Facultad
de
Ciencias

APLICACIÓN WEB AAR PARA UN VIDEOJUEGO DE SIMULACIÓN MILITAR

(AAR web application for a military simulation video game)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Francisco Álamo García

Director: Patricia López Martínez

Febrero - 2022

Agradecimientos

Lo primero que quiero es agradecer a mi familia que me ha apoyado a lo largo de esta etapa y me han dado fuerzas para continuar.

También quiero dar las gracias a mis compañeros que han estado ahí, en los momentos difíciles, pero también en los momentos felices, muchas gracias a todos.

Por último, me gustaría agradecer a los profesores por su esfuerzo bajo la pandemia, en particular a mi tutora, sin su ayuda hubiera sido muy complicado haber terminado este trabajo.

Muchas gracias a todos.

Índice de contenido

1. Introducción 9
2. Herramientas, tecnologías, materiales y metodologías utilizadas 11
   2.1 Herramientas 11
      2.1.1 Visual Studio Code 11
      2.1.2 Blender 11
      2.1.3 MongoDBCompass 11
   2.2 Tecnologías 12
      2.2.1 Node.js 12
      2.2.2 React 12
      2.2.3 TypeScript 12
      2.2.4 Ionic 12
      2.2.5 Three.js 13
      2.2.6 Nginx 12
   2.3 Materiales utilizados 13
      2.3.1 Objetos 3D 13
      2.3.2 Texturas 14
   2.4 Metodología 14
      2.4.1 Metodología iterativa incremental con prototipado 14
      2.4.2 Formación 15
3. Análisis de requisitos 16
   3.1 Análisis de la aplicación 16
   3.2 Requisitos funcionales 16
   3.3 Requisitos no funcionales 17
4. Diseño e implementación 19
   4.1 Arquitectura del sistema 19
   4.2 Diseño e implementación de la capa de presentación 20
      4.2.1 Pantalla principal 20
      4.2.2 Representación 3D 22
   4.3 Diseño e implementación de la capa de negocio 23
      4.3.1 Diseño e implementación de la API web 24
      4.3.2 Extensión de obtención de datos 26
   4.4 Diseño e implementación de la capa de persistencia 27
5. Despliegue 28
6. Evaluación y pruebas 30

6.1 Pruebas unitarias 30
6.2 Pruebas de integración 30
6.3 Pruebas de sistema 31
7. Conclusiones y trabajos futuros 33
7.1 Conclusiones 33
7.2 Trabajos futuros 33
8. Referencias 34

Índice de ilustraciones

Figura 1 - Zona de briefing y debriefing 9
Figura 2 - Comment cards 10
Figura 3 - Proyecto de Blender 11
Figura 4 - Conjunto de elementos 3D 13
Figura 5 - Iteraciones incrementales 14
Figura 6 - Arquitectura del sistema 19
Figura 7 – Diagrama de navegación 20
Figura 8 - Elementos interactivos 21
Figura 9 - Modo oscuro y modo claro 21
Figura 10 - Representación 3D 22
Figura 11 - Canvas donde se dibujan todos los elementos 3D en React 23
Figura 12 - Función para generar el token JWT 24
Figura 13 - Función para comprobar que el usuario es administrador 25
Figura 14 - Ejemplo de llamadas a la extensión 26
Figura 15 – Método RVExtensionArgs 26
Figura 16 - Función que inicializa el sistema de MongoDB 27
Figura 17 - Diagrama de clases 27
Figura 18 - Panel de control de Proxmox 28
Figura 19 - Diseño físico 29
Figura 20 - Ejemplo de prueba unitaria de componente React 30
Figura 21 - Ejemplo de prueba de integración 31
Figura 22 - Gráfica de uso de memoria 31
Figura 23 - Informe de Lighthouse 32

Índice de tablas

Tabla 1 - Requisitos funcionales 16
Tabla 2 - Requisitos no funcionales 17
Tabla 3 – Diseño de la API 24

Resumen

Almacenar eventos pasados es una de las mejores maneras de evitar los errores del futuro ya que gracias a ellos es posible adelantarse a los acontecimientos y con ello ayudar a optimizar los recursos. El AAR (After Action Review) es una estrategia de revisión de acciones, que se realiza al acabar un proyecto y que permite visualizar, normalmente en forma de gráficas, las acciones realizadas y mediante una serie de preguntas, obtener y analizar los resultados del proyecto.

En este Trabajo de Fin de Grado se va a aplicar AAR sobre un videojuego para saber cuáles han sido las acciones realizadas, aprender de los errores y tratar de revertirlos. El juego en cuestión es Arma 3, un videojuego de simulación bélica, por lo que el AAR que se puede aplicar es el mismo que se realiza en ejercicios militares.

En concreto se van a poder obtener, para cada partida, la posición de los jugadores, su dirección, las acciones que realizan y el momento en el que se realizan esas acciones. Todos estos datos serán extraídos a través de una extensión que será conectada a Arma 3 y almacenados en una base de datos NoSQL MongoDB. Para facilitar la obtención de estos datos desde el front-end, modelados en un formato específico, será necesario el desarrollo de una API web. Para la parte de front-end se va a utilizar Node.js con React como framework de interfaz e Ionic como framework de componentes de interfaz de usuario. Por último, para mostrar los datos de la base de datos se usará Three.js, que permitirá renderizar objetos 3D en la propia web. Gracias a esto, se podrá renderizar el terreno en el que se jugó una partida y también se podrá mostrar la posición de los jugadores y su dirección en cada momento de la partida.

Palabras clave: Aplicación web, API web, AAR, Arma 3.

Abstract

Storing past events is one of the best ways to avoid future mistakes, since thanks to them it is possible to anticipate events and thus help optimize resources. The AAR (After Action Review) is an action review strategy, which is carried out at the end of a project and which allows visualizing, usually in the form of graphs, the actions carried out and through a series of questions, obtaining the results of the project.

In this Final Degree Project, AAR will be applied to a video game to find out what actions have been taken, how to learn from mistakes and how to reverse them. The game in question is Arma 3, a war simulation video game, so the AAR that can be applied is the same as that performed in military exercises.

Specifically, it will be possible to obtain, for each game, the position of the players, their direction, the actions they carry out and the moment in which those actions are carried out. All this data will be extracted through an extension that will be connected to Arma 3 and which will store all the data in a NoSQL MongoDB database. To facilitate obtaining this data from the front-end, modeled in a specific format, it will be necessary to develop a web API. For the front-end part, Node.js will be used together with React as the interface framework and Ionic as the user interface component framework. Finally, to display the data from the database, Three.js will be used, which will allow rendering 3D objects on the web itself. Thanks to this, it will be possible to render the terrain in which a game was played and it will also be possible to show the position of the players and their direction at each moment of the game.

Key words: Web application, Web API, AAR, Arma 3.

1. Introducción

El almacenamiento de conocimiento y su futura transmisión y remodelación son necesarios para seguir avanzando como sociedad. Esto también se aplica a los diferentes proyectos que existen tanto en ingeniería como en la vida personal. El cómo analizar este conocimiento, cómo transmitirlo y mejorarlo es uno de los principios básicos del After Action Review. Este proceso fue creado por el ejército de EE. UU. en 1993 [1] para evitar al máximo posible errores humanos en combate y así evitar muertes. Este proceso se lleva a cabo principalmente en entrenamientos, para mejorar la estrategia y por lo tanto el rendimiento. El proceso se lleva a cabo en dos fases principales, *briefing* y *debriefing*, en una zona designada para ello [ver Figura 1]. En el caso práctico al que se aplica AAR en este trabajo, este proceso se realiza de forma online.

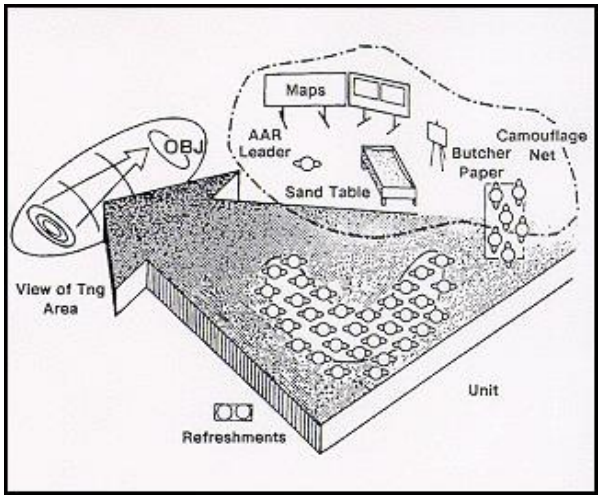

Figura 1 - Zona de briefing y debriefing

El *briefing* es la fase inicial antes de una misión o de un entrenamiento. Es en esta fase donde se explican las tareas a realizar en la misión o entrenamiento y qué esperar de ellas. Esta fase es muy importante, porque si no se realiza correctamente, puede que no se puedan identificar errores en el proceso ya que las tareas no se habían entendido correctamente en un principio y con ello se puedan contaminar los resultados del AAR. Las tareas deben de ser lo más específicas posible para poder calcular con datos empíricos (como el tiempo) los resultados y así ayudar a obtener un resultado lo más realista posible, ya que después estos datos deben ser contrastados con datos anteriores y con datos externos.

En el *debriefing* se presentan y analizan los datos de la misión. Por ejemplo, el tiempo en el que se ha realizado la misión o la precisión con la que se ha realizado. En primer lugar, se comentan los errores observados y como subsanarlos. Después se comprueban los valores empíricos de la misión y como se podrían mejorar. Una de las maneras de presentar los datos son las *comment cards*, como las que se observan en la Figura 2, que son un conjunto de textos en los que se detallan las acciones realizadas y los fallos encontrados. Es aquí donde la aplicación desarrollada en este Trabajo de Fin de Grado es de gran ayuda.

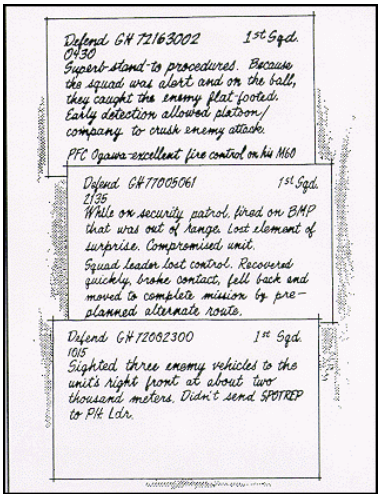

Figura 2 - Comment cards

1.1 Objetivo

El objetivo de este trabajo es principalmente obtener datos de las misiones y de las tareas asignadas en cada una para poder compararlas entre las diferentes misiones realizadas en partidas dentro del videojuego Arma 3. Para obtener esos datos, será necesario desarrollar una extensión que conectada al servidor del videojuego será quien monitorice los movimientos de los jugadores y los registre en una base de datos. A continuación, una aplicación web mostrará estos datos en una interfaz 3D para conseguir una mejor visualización de los movimientos de los jugadores y de los posibles errores cometidos, con el objetivo de proporcionar una mejor base para el análisis en el *debriefing*. Esta aplicación se comunicará con el backend mediante un servicio web para facilitar su uso y aumentar la seguridad de las transacciones con la base de datos.

1.2 Organización de la memoria

La memoria se organiza siguiendo la estructura habitual de este tipo de trabajos:

- Primero, en el capítulo 2, se definen las herramientas, tecnologías, materiales y metodologías utilizadas en el proyecto, las cuales han sido determinantes, ya que, sin estas, no habría sido posible su realización.
- Después, el capítulo 3 explica el análisis de requisitos realizado para la correcta definición de las funcionalidades necesarias para el correcto funcionamiento y aceptación de la aplicación del proyecto.
- En el capítulo 4 se exponen el diseño y la implementación aplicados en el proyecto para cumplir con los requisitos analizados previamente.
- El capítulo 5 detalla cómo se ha realizado el despliegue del código.
- En el capítulo 6 se especifican las diferentes pruebas realizadas para la comprobación del correcto funcionamiento del código desarrollado.
- Por último, se realizan las conclusiones del proyecto observando las dificultades que se han ido produciendo durante el desarrollo del proyecto, así como los trabajos futuros que se pueden realizar en el proyecto.

2. Herramientas, tecnologías, materiales y metodologías utilizadas

En este apartado se definen las herramientas utilizadas, las tecnologías y los materiales gráficos 3D y 2D usados para la realización de este trabajo y la metodología seguida para su desarrollo.

2.1 Herramientas

2.1.1 Visual Studio Code

Visual Studio Code es un IDE (*Integrated Development Environment*) extensamente utilizado en diferentes proyectos software. Es un editor de texto con extensiones que permiten aumentar sus capacidades básicas adaptándose a cualquier lenguaje de programación, permitiendo programar en múltiples lenguajes, al mismo tiempo en un solo programa. Esta herramienta ha sido usada para desarrollar el código de React utilizando múltiples extensiones para facilitar el desarrollo en TypeScript.

2.1.2 Blender

Blender es un programa informático multiplataforma, dedicado especialmente al modelado, iluminación, renderizado, animación y creación de gráficos tridimensionales [2]. También de composición digital utilizando la técnica procesal de nodos, edición de vídeo, escultura y pintura digital. Gracias a Blender, se pueden crear objetos 3D o editarlos para mejorarlos, como se observa en la Figura 3, gracias al conjunto de scripts que incluye y que permiten "limpiar" objetos 3D para mejorar su rendimiento tanto en peso, como en renderizado. También ha permitido convertir objetos 3D de terrenos en formato FBX a OBJ que tiene un menor peso y permite el uso de texturas en el propio fichero, sin necesidad de tener otros ficheros con metadatos del objeto.

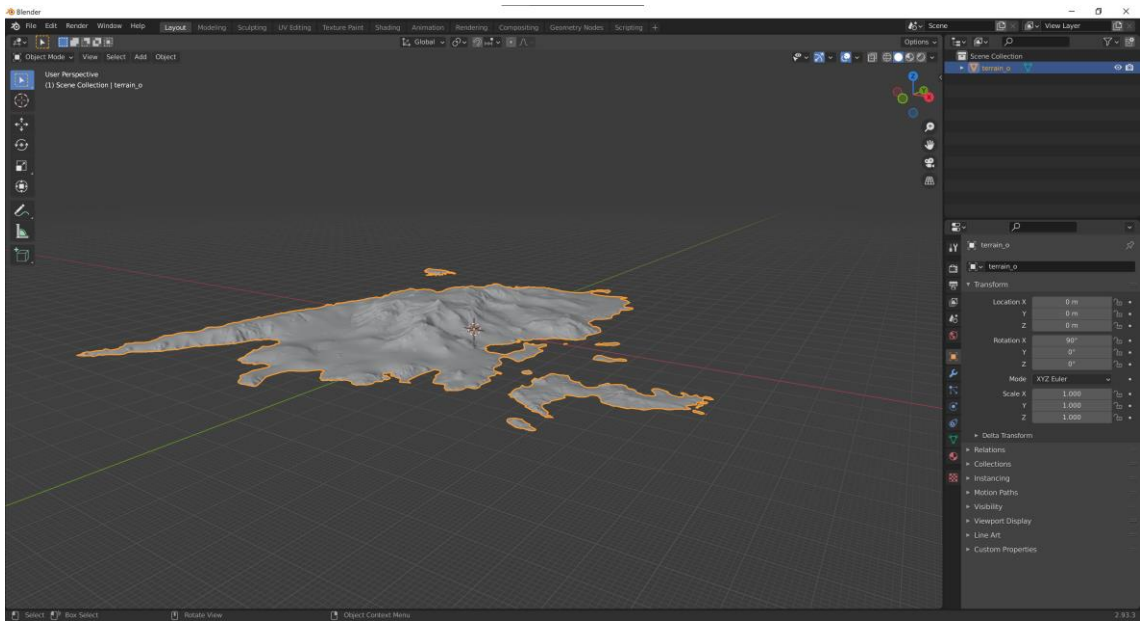

Figura 3 - Proyecto de Blender

2.1.3 MongoDBCompass

MongoDBCompass es un programa que permite observar, controlar y analizar de forma interactiva los datos almacenados en una base de datos MongoDB [14], mediante una interfaz muy fácil de usar. Gracias a este programa se ha podido observar el uso de la base de datos y la

calidad de los datos almacenados, para evitar duplicidades y calcular el peso de los datos obtenidos durante 3h con 60 jugadores y así obtener el coste en rendimiento que tendrá que soportar el servidor.

2.2 Tecnologías

En este apartado se definen y explican las diferentes tecnologías usadas en el proyecto.

2.2.1 Node.js

Node.js es un entorno de ejecución usando el motor de JavaScript de Chrome llamado V8 [3]. Este entorno orientado a eventos asíncronos está diseñado principalmente para aplicaciones de red, pero en la actualidad se usa para múltiples aplicaciones diferentes, no solo de red. Gracias a complementos o paquetes de terceros se puede conseguir que este entorno realice acciones y ayude a crear aplicaciones gráficas.

2.2.2 React

React es una librería hecha en Javascript para la construcción de interfaces de usuario [4]. Sus principales características son, de un lado, su carácter declarativo (es decir, lo que se programa es lo que se ve) y, de otro, su capacidad de integración con el motor de renderizado del navegador. Esta última característica es la que hace de React una opción de elección para la creación de interfaces de usuario en la web. React permite trabajar con el DOM (Document Object Model) como si fuera una muestra de componentes con estado (propiedades de usuario). En este contexto, el DOM se convierte en una copia de la información de estado de un componente, y no en una representación de su código. Esto es de vital importancia para mejorar el rendimiento en la representación de componentes visuales dentro de la web.

2.2.3 TypeScript

TypeScript es un lenguaje de programación que se ejecuta en el navegador. Es un lenguaje de programación interpretado, lo que significa que el código se ejecuta en el navegador y no se compila. Desarrollado por Microsoft, TypeScript es un lenguaje de programación derivado de JavaScript [5]. En TypeScript se pueden utilizar características de clases, interfaces, tipos, generics, etc.

La principal diferencia entre TypeScript y JavaScript es que TypeScript es un lenguaje de programación estático con tipos. Cuando se escribe un programa en TypeScript, se puede comprobar si el programa tiene errores de tipo antes de ejecutarlo. Esto puede reducir el tiempo de desarrollo, y aumentar la calidad del software.

2.2.4 Ionic

Ionic es una librería de código abierto que permite el desarrollo de aplicaciones móviles para dispositivos iOS y Android. Está basada en AngularJS y se puede utilizar en conjunto con Cordova / Capacitor para crear aplicaciones web-apps para iOS y Android [7].

La librería Ionic está pensada para la creación de aplicaciones web. Esta contiene múltiples componentes React que pueden ser usados de forma sencilla para generar componentes de interfaz de usuario y ayudar a mantener el mismo formato tanto en dispositivos móviles como en ordenador. Estos componentes son compatibles con el lenguaje de programación TypeScript.

2.2.5 Nginx

Nginx es un servicio de código abierto que permite la creación de un servidor web, un proxy inverso y un balanceador de carga de alto rendimiento que pueden ejecutarse en múltiples

plataformas como Windows o Unix. Es largamente utilizado en varios proyectos online como Netflix o Github ya que permite escalar mucho más fácilmente que servidores estilo Apache. En este proyecto se han utilizado el proxy inverso para permitir una comunicación exterior segura y el balanceador de carga para evitar la sobrecarga del sistema y fallos de los nodos.

2.2.6 Three.js

Three.js es una librería de código abierto que permite crear gráficos 3D en JavaScript [6]. Three.js se compone de tres partes:

- Una API de JavaScript con la que podemos crear gráficos 3D.
- Un motor de renderizado llamado WebGL que permite a los navegadores renderizar gráficos 3D.
- Una API de renderizado llamada GLGE que permite a los navegadores renderizar gráficos 3D sin depender de WebGL.

Gracias a librerías de terceros como React-three fiber, podemos crear código en TypeScript usando componentes propios de React para facilitar el uso de objetos Three.js y evitar complejidades en la construcción de elementos 3D. Esto nos permite crear componentes complejos, como el que se puede observar en la Figura 4, de una manera rápida y sin complicaciones.

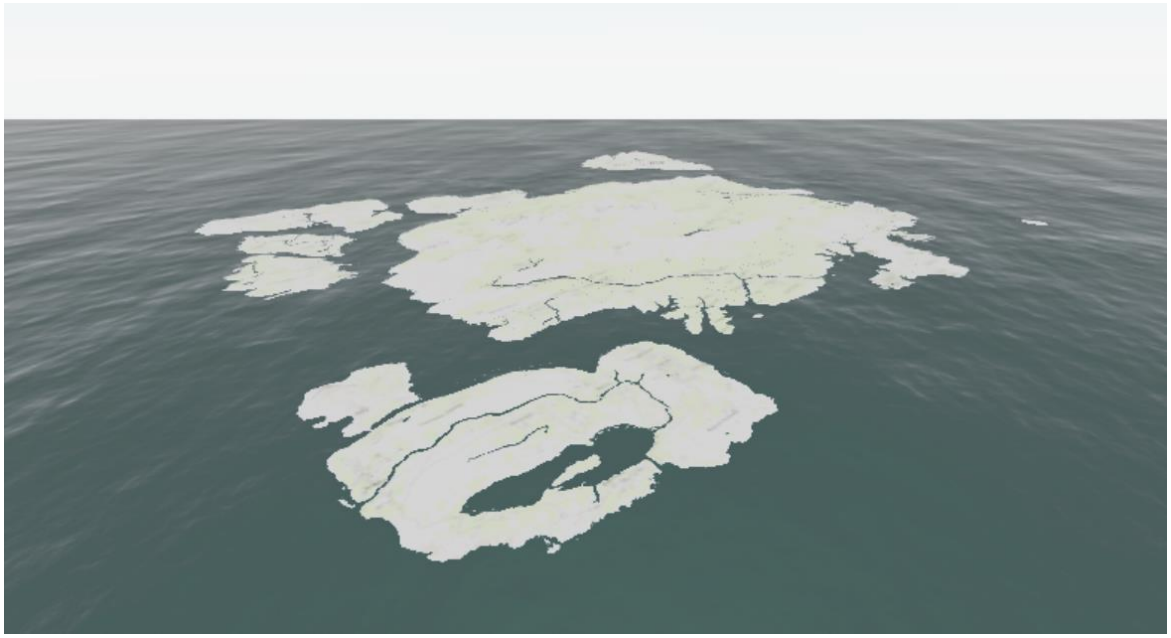

Figura 4 - Conjunto de elementos 3D

2.3 Materiales utilizados

Para este proyecto se han usado principalmente 2 materiales: objetos 3D y texturas.

2.3.1 Objetos 3D

Los objetos 3D son extraídos del juego directamente en formato XYZ. Cada línea del archivo significa un punto de altura, por ejemplo, el punto x=0, y= 1 y z=2 y el resultado sería “0 1 2”. Este formato es fácilmente transformable a un objeto 3D gracias a Blender y a la extensión de importación de XYZ [8].

2.3.2 Texturas

También las texturas han sido extraídas directamente del terreno del juego en formato SVG y han sido transformadas a JPG para los componentes de misiones de la página principal. Se ha usado la versión SVG para texturizar los objetos 3D mediante un archivo MTL.

2.4 Metodología

En este apartado se define la metodología llevada a cabo en este proyecto, que, en este caso, ha sido la iterativa incremental junto con una fase inicial de prototipado.

2.4.1 Metodología iterativa incremental con prototipado

En este proyecto se ha aplicado la metodología software iterativa incremental con un prototipado inicial para definir los requisitos de cada componente. Una vez el prototipo inicial fue fijado, se continuó con iteraciones incrementales para implementar nuevas funcionalidades y mantener una continua evolución del código. Cada fase se ha ido adaptando a la complejidad de cada componente y a los desafíos encontrados durante su desarrollo.

En cada iteración se realizaban las fases de análisis, diseño, implementación, pruebas y publicación, como se puede observar en la Figura 5:

- Análisis: En esta etapa se definen los componentes a desarrollar y los objetivos de estos. Gracias al prototipado previo se definió la estructura inicial y gracias a tener esa estructura inicial fue más sencillo analizar los nuevos componentes.
- Diseño: En esta etapa, una vez analizados los componentes a desarrollar y sus objetivos, era necesario idear nuevos componentes en la estructura ya existente sin que afectase al normal funcionamiento del resto de los componentes.
- Implementación: Durante esta etapa, se realiza su implementación siguiendo el diseño previo para alcanzar los objetivos requeridos.
- Pruebas: Se realizan pruebas a nivel de componente para evitar errores de los componentes en casos específicos, por ejemplo, que la API no esté disponible o diferentes combinaciones de casos de error.
- Publicación: En este proyecto se ha decidido tener 2 entornos de desarrollo, aceptación y producción. Gracias a esta estructura, los nuevos componentes que han de ser probados son probados en aceptación sin afectar al entorno de producción y sin afectar a los usuarios regulares. A este entorno solo pueden acceder los *testers*.

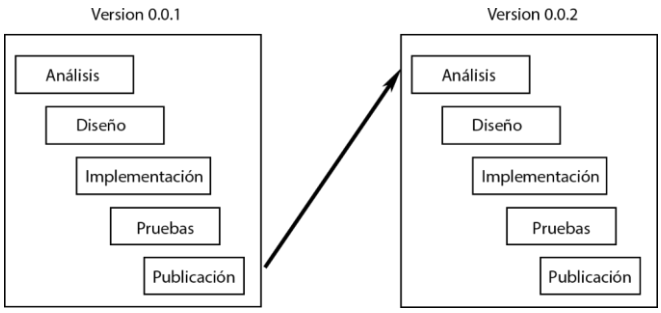

Figura 5 - Iteraciones incrementales

2.4.2 Formación

Para la realización de este proyecto se ha necesitado aprender nuevas tecnologías para la correcta implementación de nuevos componentes, como la utilización de ThreeJS para el componente 3D o Ionic para los componentes visuales. Esta formación se ha intercalado con el propio desarrollo de la aplicación ya que, sobre todo en el apartado de pruebas, se han ido descubriendo errores y su solución necesitaban una formación previa.

3. Análisis de requisitos

En este apartado se describen los diferentes requisitos necesarios para conseguir una aplicación con la calidad deseada. Los requisitos han sido extraídos de encuestas informales a los posibles usuarios de la aplicación, además de las propias ideas del autor y de la directora del proyecto.

3.1 Análisis de la aplicación

La aplicación contará con un conjunto de componentes, llamados misiones. Estas misiones se podrán observar en una representación 3D, que no será más que el modelo 3D del terreno y las unidades que participan en esa misión, en cada lugar y momento de la partida. Para observar fácilmente estas unidades, se representarán como flechas que indican la posición y su dirección en cada momento. Los eventos serán los disparos, que serán representados como líneas.

3.2 Requisitos funcionales

Los requisitos funcionales definen las diferentes funcionalidades que debe proporcionar la aplicación. Estos requisitos están expuestos en la Tabla 1.

Tabla 1 - Requisitos funcionales

ID	Descripción
RF01	El usuario podrá acceder como administrador.
RF02	El usuario administrador podrá ocultar misiones.
RF03	El usuario podrá elegir entre el modo claro y el modo oscuro.
RF04	Se usarán las preferencias del sistema del usuario para definir el modo claro o el modo oscuro.
RF05	El usuario podrá buscar una misión por su nombre o terreno.
RF06	El usuario podrá seleccionar una misión para obtener sus metadatos.
RF07	El usuario podrá abrir la misión.
RF08	La representación de la misión deberá ser 3D.
RF09	El usuario debe ser capaz de rotar la cámara de la representación 3D.
RF10	El usuario debe ser capaz de trasladar la cámara de la representación 3D.
RF11	El usuario debe ser capaz de alejar y acercar la cámara en la representación 3D.
RF12	El usuario podrá pausar el avance en el tiempo de la misión.
RF13	El usuario podrá reanudar el avance en el tiempo de la misión.

RF14	El usuario podrá aumentar la velocidad de reproducción.
RF15	El usuario podrá ver los controles de la cámara.
RF16	El usuario podrá seleccionar el momento en el tiempo en la barra temporal.
RF17	Las unidades se mostrarán como flechas usando su posición y dirección.
RF18	Las unidades que son jugadores mostrarán su nombre encima de la flecha.
RF19	Las unidades se mostrarán por colores dependiendo de su bando.
RF20	Las unidades muertas se mostrarán con color negro.
RF21	Los eventos de disparos se deberán mostrar como una línea del color del bando.
RF22	Los eventos de disparos deberán mostrar el inicio y final del disparo.
RF23	Para la representación 3D se usará el terreno de la misión junto con su textura no satelital.
RF24	El usuario podrá volver a la página principal.
RF25	El usuario administrador podrá registrar nuevos administradores.

3.3 Requisitos no funcionales

Los requisitos no funcionales son aquellos que no definen funciones del sistema, sino otras propiedades del sistema que contribuyen a su calidad. Estos requisitos se recogen en la Tabla 2.

Tabla 2 - Requisitos no funcionales

ID	Descripción	Tipo	Importancia
RNF01	La aplicación debe poder ser ejecutada en un navegador web. Como mínimo deberá poder ser ejecutada correctamente en Chrome, Firefox, Edge y Safari.	Portabilidad	Muy alta
RNF02	La aplicación debe usar menos de 1GB de RAM.	Rendimiento	Alta
RNF03	La aplicación debe cargar en menos de 5 segundos.	Rendimiento	Media
RNF04	La aplicación debe usar menos del 75% de GPU.	Rendimiento	Media

RNF05	La aplicación debe de adaptarse a cualquier resolución de pantalla superior a 320 pixeles de ancho.	Adaptabilidad	Alta
RNF06	La aplicación debe permitir al usuario seleccionar un idioma, entre español e inglés.	Accesibilidad	Muy alta

4. Diseño e implementación

En este apartado se definen la estructura general del sistema, descompuesto en subsistemas, para después definir cada subsistema en detalle y describir como se han implementado para conseguir una correcta comunicación entre ellos.

4.1 Arquitectura del sistema

El patrón arquitectónico usado en este proyecto es el patrón de capas. En este caso se han usado las tres capas software habituales, la capa de presentación, la capa de negocio y la capa de persistencia, como se puede observar en la Figura 6. Gracias a este patrón se pueden realizar cambios en cada una de las capas sin afectar al resto de capas y probar cada capa y la conectividad entre ellas de manera individual.

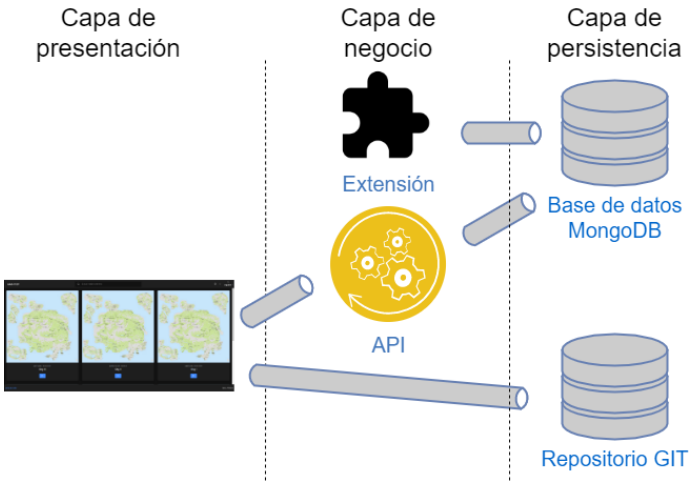

Figura 6 - Arquitectura del sistema

La capa de presentación es la capa superior, a través de la que muestra la información al cliente. Es en esta capa en la que el cliente puede realizar acciones de manera intuitiva sin acceder a la capa de negocio o a la capa de persistencia. Esta capa está desarrollada en TypeScript, React e Ionic.

La capa de negocio es la capa intermedia que sirve como conector entre la capa de presentación y la capa de persistencia. En este proyecto la capa de negocio se implementa en base a un servicio web creado en JavaScript usando Express. Es también en esta capa en la que se realiza la extracción de los datos de la partida de cada jugador para almacenarlos en la capa de persistencia usando para ello un extensor programado en C#.

La capa de persistencia es la capa final donde se almacenan todos los datos necesarios para el normal funcionamiento de la aplicación. Los elementos almacenados son los siguientes: usuarios, partidas, datos de la partida, objectos 3D, texturas del objecto 3D y la imagen del terreno que sirve como vista previa. Los datos de las partidas se almacenan en una base de datos MongoDB, mientras que los datos de las texturas y objectos 3D están almacenados en un repositorio Git, para conseguir una mayor eficiencia a la hora de actualizar modelos 3D o texturas, ya que, aunque MongoDB puede trabajar con objetos binarios, el coste de transacción es mayor que si se usa un repositorio Git. Además, es preferible separar la carga para evitar cuellos de botella y mejorar el rendimiento, a la vez que nos permite hacer peticiones a la base de datos y al repositorio en paralelo.

4.2 Diseño e implementación de la capa de presentación

La capa de presentación es la interfaz a través de la que el usuario es capaz de interactuar con el sistema sin acceder a las capas inferiores. Esta capa debe ser transparente de cara al usuario, es decir, no debe de saber de dónde salen los datos, sino que debe ser capaz de visualizar los datos almacenados en la capa de persistencia de una manera intuitiva y fácil de manejar.

Esta aplicación está diseñada para ser una SPA (*Single-page application*) [9] y hacer uso de modales para realizar las diferentes acciones de la aplicación, como se puede observar en la Figura 7, que representa las distintas opciones de navegación a través de la aplicación.

La interfaz se diseña en base a dos componentes principales:

- La pantalla principal, que permite visualizar una vista previa de las diferentes misiones realizadas.
- El modal de representación 3D, que permite observar los modelos 3D de los terrenos para una mejor identificación de los jugadores y de sus movimientos.

A continuación, se explican detalles sobre el diseño de ambos componentes.

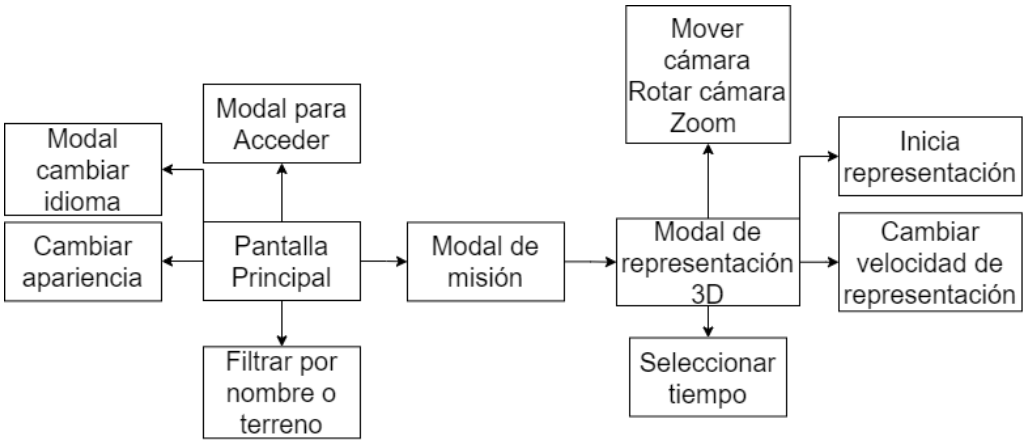

Figura 7 – Diagrama de navegación

4.2.1 Pantalla principal

La pantalla principal permite al usuario acceder a todas las herramientas (menos las de administración si no es administrador) de una manera fácil y sencilla. Como se puede observar en la Figura 8, esta pantalla consta de 6 elementos interactivos:

- **Filtro:** En la parte centro-superior se puede observar un campo de escritura, a través del que el usuario puede filtrar las misiones que se muestran en la parte inferior por el nombre de la misión o por el terreno donde se efectúa la misión.
- **Acceder como administrador:** en la parte superior derecha se encuentra el botón para abrir el modal que permite acceder como administrador. En el modal se puede escribir el usuario y la contraseña que se envían al servicio web para comprobar si los datos son correctos. Un componente *Toast* informa al usuario si el acceso ha sido correcto o el usuario o contraseña son incorrectos. En la capa de negocio se explicará más en detalle como el servicio web trabaja con estos datos.
- **Registrar usuario (administrador):** se puede registrar un nuevo administrador desde un nuevo botón que aparece al acceder como administrador. Al seleccionar el botón, se abrirá un modal para poder añadir los datos del nuevo administrador.

Figura 8 - Elementos interactivos

- **Cambiar apariencia:** El segundo botón disponible en la parte superior derecha permite cambiar entre una apariencia oscura a una clara y viceversa, como se puede observar en la Figura 9. Este componente también detecta si el usuario tiene en su sistema el tema oscuro o claro por defecto y aplica el tema dependiendo de este valor.

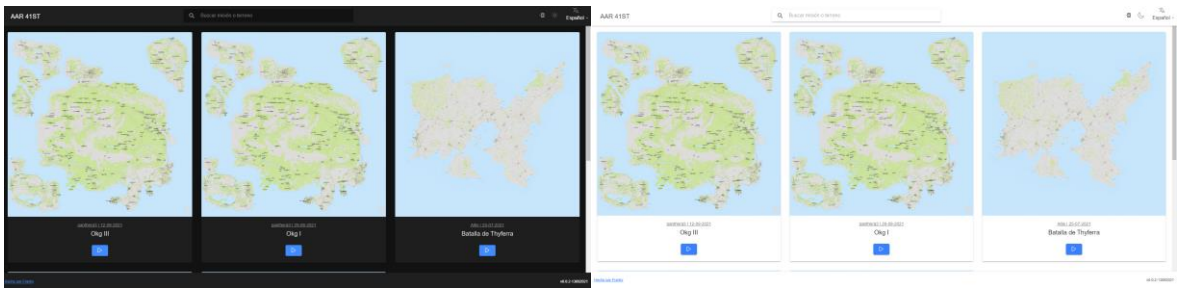

Figura 9 - Modo oscuro y modo claro

- **Cambiar el idioma:** Un tercer botón permite cambiar entre diferentes idiomas abriendo un modal que permite seleccionar un idioma entre los disponibles. Para el mantenimiento de estas traducciones se hace uso de *i18n* [15] y *localazy* [16] que permiten realizar cambios entre varios idiomas a la vez y ayuda a las traducciones de nuevas palabras o frases.

- **Seleccionar misión:** por último, el usuario puede acceder a información detallada sobre las misiones seleccionando el botón inferior de cada misión. Esto abrirá un modal donde se podrá observar los datos de la misión y desde donde se podrá abrir su representación 3D.

4.2.2 Representación 3D

La representación 3D es el componente más importante de la aplicación ya que es esencial para la correcta visualización de los datos de las partidas.

La representación 3D usa ThreeJS para representar el terreno usando OpenGL. Para ello, se obtiene el modelo 3D del repositorio Git y luego se le aplica una textura también extraída del propio repositorio. Después una librería personalizada carga el objeto 3D en formato OBJ e itera por sus vértices para formar el objeto. Una vez el objeto está formado se crea un fichero en formato MTL que su función es vincular un objeto 3D y una textura, junto con parámetros de la textura como la opacidad o el brillo.

Todo este proceso se realiza sobre un *canvas*, que no es más que el espacio dónde se va a dibujar tanto el terreno cómo diferentes objetos estáticos como el mar o el sol, para dar una mayor sensación de realismo. También se carga la cámara y los controles de esta, como se puede ver en la Figura 10, que en este caso son los controles orbitales. Una vez esté cargado todo esto, se cargan las flechas que indican la posición de los jugadores, los disparos y las flechas de los enemigos abatidos. Estos datos se actualizan cada segundo que pasa de la partida, lo que permite la actualización de la posición de estos objetos.

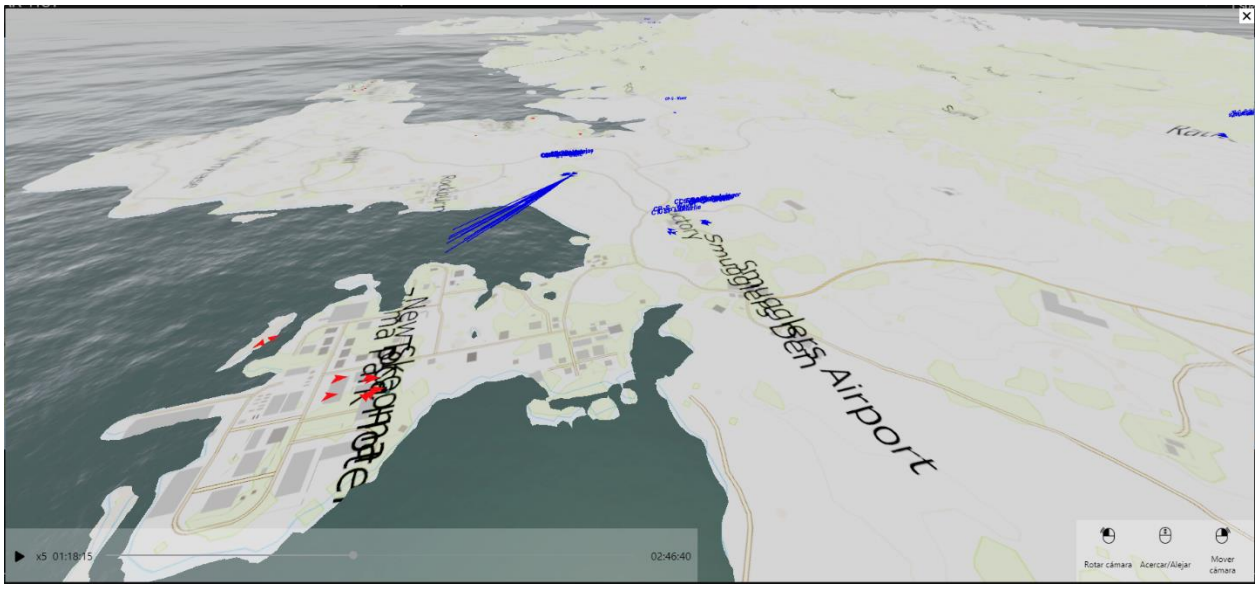

Figura 10 - Representación 3D

En la Figura 11 se muestra el fragmento de código utilizado para representar los componentes que se dibujan en el *canvas.* En primer lugar, se define el *canvas* con una serie de parámetros para configurar la cámara y usar un buffer de profundidad, que ayuda al rendimiento a costa de hacer un mayor uso de memoria. Después, dentro del *canvas*, se definen los elementos de control, en este caso a través del elemento OrbitControls, un componente que permite mover la cámara en 3D, después el componente del cielo, para dar mayor sensación de realismo, la luz ambiental para poder observar los componentes, y, por último, las flechas de las posiciones de los jugadores, de los disparos, de las posiciones de muerte y el terreno. Todo este *canvas* se inserta en un modal que contiene los controles de tiempo, de velocidad de reproducción y los controles de la cámara.

```
<Canvas
        gl={{ logarithmicDepthBuffer: true }}
        camera={{
          position: [0, this.state.size / 2, this.state.size / 2],
          fov: 75,
          far: this.state.size * 2,
          near: 100,
        }}
>
        <OrbitControls target={this.state.obj.position} screenSpacePanning />
        <Sky
          distance={450000}
          sunPosition={[0, 1, 0]}
          inclination={0}
          azimuth={0.25}
        />
        <ambientLight />
        {Array.from(this.props.arrows.values())}
        {this.props.shots}
        {Array.from(this.props.deaths.values())}
        <primitive object={this.state.obj}></primitive>
</Canvas>
```

Figura 11 - Canvas donde se dibujan todos los elementos 3D en React

4.3 Diseño e implementación de la capa de negocio

La capa de negocio es la que se encarga de implementar toda la lógica de la aplicación, tanto de sus funciones como de la aplicación de reglas de seguridad y de las interacciones con sistemas de persistencia. La capa de negocio, en este caso, está formada por dos componentes que se complementan entre sí:

- La extensión que se encarga de extraer los datos de la partida y de almacenarlos en la capa de persistencia. La extensión se ha desarrollado en C# y para la conexión con la base de datos usa la librería MongoClient.
- Un servicio (o API) web con diferentes llamadas POST que implementan las diferentes funcionalidades requeridas para la correcta comunicación entre la capa de presentación y la persistencia. El servicio web se ha creado usando Javascript con la librería *Express* y utiliza un sistema de seguridad basado en *hashing* usando *bcrypt* junto con un *salt* autogenerado y un sistema de *tokens* para dar acceso al usuario.

  Para acceder a la base de datos usa mongoose, una librería que permite conectarse a una base de datos MongoDB y mediante esquemas definir los objetos de una colección. MongoDB almacena documentos, que son las líneas de datos de la base de datos, en colecciones, que son las tablas donde los datos se almacenan. Los pasos para extraer datos de la base de datos usando esta librería son los siguientes:

  1. Conectarse a la base de datos usando su URL.
  2. Una vez conectado crear los modelos que constituyen los esquemas del tipo de datos que vas a extraer y la colección que se va a usar.
  3. Buscar los datos obteniendo los documentos de la colección.
  4. Finalmente, con los datos obtenidos, trabajar con ellos y almacenar (si es necesario) en una variable para su futuro envío en el cuerpo de la respuesta.

En las siguientes subsecciones se explican algunos detalles del diseño e implementación de estos dos componentes.

4.3.1 Diseño e implementación de la API web

La Tabla 3 muestra el diseño de la API del servicio web con las diferentes funciones proporcionadas, sus parámetros, enviados en el cuerpo de la petición, y los posibles resultados a obtener, en forma de códigos HTTP y datos retornados. A continuación, se detallan todas las funciones desarrolladas.

Tabla 3 – Diseño de la API

DIRECCIÓN	PARÁMETROS	RESULTADO
/aar/login	username: string password: string	200: string (JWT token) 403 (Forbidden) 500 (Internal Server Error)
/aar/toggleHidden	token: string mission: string hidden: boolean	200: void 401 (Unathorized) 403 500
/aar/register	token: string newUsername: string newPassword: string	200: void 401 403 500
/aar/getAllMissions		200: Mission[] 500
/aar/getMissionData	mission: string	200: Unit[] 500

login

La función login se encarga de obtener el nombre de usuario y la contraseña del cuerpo de la petición y posteriormente encriptar la contraseña usando un *salt* autogenerado por *bcrypt*. Esta información se comprueba con la base de datos MongoDB usando *mongoose* junto con un *schema* de la estructura de los usuarios, que en este caso es el nombre de usuario y la contraseña del usuario. Si el usuario no existe se retorna un estado 403 sin datos en base al cual la capa de presentación debe informar de que no se ha encontrado el usuario. Si existe retorna un estado 200 con un *token* generado usando JWT a través de la función *createToken* cuyo código se muestra en la Figura 12.

```
1.  function createToken(username, password) {
2.      return jwt.encode(
3.        {
4.          username: username,
5.          password: password,
6.          iat: moment().unix(),
7.          exp: moment().add(14, 'days').unix(),
8.        },
9.        process.env.TOKEN_SECRET,
10.     );
11.   }
```

Figura 12 - Función para generar el token JWT

Este se codifica usando el nombre de usuario, la contraseña encriptada, el momento en el que se creó el *token* y su fecha de expiración, que en este sistema es de 14 días. Todo esto se encripta con un *token* secreto del sistema que se autogenera cada vez que se inicia la API, por lo que si se reinicia la API todos los tokens quedan invalidados y han de ser recreados.

A partir de este momento, el sistema de tokens se ocupa de comprobar las credenciales del administrador antes de poder realizar cualquier acción de agregación o modificación en la base de datos. Para ello, se comprueba que el usuario que realiza la acción esté usando un *token* correcto, el cual es suministrado a través de la cabecera *authorization* del mensaje HTTP. Por cada invocación recibida, se realiza la decodificación del token suministrado y se comprueba si el *token* ha expirado. Si el *token* no ha expirado, se devuelven el usuario y la contraseña del administrador para buscarlo en la base de datos. El código necesario para implementar este proceso de autenticación se muestra en la Figura 13.

```
1.  function ensureAuthenticated(req, res) {
2.    if (!req.headers.authorization) {
3.      return res.status(403).send({
4.        message: 'Your request does not have an authorization header',
5.      });
6.    }
7.
8.    var token = req.headers.authorization.split(' ')[1];
9.    var payload = jwt.decode(token, process.env.TOKEN_SECRET);
10.
11.   if (payload.exp <= moment().unix()) {
12.     return res
13.       .status(401)
14.       .send({ message: 'The token has expired' });
15.   }
16.
17.   return { username: payload.username, password: payload.password };
18. }
```

Figura 13 - Función para comprobar que el usuario es administrador

toggleHidden

Esta función tiene como finalidad ocultar o no misiones modificando el valor *hidden* en las propiedades de una misión. Se retorna el estado 200 si se ha modificado correctamente la propiedad, 403 si no se ha podido identificar al administrador o si el mensaje no contenía una autentificación y 401 si el token usado ha expirado.

register

Esta función se ocupa de registrar un nuevo administrador. Una vez que se ha comprobado el token, se crea el nuevo usuario usando el nombre de usuario y contraseña extraídos del cuerpo de la petición. La contraseña se almacena como un *hash* más *salt*. Se retorna el estado 200 si se ha añadido correctamente el usuario, un 403 si no se ha podido identificar al administrador o si el mensaje no contenía una autentificación y un 401 si el token usado ha expirado.

getAllMissions

Esta función obtiene los metadatos de todas las misiones disponibles usando un esquema que permite la extracción de los datos de la base de datos y se crean los objetos usando los documentos. Los documentos son las líneas de la base de datos donde se almacenan los datos dentro de una colección.

getMissionData

Esta función obtiene los datos completos de una misión como las posiciones de los jugadores y enemigos, los disparos o las posiciones de los jugadores o enemigos abatidos.

4.3.2 Extensión de obtención de datos

Para extraer los datos de una partida se ha desarrollado una librería dinámica (dll) que se ejecuta en el servidor del videojuego y que se encarga de enviar los datos a la base de datos de MongoDB.

Esta librería dinámica está desarrollada en C# y hace uso de un conjunto de interfaces que proporciona la aplicación del servidor para permitir la comunicación con posibles extensiones como esta. El servidor del videojuego llama al método `RvExtensionArgs` implementado por la extensión mediante el uso de la función `callExtension`, como se puede observar en la Figura 14 [10]. Este método recibe como parámetros la función que se quiere invocar en la extensión, `start` en este ejemplo, y un array con parámetros.

```
"aar" callExtension ["start", [missionname, worldName, briefingname, worldSize]];
st41_aar_mission_running = true;
st41_aar_mission_running_time = servertime;
addMissionEventHandler ["MPEnded", {
    st41_aar_mission_running = false;
    st41_aar_mission_end_time = servertime;
    "aar" callExtension ["end", [missionname, round(st41_aar_mission_end_time - st41_aar_mission_running_time)]];
}];
```

Figura 14 - Ejemplo de llamadas a la extensión

Dependiendo de la función que se pase por parámetro, en el parámetro `function` del método `RVExtensionArgs`, como se muestra en la Figura 15, se llamará a una función implementada por la extensión, como, por ejemplo, en este caso, la función `StartData`. Esta función se encarga de inicializar el cliente de Mongo, conectándose a la base de datos de "aar", creando una nueva colección con el nombre de la misión (que se ha pasado como parámetro) y añadiendo los metadatos de la misión a la base de datos, a través del código de la Figura 16.

```
public static int RvExtensionArgs(StringBuilder output, int outputSize,
string function,string[] args, int argCount)
{
    int status = 0;
    try
    {
        outputSize--;
        switch (function)
        {
            case "start":
                Task.Run(() => {
                    StartData(args[0].Replace("\"", ""), ...);
                });
                break;
```

Figura 15 – Método RVExtensionArgs

```
169  public static void StartData(string mission, string terrain, string briefingName, string worldSize)
170          {
171              dbClient = new MongoClient("mongodb://41st:***@***:27017...");
172              db = dbClient.GetDatabase("aar");
173              db.CreateCollection(mission);
174              dataCollection = db.GetCollection<BsonDocument>(mission);
175              var indexPlayerName = Builders<BsonDocument>.IndexKeys.Ascending("name");
176              dataCollection.Indexes.CreateOneAsync(new CreateIndexModel<BsonDocument>(indexPlayerName));
177              missionCollection = db.GetCollection<BsonDocument>("missions");
178              var document = new BsonDocument { { "mission", mission},... };
179              missionCollection.InsertOne(document);
180          }
```

Figura 16 - Función que inicializa el sistema de MongoDB

4.4 Diseño e implementación de la capa de persistencia

La capa de persistencia es la capa que almacena la información obtenida a través de la extensión del servidor. En el caso de esta aplicación, se almacenan los datos de cada misión (el terreno en el que se juega, el nombre de la misión, el tamaño del terreno, la fecha y el tiempo de duración de la misión), junto con su estado *hidden*, que indica si es visible en la capa de presentación o no. También se almacenan los usuarios que son administradores. Por último, por cada misión se almacena cada unidad que ha participado en ella, el nombre de la unidad, el bando, las posiciones durante la partida, los disparos y por último la posición de la muerte si se ha producido. Por cada evento, de posición, de disparo o de muerte, se almacena el momento en el que se ha realizado (el *timestamp*), la posición en formato XYZ siendo la Z la altura y la dirección.

La Figura 17 muestra un pequeño diagrama de clases que resume los tipos de datos que se manejan en la API.

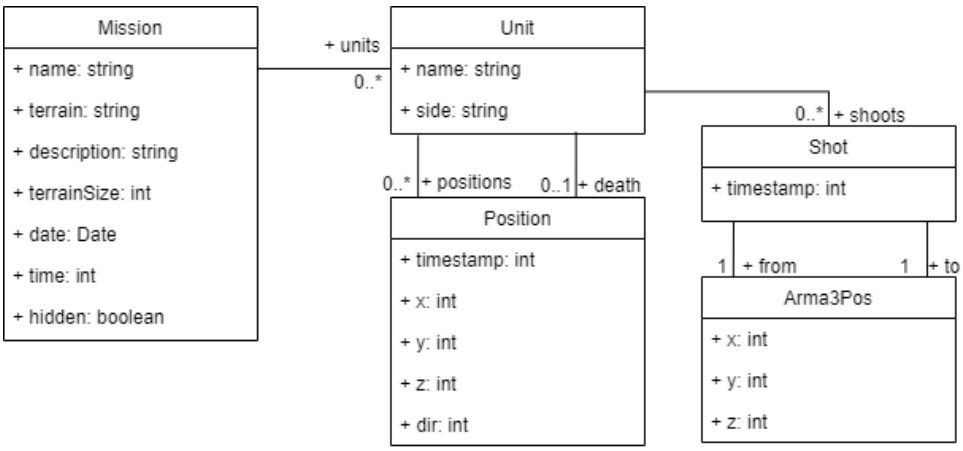

Figura 17 - Diagrama de clases

5. Despliegue

Para el despliegue del código y de los servicios utilizados en esta aplicación se han utilizado diferentes máquinas virtuales que permiten la fácil distribución de recursos en un solo sistema físico y una mayor seguridad en caso de fallo del sistema.

Para almacenar estas máquinas virtuales se han usado contenedores que son paquetes de software más ligeros, que nos permiten tener un ecosistema más estable y así evitar el bloatware, es decir, programas o servicios que no se van a utilizar y que consumen recursos.

Todos estos contenedores están controlados mediante Proxmox VE, que es un entorno de virtualización de servidores [12]. Este entorno es de código abierto, desarrollado en Debian con un kernel modificado de RedHat y permite la creación de contenedores y máquinas virtuales, en los cuales se puede desplegar servicios como MongoDB o Nginx. En la Figura 18 se puede observar el panel de control que permite manejar las diferentes máquinas virtuales desplegadas y los recursos usados en ellas. También permite crear clusters entre nodos para una mayor facilidad y control entre diferentes servidores físicos.

Type ↑	Description	Disk usage...	Memory us...	CPU usage	Uptime	Host CPU ...	Host Mem...
lxc	103 (node)	14.9 %	36.2 %	0.1% of 1 ...	5 days 00:56...	0.0% of 16...	1.2 %
lxc	105 (nginx)	13.2 %	8.7 %	0.1% of 1 ...	15 days 18:5...	0.0% of 16...	0.1 %
lxc	107 (gitea)	17.8 %	1.2 %	0.1% of 2 ...	15 days 18:5...	0.0% of 16...	0.3 %
lxc	108 (mongodb)	28.4 %	15.5 %	0.3% of 1 ...	15 days 18:5...	0.0% of 16...	0.2 %

Figura 18 - Panel de control de Proxmox

En la Figura 19 se muestra el diseño físico realizado para permitir la escalabilidad de la aplicación y mantener su correcto funcionamiento ante fallos en los nodos. Como se puede observar, los contenedores están duplicados para realizar el balanceo de carga y solo se activan los contenedores secundarios en caso de alta carga o fallo en el contenedor principal.

Los contenedores que se usan en el despliegue de esta aplicación son:

- El contenedor “node”, en el que se despliega la aplicación de NodeJS que constituye el código de la página web. El desarrollo del código se ha realizado mediante conexión SSH al contenedor “node” ya que es en el dónde se almacena la aplicación. Automáticamente al construir la aplicación una vez terminada una nueva funcionalidad o arreglado un error, se reinicia el servicio para actualizar los datos que han sido reconstruidos. Al reconstruirse, por defecto se despliega la versión de *testing* para poder ser probada por los *testers*. Una vez que esta versión ha sido comprobada, se despliega la versión de producción.
- El contenedor “nginx”, en el que se despliega el servicio de Nginx que nos servirá como punto de entrada y distribución de los datos entre aplicaciones del sistema. Se ha usado el patrón reverse proxy para facilitar la publicación de nuevas aplicaciones o subdominios sin exponer el sistema completo o sin necesidad de configurar la apertura de puertos en

el router. También Nginx permite el uso de certificados SSL para mejorar la seguridad de las transacciones entre la aplicación y el servidor. El contenedor nginx juega el rol de balanceador de carga que permite redirigir la carga de entrada entre los restantes pares de contenedores para evitar la sobrecarga del sistema y su mal funcionamiento gracias a que existe una copia exacta de los contenedores que sirve salvaguarda en caso de fallo.

- El contenedor “gitea” contiene el servicio de Gitea, que es el servidor Git que se usa para controlar las versiones de los modelos 3D y de las texturas. Este repositorio Git nos permite actualizar y añadir objetos 3D y texturas de una manera rápida y sin necesidad de reiniciar ningún servicio.
- Por último, se despliega el servicio de MongoDB en el contenedor “mongodb” que nos permitirá almacenar la información de las partidas. El peso estimado de cada misión es de 30 MB, dependiendo del número de unidades y del tiempo de actualización, que por defecto es de cinco segundos. Si este valor disminuye hasta un segundo el peso estimado es de 150 MB, por lo que no se recomienda aumentar la velocidad de actualización, ya que podría afectar al normal funcionamiento del servidor. Como respaldo se usan nodos de MongoDB, almacenando las misiones de mayor peso en el servidor principal ya que los nodos de MongoDB solo pueden almacenar 512 MB de forma gratuita. Para acceder a estos datos se ha usado MongoDBCompass ya que en el proceso de pruebas algunos datos adjuntos no eran correctos y afectaban a la capa de negocio. Estos datos incorrectos eran eliminados para evitar la inconsistencia en los datos de la base de datos.

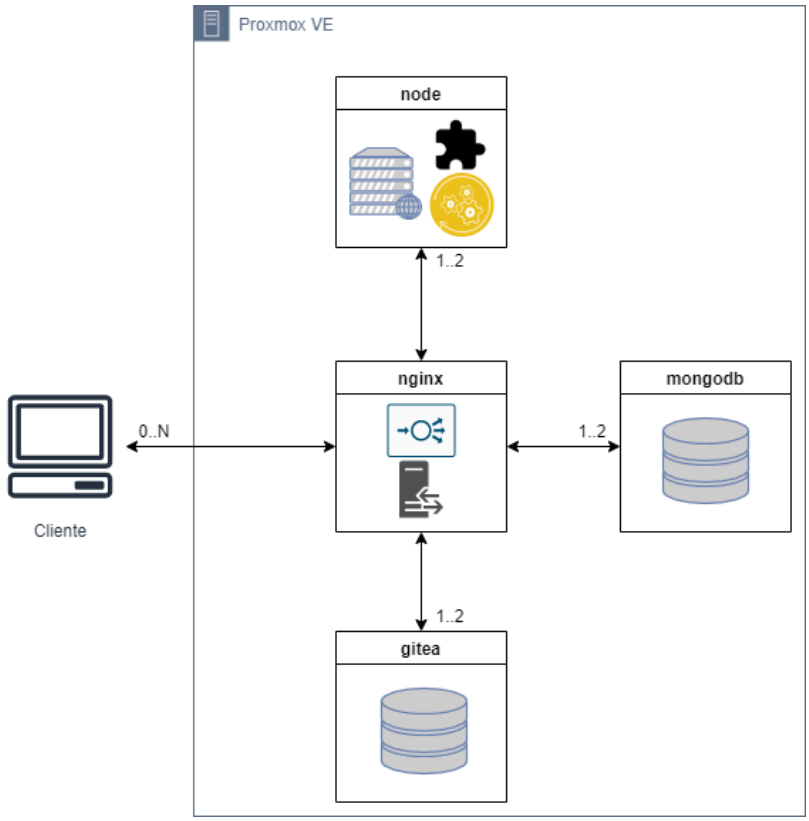

Figura 19 - Diseño físico

6. Evaluación y pruebas

En este capítulo se explican las diferentes pruebas realizadas en el sistema para comprobar que los componentes funcionan correctamente y que se han completado los requisitos definidos. Para ello se han realizado pruebas unitarias para comprobar el correcto funcionamiento de los componentes individuales, pruebas de integración, para comprobar el correcto funcionamiento entre componentes y pruebas de sistema para verificar que los requisitos no funcionales han sido completados.

6.1 Pruebas unitarias

Las pruebas unitarias son un conjunto de técnicas que permiten comprobar el correcto funcionamiento de una unidad aislada de código. En el caso de esta aplicación, esta unidad de código puede ser un componente React, al ser un diseño orientado a objetos, o puede ser una función en el caso de la API que tiene un diseño funcional. En este caso, se han aplicado pruebas unitarias a todos los componentes individuales de React, como por ejemplo Mission, que se encarga de mostrar las misiones.

Todas estas pruebas realizadas en React han sido programadas con Jest, que es uno de los *frameworks* más utilizados para realizar pruebas en Javascript, React o Typescript [13]. En la Figura 21 se muestra un fragmento del código desarrollado para llevar a cabo estas pruebas.

```
6  describe('Mission component', () => {
7
8    it('check snapshot', () => {
9      const mission = render(<Mission date='01-01-1900' ...);
10     expect(mission).toMatchSnapshot();
11   });
12
13   it('check mission metadata', () => {
14     render(<Mission date='01-01-1900' ...);
15     expect(screen.getByText('test_terrain | test_sizem²')).toBeInTheDocument();
16     expect(screen.getByText('description')).toBeInTheDocument();
17   })
18
19   it('check hidden', () => {
20     render(<Mission date='01-01-1900' description='description' hidden={true} ... />);
21     expect(screen.queryByText('test_terrain | test_sizem²')).toBeNull();
22     expect(screen.queryByText('description')).toBeNull();
23   })
24 });
```

Figura 20 - Ejemplo de prueba unitaria de componente React

6.2 Pruebas de integración

Las pruebas de integración comprueban que la interacción entre dos o más componentes es correcta. Al ser una aplicación SPA (Single-Page Application) los test de integración comprueban principalmente las interacciones que abren modales y comprueban si se muestran los datos correctos en pantalla, lo cual significa que el funcionamiento de los botones y de la API es correcto. Estas pruebas se han realizado usando Jest, ya que también permite interaccionar con botones que hayan sido renderizados, como se puede observar en la Figura 22.

```
26  it('check open modal on button click', () => {
27    render(<Mission date='01-01-1900' description='description' hidden={false} isLoggedIn={true} ... />);
28    expect(screen.getByText('test_terrain | test_sizem²')).toBeInTheDocument();
29    expect(screen.getByText('description')).toBeInTheDocument();
30    expect(screen.queryByText('test')).toBeNull();
31    const button = screen.getByRole('button');
32    button.click();
33    expect(screen.getByText('test_terrain | test_sizem²')).toBeInTheDocument();
34    expect(screen.getByText('description')).toBeInTheDocument();
35    expect(screen.getByText('test')).toBeInTheDocument(); //This check if the modal is open
36  })
```

Figura 21 - Ejemplo de prueba de integración

6.3 Pruebas de sistema

Para realizar las pruebas de sistema se han usado principalmente dos aplicaciones.

La primera aplicación es la aplicación de rendimiento de Chrome, que nos informa de cualquier problema de rendimiento como fugas de información, un uso excesivo de la tarjeta gráfica o un uso excesivo de memoria. En la Figura 23 se puede observar que el uso de memoria de la aplicación no supera 1GB, en este caso el máximo que alcanza es de 29MB como se puede observar en la parte inferior izquierda de la imagen, por lo que el requisito no funcional de que la aplicación debe usar menos de 1GB de memoria se puede considerar verificado.

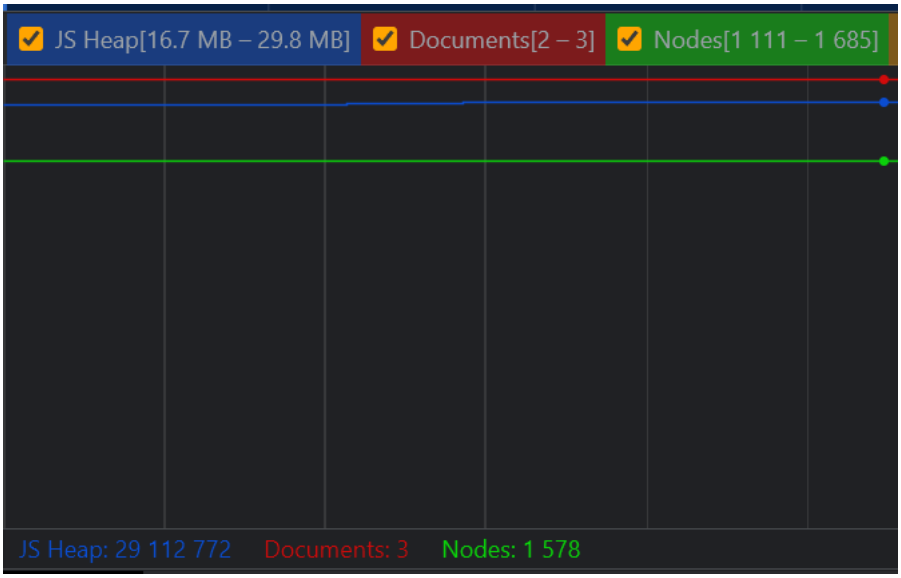

Figura 22 - Gráfica de uso de memoria

Para comprobar el uso de GPU es necesario usar las herramientas del sistema operativo. Tras comprobar el uso de GPU en el administrador de tareas de Windows, el uso máximo fue de un 7%, con una media cercana al 1%, por lo que el requisito no funcional de usar menos de un 75% de GPU está también verificado.

La otra aplicación usada es *Lighthouse*, otra aplicación de Chrome que permite visualizar problemas tanto de rendimiento, como de accesibilidad o SEO (Search Engine Optimization) desde diferentes dispositivos, con diferentes resoluciones de pantalla. Esta aplicación también permite visualizar el tiempo de carga y optimizar la carga de ficheros para evitar tiempos de carga muy amplios. Como se puede observar en la Figura 24 el tiempo de carga de la aplicación es de 1.7

segundos, por lo que el requisito no funcional que indicaba que no se podía superar los 5 segundos de tiempo de carga ha sido completado. Por último, se ha comprobado que la aplicación funcione correctamente en exploradores web como Firefox y Edge.

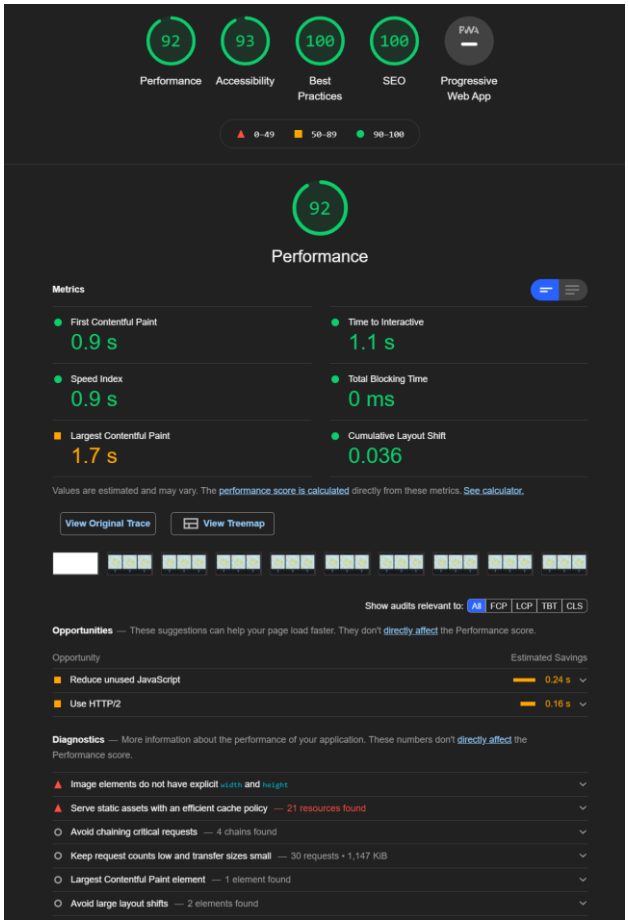

Figura 23 - Informe de Lighthouse

7. Conclusiones y trabajos futuros

En este capítulo se recaban las conclusiones alcanzadas a la finalización del proyecto y los posibles trabajos futuros que se pueden realizar sobre la aplicación.

7.1 Conclusiones

Como he explicado anteriormente, el objetivo principal de esta aplicación era realizar una representación visual de partidas y poder analizarlas *a posteriori*. Después de analizar el resultado, se considera que se han completado con éxito todos los requisitos iniciales, que se ha creado una aplicación inexistente en el mercado y que puede ser una base para una gran aplicación que puede marcar la diferencia entre usuarios que usen esta aplicación.

El proyecto ha sido complejo, más de lo que se podría pensar, ya que contiene muchos componentes que tienen que ir interconectados y se ha de tratar de evitar problemas en todos ellos, ya que, si uno cae, se puede producir una caída en cascada. Además, los conocimientos sobre Three.js eran escasos al iniciar el proyecto y fue la representación 3D de los datos lo que más trabajo conllevó, ya que fueron muchos los problemas encontrados. Gracias a los conocimientos adquiridos durante el grado ha sido mucho más fácil encontrar y superar las adversidades en el desarrollo de este proyecto.

7.2 Trabajos futuros

Esta aplicación puede aumentar enormemente en tamaño y complejidad ya que, al tener una base ya creada como la API o la representación 3D, las posibilidades de desarrollar nuevos componentes y expandir la aplicación serán mucho mayores que si no hubiera una base ya desarrollada.

Algo que aún se puede mejorar podría ser el entorno de desarrollo, ya que al haber sido desarrollado de forma individual no se ha tenido en cuenta elementos muy importantes como la integración continua o usar un sistema de *pull request* en Gitea para revisar código y hacer un proyecto más completo. También en el componente 3D sería deseable trabajar para juntar los nombres de los jugadores y así evitar que se junten todos en un mismo punto y no se pueda observar con facilidad.

Por último, una línea de trabajo futuro interesante sería crear una PWA (Progresive Web Application) para que pueda usarse como una aplicación de escritorio, más que como una página web.

8. Referencias

[1] Army, U. S. (1993), A leader's guide to after-action reviews, https://www.hsdl.org/?view&did=775082, Ultimo acceso: 14-02-2022

[2] Blender Foundation, Obtenido de https://www.blender.org/, Ultimo acceso: 15-02-2022

[3] OpenJS Foundation, Obtenido de https://nodejs.org/en/about/, Ultimo acceso: 15-02-2022

[4] Facebook Open Source, Obtenido de https://es.reactjs.org/, Ultimo acceso: 15-02-2022

[5] Microsoft, Obtenido de https://www.typescriptlang.org/, Ultimo acceso: 15-02-2022

[6] ThreeJS, Obtenido de https://threejs.org/manual/#en/fundamentals, Ultimo acceso: 15-02-2022

[7] Ionic, Obtenido de https://ionicframework.com/, Ultimo acceso: 15-02-2022

[8] Blender Foundation, Import-Export/XYZ, Obtenido de https://wiki.blender.jp/Extensions:2.6/Py/Scripts/Import-Export/XYZ, Ultimo acceso: 15-02-2022

[9] Sara López Mora, (2021), ¿Qué son las Single-page application (SPA)? El desarrollo elegido por Gmail y LinkedIn, Obtenido de https://www.digital55.com/desarrollo-tecnologia/que-son-single-page-application-spa-desarrollo-elegido-por-gmail-linkedin/, Ultimo acceso: 16-02-2022

[10] Bohemia Interactive, SQF Syntax, Obtenido de https://community.bistudio.com/wiki/SQF_Syntax , Ultimo acceso: 16-02-2022

[11] Bohemia Interactive, Extensions, Obtenido de https://community.bistudio.com/wiki/Extensions, Ultimo acceso: 16-02-2022

[12] Proxmox Server Solutions GmbH, Proxmox Virtual Environment, Obtenido de https://www.proxmox.com/en/proxmox-ve, Ultimo acceso: 16-02-2022

[13] Facebook Open Source, Obtenido de https://jestjs.io/es-ES/, Ultimo acceso: 15-02-2022

[14] MongoDB, Obtenido de https://www.mongodb.com/es, Ultimo acceso: 15-02-2022

[15] Mozilla, Obtenido de https://developer.mozilla.org/es/docs/Mozilla/Add-ons/WebExtensions/API/i18n, Ultimo acceso: 15-02-2022

[16] Localazy, Obtenido de https://localazy.com/, Ultimo acceso: 15-02-2022