eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Computación

Trabajo de Fin de Grado

Realización de una APP de monitorización neuronal

Autor/a

Pablo Portalo Indias

2018

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Computación

Trabajo de Fin de Grado

Realización de una APP de monitorización neuronal

Autor/a

Pablo Portalo Indias

Directore/a(s)

Manuel Graña Romay

Resumen

Este proyecto ha sido desarrollado con el fin de crear una aplicación web capaz de visualizar la actividad neuronal de forma práctica y sencilla. Para ello, se presenta un modelo 3D del cerebro humano, donde se plasman las señales eléctricas de los diferentes puntos del cerebro, recogiendo la información con sensores EEG inalámbricos, más concretamente con Emotiv EPOC+.

Índice general

Índice de figuras

Índice de tablas

1. CAPÍTULO

Introducción

En 1875, el científico Richard Caton utilizó un galvanómetro[1] con el fin de observar los impulsos eléctricos en la superficie del cerebro de ratones y monos vivos. Años más tarde, el alemán Hans Berger decidió retomar los estudios de Caton y fue más allá, tratando de encontrar estos impulsos eléctricos en el cerebro humano.

En 1924, Berger colocó unos electrodos sobre el cuero cabelludo de un paciente, e instalándolos en un potente galvanómetro pudo ver que este detectaba actividad eléctrica, siendo el primero en registrar un electroencefalograma (EEG) de un cerebro humano[Haas, 2003].

No fue hasta 1929 cuando publicó su obra *"Ueber das Elektrenkephalogramm des Menschen"* (en castellano "El electroencefalograma del hombre") [Leonardo, 2002]. En ella hace referencia a dos tipos de ritmos cerebrales que observó en sus estudios: las ondas alpha, aquellas de mayor voltaje y menor frecuencia; y las ondas beta, cuyo voltaje es menor y su frecuencia mayor.

1.1. La electroencefalografía

El cerebro humano está compuesto por células llamadas neuronas. Estas neuronas utilizan impulsos eléctricos para comunicarse y transferir señales entre ellas. Cuando se transmiten estas señales en cantidades grandes se puede detectar cierta cantidad de actividad eléc-

[1] Un galvanómetro es un instrumento que se usa para detectar y medir la corriente eléctrica.

trica producida por ellas. Para la detección de esta actividad se utiliza un equipamiento médico llamado electroencefalograma.

La electroencefalografía consiste en la medición de esa actividad eléctrica cerebral mediante el uso de electrodos o sensores que se adhieren al cuero cabelludo de las personas. Estos sensores son activados a través de corrientes eléctricas durante la excitación neuronal. Dada la alta atenuación generada por el cráneo y el cuero cabelludo, se utiliza el microVoltio (μV) como unidad de medida, lo que hace necesaria una amplificación de la señal para un análisis adecuado [Muñoz, 2014].

1.2. Ritmos cerebrales

Los ritmos cerebrales son definidos como ondas regulares a lo largo del tiempo. Estas ondas se caracterizan por su frecuencia, localización y asociación con varios aspectos del funcionamiento del cerebro. Dependiendo del estado del cerebro (sueño, vigilia, estado de coma...) se pueden producir un tipo u otro de ritmos.

Esta actividad se clasifica según las bandas de frecuencias que ocupan, denominándose con las letras griegas alpha (α), beta (β), delta (δ), tetha (θ) y gamma (γ); sin embargo, con el paso del tiempo se han descubierto nuevos ritmos que en algunos casos comparten estas bandas frecuenciales, pero que se distinguen unas de otras por la localización o funciones del cerebro a las que pertenecen. Un ejemplo de esto son las ondas mu (γ), las cuales pertenecen a la misma banda que la frecuencia Alpha [Cicchino, 2013].

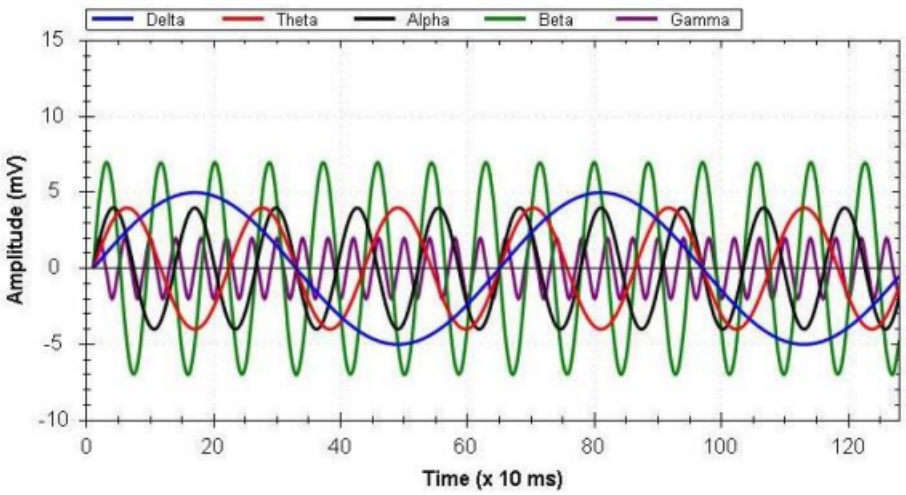

Figura 1.1: Representación de los distintos ritmos cerebrales [Larsen, 2011].

A continuación, se detallan las características de los distintos ritmos cerebrales:

Onda	Banda	Amplitud
Delta	0,5 - 3,5 Hz	20 - 200 μV
Thetha	4 - 7 Hz	20 - 100 μV
Alpha	8 - 13 Hz	20 - 60 μV
Mu	8 - 13 Hz	<50 μV
Beta	13 - 30 Hz	2 - 20 μV
Gamma	>30 Hz	5 - 10 μV

Tabla 1.1: Datos de los distintos ritmos cerebrales.

- **Ritmo delta** (δ): este ritmo se da únicamente durante el sueño profundo en individuos sanos adultos. Si se detectase en una persona despierta, sería indicativo de que hay alguna irregularidad en el cerebro.
- **Ritmo theta** (θ): aunque esta frecuencia se encuentre mayormente en niños, se pueden detectar en estado de adormecimiento y sueño en adultos. Estas ondas, se localizan en lóbulo temporal.
- **Ritmo alpha** (α): es habitual encontrarlo en la mayoría de los adultos que tienen los ojos cerrados o están en reposo visual, despiertos con un estado mental tranquilo y en reposo. El ritmo alpha es bloqueado o atenuado por la atención, especialmente visual y esfuerzo mental o físico. Durante el sueño profundo también desaparecen las ondas alpha. Este ritmo se puede observar principalmente en la zona posterior de la cabeza, en el área occipital, parietal y la región temporal posterior.
- **Ritmo mu** (μ): aunque su frecuencia y amplitud son similares a las del ritmo α, sus características topográficas y fisiológicas son distintas. Una de sus principales características es que desaparece con la realización de movimientos, estímulos táctiles y visuales. También puede llegar a desaparecer con la imaginación o preparación de un movimiento. El ritmo μ se localiza en la corteza motora primaria.
- **Ritmo beta** (β): este ritmo se puede encontrar en individuos en estado de concentración mental. Aparecen principalmente cuando la persona está realizando una actividad mental o física. Se detecta en la región central y frontal del cuero cabelludo, cerca o sobre la corteza motora primaria.
- **Ritmo Gamma** (γ): esta actividad se presenta como respuesta a estímulos sensoriales, como sonidos contundentes o luces intermitentes. Se puede encontrar a lo

largo de la corteza cerebral, localizándose en mayor proporción en la zona frontal y la central.

1.3. Artefactos

Cuando se realizan registros de actividad neuronal mediante EEG, es común encontrarnos con los llamados artefactos. Los artefactos son ondas externas a la actividad cerebral que se cuelan en el registro EEG y pueden conducir a interpretaciones erróneas.

La amplitud o voltaje de las señales EEG, no son muy superiores a las interferencias (a veces son incluso inferiores), por lo que corremos el riesgo de amplificar estos ruidos provenientes de ondas de radio, artefactos electrostáticos por el movimiento de cables, etc.

Aunque muchos artefactos puedan provenir de elementos externos, los más perjudiciales suelen tener su origen en el propio paciente, son los conocidos como artefactos fisiológicos. Las interferencias electromiográficas (características por su alta frecuencia), son un ejemplo de estos artefactos. Éstas suelen producirse por la contracción de los músculos faciales, por lo que movimientos en los maxilares producen artefactos en los lóbulos temporales, los movimientos en los músculos frontales introducen ruido en los sensores de los lóbulos frontales, etc. Para evitar o intentar disminuir este tipo de artefactos, es necesaria la relajación del sujeto y evitar la realización de tareas que requieran movimiento facial [Velasco, 2013].

Un ejemplo más de artefactos fisiológicos, son los originados por parpadeos y movimientos oculares. La amplitud de estos es habitualmente mayor a la de la propia actividad neuronal, lo que puede afectar a los distintos electrodos posicionados a lo largo del cuero cabelludo.

1.4. Utilidades y aplicaciones del EEG

Desde el descubrimiento de los EEG, éstos han servido para poder estudiar y diagnosticar distintas anomalías en el cerebro humano. Ejemplo de las afecciones que se pueden monitorizar son la epilepsia, enfermedades como el Alzheimer, traumatismos craneales, etc. Otras posibles utilidades son la evaluación de problemas del sueño y para la detección del grado de actividad cerebral en personas en estado de coma.

Además de los usos tradicionales, se puede decir que la tecnología interfaz cerebro-computador está en auge. La idea principal de la interfaz cerebro-computador o BCI (*Brain Computer Interface*) es capturar los impulsos eléctricos recogidos por los EEG y traducirlos en acciones que son interpretadas y ejecutadas por un ordenador u otro dispositivo. Estos sistemas pueden realizar acciones tan simples como encender o apagar bombillas de nuestra casa, o desarrollarse con el fin de manejar maquinas tan complejas como sillas de ruedas [Minguez, 2008]. Por ello, estas aplicaciones abarcan desde el ámbito médico hasta el más lúdico.

1.5. Representación gráfica de un EEG

Al hablar de un electroencefalograma, estamos acostumbrados a imaginarnos la representación gráfica de tipo cartesiano en la que en las ordenadas (coordenada vertical), se representa la amplitud de la onda, y en las abscisas (coordenada horizontal) se representa el tiempo.

Figura 1.2: Representación clásica de un EEG. Fuente: Wikipedia.

Además de esta representación clásica del EEG, es factible una representación denominada cartográfica o de mapeo cerebral, la cual permite observar de forma más rápida y directa la actividad neuronal en la zona del cuero cabelludo [Velasco, 2013]. Esta representación, consiste en traducir en distintos colores (o tonos de gris) la amplitud o frecuencia promedio registrada durante un intervalo de tiempo en distintas posiciones de la cabeza del sujeto.

Aunque la técnica de representación mediante mapeo cerebral sea más atractiva por proporcionar una visualización más fácil de interpretar que los diagramas bidimensionales, es importante destacar que este sistema no proporciona más información que la representación tradicional.

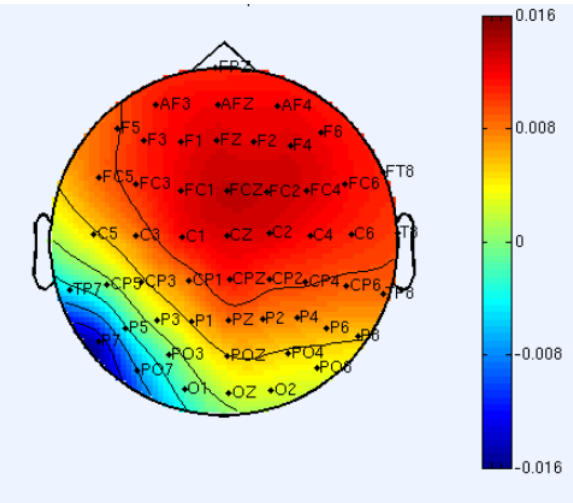

Figura 1.3: Representación cartográfica obtenida en EEGLab.

Uno de los inconvenientes del uso de esta visualización, es que puede ser inexacta en zonas alejadas de los electrodos, ya que zonas carentes de sensores, son representadas mediante aproximaciones calculadas con los canales cercanos. Esto hace que se recomiende un alto número de electrodos para una mayor precisión [Pivik et al., 1993].

2. CAPÍTULO

Documento de objetivos del proyecto

2.1. Objetivos del Proyecto

La principal motivación de este proyecto es la obtención de una herramienta con la que poder monitorizar la actividad del cerebro. De esta manera, se pretende contribuir al estudio y comprensión de las ondas neuronales en futuros desarrollos de BGI y/o aplicaciones médicas.

El objetivo que se busca alcanzar es obtener la información capturada por los sensores EEG *wireless* y plasmar la actividad neuronal en un modelo 3D del cerebro. Esta representación, se podrá mover libremente para visualizar de forma sencilla la actividad en los distintos puntos donde se encuentran los electrodos. Para todo esto, se utilizará el *headset* Emotiv EPOC+.

Objetivos de aprendizaje:

- Estudiar la puesta en marcha del EPOC+ y la utilización de Emotiv SDK.
- Investigar los métodos y funciones de la librería Three.js para la representación de un entorno 3D.
- Comprender el mapeo de texturas para su aplicación en objetos 3D.

Objetivos de diseño:

- Crear un servidor web para la obtención de los datos de los sensores neuronales en tiempo real.
- Desarrollar una aplicación para la representación y lectura de la actividad.
- Establecer un formato adecuado para el fichero de sesiones grabadas.
- Implementar una aplicación para la grabación del fichero de la actividad registrada.

2.2. Planificación

El proyecto se inicia el 5 de febrero, y según lo planificado finalizará el día 15 de julio. La fecha de solicitud de la defensa es el 24 de julio, por lo que se dejan días de margen para poder utilizarlos si fuera necesario. Además de estos días de margen, también se podrían realizar mejoras tanto en la implementación como en la documentación hasta el 6 de septiembre, fecha límite para subir la memoria a la plataforma digital ADDI.

La planificación se ha realizado de forma dinámica, por lo que se podrá modificar a medida que surjan imprevistos.

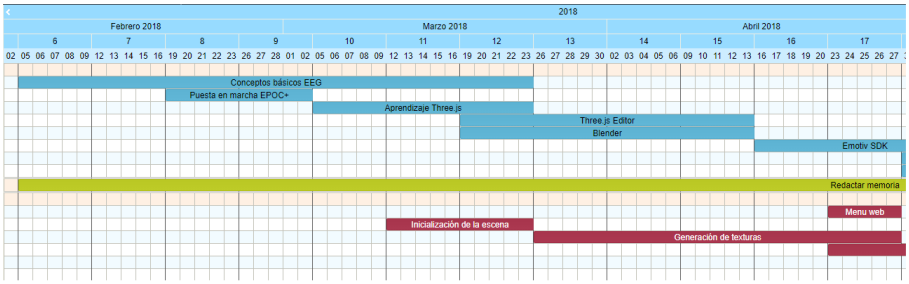

Figura 2.1: Calendario con la planificación realizado en Tom's Planner parte 1.

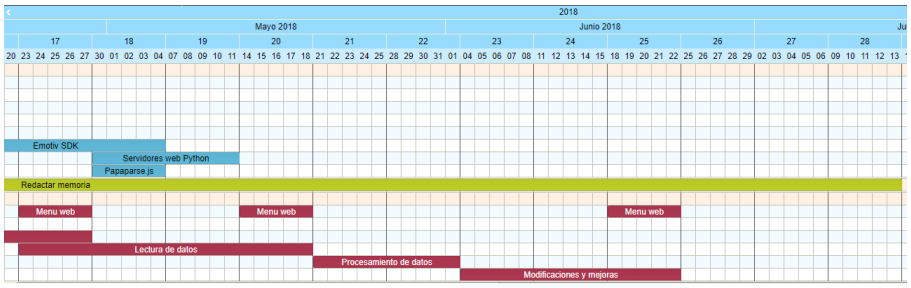

Figura 2.2: Calendario con la planificación realizado en Tom's Planner parte 2.

Como se puede observar, en la planificación se distinguen tres colores: el azul representa las tareas de aprendizaje; el granate la implementación de la aplicación y el color verde lima la redacción de la documentación.

2.3. Herramientas Utilizadas

En el siguiente apartado se exponen las distintas aplicaciones y herramientas utilizadas para el desarrollo del proyecto.

2.3.1. Emotiv EPOC+

Emotiv EPOC+ es un dispositivo basado en los sistemas electroencefalográficos que ha sido desarrollado por la empresa australiana Emotiv Systems. Este sistema cuenta con 14 canales EEG: 12 electrodos con los que medir la actividad neuronal en distintas posiciones del córtex cerebral y dos puntos de referencia.

Los electrodos están ubicados respetando el sistema internacional 10-20 para la colocación de los electrodos extracraneales. Los nombres de cada posición provienen del lóbulo en el que se encuentran: las letras señalan el área (Fp: prefrontal, F: frontal, C: central, P: parietal, T: temporal y O: occipital); mientras que los números designan el hemisferio (pares del derecho, impares del izquierdo). Los electrodos de la línea central se señalan con una “z”, por lo que Fz se encontraría frontalmente en la línea media [Velasco, 2013].

En la siguiente imagen se puede obserbar la ubicación de los electrodos en EPOC+:

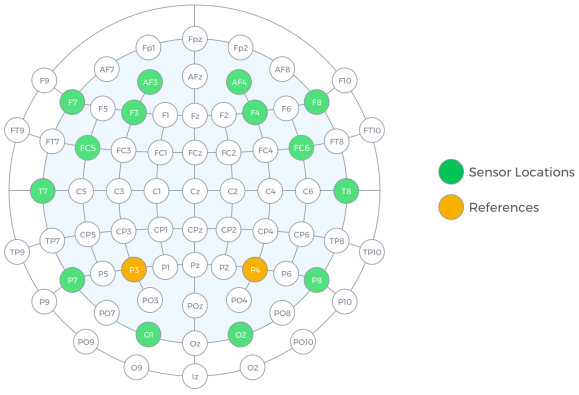

Figura 2.3: Posición sensores EPOC+.

2.3.1.1 Hardware y procedimiento de puesta en marcha

EPOC+ es un dispositivo EEG inalámbrico que contiene los distintos sensores en una especie de telaraña de plástico flexible que se ajusta a la cabeza de una forma rápida y sencilla, donde los sensores son desmontables para poder ser sustituidos con facilidad si fuera necesario. Este casco neuronal viene acompañado de un receptor USB, necesario para sincronizar el ordenador con el sistema EEG.

Figura 2.4: Emotiv EPOC+.

Una vez cargado el dispositivo mediante el cable USB, su puesta en marcha es sencilla:

1. Hidratar los sensores con disolución salina. Es importante que los sensores estén bien húmedos para un mejor contacto.
2. Instalar los sensores en el dispositivo EGG.
3. Introducir el receptor USB en el ordenador.

Figura 2.5: Receptor USB y sensores.

4. Ajustar el casco EEG. Las referencias para una buena colocación del EPOC+ es dejar un espacio de tres dedos entre las cejas y los sensores frontales, y ajustar los sensores de referencia en el dónde comienza el hueso detrás del lóbulo de la oreja.

Realizados estos pasos, se puede empezar a hacer pruebas en nuestro equipo con los programas gratuitos que se pueden descargar desde la web de Emotiv.

2.3.1.2 Emotiv SDK

Emotiv cuenta con un SDK disponible para diferentes plataformas de programación descargable desde el repositorio de Emotiv:GitHub. Mediante este SDK basado en una librería escrita en C++, podremos procesar con facilidad las señales recibidas por EPOC+ y reconocer las expresiones faciales, estados anímicos y pensamientos cognitivos del usuario. Además, cuenta con distintos ejemplos escritos en distintos lenguajes de programación.

2.3.2. Three.js

Three.js es una biblioteca escrita en lenguaje de programación JavaScript. Está orientada a la creación y visualización de gráficos animados por ordenador en 3D en un navegador web, que permite ser utilizada en conjunción con el elemento *canvas* de HTML5, SVG o WebGL. Su código fuente se puede encontrar en el repositorio Three.js:GitHub.

Herramientas como esta, permiten al autor, crear complejas animaciones en 3D, listas para ser mostradas en el navegador. Esta librería evita el esfuerzo que se requeriría para el desarrollo de una aplicación independiente o tradicional con el uso de *plugins*.

En su página oficial se puede encontrar toda la documentación necesaria y más de 150 ejemplos con los que poder empezar a trabajar. Aunque la documentación cada vez sea más elaborada y completa (por la colaboración de su comunidad), gracias al gran número de ejemplos disponibles en la web, podemos realizar un aprendizaje a través de la lectura e interpretación del código, para comprender esta librería de una forma más práctica y sencilla.

El desarrollo de escenas muy complejas y sofisticadas puede resultar complicado, por lo que esta biblioteca también cuenta con un módulo de edición con el que poder crear escenas, añadir elementos, importar y/o exportar objetos, etc. de una forma visual.

A continuación, se exponen los distintos elementos que forman Three.js:

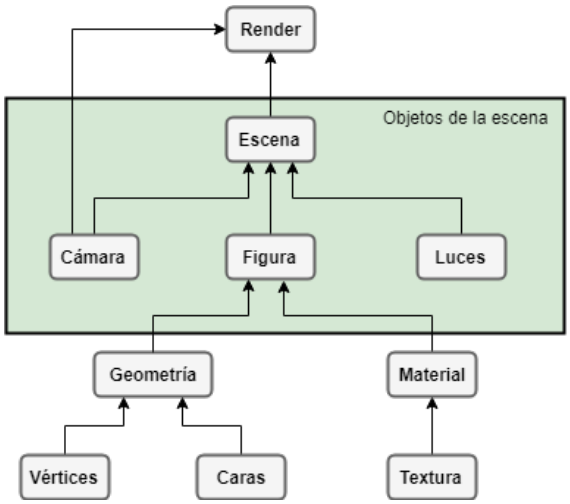

Figura 2.6: Gráfico de escena Threejs.

- **Canvas**: es un elemento HTML que permite la generación de gráficos de forma dinámica. Se accede a este objeto desde JavaScript, permitiendo elaborar escenas y animaciones en el navegador web.
- **Escena**: constituye un espacio tridimensional en el que se añaden objetos con los que se puede interactuar y por el que es posible moverse.
- **Render**: se refiere al elemento WebGL en el que la tarjeta gráfica se encarga de dibujar los objetos que se muestran en ese momento. Una escena 3D puede estar compuesta por diversos objetos, la imagen formada por el encuadre de la cámara en un momento especifico es lo que se denomina render.
- **Figura**: las figuras o *mesh* son aquellos elementos que se quieren representar dentro de la pantalla. Pueden ser objetos básicos como líneas, puntos, etc. o incluso objetos 3D creados en programas de edición que han sido importados en Three.js. Las figuras están compuestas por la geometría y por el material.
- **Geometría**: está compuesta por vértices y caras que forman una figura determinada. Estas se pueden crear manualmente, con librerías propias de Three.js (para crear cubos, círculos, . . .) o mediante loaders o importadores que recogen estos valores de objetos 3D modelados con programas externos.
- **Materiales**: representa la piel de las figuras, que sirve para colorear las distintas caras y configurar cómo la luz actúa en él. Three.js cuenta con diferentes clases de materiales predefinidos que se pueden asignar a las figuras. En caso de querer

crear un material personalizado, permite cargar texturas mediante imágenes, para así poder darle la apariencia deseada.

- **Cámara**: aunque una escena puede contener tantas cámaras como queramos, solo se puede utilizar una de ellas para renderizar la escena en cada momento. Como en un programa de televisión, existen varias cámaras de grabación, pero el realizador solo puede emitir la imagen de una de las cámaras. Estas cámaras se pueden mover libremente por la escena y enfocarla hacia los objetos que queremos mostrar.

- **Luces**: igual que en la vida real, para poder distinguir los colores de las escenas, necesitamos tener luz. Estas luces pueden ser de ambiente, direccionales, . . . y se pueden mover por la escena a nuestro gusto.

2.3.3. Blender

Blender es un programa multi plataforma, dedicado especialmente al modelado, iluminación, renderizado, animación y creación de gráficos 3D. Este programa, aunque en su origen era distribuido de forma gratuita sin el código fuente, finalmente pasó a ser software libre, lo que ha hecho que tenga una comunidad cada vez mayor y actualmente sea uno de los programas de modelado más conocidos.

Pese a que, a primera vista, su interfaz con una gran cantidad de elementos pueda llegar a intimidar, su gran comunidad hace fácil encontrar diversos tutoriales con los que empezar a trabajar rápidamente.

En este proyecto, junto al editor Three.js, Blender ha servido para ampliar los conocimientos respecto al mapeado de texturas y la edición de modelos 3D.

2.3.4. Tom's Planner

Tom's Planner es un software en línea que permite crear y compartir diagramas de Gantt. Es un programa muy intuitivo y fácil de utilizar con el que se ha creado la planificación del proyecto y ha permitido llevar a cabo el seguimiento del mismo.

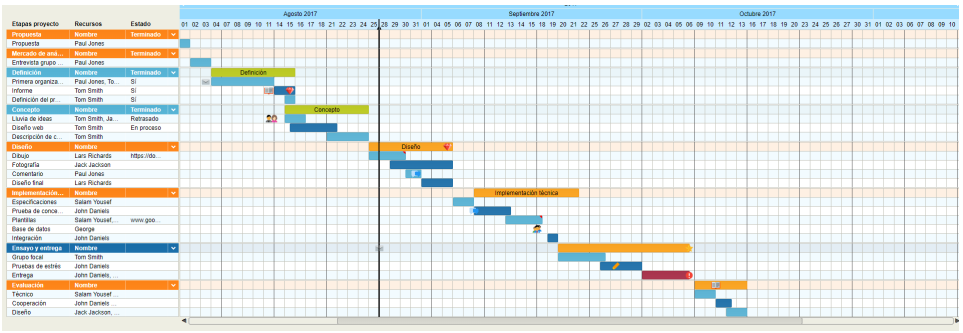

Figura 2.7: Imagen demostrativa de Tom's Planner.

2.4. Análisis de Riesgos y Factibilidad

En este apartado se enumerarán los riesgos a tener en cuenta a la hora de afrontar el proyecto, las medidas tomadas para intentar evitarlos, y en caso de que sucedan se explicará cómo solventarlo:

- **Dificultades en estimación del tiempo**: hay apartados del diseño del proyecto que no están definidos y es necesaria una formación y estudio previo para saber cómo se hará la implementación, por lo tanto, esto hace inviable una estimación correcta del tiempo. Para afrontar este riesgo, se añaden más horas de aprendizaje en la planificación.

- **Problemas al añadir funcionalidades en el código**: al trabajar con distintas librerías e ir añadiendo funcionalidades en diferentes fases a la aplicación, es fácil que a la hora de realizar cambios en el código surjan problemas con funcionalidades implementadas previamente. Para evitar la pérdida de tiempo en reestructurar el código para volver al estado previo, se hará un control de versiones, donde se irá guardando un *backup* de la aplicación a medida que se le añaden mejoras.

- **Pérdida de datos**: puede que por alguna razón se pierdan archivos del proyecto, como ficheros de código, imágenes, enlaces útiles, el documento de la memoria... Para evitarlo, se trabajará con una carpeta sincronizada en la nube, con el fin de tener una copia de todos los archivos actualizados en todo momento. Por otro lado, también se realizarán copias de seguridad en una memoria externa periódicamente.

3. CAPÍTULO

Desarrollo del proyecto

3.1. Descripción del proyecto

La propuesta se centra en la creación de una aplicación web en la que se mostrará la actividad neuronal, capturada por sensores EEG inalámbricos. Este proyecto pertenece principalmente al área de gráficos por computador, ya que, una vez controlada la lectura de datos de los sensores, el objetivo final, es la visualizan en un modelo 3D del cerebro con la actividad representada mediante colores, donde la actividad baja se representa con colores fríos y una alta actividad con colores cálidos.

La aplicación que vamos a desarrollar deberá contemplar las siguientes opciones para la representación de datos:

- **En directo**: la aplicación web lee la información de los sensores y muestra los datos obtenidos en tiempo real.
- **Sesión grabada**: se creará una aplicación con la que grabar sesiones o capturas de la actividad neuronal que posteriormente se podrán cargar en la aplicación web para poder visualizarlas.

Para la captura de datos se utilizará el controlador Emotiv EPOC+, junto a su SDK con licencia gratuita.

Tanto si la actividad se lee de un fichero externo, como si se lee en tiempo real, el funcionamiento del programa será el siguiente:

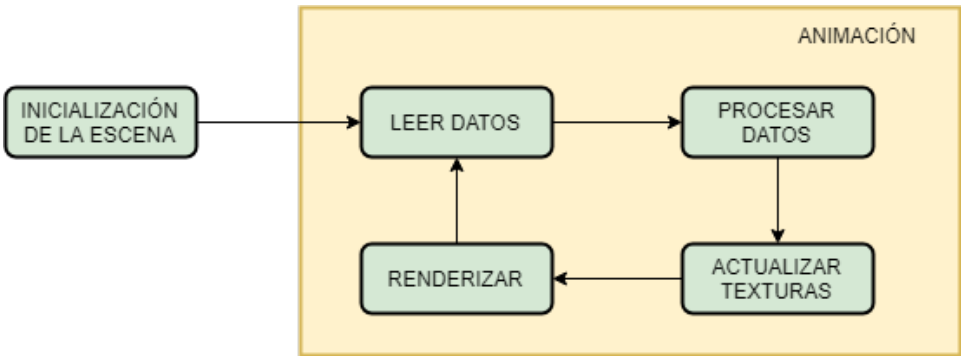

Figura 3.1: Diagrama de funcionamiento de la aplicación.

3.2. Inicialización de la escena

El primer paso a la hora de realizar la aplicación web ha sido la inicialización de la escena, es decir, la creación del espacio tridimensional donde se visualizará el cerebro 3D. En esta escena, se añadirán controles con los que poder rotar el cerebro y hacer zoom sobre él.

Aunque Three.js cuenta con un editor desde el que se pueden crear y exportar escenas para posteriormente cargarlas con funciones javascript, se ha optado por realizar la implementación completa mediante código.

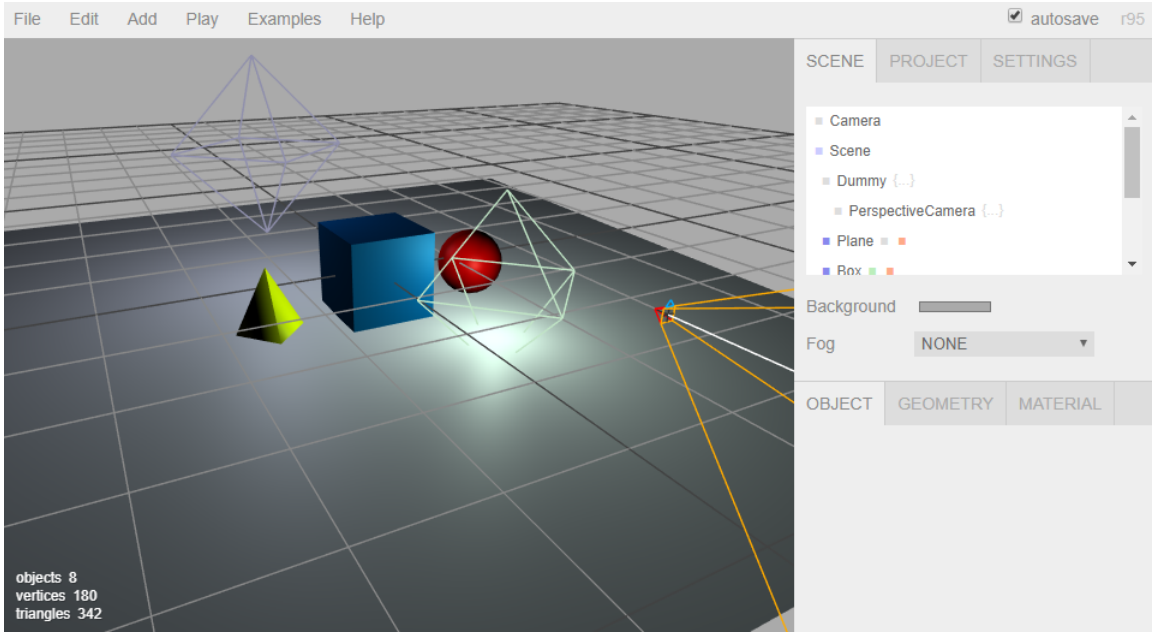

Figura 3.2: Editor Threejs.

Para la inicialización de la escena se ha utilizado la librería Three.js, además de dos librerías auxiliares incluidas también dentro del proyecto Three.js:

- **Three.js**: esta es la librería principal que permite el desarrollo de aplicaciones web three.js.
- **OBJLoader.js**: el modelo 3D que se cargará es de formato OBJ. Esta librería se encargará de la lectura de nuestro OBJ (el cerebro humano) y lo convertirá en un *mesh* o figura.
- **OrbitControls.js**: para una visualización correcta de la actividad cerebral, es necesario añadir funciones de control de movimiento a la aplicación. Para ello se utilizará esta librería que permite orbitar alrededor de un *target*, que en este caso será el cerebro humano.

Añadidas estas librerías, se procede a la creación de la escena, a la cual se le añaden los siguientes elementos:

- **Cámara**: se crea una cámara en perspectiva, a la que se le asignan los controles *OrbitControl* para permitir el movimiento alrededor del cerebro 3D.
- **Luces**: para poder visualizar de forma correcta los objetos de la escena se asigna un color de fondo gris claro a la escena, además de dos luces:
  - **AmbientLigth**: aporta claridad a la escena, como si de la luz natural del sol se tratase.
  - **PointLigth**: funciona como una bombilla, es una luz circular que alumbra en todas las direcciones. Se le asigna a la propia cámara, para que se mueva la luz junto con el movimiento de la cámara.

Una vez tenemos la escena iluminada con su cámara, se procede a la carga de los objetos. Aunque en la aplicación parezca que se ha cargado un único objeto (el cerebro), realmente son dos, ya que cada hemisferio se ha cargado de forma individual.

Al haber utilizado objetos con formato OBJ, se utiliza la clase *OBJLoader* para añadirlos a la escena. A estos objetos se les asignará una textura inicialmente azul, que representa una actividad nula. Al material, se le asigna la propiedad *needsUpdate*, necesaria para la actualización de la textura cuando hay cambios en la actividad neuronal.

Por último, se crea el render con el que podremos visualizar la escena, obteniendo un resultado como este:

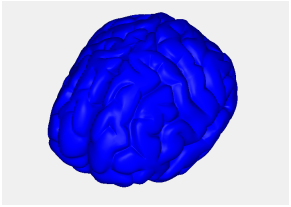

Figura 3.3: Resultado de la inicialización de la escena.

3.3. Mapeado de texturas en el modelo 3D

Para poder realizar el mapeado de texturas, se ha estudiado el funcionamiento de los objetos 3D y sus materiales con los editores Blender y Three.js. Una vez comprendido el concepto del mapeado UV y realizar distintas pruebas, se ha procedido al desarrollo de una función Javascript con la que representar estas texturas.

3.3.1. El mapeado UV

El mapeado de texturas define la forma en la que una textura se situará sobre el objeto en el momento de proyectarse. Estas texturas se representan como imágenes bidimensionales, mientras que los objetos son tridimensionales. Esta forma de representar la imagen plana en una figura tridimensional puede hacer que se observen ciertas imperfecciones o deformaciones, por lo que se utilizará la técnica de mapeado UV para evitar, en la medida de lo posible, estas imperfecciones.

Figura 3.4: Ejemplo de objeto 3D con su textura.

El mapeado UV (*UV mapping*) se puede definir como el proceso de desenvolver un modelo 3D para generar un mapa, que representa la malla del modelo en un mapa bidimensional. Este mapa, será utilizado como referencia para determinar cómo se debe colocar la textura en la malla. Por ejemplo, cuando queremos envolver un regalo utilizamos papel de regalo (un objeto bidimensional) con el que envolver el regalo (tridimensional), el mapa UV sería el patrón que nos indica como doblar este papel para ajustarlo a nuestro regalo.

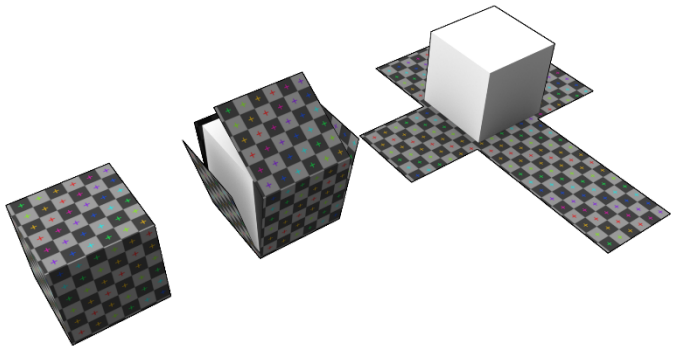

Figura 3.5: Representación del mapa UV.

El nombre UV hace referencia a las coordenadas del mapa 2D, donde se utilizan U y V para referirse a las coordenadas X e Y respectivamente, con el fin de evitar confusión entre los ejes del espacio 3D y los del mapa UV.

3.3.2. Generación del mapa UV

El objeto que se ha utilizado para la representación de la actividad cerebral, ha sido descargado de la biblioteca gratuita de objetos 3D Clara.io. Esta biblioteca permite la descarga de los objetos en distintos formatos, pero finalmente se optó por la descarga de éste en formato BLEND para realizar pruebas en Blender y formato OBJ para las pruebas en Three.js Editor.

3.3.2.1 Mapeo de texturas en Blender

Blender es una herramienta de edición 3D profesional, con la que se ha podido generar el mapa UV y comprender cómo generar las texturas para la representación de la actividad neuronal. El mapa UV, se ha generado como imagen de tamaño 1024x1024 pixels.

Como se puede observar en la figura 3.6, el exterior del cerebro está representado por la silueta A del mapa UV, mientras que la silueta B, representa el interior del hemisferio del cerebro. En la aplicación, se detecta la actividad únicamente en el cuero cabelludo, por lo que se omitirá la silueta B y siempre mostrará actividad nula en la parte interior del cerebro.

Figura 3.6: Representación del mapa UV en el objeto tridimensional.

Una vez llegado a este punto, es necesario ubicar cada electrodo dentro del mapa UV, para lo que se ha utilizado la herramienta *"Texture Paint"* de Blender. Esta herramienta permite dibujar con pinceles sobre el objeto tridimensional o sobre el mapa UV. En la figura 3.7 se puede observar el cerebro con una textura donde se representa la ubicación de cada sensor mediante puntos. Esto servirá para poder calcular las coordenadas de cada sensor en la textura y poder generar el mapa de actividad neuronal en las posiciones correctas.

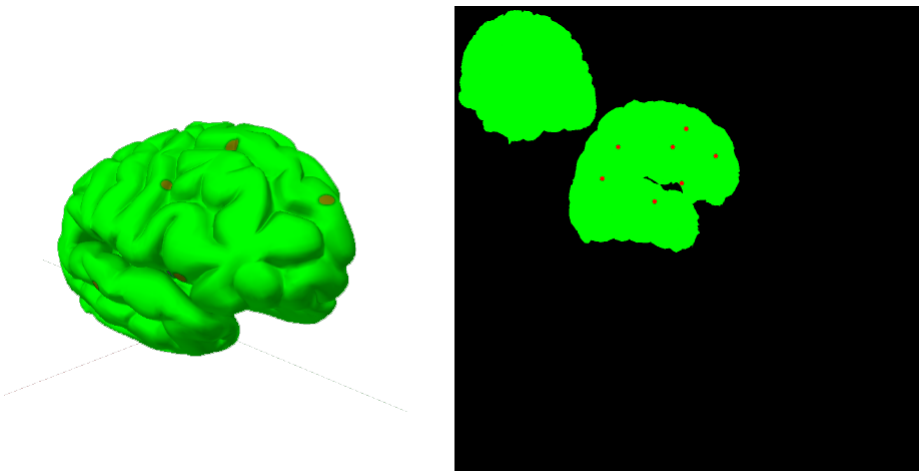

Figura 3.7: A la izquierda se obserava el resultado de haber pintado las distintas ubicaciones de los sensores. A su derecha la textura obtenida.

3.3.2.2 Mapeo de texturas en Three.js Editor

Esta herramienta se puede ejecutar tanto en local una vez descargado el proyecto completo de github, como desde la web https://threejs.org/editor/.

El editor de Three.js ha sido utilizado principalmente para realizar pruebas con las texturas generadas mediante Javascript y comprobar que se estaba realizando el mapeado de forma correcta. Aunque esto también se puede realizar en Blender, se ha utilizado el editor de threejs debido a su sencillez y a que permite simular la escena que se creará por código, añadiendo luces y cámara.

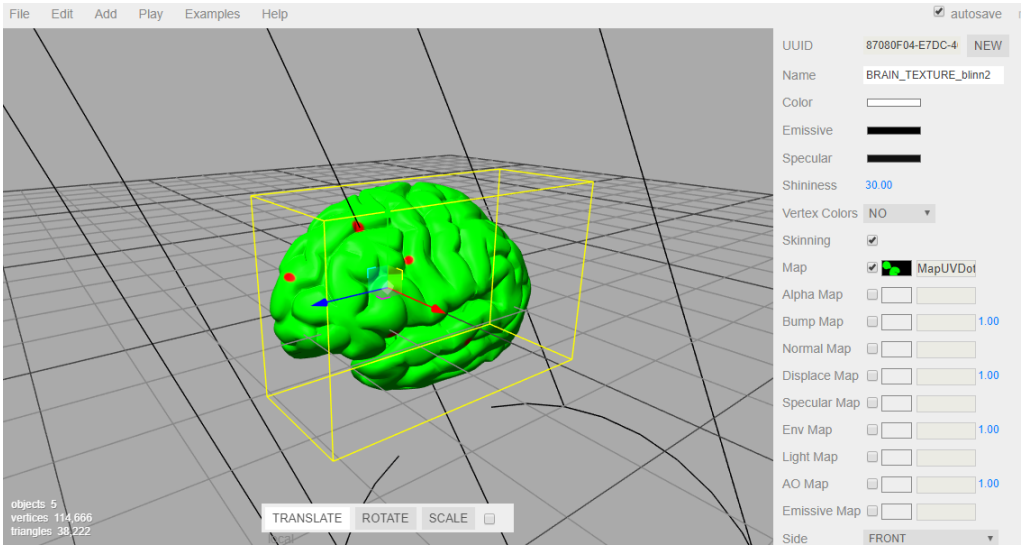

Figura 3.8: Cerebro con textura resultante de Blender en Threejs Editor.

Por otro lado, este editor también se ha utilizado para dividir el objeto descargado en dos hemisferios, exportando cada mitad en un *mesh* distinto en formato OBJ. Los dos objetos, son exactamente iguales pero volteados como si fueran el reflejo de un espejo, por lo que el mapa UV es idéntico para ambas figuras. De la misma forma, los sensores van posicionados en las mismas coordenadas en el hemisferio derecho e izquierdo del cerebro, por lo que la generación de texturas seguirá el mismo patrón en ambos casos, creándose una textura con la actividad de la izquierda y otra textura con la actividad de la derecha.

Al cargar los dos hemisferios en la escena con texturas diferentes quedaría algo similar a la imagen 3.9:

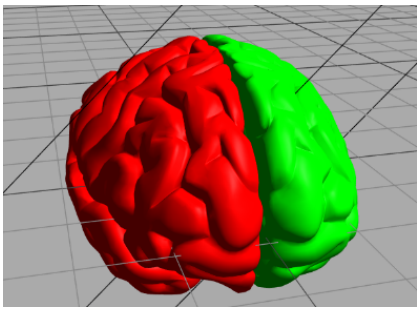

Figura 3.9: Cerebro en Three.js Editor. Cada hemisferio del cerebro está cargado con una textura, el hemisferio izquierdo en verde y el derecho en rojo.

3.3.3. Representación de actividad neuronal en la textura

Aunque en las pruebas y ejemplos encontrados en Three.js:examples cargasen las texturas de los objetos con imágenes externas, se observó que también era posible la carga de texturas mediante el elemento HTML *canvas*. Este lienzo permite generar gráficos dinámicamente en el propio navegador sin necesidad de crear imágenes externas para cada instante en el que se recibe actividad neuronal.

Para representar la actividad cerebral en las texturas, se ha partido de la librería simpleheat:GitHub. Esta sencilla librería genera una imagen en un elemento *canvas*, donde se representan en formato de mapa de calor (heat map en inglés) los valores introducidos como entrada a la clase simpleheat.

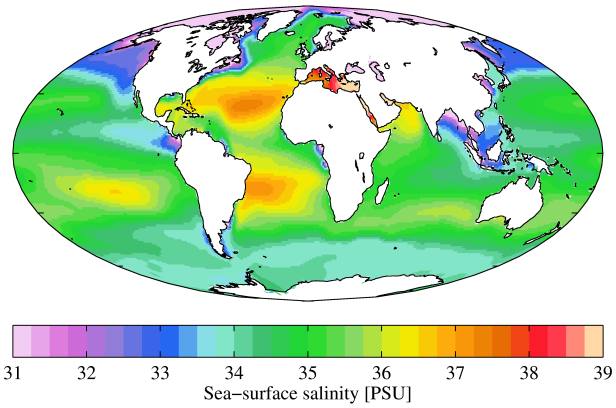

Figura 3.10: Ejemplo de heatmap. Fuente: Wikipedia.

El mapa de calor consiste en representar mediante colores la actividad, presión, tempera-

tura... en una superficie, donde un color cálido (rojo) representa un valor alto, y un color más frio (azul) un valor más bajo.

Antes de utilizar esta librería, se han llevado a cabo algunas modificaciones para adecuarla a nuestras necesidades y simplificar su uso en la aplicación. Inicialmente, para generar el mapa, era necesario introducir los valores de entrada en un array compuesto por la sucesión de valores de cada electrodo, manteniendo la estructura [coordenada x, coordenada y, actividad]. En este caso, conocemos siempre la ubicación de los distintos sensores, por lo que se ha modificado el código para que las coordenadas sean fijas y los valores de cada electrodo se introduzcan como parámetro en la inicialización. Para crear la imagen, es necesario crear un nuevo objeto simpleheat:

```
1 var heat = simpleheat(CanvasID).data(Coords).max(Max).valores(Values);
```

- **CanvasID**: corresponde al identificador del elemento *canvas* donde se generará la imagen.
- **Coords**: es un *array* con las coordenadas de cada electrodo. Estos se guardan en un archivo data.js para poder modificarlos fácilmente si fuera necesario. Representan los canales AF4, F8, F4, FC6, T8, P8, O2 en el hemisferio derecho y los canales AF3, F7, F3, FC5, T7, P7, O1 en el hemisferio izquierdo.
- **Max**: valor máximo de la actividad. Todos los electrodos que superen este valor se mostrarán en rojo. Este valor se podrá modificar desde la aplicación para ajustar la ganancia a cada banda de frecuencias.
- **Values**: valor recogido de cada sensor.

Una vez creada la clase, será necesaria la llamada al método *draw()* para su visualización. Éste sigue los siguientes pasos:

```
1 heat.draw();
```

1. Define las características de los distintos puntos que representarán la actividad de cada electrodo, para ello tendrá en cuenta el radio y desenfoque definidos.

2. Hace una lectura de todos los valores introducidos y genera un mapa en escala de grises. Cada electrodo será un circulo con las características definidas anteriormente, donde se le dará la opacidad resultante de dividir la actividad del sensor entre la actividad máxima (*Max*), pudiendo ser como máximo valor 1. El mapa final será el resultante de dibujar cada uno de los electrodos.

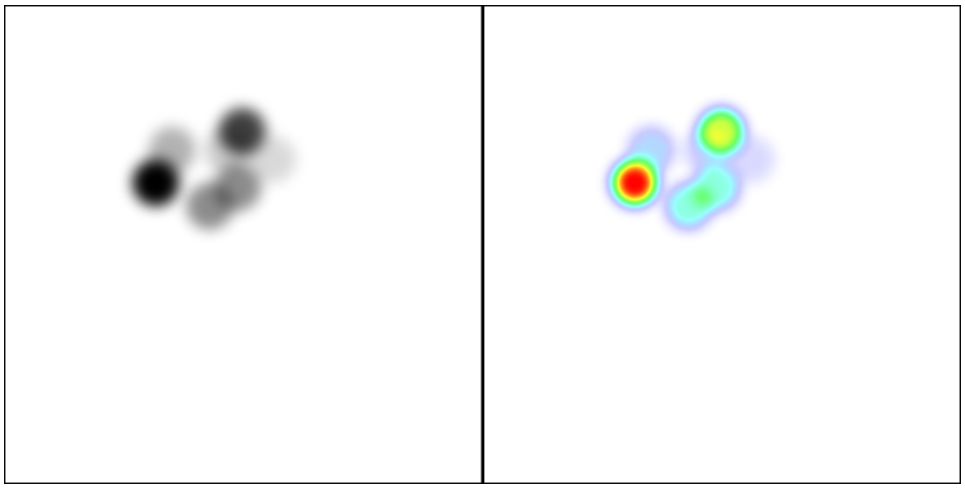

Figura 3.11: A la izquierda se muestra el mapa en escala de grises y a su derecha el resultado tras colorear el mapa.

3. En función de la opacidad de cada punto del lienzo, se colorea el mapa de grises, dándole un color frío a los valores cercanos al 0 y asignando un color más cálido a medida que se acercan al valor máximo 1.

Las funciones definidas para la creación de estas texturas se encuentran en la librería *createCanvas.js* (Anexo C). En esta librería nos encontramos con las distintas funciones:

- **createEmptyCanvas()**: es importante señalar que la actividad nula no será representada mediante el método *draw()* de la clase *simpleheat*, por lo tanto, el resultado de ésta será una imagen con transparencias en las zonas sin actividad. Para poder representar la actividad vacía, creamos una textura de color azul oscuro, que utilizaremos como primera capa de la textura, donde se pondrá encima el mapa de calor cuando se detecte actividad.

- **createDotsCanvas()**: se ha introducido una función auxiliar con la que visualizar la ubicación de los electrodos con un punto negro. Esta función genera un lienzo con

círculos en las posiciones definidas para cada sensor y se ubicará como una capa más, encima de la textura con la actividad cerebral.

- **createAllCanvas()**: es una función de inicialización a la que se llama cuando creamos la escena en threejs. Esta función llama a las dos funciones descritas anteriormente y crea cuatro elementos canvas más; dos donde se mostrarán los mapas de calor para la parte derecha e izquierda, y otros dos donde se mostrará el resultante de la suma de las distintas capas.

- **resetValues()**: función auxiliar con la que actualizar los valores de la actividad a cero.

- **updateCanvas(leftValues, rigthValues)**: mediante la llamada a esta función se actualizarán los valores de la textura. Se introducen como entrada los valores de los electrodos y se genera nuevamente el mapa de calor. A continuación, se combinan las capas de actividad vacía, mapa de calor y en caso de que este seleccionado desde la aplicación web, la posición de los electrodos.

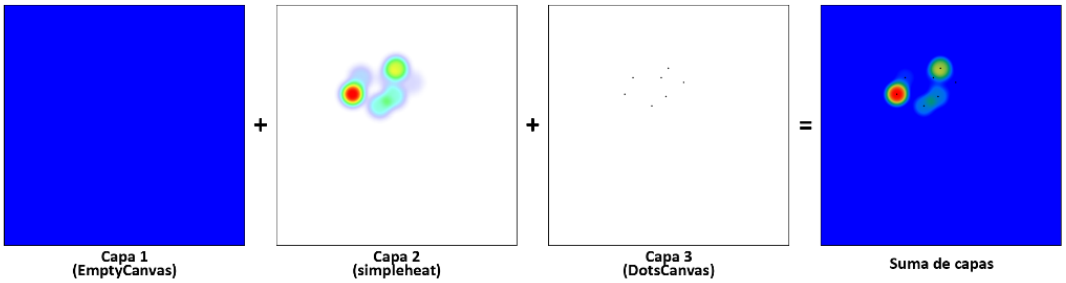

Figura 3.12: Se muestran las distintas capas que forman la textura de la acividad neuronal.

3.4. Lectura de datos

Como se ha dicho anteriormente, esta aplicación contempla dos formas de uso: en tiempo real conectándose directamente al dispositivo EPOC+, y mediante ficheros CSV donde se han almacenado previamente los valores de la sesión grabada.

3.4.1. Lectura en tiempo real

La recogida de datos ha sido posible con la creación de un servidor web creado en Python y utilizando Emotiv SDK.

3.4.1.1. Implementación del servidor HTTP

El módulo utilizado para la creación del servidor ha sido *BaseHTTPServer*, el cual está definido por dos clases que deberemos implementar:

- **HTTPServer(server_address, RequestHandlerClass)**: en esta clase se almacenan la dirección del servidor como variables de instancia llamadas *server_name* y *server_port*. El servidor es accesible para el gestor o *handler*, generalmente a través de la variable de instancia del gestor server.

- **BaseHTTPRequestHandler(request, client_address, server)**: mediante esta clase se gestionarán las peticiones HTTP que llegan al servidor. Es aquí donde se debe definir cómo se gestiona cada petición GET o POST.

En este desarrollo, se ha utilizado el puerto 8080 y el server_name vacío, por lo que se podrá acceder al servidor desde la web `http://localhost:8080/`.

Para la implementación del gestor de peticiones se ha definido una petición GET en la que se cargarán las distintas librerías y objetos necesarios para la visualización; y una petición POST, en la que se devolverá la información de los sensores de la banda de frecuencias solicitada.

3.4.1.2. Lectura de la actividad neuronal

Se ha creado una petición POST mediante la que, facilitando la banda que se quiere mostrar, devuelve los datos de la actividad neuronal de los distintos sensores. Para leer estos valores, la petición llama a la función definida dentro del archivo *server.py info(band)* (Anexo B), que ha sido implementada con las librerías Emotiv SDK.

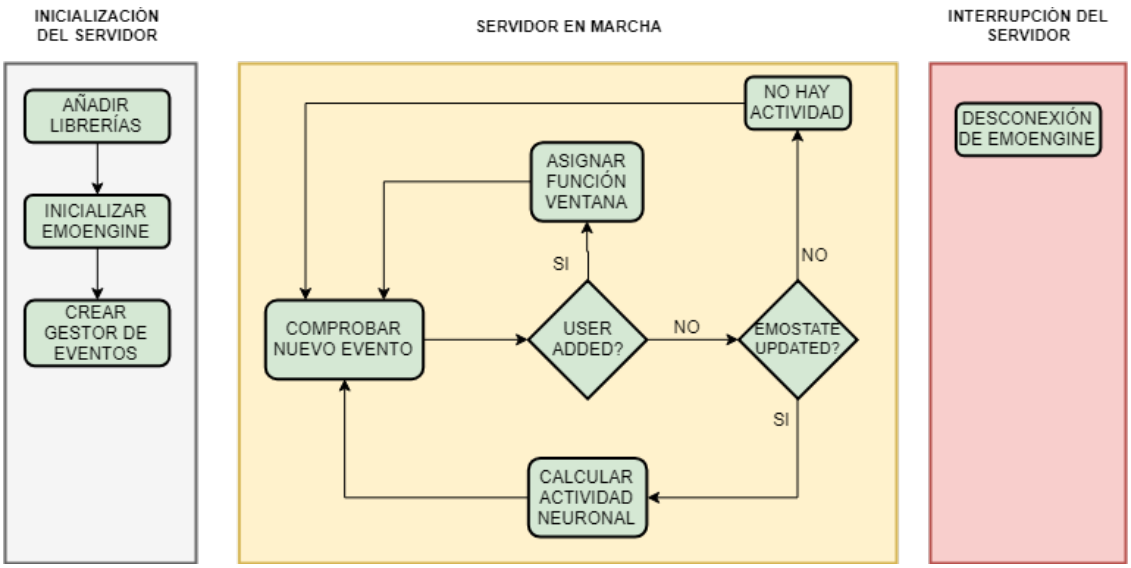

Figura 3.13: Diagrama de lectura de actividad neuronal en tiempo real.

Para la utilización de este SDK tendremos que hacer la siguiente inicialización:

- **Añadir librerías**: el código consulta el sistema operativo y arquitectura del ordenador, cargando la librería necesaria en el equipo en el que se ejecuta el programa.
- **Inicializar la instancia EmoEngine**: esta permitirá la lectura de datos de los sensores. Para ello tendremos que llamar a la función *IEE_EngineConnect(Emotiv Systems-5)*.
- **Crear gestor de eventos Emotiv**: cada vez que se intenta obtener información del EPOC+, es necesario consultar el tipo de evento que nos devuelve el *EmoEngine*. Para poder consultar los eventos se utiliza la función *IEE_EmoEngineEventCreate()*, con la que se creará un gestor de eventos en memoria donde se almacenará cada uno de ellos. Éstos, sirven para saber si se ha añadido un dispositivo Emotiv, si se han actualizado los datos, etc.

Una vez inicializado, cada vez que se haga una petición POST para la reproducción de la actividad cerebral en vivo, se comprobará que existe un próximo evento. Pese a que los eventos puedan ser de diversos tipos, se manejarán dos tipos de eventos:

- **IEE_UserAdded**: significa que el programa ha detectado un nuevo dispositivo

Emotiv. En este punto, se le asigna el tipo de función ventana[1] que se le aplicará para el cálculo de la potencia de la señal en cada banda.

- **IEE_EmoStateUpdated**: si se devuelve este evento, quiere decir que se ha conectado un dispositivo con anterioridad y que sigue teniendo conexión.

Cualquiera de estos dos eventos daría pie a un cálculo de la actividad neuronal, que se calcularía mediante la función *IEE_GetAverageBandPowers(userId, channel, theta, Alpha, low_beta, high_beta, gamma)*. En caso contrario, no se devolverá nada, ya que significaría que no hay conexión con el casco neuronal.

Esta función devuelve la potencia media en las diferentes bandas de actividad cerebral para un sensor especifico. El cálculo de la potencia media se realiza en los 2 segundos previos a la llamada y se actualiza cada 0,125 segundos (8 Hz).

Onda	**Banda**
Theta	4 - 8 Hz
Alpha	8 - 16 Hz
Low Beta	12 - 16 Hz
High Beta	12 - 16 Hz
Gamma	25 - 45Hz

Tabla 3.1: Bandas de frecuencias calculadas por la función GetAverageBandPowers().

En la petición POST, con la función *info(band)*, se recopilarán los valores de cada electrodo en la banda de frecuencias facilitada por el usuario, devolviendo todos estos datos para ser procesados y mapeados en el cerebro 3D.

La desconexión del *EmoEngine* solo se dará en caso de que se pare el servidor web.

3.4.2. Lectura en diferido

La reproducción en diferido consiste en mostrar en el navegador la actividad neuronal que ha sido almacenada en un fichero anteriormente. Para ello, el primer paso que se ha realizado para poder hacer la captura, ha sido la decisión del formato y estructura del archivo de sesiones.

[1] Función ventana: Las ventanas son funciones matemáticas usadas en el análisis y el procesamiento de señales para evitar las discontinuidades al principio y al final de los bloques analizados.

Una vez definida la estructura que tendrá el archivo de sesiones, el proceso será el siguiente:

Figura 3.14: Diagrama del funcionamiento de la lectura en diferido.

3.4.2.1. Estructura del archivo de sesiones

Los datos recogidos se han guardado en formato CSV (*Comma-Separated Values*). Estos archivos son un tipo de documento en formato abierto sencillo, para la representación de datos en forma de tabla. Las distintas columnas se separan por comas y las filas se separan por salto de línea. En este archivo se reflejan todos los datos necesarios para la futura reproducción: el segundo en el que ha sido recogido ese dato (nos referiremos a él como *timestamp*) y los valores obtenidos de cada uno de los sensores.

Emotiv SDK nos proporciona los valores de cada banda de frecuencias por separado, es decir, nos da los valores de cada sensor para las siguientes bandas, por lo que por cada timestamp se escribirán 5 filas en el archivo CSV, donde se recogerá separado por comas el *timestamp* seguido de la actividad en los puntos AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8 y AF4. Por ejemplo, para el timestamp (TS) 3 segundos, obtendremos una tabla (interpretando el formato CSV) con esta apariencia:

TS	**AF3**	**F7**	**F3**	**FC5**	**T7**	**P7**	**O1**	**O2**	**P8**	**T8**	**FC6**	**F4**	**F8**	**AF4**
3	25	25	25	25	25	25	25	25	25	25	25	25	25	25
3	20	20	20	20	20	20	20	20	20	20	20	20	20	20
3	15	15	15	15	15	15	15	15	15	15	15	15	15	15
3	10	10	10	10	10	10	10	10	10	10	10	10	10	10
3	5	5	5	5	5	5	5	5	5	5	5	5	5	5

Tabla 3.2: Representación del formato CSV.

3.4.2.2. Captura de sesión

Al igual que el servidor web, la captura de sesiones se realiza en Python mediante Emotiv SDK. Para grabar una captura tendremos que ejecutar el programa *AverageBandPowers-Logger.py* (Anexo A) fuera de la aplicación web.

Su funcionamiento es idéntico a la función definida para la lectura en vivo, con la dife-

rencia de que una vez inicializado, se crea un fichero CSV donde se irán almacenando los datos obtenidos por el EPOC+.

Tras crear este fichero, el programa consultará continuamente los eventos, y en caso de detectar actividad la irá guardando en el fichero CSV respetando la estructura definida, es decir, guardando los datos de las 5 posibles bandas de frecuencia.

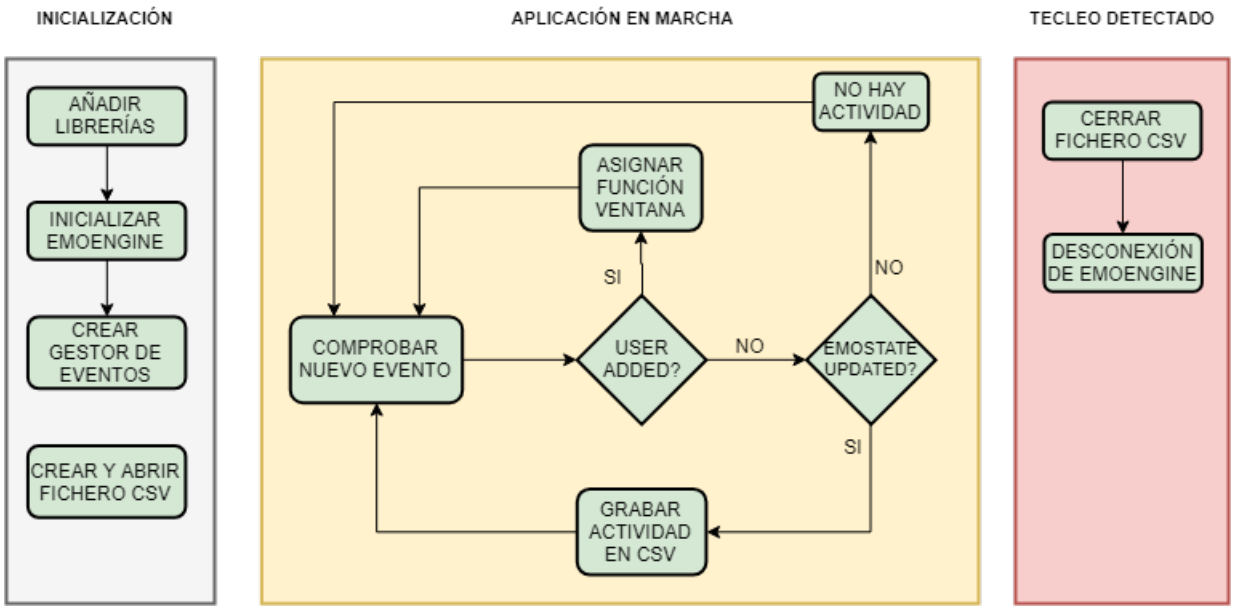

Figura 3.15: Diagrama de lectura y grabación en CSV de actividad neuronal.

Para salir del bucle, bastará con pulsar una tecla con la que se cerrará el fichero CSV y se desconectará del *EmoEngine*. El fichero de sesión quedará guardado en el mismo directorio donde se encuentra el programa en Python.

Debido al funcionamiento de la función que calcula la potencia media, se puede observar que las sesiones no tienen actividad guardada los 2 primeros segundos. Esto es debido a que, como se comentaba anteriormente, esta función calcula la media en los dos segundos previos a la llamada.

3.5. Procesamiento de datos

Cuando hablamos del procesamiento de datos nos referimos a cómo se cargan y traducen los datos obtenidos por los sensores y se interpretan en formato visual mediante el mapeado de texturas.

Este proceso, aunque sea muy parecido tanto en el caso de que estemos reproduciendo una actividad grabada, como la actividad en directo, tiene algunos matices que debemos comentar.

3.5.1. Procesamiento de datos en tiempo real

La respuesta recibida por la petición POST contiene una línea con los valores de los distintos canales para la banda solicitada separados por comas (en caso de detectar actividad). Esta línea se trata convirtiendo los valores en un *array* (utilizando la coma como separador) y se divide en los *arrays leftValues[]* y *rigthValues[]*, uno por cada hemisferio del cerebro. Con estos datos se hará una llamada a la función *UpdateCanvas(leftValues, rigthValues)*, con la que actualizar la textura con los datos obtenidos en ese instante de tiempo.

En caso de no recibir respuesta con actividad en dos segundos y medio, se mostrará en pantalla el mensaje *"No activity detected"*, en caso contrario, aparecerá el mensaje *"Reading Data"*.

3.5.2. Procesamiento de datos grabados

El procesamiento de datos obtenidos mediante el CSV es un poco más complejo. En primer lugar, se ha creado un menú donde cargar el fichero CSV, el cual una vez cargado, almacenará los datos de toda la sesión en memoria. Para realizar esta carga de datos se ha usado la librería javascript PapaParse.js con la que los datos obtenidos, se guardarán ordenados y agrupados por el timestamp en el que se ha recogido cada información.

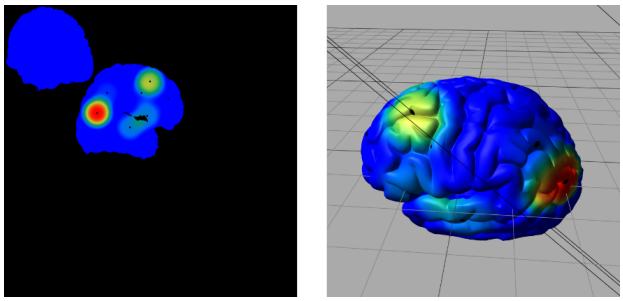

Figura 3.16: Resultado tras aplicar la textura generada en javascript en el modelo 3D.

En la aplicación web, una vez cargado el fichero, se inicializa una variable de tiempo que

se utilizará a modo de reloj. A lo largo de la simulación, esta variable se irá comparando con el valor del *timestamp*, y en el momento en el que el segundo de la simulación sea el mismo que el del timestamp, se sacarán esos datos de memoria y se descartarán los datos de ese segundo que no necesitamos, quedándonos únicamente con la actividad de la banda de frecuencias que tenemos seleccionada.

Llegados a este punto, ya se podrán tratar los datos al igual que en la simulación en tiempo real, dividiendo los datos en los *arrays leftValues[]* y *rigthValues[]*, con los que llamando a la función *UpdateCanvas(leftValues, rigthValues)*, actualizaremos la textura del cerebro.

Este proceso se dará mientras queden datos en memoria, es decir, hasta leer todos los datos de la sesión. Cuando finalice, el cerebro se mostrará con actividad nula y un mensaje *"Session Ended"*.

En caso de que el usuario quiera cambiar de reproducción de sesión grabada a modo en directo, se liberará la memoria utilizada para almacenar la sesión y pasará a capturar los datos en tiempo real.

3.6. Aplicación web

Para ejecutar la aplicación y poder visualizar la actividad neuronal, lo primero que se debe hacer es poner en marcha el servidor web. Para ello, basta con tener Python instalado en el equipo y ejecutar el archivo server.py.

Arrancado el servidor, accederemos en el navegador a la dirección `http://localhost:8080/`, donde tras visualizar un *loader* mientras se cargan los datos de la web, se verá lo siguiente:

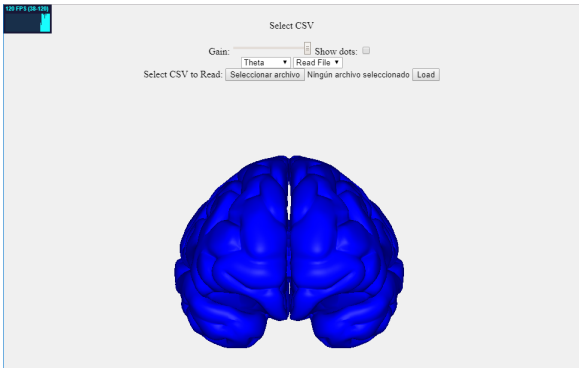

Figura 3.17: Apariencia de la aplicación tras el proceso de inicialización.

Una vez aquí, bastará con cargar el fichero CSV o cambiar al modo de reproducción en vivo para poder visualizar la actividad neuronal.

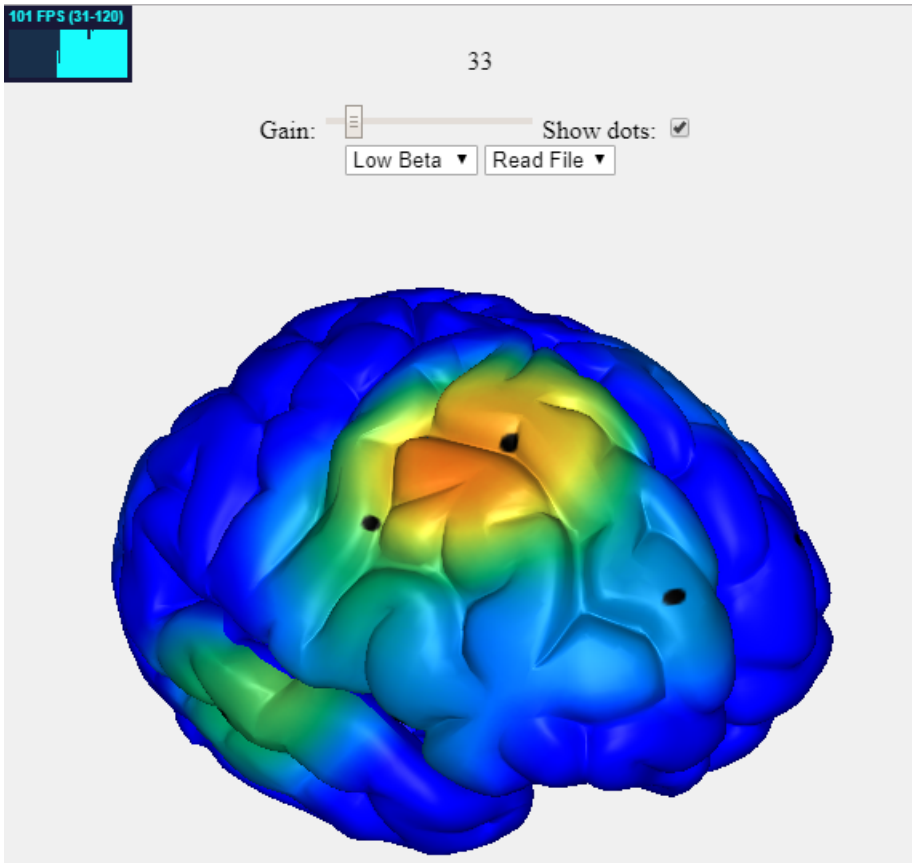

Figura 3.18: Aplicación reproduciendo sesión grabada.

3.6.1. Menú de la aplicación

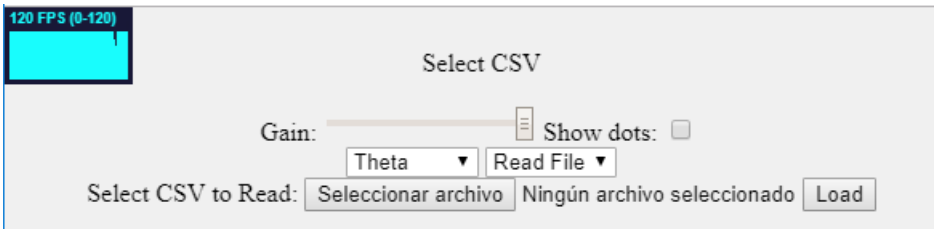

Figura 3.19: Menu de la aplicación.

- **Texto informativo**: se utiliza para dar indicaciones al usuario y mostrar el estado en el que se encuentra la aplicación.

- **Ganancia**: como se ha podido observar en el estudio de los distintos tipos de ritmos cerebrales, las distintas bandas de frecuencias pueden tener una amplitud de onda distinta. Por esta razón, ha sido necesaria la implementación de un *slider* con el que regular la amplitud máxima en cada momento, dependiendo de la onda en la que nos encontremos. Este valor será el utilizado como max cuando se define la clase simpleheat.

- **Mostrar puntos**: esta opción nos permite visualizar la ubicación de los distintos electrodos mediante un punto. Se puede activar o desactivar en cualquier momento.

- **Selector de banda**: es un desplegable en el que se seleccionará la banda de frecuencias que se desea visualizar.

- **Opción de visualización**: este desplegable permitirá elegir en que modo se desea trabajar (lectura de CSV o en tiempo real).

- **Menú de carga**: permitirá cargar los ficheros CSV. Si nos encontramos en la opción de visualización de tiempo real o si el fichero ya ha sido cargado, este menú desaparecerá.

- **Stats**: es una herramienta propia de la librería Three.js que nos permite poder visualizar el rendimiento de la aplicación.

4. CAPÍTULO

Conclusiones

4.1. Complicaciones

Por lo general el proyecto se ha llevado a cabo sin incidencias, sin embargo, la complicación principal ha sido el desconocimiento de métodos y medios para la realización de ciertas funcionalidades de la aplicación. Pese a que estos problemas no han sido críticos y se ha conseguido cumplir con todos los objetivos del proyecto, se mencionan las siguientes complicaciones:

- En un primer momento se planteó realizar la visualización de la actividad neuronal mediante el mapeado de normales, es decir, aplicándole un color a cada cara del objeto tridimensional en lugar de realizarlo con texturas (como se hizo finalmente). Haciéndolo de este modo no se conseguía una animación fluida ni precisa, ya que era necesario calcular el color para cada cara (teniendo en cuenta la actividad de los 14 electrodos), e impedía conseguir un rendimiento adecuado.

- Aunque se haya asignado a la formación un número importante de horas previas al desarrollo de la aplicación, ha sido necesario un aprendizaje continuo a medida que se encontraban conceptos nuevos en la implementación. Esto ha hecho que se tenga que modificar la planificación para poder así cumplir con las fechas.

- Una vez implementado el código, en las primeras pruebas se pudo observar que el ordenador en el que se estaba ejecutando la aplicación consumía una gran cantidad

de recursos. Esto hacía que no se visualizaran los datos de forma fluida, por lo que fue necesaria una revisión del código. Tras analizarlo, el problema provenía de la animación del objeto, ya que se recalculaban las texturas incluso cuando no había variación en la actividad neuronal.

4.2. Resultados del proyecto

Una vez finalizado el proyecto, considero que tanto los objetivos de aprendizaje como los de diseño han sido alcanzados, obteniendo los conocimientos necesarios para la realización del proyecto y logrando implementar una aplicación que cumple con las características señaladas en la descripción del proyecto.

La aplicación que he creado hace posible utilizar el casco neuronal Emotiv EPOC+ de una forma sencilla y *plug & play*, sin la dificultad que supone reproducir la actividad neuronal en herramientas como EEGLab, donde es necesaria una compleja configuración para poder visualizar las capturas de la actividad neuronal. Por otro lado, al ser una aplicación web, también se ha podido comprobar que la visualización funciona correctamente en dispositivos móviles, lo cual supone una gran ventaja respecto al resto de herramientas que requieren una instalación de software externo en el equipo.

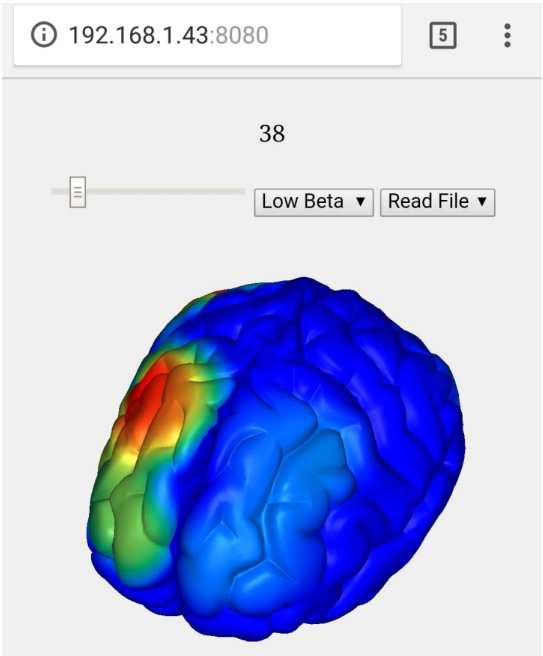

Figura 4.1: Aplicación funcionando en smartphone.

En lo que a rendimiento se refiere, los resultados obtenidos tras la puesta en marcha de la aplicación han sido satisfactorios. Se ha podido observar mediante la herramienta stats de Three.js, que una vez hecha la inicialización el rendimiento de la aplicación es óptimo, manteniéndose constante en los 120 FPS en el ordenador en el que se han realizado las pruebas. Cuando se carga el fichero CSV o visualizamos la actividad en tiempo real, el rendimiento disminuye mínimamente manteniéndose en una media de 110 FPS.

En pruebas iniciales, el rendimiento no subía de los 50 FPS, lo que hizo necesaria una revisión del código. Esto me ha hecho ser consciente de la gran importancia que tiene una buena optimización en los programas, viendo como se ha conseguido multiplicar por dos el rendimiento con pequeñas modificaciones.

Pese a las distintas dificultades que han surgido a lo largo del desarrollo del proyecto, estoy satisfecho con los resultados obtenidos, ya que pueden servir como base a futuros proyectos. En resumen, considero que ha sido un proyecto muy gratificante.

4.3. Líneas futuras

Mediante este proyecto se ha logrado crear una base desde la que poder seguir desarrollando una aplicación de monitorización neuronal con más funcionalidades.

A continuación, se muestra un listado con las implementaciones que considero de interés:

- **Visualización de distintos tipos de ondas al mismo tiempo**: en la aplicación desarrollada se muestra un único cerebro donde poder seleccionar el tipo de onda que se quiere visualizar. Sería de gran utilidad añadir a la aplicación la posibilidad de visualizar al mismo tiempo distintas ondas cerebrales en diferentes cerebros.

- **Optimización del código**: en caso de añadir la funcionalidad anterior, sería recomendable aumentar el rendimiento de la aplicación mediante procesos en paralelo o concurrentes, ya que el aumento de datos a procesar por la aplicación hará que el sistema consuma más recursos.

- **Posibilidad de grabar sesiones desde la propia aplicación web**: para la recogida de datos de una sesión se ha creado una aplicación en Python que hay que ejecutar fuera de la aplicación web. Sería de gran utilidad poder realizar estas capturas desde la misma aplicación web.

- **Implementar controles de reproducción en la visualización de sesiones grabadas**: tras realizar diversas pruebas en la aplicación web, se ha concluido que sería necesario añadir controles de reproducción para poder avanzar en la sesión, visualizar los datos o ir a un segundo concreto, por ejemplo.

Anexos

A. ANEXO

AverageBandPowersLogger.py: código Python para capturar sesiones en CSV

```
import sys
import os
import platform
import time
import ctypes

from array import *
from ctypes import *
from __builtin__ import exit

if sys.platform.startswith('win32'):
    import msvcrt
elif sys.platform.startswith('linux'):
    import atexit
    from select import select

try:
    if sys.platform.startswith('win32'):
        libEDK = cdll.LoadLibrary("bin/win32/edk.dll")
    elif sys.platform.startswith('linux'):
        srcDir = os.getcwd()
        if platform.machine().startswith('arm'):
            libPath = srcDir + "/bin/armhf/libedk.so"
        else:
            libPath = srcDir + "/bin/linux64/libedk.so"
        libEDK = CDLL(libPath)
    else:
        raise Exception('System not supported.')
except Exception as e:
```

```
30      print ("Error: cannot load EDK lib:", e)
31      exit()
32
33  IEE_EmoEngineEventCreate = libEDK.IEE_EmoEngineEventCreate
34  IEE_EmoEngineEventCreate.restype = c_void_p
35  eEvent = IEE_EmoEngineEventCreate()
36
37  IS_GetTimeFromStart = libEDK.IS_GetTimeFromStart
38  IS_GetTimeFromStart.argtypes = [ctypes.c_void_p]
39  IS_GetTimeFromStart.restype = c_float
40
41  IEE_EmoEngineEventGetEmoState = libEDK.IEE_EmoEngineEventGetEmoState
42  IEE_EmoEngineEventGetEmoState.argtypes = [c_void_p, c_void_p]
43  IEE_EmoEngineEventGetEmoState.restype = c_int
44
45  IEE_EmoStateCreate = libEDK.IEE_EmoStateCreate
46  IEE_EmoStateCreate.restype = c_void_p
47  eState = IEE_EmoStateCreate()
48
49  userID = c_uint(0)
50  user   = pointer(userID)
51  ready  = 0
52  state  = c_int(0)
53  systemUpTime     = c_float(0.0)
54
55  alphaValue     = c_double(0)
56  low_betaValue  = c_double(0)
57  high_betaValue = c_double(0)
58  gammaValue     = c_double(0)
59  thetaValue     = c_double(0)
60
61  alpha     = pointer(alphaValue)
62  low_beta  = pointer(low_betaValue)
63  high_beta = pointer(high_betaValue)
64  gamma     = pointer(gammaValue)
65  theta     = pointer(thetaValue)
66
67  newLine = 0
68
69  def kbhit():
70      ''' Returns True if keyboard character was hit, False otherwise.
71      '''
72      if sys.platform.startswith('win32'):
73          return msvcrt.kbhit()
74      else:
75          dr,dw,de = select([sys.stdin], [], [], 0)
76          return dr != []
77
78  channelList = array('I',[3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16])
79  header = "Time, AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4"
80
81  # ----------------------------------------------------------------------
```

```
print "================================================================="
print "Get the average band power from the epoch."
print "================================================================="

# -------------------------------------------------------------------------
if libEDK.IEE_EngineConnect("Emotiv Systems-5") != 0:
        print "Emotiv Engine start up failed."
        exit();

print "Press any key to stop logging...\n"

f = file('AverageBandPowersLogger.csv', 'w')
f = open('AverageBandPowersLogger.csv', 'w')

print >>f, header, "\n",

while True:
    if kbhit():
        break
    state = libEDK.IEE_EngineGetNextEvent(eEvent)
    if state == 0:
        eventType = libEDK.IEE_EmoEngineEventGetType(eEvent)
        libEDK.IEE_EmoEngineEventGetUserId(eEvent, user)
        if eventType == 16:  # libEDK.IEE_Event_enum.IEE_UserAdded
            ready = 1
            libEDK.IEE_FFTSetWindowingType(userID, 1);  # 1: libEDK.IEE_WindowingTypes_enum.
    IEE_HAMMING
            print "User added"
        if eventType == 64:  # libEDK.IEE_Event_enum.IEE_EmoStateUpdated

            libEDK.IEE_EmoEngineEventGetEmoState(eEvent, eState);
            systemUpTime = IS_GetTimeFromStart(eState);
        if ready == 1:
            for i in channelList:
                result = c_int(0)
                result = libEDK.IEE_GetAverageBandPowers(userID, i, theta, alpha, low_beta,
    high_beta, gamma)

                if result == 0:    #EDK_OK
                    if newLine == 0:
                        ThetaTxt = str(systemUpTime)
                        AlphaTxt = str(systemUpTime)
                        LBetaTxt = str(systemUpTime)
                        HBetaTxt = str(systemUpTime)
                        GammaTxt = str(systemUpTime)
                        newLine = 1
                    ThetaTxt += ", %.6f" % (thetaValue.value)
                    AlphaTxt += ", %.6f" % (alphaValue.value)
                    LBetaTxt += ", %.6f" % (low_betaValue.value)
                    HBetaTxt += ", %.6f" % (high_betaValue.value)
                    GammaTxt += ", %.6f" % (gammaValue.value)
            if newLine == 1:
```

```
                print >>f, ThetaTxt
                print >>f, AlphaTxt
                print >>f, LBetaTxt
                print >>f, HBetaTxt
                print >>f, GammaTxt
                newLine = 0
    elif state != 0x0600:
        print "Internal error in Emotiv Engine ! "
    time.sleep(0.1)
# ---------------------------------------------------------------------------
libEDK.IEE_EngineDisconnect()
libEDK.IEE_EmoStateFree(eState)
libEDK.IEE_EmoEngineEventFree(eEvent)
```

B. ANEXO

Server.py: código del servidor HTTP

```
#!/usr/bin/python
from BaseHTTPServer import BaseHTTPRequestHandler,HTTPServer
from os import curdir, sep
PORT_NUMBER = 8080

import sys
import os
import platform
import time
import ctypes

from array import *
from ctypes import *
from __builtin__ import exit

if sys.platform.startswith('win32'):
    import msvcrt
elif sys.platform.startswith('linux'):
    import atexit
    from select import select

from ctypes import *

try:
    if sys.platform.startswith('win32'):
        libEDK = cdll.LoadLibrary("bin/win32/edk.dll")
    elif sys.platform.startswith('linux'):
        srcDir = os.getcwd()
        if platform.machine().startswith('arm'):
            libPath = srcDir + "/bin/armhf/libedk.so"
        else:
```

```
            libPath = srcDir + "/bin/linux64/libedk.so"
        libEDK = CDLL(libPath)
    else:
        raise Exception('System not supported.')
except Exception as e:
    print ("Error: cannot load EDK lib:", e)
    exit()

IEE_EmoEngineEventCreate = libEDK.IEE_EmoEngineEventCreate
IEE_EmoEngineEventCreate.restype = c_void_p
eEvent = IEE_EmoEngineEventCreate()

userID = c_uint(0)
user   = pointer(userID)
ready  = 0
state  = c_int(0)

alphaValue     = c_double(0)
low_betaValue  = c_double(0)
high_betaValue = c_double(0)
gammaValue     = c_double(0)
thetaValue     = c_double(0)

alpha     = pointer(alphaValue)
low_beta  = pointer(low_betaValue)
high_beta = pointer(high_betaValue)
gamma     = pointer(gammaValue)
theta     = pointer(thetaValue)

channelList = array('I',[3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16])

if libEDK.IEE_EngineConnect("Emotiv Systems-5") != 0:
        print "Emotiv Engine start up failed."
        exit();

def info(entrada):
    nuevalinea = 0
    state = libEDK.IEE_EngineGetNextEvent(eEvent)
    if state == 0:
        eventType = libEDK.IEE_EmoEngineEventGetType(eEvent)
        libEDK.IEE_EmoEngineEventGetUserId(eEvent, user)
        if eventType == 16:  # libEDK.IEE_Event_enum.IEE_UserAdded
            ready = 1
            libEDK.IEE_FFTSetWindowingType(userID, 1);  # 1: libEDK.IEE_WindowingTypes_enum.
    IEE_HAMMING
            print "User added"
        if eventType == 64:
            ready = 1
        if ready == 1:
            for i in channelList:
                result = c_int(0)
```

```
82                 result = libEDK.IEE_GetAverageBandPowers(userID, i, theta, alpha, low_beta,
       high_beta, gamma)
83
84                 if result == 0:    #EDK_OK
85                     if nuevalinea == 0:
86                         ThetaTxt = "now"
87                         AlphaTxt = "now"
88                         LBetaTxt = "now"
89                         HBetaTxt = "now"
90                         GammaTxt = "now"
91                         nuevalinea = 1
92                     ThetaTxt += ", %.6f" % (thetaValue.value)
93                     AlphaTxt += ", %.6f" % (alphaValue.value)
94                     LBetaTxt += ", %.6f" % (low_betaValue.value)
95                     HBetaTxt += ", %.6f" % (high_betaValue.value)
96                     GammaTxt += ", %.6f" % (gammaValue.value)
97             if nuevalinea == 1:
98                 if (entrada == "0"):
99                     return ThetaTxt
100                if (entrada == "1"):
101                    return AlphaTxt
102                if (entrada == "2"):
103                    return LBetaTxt
104                if (entrada == "3"):
105                    return HBetaTxt
106                if (entrada == "4"):
107                    return GammaTxt
108                nuevalinea = 0
109
110    elif state != 0x0600:
111        print "Internal error in Emotiv Engine ! "
112    time.sleep(0.1)
113
114class myHandler(BaseHTTPRequestHandler):
115
116    #Handler for the GET requests
117    def do_GET(self):
118        if self.path=="/":
119            self.path="/index.html"
120
121        try:
122            #Check the file extension required and
123            #set the right mime type
124
125            sendReply = False
126            if self.path.endswith(".html"):
127                mimetype='text/html'
128                sendReply = True
129            if self.path.endswith(".jpg"):
130                mimetype='image/jpg'
131                sendReply = True
132            if self.path.endswith(".js"):
```

```
133                 mimetype='application/javascript'
134                 sendReply = True
135             if self.path.endswith(".css"):
136                 mimetype='text/css'
137                 sendReply = True
138             if self.path.endswith(".obj"):
139                 mimetype='text/plain'
140                 sendReply = True
141 
142             if sendReply == True:
143                 #Open the static file requested and send it
144                 f = open(curdir + sep + self.path)
145                 self.send_response(200)
146                 self.send_header('Content-type',mimetype)
147                 self.end_headers()
148                 self.wfile.write(f.read())
149                 f.close()
150             return
151 
152         except IOError:
153             self.send_error(404,'File Not Found: %s' % self.path)
154 
155     #Handler for the POST requests
156     def do_POST(self):
157         if self.path=="/send":
158             length = int(self.headers.getheader('content-length'))
159             data = self.rfile.read(length)
160             print data
161             # Seguir por aqui
162             self.send_response(200)
163             self.end_headers()
164             datos = info(data)
165             #datos = info()
166             self.wfile.write(datos)
167             return
168 
169 
170 try:
171     #Create a web server and define the handler to manage the
172     #incoming request
173     server = HTTPServer(('', PORT_NUMBER), myHandler)
174     print 'Started httpserver on port ' , PORT_NUMBER
175 
176     #Wait forever for incoming http requests
177     server.serve_forever()
178 
179 except KeyboardInterrupt:
180     print '^C received, shutting down the web server'
181     server.socket.close()
182 
```

C. ANEXO

CreateCanvas.js: librería Javascript desarrollada para la creación de textura mediante canvas

```
var ctxLeft, ctxRigth;
var leftCanDraw, rigthCanDraw;
var emptyCanvas;
var canLeft, canRigth;
var heatLeft, heatRigth;
var leftValues = [0,0,0,0,0,0,0];
var rigthValues = [0,0,0,0,0,0,0];
var DotsCanvas;

function createEmptyCanvas() {
    emptyCanvas = document.createElement( "canvas" );
    emptyCanvas.width = emptyCanvas.height = 1024;
    emptyCanvas.setAttribute("id", "EmptyCanvas");
    emptyCanvas.setAttribute("hidden", "");
    document.getElementById("container").appendChild(emptyCanvas);

    var ctxEmpty = emptyCanvas.getContext('2d');
    ctxEmpty.fillStyle="blue";
    ctxEmpty.fillRect(0,0,1024,1024);
}

function createDotsCanvas() {
    DotsCanvas = document.createElement( "canvas" );
    DotsCanvas.width = DotsCanvas.height = 1024;
    DotsCanvas.setAttribute("id", "DotsCanvas");
    DotsCanvas.setAttribute("hidden", "");
    document.getElementById("container").appendChild(DotsCanvas);

    var ctxDots = DotsCanvas.getContext('2d');
```

```
    for (i = 0; i < data.length; i++) {
        var coordPos = data[i];
        ctxDots.beginPath();
        ctxDots.arc(coordPos[0],coordPos[1],3,0,2*Math.PI);
        ctxDots.fill();
    }
}

function createAllCanvas() {
    createEmptyCanvas();
    createDotsCanvas();

    leftCanDraw = document.createElement( "canvas" );
    leftCanDraw.width = leftCanDraw.height = 1024;
    leftCanDraw.setAttribute("id", "CanvasLeftDraw");
    leftCanDraw.setAttribute("hidden", "");
    document.getElementById("container").appendChild(leftCanDraw);

    rigthCanDraw = document.createElement( "canvas" );
    rigthCanDraw.width = rigthCanDraw.height = 1024;
    rigthCanDraw.setAttribute("id", "CanvasRigthDraw");
    rigthCanDraw.setAttribute("hidden", "");
    document.getElementById("container").appendChild(rigthCanDraw);

    canLeft = document.createElement( "canvas" );
    canLeft.width = canLeft.height = 1024;
    canLeft.setAttribute("id", "CanvasLeft");
    document.getElementById("container").appendChild(canLeft);

    ctxLeft = canLeft.getContext('2d');
    ctxLeft.drawImage(emptyCanvas, 0, 0);

    canRigth = document.createElement( "canvas" );
    canRigth.width = canRigth.height = 1024;
    canRigth.setAttribute("id", "CanvasRigth");
    document.getElementById("container").appendChild(canRigth);

    ctxRigth = canRigth.getContext('2d');
    ctxRigth.drawImage(emptyCanvas, 0, 0);
}
function resetValues(){
    leftValues = [0,0,0,0,0,0,0];
    rigthValues = [0,0,0,0,0,0,0];
}
function updateCanvas(leftValues, rigthValues){
    heatLeft = simpleheat('CanvasLeftDraw').data(data).max(Gain.value).valores(leftValues);
    heatLeft.draw();
    heatRigth = simpleheat('CanvasRigthDraw').data(data).max(Gain.value).valores(rigthValues);
    heatRigth.draw();

    ctxLeft.drawImage(emptyCanvas, 0, 0);
    ctxLeft.drawImage(leftCanDraw, 0, 0);
```

```
82
83      ctxRigth.drawImage(emptyCanvas, 0, 0);
84      ctxRigth.drawImage(rigthCanDraw, 0, 0);
85
86      if(document.getElementById("showDots").checked){
87          ctxLeft.drawImage(DotsCanvas, 0, 0);
88          ctxRigth.drawImage(DotsCanvas, 0, 0);
89      }
90  }
91  function draw(e) {
92      var canvas = document.getElementById("imgCanvas");
93      var context = canvas.getContext("2d");
94      var rect = canvas.getBoundingClientRect();
95      var posx = e.clientX - rect.left;
96      var posy = e.clientY - rect.top;
97
98      context.fillStyle = "#000000";
99      context.beginPath();
100     context.arc(posx, posy, 50, 0, 2 * Math.PI);
101     context.fill();
102 }
```

D. ANEXO

Index.html: código de la aplicación web

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Activity</title>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, user-scalable=no, minimum-scale=1.0, maximum-scale=1.0">
        <link rel="stylesheet" href="CSS/style.css">
    </head>

    <body>
        <div id="loader"></div>
        <div id="buttons">

            <p id="infoTxt">Select CSV</p>
            Gain: <input type="range" min="0.5" max="20" value="20" step="0.5" class="slider" id="Gain">
            Show dots: <input type="checkbox" id="showDots" />
            </br>
            <select id="Selector">
                <option value="0">Theta</option>
                <option value="1">Alpha</option>
                <option value="2">Low Beta</option>
                <option value="3">High Beta</option>
                <option value="4">Gamma</option>
            </select>

            <select id="Visualize">
                <option value="1">Read File</option>
                <option value="2">Live</option>
            </select>
```

```
30
31          <div id="LoadButtons">
32              <form id="LoadCSV">
33                  <label for="files">Select CSV to Read:</label>
34                  <input type="file" id="files" accept=".csv" required />
35                  <button type="submit" id="submit-file">Load</button>
36              </form>
37          </div>
38      </div>
39
40      <div class="container" id="container" hidden>
41      </div>
42
43      <script src="js/three.js"></script>
44      <script src="js/loaders/OBJLoader.js"></script>
45      <script src="js/controls/OrbitControls.js"></script>
46      <script src="js/canvas/simpleheat.js"></script>
47      <script src="js/canvas/data.js"></script>
48      <script src="js/jquery.min.js"></script>
49      <script src="js/papaparse.min.js"></script>
50      <script src="js/canvas/createCanvas.js"></script>
51      <script src="stats.min.js"></script>
52
53      <script>
54          var timeStart = new Date().getTime();
55          var Gain = document.getElementById("Gain");
56          var displayType =  document.getElementById("Visualize"); //1: Lectura, 2: Directo
57          var whatWave = document.getElementById("Selector"); //0: Theta, 1: Alpha, 2: LowBeta,
    3: HighBeta, 4: Gamma
58
59          var attempts = 0;
60          var wave = [];
61          var waves = [];
62          var CSV = [];
63          var waveCSV = [];
64
65          var isLoaded = 0;
66          var milis;
67          var camera, scene, renderer;
68          var BrainL, BrainR;
69          var windowHalfX = window.innerWidth / 2;
70          var windowHalfY = window.innerHeight / 2;
71          var LeftTexture, RigthTexture;
72          var container;
73          var stats;
74          createAllCanvas();
75          init();
76          animate();
77
78          function init() {
79              container = document.createElement( 'div' );
80              document.body.appendChild( container );
```

```
81
82                createCamera();
83                stats = new Stats();
84                container.appendChild( stats.dom );
85                createScene();
86                createObjects();
87                createRender();
88
89                window.addEventListener( 'resize', onWindowResize, false );
90            }
91
92            function createCamera() {
93                camera = new THREE.PerspectiveCamera( 45, window.innerWidth / window.innerHeight,
      1, 400 );
94                camera.position.z = 10;
95                var controls = new THREE.OrbitControls( camera, container );
96            }
97            function createScene() {
98                scene = new THREE.Scene();
99
100               scene.background = new THREE.Color( 0xf0f0f0 );
101
102               var ambientLight = new THREE.AmbientLight( 0xcccccc, 0.4 );
103               scene.add( ambientLight );
104
105               var pointLight = new THREE.PointLight( 0xffffff, 0.8 );
106               camera.add( pointLight );
107               scene.add( camera );
108           }
109           function createObjects() {
110               var manager = new THREE.LoadingManager();
111               manager.onProgress = function ( item, loaded, total ) {
112                   console.log( item, loaded, total );
113               };
114
115               LeftTexture = new THREE.Texture(canLeft);
116               RigthTexture = new THREE.Texture(canRigth);
117
118               var onProgress = function ( xhr ) {
119                   if ( xhr.lengthComputable ) {
120                       var percentComplete = xhr.loaded / xhr.total * 100;
121                       console.log( Math.round(percentComplete, 2) + '% downloaded' );
122                   }
123               };
124
125               var onError = function ( xhr ) {
126               };
127
128               var loader = new THREE.OBJLoader( manager );
129               loader.load( 'obj/BrainL.obj', function ( object ) {
130                   BrainL = object;
131                   BrainL.traverse( function ( child ) {
```

```
                        if (child instanceof THREE.Mesh) {
                            child.material.map = LeftTexture;
                            child.material.needsUpdate = true;
                            child.visible = false;
                        }
                    });
                    scene.add( object );
                }, onProgress, onError );

                loader.load( 'obj/BrainR.obj', function ( object ) {
                    BrainR = object;
                    BrainR.traverse( function ( child ) {
                        if (child instanceof THREE.Mesh) {
                            child.material.map = RigthTexture;
                            child.material.needsUpdate = true;
                            child.visible = false;
                        }
                    });
                    scene.add( object );
                }, onProgress, onError );

                manager.onLoad = function ( ) {
                    console.log( 'Loading complete!');
                    document.getElementById("loader").style.display = "none"; //hide loader
                    BrainL.traverse( function ( child ) {
                        if (child instanceof THREE.Mesh) {
                            child.visible = true;
                        }
                    });
                    BrainR.traverse( function ( child ) {
                        if (child instanceof THREE.Mesh) {
                            child.visible = true;
                        }
                    });
                };
            }

            function createRender() {
                renderer = new THREE.WebGLRenderer();
                renderer.setPixelRatio( window.devicePixelRatio );
                renderer.setSize( window.innerWidth, window.innerHeight );
                container.appendChild( renderer.domElement );
            }

            $('#Visualize').on('change', function() {
                resetValues();
                CSV = [];
                isLoaded = 0;
                if (displayType.value == "1"){
                    document.getElementById("infoTxt").innerHTML = "Select CSV";
                    document.getElementById("LoadButtons").style.display = "block";
                }else{
```

```
184                     document.getElementById("LoadButtons").style.display = "none";
185                 }
186             })
187 
188             $('#showDots').on('change', function() {
189                 updateCanvas(leftValues, rigthValues);
190             })
191 
192             $(document).ready(function(){
193                 $('#submit-file').on("click",function(e){
194                     e.preventDefault();
195                     $('#files').parse({
196                         config: {
197                             delimiter: "auto",
198                             complete: ReadData,
199                         },
200                         complete: function()
201                         {
202                             waveCSV = CSV.shift();
203                             isLoaded = 1;
204                             document.getElementById("LoadButtons").style.display = "none";
205                             timeStart = new Date().getTime();
206                         }
207                     });
208                 });
209 
210                 function ReadData(results){
211 
212                     var DatosCSV = results.data;
213                     var timestamp = 0;
214                     var position = 0;
215                     for(i=1;i<DatosCSV.length;i++){
216                         var row = DatosCSV[i];
217                         var cells = row.join(",").split(",");
218                         if(timestamp != cells[0]){
219                             if(timestamp != 0){
220                                 timestamp = cells[0];
221                                 CSV.push(waves);
222                                 waves = [];
223                                 position = 0;
224                             }else timestamp = cells[0];
225                         }
226                         for(j=0;j<cells.length;j++){
227                             wave.push(cells[j]);
228                         }
229                         waves[position] = wave;
230                         position++;
231                         wave = [];
232                     }
233                 }
234             });
235 
```

```
236             function UpdateData() {
237                 var now = new Date().getTime();
238                 if (displayType.value == 1 && isLoaded == 1){
239                     if (CSV.length>0){
240                         milis =  millisecondsToSeconds(now - timeStart);
241                         document.getElementById("infoTxt").innerHTML = parseInt(milis);
242                     } else {
243                         isLoaded = 0;
244                         document.getElementById("infoTxt").innerHTML = "Session Ended";
245                         resetValues();
246                         document.getElementById("LoadButtons").style.display = "block";
247                     }
248 
249                     if (milis > waveCSV[whatWave.value][0]){
250                         waveCSV = CSV.shift();
251                         leftValues = waveCSV[whatWave.value].slice(1,8);
252                         rigthValues = waveCSV[whatWave.value].slice(8,15).reverse();
253                         updateCanvas(leftValues, rigthValues);
254                     }
255                 }
256 
257                 if (displayType.value == 2){
258                     milis =  now - timeStart;
259                     if (milis > 100){
260                         doPostPython();
261                         timeStart = new Date().getTime();
262                     }
263                 }
264             }
265 
266             function doPostPython() {
267                 var xhttp = new XMLHttpRequest();
268                 xhttp.onreadystatechange = function() {
269                     if (this.readyState == 4 && this.status == 200) {
270 
271                         var rs = this.responseText;
272                         if (rs == "No activity detected" && displayType.value == "2"){
273                             document.getElementById("infoTxt").innerHTML = rs;
274                         }
275                         if (rs != ""){
276                             if (rs != "None" && displayType.value == "2"){
277                                     rs = rs.split(", ");
278                                     leftValues = rs.slice(1,8);
279                                     rigthValues = rs.slice(8,15).reverse();
280                                     updateCanvas(leftValues, rigthValues);
281                                     document.getElementById("infoTxt").innerHTML = "Reading Data
    ";
282                                     attempts = 0;
283                             }else{
284                                 attempts++;
285                             }
286                         }
```

```
                        if (attempts>25){
                            document.getElementById("infoTxt").innerHTML = "No activity detected
    ";
                            resetValues();
                        }
                    }
                };
                xhttp.open("POST", "/send", true);
                xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
                xhttp.send(document.getElementById("Selector").value);
            }

            function millisecondsToSeconds(milliseconds)
            {
                var seconds = milliseconds / 1000;
                return seconds;
            }

            function onWindowResize() {

                windowHalfX = window.innerWidth / 2;
                windowHalfY = window.innerHeight / 2;
                camera.aspect = window.innerWidth / window.innerHeight;
                camera.updateProjectionMatrix();
                renderer.setSize( window.innerWidth, window.innerHeight );
            }

            function animate() {
                requestAnimationFrame( animate );
                UpdateData();
                LeftTexture.needsUpdate = true;
                RigthTexture.needsUpdate = true;
                render();
                stats.update();
            }

            function render() {
                renderer.render( scene, camera );
            }

        </script>
    </body>
</html>
```

Bibliografía

[Cicchino, 2013] Cicchino, A. N. B. (2013). *Técnicas de procesamiento de EEG para detección de eventos*. Universidad Nacional de la Plata.

[Haas, 2003] Haas, L. F. (2003). *Hans Berger (1873–1941), Richard Caton (1842–1926), and electroencephalography*. Journal of Neurology, Neurosurgery & Psychiatry.

[Larsen, 2011] Larsen, E. A. (2011). *Classification of EEG Signals in a BrainComputer Interface System*. Norwegian University of Science and Technology.

[Leonardo, 2002] Leonardo, P. S. (2002). *Breve historia de la electroencefalografía*. Acta Neurol Colomb.

[Minguez, 2008] Minguez, J. (2008). *Tecnología de interfaz cerebro-computador*. Universidad de Zaragoza.

[Muñoz, 2014] Muñoz, C. N. H. (2014). *Estudio de Técnicas de análisis y clasificación de señales EEG en el contexto de Sistemas BCI (Brain Computer Interface)*. Universidad Autónoma de Madrid.

[Pivik et al., 1993] Pivik, R., Brougton, R., Coppola, R., Davidson, R., Fox, N., and Nuwer, M. (1993). *Guidelines for the recording and quantitative analysis of electroencephalographic activity in research contexts*. Psychophysiology, 30.

[Velasco, 2013] Velasco, P. M. G. (2013). *Influencia de la estimulación sonora binaural en la generación de ondas cerebrales. Estudio electroencefalográfico*. Universidad Complutense de Madrid.