capturas presenta un listado de las imágenes disponibles en el sistema de persistencia asociadas a una miniatura las cuales se generan de forma dinámica cuando un usuario intenta acceder a ellas y no están disponibles. La creación de miniaturas es necesaria ya que las imágenes procedentes del sistema de persistencia son de muy alta resolución y por lo tanto el consumo de ancho de banda sería demasiado elevado.

Una vez el usuario seleccione una miniatura el sistema representará la captura como si el usuario se encontrara dentro de la herramienta de visualización colaborativa. Para ello, se hace uso de Three.js, una librería bastante liviana y muy eficiente para generar y animar gráficos en 3D dentro del navegador, siendo capaz de generar escenas 3D con WebGL. Adaptando la captura a una esfera, estableciendo la cámara en el centro de la misma y pivotando el giro de esta esfera manteniendo el centro como punto fijo.

En las figuras Figura 38 y Figura 39 se presenta el resultado final de esta implementación.

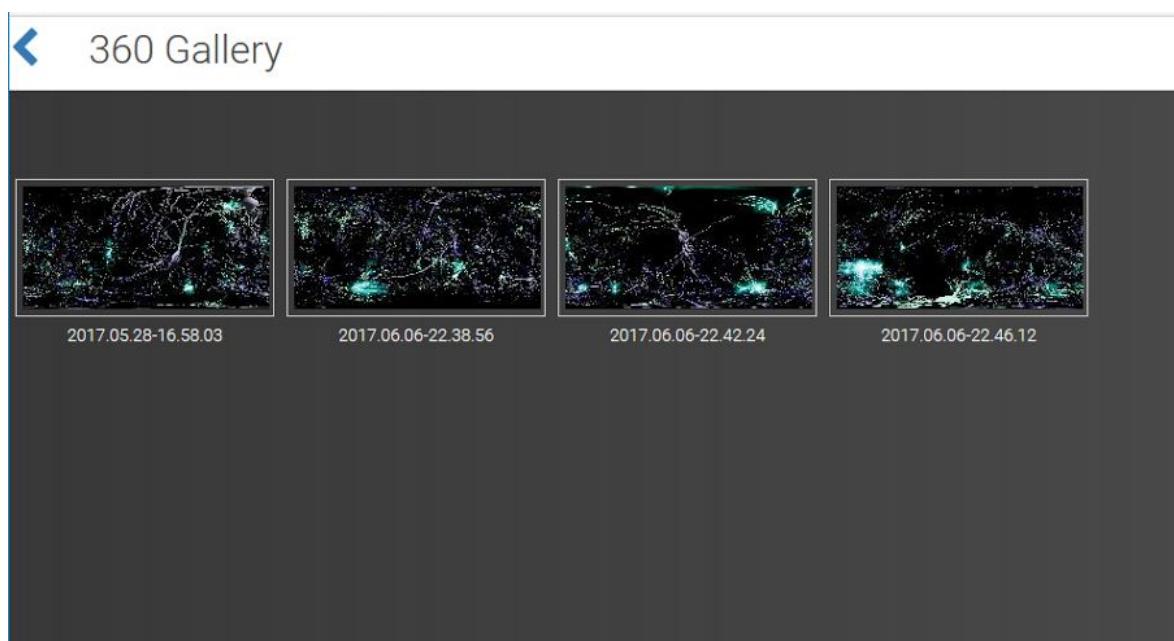


Figura 38: Galería 360

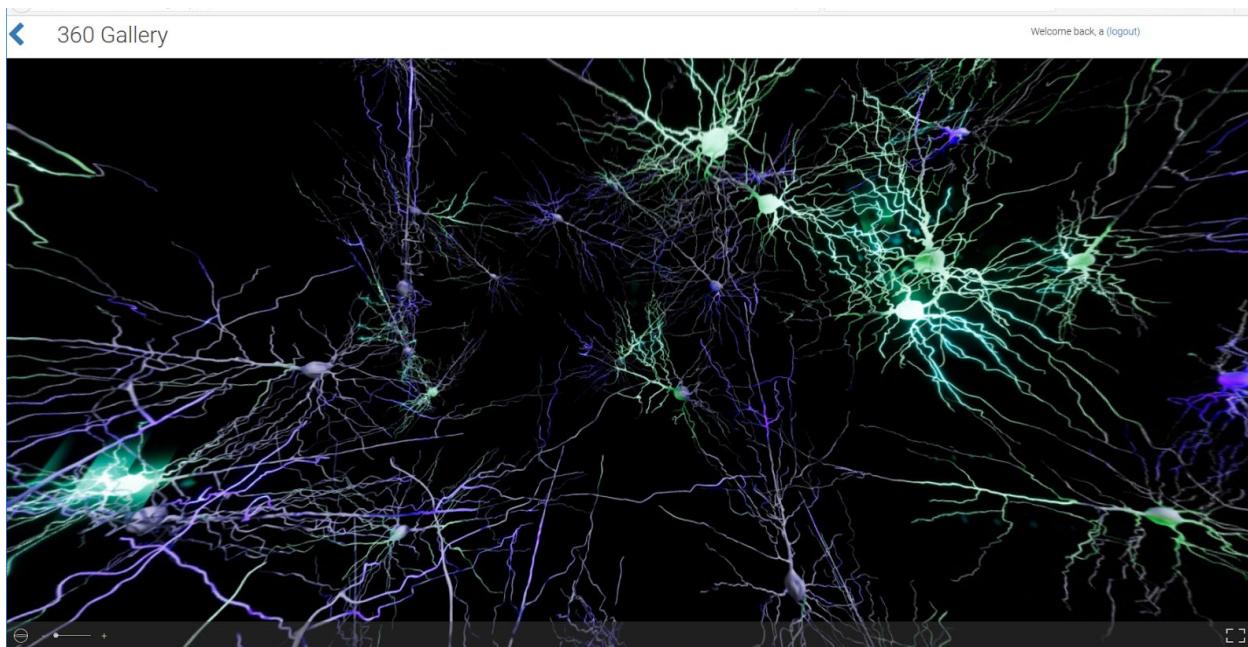


Figura 39: Visor 360

5.5. Visor de Chats

El sistema de visualización de chats recupera un listado de los chats almacenados en el sistema de persistencia divididos por el identificador de sesión a la que pertenecen. Además dispone de un buscador de mensajes para facilitar la recuperación de información.

En la Figura 40 se muestra el listado de chats por sesiones y el buscador.

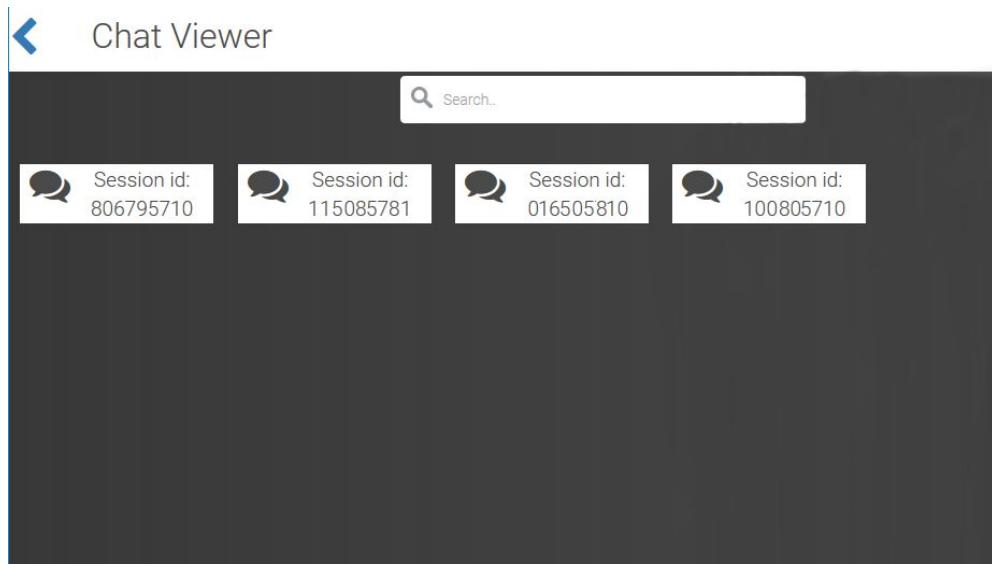


Figura 40: Chat viewer

En la Figura 41 se muestra una búsqueda.

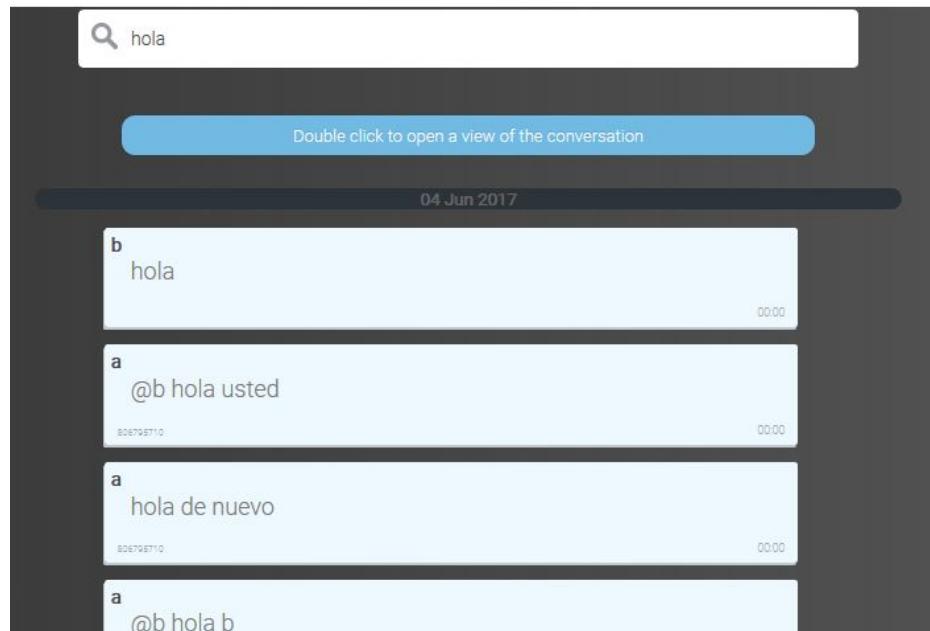


Figura 41: Muestra de búsqueda de chat

En la figura 42 se muestra la representación de un chat recuperado desde el sistema de persistencia.

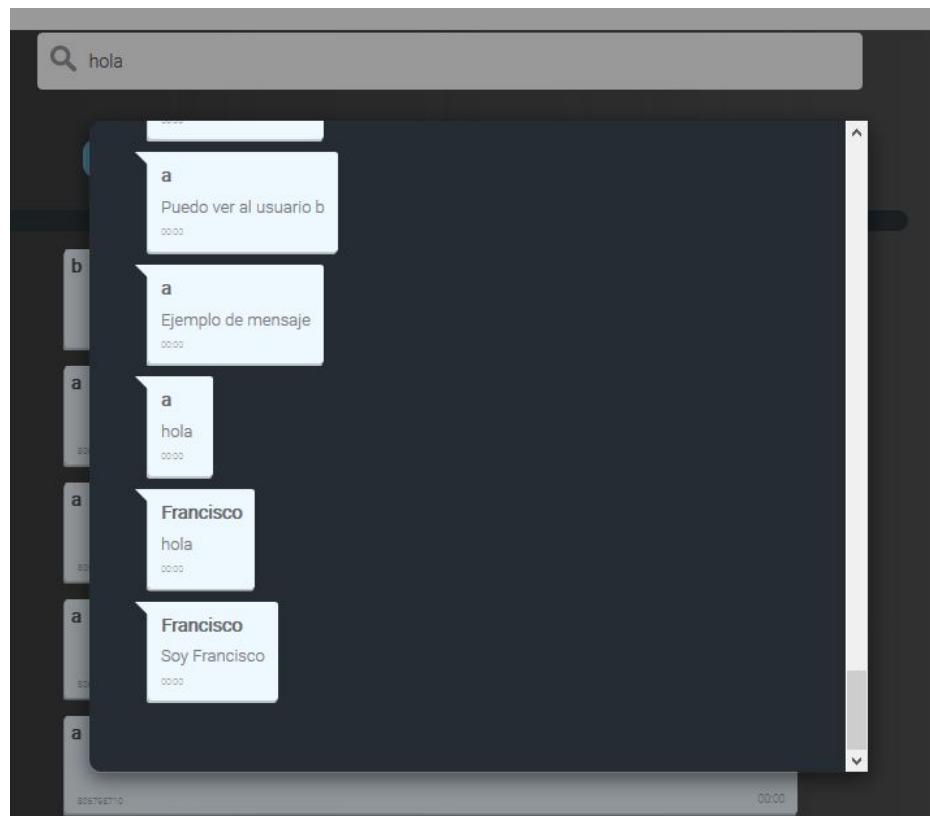


Figura 42: Muestra de chat recuperado

6. Resultados

Como resultado de la realización de este proyecto se ha obtenido una infraestructura o framework que permite colaborar a distintos investigadores en el análisis de escenas neuronales y extraer la información que se genera de forma colaborativa a través de una API REST que ofrece recursos los recursos tanto para las herramientas de esta infraestructura como a futuras herramientas o herramientas de terceros que necesiten hacer uso de esta información. Además a modo de ejemplo sobre las posibilidades que ofrece usar un servicio web RESTful, se desarrolla como parte de la infraestructura una interfaz web que extrae información útil y la presenta de una forma intuitiva y fácil de manejar, permitiendo obtener datos de forma rápida sin necesidad de utilizar o tener disponible la herramienta de visualización.

El prototipo anterior realizado también para la iniciativa Human Brain Project, investigaba la capacidad de procesamiento y cómo podría usarse el motor de juegos Unreal Engine para cargar escenario neuronales muy amplias.

La herramienta desarrollada para el análisis de escenas neurocientíficas de forma colaborativa en este proyecto utiliza el mecanismo de carga de escenas neuronales existente y además implementa la capacidad de representar los impulsos nerviosos de las neuronas a través de la información obtenida de una escena, dando un paso más en el desarrollo de la visualización y por tanto se proporciona mucha más información a los futuros análisis de forma visual. Dentro de los aspectos colaborativos, se introduce todo un sistema que permite la creación de sesiones cargando una escena neurocientífica en la que cualquier otro usuario podría unirse y ver la información sincronizada con el resto de usuarios, así mismo en conjunto con el sistema de sesiones, se implementan diferentes métodos para establecer información dentro de la escena y ver lo mismo que otro usuario. Por último los sistemas de comunicación por voz y mensajes a través de chat, completan en gran medida una herramienta útil para el análisis colaborativo a través del uso de técnicas de visualización de datos.

Es necesario destacar que no ha sido posible implementar completamente un sistema que permita a los usuarios repetir los pasos seguidos ya que el desarrollo se ha visto limitado por el estado experimental de esta característica dentro del motor Unreal Engine.

La información utilizada para la generación de escenas ha sido producida sintéticamente para simplificar los procesos de desarrollo, pero no cabe duda de que esta infraestructura es totalmente funcional para su uso en análisis profesionales de los cuales se puede extraer información útil.

7. Conclusiones

El uso del motor Unreal Engine para el desarrollo de herramientas de visualización científica tiene un gran potencial. Sin embargo, la curva de aprendizaje está bastante marcada en gran medida por la poca documentación y proyectos de ejemplo disponibles, obteniendo gran parte de la información de los comentarios en el código del propio UDK, el cuál es público y permite su modificación por parte de los desarrolladores. El uso de la herramienta Github ha sido totalmente obligatoria para obtener esta información ya que aunque el código es libre, se tiene un gran control sobre su uso.

Todo ello a ralentizando en gran medida el proceso de toma de contacto con las herramientas de desarrollo de un proyecto. No obstante el desarrollo posterior utilizando esta tecnología ha resultado más sencillo de lo esperado ya que las ventajas obtenidas por el uso esté motor son considerables:

- Arquitectura Cliente-Servidor estándar que permite generar sesiones de trabajo con multitud de funcionalidades y un dinamismo que no sería posible igualarse con un sistema alternativo de colaboración como una videollamada
- Lenguaje C++. Durante muchos años, C++ ha supuesto la única opción para el desarrollo de aplicaciones de alto rendimiento de cualquier tipo, incluyendo juegos. Como resultado, hubo inversiones significativas en motores de juegos y bibliotecas escritas en este lenguaje. Además la posibilidad de compilar directamente este lenguaje a distintos sistemas operativos permite una gran expansión de los productos desarrollados.
- Mecanismo de programación por bloques enfocado en el diseño. Permite que diferentes artistas también puedan realizar pequeños algoritmos y funciones sin tener grandes conocimientos de programación.
- Disponibilidad de últimas tecnologías como la realidad virtual.

Aplicar este potencial a la colaboración científica ha sido el desafío de este proyecto. Permitiendo que grupos de analistas geográficamente separados puedan unirse para abordar grandes escenas neurocientíficas, y a su vez puedan comunicarse de una forma totalmente dinámica. Además se ha tenido en consideración la existencia de posibles colaboradores que no necesitan ser expertos fomentando la aplicación didáctica de la herramienta.

Todo ello desarrollado sobre un motor de juegos, siendo pioneros en su uso para el análisis de escenas neurocientíficas y generando un entorno de colaboración.

Adicionalmente como parte de esta infraestructura se ha desarrollado un sistema de persistencia con un servicio web asociado, el cual se trata como una parte crítica del proyecto. Establece la capacidad de almacenar toda la información generada en el análisis colaborativo. Su implementación ha sido sencilla en comparación con la herramienta de análisis, al implementarse haciendo uso de tecnologías estándares para el desarrollo web y almacenamiento de la información.

Finalmente existen diversas líneas futuras sobre las que trabajar en este proyecto, cómo:

- El desarrollo de un sistema de repetición funcional con próximas versiones del motor.
- El establecimiento de sistemas seguros de conexión en red a través de https.
- Creación de un subsistema administrador de conexiones y un servidor externo que permita una mayor eficiencia en la gestión de escenas neuronales, sin depender de un cliente y su capacidad de conexión.

Según se ha podido ver durante el proceso de desarrollo de la herramienta, todos estos problemas tienen cobertura en el motor pero las limitaciones de tiempo han impedido su desarrollo en el proyecto actual.

A través de esta infraestructura, se ha fijado un camino futuras expectativas y posibilidades que permitan la continuidad y focalización en objetivos futuros sin tener que volver a pasar por el difícil camino recorrido.

Bibliografía

- [1] Trevor Nath. Investing in Video Games: This Industry Pulls In More Revenue Than Movies, Music, Nasdaq, 2016 [Online].
<http://www.nasdaq.com/article/investing-in-video-games-this-industry-pulls-in-more-revenue-than-movies-music-cm634585>
- [2] Evan Rawn, Unreal Visualizations: 3 Pros and 3 Cons of Rendering with a Video Game Engine, Archdaily [Online].
<http://www.archdaily.com/607849/unreal-visualizations-3-pros-and-3-cons-of-rendering-with-a-video-game-engine>
- [3] Unreal Engine Awards [Online]. <https://www.unrealengine.com/awards>
- [4] Tamara Munzner. Visualization Analysis and Design. A K Peters Visualization Series, CRC Press, 2014.
- [5] Georgia Institute of Technology. Scientific Visualization Tutorials. [Online].
<http://www.cc.gatech.edu/scivis/tutorial/tutorial.html>
- [6] Gregory M. Nielson, Hans Hagen, Heinrich Müller. Scientific visualization: overviews, methodologies, and techniques. Los Alamitos, Calif., IEEE Computer Society Press, 1997.
- [7] Petra Isenberg, Niklas Elmquist, Jean Scholtz, Daniel Cernea, Kwan-Liu Ma, and Hans Hagen. Collaborative Visualization: Definition, Challenges, and Research Agenda. *Information Visualization Journal (IVS)*, Special Issue on Information Visualization: State of the Field and New Research Directions, October 2011.
- [8] Roy D. Pea. Learning Science through Collaborative Visualization over the Internet. Nobel Symposium: Virtual museums and public understanding of science and culture, May 26-29, 2002, 2002, Stockholm, Sweden.
- [9] Vatika Sharma, Meenu Dave, SQL and NoSQL Databases, International Journal of Advanced Research in Computer Science and Software Engineering, 2012.
- [10] Rick Cattell, Scalable SQL and NoSQL data stores, ACM SIGMOD Record, 2010.
- [11] Lembo, Phil. eldapo: Let's get a real database. [Online].
<https://web.archive.org/web/20120305152003/http://eldapo.blogspot.com/2007/05/lets-get-real-database.html>, 2013.
- [12] Roy Thomas Fielding, Chapter 5: Representational State Transfer (REST), Architectural Styles and the Design of Network-based Software Architectures, University of California, 2000.

- [13] Actors. [Online].
<https://docs.unrealengine.com/latest/INT/Programming/UnrealArchitecture/Actors/>
- [14] UObject. [Online].
<https://docs.unrealengine.com/latest/INT/API/Runtime/CoreUObject/UObject/UObject/index.html>
- [15] Pawn. [Online].
<https://docs.unrealengine.com/latest/INT/Gameplay/Framework/Pawn/>
- [16] Character. [Online].
<https://docs.unrealengine.com/latest/INT/Gameplay/Framework/Pawn/Character/>
- [17] PlayerController. [Online].
<https://docs.unrealengine.com/latest/INT/Gameplay/Framework/Controller/PlayerController/>
- [18] APlayerState. [Online].
<https://docs.unrealengine.com/latest/INT/API/Runtime/Engine/GameFramework/APlayerState/index.html>
- [19] Game Mode and Game State. [Online].
<https://docs.unrealengine.com/latest/INT/Gameplay/Framework/GameMode/>
- [20] APlayerCameraManager. [Online].
<https://docs.unrealengine.com/latest/INT/API/Runtime/Engine/Camera/APlayerCameraManager/index.html>
- [21] AHUD. [Online].
<https://docs.unrealengine.com/latest/INT/API/Runtime/Engine/GameFramework/AHUD/index.html>
- [22] UWorld. [Online].
<https://docs.unrealengine.com/latest/INT/API/Runtime/Engine/Engine/UWorld/index.html>
- [23] Garbage Collection & Dynamic Memory Allocation. [Online].
https://wiki.unrealengine.com/Garbage_Collection_%26_Dynamic_Memory_Allocation
- [24] Cedric ‘eXi’ Neukirchen, ‘Unreal Engine 4’ Network Compendium, cedric.bnslv.de.
- [25] Carlos B. Moreno, Transmisión sináptica-canales de calcio y liberación de neurotransmisor, Revista Ciencias de la Salud, 2010.
- [26] Gavin Costello, Capturing Stereoscopic 360 Screenshots and Movies from Unreal Engine 4, 2016 [Online].
<https://www.unrealengine.com/en-US/blog/capturing-stereoscopic-360-screenshots-video-movies-unreal-engine-4>

[27] Replay System [Online]. <https://docs.unrealengine.com/latest/INT/Engine/Replay/>