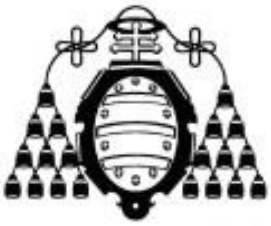
UNIVERSIDAD DE OVIEDO



ESCUELA DE INGENIERÍA INFORMÁTICA

TRABAJO FIN DE GRADO

Portal web para mejorar en el juego *Dots and Boxes* de forma autónoma y competir contra otros jugadores

DIRECTOR: Cristian González García

AUTOR: Víctor Gonzalo Cristóbal

Resumen

El objetivo de este proyecto es desarrollar una aplicación de navegador que permita a los usuarios aprender a jugar al juego *Dots and Boxes* (o seguir mejorando sus habilidades), además de competir contra otros jugadores, estableciendo un *ranking* de puntuaciones. Para poder identificar a cada usuario, estos deberán registrarse en el portal *web*.

La inteligencia artificial ofrece un gran potencial para que los ordenadores tomen decisiones, en cierta medida, similares a las de los humanos. Sin embargo, uno de los principales problemas que presenta es que, normalmente, resulta muy difícil entender o explicar el porqué de las decisiones que toma un sistema de este tipo. Teniendo esto en mente, y con la finalidad de acelerar el proceso de aprendizaje, este proyecto intentará desarrollar una herramienta que transmita de forma efectiva el conocimiento generado por un sistema de inteligencia artificial a los usuarios. La aplicación *web* mostrará a los jugadores la información producida de forma que sea fácilmente asimilable por estos, ofreciéndoles distintas formas de acceder a ella: un árbol interactivo en el que aparece el estado actual del tablero y todos los movimientos posibles (representados como nodos hijos del estado actual), junto con información para cada movimiento; un conjunto de consultas que permiten obtener distintos caminos, y una sección donde aparecen el movimiento recomendado y aquellos movimientos que se deberían evitar, acompañados de una breve explicación textual.

Para generar ese conocimiento, utilizamos el sistema propuesto por Alba Cotarelo *et al.* en un trabajo previo, basado en una combinación de un algoritmo de árbol de búsqueda de Monte Carlo y redes neuronales. A partir de los datos producidos por este algoritmo, se confeccionará toda la información que será presentada al usuario en la aplicación *web*. Además, las capacidades del algoritmo permitirán que, en combinación con un servidor que implementará toda la lógica necesaria, pueda actuar como rival virtual contra el que los jugadores podrán entrenar.

El desarrollo del cliente *web* se ha llevado a cabo utilizando el *framework* Angular. El servidor y la lógica están implementados en Java. La comunicación entre cliente y servidor se realiza mediante una API HTTP para la autenticación de usuarios, y mediante una conexión WebSocket para el envío y recepción de datos. Esta última permitirá que se desarrollen los distintos tipos de partidas (ya sean contra el rival virtual, o contra otro jugador). Para el almacenaje de datos se utiliza un *cluster* MongoDB con el que únicamente se comunicará el servidor.

Palabras Clave

Inteligencia artificial, autoaprendizaje, Dots and Boxes, juegos de mesa, IA explicable, WebSockets, Angular

Abstract

The aim of this project is to develop a browser application which permits users to both learn to play the Dots and Boxes game and continue to improve their skills. Besides that, it will allow them competing against others, stablishing a player ranking. In order to identify each of those users, they must create an account in the web application.

Artificial intelligence offers great potential for allowing computers to make decisions in a similar way as humans do. However, it presents a relevant drawback: commonly, it is a hard task to explain or identify the reasons behind the decisions made by a system of this kind. Having this in mind, and with the objective of accelerating the learning process of users, this project will develop a tool trying to effectively transmit to humans the expert knowledge generated by an intelligent system. The web application will present the information to the players in an easily interpretable way, offering them different ways of accessing it. They will be able to reach the information in three main ways: an interactive tree showing the current state of the game and all the possibilities from it (each of them represented as a child node), along with some information for each node; a set of queries which allow obtaining different kinds of paths, and a section where users are recommended a particular movement and are advised to avoid some others. Each of those pieces of advice appears accompanied by a brief textual explanation.

For generating all that knowledge, we are employing the system proposed by Alba Cotarelo *et al.*, based in a combination of a Monte Carlo Tree Search algorithm with artificial neural networks. All the information that will be presented to the user in the web application will be extracted from the data produced by this algorithm. In addition, its capacities will be used by a server implementing all the necessary logic to perform as a virtual rival against which the users will train.

The web development framework Angular has been used to develop the web client for this application. The server and the logic, meanwhile, are implemented in Java. Clients and server communicate through a HTTP web API for user authentication and employing the WebSockets protocol for transferring data related with playing a game. Data will be stored in a MongoDB cluster with whom only the server communicates.

Keywords

Artificial intelligence, self-learning, Dots and Boxes, board games, explainable AI, WebSockets, Angular

Índice General

Índice de Figuras

Capítulo 1. Memoria del Proyecto

En este primer capítulo se resumirá el proyecto y sus objetivos prescindiendo de términos técnicos, de manera que las personas sin conocimientos específicos en el área puedan comprenderlo fácilmente.

1.1 Resumen de la Motivación, Objetivos y Alcance del Proyecto

Es innegable la gran relevancia que han tomado la inteligencia artificial y el aprendizaje automático en las vidas de todos nosotros: desde las aplicaciones de traducción, o los coches con capacidades de conducción autónoma, hasta la sistematización de procesos industriales.

Sin embargo, el saber que estos sistemas inteligentes atesoran, raras veces es utilizado para enseñar a humanos sobre un área de conocimiento específica. El concepto de trasladar conocimiento desde un sistema de este tipo hacia un individuo es interesante y puede resultar de utilidad, por ejemplo, para un caso en que sea necesario enseñar o mejorar las habilidades de una persona que debe tomar decisiones a lo largo de un proceso industrial. Ya que es posible que para ese proceso no sea viable la aplicación directa del sistema inteligente debido a restricciones de diversas naturalezas.

Precisamente, con la idea de demostrar la posibilidad de que exista una transferencia efectiva de conocimiento desde una inteligencia artificial hacia un usuario humano, nos propusimos la creación de esta herramienta.

La aplicación presentada pretende presentar a sus usuarios de forma amigable y fácilmente asimilable la información generada por el algoritmo inteligente desarrollado por Alba Cotarelo et al. **[1]**, que, utilizando un árbol de búsqueda de Monte Carlo en combinación con redes neuronales artificiales, es capaz de evaluar la idoneidad de todos los movimientos posibles a partir de un estado de tablero del juego *Dots and Boxes*. Aunque se explicará en profundidad más adelante, este juego consiste en una matriz de puntos entre los cuales dos (o más) jugadores trazarán distintas líneas con el fin de conquistar el mayor número de cajas posible. Para conseguir este objetivo, nuestra herramienta ofrecerá un movimiento recomendado para cada momento de la partida, junto con una breve explicación textual y un conjunto de datos que pretenden reforzar la confianza que el usuario tiene en la recomendación, facilitando el proceso de aprendizaje. De forma análoga, también se ofrecerá una explicación sobre cuáles son los movimientos a evitar (porque sean más beneficiosos para el rival).

Los usuarios de la aplicación *web* podrán recibir las recomendaciones del sistema inteligente mientras jueguen partidas en las que ese mismo sistema inteligente también actúa como rival virtual. Además de eso, podrán probar sus habilidades jugando contra el sistema sin obtener recomendaciones, y batiéndose contra otros jugadores en partidas normales o clasificatorias.

1.2 Resumen de Todos los Aspectos

Este documento está dividido en 13 capítulos distintos, cada uno de ellos trata los temas siguientes.

Capítulo 1. Resumen del proyecto, incluyendo motivación, objetivos, alcance y los contenidos de la documentación.

Capítulo 2. Justificación y objetivos del proyecto más en profundidad, además de una evaluación de la situación actual y un estudio de alternativas.

Capítulo 3. Breve descripción de los conceptos, herramientas y tecnologías que resultan especialmente relevantes para el proyecto.

Capítulo 4. Explicación de cómo se han gestionado las tareas a realizar dentro del proyecto.

Capítulo 5. Análisis del sistema a desarrollar. Aquí se incluye una definición del mismo, para establecer sus límites, y diversos análisis de requisitos, casos de uso, subsistemas, clases e interfaces.

Capítulo 6. Diseño del sistema, evolucionado a partir del análisis previo. Aquí se explicará la arquitectura del sistema, sus clases, la base de datos que usará, las interfaces de usuario y se definirá el plan de pruebas.

Capítulo 7. Trata la implementación del sistema. Describe los estándares, lenguajes de programación y herramientas empleados en el desarrollo. También explica los aspectos relevantes surgidos a la hora de crear el código.

Capítulo 8. Desarrollo, resultados y conclusiones de las distintas pruebas previamente definidas.

Capítulo 9. Manuales de instalación, ejecución, usuario y programación del sistema presentado.

Capítulo 10. Conclusiones del desarrollo y posibles ampliaciones posteriores.

Capítulo 11. Resumen de la planificación final seguida a lo largo del proyecto.

Capítulo 12. Referencias bibliográficas: artículos, libros y sitios web consultados para el desarrollo del proyecto o su documentación.

Capítulo 13. Apéndices. Este capítulo es, principalmente, una breve explicación del material entregado junto a este documento.

Capítulo 2. Introducción

En este capítulo se explicará el origen del proyecto y de dónde surge; se listarán los objetivos que se quieren lograr, y se expondrá el estudio realizado de la situación actual.

2.1 Justificación del Proyecto

La inteligencia artificial y el aprendizaje automático han permitido, en las últimas décadas, aplicar las computadoras a la resolución de problemas o a procesos que, hasta hace no mucho, solo podían ser abordados por humanos, o ni siquiera se habían planteado de forma seria, al verse como algo casi imposible.

Ya sea proveniente de conjuntos de datos generados por humanos, o de información generada por los mismos sistemas inteligentes, estos programas guardan, en muchas ocasiones, un conocimiento excelso en ciertas áreas de conocimiento. Este proyecto nace, precisamente, para explorar la idea de trasladar este conocimiento desde las inteligencias artificiales hasta individuos humanos.

Esta idea de transmitir conocimiento en el sentido inverso al que solemos ver surge, en parte, de ver cómo los mejores ajedrecistas del mundo aprenden de inteligencias artificiales como AlphaZero **[2]**, que son capaces de desarrollar por sí solas (jugando contra ellas mismas) estrategias ganadoras basadas, incluso, en movimientos que clásicamente habían estado prácticamente prohibidos, por ser considerados pésimos.

Como base para nuestra aplicación, utilizaremos el algoritmo inteligente ya mencionado en los apartados previos, aplicándolo al juego *Dots and Boxes* como caso de estudio. Para facilitar el aprendizaje de sus usuarios, la aplicación permitirá acceder a la información y las recomendaciones obtenidas empleando este sistema inteligente mientras juegan contra el propio sistema, actuando como rival artificial. Además de esto, permitirá medirse contra la inteligencia artificial, sin la ayuda de sus recomendaciones, además de contra otros jugadores en partidas normales y clasificatorias.

El hecho de propiciar esta adquisición de conocimiento mientras se juega, ofrecerá resultados mejores de los que podría arrojar la simple exposición de los datos generados por la inteligencia artificial a los usuarios. Este método permitirá estudiar la información mientras se juega, haciendo el proceso de aprendizaje más ameno. Además, estimulará el afán competitivo de algunos jugadores.

2.2 Objetivos del Proyecto

El objetivo de este proyecto es desarrollar desde cero una aplicación *web*, tanto la parte de cliente como la de servidor) que permita explorar la idea de transmisión de conocimiento ya explicada en los apartados previos, además de facilitar a sus usuarios la evaluación de estos conocimientos y habilidades compitiendo contra otros jugadores. Para ello, la aplicación implementará las siguientes funcionalidades mínimas:

1. Permitir que los usuarios de la aplicación se registren e identifiquen para empezar a jugar.
2. Ofrecer la posibilidad de jugar partidas de entrenamiento contra la inteligencia artificial, la cual tendrá una dificultad configurable por el usuario.
3. Exponer a los jugadores, si lo desean, y de forma fácilmente asimilable, la información extraída del algoritmo inteligente para cada posible movimiento que puedan realizar. Esta información se expondrá, principalmente, de tres formas distintas:
    a. Un árbol interactivo que permitirá explorar todos los posibles movimientos (representados como nodos), junto con una serie de datos, y con conjunto de los posibles estados de la partida a partir de cada movimiento.
    b. Una serie de consultas que permitirán buscar caminos con diversas características en el árbol (caminos que lleven a la victoria de uno u otro jugador, camino más probable, camino con más posibilidad de victoria para uno u otro jugador).
    c. Una tabla en la que se mostrará el movimiento recomendado por la inteligencia artificial, junto con una breve explicación textual y una representación en el árbol de cómo es más posible que trascurra la partida a partir de ese movimiento. De forma similar, también aparecerá una tabla con aquellas opciones consideradas más peligrosas o con menos posibilidades de llegar a victoria.
4. Ofrecer la posibilidad de entrar en partidas multijugador, en las que se competirá contra otros usuarios. Estas partidas pueden ser normales o clasificatorias, únicamente afectando a la puntuación de cada jugador las segundas.
    a. Tanto para las partidas normales como clasificatorias, existirán distintos subtipos en función de la limitación de tiempo: límite de tiempo por movimiento, por partida y jugador, o sin límite de tiempo.
5. Mantener una clasificación de mejores jugadores basada en los puntos de cada uno de ellos.
    a. Todos empezarán con cero puntos cuando registren una nueva cuenta.
    b. Al final de una partida clasificatoria, al ganador y al perdedor se les sumarán y restarán puntos, respectivamente.
    c. La puntuación a sumar o restar a cada jugador se calculará en función de la diferencia que exista entre ambos.
6. Permitir que los usuarios, una vez identificados, accedan a un *ranking* de mejores jugadores, en el que podrán ver el puesto, nombre de usuario y puntuación de cada uno.

7. La aplicación guardará en su base de datos las secuencias de movimientos de todas las partidas que se hayan producido, ya sea entre dos jugadores, o entre un jugador y la inteligencia artificial. Esto abrirá la posibilidad de que, en un futuro y con la suficiente cantidad de datos, la inteligencia artificial también sea capaz de aprender de sus rivales humanos.

2.3 Estudio de la Situación Actual

Las aplicaciones para jugar a diversos juegos de mesa no son nada reciente. Se pueden encontrar numerosos ejemplos de software de este tipo aplicado al juego *Dots and Boxes*, ya sean aplicaciones *web*, programas de escritorio o aplicaciones móviles para cualquier tipo de dispositivo.

En nuestro caso concreto, hemos optado por el desarrollo de una aplicación web, ya que permite que cualquier usuario juegue contra la inteligencia artificial, o contra otro jugador, simplemente utilizando un navegador desde cualquier dispositivo. De esta forma, no es necesario el desarrollo de software distinto, dependiendo del sistema operativo al que vaya dirigido, ni se obliga al usuario a descargar una aplicación más.

Para el desarrollo del cliente *web* se ha decidido utilizar el *framework* Angular. Por otra parte, el servidor y toda la lógica de *backend* estará implementado en Java. Las razones principales son la gran cantidad de librerías de código abierto que existen, y que es la tecnología con la que más nos familiarizamos a la hora del desarrollo *web backend* a lo largo de la carrera.

Para la comunicación entre cliente y servidor, será necesario utilizar la tecnología WebSockets, que permite establecer una conexión inicial que servirá como flujo de información y, a partir de ella, cliente y servidor podrán intercambiarse información sin que el otro haya hecho una solicitud previa. Esta tecnología es muy útil para juegos de navegador en línea, ya que permite que el servidor actualice tu juego siempre que sea necesario.

Para el almacenamiento de datos se ha decidido utilizar MongoDB, ya que cuenta con una API sencilla y extensamente documentada, y es ideal para casos como este en el que hay un modelo de datos muy sencillo y sin numerosas relaciones, y en el que no se necesita implementar transacciones complejas. Además, MongoDB nos permite almacenar los datos en *clusters* en su nube propia, relegándoles ciertas tareas, como la de mantener una copia de seguridad.

2.3.1 Evaluación de Alternativas

Aquí se describirán y se debatirán las ventajas e inconvenientes que presentan algunas de las tecnologías y proyectos similares evaluados para el desarrollo de este sistema.

2.3.1.1 Interfaz Web

Para la implementación de una interfaz gráfica para navegador existen dos opciones principales: usar HTML, JavaScript y CSS puros, o emplear uno de los diversos *frameworks* o librerías para desarrollo *web* que existen. Las principales ventajas de trabajar con uno de estos *frameworks* son las siguientes (no necesariamente ordenadas por relevancia):

- Aceleran enormemente el proceso de implementación. Estas librerías ofrecen una gran cantidad de componentes, estructuras y funcionalidades básicas ya creadas, por

lo que el desarrollador puede centrar sus esfuerzos en las partes y comportamientos específicos de su sistema.

- Por la misma razón que se acaba de mencionar, reducen de forma notable la cantidad de código a escribir.
- Mejoran la seguridad de las aplicaciones. Estos marcos de trabajo ya cuentan con muchas de las medidas de seguridad más habituales para evitar ciertos ataques como inyecciones SQL o manipulaciones no autorizadas de datos.
- Abordan el problema de mantener la interfaz sincronizada con el estado de la aplicación. Esto es algo que, utilizando JavaScript puro, es necesario hacer de forma explícita y, si no se tiene cuidado, puede afectar de forma muy negativa al rendimiento de la aplicación.
- Generalmente, dividen el código en componentes reutilizables, lo que facilita enormemente el uso de patrones de diseño y buenas prácticas. Esto permite estructurar la aplicación en distintos módulos y crear un código mucho más fácil de mantener, adaptar, modificar, corregir y reutilizar.
- Cuentan con una gran cantidad de librerías y componentes desarrollados por la comunidad y de uso gratuito.
- Permiten crear aplicaciones de una sola página muy fácilmente. Este tipo de aplicaciones se presentan al usuario como una única página HTML cuyo contenido se carga y modifica de forma dinámica, ofreciendo una experiencia similar a la de una aplicación de escritorio.

Aunque ofrecen una gran cantidad de ventajas, estos *frameworks* también tienen ciertos inconvenientes. Estos son, quizás, algunos de los más relevantes:

- Obligan a seguir un formato y unos procedimientos concretos, estableciendo un paradigma de programación menos flexible.
- Empeoran el tiempo de carga inicial de la página *web*. Los distintos marcos de trabajo vienen acompañados de una gran cantidad de código predefinido que tiene que llegar al navegador de la máquina cliente. Esto causa que la primera carga del sitio web sea notablemente más lenta.
- El código tiene que compilarse antes de ser servido. Aunque esto no afecta al usuario final, los desarrolladores y administradores del sistema deberán compilar el código antes de probarlo y servirlo en una dirección *web*.

Una vez tomada la decisión de emplear un *framework*, es necesario decidir cuál será. Las dos alternativas principales eran Angular y React. Finalmente, se ha optado por Angular ya que, aunque requiere mayor tiempo de aprendizaje, es un *framework* complejo que no requiere de otras librerías para funcionar, es más fácil configurar un proyecto y está pensado para el desarrollo con TypeScript. Este es un superconjunto de JavaScript que añade objetos basados en clases y tipos estáticos, lo que permite evitar gran parte de los errores en tiempo de ejecución.

2.3.1.2 Proyectos Parecidos

En esta sección se estudiará una serie de aplicaciones que permiten jugar *Dots and Boxes*, las cuales también se podrían utilizar para practicar y aprender o mejorar nuestras habilidades en este juego. Para ello, se destacarán sus características principales y se concretarán las ventajas e inconvenientes que presentan al compararlas con este trabajo.

Esta evaluación de alternativas ha permitido, además, extraer algunas ideas que han acabado siendo implementadas en nuestro trabajo.

2.3.1.2.1 DotsAndBoxes.org

Figura 2.1. Logo de DotsAndBoxes.org

Esta es una aplicación *web* con una interfaz gráfica muy sencilla (y no especialmente bonita) que permite jugar en un tablero de tamaño ampliamente configurable. Con esta aplicación, un usuario humano puede jugar partidas rápidas y sin complicación contra un algoritmo programado explícitamente para *Dots and Boxes*. También permite partidas de un jugador humano contra otro, pero con la desventaja de que no se puede jugar a través de internet, sino que ambos contrincantes usarán la misma pestaña de navegador para juagar en turnos alternos.

2.3.1.2.1.1 Ventajas

- Se puede acceder desde cualquier dispositivo al ser una aplicación de navegador.
- Interfaz muy sencilla de comprender y utilizar.
- Posibilita escoger entre una enorme variedad de tamaños de tablero.
- Permite jugar tanto contra su algoritmo como en multijugador local.
- Ofrece recomendaciones sobre qué movimiento realizar si el usuario las solicita.

2.3.1.2.1.2 Inconvenientes

- No ofrece ningún tipo de explicación de las recomendaciones.
- No permite registrarse.
- No implementa un *ranking* de jugadores (a pesar de que presenta un botón para ello).
- No es posible jugar partidas multijugador *online*.
- No permite configurar la dificultad del rival virtual.
- Los menús no son muy estéticos.

2.3.1.2.2 Gametable.org – Dots and Boxes

Figura 2.2. Logo de Gametable.org

Una aplicación de navegador que cuenta con una interfaz gráfica muy agradable y configurable. En ella, se pueden jugar partidas de entre dos y cuatro jugadores, que pueden estar controlados por un usuario humano o por el algoritmo. Permite configurar el tamaño del tablero (entre una pequeña variedad prefijada) y la dificultad del algoritmo rival. De nuevo, no implementa partidas a través de internet.

2.3.1.2.2.1 Ventajas

- Permite jugar desde cualquier dispositivo, al ser una aplicación *web*.
- Interfaz agradable y fácil de utilizar.
- Ofrece la posibilidad de escoger entre varios tamaños de tablero.
- Tiene una dificultad configurable para el rival virtual.
- Permite crear partidas de entre dos y cuatro jugadores en local.
- Para cada partida, es posible elegir qué jugadores controlará el algoritmo y cuáles serán controlados por los usuarios.

2.3.1.2.2.2 Inconvenientes

- No ofrece recomendaciones sobre qué movimiento es preferible realizar.
- No permite registrarse.
- No implementa un *ranking* de mejores jugadores.
- No es posible jugar partidas multijugador *online*.

2.3.1.2.3 OutOfTheBit – Puntos y Cajas

Figura 2.3. Logo de Puntos y Cajas, de OutOfTheBit

Puntos y Cajas es una aplicación móvil, disponible para dispositivos Android y iOS. Permite jugar partidas de jugador humano contra algoritmo, de dos jugadores desde el mismo dispositivo o de dos jugadores a través de internet. Además, permite asignar a las partidas *online* un código que se puede enviar a una persona concreta para jugar contra ella. En el aspecto gráfico no es especialmente mala, aunque sitúa los menús de forma un tanto caótica, lo que los hace confusos.

2.3.1.2.3.1 Ventajas

- Permite identificarse con tu cuenta de Google o Apple.
- Permite jugar partidas contra su algoritmo o contra otro jugador en local.
- Ofrece la posibilidad de jugar partidas contra otro jugador *online*.
- Implementa una clasificación de jugadores basada en puntos.
- Implementa estadísticas del jugador y logros.

2.3.1.2.3.2 Inconvenientes

- Solo se puede jugar desde un dispositivo Android o iOS.
- No ofrece recomendaciones sobre qué movimiento es preferible realizar.
- Presenta una interfaz y menús confusos y muy extensos para las pocas opciones de configuración que ofrece.
- No permite configurar la dificultad del rival virtual.
- No es posible variar el tamaño del tablero.

Capítulo 3. Aspectos Teóricos

En este capítulo serán brevemente descritos todos aquellos conceptos, tecnologías y herramientas que se utilizarán para la consecución de este proyecto.

3.1 Dots and Boxes

Dots and Boxes (también conocido en castellano como timbiriche o cajas y puntos, entre otros) es un popular juego de lápiz y papel. La primera referencia de la historia sobre este juego aparece en *L'Arithmétique Amusante* **[3]**, libro publicado en 1889 y escrito por el matemático francés Édouard Lucas.

Este juego, como el ajedrez, es un juego de información perfecta: los jugadores tienen toda la información de lo que ha pasado desde el principio del juego a la hora de realizar un movimiento o tomar una decisión. Además, en el juego no existen variables aleatorias que puedan influenciar el trascurso o el resultado de una partida, todo depende de las decisiones de los jugadores implicados.

El juego consiste en una matriz de $m \times n$ puntos. En cada turno, un jugador debe dibujar una línea vertical u horizontal conectando dos puntos consecutivos. El jugador que consiga "cerrar" una caja (*i.e.*, quien dibuje el último de los cuatro lados que componen un cuadrado concreto) sumará un punto a su marcador. Los turnos se suceden de forma alternativa, pero, en caso de que un jugador consiga un punto, podrá dibujar más líneas hasta que una no consiga cerrar otra caja. Cuando un jugador conquista una caja, se suele dibujar una marca dentro para facilitar el recuento final, ganando aquel jugador cuyo número de cajas cerradas sea mayor. Comúnmente, en una partida de este juego, participan entre dos y cuatro personas.

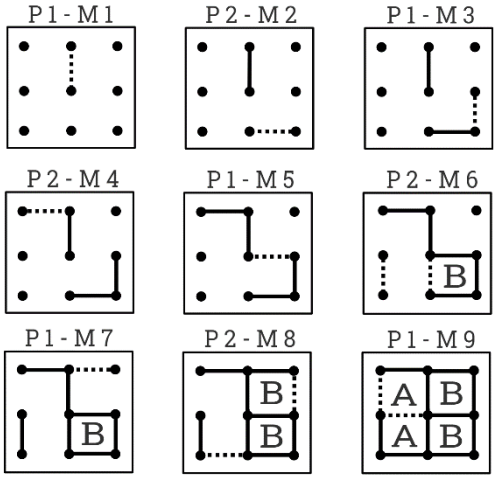


Figura 3.1. Partida de Dots and Boxes

La estrategia más inmediata a la hora de jugar a este juego es intentar evitar dibujar el tercer lado de un cuadrado, ya que esto propiciaría que tu rival conquistase esa caja. Si no hay

alternativa, se intenta minimizar el número de cajas que el rival podría conseguir a partir de ahí.

En la Figura 3.1 se puede observar el desarrollo de una partida de Dots and Boxes entre dos personas, llevada a cabo en un tablero de *3 × 3* puntos. En la imagen, *P1* y *P2* representan qué jugador ha realizado el movimiento, mientras que *Mn* hace referencia al número del movimiento que aparece inmediatamente debajo. Además, las marcas *A* y *B* indican, respectivamente, que el cuadrado en el que se encuentran ha sido cerrado por *P1* y *P2*.

3.2 Inteligencia Artificial

El término inteligencia artificial (IA), de forma general, engloba todas las estrategias y algoritmos que permiten que las computadoras, bajo unas condiciones determinadas, tomen decisiones o resuelvan problemas de forma parecida a como lo haría un humano. Para una definición más en profundidad de este concepto, así como para ver distintos tipos y aplicaciones de la IA, puede revisar el artículo escrito por González García *et al.* **[4]**.

3.2.1 Aprendizaje Automático

Por su parte, el aprendizaje automático (ML, por sus siglas en inglés) es una rama más de la inteligencia artificial, quizás la más importante en la llamada era del *Big Data*.

Un algoritmo de ML es un proceso computacional que utiliza datos de entrada para conseguir llevar a cabo una tarea deseada, pero sin estar explícitamente programado para producir un resultado concreto. Estos algoritmos adaptan su arquitectura a través de procesos de repetición para lograr ser mejores a la hora de realizar esa tarea deseada. Este proceso de adaptación es llamado entrenamiento. En él, se proporcionan al algoritmo muestras de datos de entrada y, en el caso de ML supervisado, los resultados esperados como salida para esos datos. Con los datos otorgados, el algoritmo intenta configurarse de forma óptima, no solo para obtener la salida esperada cuando se le pida evaluar parte de los datos de entrada, sino también para ser capaz de generalizar y generar buenos resultados cuando deba procesar datos nuevos para él, que nunca haya visto. En general, esta es la forma de "aprender" que tienen los algoritmos de aprendizaje automático. Además, y como punto a favor, este proceso de entrenamiento no tiene por qué estar limitado a una única fase inicial, sino que puede producirse a lo largo de toda la vida del algoritmo, aprendiendo continuamente de sus errores o de nuevos datos que se le presenten.

La idea tras el ML es emular, en cierta forma, la forma en la que los humanos y otros organismos con cerebros complejos aprenden, procesando los datos que obtienen a través de sus sentidos y adaptándose a ellos.

Generalmente, los algoritmos ML se dividen en tres tipos principales: supervisados, no supervisados y semisupervisados **[5]**. El primer tipo se caracteriza por que cada ejemplo de datos de entradas aparece acompañado de la etiqueta que lo clasifica, *i.e.*, de la respuesta que se espera del algoritmo si se le presenta ese dato de entrada. Los algoritmos no supervisados

también reciben datos de entrada, pero sin etiquetar. Por lo general, este tipo de sistemas se utilizan para identificar y extraer similitudes entre los datos de entrada y así encuadrarlos en una cierta categoría que el propio algoritmo ha establecido. El último tipo, el ML semisupervisado, se utiliza cuando se dispone de un conjunto de datos de entrada de los cuales solo una parte (generalmente pequeña en comparación con el total de datos) está etiquetada. La parte de datos etiquetados suelen servir para ayudar al algoritmo a clasificar y aprender de los datos no etiquetados. Quizás este último tipo, que se encuentra en el término medio, es el más similar a la forma en que los humanos aprendemos de nuestro entorno.

Este tipo de algoritmos tienen la capacidad de sustituir a humanos en trabajos que requieran una labor repetitiva y, además, presentan un gran potencial para encontrar y aprender patrones más complicados o sutiles en los datos de entrada de lo que la mayoría de (o todas) las personas son capaces de detectar. Gracias a estas características principales, este tipo de algoritmos se aplican hoy en día en una gran variedad de ámbitos que afectan a nuestras vidas: biomedicina **[6]**, oncología **[7]**, traducción automática **[8]**, conducción autónoma **[9]**, reconocimiento facial **[10]**, automatización y optimización de procesos industriales **[11]**, etc.

Las redes neuronales artificiales son uno de los métodos englobados dentro del aprendizaje automático. Estas redes utilizan diversas capas de procesamiento matemático conectadas, generalmente, unas tras otras, para adecuarse a los datos de entrada con los que son alimentadas. Un algoritmo de este tipo forma parte del sistema que tomamos como base para el desarrollo de este proyecto.

3.2.2 Árbol de Búsqueda de Monte Carlo

Los algoritmos de árbol de búsqueda de Monte Carlo (MCTS, por sus siglas en inglés) surgieron en 2006 como una forma de aplicar las simulaciones de Monte Carlo a los problemas de árboles de búsqueda **[12]**. En pocas palabras, el método de Monte Carlo es un método estadístico aplicado por primera vez en 1985 y que consiste en secuencias de acciones aleatorias para conseguir aproximar una expresión matemática compleja y costosa de evaluar de forma exacta.

MCTS es un método de búsqueda *best-first* ("mejor primero", en castellano) guiado por simulaciones de Monte Carlo. En contraposición a los algoritmos clásicos de árboles de búsqueda, como podría ser *A**, MCTS no requiere de una función de evaluación heurística.

Los algoritmos MCTS constan de dos partes: una estructura de árbol relativamente superficial y una serie de simulaciones en profundidad. Aplicado a un juego, la estructura de árbol determinaría los primeros movimientos de las partidas simuladas, mientras que los resultados de esas simulaciones irían dando forma al árbol.

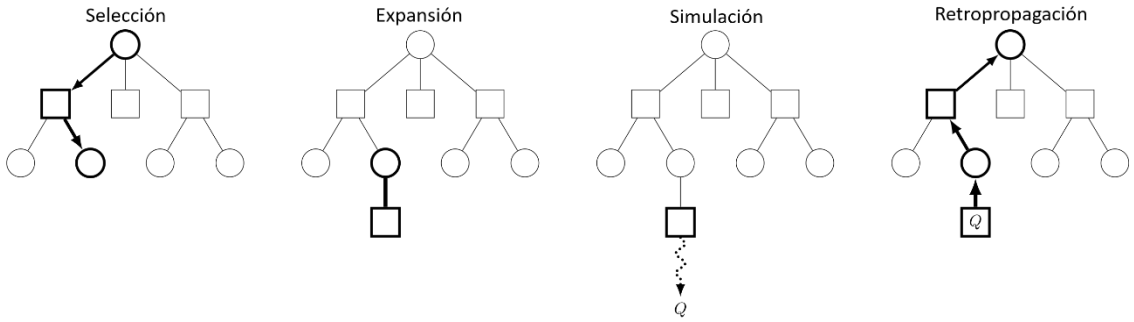


Figura 3.2. Fases de MCTS

Estos sistemas se dividen en cuatro fases principales que se irán repitiendo a lo largo de la ejecución del algoritmo. Cuantas más veces se itere sobre este proceso, el árbol crecerá más y será más fácil obtener un resultado significativo. También hay que destacar que, como cabe esperar, a mayor número de iteraciones, mayor tiempo de ejecución. En la fase de selección, el árbol es visitado desde su nodo raíz hasta un nodo terminal. En la fase de expansión, se añaden uno o más nodos al árbol, bajo el último nodo visitado. Después, a lo largo de la fase de simulación, se elige uno de estos nodos añadidos y se escogen estados posibles de forma aleatoria (o utilizando un enfoque que mejore los resultados, como es el caso del sistema que utilizaremos como base para nuestro trabajo, que aquí emplea redes neuronales artificiales) hasta llegar a un estado terminal. Finalmente, en la fase de retropropagación, los resultados obtenidos en la fase de simulación se actualizan desde las hojas hasta la raíz del árbol.

MCTS ha demostrado ser un método especialmente interesante y efectivo para dominios donde encontrar una función de evaluación heurística con la que se consigan buenos resultados es un trabajo especialmente difícil y que tome mucho tiempo. Un ejemplo de esto, son algunos juegos de mesa como, por ejemplo, el Go **[13]** o, como en nuestro caso, *Dots and Boxes* **[1]**.

3.3 Angular

Angular es un *framework* de código abierto para el desarrollo de aplicaciones de navegador (en concreto, la parte del cliente e interfaz de usuario de estas), tanto para dispositivos móviles como para ordenadores de sobremesa o portátiles y aptas para funcionar bajo cualquier sistema operativo. Este popular *framework* está desarrollado y mantenido por Google, con la ayuda de la comunidad, y su primera versión fue lanzada en el año 2016.

Utilizar Angular para el desarrollo de aplicaciones ofrece numerosas ventajas al estar construido bajo el patrón de arquitectura modelo-vista-controlador (MVC) y basar su estructura en componentes:

Figura 3.3. Logo de Angular

- Evita implementar características muy básicas, ya que contiene una serie de funcionalidades preparadas para ser utilizadas, por lo que el desarrollador puede centrarse en los requerimientos específicos de su software.
- Facilita el empleo de patrones de diseño y la reutilización de código. Los módulos en los que se divide una aplicación (componentes, siguiendo la terminología de Angular) pueden ser reutilizados, sobrescritos o adaptados a una necesidad particular.
- Facilita el mantenimiento del software y las pruebas unitarias.
- Propicia que varias personas trabajen en paralelo, ya que los distintos componentes pueden implementarse de forma individual, y después emplearse de forma conjunta para obtener el resultado esperado.
- Ofrece una gran cantidad de componentes, de código abierto y uso libre, ya desarrollados por la comunidad, y que pueden evitarte repetir la implementación de una funcionalidad que alguien ya ha compartido.

Uno de los principales atractivos de Angular es que permite implementar aplicaciones *web* de una sola página, las cuales ofrecen una experiencia más fluida a los usuarios, al emular en cierta forma una aplicación de escritorio. Con este tipo de aplicaciones, el navegador no necesita viajar entre distintas URLs y cargar páginas diferentes, sino que puede simplemente modificar y cargar dinámicamente el contenido de la página que está mostrando.

Además, Angular utiliza el lenguaje de programación TypeScript, un superconjunto de JavaScript publicado en 2012 por Microsoft. Este añade tipos estáticos y objetos basados en clases, lo que permite construir un código más robusto y evitar gran parte de los fallos y errores en tiempo de ejecución. También ofrece la ventaja de que, al ser un superconjunto de JavaScript y compartir su misma sintaxis, el código JavaScript ya existente también funciona en él. El código desarrollado en este lenguaje se traduce a código JavaScript, una vez compilado.

Como punto extra, este *framework* ofrece una gran versatilidad para el desarrollador al ser multientorno, lo que significa que se puede desarrollar en esta tecnología desde numerosos entornos de desarrollo integrados (IDEs, por sus siglas en inglés) o editores de texto, como pueden ser VisualStudio Code, JetBrains WebStorm, o incluso Notepad++.

Para el desarrollo de este trabajo se han usado las versiones 11.0.3 y 4.0.5 de Angular y TypeScript, respectivamente, para la implementación de la parte de cliente de nuestra aplicación *web*.

3.4 Java

Java es un lenguaje de programación y la plataforma para la que está diseñado el lenguaje con el mismo nombre. Ambos fueron publicados en 1996 por la compañía Sun Microsystems, que fue adquirida en 2010 por Oracle, quien ahora mantiene y evoluciona Java.

Figura 3.4. Logo de Java

Una de las principales ventajas de Java radica en que las aplicaciones desarrolladas usando este lenguaje, pueden compilarse una sola vez y ser ejecutadas sobre máquinas (al menos, en teoría) con cualquier sistema operativo y arquitecturas muy diversas. Esto es debido a que, utilizando esta tecnología, el código fuente es compilado a *bytecode* (archivos .class) que es ejecutado por la máquina virtual de Java (JVM, por sus siglas en inglés). La JVM interpreta estos archivos y los traduce en instrucciones nativas de la plataforma de destino. Evidentemente, para que esto sea posible, el sistema nativo ha de tener una versión correcta de la JVM instalada, y el código debe desarrollarse sin realizar de forma directa llamadas nativas o funciones específicas de la plataforma.

Otra característica que hace de Java una tecnología interesante es la orientación a objetos, un paradigma que permite que los distintos tipos de datos estén ligados a sus operaciones propias. Esto también permite separar el comportamiento (la clase, lo que no cambia), del estado (la instancia, los datos concretos, lo que es mutable). Además, esto también permite, en principio, la creación de objetos más genéricos que permitan la reutilización de código y faciliten la aplicación de patrones de diseño.

Además de todo esto, Java es seguro, fiable, relativamente rápido, y cuenta con una sintaxis (inspirada en las de C y C++) muy sencilla y fácil de aprender.

Aunque no es fácil determinar qué lenguaje de programación es el más usado, la mayoría de las fuentes disponibles coinciden en que Java sigue siendo uno de los más populares. Esto se traduce en que tiene una gran comunidad y existen numerosas librerías de código abierto que se pueden reutilizar.

Quizás por todo esto Java es, posiblemente, el lenguaje que cuenta con mayor presencia en el software desarrollado para servidores *web*. En nuestro caso, también hemos escogido esta

tecnología para el desarrollo del *backend* de nuestro proyecto, empleando para ello la versión 13.0.2 del *Java Development Kit* (referido normalmente como Java 13).

3.5 WebSockets

WebSockets es una tecnología que permite establecer una sesión de comunicación bidireccional, simultánea y de largo recorrido en el tiempo, sobre un socket TCP (siglas en inglés de Protocolo de Control de Transmisión). Implementada entre un cliente y un servidor, y una vez se establezca la conexión inicial, permite que ambos envíen y reciban respuestas controladas por eventos sin que el cliente deba consultar al servidor continuamente.

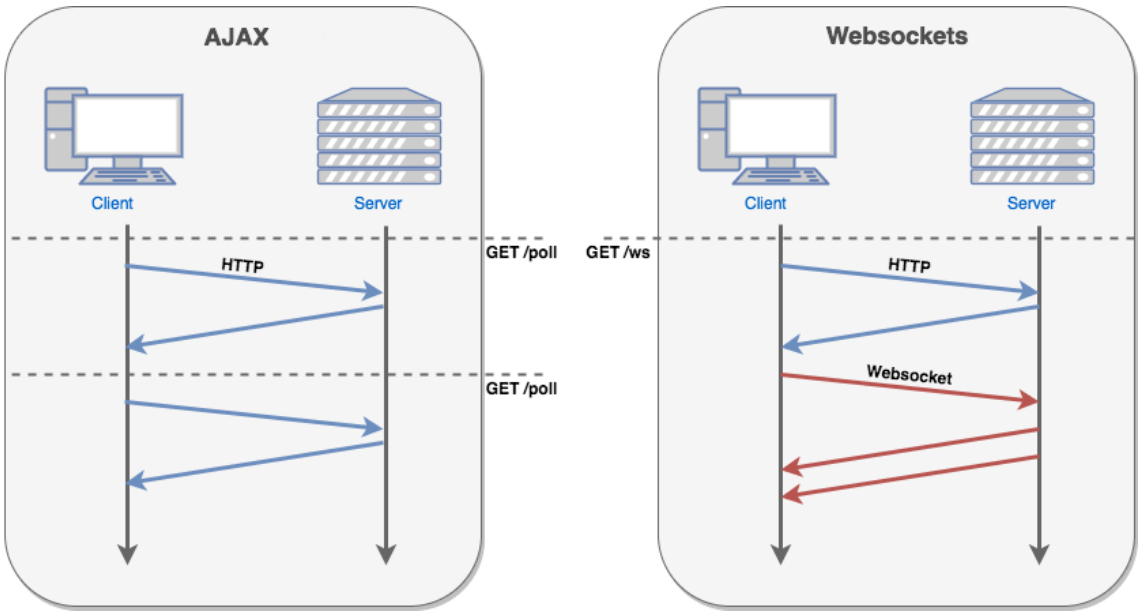


Figura 3.5. Comunicación mediante WebSockets en oposición a comunicación AJAX

Estas capacidades convierten a esta tecnología en un candidato ideal para implementar la conexión entre cliente y servidor *web* para un juego, ya que permite que, estableciendo una única conexión inicial, ambos compartan, en cualquier momento, todas las actualizaciones necesarias sobre el estado de la partida. Esa es, precisamente, la parte de nuestra aplicación *web* que cubriremos con el uso de este protocolo.

Las conexiones de este tipo se establecen tras una petición de negociación (*handshake* en inglés) del cliente, a la que el servidor enviará su respuesta, de forma similar a como se haría al establecer una conexión HTTP. La especificación normalizada de esta tecnología establece, además, dos nuevos esquemas de URI (siglas en inglés de Identificador de Recursos Uniforme), *ws:* y *wss:* para conexiones no cifradas y cifradas, respectivamente. El resto de la sintaxis del URI no se diferencia de la utilizada para el protocolo HTTP.

3.6 MongoDB

Figura 3.6. Logo de MongoDB

MongoDB es una base de datos NoSQL, de código abierto, multiplataforma y orientada a documentos. Apareció por primera vez en 2009, siendo su creadora la compañía de igual nombre. Está desarrollada utilizando los lenguajes de programación C++, JavaScript y Python, y utiliza el formato de intercambio de datos BSON (acrónimo del inglés *Binary JSON*), similar a JSON pero que emplea una representación binaria.

MongoDB es una base de datos rápida, eficiente y que implementa amplias posibilidades a la hora de realizar consultas. Además, cuenta con un esquema de estructuras de datos dinámico, lo cual facilita el guardado de datos. Aunque en versiones anteriores esta base de datos no soportaba transacciones ACID (siglas en inglés de Atomicidad, Consistencia, Aislamiento y Durabilidad), esta funcionalidad se añadió en 2018. También en versiones anteriores se acusó a MongoDB de tener problemas de rendimiento y consistencia cuando se excedía un cierto volumen de datos.

En este proyecto se ha utilizado la versión 4.4.11 de MongoDB como base de datos, ya que parece ideal para aplicaciones en las que el volumen de datos no va a ser excesivo, cuyo esquema de datos no vaya a ser especialmente complicado y que no requiera de transacciones complejas. La compañía también ofrece la posibilidad de alojar tus bases de datos en un *cluster* alojado en su nube, lo que permite relegar tareas como la de mantener una copia de seguridad.

3.7 Métrica 3

Figura 3.7. Logo del Ministerio de Hacienda y Función Pública del Gobierno de España

Métrica 3 (o Métrica Versión 3) es una metodología de planificación, desarrollo y mantenimiento de sistemas de información. Fue publicada en 2001, y creada por el Consejo

Superior de Informática y la consultora IECISA, bajo la tutela del antiguo Ministerio de Administraciones Públicas del Gobierno de España (ahora Ministerio de Hacienda y Función Pública, quien la promueve actualmente). Su objetivo es conseguir la sistematización de todas las actividades comprendidas en el ciclo de vida de los proyectos software, sobre todo en el ámbito de la administración pública. Es de uso libre, con la condición de citar la entidad a la cual pertenece su propiedad intelectual: el Ministerio de Hacienda y Función Pública del Gobierno de España.

Los procesos detallados dentro de esta metodología son los siguientes:

- Planificación de Sistemas de Información (PSI).
- Desarrollo de Sistemas de Información (DSI); dividido, a su vez, en subprocesos:
    - Estudio de Viabilidad del Sistema (EVS).
    - Análisis del Sistema de Información (ASI).
    - Diseño del Sistema de Información (DSI).
    - Construcción del Sistema de Información (CSI).
    - Implantación y Aceptación del Sistema (IAS).
- Mantenimiento de Sistemas de Información (MSI).

Métrica 3 establece, además, cuatro interfaces cuya finalidad es mejorar los procesos anteriormente descritos y garantizar que se consigue el objetivo del desarrollo:

- Gestión de Proyectos (GP).
- Seguridad (SEG).
- Aseguramiento de la Calidad (CAL).
- Gestión de la Configuración (GC).

Es posible encontrar más información sobre cualquiera de estos procesos e interfaces en el Portal de Administración Electrónica.

Esta metodología será la empleada para el desarrollo del presente documento, aunque limitándola a los apartados más adecuados para este tipo de proyecto, y adaptándola a sus necesidades específicas.

3.8 Scrum

Scrum es un marco de trabajo en el cual se busca abordar los problemas complejos de manera adaptativa. Cuando se emplea esta metodología de desarrollo ágil, se intenta trabajar de una forma productiva y creativa para entregar productos del mayor valor posible.

El proceso Scrum (representado en la Fig. 3.11) se compone de una serie de iteraciones, llamadas *sprints*. Al principio del proceso, el *Product Owner* (que actuaría como representante dentro del proceso de las partes interesadas, o *stakeholders*) vuelca un problema complejo en un *Product Backlog* (que podría traducirse como "pila de producto"), dividiéndolo en distintos requerimientos. En cada *sprint*, el *Scrum Team* convierte una selección del trabajo del *Product Backlog* (el *Sprint Backlog*) en un incremento del valor del producto. Al final de un *sprint*, el *Scrum Team* (y las partes interesadas) se encargan de inspeccionar y valorar los resultados obtenidos, así como planificar el *sprint* siguiente.

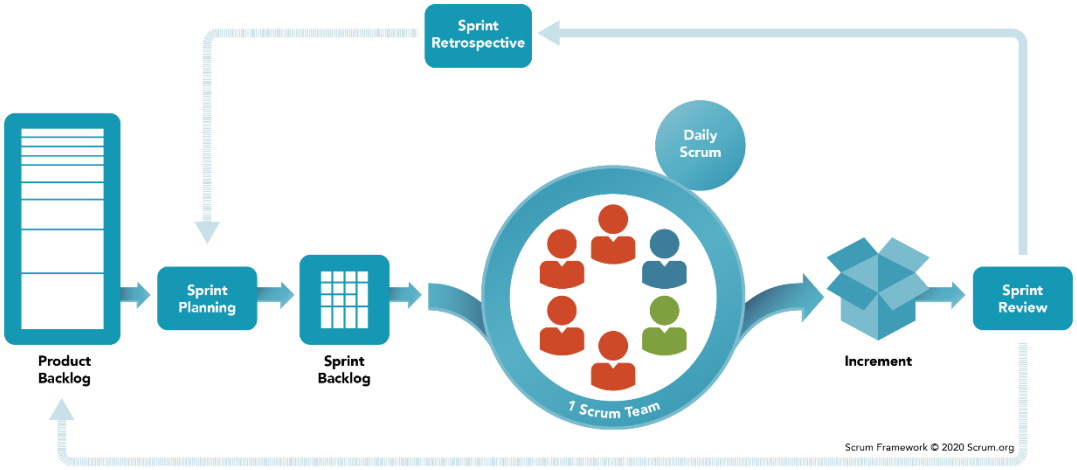


Figura 3.8 . Esquema del proceso Scrum (fuente: scrum.org)

En una aplicación clásica de esta metodología, el *Scrum Team*, por su parte, está formado por el equipo de desarrollo, un *Scrum Master* (que es quien coordina el proceso Scrum y puede hacer de mediador entre partes) y un *Product Owner*.

Capítulo 4. Gestión del Proyecto

Para este proyecto se ha empleado un modo de desarrollo ágil e iterativo. Asimilando la metodología seguida a una metodología Scrum clásica, yo (el autor de este TFG) sería el equipo de desarrollo. Los *Product Owners* (representantes de todos los actores interesados) serían, durante mi época de becado en la FUO (Fundación Universidad de Oviedo), los responsables del equipo de investigación, y, después, el tutor del TFG.

El desarrollo del producto se puede dividir en dos grandes conjuntos de *sprints*. El primero de ellos mientras estuve en el grupo de investigación, en el que las reuniones con los *Product Owners* tenían un carácter semanal. Y el segundo, desde mi salida del grupo, hasta la finalización del proyecto, en el que, además de reuniones puntuales, la mayoría de las comunicaciones con el *Product Owner* se realizaron vía email.

En el primero de estos conjuntos se quería conseguir una herramienta que permitiese a usuarios humanos estudiar y aprender de las decisiones tomadas en partidas del juego *Dots and Boxes* por una IA (desarrollada en un proyecto previo del equipo de investigación). Una vez conseguido lo que se buscaba, se realizaron una serie de pruebas con usuarios reales para evaluar si la herramienta ayudaba al aprendizaje.

En el segundo conjunto de *sprints* se quiso aumentar las funcionalidades de ese sistema ya desarrollado, hacerlo más atractivo para los usuarios y aplicar las correcciones extraídas de las pruebas con usuarios.

A continuación, podrán verse dos tablas (Tablas 1 y 2). La primera se corresponderá con el *Product Backlog* original, para la primera de las divisiones del desarrollo. En la segunda, por su parte, estarán contenidos los elementos fijados en el *Product Backlog* del segundo conjunto de iteraciones, que buscaba aumentar las posibilidades de la herramienta desarrollada en el anterior.

ID	Historia	Prioridad	Dificultad
1	Implementar una herramienta que permita analizar y estudiar las decisiones del sistema inteligente MCTS con redes neuronales.	Alta	Alta
1.1	Exportar las decisiones desde el proyecto Java en un formato estándar.	Alta	Baja
1.1.1	El formato debe contener el árbol generado por el algoritmo.	Alta	Baja
1.1.2	Cada nodo del árbol debe ir acompañado del estado del tablero que representa, el número de visitas que ha recibido por parte del algoritmo, la puntuación de cada jugador en ese momento y la probabilidad de victoria calculada para cada jugador.	Media	Baja
1.2	Crear una interfaz gráfica que sea capaz de mostrar ese árbol a los usuarios.	Alta	Alta
1.2.1	Permitir que el usuario pueda interactuar con el árbol, colapsando y expandiendo nodos.	Media	Media
1.2.2	Implementar un modo experto dentro de la interfaz gráfica que permita explorar el árbol de formas distintas.	Alta	Alta
1.2.2.1	El modo experto debe ofrecer una serie de consultas para obtener distintos tipos de caminos en el árbol.	Alta	Alta
1.2.2.1.1	Consultas que permitan obtener caminos que lleven a la victoria del Jugador 1: todos los caminos, el más rápido, el más posible (el que tiene más visitas) y el que tiene mejor porcentaje para este jugador.	Media	Media
1.2.2.1.2	Consultas que permitan obtener caminos que lleven a la victoria del Jugador 2: todos los caminos, el más rápido, el más posible (el que tiene más visitas) y el que tiene mejor porcentaje para este jugador.	Media	Media
1.2.2.1.3	Consultas que permitan obtener caminos que lleven a empate: todos los caminos, el más rápido, el más posible (el que tiene más visitas) y el que tiene mayor porcentaje de empate.	Baja	Media
1.2.2.1.4	Consulta que permita obtener el camino más probable (el más visitado)	Alta	Baja
1.2.2.1.5	Consulta que permita buscar el camino con mejor porcentaje de victoria para el Jugador 1	Alta	Baja
1.2.2.1.6	Consulta que permita buscar el camino con mejor porcentaje de victoria para el Jugador 2	Alta	Baja
1.2.2.2	El modo experto permitirá al usuario configurar el número máximo de nodos que quiere expandir y/o su profundidad máxima al hacer *click* en un nodo.	Baja	Media
1.2.3	Implementar un modo no experto dentro de la herramienta, que ofrezca recomendaciones justificadas al usuario.	Alta	Alta
1.2.3.1	Ofrecer al usuario un movimiento recomendado, que será el que ofrezca un mejor balance entre probabilidad de victoria y número de visitas (confiabilidad).	Alta	Baja
1.2.3.2	Ofrecer al usuario un conjunto de movimientos no recomendados o movimientos que debería evitar.	Media	Media
1.2.3.3	Para cada una de esas recomendaciones, añadir una	Media	Media

	breve explicación textual y un camino en el árbol que muestre lo que se espera que pase a partir de ese movimiento.		
1.2.4	Marcar en el árbol el nodo que representa el movimiento recomendado.	Alta	Baja
1.2.5	El usuario debe poder ver, en el momento que desee, toda la información disponible sobre un nodo: estado del tablero; puntuaciones y probabilidades de victoria para cada jugado, y número de visitas.	Media	Baja
1.2.6	Crear una ventana de ayuda que ofrezca información de todas las posibilidades que implementa la herramienta y de qué significan los distintos datos disponibles.	Media	Baja
2	Permitir al usuario jugar partidas contra la IA para entrenar sus habilidades.	Alta	Alta
2.1	Añadir a la herramienta una pantalla desde la que jugar.	Alta	Media
2.1.1	Implementar un tablero interactivo desde el que el usuario podrá realizar movimientos y ver los que hace el rival.	Alta	Media
2.1.2	Crear un área de información que permita ver cuántos puntos lleva cada jugador sin tener que contar el número de cajas cerradas.	Media	Baja
2.2	Implementar un servidor que se encargue de toda la lógica de la partida y sea capaz de comunicarse con el algoritmo y con el usuario.	Alta	Alta
2.3	El usuario podrá decidir si quiere jugar recibiendo recomendaciones de la IA, o sin ningún tipo de ayuda.	Baja	Baja
2.3.1	En caso de que quiera jugar con recomendaciones, se generará la decisión del algoritmo para la situación del jugador y se le permitirá analizarla mediante la herramienta que se va a implementar.	Alta	Baja
2.3.1.1	El usuario podrá cambiar entre la vista de tablero y la de árbol en cualquier momento.	Media	Baja

Tabla 1. Product Backlog del primer conjunto de sprints

ID	Historia	Prioridad	Dificultad
1	Reducir el tiempo de carga inicial para los juegos contra la IA.	Baja	Baja
2	Dejar más claro cuándo se está esperando por un movimiento rival.	Media	Baja
3	Implementar que el tablero de juego ajuste su tamaño a las dimensiones de la ventana del navegador.	Media	Media
4	Aumentar la *hitbox* (el área a su alrededor donde se detecta un *click*) de los movimientos en el tablero.	Baja	Baja
5	Crear un historial de movimientos que esté disponible mientras se juega.	Alta	Media
5.1	Guardar cada movimiento realizado por el jugador y por el rival.	Alta	Baja
5.2	Mostrarlos en una nueva pantalla de interfaz gráfica, de forma ordenada.	Alta	Media
6	Arreglar los defectos encontrados.	Alta	Baja
6.1	Solventar el *bug* que permite pulsar en componentes que no están siendo mostrados en pantalla.	Alta	Baja
6.2	Atajar el bug que permite realizar movimientos, en ocasiones concretas, mientras se está esperando.	Alta	Baja
7	Guardar la secuencia de movimientos de todas las partidas, para reentrenar a la IA en un futuro.	Media	Baja
7.1	Implementar guardado en archivo.	Baja	Baja
7.2	Implementar guardado en una base de datos.	Media	Baja
8	Hacer visible el área de información sobre la partida en cualquier lugar de la pantalla de juego.	Baja	Baja
9	Permitir al usuario configurar la dificultad del rival virtual.	Media	Media
9.1	Añadir un área en la interfaz gráfica para que el usuario elija la dificultad del rival antes de empezar una partida.	Media	Baja
9.2	Implementar que el servidor sea capaz de recibir este parámetro de dificultad y adecuar las iteraciones del algoritmo cuando está actuando como rival virtual.	Media	Media
10	Crear un archivo de configuración para que el administrador del sistema pueda modificar desde ahí ciertos parámetros del servidor.	Baja	Media
10.1	Definir qué parámetros serán extraídos a este archivo.	Baja	Baja
10.2	Implementar que las distintas clases del servidor sean capaces de extraer estos parámetros.	Baja	Media
11	Permitir a los usuarios jugar partidas multijugador (partidas online contra otro humano)	Alta	Alta
11.1	Implementar la lógica necesaria para esto en el servidor.	Alta	Alta
11.1.1	Implementar la lógica de comunicaciones con los clientes.	Alta	Alta
11.1.2	Implementar la lógica de juego.	Alta	Alta
11.2	Permitir a los usuarios jugar partidas multijugador clasificatorias (que afecten al total de puntos de cada jugador).	Alta	Baja
11.3	Permitir a los usuarios jugar partidas multijugador normales (que en ningún caso afecten a los puntos de los jugadores).	Media	Baja
11.4	Dejar a los jugadores escoger entre distintas limitaciones de tiempo para una partida.	Baja	Alta

11.4.1	Limitación de tiempo por partida y jugador.	Baja	Media
11.4.2	Limitación de tiempo por movimiento.	Baja	Media
11.4.3	Sin limitación de tiempo.	Baja	Baja
11.4.4	El usuario podrá ver el tiempo que le queda en el área de información de la partida.	Baja	Baja
11.5	Implementar que el sistema pueda detectar si un jugador abandona la partida de cualquier manera, en cuyo caso perderá.	Alta	Alta
11.6	Permitir que el sistema sea capaz de detectar si un jugador intenta hacer trampas. En ese caso, perderá la partida.	Alta	Media
12	Crear una base de datos donde el sistema pueda guardar la información relativa a usuarios y partidas.	Alta	Media
12.1	Escoger la base de datos a usar.	Alta	Media
12.2	Implementar la comunicación entre el servidor y la BD.	Alta	Media
13	Los usuarios deben poder iniciar sesión con su cuenta.	Alta	Alta
13.1	Un usuario podrá registrarse en la aplicación introduciendo un nombre de usuario, un email y una contraseña.	Alta	Alta
13.1.1	Estos datos deben ser validados.	Alta	Baja
13.1.1.1	Definir las validaciones a realizar.	Alta	Baja
13.1.1.2	Implementar esas validaciones.	Alta	Baja
13.2	Los datos se guardarán en la BD.	Alta	Media
13.2.1	La contraseña debe guardarse encriptada.	Media	Baja
13.2	Un usuario podrá acceder a la aplicación con las credenciales de una cuenta ya creada (email y contraseña).	Alta	Media
14	Los usuarios identificados podrán ver su perfil.	Baja	Baja
15	Los usuarios identificados podrán ver *ranking* de jugadores ordenado por puntuación de forma descendente.	Media	Media

Tabla 2. Product Backlog del segundo conjunto de sprints

Capítulo 5. Análisis

En este capítulo se documentará el análisis de los distintos módulos del sistema. En las siguientes páginas se establecerá hasta dónde llegará el desarrollo del sistema, se extraerán los requisitos que debe cumplir, y se realizarán los análisis preliminares de subsistemas, clases, interfaces de usuario y pruebas.

5.1 Definición del Sistema

En esta sección se tratará, principalmente, el alcance del sistema a desarrollar, hasta donde se quiere llegar y qué límites se van a fijar.

5.1.1 Determinación del Alcance del Sistema

El sistema a implementar será una SPA (siglas en inglés de aplicación de página única) para navegador, que permitirá a los usuarios registrados jugar al juego de mesa *Dots and Boxes*, en partidas de distintas modalidades. Para ello será necesario implementar tres subsistemas diferenciados: la aplicación de cliente *web*, el software para el servidor y la base de datos.

Lo primero que un usuario cualquiera podrá hacer una vez acceda a la aplicación de navegador será, si tiene una cuenta creada, iniciar sesión utilizando un email y una contraseña y, si no tiene una cuenta creada, registrarse. Para esto último deberá introducir un email, un nombre de usuario y una contraseña. Se realizarán comprobaciones sobre estos campos tanto en la parte de cliente (para así evitar enviar una conexión al servidor si los fallos se pueden detectar de forma precoz), como en la parte del servidor (para aquellas comprobaciones que sea imposible realizar de otra forma, como puede ser la presencia del email utilizado en la base de datos, y para evitar que potenciales usuarios con malas intenciones eviten las comprobaciones de seguridad).

Una vez tenga la sesión iniciada, al usuario se le presentará un menú con varias opciones distintas: cerrar sesión, ver el *ranking* de jugadores, ver la información relativa a su perfil (su cuenta) y escoger entre distintos tipos de partida.

Los tres tipos principales de partida que se implementarán serán los siguientes: partida de entrenamiento contra la IA, partida multijugador normal (no afecta a la puntuación de los jugadores), y partida multijugador clasificatoria (tras la cual se modificarán las puntuaciones de ambos jugadores en función de quién gane o pierda.

Antes de empezar una partida de entrenamiento, el jugador entrará en una pantalla de configuración, en la que decidirá quién realiza el primer movimiento (él o la IA), tendrá la capacidad de modificar la dificultad del rival virtual (escogiendo entre cuatro opciones) y podrá escoger si quiere, o no, recibir las recomendaciones producidas por la IA. Esto último, además, también se podrá modificar durante el trascurso de la partida. En este tipo de partidas, el

usuario tendrá a su disposición tres pantallas distintas entre las que podrá cambiar en cualquier momento de la partida:

- Un tablero interactivo, en el que podrá observar el estado actual de la partida, además de realizar movimientos desde él, simplemente pulsando encima de la línea que quiere dibujar, siempre que sea su turno.
- Un árbol interactivo, desde el que podrá estudiar todos los movimientos que tiene disponibles a partir del estado actual, junto con toda la información que se disponga sobre estos (estado del tablero, posibles movimientos posteriores del rival, probabilidad calculada de victoria, etc.). Además, en esta pantalla, aparecerán tanto el movimiento recomendado como aquellos a evitar, acompañados todos de sendas explicaciones. También, sobre ese mismo árbol, podrá realizar una serie de consultas predefinidas para estudiar en más profundidad sus posibilidades.
- Un historial, donde se mostrarán, de forma ordenada, todos los movimientos realizados por ambos jugadores, desde el estado inicial del tablero, al actual.

Para las partidas multijugador, tanto normales como clasificatorias, el usuario podrá escoger entre otros tres subtipos, atendiendo a la limitación de tiempo: límite por movimiento, límite por partida y jugador, o sin ningún límite de tiempo. En este tipo de partidas solo estarán disponibles el tablero interactivo y el historial de movimientos.

Para guardar los datos, se empleará una base de datos MongoDB alojada en un *cluster* en los servidores que la propia compañía ofrece.

No se implementarán tipos de partidas distintas a los mencionados (como podrían ser partidas con más de dos jugadores o partidas multijugador en las que se cuente con asistencia de la IA). Tampoco se desarrollará un cliente específico para el administrador del sistema, ya que MongoDB ya ofrece una interfaz *web* desde la que modificar los datos guardados y, además, habrá ciertos parámetros relativos a la lógica del servidor que podrán ser modificados desde un archivo de configuración, sin necesidad de recompilar el programa. De igual manera, el sistema no contará con funcionalidades, además de las mencionadas, relativas a las cuentas de usuario (validar cuenta, cambiar contraseña, añadir amigos, etc.).

5.2 Requisitos del Sistema

Esta sección está dedicada a fijar los requisitos del sistema, los actores identificados y los distintos casos de uso

5.2.1 Obtención de los Requisitos del Sistema

En este apartado se procederá a establecer las especificaciones del sistema en forma de tablas de requisitos. La primera y segunda tabla contendrán, respectivamente, los requisitos funcionales orientados al usuario y los requisitos funcionales relativos a la lógica del servidor. La tercera tabla guardará los requisitos no funcionales.

Todos los requisitos presentes en esta sección deberán ser aprobados e implementados en el sistema final.

5.2.1.1 Requisitos Funcionales

Código	Nombre Requisito	Descripción del Requisito
RF1	Registro de usuario	El sistema permitirá a los usuarios no identificados crear una nueva cuenta. Para ello, estos deberán rellenar una serie de campos en un formulario.
RF1.1	Validación de datos	Para cada campo, se establecerán una o varias características a comprobar. Estas comprobaciones podrán ocurrir en el lado del cliente, si es posible, y en el servidor de forma obligatoria, hayan sido o no validadas por el cliente *web*.
RF1.1.1	Validación fallida	En caso de que los datos insertados por el usuario no superasen estas comprobaciones, el sistema deberá ser capaz de mostrarle un mensaje que se adecúe a la condición por la que el registro no ha sido efectivo. Si se da el caso de que la validación de esos datos falle, el nuevo usuario no será insertado en la base de datos del sistema.
RF1.1.2	Validación exitosa	Si y solo si los datos superan los procesos de comprobación, el nuevo usuario será insertado en la base de datos. Además, tras este registro exitoso, al usuario se le deberá mostrar, sin ningún paso intermedio, la pantalla de sesión iniciada.
RF1.2	Datos a introducir	Estos serán los datos que el usuario deberá introducir para su registro, junto con las comprobaciones que deberán superar.
RF1.2.1	Nombre de usuario	Este campo no puede estar vacío, debe ser único en el sistema, y tener una longitud máxima de 20 caracteres.
RF1.2.2	Contraseña	Este campo no puede estar vacío, y debe contar con una longitud de entre seis y 40 caracteres

		(ambos inclusive).
RF1.2.3	Contraseña repetida	El contenido de este campo deberá coincidir exactamente con el de Contraseña.
RF1.2.4	Email	Este campo no puede estar vacío y debe ser único en el sistema. Además, se deberá comprobar que es un formato de email válido.
RF2	Inicio de sesión	Nada más abrir la aplicación *web*, los usuarios podrán iniciar sesión si ya tienen una cuenta creada. Para esto se pedirán un email y una contraseña.
RF2.1	Comprobación de datos	El cliente deberá comprobar que estos campos no están vacíos y, de ser así, la petición de inicio de sesión será trasladada al servidor. El servidor comprobará si, en efecto, existe ese par email-contraseña en la base de datos.
RF2.1.1	Comprobación fallida	En caso de que estas comprobaciones fallen, el usuario no iniciará sesión y se le mostrará un mensaje de error.
RF2.1.2	Comprobación exitosa	Si este es el caso, al usuario se le mostrará, sin ningún paso intermedio, la pantalla de sesión iniciada.
RF3	Pantalla de sesión iniciada	Si el usuario se ha conseguido registrar o identificar correctamente, se le mostrará una pantalla en la que tendrá acceso a las siguientes opciones.
RF3.1	Información de usuario	Esta opción permitirá al usuario identificado examinar distintos datos sobre su perfil.
RF3.1.1	Usuario	
RF3.1.2	Email asociado	
RF3.1.3	Puntuación de clasificatorias	Cada cuenta tendrá asociada una puntuación que servirá para posicionarla en el *ranking*. Al registrarse, todas las cuentas empezarán con 0.
RF3.2	*Ranking* de jugadores	Mediante esta opción, el usuario podrá ver información de todos los usuarios, ordenados por puntuación en sentido decreciente, y de forma paginada (máximo 10 jugadores por página). Sobre cada jugador se mostrarán varios datos.
RF3.2.1	Posición en el *ranking*	
RF3.2.2	Nombre de usuario	
RF3.2.3	Puntuación de clasificatorias	
RF3.3	Cerrar sesión	Esta opción permitirá al usuario terminar su sesión actual, lo que le llevará de vuelta a la pantalla de inicio de sesión y registro.
RF3.4	Partida de entrenamiento	Desde esta opción, el usuario podrá solicitar al servidor una partida en la que la IA actuará como rival virtual. Además, antes de esto, podrá configurar una serie de parámetros.
RF3.4.1	Jugador 1 o Jugador 2	Aquí, el usuario podrá elegir si él realiza el primer movimiento, escogiendo ser el Jugador 1, o si mueve primero su rival, escogiendo ser el Jugador 2.

RF3.4.2	Dificultad	El jugador podrá variar la dificultad de la IA que actúa como rival, escogiendo entre cuatro opciones distintas.
RF3.4.3	Recibir recomendaciones	El usuario también tendrá la capacidad de decidir si quiere, o no, recibir recomendaciones de la IA durante la partida. Esto también deberá poder modificarse mientras se juega.
RF3.5	Partida multijugador normal	Mediante esta opción el usuario podrá solicitar al servidor una partida contra otro jugador humano. Este tipo de partidas no afectará a la puntuación de cada jugador. Además, se podrá escoger entre tres subtipos.
RF3.5.1	Tiempo máximo por partida	En este tipo de juegos, cada jugador dispondrá de un tiempo máximo para decidir sus movimientos durante toda la partida. En todo momento deberá tener en pantalla la información de cuánto tiempo le queda. El tiempo restante de cada jugador solo deberá decrecer durante su turno. Si el tiempo se acaba, el jugador pierde automáticamente.
RF3.5.2	Tiempo máximo por movimiento	Aquí cada jugador dispondrá de un tiempo máximo para decidir cada movimiento. Tras realizar un movimiento, volverá a tener el tiempo máximo disponible. De nuevo, debe poder ver en todo momento cuánto tiempo le queda. De agotarse el tiempo, el jugador perderá automáticamente.
RF3.5.3	Tiempo ilimitado	En este tipo de partidas cada jugador dispondrá de todo el tiempo que quiera para realizar sus movimientos.
RF3.6	Partida multijugador clasificatoria	Esta opción permitirá al usuario solicitar una partida contra otro jugador humano, que además afectará a sus puntos de clasificatoria. Este tipo de partidas contará con los mismos subtipos que la partida multijugador normal.
RF3.6.1	Modificación de la puntuación	Al acabar una partida clasificatoria, se calcularán los puntos a sumar y restar al ganador y perdedor, respectivamente. Estos puntos deberán ser calculados a partir de la diferencia de puntuación que exista entre ambos jugadores, estableciendo también un máximo y un mínimo por partida.
RF3.6.1.1	Puntuación mínima	También se establecerá un límite de puntuación mínima para una cuenta. Si un jugador alcanza ese límite y, posteriormente, pierde una partida clasificatoria, su puntuación nunca podrá volver a situarse por debajo del límite mentado.
RF4	Partida iniciada	Una vez el servidor haya incluido al jugador en una nueva partida, este tendrá disponible diversas opciones.
RF4.1	Abandonar partida	Esta opción permitirá al usuario salir de una partida empezada. De ser una partida clasificatoria (a no ser que la partida ya haya acabado), esto será considerado una derrota igualmente.

RF4.2	Información de la partida	Aquí el jugador dispondrá de un marcador con sus puntos y los de su rival, acompañados de sendos nombres de usuario. También aparecerá el tiempo que le queda disponible, únicamente en aquellas partidas que implementen esta limitación.
RF4.3	Recibir recomendaciones	Mediante esta opción, el jugador decidirá si, para los movimientos siguientes, desea recibir las recomendaciones de la IA, o no. Esto solo estará disponible en las partidas de entrenamiento.
RF4.4	Tablero interactivo	Desde aquí, el jugador podrá contemplar el estado actual de la partida, además de realizar movimientos pinchando en la línea que desea dibujar, siempre que sea su turno.
RF4.5	Historial de movimientos	Aquí el usuario podrá estudiar todos los movimientos que tanto él, como su rival, hasta el momento presente. Esto incluirá todos los estados de la partida, desde el estado inicial, hasta el actual.
RF4.6	Árbol interactivo de posibles movimientos	Mediante esta herramienta, el jugador podrá estudia las recomendaciones e información producida por la IA para sus movimientos disponibles. Esto será posible mediante una serie de opciones y herramientas. Esta herramienta solo estará disponible en las partidas de entrenamiento, mientras la opción de recibir recomendaciones se encuentre activada.
RF4.6.1	Tabla de recomendaciones	Aquí, el jugador podrá ver el movimiento que, según la IA, sería el más beneficioso, además de aquellos movimientos más peligrosos o perjudiciales. Todos ellos irán acompañados de una explicación textual, y una opción que permitirá ver el posible desarrollo de la partida a partir de ellos.
RF4.6.2	Consultas sobre el árbol	Se le ofrecerá al jugador una serie de consultas para encontrar caminos o nodos con ciertas características que puedan ser de interés. También podrá plegar y expandir los hijos de cada nodo a su antojo.
RF4.6.3	Ventana de ayuda	Debido a la complejidad de este sistema, al usuario se le ofrecerá un apartado de ayuda, donde se explicarán las diferentes opciones disponibles y su utilidad.
RF4.6.4	Realizar movimiento	El jugador también podrá tomar una decisión desde el árbol interactivo, seleccionando el movimiento que quiera llevar a cabo.
RF4.7	Menú entre pantallas	Esto permitirá al usuario cambiar en cualquier momento, y sin afectar a la partida, entre el tablero interactivo, el historial y el árbol interactivo.
RF4.8	Partida finalizada	Una vez el servidor haya decidido que una partida ha llegado a su final, a los jugadores se les mostrará un mensaje diciendo si han ganado o perdido y, en caso de estar jugando una clasificatoria, la

		diferencia positiva o negativa de puntuación que se ha aplicado a su cuenta.

Código	Nombre Requisito	Descripción del Requisito
RF5	Lógica de partidas	El servidor deberá implementar toda la lógica necesaria para que los requisitos previos funcionen como se espera.
RF5.1	Emparejamiento	El servidor será capaz de emparejar a dos jugadores humanos que busquen una partida del mismo tipo.
RF5.2	Turnos	El servidor debe saber, en todo momento, a qué jugador le toca mover, siguiendo las reglas del juego. Esta lógica debe funcionar tanto para partidas multijugador, como para partidas de entrenamiento.
RF5.3	Partida acabada	El servidor debe ser capaz de decidir cuándo una partida ha acabado, y ejecutar la lógica correspondiente (por ejemplo, la asignación de puntos en partidas clasificatorias). Habrá varios supuestos que el servidor podrá detectar.
RF5.3.1	Tablero completo	Si no es posible realizar más movimientos, gana el jugador con una puntuación mayor.
RF5.3.2	Conexión cerrada	El servidor debe poder detectar cuándo un jugador ha cerrado una conexión (*i.e.*, ha cerrado el navegador), dándolo como perdedor.
RF5.3.3	Partida abandonada	El servidor también sabrá si un jugador ha abandonado una partida empezada, de nuevo asignándole la derrota.
RF5.3.4	Trampas	Para evitar potenciales usuarios con malas intenciones que puedan utilizar programas externos para enviar peticiones ajenas al cliente *web* y conseguir una ventaja, el servidor podrá detectar cuándo alguien intenta realizar un movimiento no permitido, o mueve fuera de su turno. Si esto sucede, el jugador que ha intentado hacer trampas será el perdedor.
RF5.3.5	Prohibidas partidas simultáneas	No sé permitirá que un usuario juegue varias partidas a la vez. Esto solo es necesario para las partidas multijugador.
RF5.4	Guardar juego	Con la finalidad de construir un conjunto de datos con el que poder seguir entrenando la IA en un futuro, el servidor guardará en la base de datos la secuencia de movimientos de todas las partidas jugadas, tanto multijugador como de entrenamiento.
RF6	Configuración	Es necesario que se establezca un archivo de configuración, desde el cuál se podrán modificar una serie de parámetros mínimos sin recompilar el software para el servidor.
RF6.1	Dificultad	Habrá un parámetro numérico que permitirá modificar la habilidad del rival virtual por cada nivel

		de dificultad.
RF6.2	Puntuaciones	Varios parámetros relativos al cálculo de las puntuaciones deberán poder configurarse desde este archivo: puntuación máxima y mínima por partida, puntuación mínima para una cuenta y una cantidad numérica que represente cuánto afecta la diferencia de puntuaciones de ambos jugadores a la puntuación calculada al final de una partida.
RF6.3	Tiempos	Desde este archivo también será configurable el límite de tiempo que los jugadores tienen tanto por movimiento como por partida.

5.2.1.2 Requisitos No Funcionales

Código	Nombre Requisito	Descripción del Requisito
RNF1	Base de datos	Esta deberá estar desplegada en una nube.
RNF2	Idioma del cliente	Los textos y explicaciones que aparezcan en la interfaz de usuario deberán estar en inglés.
RNF3	Comunicación entre cliente y servidor	Para la comunicación, deberá emplearse la tecnología WebSockets, al menos para la transmisión de información relativa a las partidas.
RNF4	Encriptación	Las contraseñas se enviarán encriptadas a la base de datos.
RNF5	Sesión de usuarios	Por seguridad, un usuario no podrá iniciar más de una sesión a la vez, y estas sesiones caducarán tras un periodo de inactividad.
RNF6	Multiplataforma	La aplicación *web* debe ser capaz de servirse desde un entorno Linux o Windows.
RNF7	Navegadores	Los usuarios deben poder acceder a la aplicación *web* desde todos los navegadores principales.
RNF8	Interfaz	La interfaz de usuario de la aplicación debe ser sencilla y fácil de usar.

5.2.2 Identificación de Actores del Sistema

- Algoritmo inteligente **[1]**: este sistema cumple numerosas funciones en nuestro proyecto: genera los datos necesarios para producir las recomendaciones que el usuario estudiará, produce los movimientos del rival virtual y funciona como motor de juego, en el sentido de que también se usa para verificar si un estado de tablero es alcanzable desde el estado actual (esto le sirve al servidor para comprobar si un movimiento es reglamentario).
- Usuario no identificado: este será cualquier usuario que accede a la aplicación de navegador y aún no haya iniciado sesión. Sus únicas opciones son iniciar sesión con una cuenta ya existente, o registrarse, creando así un perfil nuevo.
- Usuario identificado: un usuario que ya ha iniciado sesión (ya sea tras registrarse, o empleando unas credenciales válidas).

- Administrador del sistema: este no interactuará directamente con la aplicación. Para modificar algún parámetro del servidor, simplemente acudirá al archivo de configuración, y para cualquier acción relacionada con la base de datos, empleará el cliente *web* de MongoDB.

5.2.3 Especificación de Casos de Uso

En esta sección se describirá el comportamiento del sistema, utilizando para ello casos de uso. Estos casos de uso aparecerán, primero, en forma de diagrama, para más tarde ser descritos textualmente.

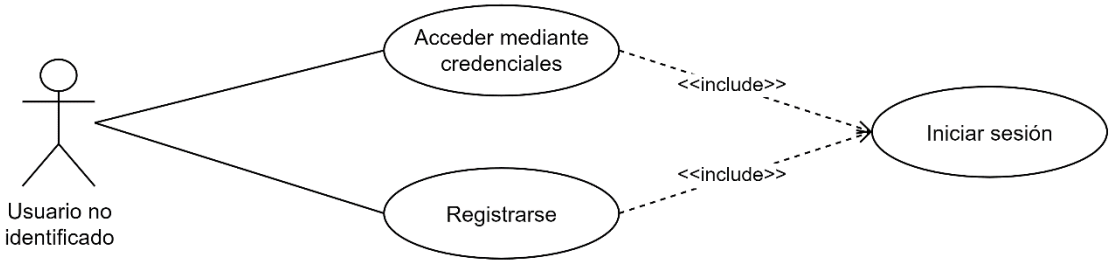


Figura 5.1. Casos de uso del usuario no identificado

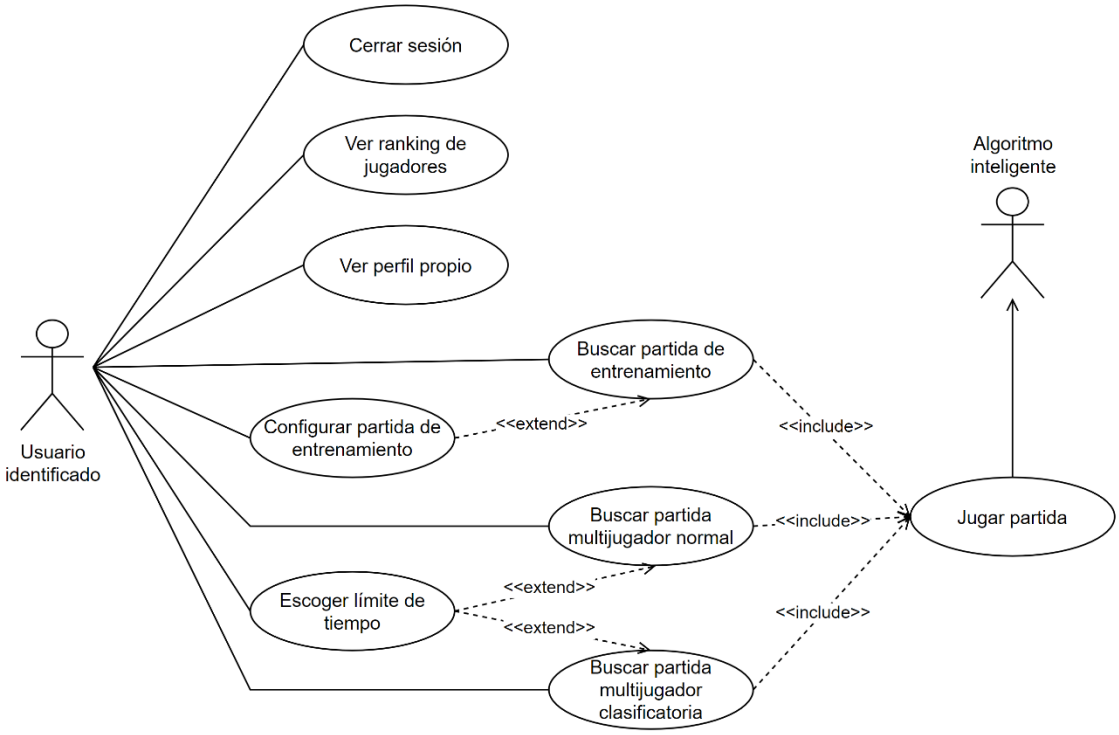


Figura 5.2. Casos de uso del usuario identificado

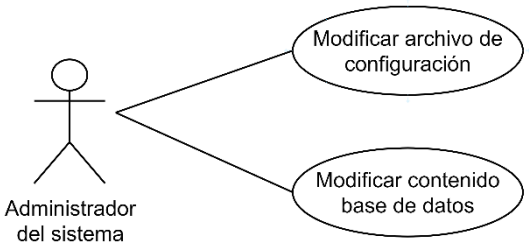


Figura 5.3. Casos de uso del administrador del sistema

A continuación, se describirá brevemente el objetivo de cada uno de los casos de uso que aparecen en las Figuras 5.1, 5.2 y 5.3.

Nombre del Caso de Uso
Acceder mediante credenciales (*Log In*)
Descripción
Si el usuario no identificado dispone de un perfil ya creado, podrá iniciar sesión en la aplicación *web* con sus credenciales: email y contraseña. En caso de que estas superen las comprobaciones y el servidor confirme que existen en el sistema, se iniciará una sesión de su cuenta en la aplicación de navegador.

Nombre del Caso de Uso
Registrase
Descripción
Un usuario no identificado, que no tiene una cuenta en el sistema, podrá registrarse para así crear un nuevo perfil e iniciar sesión. Para ello deberá proporcionar un nombre de usuario, una contraseña y un email. Si estos datos llegan al servidor y superan las comprobaciones pertinentes, se creará esa nueva cuenta de usuario y se iniciará sesión en la aplicación de navegador.

Nombre del Caso de Uso
Iniciar sesión
Descripción
Cuando un usuario no identificado consigue iniciar sesión con una cuenta de la aplicación, por cualquier de las dos vías disponibles, pasará a poder realizar todas las acciones propias de un usuario identificado.

Nombre del Caso de Uso
Cerrar sesión
Descripción
Un usuario identificado podrá, en el momento que desee, cerrar la sesión de su cuenta. Con esto pasará a tener solo las funcionalidades disponibles para un usuario no identificado.

Nombre del Caso de Uso
Ver *ranking* de jugadores
Descripción

Cualquier usuario identificado podrá ver una lista de todos los jugadores de la aplicación, ordenada por puntuación en orden decreciente, y dividida en páginas. En esta lista aparecerá, por cada jugador, el número que ocupa en el *ranking*, su nombre de usuario y su puntuación total.

Nombre del Caso de Uso
Ver perfil propio
Descripción
Los usuarios identificados podrán, desde el menú principal de la aplicación *web*, inspeccionar la información de la cuenta con la que hayan iniciado sesión. Esta información incluirá su nombre de usuario, el email vinculado a la cuenta y su puntuación total.

Nombre del Caso de Uso
Buscar partida de entrenamiento
Descripción
Un usuario identificado podrá solicitar al servidor una partida de entrenamiento contra el rival virtual. El servidor creará la partida y se encargará de toda la lógica posterior.

Nombre del Caso de Uso
Configurar partida de entrenamiento
Descripción
Antes de buscar una partida de entrenamiento, el usuario identificado podrá configurar para ella una serie de parámetros como quien juega primero, la dificultad del rival virtual y si quiere o no recibir recomendaciones.

Nombre del Caso de Uso
Buscar partida multijugador normal
Descripción
Los usuarios identificados podrán solicitar al servidor una partida multijugador normal. El servidor emparejará a aquellos que busquen el mismo tipo de partida, creará una para ellos, y controlará la lógica posterior.

Nombre del Caso de Uso
Buscar partida multijugador clasificatoria
Descripción
De forma similar, un usuario identificado tendrá la opción de empezar una partida multijugador clasificatoria. De nuevo, el servidor lo emparejará con alguien que busque una partida de igual tipo y asegurará el funcionamiento de la lógica posterior.

Nombre del Caso de Uso
Escoger límite de tiempo
Descripción
Antes de buscar una partida multijugador, ya sea normal o clasificatoria, el usuario identificado escogerá el tipo de partida que desea en función de la limitación de tiempo: tiempo ilimitado, límite por partida y jugador, o límite por movimiento.

Nombre del Caso de Uso
Jugar partida
Descripción
Tras solicitar una partida, del tipo que sea, y haber sido emparejado con un rival, el usuario identificado podrá jugarla. Esto será más o menos similar en cualquier partida, independientemente de si es de entrenamiento o multijugador. Para ciertos aspectos de la lógica de las partidas, las recomendaciones y los movimientos del rival virtual, el servidor recurrirá al algoritmo inteligente ya mencionado.

Nombre del Caso de Uso
Modificar archivo de configuración
Descripción
Para ello cambiar los parámetros configurables del servidor, el administrador del sistema simplemente tendrá que localizar el archivo, abrirlo con cualquier editor de texto y modificar los valores que sean necesarios. Para que algunas de estas modificaciones se apliquen, además, puede ser necesario reiniciar el servidor.

Nombre del Caso de Uso
Modificar el contenido de la base de datos
Descripción
Gracias a emplear una base de datos MongoDB alojada en la nube de la compañía, el administrador del sistema solo tendrá que acceder con las credenciales correctas a la interfaz *web* que MongoDB ofrece.

5.3 Identificación de los Subsistemas en la Fase de Análisis

En esta sección se presentará el sistema, descompuesto en varios subsistemas más que, posteriormente, servirán para facilitar su análisis.

5.3.1 Descripción de los Subsistemas

La arquitectura de la aplicación es similar al patrón de arquitectura modelo-vista-controlador. La vista corresponde con la interfaz *web* desarrollada en Angular. Por otra parte, tanto el modelo como el controlador estarían implementados en un mismo proyecto Java.

Los subsistemas identificados en esta parte del proceso de desarrollo son los siguientes:

- La interfaz de usuario para navegador, que será el subsistema con el que interactúen directamente los usuarios de esta aplicación, tanto para ver datos como para acceder a todas las funcionalidades implementadas. Este subsistema se encuentra, a su vez, dividido en numerosos módulos, implementando cada uno una pantalla o funcionalidad distinta.
- El subsistema encargado de la comunicación con el cliente y la lógica de usuarios. Este se encargará de la comunicación con la interfaz de usuario, siendo el único que reciba y envíe información a la misma. Además, también será el responsable de toda la lógica de inicio de sesión, registro y autorización.
- El subsistema de la lógica interna del juego, que implementará todo lo relativo a los distintos tipos de partidas y su funcionamiento (reglas de juego, puntuaciones, límites de tiempo, lógica de victoria y derrota, etc.). Este será, además, el que empleará llamadas al sistema de inteligencia artificial. Este último sistema inteligente no será descrito en este apartado ya que ya está implementado y su desarrollo no corresponde a este proyecto.
- Subsistema de base de datos. Este subsistema servirá como puente de comunicación entre la base de datos propiamente dicha y el resto de subsistemas de la aplicación.

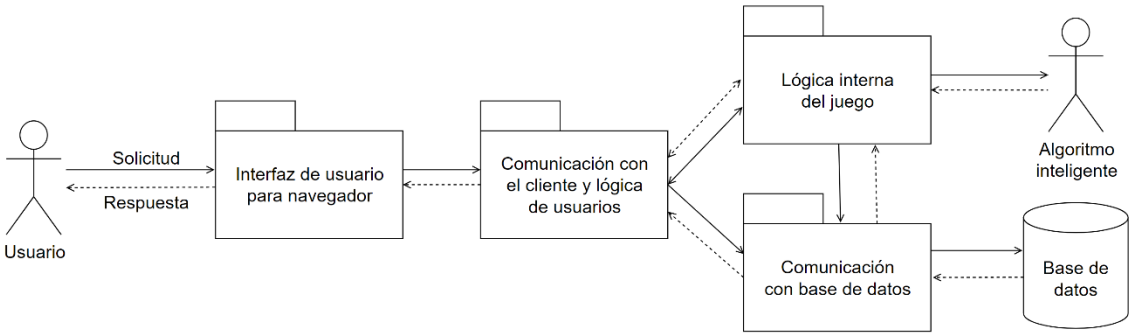


Figura 5.4. Diagrama de los subsistemas identificados

5.3.2 Descripción de los Interfaces entre Subsistemas

La comunicación entre la interfaz de usuario y el subsistema de comunicación con el cliente y lógica de usuarios será a través de red, empleando los protocolos HTTP y WebSockets (*ws:*), según sean más convenientes para cada tarea concreta. La información se trasferirá siempre utilizando JSON como formato de texto.

La comunicación entre el subsistema implementado en Java de comunicación con base de datos y la propia base de datos también se dará por red. A pesar de esto, desde la perspectiva del desarrollo, será como hacer llamadas locales, ya que se empleará la librería que ofrece MongoDB para la comunicación con la base de datos, y esta librería permite utilizar una interfaz basada en métodos que oculta al programador las comunicaciones subyacentes. Para ello, se implementarán, dentro del subsistema de comunicación con base de datos, métodos capaces de traducir las entidades Java a BSON, el formato de texto que emplea MongoDB para sus documentos.

Las comunicaciones entre el resto de subsistemas que se pueden observar en la Figura 5.4 (incluida la comunicación entre el subsistema de lógica de juego y el algoritmo inteligente), sucederán siempre de manera local (y dentro del mismo programa, además), a través de las interfaces concretas que cada uno de ellos implemente.

5.4 Diagrama de Clases Preliminar del Análisis

En este apartado se concretarán las clases identificadas hasta el momento. Estas clases no tienen por qué aparecer implementadas en el producto final, sufrirán modificaciones a lo largo del proceso y, durante el desarrollo, irán apareciendo más clases de las documentadas aquí.

5.4.1 Diagrama de Clases

A continuación, se mostrarán dos diagramas en los que será posible ver todas las clases identificadas en la fase de análisis del proyecto, junto con algunos de sus métodos posibles. No se ha querido entrar en profundidad en métodos y atributos de clases ya que es algo complicado de establecer en esta fase, por lo que solo aparecerán los métodos y atributos que serán implementados con casi total seguridad (aunque puede que sus nombres varíen).

En el primer diagrama (Fig. 5.5) se pueden observar todas las clases identificadas para el subsistema de la interfaz de usuario *web* (como será desarrollado en Angular, técnicamente son componentes, no clases, pero son conceptos similares), mientras que en el segundo (Fig. 5.6) aparecen las clases del resto de subsistemas.

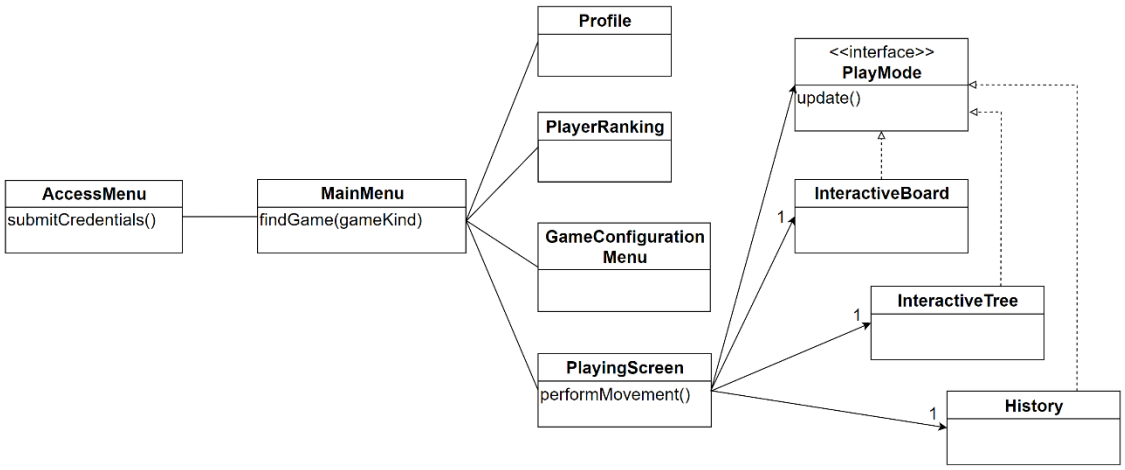


Figura 5.5. Diagrama de clases preliminar de la interfaz de usuario

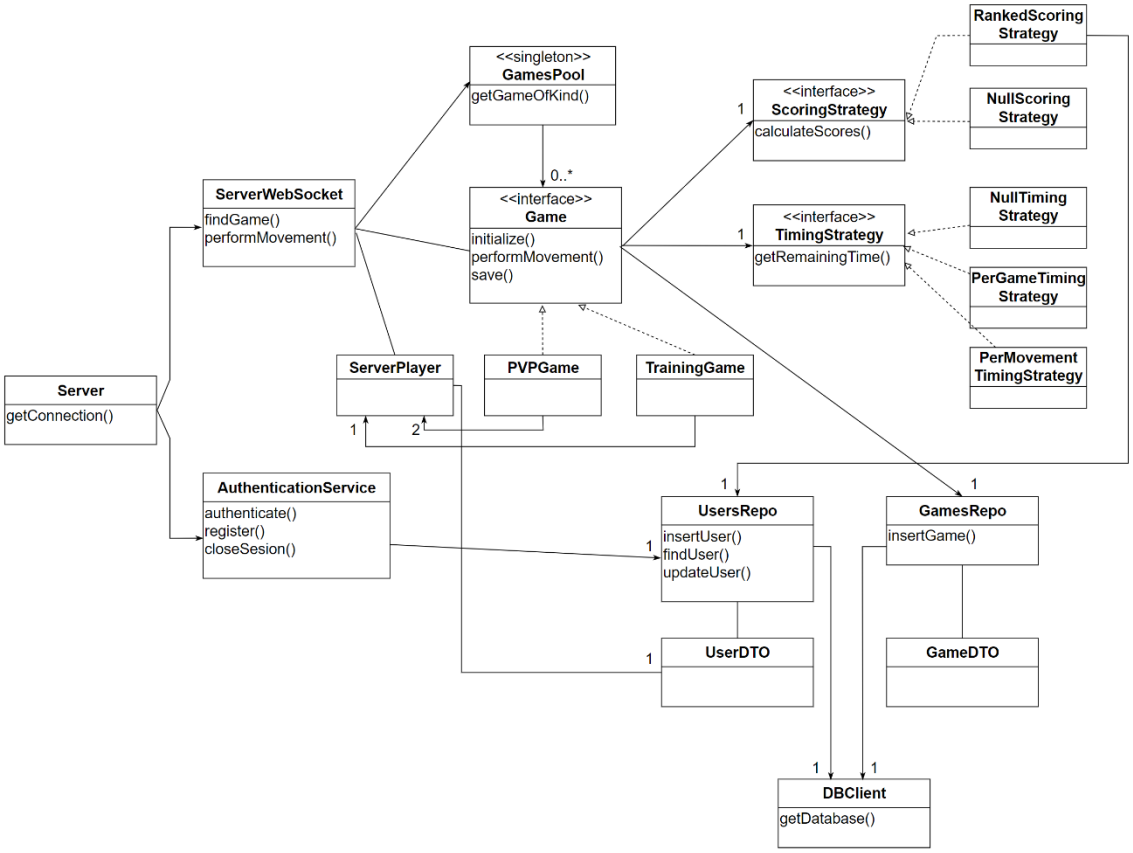


Figura 5.6. Diagrama de clases preliminar para el resto del sistema

5.4.2 Descripción de las Clases

En los diagramas anteriores, con ánimo de ahorrar espacio, solo se han incluido aquellos métodos clave para entender la estructura de clases. Ahora, estas clases serán detalladas con más profundidad, incluyendo sus funciones, responsabilidades, métodos y atributos que se han podido identificar, divididas según el subsistema al que pertenezcan.

5.4.2.1 Interfaz de Usuario para Navegador

Nombre de la Clase
AccessMenu
Descripción
Será la primera pantalla que se encuentre un usuario no identificado al entrar en la aplicación *web*, desde ahí podrá identificarse mediante credenciales de una cuenta preexistente, o registrarse y crear un nuevo perfil.
Responsabilidades
Este componente deberá ofrecer al usuario los formularios, cada uno con sus campos correspondientes, para el inicio de sesión y el registro. Además, será quien lleve a cabo las validaciones implementadas en el lado del cliente. Cuando el usuario lo decida, y si los datos han superado la validación, enviará estos al servidor. Si el servidor da una respuesta negativa,

o si la validación previa falla, mostrará al usuario un mensaje de error, y, en caso contrario, se encargará de que se muestre la siguiente pantalla, el menú principal (MainMenu).
Métodos Propuestos
SubmitCredentials: Este será el método encargado de establecer conexión con el servidor y enviarle todos los datos necesarios. Posiblemente en la implementación final acabe dividido en dos: uno para inicio de sesión con credenciales y otro para registrar una nueva cuenta.

Nombre de la Clase
MainMenu
Descripción
Funcionará como menú principal de la aplicación, desde el cual un usuario identificado podrá acceder a todas las funcionalidades posibles: configurar y buscar partida, ver *ranking* de jugadores, ver perfil propio, cerrar sesión, etc.
Responsabilidades
Este componente será quien maneje la lógica de qué pantalla se enseña al usuario en cada momento y servirá como componente padre para el resto de la aplicación. Además, será el encargado de establecer una conexión WebSockets con el servidor para poder iniciar la búsqueda de partida, enviándole todos los parámetros necesarios, incluido el *token* de acceso para la sesión. También será el encargado de establecer contacto con el servidor para recuperar datos sobre el usuario y cerrar sesión. Además, será quien muestre los mensajes de error pertinentes en caso de que una de estas comunicaciones falle por algún motivo.
Atributos Propuestos
SessionToken: Será el *token* de sesión que el servidor habrá enviado tras un inicio de sesión satisfactorio. Deberá emplearse para todas aquellas peticiones al servidor que requieran una sesión iniciada de un usuario autenticado (como buscar cualquier tipo de partida, por ejemplo). **User**: Este atributo guardará toda la información necesaria sobre el usuario cuya sesión está iniciada: nombre, email vinculado y puntuación.
Métodos Propuestos
FindGame: Este método establecerá una conexión WebSockets con el servidor, añadiendo todos los parámetros necesarios, como el tipo de partida o el *token* de sesión. **CloseSession**: Mediante este método se podrá avisar al servidor de que cierre la sesión actual para el usuario identificado.

Nombre de la Clase
Profile
Descripción
Esta pantalla podrá ser accedida en cualquier momento desde el menú principal para ver los datos del usuario con el que se ha accedido a la aplicación: nombre, email y puntuación.
Responsabilidades
Las responsabilidades de este componente serán bastante limitadas, únicamente tendrá que ser capaz de mostrar información cuando se abra, y cerrarse.
Atributos Propuestos
User: El componente deberá tener la información del usuario almacenada en un atributo para así poder mostrarla en cualquier momento.
Métodos Propuestos
CloseProfile: La única función de este método será que el perfil pueda cerrarse a sí mismo, volviendo al menú principal.

Nombre de la Clase
PlayerRanking
Descripción
Este componente implementará una pantalla de la aplicación *web* desde la que se podrá ver un *ranking*, ordenado por puntuación de forma descendente, de todos los jugadores. Además, este *ranking* se mostrará por páginas de un máximo de diez usuarios cada una.
Responsabilidades
PlayerRanking deberá implementar toda la lógica necesaria para mostrar esta lista paginada de jugadores, incluyendo su posición en el *ranking*, nombre de usuario y puntuación de clasificaria. También será quien establezca la conexión necesaria con el servidor para solicitar esta información.
Atributos Propuestos
PageMaxSize: Este será un número representando el tamaño máximo de cada página (el número de usuarios que aparecerán en ella). **PageNumber**: Aquí se guardará el número de página actual, de la página que se está mostrando. **PlayersInPage**: Este atributo representa una lista de los jugadores (y su información) que se mostrarán en la página actual.
Métodos Propuestos
GetPagedPlayers: Mediante este método, el componente podrá conectarse con el servidor para pedir la lista de jugadores correspondiente a la página que quiera mostrar, indicándole también el tamaño máximo de página y el número de página actual. **CalculateRankingPosition**: Este método será utilizado para calcular la posición en el *ranking* de cada usuario, esto se hará a partir del número de página, el tamaño máximo de página y la posición del jugador en el atributo PlayersInPage. **CloseRanking**: Un método que simplemente permitirá al PlayerRanking ser capaz de cerrarse a sí mismo, volviendo a mostrar el menú principal.

Nombre de la Clase
GameConfigurationMenu
Descripción
Representará un menú de configuración de todos los parámetros que el usuario identificado pueda elegir o modificar previamente a una partida. Posiblemente esta clase de desdoble en varias en la implementación final, dependiendo de las necesidades de cada tipo de partida.
Responsabilidades
Además de implementar una interfaz gráfica para modificar y elegir estos atributos configurables, este componente deberá ser capaz de guardar las preferencias del usuario y transmitirlas a la clase encargada de la conexión en el momento de buscar partida.
Atributos Propuestos
Configuration: Este atributo pretende almacenar todas las opciones que el jugador haya escogido. En la implementación final, aparecerá, posiblemente, como varios atributos: uno para cada parámetro configurable.
Métodos Propuestos
GetConfiguration: Permitirá a la clase encargada de la comunicación con el servidor extraer todos los valores elegidos para las distintas configuraciones.

Nombre de la Clase
PlayingScreen
Descripción
Esta es la pantalla que aparecerá cuando el usuario identificado entre en una partida. Desde aquí podrá realizar sus movimientos y acceder a todas las funcionalidades disponibles mientras se juega una partida: tablero interactivo, historial de movimientos y árbol interactivo.
Responsabilidades
Este componente funcionará como padre del tablero, historial y árbol, decidiendo cuál mostrar en cada momento. Para ello, implementará un pequeño menú en el cual el jugador podrá modificar la vista actual. Se encargará también de transmitir y recibir información sobre la partida del servidor. Tendrá que ser capaz de enviar los movimientos del jugador, y recibir y distribuir entre sus componentes hijos las actualizaciones que envíe el servidor. Además, deberá mostrar de alguna forma la siguiente información sobre la partida: el tiempo restante, los nombres de ambos jugadores y el número de cajas que ha cerrado cada uno.
Atributos Propuestos
Board: Una instancia de la clase InteractiveBoard. **Tree**: Una instancia de la clase InteractiveTree. **History**: Una instancia de la clase History. **BoardData**: La información recibida del servidor que codifica el estado actual del tablero y deberá transmitir al InteractiveBoard. **TreeData**: La información recibida del servidor con los datos que debe contener el árbol interactivo, y que deberá transmitir a este. **HistoryData**: La información recibida del servidor y/o guardada por él mismo que codificará el contenido del historial y que debe pasarle cuando lo haga visible. **P1Username**: El nombre de usuario del jugador 1. **P2Username**: El nombre de usuario del jugador 2. **P1Score**: El número de cajas cerradas en esta partida por el jugador 1. **P2Score**: El número de cajas cerradas en esta partida por el jugador 2. **RemainingTime**: Un valor numérico que representará el tiempo que le queda al jugador, ya sea para un movimiento o para la partida.
Métodos Propuestos
PerformMovement: Este método podrá usarse para enviar al servidor el movimiento que ha realizado el jugador identificado. **LeaveGame**: Servirá para notificar al servidor de que el jugador ha decidido abandonar la partida (ya sea a través de un botón o cerrando el navegador).

Nombre de la Clase
PlayMode
Descripción
Una interfaz que representará a los distintos tipos de modos se visualizar la información que el usuario tiene mientras juega (tablero, árbol e historial).
Responsabilidades
Esta interfaz simplemente servirá para establecer los métodos que deberán implementar estos otros componentes. La clase PlayingScreen deberá invocar solo aquellos métodos definidos en esta interfaz.
Métodos Propuestos
Update: Este método servirá para que el componente padre actualice la información que se debe representar.

Nombre de la Clase
InteractiveBoard
Descripción
Un tablero interactivo en el cual el usuario podrá visualizar el estado actual de la partida y escoger el movimiento que desea realizar.
Responsabilidades
Este componente debe ser capaz de representar en forma de tablero de *Dots and Boxes* la información que reciba (siempre que sea correcta). También ha de poder detectar si el jugador ha realizado un movimiento (posiblemente sea pinchando encima de la línea que se quiere dibujar), y se los transmitirá a la clase PlayingScreen.
Atributos Propuestos
BoardData: La información recibida de su componente padre que codifica el estado actual del tablero y que deberá representar.

Nombre de la Clase
InteractiveTree
Descripción
El árbol interactivo desde el que el jugador podrá explorar y estudiar las recomendaciones e informaciones disponibles. A parte de eso, también podrá realizar un movimiento desde él.
Responsabilidades
Será capaz de representar en forma de árbol la información que reciba (siempre que el formato sea correcto). Además, deberá permitir al usuario plegar y expandir los hijos de cualquier nodo, realizar consultas, y escoger un movimiento.
Atributos Propuestos
TreeData: La información recibida de su componente padre que codifica el estado actual del árbol y que deberá representar.

Nombre de la Clase
History
Descripción
Este componente mostrará, de forma ordenada, todos los movimientos que han realizado a lo largo de la partida tanto el jugador identificado como su rival.
Responsabilidades
Siempre que sea en un formato correcto, será capaz de representar de forma adecuada la información que reciba de la clase PlayingScreen.
Atributos Propuestos
HistoryData: La información recibida de su componente padre que codifica el estado actual del historial y que deberá representar.

5.4.2.2 Subsistema de Comunicación con Cliente y Lógica de Usuarios

Nombre de la Clase
Server
Descripción
Representa la entrada de la aplicación que se desplegará sobre un puerto abierto de la máquina empleada como servidor. Será quien se comunique con los distintos clientes, a través de los protocolos HTTP y WebSockets.
Responsabilidades
Deberá ser quien actúe de intermediario entre los clientes y el resto del sistema *backend*. Podrá de recibir peticiones de clientes a través de la red, redirigiéndolas a las partes adecuadas del sistema que vayan a tratar esas peticiones. También deberá ser capaz de hacer fluir la información en sentido inverso.
Atributos Propuestos
Servlets: Este atributo representa a todos los servicios que se implementarán en el servidor para atender a los distintos tipos de peticiones que podrán hacer los clientes.
Métodos Propuestos
GetConnection: Mediante este método se solicitarán los distintos tipos de conexiones HTTP y WebSockets.

Nombre de la Clase
ServerWebSocket
Descripción
Esta clase será a quien el servidor principal delegue la comunicación WebSockets con el cliente. Su misión principal será servir como canal de comunicación entre el cliente y la lógica de juego.
Responsabilidades
Deberá implementar todo lo necesario para transmitir la información del cliente sobre una partida (configuración, movimientos realizados, etc.) al motor del juego. También se encargará de la comunicación desde el motor hacia el cliente. Además, será el responsable de implementar la lógica para emparejar a aquellos jugadores que busquen el mismo tipo de partida.
Atributos Propuestos
GamesPool: Una instancia (la única, ya que deberá ser implementado como *singleton*) de la clase con mismo nombre.
Métodos Propuestos
FindGame: Será el método a través del cual el cliente solicitará una conexión a una nueva partida, especificando el tipo de partida que busca. **PerformMovement**: Mediante este método, el WebSocket podrá recibir los movimientos que un jugador haya realizado. Estos movimientos serán después transferidos al subsistema de lógica de juego. **SendResponseToClient**: Este método será el opuesto al anterior, el que el subsistema que implementa la lógica del juego empleará para enviar toda la información necesaria al cliente correspondiente. **CloseConnection**: La única función que tendrá será la de permitir que, tanto el cliente como el subsistema de la lógica, cierren una conexión previamente establecida.

Nombre de la Clase
AuthenticationService
Descripción
La clase que se encargará de todas las peticiones del cliente relacionadas con identificación, registro, obtención de *token*, etc.
Responsabilidades
Entre otras, deberá ser capaz de validar credenciales, comprobar el registro de nuevos usuarios y añadirlos a la base de datos, generar *tokens* para los usuarios identificados y mantener una colección de aquellos *tokens* activos, etc. Debido a esta gran cantidad de funcionalidades, y las que posiblemente aparezcan más tarde, AuthenticationService se dividirá en varias clases en el proceso de implementación.
Atributos Propuestos
Tokens: Una colección que contendrá todos los *tokens* activos de usuarios con sesión iniciada. **UsersRepo**: Una instancia de la clase con igual nombre que le servirá para añadir entradas a la base de datos y comprobar credenciales.
Métodos Propuestos
Authenticate: Implementará la lógica necesaria para recibir las peticiones de aquellos usuarios que quieran iniciar sesión con las credenciales de una cuenta ya existente. **Register**: De forma similar al anterior, será el método que gestione las peticiones de usuarios que busquen crear una nueva cuenta en el sistema. **CloseSesion**: Método que servirá para avisar de que una sesión ha acabado, por lo que el token deberá ser invalidado.

5.4.2.3 Subsistema de Lógica Interna del Juego

Nombre de la Clase
GamesPool
Descripción
Guardará una colección de partidas que estará disponible para todas las conexiones con el servidor.
Responsabilidades
Para poder emparejar a los jugadores, las distintas conexiones y procesos del sistema han de tener acceso a la misma colección de partidas, por eso esta clase ha de ser implementada como *singleton* para envolver esa colección.
Atributos Propuestos
GamesPoolInstance: Como es un *singleton*, este atributo guardará la única instancia de esta clase en tiempo de ejecución. **Games**: Una colección en la que se almacenarán todas las partidas en proceso en el servidor. Dependiendo de las necesidades específicas que se descubran más tarde, esta podrá ser implementada como una lista, mapa, set, etc.
Métodos Propuestos
GetInstance: Permitirá obtener la única instancia de esta clase desde cualquier punto del sistema. **AddGame**: Servirá para añadir una nueva partida a la colección. **FindGame**: Método que se podrá utilizar para obtener partidas en la colección. Habrá una implementación para buscar partidas según su tipo, pero pueden existir más métodos de este tipo si son necesarios. **RemoveGame**: Para eliminar una partida de esta colección.

Nombre de la Clase
Game
Descripción
Una interfaz que englobará a los distintos tipos de partidas.
Responsabilidades
Esta interfaz simplemente servirá para establecer los métodos que deberán implementar las distintas clases partida. El resto de elementos del sistema que recurran a este tipo de clases, deberán hacerlo siempre empleando los métodos establecidos en esta interfaz.
Métodos Propuestos
Initialize: Servirá para empezar todos los procesos necesarios para la lógica de las partidas una vez la partida cuente con su número máximo de jugadores. **PerformMovement**: Este será el método que empleará la clase ServerWebSocket para informarle del movimiento que un jugador ha llevado a cabo. **Save**: Una vez una partida haya acabado, este método servirá para guardarla en la base de datos. **PlayerLost**: Este será utilizado por las clases ajenas a las distintas partidas para notificar que un jugador ha perdido por alguna razón externa a la propia lógica del juego (ha cerrado la conexión, se le ha acabado el tiempo, etc.).

Nombre de la Clase
PVPGame
Descripción
Esta clase representará y contendrá la lógica de las partidas multijugador. Implementará la interfaz Game, por lo que contendrá sus mismos métodos.
Responsabilidades
Implementar toda la lógica necesaria para que se desarrollen las partidas multijugador: turnos, comprobaciones, lógica de derrota, etc.
Atributos Propuestos
Player1: Este atributo contendrá toda la información que esta clase necesite sobre el jugador 1 y su cuenta. **Player2**: Este atributo contendrá toda la información que esta clase necesite sobre el jugador 2 y su cuenta.

Nombre de la Clase
TrainingGame
Descripción
Esta clase representará y contendrá la lógica de las partidas de entrenamiento. Implementará la interfaz Game, por lo que contendrá sus mismos métodos.
Responsabilidades
Implementar toda la lógica necesaria para que se desarrollen las partidas de entrenamiento: turnos, comprobaciones, movimientos del rival virtual, recomendaciones, etc.
Atributos Propuestos
Player1: Este atributo contendrá toda la información que esta clase necesite sobre su único jugador humano y su cuenta.

Nombre de la Clase
ServerPlayer
Descripción
Esta clase representará a cada usuario humano que se encuentre jugando una partida en el sistema.
Responsabilidades
Únicamente guardará datos sobre este usuario, para que estén disponibles para todos aquellos subsistemas que los necesiten a lo largo del flujo de vida de una partida.
Atributos Propuestos
Session: Este atributo contendrá alguna forma de identificar la sesión WebSockets con la que el usuario está conectado al servidor. **User**: Una instancia de la clase UserDTO que contendrá todos los datos relativos a la cuenta del jugador.

Nombre de la Clase
ScoringStrategy
Descripción
Una interfaz que englobará a las distintas estrategias que tendrán las partidas para calcular las puntuaciones al final de las mismas.
Responsabilidades
Esta interfaz simplemente servirá para establecer los métodos que deberán implementar las distintas estrategias de cálculo de puntuaciones. Las diferentes clases partida, cuando invoquen alguna de estas estrategias, tendrán que hacerlo a través de los métodos presentes en esta interfaz.
Métodos Propuestos
CalculateScores: Este método servirá para calcular los puntos que se sumarán y restarán a los jugadores al final de una partida.

Nombre de la Clase
RankedScoringStrategy
Descripción
Una implementación de estrategia de cálculo de puntuaciones que tiene en cuenta la diferencia que haya entre ambos jugadores para decidir cuántos puntos se suman o se restan.
Responsabilidades
Deberá calcular las nuevas puntuaciones de cada jugador al final de una partida teniendo en cuenta la diferencia existente entre ellos. Además, deberá actualizar estas puntuaciones en la base de datos.
Atributos Propuestos
UsersRepo: Una instancia de la clase de igual nombre que le servirá para actualizar las puntuaciones de los jugadores en la base de datos.

Nombre de la Clase
NullScoringStrategy
Descripción
Una implementación de estrategia de cálculo de puntuaciones que se empleará en aquellas partidas que no deban afectar a las puntuaciones de los jugadores.
Responsabilidades
Simplemente absorberá las peticiones que reciba, sin dar ningún error, pero sin modificar las puntuaciones de los jugadores.

Nombre de la Clase
TimingStrategy
Descripción
Una interfaz que englobará a las distintas estrategias que tendrán las partidas para calcular el tiempo restante del que dispone cada jugador
Responsabilidades
Esta interfaz simplemente servirá para establecer los métodos que deberán implementar las distintas estrategias de cálculo de tiempo restante. Las diferentes clases partida, cuando invoquen alguna de estas estrategias, tendrán que hacerlo a través de los métodos presentes en esta interfaz.
Métodos Propuestos
GetRemainingTime: Permitirá obtener, en milisegundos, el tiempo que le queda a un jugador determinado. **MovementPerformed**: Permitirá avisar a la estrategia de que un jugador concreto ha realizado un movimiento.

Nombre de la Clase
PerGameTimingStrategy
Descripción
Una implementación de estrategia de cálculo de tiempos que servirá para aquellas partidas con tiempo limitado por partida y jugador.
Responsabilidades
Deberá mantener, en todo momento, el tiempo restante que le queda a cada uno de los jugadores de una partida, implementando la lógica necesaria para que el tiempo de cada jugador solo se consuma en su turno. Además, deberá ser capaz de restar tiempo al contador de cada jugador mientras sea su turno.
Atributos Propuestos
BaseTimePerGame: Atributo que almacena el tiempo total del que dispondrá cada jugador al iniciar una partida con este tipo de limitación. **Player1RemainingTime**: Mantendrá el tiempo restante disponible para que el jugador 1 realice sus movimientos. **Player2RemainingTime**: Mantendrá el tiempo restante disponible para que el jugador 2 realice sus movimientos.

Nombre de la Clase
PerMovementTimingStrategy
Descripción
Una implementación de estrategia de cálculo de tiempos que servirá para aquellas partidas con tiempo limitado para cada movimiento.
Responsabilidades
Implementará toda la lógica necesaria para saber, en caso de que se acabe el tiempo, de quién era el turno (qué jugador ha perdido). Por supuesto, deberá ser capaz de controlar cuándo ha pasado el tiempo establecido para un movimiento.
Atributos Propuestos
BaseTimePerMovement: Atributo que almacena el tiempo total del que dispondrán los jugadores para cada uno de sus movimientos.

Nombre de la Clase
NullTimingStrategy
Descripción
Una implementación de estrategia de cálculo de tiempos para las partidas en las que no deba existir límite de tiempo.
Responsabilidades
Simplemente absorberá las peticiones que reciba, sin dar ningún error, pero sin implementar ninguna cuenta atrás ni provocar la derrota de un jugador.

5.4.2.4 Subsistema de Base de Datos

Nombre de la Clase
UsersRepo
Descripción
Será la clase encargada de todas las operaciones de escritura, borrado y búsqueda de la base de datos que tengan que ver con usuarios.
Responsabilidades
Deberá ser capaz de recibir peticiones del tipo mencionado y traducirlas a una operación de la base de datos, utilizando la API de esta y el formato de intercambio de datos BSON.
Atributos Propuestos
DBClient: Una instancia de la clase con mismo nombre que servirá para obtener la conexión con la base de datos.
Métodos Propuestos
InserUser: Permitirá añadir nuevos usuarios a la base de datos. Útil, por ejemplo, cuando un jugador cree una cuenta nueva en la aplicación. **FindUser**: Un método que se podrá utilizar para buscar usuarios en la base de datos. En la implementación final, es muy posible que se convierta en varios métodos distintos (para buscar por email, nombre de usuario, identificador, etc.), según sea necesario. **UpdateUser**: Para añadir o modificar alguna de la información presente en un registro de usuario (para modificar su puntuación total, por ejemplo). **RemoveUser**: Método que permitirá eliminar usuarios de la base de datos.

Nombre de la Clase
GamesRepo
Descripción
Será la clase encargada de las operaciones de escritura de partidas (solo las secuencias de movimientos) en la base de datos, para más tarde ser potencialmente usadas para reentrenar la IA.
Responsabilidades
Deberá ser capaz de recibir peticiones del tipo mencionado y traducirlas a un a operación de la base de datos, utilizando la API de esta y el formato de intercambio de datos BSON.
Atributos Propuestos
DBClient: Una instancia de la clase con mismo nombre que servirá para obtener la conexión con la base de datos.
Métodos Propuestos
InsertGame: Permitirá añadir nuevas secuencias de movimientos a la base de datos. Para guardar una partida cuando esta finalice.

Nombre de la Clase
UserDTO
Descripción
Representará y transferirá la información de los usuarios de la base de datos entre las partes del sistema que lo requieran.
Responsabilidades
En principio, no implementará ningún tipo de lógica. Únicamente servirá como clase auxiliar para la transmisión de datos entre subsistemas.
Atributos Propuestos
Id: El identificador del usuario en la base de datos. **Username**: Nombre de usuario del jugador al que representa. **Email**: Email vinculado a la cuenta de este usuario. **Contraseña**: Contraseña, debidamente encriptada, de la cuenta de usuario. **Puntuación**: Número de puntos con los que cuenta el jugador.
Métodos Propuestos
Getters: Uno para cada atributo, para pedirle la información necesaria. **Setters**: De nuevo, uno por atributo. Para crear o modificar la instancia.

Nombre de la Clase
GameDTO
Descripción
Representará y transferirá la información de las partidas jugadas hacia el subsistema de base de datos.
Responsabilidades
En principio, no implementará ningún tipo de lógica. Únicamente servirá como clase auxiliar para la transmisión de datos entre subsistemas.
Atributos Propuestos
Movements: Una colección ordenada de movimientos que sirva para representar cómo se ha desarrollado la partida.
Métodos Propuestos
Getter: Para obtener esa colección de movimientos. Podrán aparecer más métodos de este tipo si es necesario. **Setter**: Para fijar la colección de movimientos. Podrán aparecer más métodos de este tipo si resulta necesario.

Nombre de la Clase
DBClient
Descripción
Esta clase implementará la conexión con la base de datos.
Responsabilidades
Deberá ser capaz de establecer conexiones con la base de datos MongoDB empleando la API correspondiente y manejar la lógica necesaria para ello.
Atributos Propuestos
MongoDBClient: Una instancia a la que solicitará las distintas conexiones con la base de datos.
Métodos Propuestos
GetDatabase: Permitirá que las clases ajenas a esta le soliciten una conexión con la base de datos.

5.5 Análisis de Casos de Uso y Escenarios

En esta sección se estudiarán más en detalle los casos de uso identificados previamente. Primero aparecerán dos diagramas de robustez que muestran cómo interactúan las distintas entidades, acciones e interfaces. El primero (Fig. 5.7) hace referencia a los escenarios derivados de los casos de uso relacionados con el acceso a la aplicación, mientras que el segundo (Fig. 5.8) muestra los que surgen de los casos de uso del usuario ya identificado.

A continuación, se explicarán más en detalle todos estos escenarios, haciendo uso de un formato de tabla.

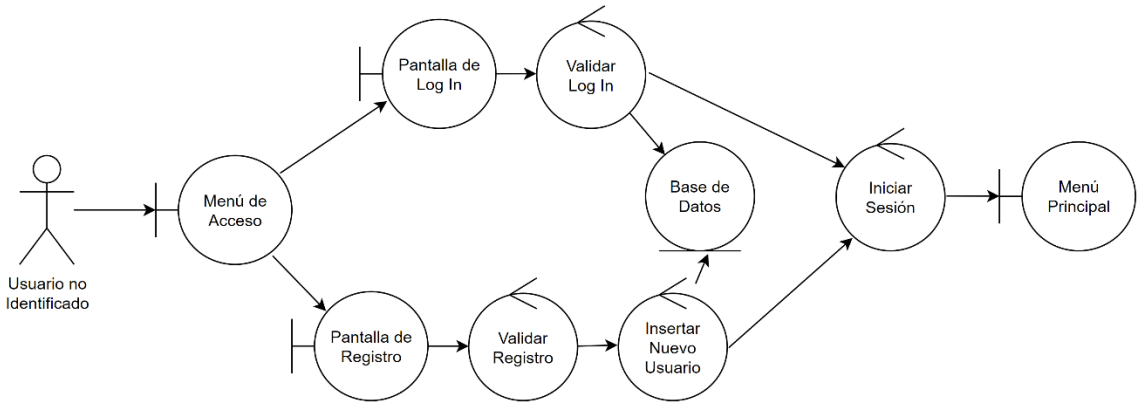


Figura 5.7. Diagrama de robustez para los escenarios relativos al acceso

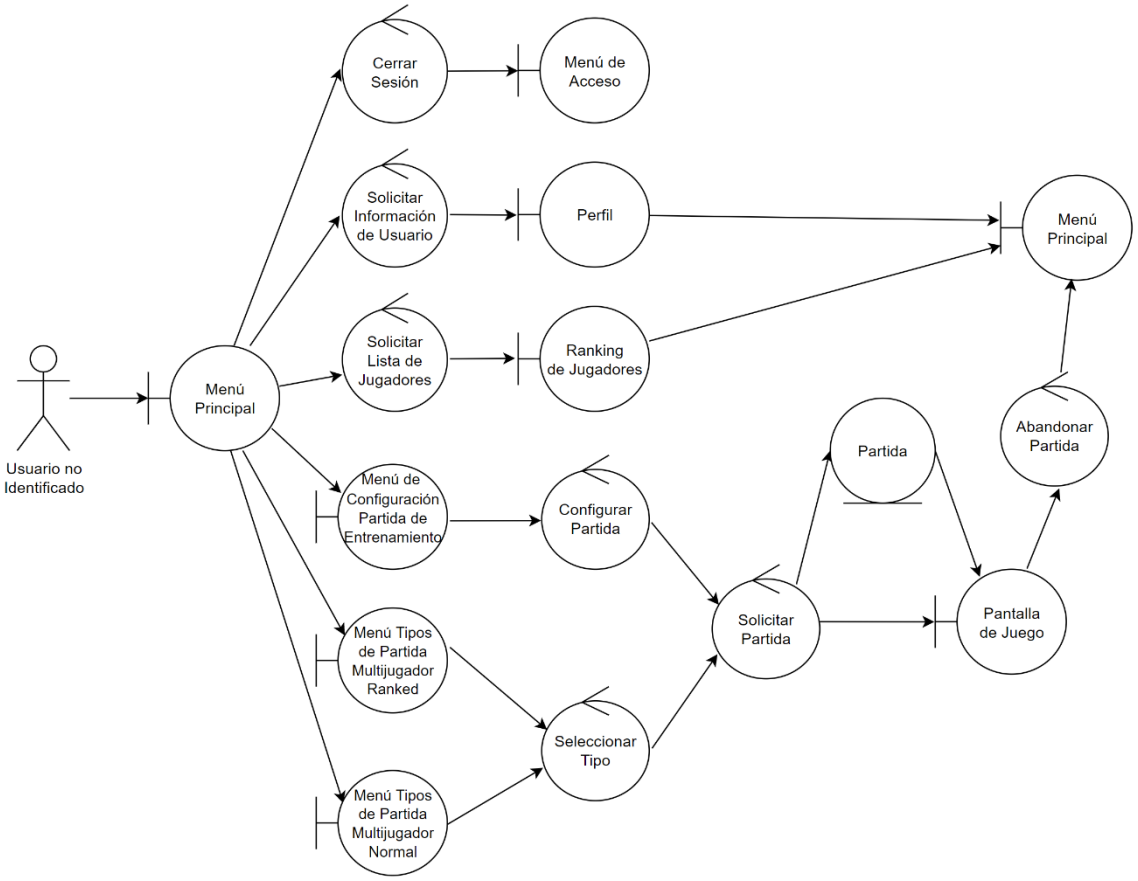


Figura 5.8. Diagrama de robustez para los escenarios desde el menú principal

5.5.1 Caso de Uso: Acceder Mediante Credenciales (*Log In*)

Acceso mediante credenciales	
Precondiciones	No tener una sesión iniciada en la aplicación.
Postcondiciones	Al usuario se le iniciará una sesión en la aplicación.
Actores	Usuario no identificado.
Descripción	1. El sistema muestra el menú de acceso. 2. El usuario escoge la opción de *Log in*. 3. El sistema muestra el formulario con todos los campos necesarios. 4. El usuario introduce un email y una contraseña. 5. El sistema valida que la información proporcionada sea correcta. 6. El usuario entra en la aplicación con su sesión iniciada.
Variaciones (escenarios secundarios)	• **Escenario Alternativo 1**: La validación de credenciales falla porque no se ha introducido alguno de los campos obligatorios. o Mostrar un mensaje de error adecuado al usuario. o Volver al paso 3 del escenario principal.

	• **Escenario Alternativo 2**: Par email-contraseña inválido (no presente en base de datos). o Mostrar mensaje de error al usuario indicando que las credenciales no son válidas. o Volver al paso 3 del escenario principal. • **Escenario Alternativo 3**: El usuario no recuerda su cuenta o no dispone de una. o Ofrecer la posibilidad de volver atrás, al menú principal de acceso. • **Escenario Alternativo 4**: Las credenciales son correctas, pero ya hay una sesión de esa cuenta activa. o Informar al usuario de este acontecimiento y pedirle que lo intente más tarde.
Excepciones	• **No es posible establecer conexión con el servidor**: Sin esto no es posible validar las credenciales y crear una sesión nueva. o Mostrar al usuario un mensaje de error adecuado.
Notas	-

5.5.2 Caso de Uso: Registrarse

Registro en la aplicación	
Precondiciones	No tener una sesión iniciada en la aplicación.
Postcondiciones	Al usuario se le iniciará una sesión en la aplicación y se insertará en la base de datos del sistema.
Actores	Usuario no identificado.
Descripción	1. El sistema muestra el menú de acceso. 2. El usuario escoge la opción de *Register*. 3. El sistema muestra el formulario con todos los campos necesarios. 4. El usuario introduce un nombre de usuario, email, contraseña y repetición de la contraseña. 5. El sistema valida que la información proporcionada sea correcta y cumpla con las condiciones establecidas. 6. El sistema inserta el nuevo usuario en su base de datos. 7. El usuario entra en la aplicación con su sesión iniciada.
Variaciones (escenarios secundarios)	• **Escenario Alternativo 1**: Alta errónea porque faltan campos obligatorios en el formulario. o Notificar el hecho al usuario, mostrándole un mensaje de error adecuado a la situación. o Volver al paso 3 del escenario principal, aunque manteniendo la información introducida en los distintos campos. • **Escenario Alternativo 2**: Nombre de usuario ya presente en el

	sistema. o Mostrar un error al usuario, diciendo que los nombres de usuario deben ser únicos. o Volver al paso 3 del escenario principal • **Escenario Alternativo 3**: Email ya presente en el sistema. o Mostrar un error al usuario, diciendo que un email solo puede asociarse a una única cuenta. o Volver al paso 3 del escenario principal, aunque manteniendo la información introducida en los distintos campos. • **Escenario Alternativo 4**: El nombre de usuario no cumple los requisitos establecidos. o Mostrar un error al usuario, especificando qué condiciones debe cumplir un nombre de usuario. o Volver al paso 3 del escenario principal, aunque manteniendo la información introducida en los distintos campos. • **Escenario Alternativo 5**: El email no tiene un formato válido. o Mostrar un error al usuario, pidiéndole que introduzca un email válido. o Volver al paso 3 del escenario principal, aunque manteniendo la información introducida en los distintos campos. • **Escenario Alternativo 6**: La contraseña no cumple los requisitos establecidos. o Mostrar un error al usuario, especificando las condiciones que debe cumplir una contraseña. o Volver al paso 3 del escenario principal, aunque manteniendo la información introducida en los distintos campos. • **Escenario Alternativo 7**: El contenido de los dos campos de contraseña no coincide. o Mostrar un error al usuario, diciéndole que deben coincidir. o Volver al paso 3 del escenario principal, aunque manteniendo la información introducida en los distintos campos.
Excepciones	• **No es posible establecer conexión con el servidor**: Sin esto no es posible insertar un nuevo usuario y crear una sesión nueva. o Mostrar al usuario un mensaje de error adecuado.
Notas	También se ofrecerá una opción para volver al menú principal de acceso.

5.5.3 Caso de Uso: Cerrar Sesión

Cerrar sesión	
Precondiciones	Tener una sesión iniciada en la aplicación.
Postcondiciones	La sesión del usuario se invalidará, volviendo al menú de acceso.
Actores	Usuario identificado.
Descripción	1. El sistema muestra el menú principal. 2. El usuario escoge la opción de *Log out*. 3. El sistema invalidará la sesión actual. 4. Al usuario se le muestra el menú de acceso.
Variaciones (escenarios secundarios)	-
Excepciones	-
Notas	Aunque, por alguna razón, no se pueda informar al servidor de que la sesión ha sido cerrada, la aplicación del cliente si cerrará la sesión local y mostrará el menú de acceso. Es posible que esto cause algún problema sin intenta volver a iniciar sesión con la misma cuenta, pero, dado que esto ocurrirá en ocasiones muy limitadas y se solucionará simplemente esperando, no será necesario implementar más mecanismos para evitarlo, ya que podrían comprometer la seguridad.

5.5.4 Caso de Uso: Ver Ranking de Jugadores

Ver ranking de jugadores	
Precondiciones	Tener una sesión iniciada en la aplicación.
Postcondiciones	Se mostrará un *ranking* de jugadores ordenado por puntuaciones, de más alta a más baja, en el que se mostrará información sobre ellos, permitiéndole moverse entre páginas.
Actores	Usuario identificado.
Descripción	1. El sistema muestra el menú principal. 2. El usuario escoge la opción de *Player Ranking*. 3. El sistema obtendrá la información necesaria sobre los jugadores. 4. Se le mostrará al usuario una tabla paginada de todos los jugadores ordenados por puntuación.
Variaciones (escenarios secundarios)	-
Excepciones	• **No es posible establecer conexión con el servidor**: Sin esta conexión, no se puede recuperar la información de los distintos jugadores. o La tabla simplemente se mostrará vacía para no molestar al usuario con excesivos mensajes de error.

Notas	-

5.5.5 Caso de Uso: Ver Perfil Propio

Ver perfil propio	
Precondiciones	Tener una sesión iniciada en la aplicación.
Postcondiciones	Se mostrará una ventana de perfil simple y que contenga toda la información necesaria.
Actores	Usuario identificado.
Descripción	1. El sistema muestra el menú principal. 2. El usuario escoge la opción *Profile*. 3. El sistema obtendrá la información necesaria sobre el usuario con sesión iniciada. 4. Se le mostrará su información de forma simple y agradable.
Variaciones (escenarios secundarios)	-
Excepciones	• **No es posible establecer conexión con el servidor**: Sin esta conexión, no se puede recuperar la información del usuario. o Se mostrará un perfil con datos por defecto, para dar a entender al jugador que la información no se ha podido recuperar, pero no molestarle con excesivas alertas de error.
Notas	-

5.5.6 Caso de Uso: Buscar Partida de Entrenamiento

Buscar partida de entrenamiento	
Precondiciones	Tener una sesión iniciada en la aplicación.
Postcondiciones	Se solicitará al sistema una nueva partida de este tipo, que el usuario podrá jugar.
Actores	Usuario identificado.
Descripción	1. El sistema muestra el menú principal. 2. El usuario escoge la opción de *Tranining Game*. 3. El sistema mostrará al usuario la pantalla donde podrá configurar (o dejar por defecto, si lo desea) los parámetros disponibles. 4. El usuario escogerá la opción *Play*. 5. Se solicitará una partida de este tipo al sistema para el usuario

	identificado. 6. Se le mostrará una pantalla de carga al usuario hasta que el sistema responda con la partida. 7. El sistema mostrará al usuario la pantalla desde la que podrá jugar la partida.
Variaciones (escenarios secundarios)	• **Escenario Alternativo 1**: El usuario decide cancelar la partida en la pantalla de carga. o Se enviará la notificación pertinente al servidor. o Al usuario se le mostrará de nuevo el menú principal.
Excepciones	• **No es posible establecer conexión con el servidor**: Sin esta conexión, no se puede crear ni jugar la partida. o Se le mostrará al usuario un mensaje informándole del error y pidiendo que lo intente más tarde.
Notas	Es necesario que la configuración que el usuario ha visto en su pantalla coincida con la que se utilice para el paso 5 del escenario principal.

5.5.7 Caso de Uso: Configurar Partida de Entrenamiento

Configurar partida de entrenamiento	
Precondiciones	Tener una sesión iniciada en la aplicación.
Postcondiciones	Se guardará la configuración escogida por el usuario para
Actores	Usuario identificado.
Descripción	1. El sistema muestra el menú principal. 2. El usuario escoge la opción de *Tranining Game*. 3. El sistema mostrará al usuario la pantalla para configurar los parámetros disponibles. 4. El usuario modificará aquellos que desee. 5. El sistema registrará todos los cambios llevados a cabo por el usuario.
Variaciones (escenarios secundarios)	-
Excepciones	-
Notas	Se deberán implementar los mecanismos necesarios para asegurarse de que la configuración que el usuario identificado ve en su pantalla se corresponde con la que el sistema guarda y utilizará posteriormente para solicitar una nueva partida.

5.5.8 Caso de Uso: Buscar Partida Multijugador Normal

Buscar partida multijugador normal	
Precondiciones	Tener una sesión iniciada en la aplicación.
Postcondiciones	Se solicitará al sistema una nueva partida de este tipo, que el usuario podrá jugar.
Actores	Usuario identificado.
Descripción	1. El sistema muestra el menú principal. 2. El usuario escoge la opción de *Normal PVP Game*. 3. El sistema mostrará al usuario la pantalla donde podrá escoger el tipo de partida que desea en función de la limitación de tiempo. 4. El usuario escogerá la opción *Play*. 5. Se solicitará una partida de este tipo al sistema para el usuario identificado. 6. Se le mostrará una pantalla de carga al usuario hasta que el sistema responda con la partida y mientras se le busca un rival. 7. El sistema mostrará al usuario la pantalla desde la que podrá jugar la partida.
Variaciones (escenarios secundarios)	• **Escenario Alternativo 1**: El usuario decide cancelar la partida mientras se busca un rival. o Se enviará la notificación pertinente al servidor. o Al usuario se le mostrará de nuevo el menú principal.
Excepciones	• **No es posible establecer conexión con el servidor**: Sin esta conexión, no se puede crear ni jugar la partida. o Se le mostrará al usuario un mensaje informándole del error y pidiendo que lo intente más tarde. • **El usuario ya tiene una partida multijugador en proceso**: Las especificaciones del sistema explicitan que no es posible jugar varias partidas multijugador de forma simultánea. Se le mostrará un mensaje de error informándole de esto.
Notas	Se deberán implementar los mecanismos necesarios para asegurarse de que el tipo de partida que quiere jugar el usuario coincide con el tipo que se solicita al sistema.

5.5.9 Caso de Uso: Buscar Partida Multijugador Clasificatoria

Buscar partida multijugador clasificatoria	
Precondiciones	Tener una sesión iniciada en la aplicación.

Postcondiciones	Se solicitará al sistema una nueva partida de este tipo, que el usuario podrá jugar.
Actores	Usuario identificado.
Descripción	1. El sistema muestra el menú principal. 2. El usuario escoge la opción de *Ranked PVP Game*. 3. El sistema mostrará al usuario la pantalla donde podrá escoger el tipo de partida que desea en función de la limitación de tiempo. 4. El usuario escogerá la opción *Play*. 5. Se solicitará una partida de este tipo al sistema para el usuario identificado. 6. Se le mostrará una pantalla de carga al usuario hasta que el sistema responda con la partida y mientras se le busca un rival. 7. El sistema mostrará al usuario la pantalla desde la que podrá jugar la partida.
Variaciones (escenarios secundarios)	• **Escenario Alternativo 1**: El usuario decide cancelar la partida mientras se busca un rival. o Se enviará la notificación pertinente al servidor. o Esta decisión no afectará a la puntuación del usuario, al no haber comenzado la partida. o Al usuario se le mostrará de nuevo el menú principal.
Excepciones	• **No es posible establecer conexión con el servidor**: Sin esta conexión, no se puede crear ni jugar la partida. o Se le mostrará al usuario un mensaje informándole del error y pidiendo que lo intente más tarde. • **El usuario ya tiene una partida multijugador en proceso**: Las especificaciones del sistema explicitan que no es posible jugar varias partidas multijugador de forma simultánea. Se le mostrará un mensaje de error informándole de esto.
Notas	Se deberán implementar los mecanismos necesarios para asegurarse de que el tipo de partida que quiere jugar el usuario coincide con el tipo que se solicita al sistema.

5.5.10 Caso de Uso: Escoger Límite de Tiempo

Escoger límite de tiempo	
Precondiciones	Tener una sesión iniciada en la aplicación.
Postcondiciones	Se guardará el tipo de partida que el jugador busca en función de cómo se aplique el tiempo límite.
Actores	Usuario identificado.
Descripción	1. El sistema muestra el menú principal. 2. El usuario escoge una de las opciones de *PVP Game*. 3. El sistema mostrará al usuario la pantalla donde podrá escoger

	el tipo de partida que desea en función de la limitación de tiempo. 4. El usuario seleccionará el tipo de partida que desea atendiendo al límite de tiempo. 5. El sistema registrará esta información.
Variaciones (escenarios secundarios)	-
Excepciones	-
Notas	El sistema deberá asegurarse de que se aplica el tipo de limitación que el jugador ha escogido.

5.5.11 Caso de Uso: Jugar Partida

Jugar partida	
Precondiciones	Tener una sesión iniciada en la aplicación, haber solicitado una partida del tipo que sea y que esta partida haya sido iniciada.
Postcondiciones	El sistema se encargará de toda la lógica posterior a una partida (puntuaciones, guardar secuencia de movimientos, etc.).
Actores	Usuario identificado.
Descripción	1. El sistema muestra la pantalla desde la que jugar la partida. 2. El usuario juega contra su rival, desponiendo de todas las funcionalidades dependiendo del tipo de partida. 3. La partida llega a su estado final. 4. El sistema ejecutará toda la lógica de final de partida. 5. Al jugador se le informará de si ha ganado o perdido, dejándole también observar el estado final de la partida e historial. 6. Al usuario se le ofrecerá una opción para volver, cuando desee, al menú principal.
Variaciones (escenarios secundarios)	• **Escenario Alternativo 1**: El usuario agota, de haberlo, su tiempo límite. o Habrá perdido la partida, aunque esta no haya llegado a su estado final. o Se procederá con el escenario principal a partir del paso 4. o Las secuencias de movimientos de las partidas que no hayan llegado a su estado final no se guardarán. • **Escenario Alternativo 2**: El usuario abandona la partida antes de que termine (cierra el navegador o elige la opción de volver al menú principal). o Habrá perdido la partida, aunque esta no haya llegado a su estado final. o Se procederá con el escenario principal a partir del paso 4. o Las secuencias de movimientos de las partidas que no

	hayan llegado a su estado final no se guardarán.
Excepciones	• **El usuario pierde la conexión con el servidor**: Si no la recupera en un tiempo corto, se considerará que ha perdido la partida. o Se procederá con el escenario principal a partir del paso 4. o Las secuencias de movimientos que no hayan llegado a su estado final no se guardarán. • **El usuario intenta hacer trampas utilizando un software externo**: Un jugador podría tratar de hacer peticiones directamente al servidor para realizar movimientos que pertenezcan a su rival, por ejemplo. Esto será detectado y el jugador tramposo perderá la partida. o Se procederá con el escenario principal a partir del paso 4. o Las secuencias de movimientos de las partidas que no hayan llegado a su estado final no se guardarán.
Notas	De este escenario podrían derivar muchos escenarios más: jugar desde el tablero interactivo, árbol interactivo y todas sus funcionalidades, o los que tienen que ver con el historial de movimientos. Se ha preferido no detallar estos escenarios, ya que su implementación aún no está clara y sería entrar a detalla aspectos de muy bajo nivel para esta fase.

5.5.12 Caso de Uso: Modificar Archivo de Configuración

Modificar archivo de configuración	
Precondiciones	Tener localizado este archivo de configuración dentro de la máquina (física o virtual) utilizada como servidor.
Postcondiciones	De no contener ningún error, se aplicarán estas nuevas condiciones.
Actores	Administrador del sistema.
Descripción	1. El administrador abrirá el archivo con cualquier editor de texto. 2. El administrador modificará todos aquellos parámetros que desee. 3. El administrador podrá relanzar el programa del servidor para asegurarse de que las configuraciones se apliquen (aunque esto no será necesario para la mayoría de los parámetros. 4. Estos parámetros serán leídos y aplicados por los subsistemas correspondientes.
Variaciones (escenarios secundarios)	-
Excepciones	• **El administrador introduce errores en el archivo de configuración**: No se pueden recuperar algunos parámetros.

	○ Aquellos parámetros que no se puedan recuperar del archivo, emplearán su valor por defecto, especificado en el código. ○ Los parámetros que sigan pudiendo leerse serán aplicados. ○ El servidor avisará al administrador de esto, y extraerá el error a un archivo de registro. • **El sistema no es capaz de encontrar el archivo de configuración**: No se puede recuperar ningún parámetro. ○ Todos los parámetros emplearán su valor por defecto, especificado en el código. ○ El servidor avisará al administrador de esto, y extraerá el error a un archivo de registro.
Notas	Este caso de uso es totalmente ajeno a la interfaz de usuario.

5.5.13 Caso de Uso: Modificar el Contenido de la Base de Datos

Modificar el contenido de la base de datos	
Precondiciones	Disponer de un dispositivo con navegar y conexión a internet.
Postcondiciones	Se aplicarán los cambios realizados.
Actores	Administrador del sistema.
Descripción	1. El administrador accederá a la página principal de MongoDB. 2. Se identificará con las credenciales utilizadas para solicitar el *cluster* en la nube. 3. El administrador accederá a la interfaz de usuario que ofrece MongoDB para administrar los contenidos de la base de datos. 4. Realizará las modificaciones necesarias. 5. Confirmará esas modificaciones. 6. Por seguridad, el usuario deberá siempre cerrar la sesión.
Variaciones (escenarios secundarios)	-
Excepciones	• **No es posible establecer conexión con MongoDB**: El administrador deberá esperar a que se solucione el problema, ya que esto no depende de nuestro sistema.
Notas	Este caso de uso es totalmente ajeno a la interfaz de usuario.

5.6 Análisis de Interfaces de Usuario

En la siguiente sección se presentará, de forma esquemática, el diseño que tendrán las distintas pantallas de la aplicación *web*, acompañado de las aclaraciones que se consideren necesarias, y se mostrará un diagrama de cómo el usuario va a poder viajar entre ellas.

5.6.1 Descripción de la Interfaz

En esta sección podrán verse los prototipos de todas las pantallas que se ha decidido que la aplicación de navegador tendrá. De considerarse necesario, irán acompañadas de una breve explicación textual de su función o aspectos que puedan no quedar claros.

Las tres primeras pantallas solo estarán disponibles para usuarios no identificados, mientras que el resto serán exclusivas para usuarios identificados.

5.6.1.1 Interfaz de Menú de Acceso

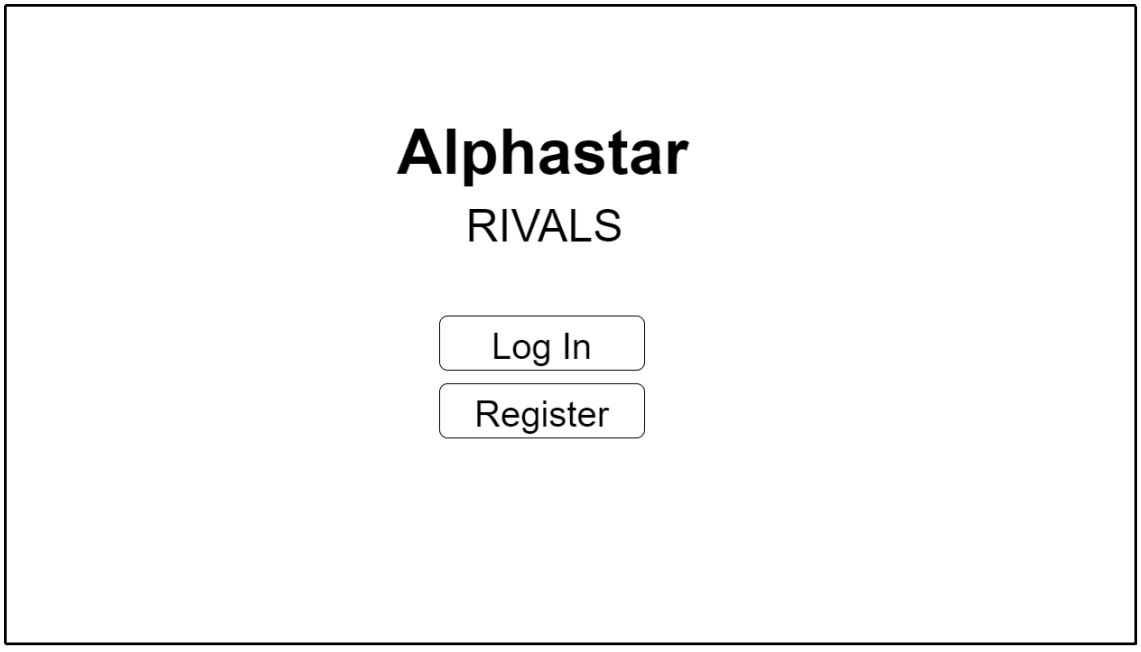


Figura 5.9. Pantalla de Menú de Acceso

5.6.1.2 Interfaz de Log In

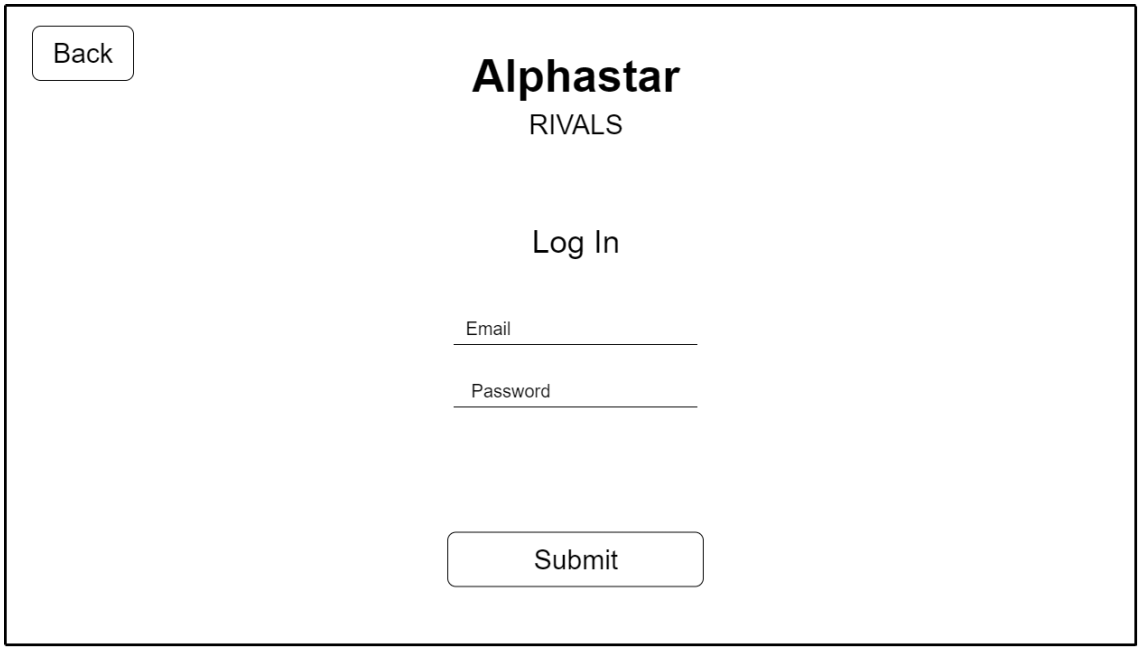


Figura 5.10. Pantalla de Log In

5.6.1.3 Interfaz de Registro

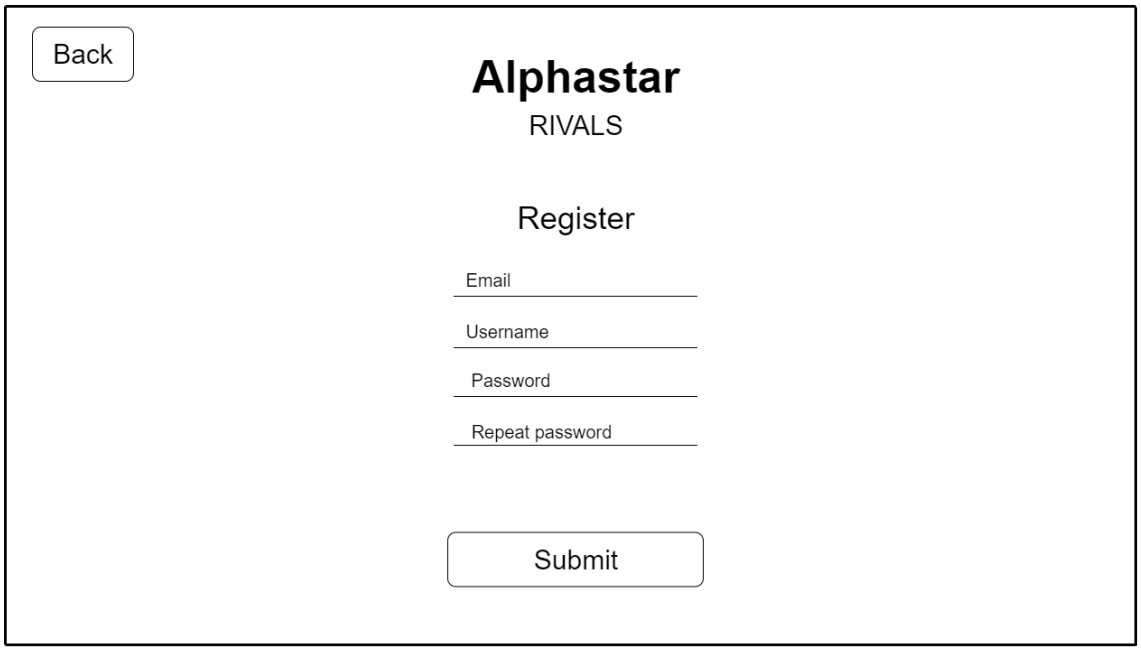


Figura 5.11. Pantalla de Registro

5.6.1.4 Interfaz de Menú Principal

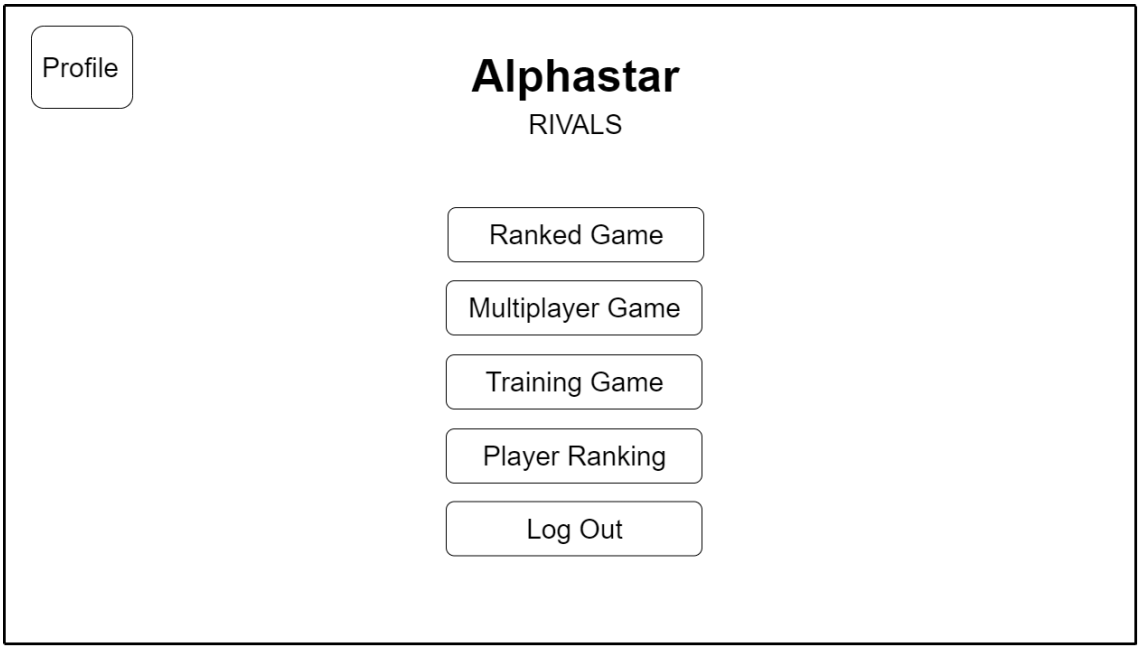


Figura 5.12. Pantalla de Menú Principal

5.6.1.5 Interfaz de Selección de Tipo de Partida

Esta interfaz será la misma (o muy similar) tanto para partidas multijugador normales como para partidas multijugador clasificatorias.

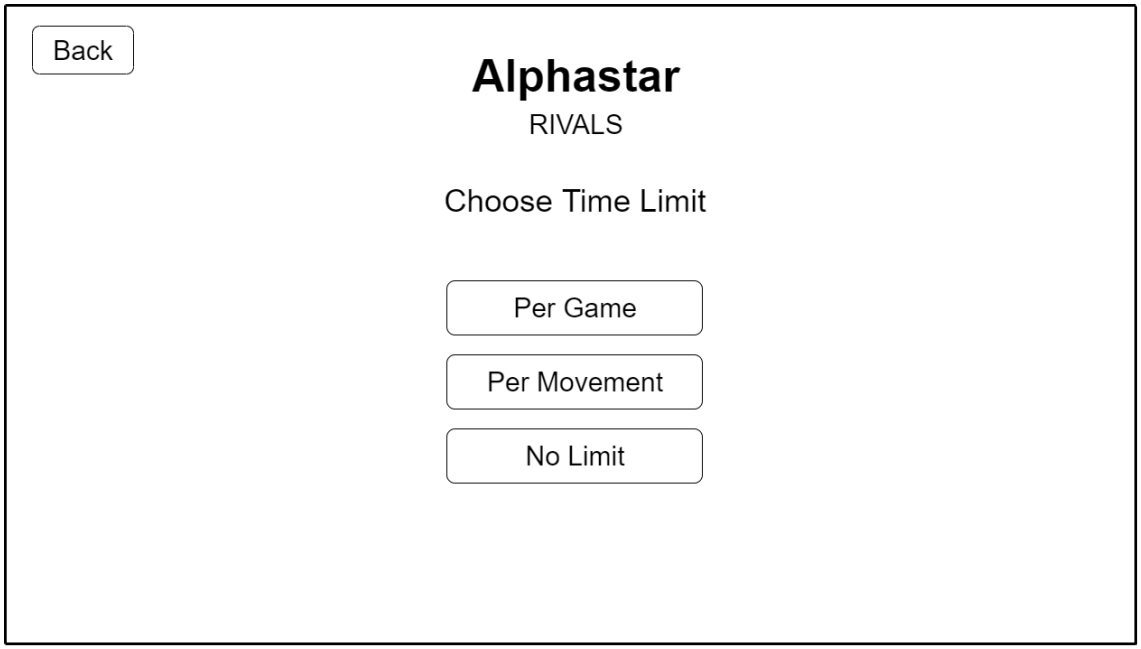


Figura 5.13. Pantalla de Selección de Tipo de Partido

5.6.1.6 Interfaz de Configuración de Partida de Entrenamiento

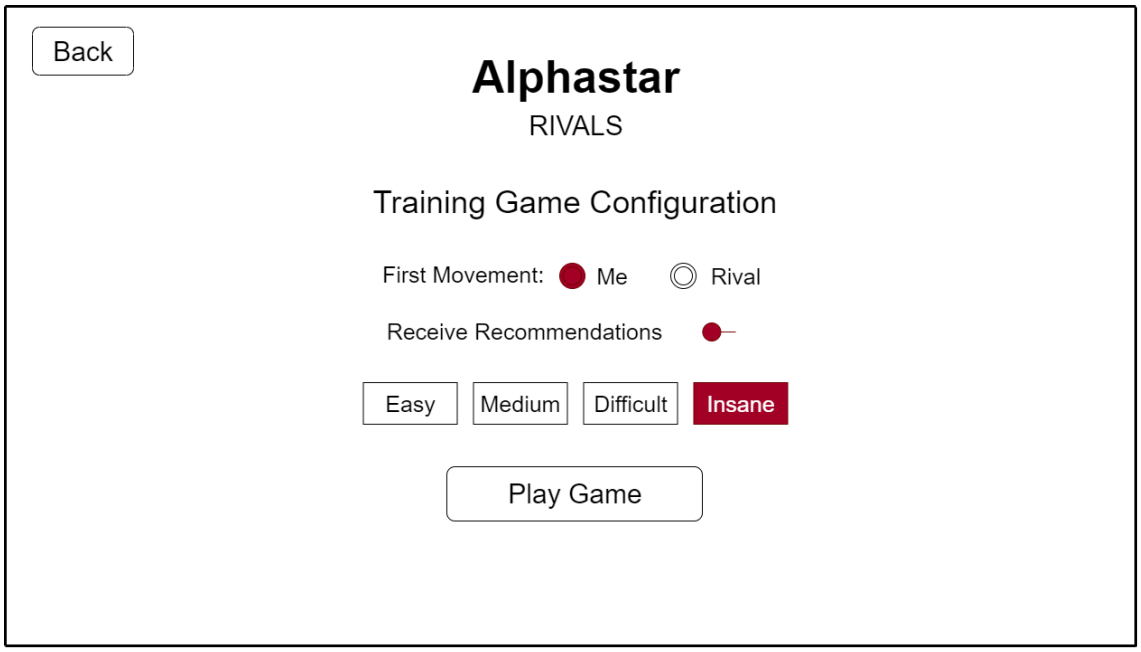


Figura 5.14. Pantalla de Configuración de Partida de Entrenamiento

5.6.1.7 Interfaz de Ranking de Jugadores

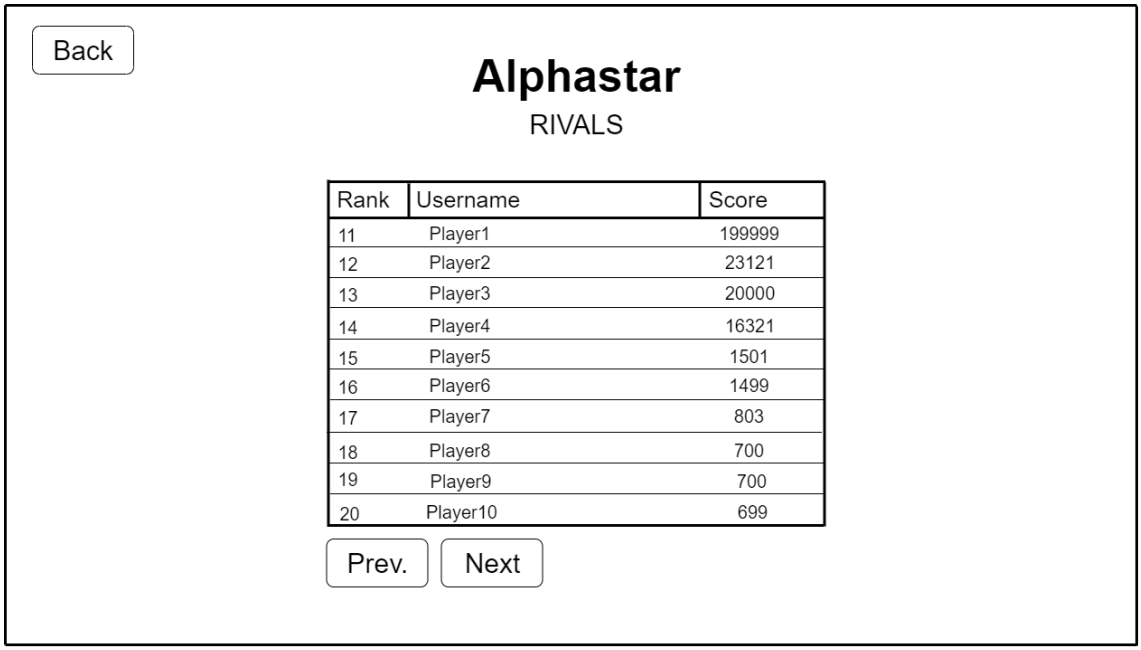


Figura 5.15. Pantalla de Ranking de Jugadores

5.6.1.8 Interfaz de Perfil de Usuario

Esta interfaz no está diseñada para ocupar la pantalla completa, sino para ser un cuadro de diálogo que surja cuando sea necesario.

Figura 5.16. Pantalla de Perfil de Usuario

5.6.1.9 Interfaz de la Pantalla de Juego

En el centro de esta pantalla, ocupando la mayoría del espacio disponible, aparecerán las distintas formas de visualizar información que tendrá el usuario durante una partida: tablero interactivo, árbol interactivo e historial.

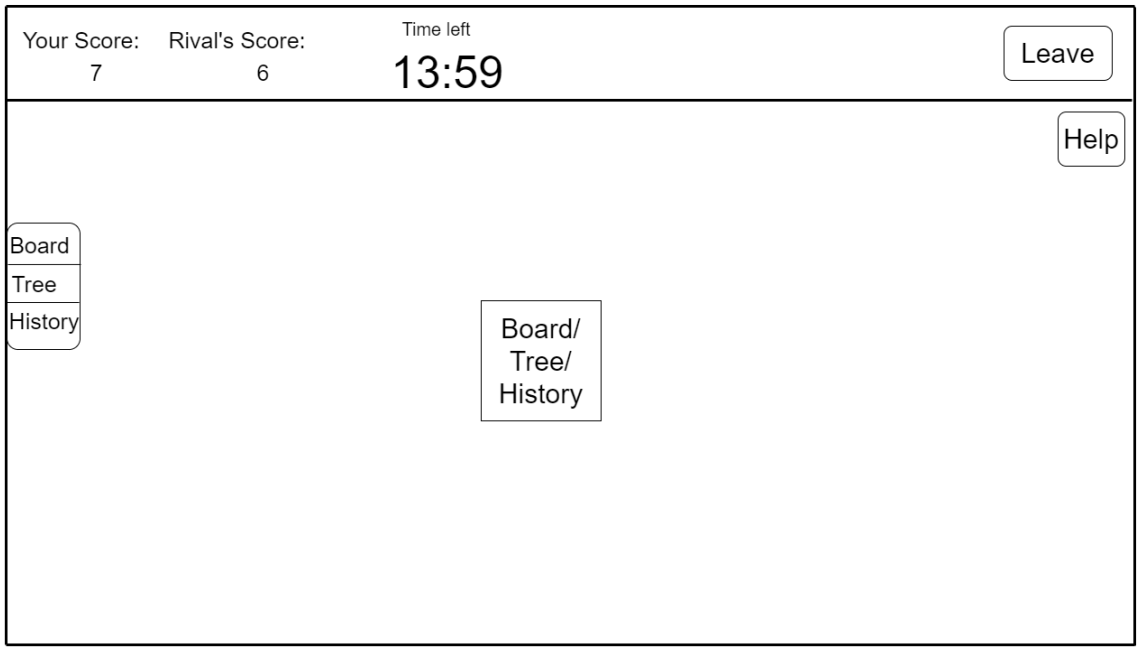


Figura 5.17. Pantalla de Juego

5.6.1.10 Interfaz de la Pantalla de Ayuda

Esta interfaz no está diseñada para ocupar toda la pantalla, sino que actuará como un cuadro de diálogo flotante que se mostrará sobre la pantalla de juego.

Las secciones que en la Figura 5.18 aparecen representadas como cuadrados contendrán toda aquella información que se considere para que el usuario sea capaz de entender el funcionamiento de todas las herramientas de las que dispone.

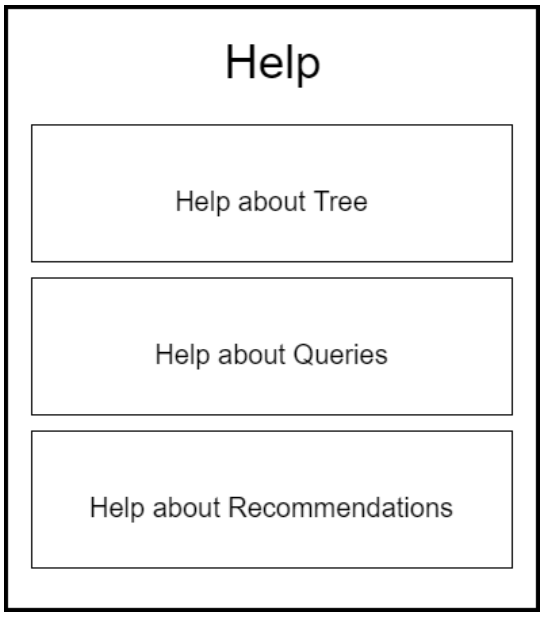


Figura 5.18. Pantalla de Ayuda

5.6.1.11 Interfaz de Partida Finalizada

De forma similar a la anterior, la interfaz no ocupará toda la pantalla, sino que se mostrará como un cuadro de diálogo flotante por encima de la pantalla de juego. La puntuación sumada o restada solo se mostrará en caso de que la partida terminada sea una clasificatoria.

Figura 5.19. Pantalla de Partida Finalizada

5.6.2 Descripción del Comportamiento de la Interfaz

En este apartado se especificarán todas las validaciones que el sistema efectuará sobre los datos introducidos por el usuario, qué mensajes de error se mostrarán en la aplicación y qué ayudas se proporcionarán a los usuarios.

Una de las partes de la interfaz que realizará comprobaciones sobre los datos introducidos por el usuario será la pantalla de *Log In* (inicio de sesión mediante credenciales de una cuenta existente). Aquí, el usuario no identificado deberá introducir un email y una contraseña, de los que se validará lo siguiente:

- Que ninguno de los campos esté vacío, ya que ambos son de carácter obligatorio.
- Que el par email-contraseña proporcionado por el cliente aparezca en la base de datos del sistema.
- Que no exista una sesión activa que se haya creado empleando esas mismas credenciales.

Si alguna de estas comprobaciones fallase, el sistema mostrará un mensaje de error advirtiendo de que no se han cubierto todos los campos obligatorios, que las credenciales no son válidas o que esa cuenta ya tiene una sesión iniciada, según el caso concreto.

La pantalla de registro también deberá comprobar los datos que el usuario introduzca en el formulario que implementa. Los campos requeridos serán nombre de usuario, email, contraseña y un campo extra en el que se debe repetir esa misma contraseña. De estos campos se comprobarán estos requisitos:

- Que no haya ningún campo vacío, ya que todos ellos son obligatorios. Tampoco podrán llenarse de espacios en blanco (excepto los campos contraseña: para estos el espacio en blanco se entiende como otro carácter válido).
- Que no exista ningún usuario con el mismo nombre en el sistema.
- Que el sistema no tenga vinculado a ninguna cuenta el email que se intenta usar para el registro.
- Que el nombre de usuario no supere los 20 caracteres de longitud.
- Que el email introducido presente un formato válido.
- Que la contraseña cuente con una longitud de entre seis y 40 caracteres (ambos incluidos).
- Que el contenido de los dos campos en los que se ha de introducir la contraseña coincida exactamente.

De nuevo, si alguna parte de la validación falla, el sistema deberá exponer al usuario el error, mostrando un mensaje específico para cada uno de los casos expuestos en la anterior lista.

También se mostrará un mensaje de error adecuado a la situación cuando al usuario no se le permita buscar partida porque ya se encuentre en una partida multijugador en proceso y cuando alguna conexión con el servidor falle. Excepto cuando, por cualquier motivo, no sea posible recuperar el *ranking* de jugadores o la información que se ha de exhibir en el perfil. En estos casos, se evitará emitir un mensaje de error para no molestar al usuario, al no considerar esto un perjuicio grave para el resto de la funcionalidad de la aplicación.

Todos los mensajes de error que ha de implementar la aplicación se expondrán al usuario en forma de notificación de tipo *toast* (aparecerán en la esquina superior derecha, pudiendo apilarse unas debajo de otras, y desaparecerán cuando el usuario las cierre o automáticamente tras un período de tiempo determinado).

La única información de ayuda que se proporcionará al usuario será en la pantalla para jugar una partida. Esta ayuda aparecerá por encima de la pantalla de juego en cualquier momento, siempre que el usuario lo solicite, en forma de cuadro de diálogo. Aquí se explicará de forma textual y, si es necesario, mediante imágenes y leyendas toda la funcionalidad relativa al árbol. Deberá aclarar como explorar el árbol, lo que significan cada uno de los símbolos y colores, dónde encontrar las distintas informaciones disponibles y lo que estas significan, como usar todas las consultas implementadas y qué utilidad tienen, y cómo interpretar las recomendaciones que el sistema ofrecerá. Si durante el resto del proceso de desarrollo se identifica algún otro detalle similar, también se incluirá en este diálogo de ayuda.

5.6.3 Diagrama de Navegabilidad

En la Figura 5.20 se muestra un diagrama en el que se detallan todas las posibles transiciones entre distintas pantallas de la interfaz de usuario.

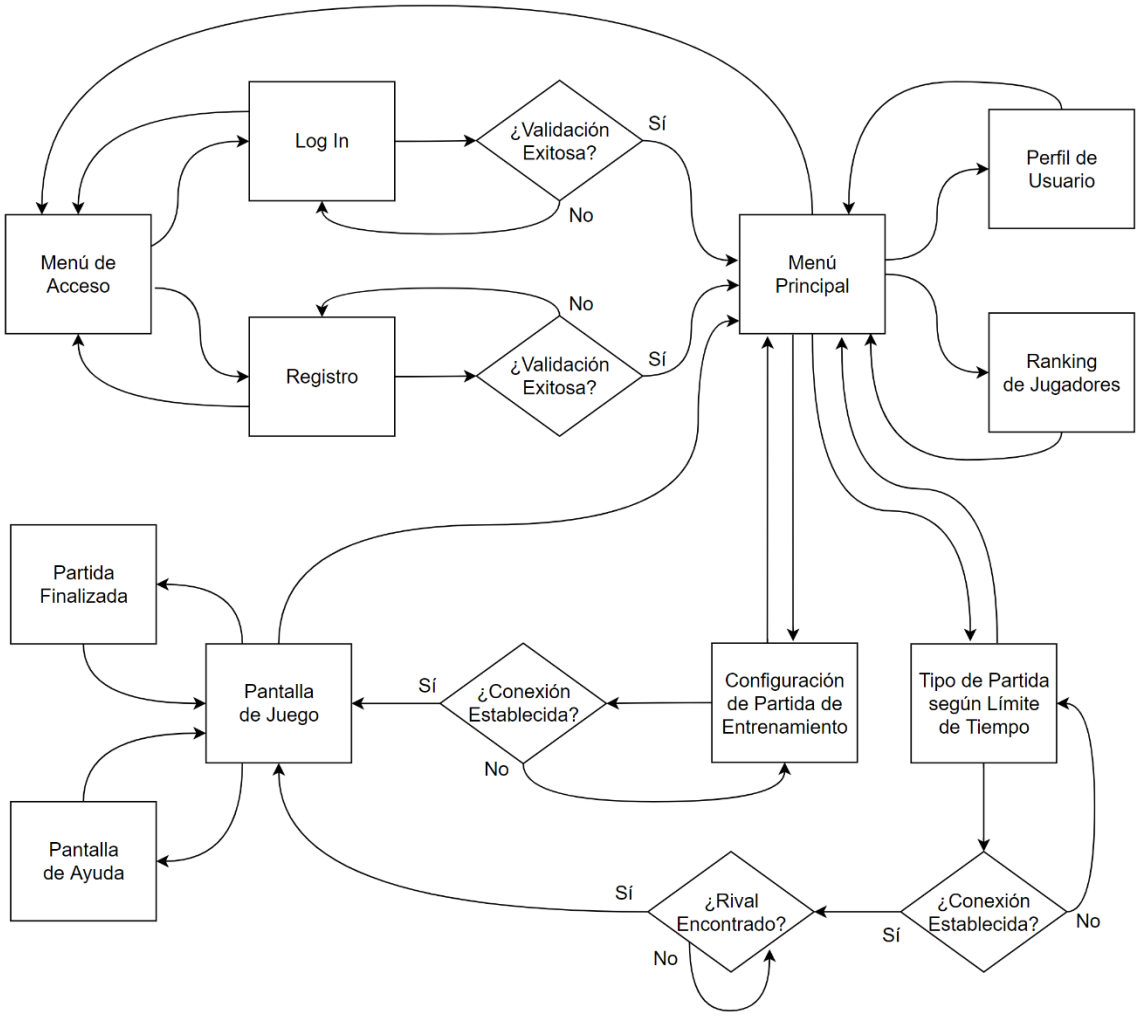


Figura 5.20. Diagrama de navegación entre pantallas

5.7 Especificación del Plan de Pruebas

En esta sección se concretará el plan de pruebas para la aplicación y sus subsistemas. Estas pruebas servirán para detectar errores y corregirlos en la fase de implementación.

Las pruebas podrán encuadrarse dentro de uno de los tipos siguientes.

- **Pruebas Unitarias**: Este tipo de pruebas sirve para asegurar el correcto funcionamiento de un módulo de código, una clase o una funcionalidad monolítica. De este modo, nos cercioraremos del correcto funcionamiento de esos módulos, clases o funcionalidades. Este tipo de pruebas tienen especial interés en aquellos segmentos del sistema que resulten críticos para el correcto funcionamiento de la aplicación en su conjunto, o para los que sea especialmente complejos de implementar.
- **Pruebas de Integración**: Estas pruebas sirven para verificar que un grupo de varios componentes funciona de la manera esperada cuando se utilizan en conjunción.
- **Pruebas del Sistema**: Similares a las anteriores, pero se ejecutan sobre el sistema completo. Permiten comprobar que el sistema funciona en conjunto, que sus partes se comunican adecuadamente y que todos los subsistemas responden como deben ante cualquier tipo de entrada (ya sean correctas o incorrectas).
- **Pruebas de Usabilidad**: Servirán para comprobar si el cliente está satisfecho con los resultados. Aunque este tipo de pruebas no serán especificadas en esta fase, aquí podrán aparecer, por ejemplo, pruebas de accesibilidad o de rendimiento de la aplicación *web*.

A continuación, se presentarán una serie de pruebas que han sido desarrolladas a partir de los casos de uso descritos en apartados anteriores. Para ello, se han intentado tener en cuenta una parte representativa de las entradas correctas, escenarios alternativos y posibles excepciones.

Caso de Uso 1: Acceder Mediante Credenciales (Log In)	
Prueba	**Resultado Esperado**
El usuario introduce unas credenciales válidas.	El sistema crea una nueva sesión para ese usuario, generando un *token* de autenticación que enviará al cliente del usuario.
Prueba	**Resultado Esperado**
El usuario deja vacío alguno de los campos obligatorios.	El sistema no genera el *token*, y advierte al usuario no identificado de su error.
Prueba	**Resultado Esperado**
El usuario introduce unas credenciales que no están presentes en el sistema.	Al contrastarlas con la base de datos, el sistema comprueba que esas credenciales no son válidas. También le muestra al usuario un mensaje de error y no genera ningún nuevo *token*.
Prueba	**Resultado Esperado**
El usuario introduce unas credenciales válidas, pero con una sesión activa.	El sistema comprueba en la base de datos que las credenciales son válidas, pero, tras buscar en la colección de *tokens* activos, encuentra que esa cuenta ya tiene un *token* asignado. No se genera ningún *token* y se advierte al usuario con un mensaje específico.
Prueba	**Resultado Esperado**
El usuario cancela la operación.	No se aplica ningún cambio al sistema.

Prueba	**Resultado Esperado**
El usuario introduce credenciales válidas, pero la conexión con el servidor falla.	No se aplica ningún cambio al sistema.
Prueba	**Resultado Esperado**
El usuario emplea las herramientas de desarrollador del navegador, por ejemplo, para intentar inhabilitar las validaciones.	El sistema realizará todas las validaciones igualmente, al estar presentes también en el servidor. Se abrirá una sesión nueva y se generará un *token* solo si la validación es exitosa, las credenciales son correctas y el usuario no tiene ya una sesión iniciada.
Prueba	**Resultado Esperado**
La conexión con el servidor falla.	El sistema permanecerá sin cambios.

Caso de Uso 2: Registrarse	
Prueba	**Resultado Esperado**
El usuario rellena todos los campos con datos correctos.	El sistema, tras validar esos datos, inserta al nuevo usuario en la base de datos. Tras esto, crea una sesión para él, generando un *token* nuevo y enviándoselo.
Prueba	**Resultado Esperado**
El usuario deja vacío alguno (o varios) de los campos obligatorios.	Esto es detectado durante el proceso de validación, por lo que el sistema no crea ninguna cuenta nueva ni genera un *token* de sesión. Además, advierte al usuario de su error.
Prueba	**Resultado Esperado**
El usuario introduce datos incorrectos, al fallar alguno de los requisitos de validación establecidos previamente.	De nuevo, el sistema es capaz de detectar el error y no generará una nueva cuenta ni un *token* de sesión. También mostrará al usuario un mensaje de error adecuado a cada situación.
Prueba	**Resultado Esperado**
El usuario cancela la operación.	El sistema no aplica ningún cambio.
Prueba	**Resultado Esperado**
El usuario se sirve de las herramientas de desarrollador del navegador para intentar inhabilitar las validaciones.	El sistema realizará todas las validaciones igualmente, al estar presentes también en el servidor. Solo se insertará el nuevo usuario si los datos introducidos pasan todas las comprobaciones.
Prueba	**Resultado Esperado**
La conexión con el servidor falla.	El sistema permanecerá sin cambios.

Caso de Uso 3: Cerrar Sesión	
Prueba	**Resultado Esperado**
El usuario identificado selecciona la opción correspondiente en el menú principal.	El sistema cancela el *token* de sesión perteneciente a ese usuario, lo que hará posible que acceda de nuevo con sus credenciales.
Prueba	**Resultado Esperado**

El usuario identificado pulsa la opción correcta, pero la conexión con el servidor falla.	La aplicación *web* muestra al usuario el menú de acceso. El sistema no cancelará el *token* de sesión, por lo que para entrar con las mismas credenciales deberá esperar a que se agote el tiempo de inactividad. Sí podrá entrar con otras credenciales o registrarse.

Caso de Uso 4: Ver Ranking de Jugadores	
Prueba	**Resultado Esperado**
El usuario identificado selecciona la opción para ver el *ranking* de forma normal.	El sistema obtendrá la primera página del *ranking* de jugadores (con un máximo de 10 entradas) y se lo mostrará en la pantalla adecuada. Además, permitirá que el usuario se mueva a la página previa (si no se encuentra ya en la primera) y a la siguiente (si no se encuentra en la última). No se realizarán cambios en el sistema.
Prueba	**Resultado Esperado**
El usuario identificado pulsa la opción correspondiente, pero la conexión con el servidor falla.	La aplicación *web* muestra al usuario la interfaz de ver *ranking*, aunque vacía, al no haber podido recuperar la información. No se efectuará ningún cambio en el sistema.
Prueba	**Resultado Esperado**
El usuario identificado modifica los parámetros de la petición (número de página o entradas por página) empleando las herramientas para desarrolladores.	El sistema será capaz de mostrar igualmente el *ranking*, adaptándose a los parámetros. Si los parámetros se sitúan fuera de los límites de la colección de jugadores, el *ranking* aparecerá vacío. Si los parámetros son eliminados, el sistema dispondrá de unos por defecto (primera página y 10 entradas por página). El sistema permanecerá sin cambios.

Caso de Uso 5: Ver Perfil Propio	
Prueba	**Resultado Esperado**
El usuario identificado abre su perfil desde el menú principal.	El sistema le mostrará en un cuadro de diálogo la información de su cuenta. No se realizarán cambios en el sistema.
Prueba	**Resultado Esperado**
El usuario identificado pulsa la opción correspondiente, pero la conexión con el servidor falla.	El sistema mostrará el cuadro de diálogo correspondiente, pero con datos por defecto. No se efectuarán cambios en el sistema.

Caso de Uso 6: Buscar Partida de Entrenamiento	
Prueba	**Resultado Esperado**
El usuario identificado selecciona la opción correspondiente.	El sistema comprobará que cuenta con un *token* activo y creará una partida nueva de entrenamiento para él. Tras un breve tiempo de carga, empezará la partida para el jugador.
Prueba	**Resultado Esperado**
El usuario identificado pulsa la opción correspondiente, pero la conexión con el servidor falla.	No se creará ninguna partida para el jugador. El sistema le mostrará un mensaje advirtiendo de que no ha sido posible establecer la conexión.

Prueba	Resultado Esperado
El usuario identificado pulsa la opción correspondiente, pero no cuenta con un *token* activo o este ha expirado.	No se creará ninguna partida para el jugador. El sistema le advertirá del error y le recomendará acceder de nuevo con sus credenciales.
Prueba	**Resultado Esperado**
El usuario identificado modifica los parámetros de configuración que se envían junto con la petición, empleando las herramientas de desarrollador.	Si los parámetros introducidos representan opciones de configuración válidas, el sistema creará la partida de la forma solicitada. Si esos parámetros no son válidos, el sistema contará con una configuración por defecto y le creará una partida con esta configuración.
Prueba	**Resultado Esperado**
El usuario identificado cancela la partida durante la pantalla de carga.	El sistema lo detecta y le lleva de nuevo al menú principal.

Caso de Uso 7: Configurar Partida de Entrenamiento	
Prueba	**Resultado Esperado**
El usuario identificado selecciona no modifica ningún parámetro.	La aplicación de navegador guardará y mostrará las configuraciones por defecto, para luego enviarlas a la hora de solicitar una partida.
Prueba	**Resultado Esperado**
El usuario identificado modifica algún parámetro (o varios, desde la interfaz).	La aplicación rastreará estos cambios y los guardará para luego usarlos para solicitar una partida. La configuración que guarda la aplicación se corresponderá con la que le muestra al usuario, aunque cambie de pantalla y luego vuelva.
Prueba	**Resultado Esperado**
El usuario identificado varía las opciones de configuración mediante las herramientas para desarrolladores del navegador.	Si estas son válidas, la interfaz las mostrará. Si son inválidas, no se puede predecir el comportamiento de la interfaz *web*, ya que el usuario puede, potencialmente, modificarlo todo. De todas formas, estas configuraciones se validarán en el servidor cuando se solicite una partida, donde habrá una configuración por defecto para estos casos.

Caso de Uso 8: Buscar Partida Multijugador Normal	
Prueba	**Resultado Esperado**
El usuario identificado selecciona la opción correspondiente.	El sistema comprobará que cuenta con un *token* activo y creará una partida multijugador normal nueva para él, o lo introducirá en una ya creada del mismo tipo. Cuando se encuentre un rival que busca el mismo tipo de partida, empezará la partida para el jugador.
Prueba	**Resultado Esperado**
El usuario identificado pulsa la opción correspondiente, pero la conexión con el servidor falla.	No se creará ninguna partida para el jugador. El sistema le mostrará un mensaje advirtiendo de que no ha sido posible establecer la conexión.
Prueba	**Resultado Esperado**
El usuario identificado pulsa	No se creará ninguna partida para el jugador. El sistema le

la opción correspondiente, pero no cuenta con un *token* activo o este ha expirado.	advertirá del error y le recomendará acceder de nuevo con sus credenciales.
Prueba	**Resultado Esperado**
El usuario identificado intenta solicitar una partida cuando ya se encuentra jugando una partida multijugador.	El sistema lo detecta y le mostrará un mensaje de error diciéndole que no es posible jugar varias partidas multijugador de forma simultánea.
Prueba	**Resultado Esperado**
El usuario identificado modifica los parámetros de configuración que se envían junto con la petición, empleando las herramientas de desarrollador.	Si los parámetros introducidos representan opciones de configuración válidas, el sistema creará la partida de la forma solicitada. Si los parámetros no son válidos, la partida multijugador no se creará y el usuario permanecerá en el menú principal.

Caso de Uso 9: Buscar Partida Multijugador Clasificatoria	
Prueba	**Resultado Esperado**
El usuario identificado selecciona la opción correspondiente.	El sistema comprobará que cuenta con un *token* activo y creará una partida multijugador clasificatoria nueva para él o, si hay una partida del mismo tipo esperando por otro jugador, lo introducirá en ella. Cuando se encuentre un rival que busca el mismo tipo de partida, empezará la partida para el jugador.
Prueba	**Resultado Esperado**
El usuario identificado pulsa la opción correspondiente, pero la conexión con el servidor falla.	No se creará ninguna partida para el jugador. El sistema le mostrará un mensaje advirtiendo de que no ha sido posible establecer la conexión.
Prueba	**Resultado Esperado**
El usuario identificado pulsa la opción correspondiente, pero no cuenta con un *token* activo o este ha expirado.	No se creará ninguna partida para el jugador. El sistema le advertirá del error y le recomendará acceder de nuevo con sus credenciales.
Prueba	**Resultado Esperado**
El usuario identificado intenta solicitar una partida cuando ya se encuentra jugando una partida multijugador.	El sistema lo detecta y le mostrará un mensaje de error diciéndole que no es posible jugar varias partidas multijugador de forma simultánea.
Prueba	**Resultado Esperado**
El usuario identificado modifica los parámetros de configuración que se envían junto con la petición, empleando las herramientas de desarrollador.	Si los parámetros introducidos representan opciones de configuración válidas, el sistema creará la partida de la forma solicitada. Si los parámetros no son válidos, la partida multijugador no se creará y el usuario permanecerá en el menú principal.

Caso de Uso 10: Escoger límite de tiempo	
Prueba	**Resultado Esperado**
El usuario identificado escoge una de las opciones disponibles en la interfaz.	Cuando busque partida, el sistema deberá asegurarse de que entrará en una partida del tipo exacto seleccionado.
Prueba	**Resultado Esperado**
El usuario cambiar, empleando las herramientas de desarrollador, el tipo de partida que va a pedir.	Si el tipo de partida no es válido, no se creará ninguna partida para el jugador. Si el tipo de partida es válido, el sistema se asegurará de nuevo de que entre en una partida del mismo tipo que ha elegido.

Caso de Uso 11: Jugar Partida	
Prueba	**Resultado Esperado**
El jugador realiza todos los movimientos de forma correcta, empleando el tablero y el árbol interactivos, y sin agotar la limitación de tiempo si la hubiera.	El sistema garantizará que el juego se desarrolle como debe, controlando toda la lógica de turnos, puntuaciones, reglas del juego, etc. Cuando no sea posible realizar más movimiento, el sistema decidirá quien ha ganado en función de las cajas cerradas, mostrando al ganador y al perdedor los mensajes de victoria y derrota, respectivamente. Estos mensajes irán acompañados de la puntuación perdida o ganada, si se trata de una partida clasificatoria. La secuencia de movimientos de esta partida se guardará en la base de datos.
Prueba	**Resultado Esperado**
El jugador realiza los movimientos de forma correcta, pero agota su tiempo por partida o por movimiento.	El sistema será capaz de detectar esto, ejecutando la lógica correspondiente a cada tipo de partida para cuando esta acaba. El proceso se realizará de forma normal, repartiendo puntuaciones si procediese. La secuencia de movimientos no se guardará en la base de datos.
Prueba	**Resultado Esperado**
El jugador abandona la partida antes de que termine empleando la opción de salir al menú principal.	El sistema será capaz de detectar esto, ejecutando la lógica correspondiente a cada tipo de partida para cuando esta acaba. El proceso se realizará de forma normal, repartiendo puntuaciones si procediese. La secuencia de movimientos no se guardará en la base de datos. Por otra parte, la sesión del usuario no se cerrará y este podrá volver a jugar otra partida desde el menú principal.
Prueba	**Resultado Esperado**
El jugador abandona la partida antes de que termine cerrando el navegador.	El sistema será capaz de detectar esto, ejecutando la lógica correspondiente a cada tipo de partida para cuando esta acaba. El proceso se realizará de forma normal, repartiendo puntuaciones si procediese. La secuencia de movimientos no se guardará en la base de datos. Por otra parte, la sesión del usuario se cerrará y su *token* será cancelado. Si vuelve a abrir la aplicación, podrá acceder con sus credenciales de nuevo.
Prueba	**Resultado Esperado**
El usuario pierde la conexión con el servidor.	El sistema será capaz de detectar esto, ejecutando la lógica correspondiente a cada tipo de partida para cuando esta acaba. El proceso se realizará de forma normal, repartiendo puntuaciones si procediese. La secuencia de movimientos no se guardará en la base de datos. Si recupera la conexión antes

	de que se supere el tiempo de inactividad de un *token* (15 minutos) podrá seguir con su sesión iniciada, aunque habrá perdido la partida.
Prueba	**Resultado Esperado**
El jugador intenta hacer trampas utilizando las herramientas para desarrolladores o un software externo. Podrá intentar realizar movimientos ilegales o decidir los de su rival.	El sistema será capaz de discernir qué movimientos son reglamentarios y de donde proceden. Si un jugador intenta realizar un movimiento que va contra las reglas del juego o intenta mover fuera de su turno, el sistema lo penalizará con la derrota. De nuevo, ejecutará la lógica correspondiente a cada tipo de partida para cuando esta acaba. El proceso se realizará de forma normal, repartiendo puntuaciones si procediese. La secuencia de movimientos no se guardará en la base de datos.

Caso de Uso 12: Modificar Archivo de Configuración	
Prueba	**Resultado Esperado**
El administrador del sistema modifica uno, varios o todos los parámetros correctamente.	El servidor será capaz de extraer estos parámetros del archivo de configuración y adecuarse a todos ellos.
Prueba	**Resultado Esperado**
El administrador del sistema introducirá errores en uno, varios o todos los parámetros configurables.	El servidor será capaz de extraer del archivo todos aquellos parámetros que estén bien formados y adaptarse a ellos. Para los que, por alguna razón, no puedan ser leídos, se aplicarán las configuraciones por defecto establecidas en el código de los distintos subsistemas. El sistema avisará del error y se extraerá el mensaje a un archivo de registro.
Prueba	**Resultado Esperado**
El administrador borrará o cambiará la localización del archivo de configuración.	El servidor detectará el fallo, empleando las configuraciones por defecto en todos los casos. El sistema avisará del error y se extraerá el mensaje a un archivo de registro.

Caso de Uso 13: Modificar el Contenido de la Base de Datos	
Prueba	**Resultado Esperado**
El administrador del sistema accederá a la interfaz *web* de MongoDB para insertar un usuario.	El sistema también permitirá acceder con las credenciales de esta nueva cuenta.
Prueba	**Resultado Esperado**
El administrador del sistema accederá a la interfaz *web* de MongoDB para modificar un usuario.	Los datos que mostrará el sistema para esta cuenta serán los nuevos datos introducidos, tanto en el perfil como en el *ranking* de jugadores. Además, de haber cambiado las credenciales, las anteriores quedarán inservibles y solo será posible acceder con las nuevas.
Prueba	**Resultado Esperado**
El administrador del sistema accederá a la interfaz *web* de	El sistema aplicará los cambios: las credenciales de esta antigua cuenta ya no podrán usarse para iniciar sesión en la

MongoDB para borrar un usuario.	aplicación y, tanto su nombre de usuario como su email, volverán a estar disponibles para usar durante un proceso de registro.

Capítulo 6. Diseño del Sistema

Este capítulo expondrá el diseño del sistema por medio de diferentes estrategias: diagramas de paquetes, de clases y de diversos tipos más; explicaciones textuales de todo aquello que no sea representado en los diagramas o que pueda no quedar claro; etc.

6.1 Arquitectura del Sistema

Esta sección tratará de detallar la arquitectura del sistema a través de diagramas de paquetes (Fig. 6.1), componentes (Fig. 6.2) y despliegue (Fig. 6.3). Cada uno de ellos sucedido de varios subapartados en los que se realizan las explicaciones pertinentes.

6.1.1 Diagrama de Paquetes

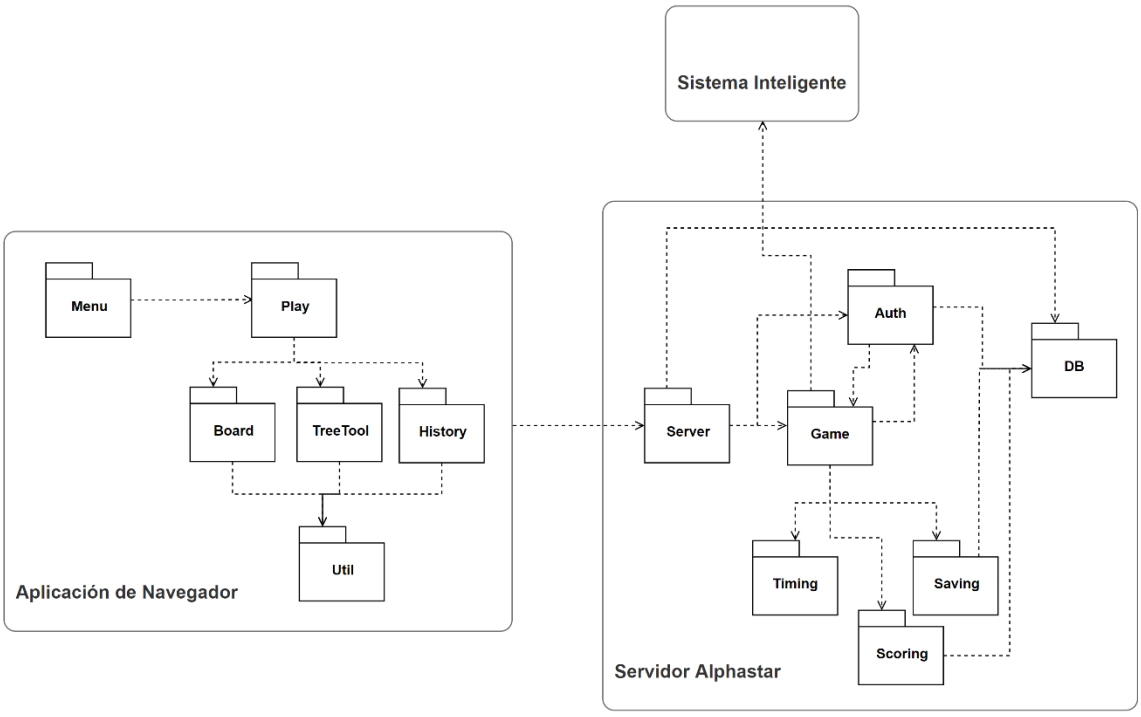


Figura 6.1. Diagrama de paquetes del sistema

6.1.1.1 Menu

Este paquete contendrá la implementación de todas las interfaces de usuario relacionadas con los menús de la aplicación *web*, el perfil y el *ranking* de jugadores. Estas interfaces serán divididas en tantos componentes como sea necesario. Además, dentro de este paquete se establecerá la conexión con el servidor para conseguir un *token* de autenticación, insertar usuarios nuevos, o pedir la información del perfil y *ranking* de jugadores. También desde este

paquete se solicitarán las nuevas partidas al servidor cuando un jugador quiera iniciar una, sea del tipo que sea.

6.1.1.2 Play

De forma similar, aquí estarán contenidas las interfaces a las que accederá el jugador una vez esté jugando la partida: la propia pantalla de juego, la pantalla de partida finalizada, etc. En este paquete se implementará toda la lógica necesaria del cliente para que el jugador realice sus movimientos, reciba la información del servidor y pueda analizar esa información. Además, será donde se realice la comunicación con el servidor una vez haya empezado una partida: para enviar y recibir información sobre movimientos, recomendaciones, eventos en la partida, etc. También será donde se decidirá qué modo de visualización de la información se muestra (tablero y árbol interactivos, o historial), actualizará estos modos y actuará como observador para detectar si, desde ellos, se ha realizado un movimiento.

6.1.1.3 Board

En este paquete se implementará toda la lógica relativa al tablero interactivo. Sabrá cómo representar la información que reciba del paquete Play y generará los eventos necesarios para que sean capturados por este último cuando se realice un movimiento.

6.1.1.4 TreeTool

Aquí se implementará todo lo que tenga que ver con el árbol interactivo. Tendrá que ser capaz de dibujar este árbol en función de los datos que le envíe el paquete *Play*, implementará los métodos necesarios para navegar por el árbol, la lógica de todas las consultas que harán posible analizarlo, y la forma de representar las recomendaciones y sus explicaciones textuales. Además, como el anterior, será donde se generen los eventos de movimiento realizado que se capturarán desde el paquete *Play* y, posteriormente, se enviarán al servidor.

6.1.1.5 History

Este paquete contendrá lo necesario para representar la información que reciba de la secuencia de movimientos durante toda la partida. Permitirá que el jugador inspeccione todos estos estados, mostrándolos en orden temporal.

6.1.1.6 Util

Aquí se implementarán utilidades que sean comunes a todos los paquetes anteriores. Por ejemplo, la función que permita representar traducir un vector de números a un tablero de *Dots and Boxes*, ya que tanto el tablero, como el árbol y el historial, requerirán de esta funcionalidad.

6.1.1.7 Server

Será el paquete donde se implementen todas aquellas clases que actuarán de intermediarias entre la aplicación de navegador y la lógica de partidas o de gestión de usuarios y autenticación. Básicamente, a las clases de este paquete llegarán todas las peticiones a través de red (mediante protocolo HTTP o WebSockets) y serán traducidas a métodos de los distintos subsistemas. También serán contenidos aquí algunos servicios que no cuadren en los demás paquetes y sean demasiado simple como para constituir otro paquete. Dos ejemplos de lo anterior serían los servicios que permiten obtener la información del perfil de usuario y el *ranking* de jugadores.

6.1.1.8 Auth

En este paquete estarán todas las clases encargadas de la validación de datos de acceso, identificación de usuarios, registro de nuevas cuentas, todas las funcionalidades relacionadas con los *tokes* de autorización, etc.

6.1.1.9 DB

Aquí se guardarán todas esas clases cuya función sea realizar la mediación entre los distintos subsistemas de la aplicación y la base de datos: repositorios, entidades DTO (siglas en inglés de Objetos para la Transferencia de Datos), cliente de base de datos, etc.

6.1.1.10 Game

Un paquete en el que estará contenido toda la lógica de las partidas, ya sean multijugador o de entrenamiento. Deberá ser quien implemente la lógica interna del juego, la lógica de turnos, la lógica de derrotas, etc. Las clases en este paquete serán las únicas que acudirán al sistema que contiene el algoritmo inteligente.

6.1.1.11 Timing

Aquí serán contenidas todas las clases que implementen las distintas estrategias para la limitación de tiempos en una partida, además de la interfaz que todas ellas implementan. Si apareciesen más tipos de limitaciones de tiempo en una partida, serían implementadas dentro de este mismo paquete.

6.1.1.12 Scoring

Contendrá las distintas estrategias para calcular la puntuación que se debe añadir o restar a los jugadores al final de una partida. Cada una de estas estrategias aparecerá encapsulada en una clase que implementará la interfaz principal. Si se decidiese implementar otra forma de calcular los puntos al final de la partida, también se incluiría en este paquete.

6.1.1.13 Saving

En este paquete estarán las distintas estrategias para guardar una partida al final de esta, así como la interfaz que todas deben implementar. En principio habrá dos, para guardar en base de datos MongoDB y para guardar en un archivo JSON, y se utilizará por defecto para todas las partidas la de guardar en base de datos. Si fuese necesario, aquí se implementarían otras estrategias, como la de guardar a otro tipo de base de datos o a ficheros con formatos distintos.

6.1.2 Diagrama de Componentes

A continuación, se puede ver la Figura 6.2, un diagrama de componentes en el que aparecen todos los módulos del sistema. Cada uno de ellos se describirá brevemente más tarde.

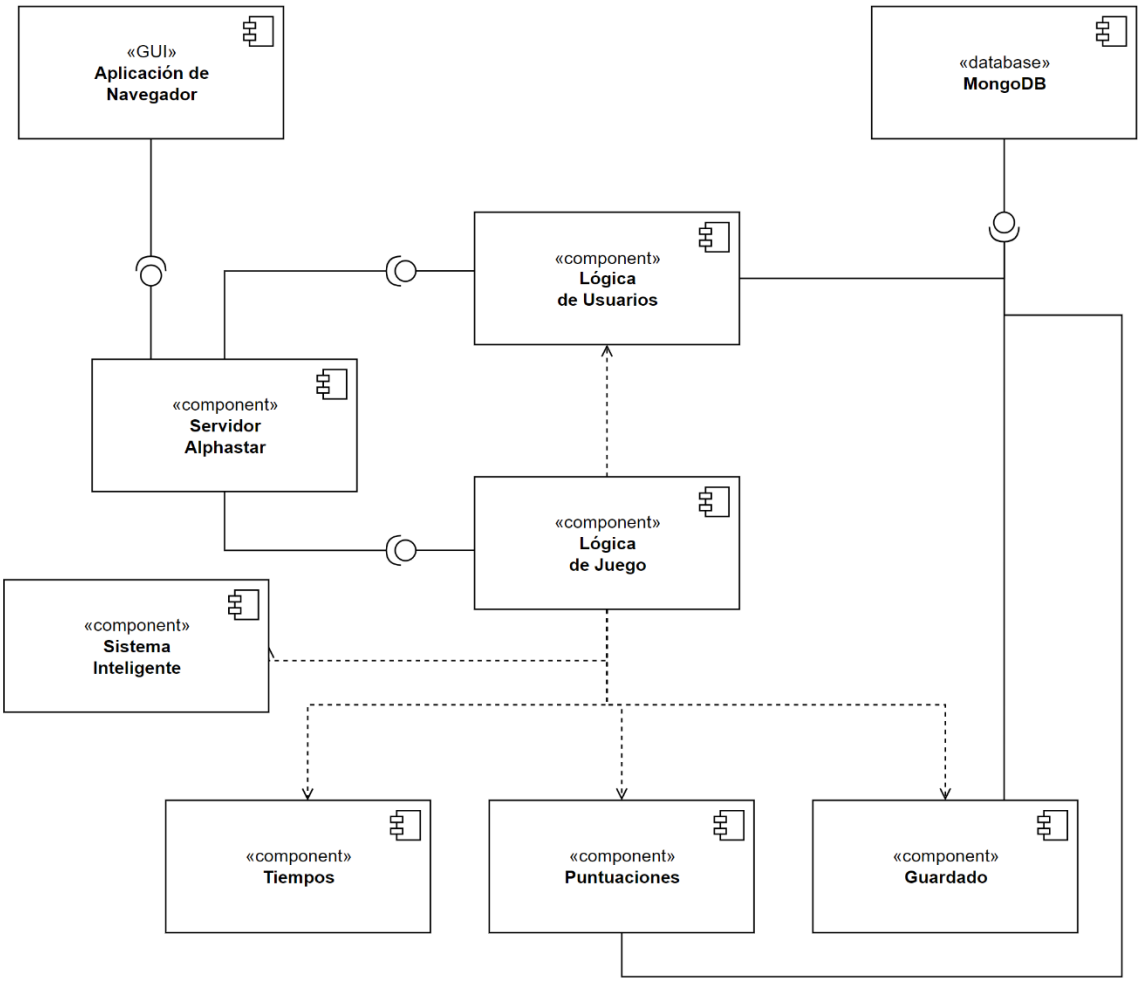


Figura 6.2. Diagrama de componentes del sistema

La aplicación de navegador comprende todo el código que se desarrollará en Angular para la interfaz de usuario *web*, así como los métodos de comunicación necesarios para que esta GUI (siglas en inglés de Interfaz Gráfica de Usuario) se comunique con el servidor.

En el servidor Alphastar entra toda la parte de código del servidor que manejará y hará posibles las comunicaciones entre la interfaz de usuario, desde el navegador de cada uno, y el resto del sistema.

El componente de lógica de usuario es el que se encarga de todo lo relacionado con la identificación, autorización y autenticación de usuarios. Así como de validar las comunicaciones.

En la lógica de juego entra todo aquello que se debe implementar para que una partida consiga llevarse a cabo, de inicio a final, tal como se espera. Recurrirá al sistema inteligente parar generar los movimientos del rival virtual y las recomendaciones para el usuario. A parte, dispondrá de los distintos componentes que permiten las funcionalidades para establecer un límite de tiempo para los movimientos de los jugadores, calcular las puntuaciones al final de una partida y guardar las secuencias de movimientos sucedidas a lo largo de una partida.

Finalmente, el componente MongoDB hace referencia, evidentemente, al código necesario para realizar las operaciones sobre la base de datos. Para ello, se utilizará la API proporcionada por MongoDB.

6.1.3 Diagrama de Despliegue

Aquí se representarán, mediante un diagrama (Fig. 6.3), todos aquellos procesos software y máquinas que intervendrán en el funcionamiento de la aplicación, así como sus conexiones.

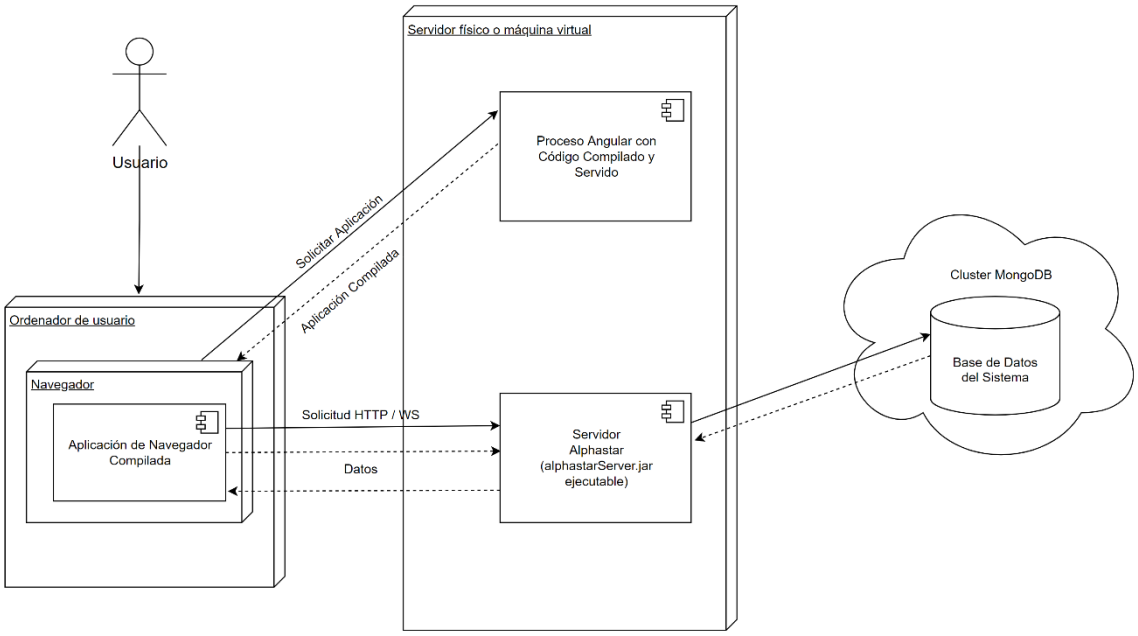


Figura 6.3. Diagrama de despliegue del sistema

6.1.3.1 Servidor

Podrá ser tanto una máquina física, como una máquina virtual. El software estará desarrollado para poder ser ejecutado en tanto en máquinas Linux como en máquinas Windows. Deberá tener en su interior los dos procesos que se muestran en el diagrama y, al menos, contará con dos puertos abiertos: uno para cada proceso.

6.1.3.2 Proceso Angular

Este proceso compilará todo el código desarrollado en Angular a una aplicación de una sola página. Además, podrá servir el código HTML, JS y CSS generado para ese fin, a cualquier que lo solicite mediante una petición HTTP.

6.1.3.3 Proceso Servidor Alphastar

Un archivo Java ejecutable que implementará toda la lógica de la aplicación que deba ejecutarse en el servidor. Enviará y recibirá datos a los navegadores de los usuarios a través de una serie de peticiones HTTP y WebSockets definidas. También será el único en comunicarse con la base de datos.

6.1.3.4 Base de Datos del Sistema

Como ya se ha comentado, esta estará alojada en la nube que ofrece MongoDB. Aquí se guardarán todos los datos necesarios, que el servidor leerá y escribirá cuando sea necesario.

6.1.3.5 Aplicación de Navegador

Este será el código desarrollado en Angular, una vez haya sido compilado, servido y esté siendo ejecutado en el navegador del ordenador personal de un usuario. Los únicos requerimientos para acceder a la aplicación es disponer de una máquina con un navegador moderno con interfaz gráfica de usuario.

6.2 Diseño de Clases

Aquí se representarán, empleando diagramas, las clases que formarán parte de la implementación final del sistema y sus relaciones. Este diseño es una evolución de lo conseguido durante la fase de análisis, por lo que resulta fácil ver la relación entre ellas. Aunque todas las clases aquí representadas estarán, de una manera u otra, en el sistema final, es posible que durante la fase de implementación aparezcan nuevas clases auxiliares que también deberán ser desarrolladas para conseguir las finalidades buscadas.

6.2.1 Diagramas de Clases

En el primer diagrama (Fig. 6.4) podemos observar el diagrama de clases de la aplicación de navegador. Aunque técnicamente en Angular no existen las clases, hay componentes (normalmente formados por un archivo TypeScript, otro HTML y otro CSS), que representan son un concepto similar. Lo que en el diagrama aparece como clases, son los archivos de código TypeScript de cada componente. Por otra parte, la clase *Tree* del diagrama es parte del código de una librería (*D3Tree)*, pero adaptado a nuestro proyecto concreto, para conseguir todas las funcionalidades que se requieren.

El resto de los diagramas de clases (Figuras 6.5, 6.6 y 6.7) representarán las clases identificadas para el resto del sistema. Cada diagrama muestra las clases contenidas en uno o varios de los paquetes representados anteriormente. En la Figura 6.6 se puede observar cómo *AbstractServerGame* contiene un atributo de tipo *IMCTS*, esa es la clase que sirve como interfaz para acceder a las funcionalidades que implementa el mentado sistema inteligente.

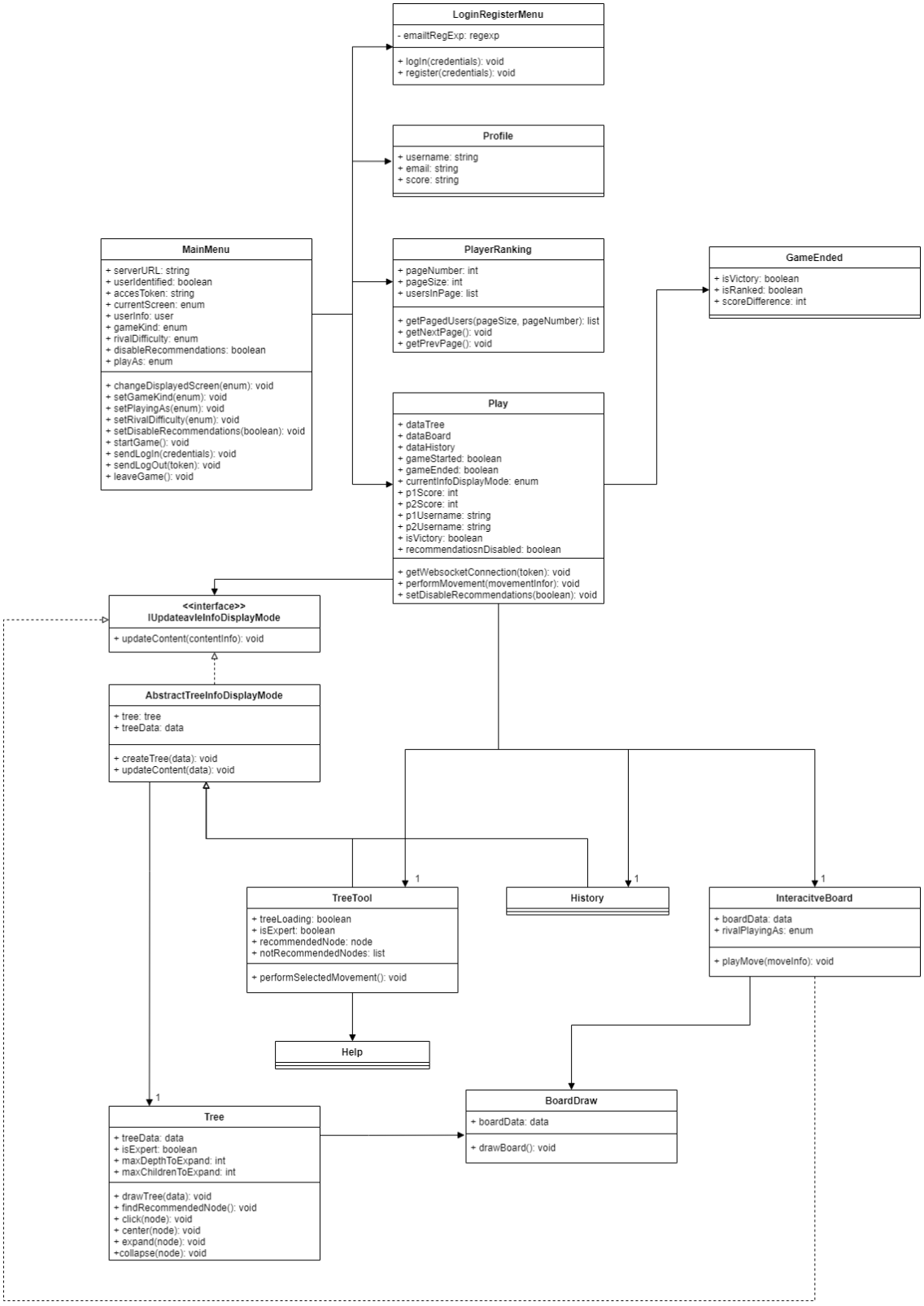


Figura 6.4. Diagrama de clases de la aplicación para navegador

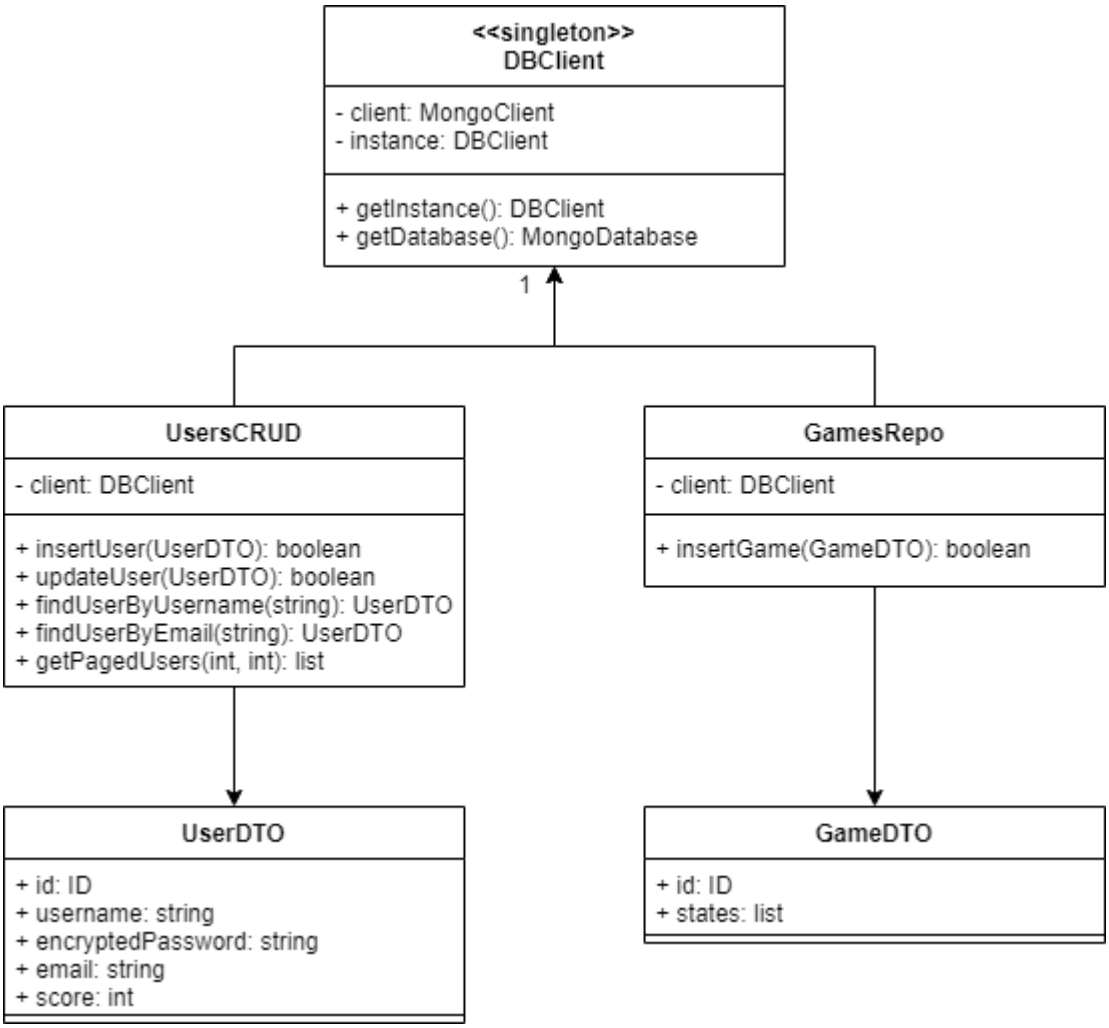


Figura 6.5. Diagrama de clases del paquete DB

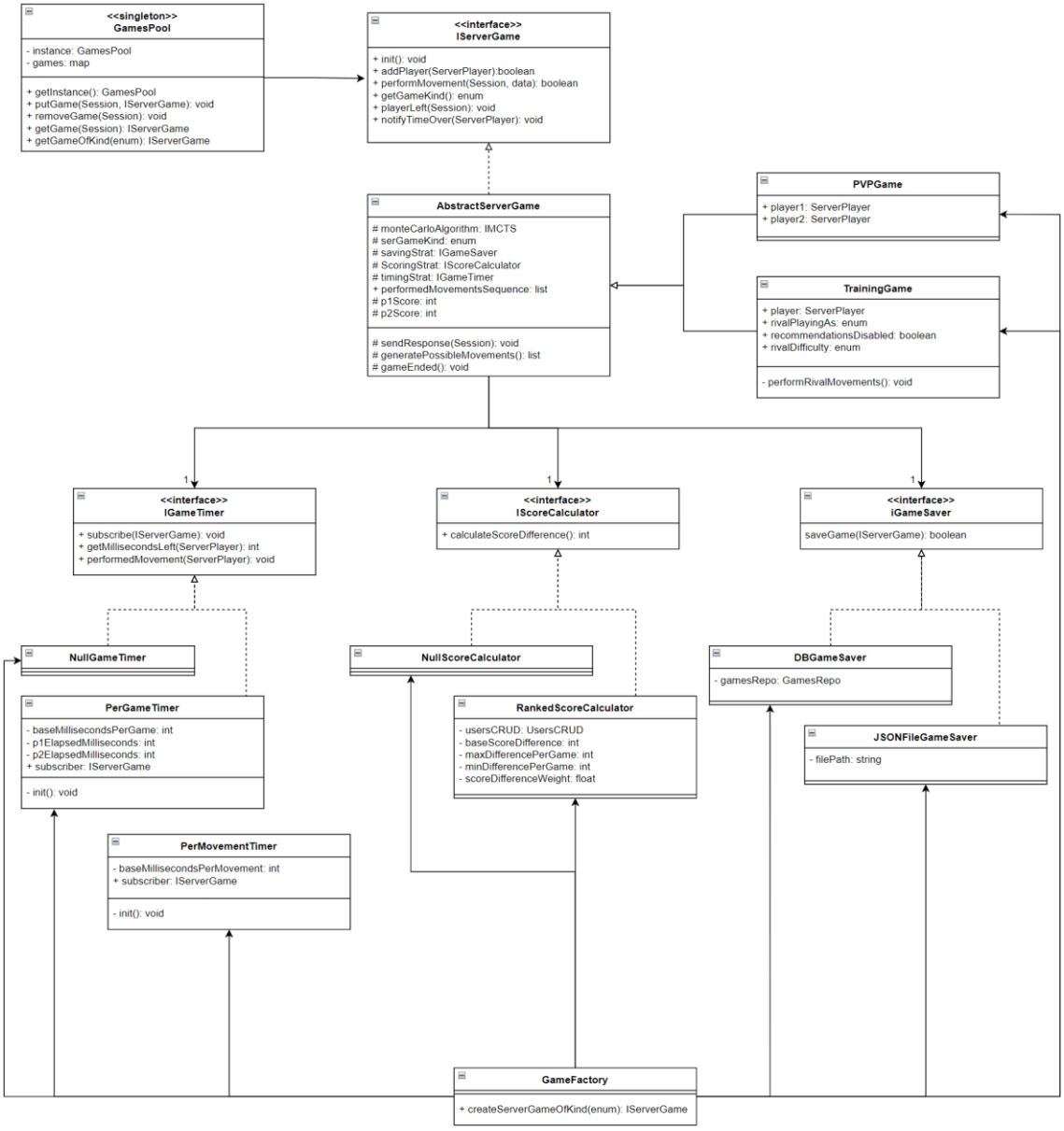


Figura 6.6. Diagrama de clases conjunto de los paquetes Game, Scoring, Timing y Saving

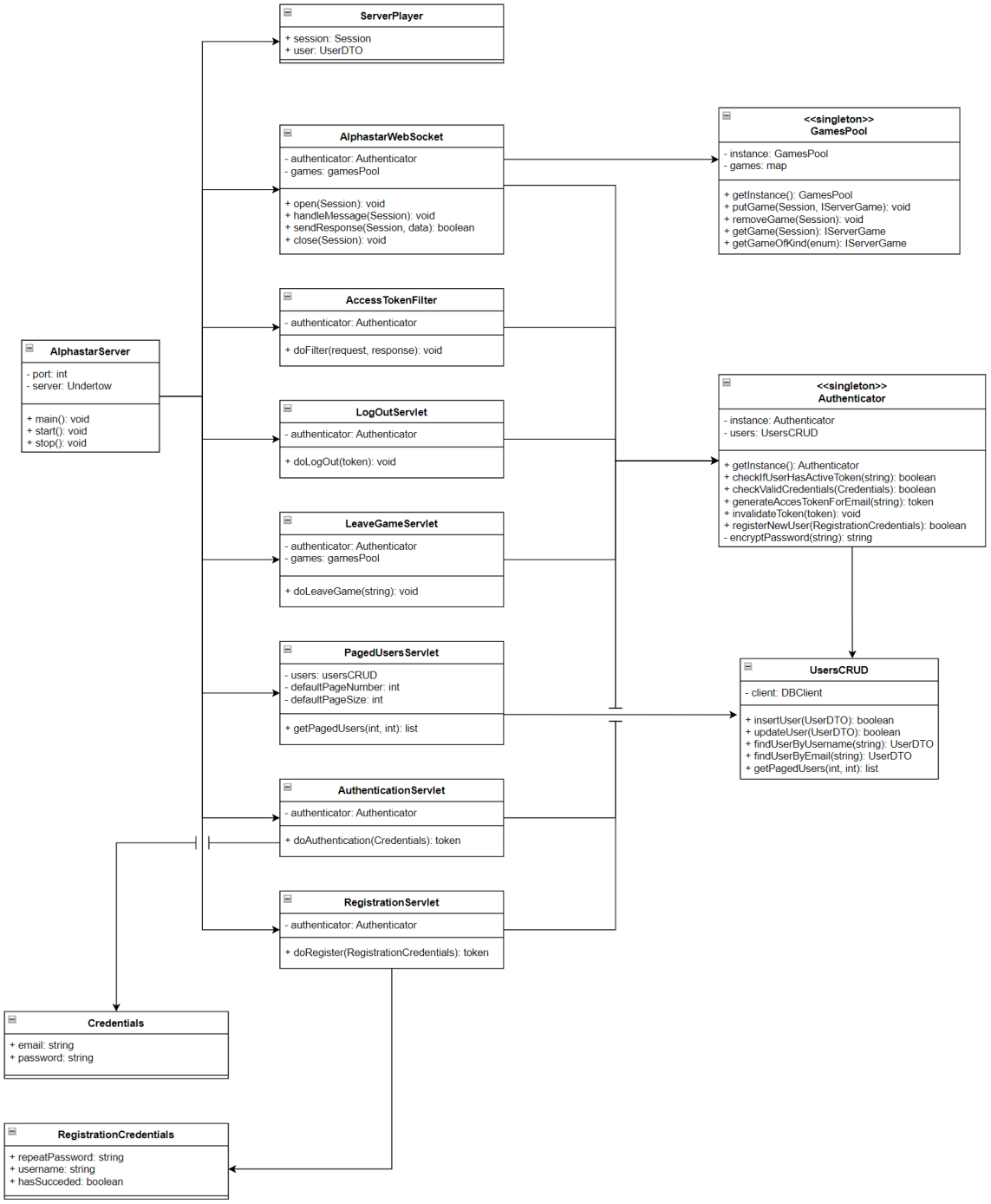


Figura 6.7. Diagrama de clases conjunto de los paquetes Server y Auth

6.3 Diagramas de Interacción y Estados

A continuación, se presentarán una serie de diagramas de interacción para ayudar a comprender cómo se comporta el sistema en conjunto y las partes que intervienen ante un determinado caso de uso. Únicamente se incorporarán aquellos más significativos y que ayuden a mejorar la comprensión del sistema, evitando generar diagramas clónicos. Por ejemplo, la comunicación en el caso de uso de un registro es muy similar a la de acceso con credenciales: solo cambia el *servlet* encargado de gestionar la petición y que, además, se inserta un nuevo usuario en la base de datos.

Tampoco se incluirán diagramas de estados, ya que, en este caso, no aportarían mucha información extra. Por la forma en la que se ha diseñado el sistema, cada clase implementará una única funcionalidad. El único cambio de estado relevante será, presumiblemente, el de las clases que implementen *IServerGame*, que, cuando presenten el estado de "terminadas", no permitirán que se ejecute su lógica de juego, únicamente extraer información de ellas.

6.3.1 Caso de Uso: Acceder Mediante Credenciales (*Log In*)

6.3.1.1 Diagrama de Interacción (Comunicación y Secuencia)

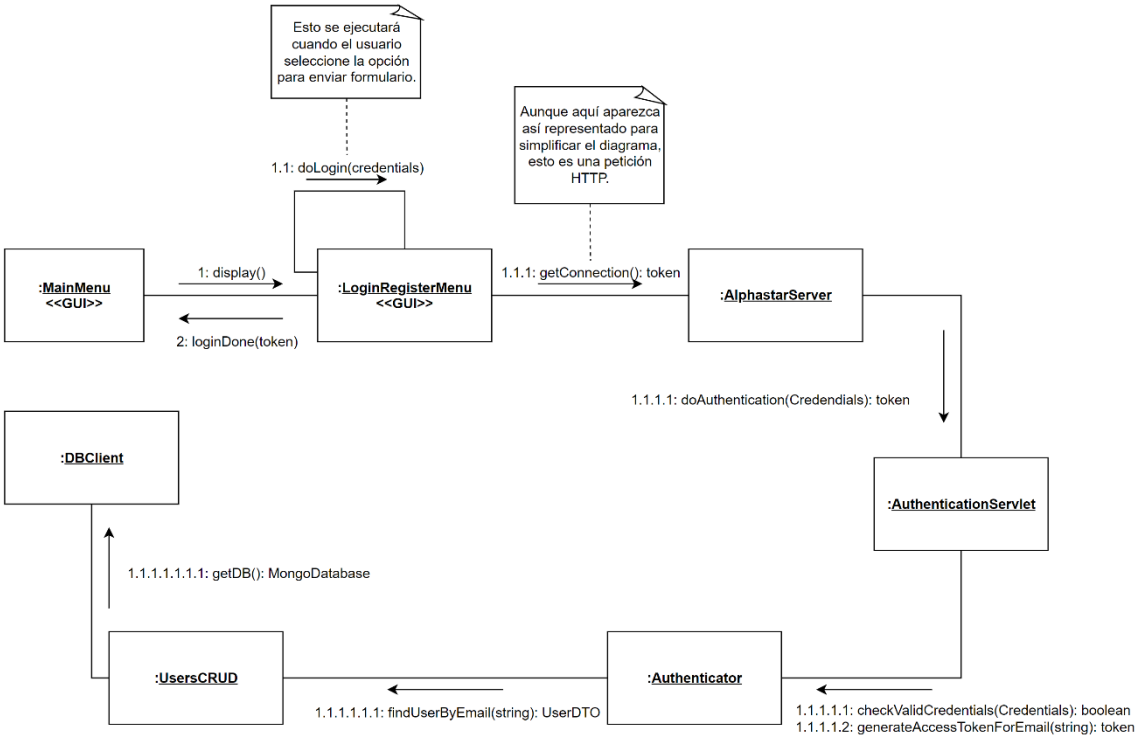


Figura 6.8. Diagrama de interacción para el caso de uso de Log In

6.3.2 Caso de Uso: Cerrar sesión

6.3.2.1 Diagrama de Interacción (Comunicación y Secuencia)

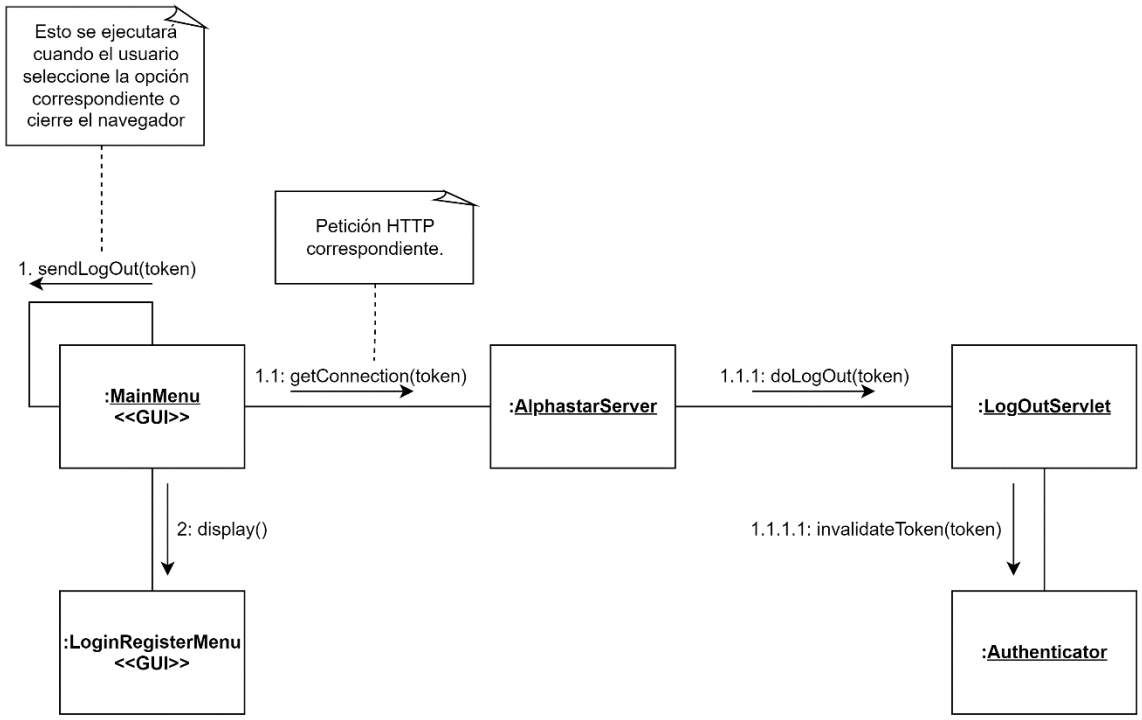


Figura 6.9. Diagrama de interacción para el caso de uso de cerrar sesión

6.3.3 Caso de Uso: Ver Ranking de Jugadores

6.3.3.1 Diagrama de Interacción (Comunicación y Secuencia)

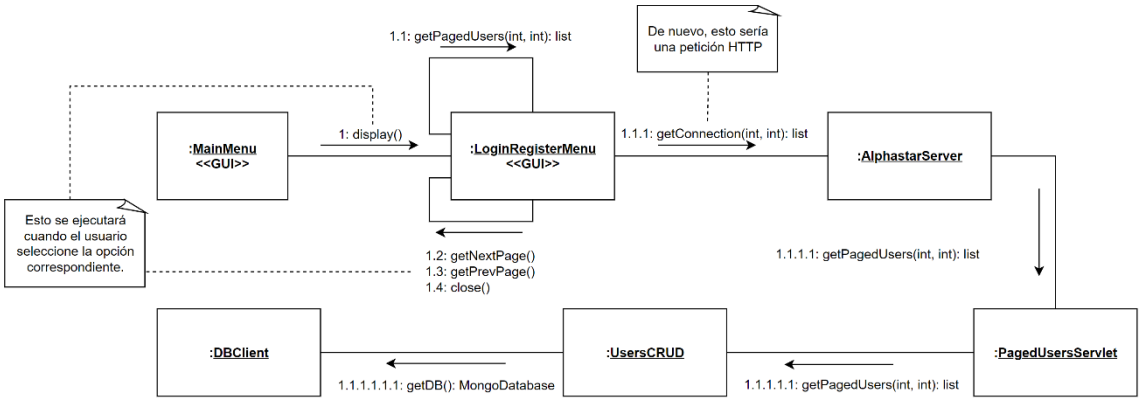


Figura 6.10. Diagrama de interacción para el caso de uso de ver ranking de jugadores

6.3.4 Caso de Uso: Buscar y Conectarse a una Partida

6.3.4.1 Diagrama de Interacción (Comunicación y Secuencia)

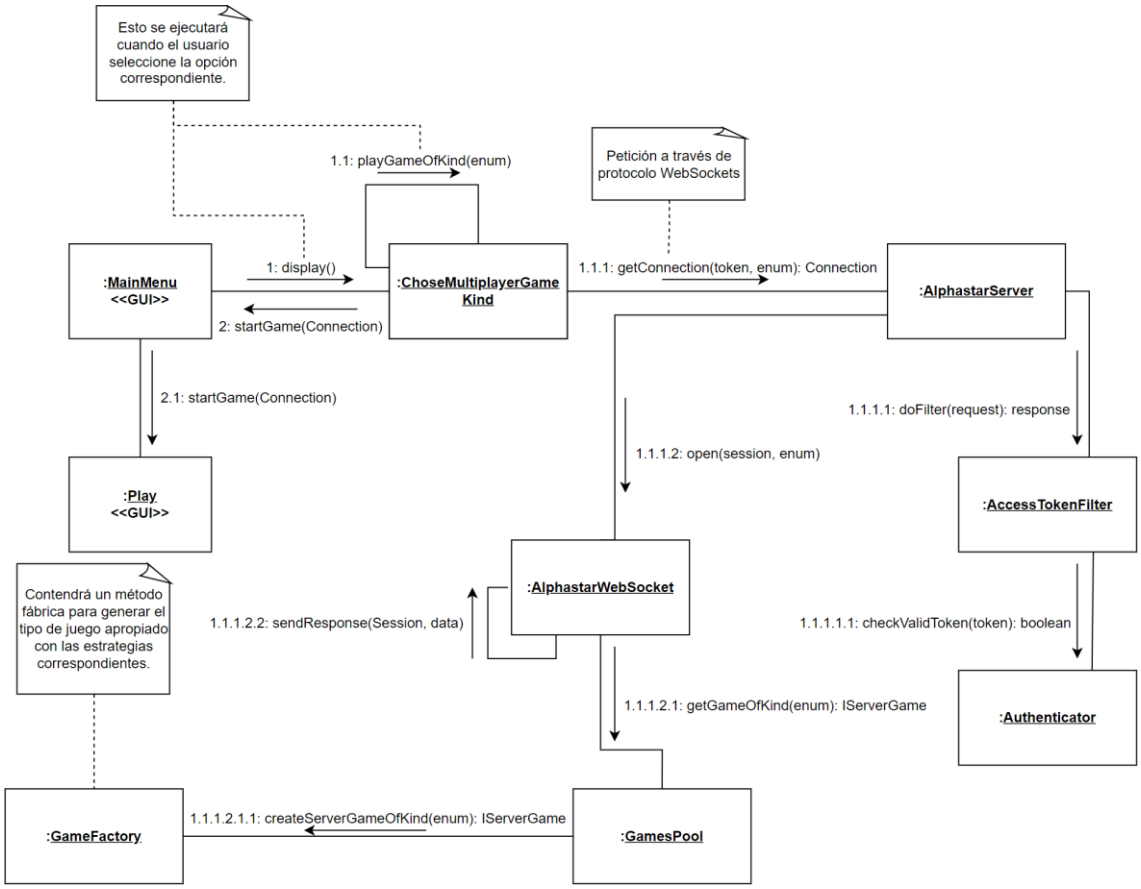


Figura 6.11. Diagrama de interacción para el caso de uso de buscar y conectarse a una partida

6.4 Diseño de la Base de Datos

En esta sección se comentará la base de datos empleada, así como su integración en nuestro sistema y la estructura que sigue.

6.4.1 Descripción del SGBD Usado

Como SGDB se ha decidido emplear MongoDB, un gestor basado en ficheros, ya que se adapta a las necesidades de proyectos como este. Es rápido y permite realizar todo tipo de consultas; implementa una API muy completa y de uso sencillo y transparente, y, como es no relacional, permite almacenar datos semiestructurados, estableciendo un esquema flexible y adaptable a necesidades futuras. También tiene la desventaja de que no soporta transacciones complejas, pero esto no es un problema, ya que no serán necesarias en nuestro caso.

Además, MongoDB permite almacenar la base de datos en un *cluster* alojado en su nube. A parte de evitar tareas como tener que mantener una copia de seguridad, esto permite tener en todo momento un servidor de base de datos que se adapte a las necesidades concretas de un sistema. Así, ese sistema, mientras almacene pocos datos y no requiera de gran capacidad de computación, podrá mantener unos gastos mínimos asociados a la base de datos. Y, si llegado el momento, el volumen de datos y operaciones aumenta, solo es necesario elegir el *tier* (uno de los distintos niveles que ofrecen) adecuado.

La versión concreta de MongoDB empleada para esta aplicación es la 4.4.11.

6.4.2 Integración del SGBD en Nuestro Sistema

Todo lo necesario para la integración de la base de datos MongoDB con el sistema se encuentra en el paquete DB (en la Figura 6.5 hay un pequeño diagrama de las clases que incluye). En resumen, aquí se implementarán todos los repositorios, DTOs y clases auxiliares necesarias para realizar lecturas y escrituras en la base de datos. Estas clases emplearán, para ello, la API de MongoDB para Java, además de BSON como lenguaje para la transferencia de datos.

6.4.3 Formato de Datos

Aunque MongoDB es una base de datos no relacional, aquí se incluirá un diagrama mostrando el formato que tendrán todos los documentos en cada una de las dos colecciones que habrá en la base de datos (Figura 6.12).

Los objetos de tipo *State* no tienen identificador, ya que no existirán fuera de la partida correspondiente, únicamente como parte de ella.

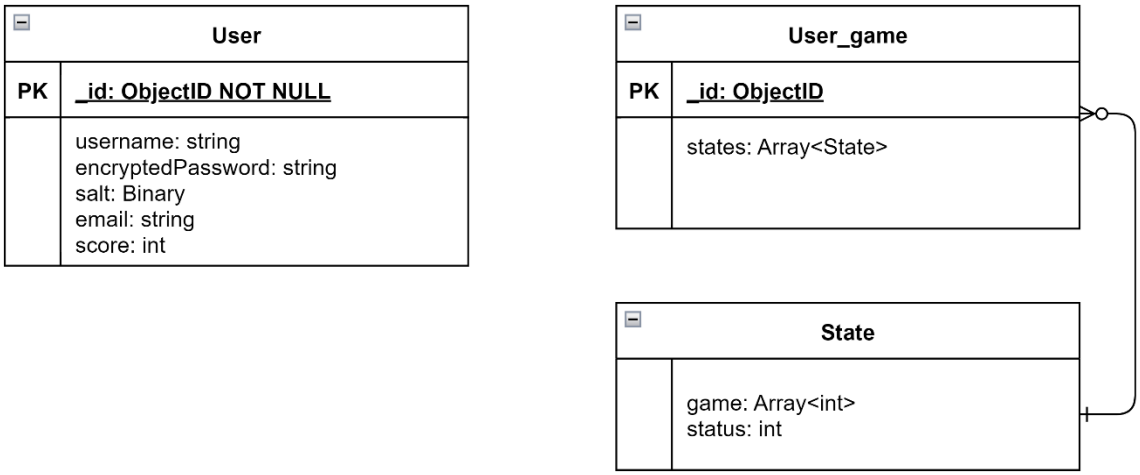


Figura 6.12. Diagrama del formato que seguirá la base de datos

6.5 Diseño de la Interfaz

A continuación, se mostrará la interfaz definitiva de la aplicación, desarrollada a partir de los diseños presentados en el capítulo de análisis. Además, se explicarán todas aquellas partes o elementos que lo requieran.

6.5.1 Interfaz de Menú de Acceso

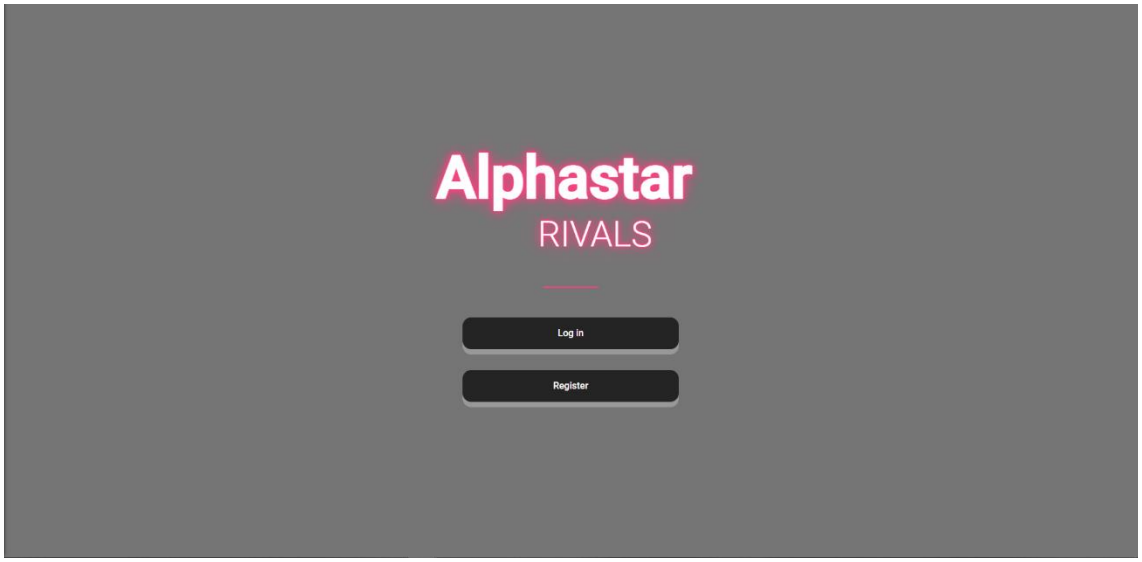


Figura 6.13. Diseño final del menú de acceso

6.5.2 Interfaz de *Log In*

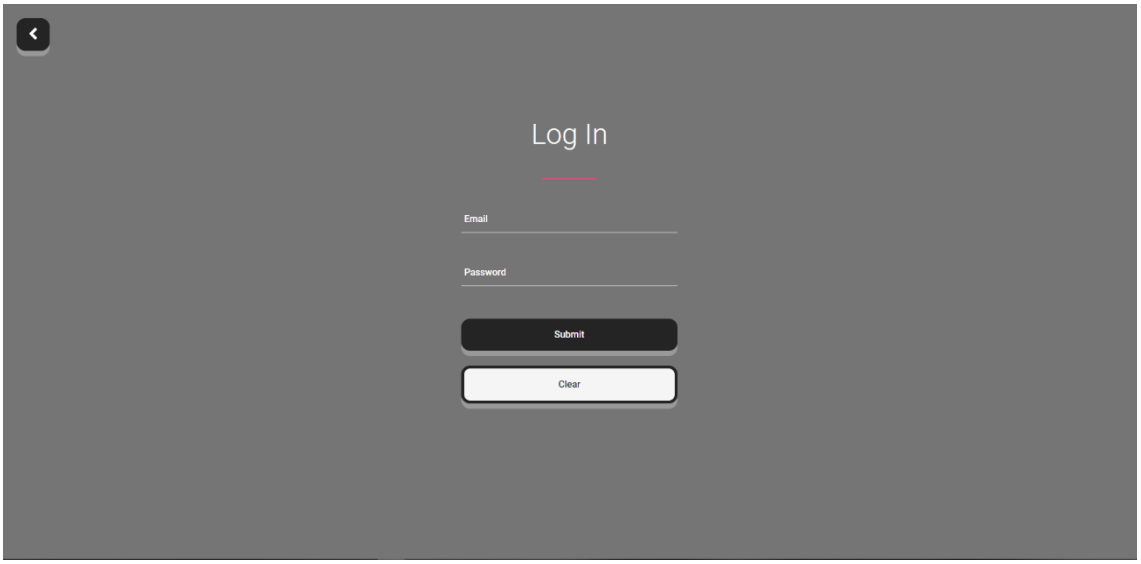


Figura 6.14. Diseño final de la pantalla de log in

6.5.3 Interfaz de Registro

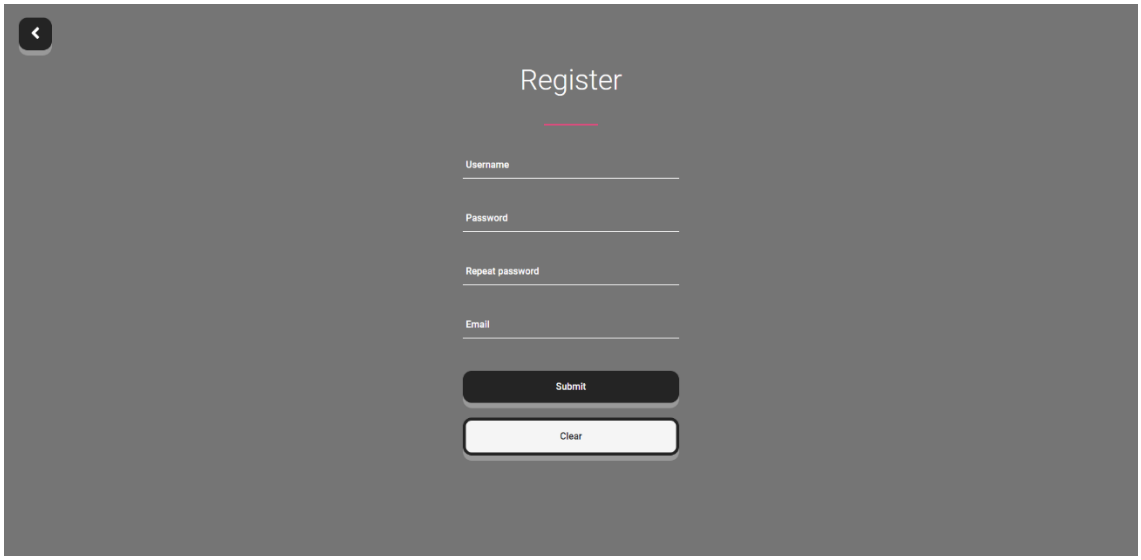


Figura 6.15. Diseño final de la pantalla de registro

6.5.4 Interfaz de Menú Principal

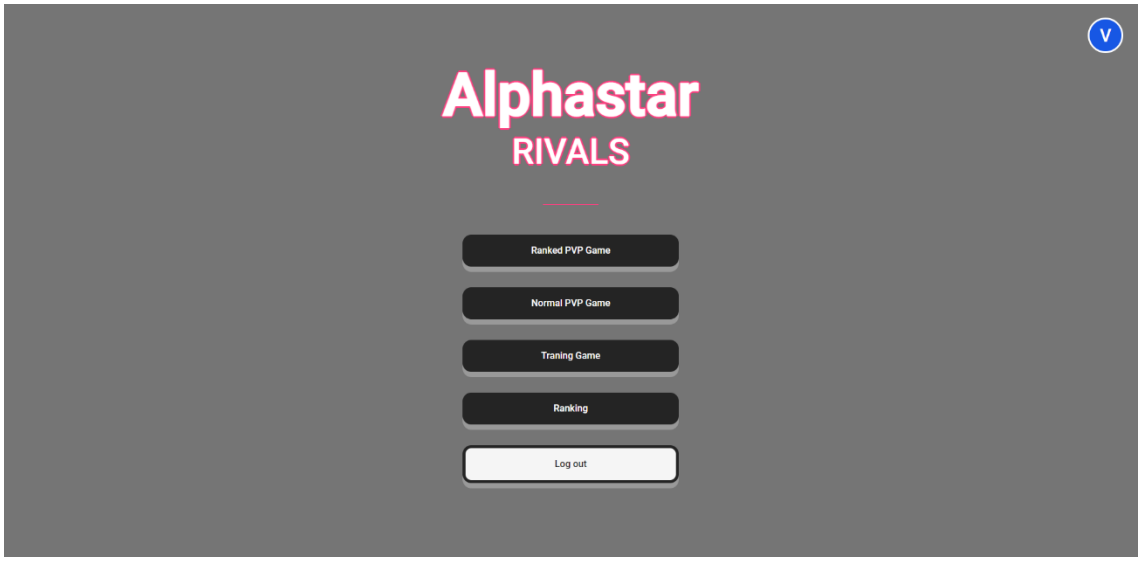


Figura 6.16. Diseño final del menú principal

6.5.5 Interfaz de Selección de Tipo de Partida

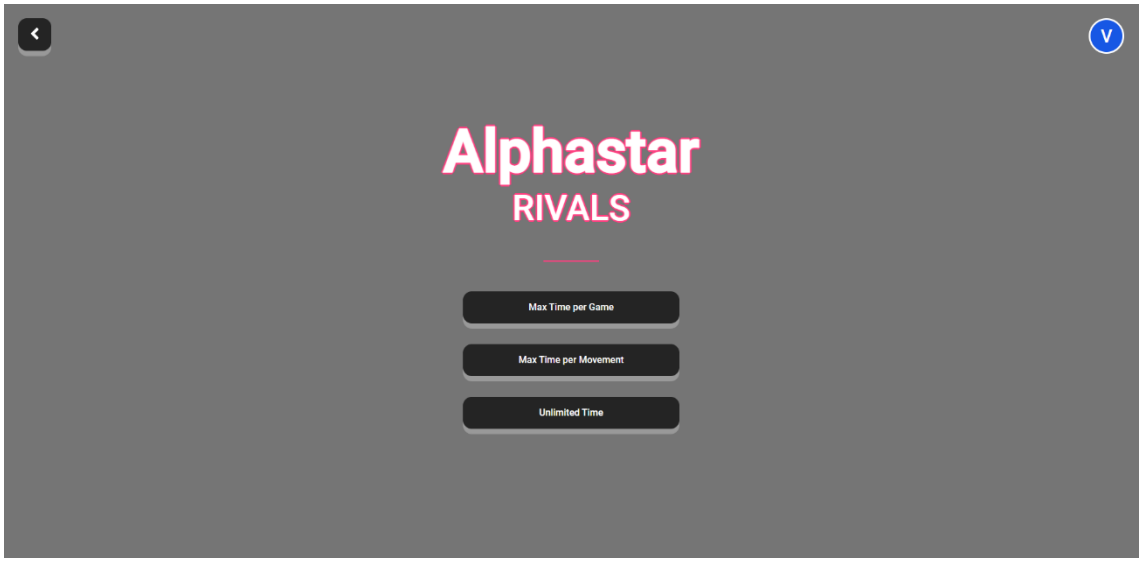


Figura 6.17. Diseño final del menú de selección de tipo de partida

6.5.6 Interfaz de Configuración de Partida de Entrenamiento

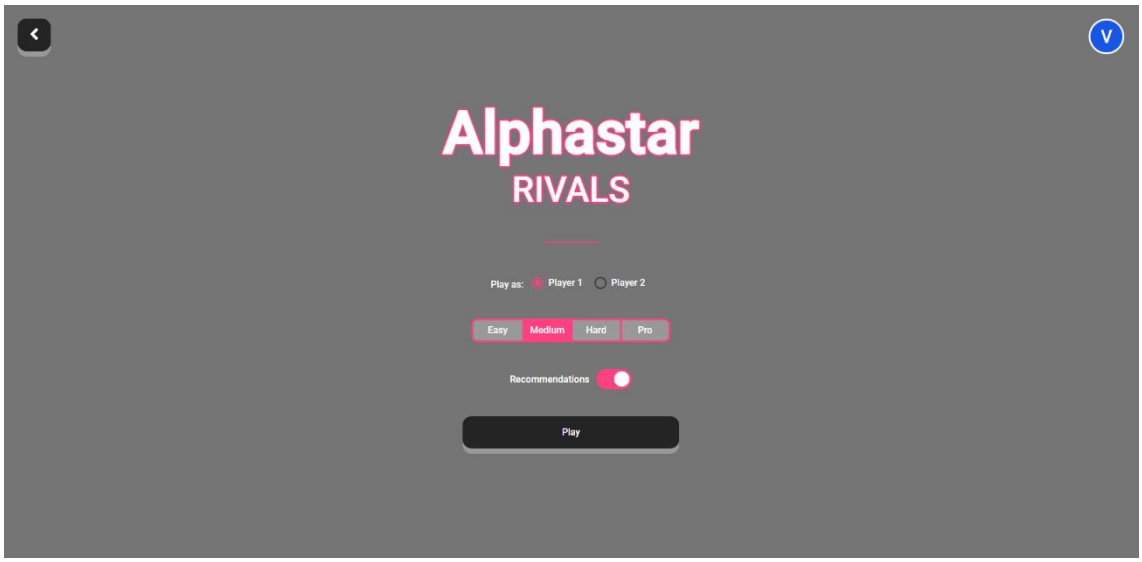


Figura 6.18. Diseño final del menú de configuración de partida de entrenamiento

6.5.7 Interfaz de Ranking de Jugadores

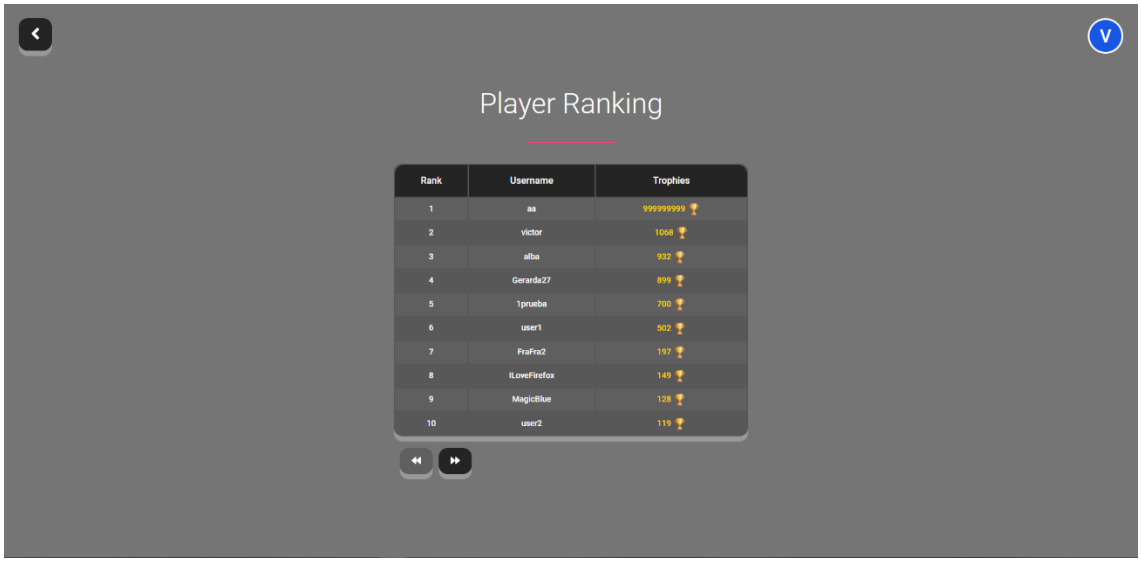


Figura 6.19. Diseño final de la pantalla de ranking de jugadores

6.5.8 Interfaz de Perfil de Usuario

Lo que aparece en la siguiente imagen no es una pantalla en sí misma, sino una sección que aparecerá en forma de cuadro de diálogo flotante en cualquier parte del menú de la aplicación.

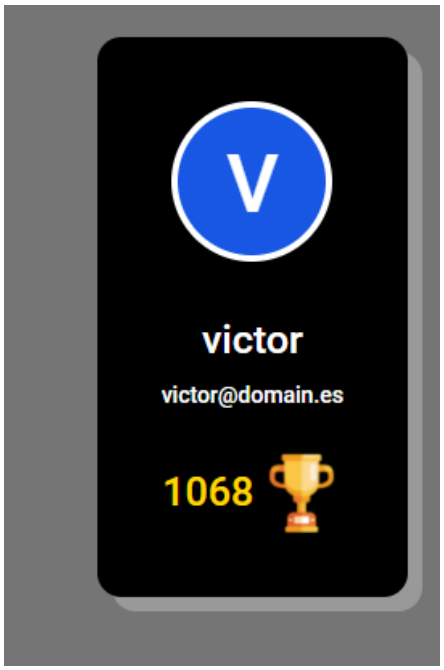


Figura 6.20. Diseño final de la interfaz de perfil de usuario

6.5.9 Interfaz de Pantalla de Juego

Aquí aparecerán varias imágenes para mostrar cómo se vería esta pantalla con sus distintas configuraciones: con temporizador o sin temporizador; mostrando el tablero interactivo; enseñando el árbol de posibles movimientos, o con el historial en el centro.

Además, cuando se está visualizando la información desde la vista de árbol, se podrá elegir entre otros dos modos para estudiar la información que proporciona este: experto y no experto. Desde el primero, el usuario podrá realizar una serie de consultas y expandir y colapsar los distintos nodos a su antojo. En el segundo, se presentará una tabla que indica el movimiento recomendado, junto con su explicación, y otra donde aparecen movimientos poco aconsejables, también junto con una breve explicación. Esas explicaciones constarán de un pequeño texto que se amolde a cada situación particular y de un botón que nos mostrará en el árbol cómo es más posible que se desarrolle una partida a partir de cada uno de los nodos. En ambos modos, cuando se sitúa el cursor encima de un nodo, aparecerá un *popup* con la información del nodo producida por el algoritmo de árbol de Monte Carlo.

También se puede observar cómo, en el menú lateral, está disponible o no la opción de *Tree* (en función de si la partida de entrenamiento o multijugador, respectivamente).

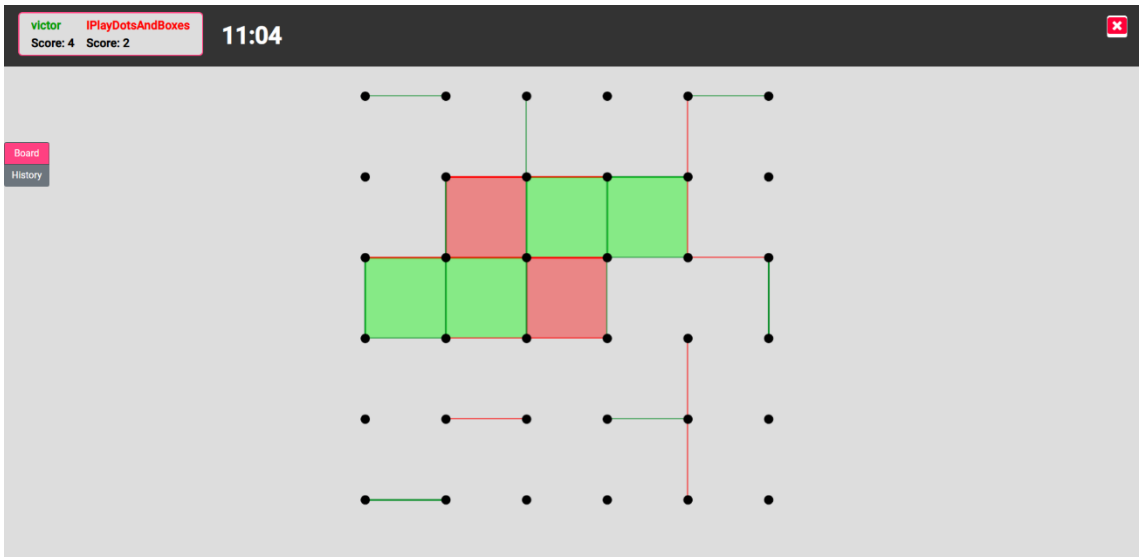


Figura 6.21. Diseño final de la interfaz de juego: tablero

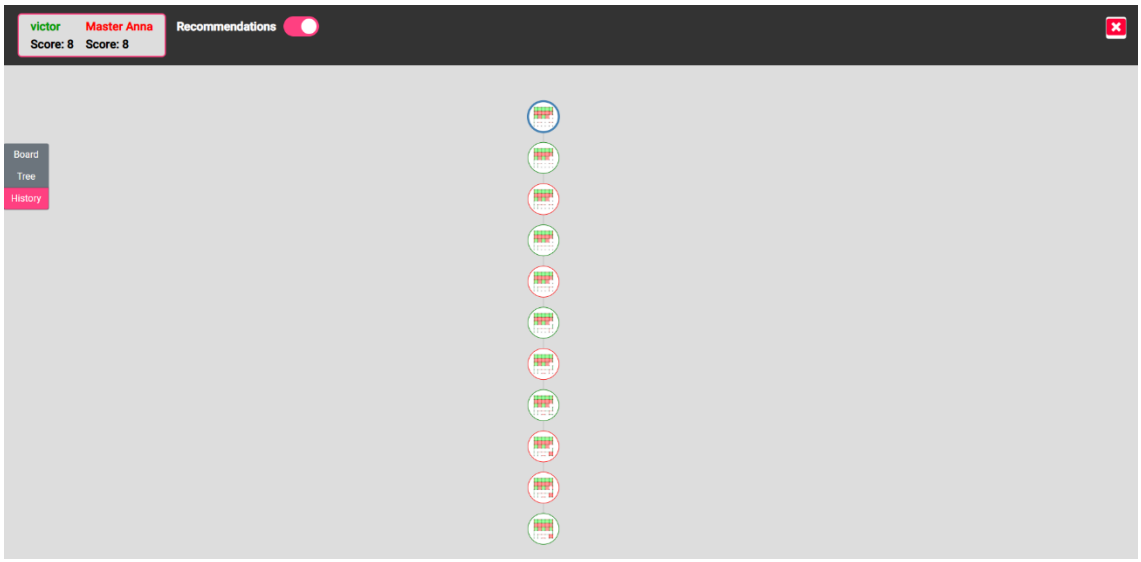


Figura 6.23. Diseño final de la interfaz de juego: historial

Figura 6.22. Diseño final de la interfaz de juego: árbol modo experto

Figura 6.25. Diseño final de la interfaz de juego: popup con información

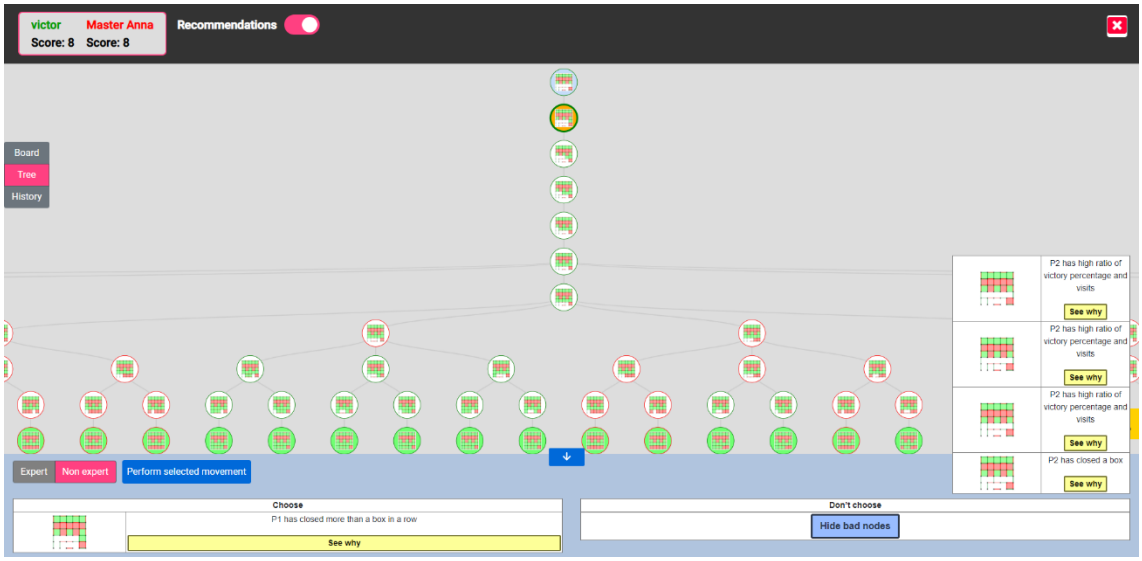


Figura 6.24. Diseño final de la interfaz de juego: árbol modo no experto

6.5.10 Interfaz de Pantalla de Ayuda

Esta pantalla ayuda a comprender toda la información presente en el árbol, así como a utilizar las distintas herramientas que propone. Está dividida en tres apartados principales: navegación por el árbol, modo experto y modo no experto.

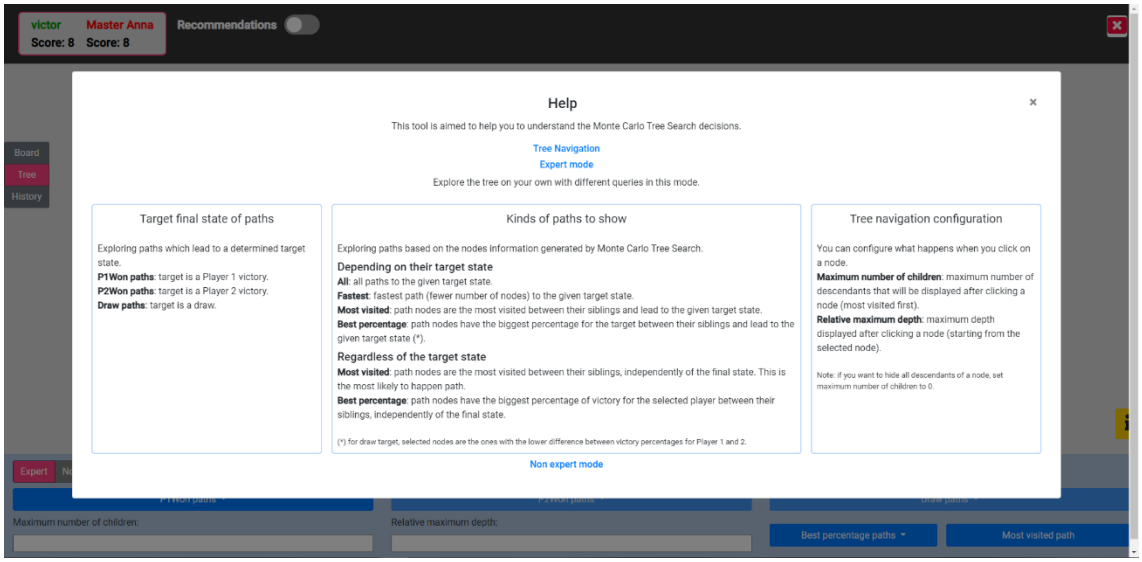


Figura 6.26. Diseño final de la pantalla de ayuda

6.5.11 Interfaz de Partida Finalizada

El contenido de esta pantalla dependerá de si el jugador ha ganado o perdido la partida. Además, la puntuación sumada o restada a su cuenta, solo aparecerá si se trata de una partida clasificatoria.

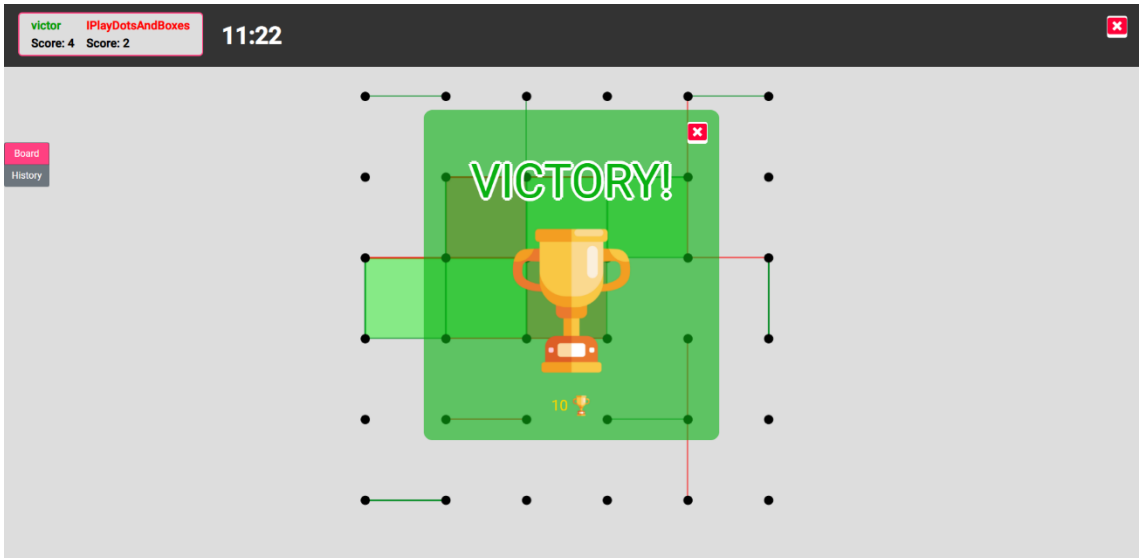


Figura 6.27. Diseño final del cuadro de diálogo de partida finalizada: victoria

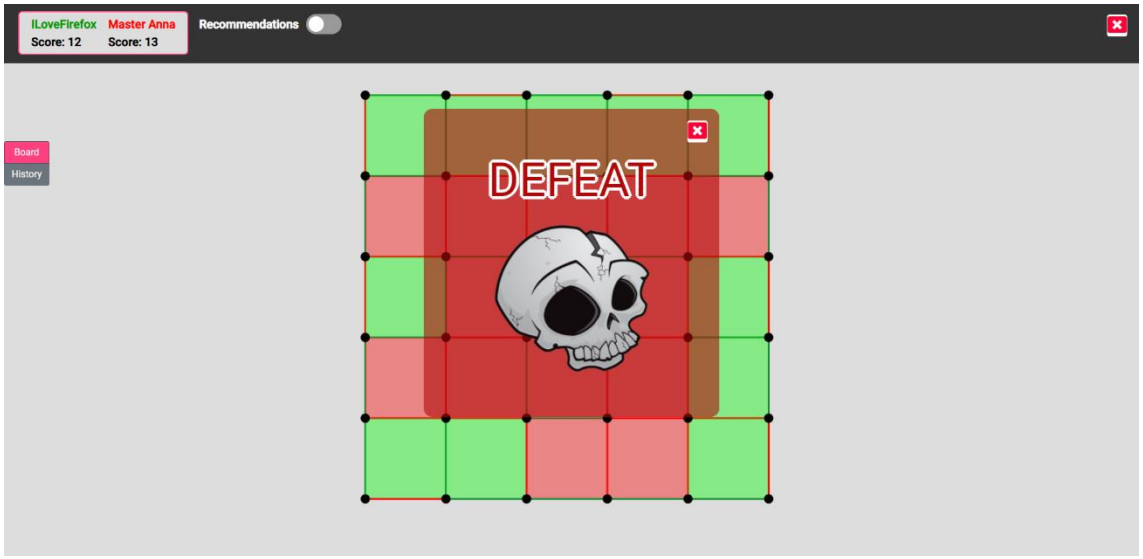


Figura 6.28. Diseño final del cuadro de diálogo de partida finalizada: derrota

6.6 Especificación Técnica del Plan de Pruebas

En esta sección se especificarán las distintas pruebas y metodologías que se emplearán para probar el buen funcionamiento de la aplicación *web* y el sistema, tanto a lo largo del desarrollo como una vez que la implementación esté finalizada.

Todas las pruebas se realizarán empleando como servidor un Lenovo Z70-80, con 16 GB de RAM y un i7-5500U a 2.4 GHz de base. También es importante destacar que el equipo únicamente cuenta con discos duros tradicionales. El sistema operativo empleado es Windows 10, versión 20H2. El código de *backend* se ejecutará desde el IDE de Eclipse, versión 4.19.0, JDK 13.0.2. Por otra parte, la versión de Angular empleada para servir el código a los clientes será la 11.0.3.

Como cliente podrá usarse bien el mismo equipo que sirve la aplicación, o bien el equipo de cada usuario real.

6.6.1 Pruebas Unitarias

Estas pruebas se realizarán de forma independiente para los dos subsistemas principales de la aplicación: la interfaz de usuario para navegador y el proceso de *backend* que se ejecutará en el servidor.

La mayoría de las funcionalidades de la interfaz de usuario será probada interactuando con su implementación en un navegador. Además de esto, deberá probarse el módulo de código que busca distintos tipos de caminos en los árboles generados por el algoritmo MCTS y extrae de ellos las recomendaciones. Una vez implementado este módulo, se probará con, al menos, 10 árboles distintos generados por ese sistema inteligente. Para cada árbol utilizado como entrada, se comprobará que todas las consultas funcionan, mostrando (todos y solo) los caminos correctos, y que las recomendaciones y sus explicaciones son las esperadas.

Para probar el código de *backend* se podrán emplear tanto clases de prueba como software externo para ejecutar pruebas sobre APIs (como puede ser Postman). Se pondrá especial hincapié en la lógica de identificación, registro, *tokens* de sesión y filtro de peticiones no autorizadas, ya que es la parte más crítica para la seguridad del sistema. Se comprobará que todo funcione como se espera: que un usuario con una sesión activa no pueda iniciar otra, que quien tenga una partida multijugador en curso no pueda pedir otra, que los *tokens* se generen solo para credenciales válidas y caduquen si no hay actividad, etc.

Además, será necesario que, tras implementar una funcionalidad compleja se prueba, no solo que la misma no tiene errores, sino que no ha afectado al buen funcionamiento del resto.

En caso de que alguna de las pruebas fallase, se intentará dar prioridad a averiguar y solventar la causa del fallo. Esto es crucial, ya que detectar el origen concreto de los fallos cuando estos se amontonan puede llegar a convertirse en un verdadero problema, afectando negativamente al desarrollo de la aplicación y su entrega.

6.6.2 Pruebas de Integración y del Sistema

Las pruebas de funcionalidad e integración del sistema que se apliquen durante el desarrollo consistirán, en un principio, en comprobar que los comportamientos detallados en las tablas de la Sección 5.7, efectivamente, se dan en el producto final. Aunque estas tablas no deben ser una limitación en caso de que, durante el desarrollo, surgiesen nuevas pruebas que llevar a cabo.

Estas pruebas se realizarán tanto con datos inválidos como con datos válidos. En las situaciones en las que tenga sentido, será especialmente importante probar los valores límite y que el sistema los trata como se espera. También será importante probar el comportamiento del sistema ante conductas inesperadas o "poco lógicas", como podría ser interrumpir la conexión, cerrar el navegador, etc.

Aunque este tipo de pruebas no se pueden llevar a cabo desde el principio, cada una de ellas se intentará ejecutar cuanto antes, en cuanto el estado de la implementación lo permita. También es importante reiterar su ejecución cuando haya habido cambios significativos en el sistema en su conjunto, en partes clave que afecten a la funcionalidad probada, o cuando se considere que se ha avanzado lo suficiente en el desarrollo.

6.6.3 Pruebas de Usabilidad y Accesibilidad

Este tipo de pruebas son especialmente relevantes, ya que intentan medir la satisfacción del cliente final con el producto desarrollado y tratan de evaluar su desempeño en un uso real.

Su objetivo es mejorar la aplicación, especialmente aquellos aspectos relacionados con la interacción de los usuarios con el sistema, y reducir el tiempo de aprendizaje, haciéndola más intuitiva y autoexplicativa.

Para ello, se contará con la colaboración de varios usuarios reales. Al menos habrá uno de cada uno de los siguientes tipos:

- Persona joven, acostumbrada al manejo de ordenadores, y usuaria habitual de juegos online.
- Persona joven, acostumbrada a trabajar con ordenadores, pero jugador online ocasional.
- Persona adulta, jubilada y que nunca usa ordenadores, pero emplea regularmente un *smartphone* en su vida cotidiana.
- Persona adulta, en activo y que usa ordenadores de forma regular en su trabajo.

Todas las pruebas de esta categoría se realizarán en las casas de los usuarios finales. A los usuarios se les pedirá que realicen varias actividades (registrarse, jugar una partida multijugador y otra de entrenamiento…).

Posteriormente, se les pedirá que rellenen un breve cuestionario para que expresen sus opiniones para con la aplicación. El responsable de las pruebas hará lo propio, evaluando desde otra perspectiva el uso de la aplicación por parte de los usuarios.

6.6.3.1 Actividades de las Pruebas de Usabilidad

6.6.3.1.1 Preguntas de carácter general

Antes de las pruebas, los usuarios rellenarán este breve cuestionario para conocer el perfil de cada uno.

¿Usa un ordenador frecuentemente?
1. Todos los días 2. Varias veces a la semana 3. Ocasionalmente 4. Nunca o casi nunca
¿Cómo describiría su experiencia cuando usa un ordenador?
1. Tediosa 2. Indiferente 3. Amena 4. Entretenida
¿Juega a juegos de mesa?
1. Sí, varias veces a la semana 2. Sí, varias veces al mes 3. Con muy poca frecuencia 4. No, nunca
¿Juega juegos online?
1. Sí, varias veces a la semana 2. Sí, varias veces al mes 3. Con muy poca frecuencia 4. No, nunca
¿Qué busca Vd. Principalmente en un programa?
1. Que sea fácil de usar 2. Que sea intuitivo 3. Que sea rápido 4. Que tenga todas las funciones necesarias

6.6.3.1.2 Actividades guiadas

A continuación, se fijarán una lista de las actividades que realizarán los usuarios. Tras ello, expresarán los problemas y dificultades que han sentido.

- Registrarse en la aplicación
- Cerrar sesión
- Volver a iniciar sesión con sus credenciales
- Buscar su perfil para ver cuántos puntos tienen
- Jugar una partida de entrenamiento contra el rival virtual, intentando aprender de las recomendaciones
- Jugar una partida, del tipo que elijan, online
- Acceder al *ranking* de usuarios y localizar cuánta puntuación tienen tras la partida

6.6.3.1.3 Preguntas Cortas sobre la Aplicación y Observaciones

Este será el cuestionario que se presentará a cada usuario al final de su prueba.

Facilidad de Uso	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Sabe dónde está dentro de la aplicación?				
¿Existe ayuda para las funciones en caso de que tenga dudas?				
¿Le ha resultado de utilidad esa ayuda?				
¿Le resulta sencillo el uso de la aplicación?				
¿Ha sabido identificar las distintas opciones disponibles a lo largo de la aplicación?				
¿Le ha resultado fácil, una vez dentro, jugar una partida?				
¿Le ha resultado fácil entender las recomendaciones de la aplicación?				
Funcionalidad	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Funciona cada tarea como Vd. Espera?				
¿El tiempo de respuesta de la aplicación es muy grande?				
¿Ha sentido que la aplicación respondía a sus comandos como debería?				
¿Ha encontrado difícil de batir al rival virtual?				
Calidad del Interfaz				
Aspectos gráficos	**Muy Adecuado**	**Adecuado**	**Poco Adecuado**	**Nada Adecuado**
El tipo y tamaño de letra es				

Los iconos e imágenes usados son				
Los colores empleados son				
El dibujo del tablero es				
La disposición del árbol es				
Diseño de la Interfaz		**Si**	**No**	**A veces**
¿Le resulta fácil de usar?				
¿El diseño de las pantallas es claro y atractivo?				
¿Cree que el programa está bien estructurado?				
¿Cree que los menús están bien diseñados?				
¿Es adecuada la cantidad de opciones en cada menú?				
¿Le ha resultado agradable el diseño de la pantalla de juego?				
Observaciones				
Cualquier comentario del usuario				

6.6.3.1.4 Cuestionario para el Responsable de las Pruebas

Aspecto Observado	**Notas**
El usuario comienza a usar las distintas funcionalidades rápidamente	
Tiempo en realizar cada tarea	
Errores leves cometidos	
Errores graves cometidos	
El usuario comprende las recomendaciones	
El usuario se atreve a explorar el árbol	
El usuario ha disfrutado la experiencia de juego	

Tras esto, se analizarán los resultados de las pruebas con usuarios para extraer conclusiones sobre los siguientes aspectos:

1. **Facilidad de aprendizaje**: Capacidad para aprender la funcionalidad de la aplicación y desarrollar las tareas de manera adecuada, midiendo cuanto se tarda en hacer las distintas tareas.
2. **Eficiencia**: Cuánto mejora la labor de los usuarios por usar la aplicación respecto a lo que se hacía anteriormente.
3. **Errores**: Cuántos errores cometen los usuarios en las distintas tareas. Eso podría significar que la aplicación es poco intuitiva o usable.
4. **Satisfacción del usuario**: Impresión general de los usuarios al usar la aplicación.

6.6.3.2 Pruebas de Accesibilidad

Se intentará desarrollar la aplicación siguiendo las pautas que establece WCAG (*Web Content Accessibility Guidelines*), y se analizará con herramientas externas cuya finalidad es evaluar la accesibilidad, como podría ser *Lighthouse* en Chrome. Este software de código abierto sirve par auditar aplicaciones y páginas *web* en lo relativo a diversos aspectos como accesibilidad, calidad y rendimiento.

6.6.4 Pruebas de Rendimiento

El rendimiento es algo a tener en cuenta en aplicaciones con las que el usuario interactúe ya que, si estas no responden rápido, es común que el usuario las abandone. Además, si esta aplicación está, en cierta parte, orientada a que los usuarios puedan usarla para aprender, estos problemas podrían causar una total pérdida del interés.

Por ello, se realizarán pruebas para medir los tiempos de respuesta del sistema ante las acciones más comunes, especialmente cuando el número de usuarios es relativamente alto. Se vigilará con especial cuidado el tiempo que tardan en generarse los movimientos del rival virtual y las recomendaciones.

Capítulo 7. Implementación del Sistema

Este capítulo trata todo lo relativo al desarrollo del código de nuestro sistema: estándares y normas seguidas; herramientas y lenguajes de programación utilizados, y algunos aspectos relevantes sucedidos a lo largo de la implementación.

7.1 Estándares y Normas Seguidos

El principal estándar seguido es HTML5 a la hora de construir la estructura de las distintas pantallas y cuadros de diálogo de la aplicación *web*. En realidad, durante el desarrollo, no se ha empleado HTML5 puro, sino que se han aprovechado las posibilidades que ofrece la sintaxis basada en plantillas de Angular. Este HTML5 "aumentado" o "con anabolizantes" permite, por ejemplo, incrustar un código HTML dentro de diversos archivos, reutilizando el código; emplear operaciones lógicas; establecer datos de entrada o eventos, y emplear distintos tipos de variables y objetos para construir estructuras HTML mutables o sensibles al contexto. De todas formas, una vez compilado, este código se traduce a HTML5 estándar y JavaScript.

Además, se han intentado seguir las convenciones de nomenclatura y código para Java, TypeScript y las estructuras de la base de datos Mongo. Esto permite que el código desarrollado sea más legible y fácil de comprender sin un análisis en profundidad.

7.2 Lenguajes de Programación

Aquí se enumerarán los distintos lenguajes de programación empleados para el desarrollo de la aplicación y se ofrecerá una breve descripción de los mismos.

7.2.1 Java

Java es un lenguaje de programación orientado a objetos y fuertemente tipado. Fue publicado en 1996 por Sun Microsystems, que pertenece, desde 2010, al gigante tecnológico Oracle.

Este lenguaje es, aún a día de hoy, uno de los lenguajes más populares para el desarrollo de *software* para servidores en aplicaciones de tipo cliente-servidor, como es la nuestra.

Principalmente se ha escogido este lenguaje por ser seguro, contar con una sintaxis sencilla y ser, al menos en teoría, multiplataforma. En concreto la versión utilizada ha sido Java EE 13 (JDK 13.0.2). Como complemento se ha empleado la versión 3.6.3 de Maven, una herramienta para creación y gestión de proyectos publicada por Apache.

7.2.2 TypeScript

TypeScript es un superconjunto de JavaScript que apareció en 2012 de la mano de Microsoft. Las principales ventajas que agrega con respecto a JavaScript es que es orientado a objetos y fuertemente tipado, lo cual permite detectar la mayoría de los fallos durante la implementación, evitando así gran parte de los errores en tiempo de ejecución. Además, si fuese necesario, TypeScript soporta todo el código JavaScript puro (al ser un superconjunto del propio JavaScript) a pesar de ser, en principio, un lenguaje orientado a objetos. La versión utilizada en este proyecto es la 4.0.5.

Figura 7.1. Logo de TypeScript

7.3 Herramientas y Programas Usados para el Desarrollo

Aquí se describirán las herramientas y *software* que se han empleado principalmente para el desarrollo de este sistema.

7.3.1 Visual Studio Code

Visual Studio Code (o también VSCode) es un editor de código fuente, construido sobre el *framework* Electron, lanzado en 2015 por Microsoft. VSCode, además, es gratuito y de código abierto. Está desarrollado en TypeScript y ofrece versiones para Windows, Linux, macOS e incluye versión *web* para navegador.

VSCode se ha elegido como IDE para desarrollar el cliente *web* en este proyecto debido a las comodidades que ofrece: es compatible con numerosos lenguajes de programación y marcado, es altamente personalizable (tema del editor, preferencias, atajos de teclado, etc.),

Figura 7.2. Logo de Visual Studio Code

implementa numerosas utilidades (formateo y refactorización de código, integración con Git, soporte para la depuración, etc.) y cuenta con infinidad de extensiones desarrolladas por la comunidad. La versión utilizada en este caso ha sido la versión 1.63.2.

7.3.2 Eclipse IDE

El IDE de Eclipse es un entorno de desarrollo integrado de código abierto, ideado principalmente para el desarrollo de software empleando el lenguaje de programación Java. Está también programado en Java, y fue lanzado en 2001 por la Eclipse Foundation.

Figura 7.3. Logo del IDE de Eclipse

Gracias a que Eclipse consigue proporcionar sus distintas funcionalidades utilizando una arquitectura de módulos (*plug-ins*, en inglés), se pueden aumentar sus características y sus posibilidades instalando más módulos (para Git, Maven, C++, Python, LaTeX, etc.). Además, Eclipse cuenta con un compilador de Java interno, lo que permite ofrecer a sus usuarios técnicas más avanzadas de refactorización, y análisis y generación de código. Por todas estas bondades, se ha decidido emplear este IDE para escribir el código de *backend* de nuestro proyecto, y para compilarlo y ejecutarlo durante las pruebas. La versión concreta utilizada es la de marzo del 2021, 4.19.0.

7.3.3 Angular

Como ya se ha explicado, Angular es un *framework* de código abierto especialmente enfocado al desarrollo de aplicaciones de navegador, aunque también ofrece la posibilidad de ser usado para crear aplicaciones para dispositivos móviles. Esta herramienta permite organizar el código en distintos componentes, facilitando enormemente su reutilización. Además, permite el uso de una gran cantidad de componentes de código abierto ya desarrollados por la comunidad.

Se ha empleado, junto con NPM (Node Package Manager), para el desarrollo de la aplicación de navegador y la interfaz *web* que se comunicarán con el proceso del servidor en este sistema. NPM, por su parte, es un gestor de paquetes que permite descargar *software* y componente de código abierto, y controlar de forma centralizada sus versiones.

Figura 7.4. Logo de NPM

Las versiones de Angular y NPM empleadas son, respectivamente, la 4.0.5 y la 12.16.1.

7.3.4 Git

Git es un conocidísimo y ampliamente usado sistema distribuido de control de versiones. Es gratuito y de código abierto, y permite mantener el control de versiones de proyectos que cuentan con un gran número de archivos de código fuente de forma fácil, eficiente y fiable. Este software fue diseñado por Linus Torvalds y se lanzó en 2007.

En este proyecto, se ha empleado la versión 2.23.0. Además de para el control de versiones, se utilizará para mantener el código actualizado en dos repositorios remotos de GitHub (uno para el código de *frontend* y otro para el de *backend*).

Figura 7.5. Logo de Git

7.3.5 Postman

Este es un software específicamente diseñado para probar y ayudar al desarrollo de distintos tipos de APIs *web*. Permite generar peticiones, controlando todos los parámetros de la consulta, cabecera y cuerpo; enviarlas a un servidor, y estudiar la respuesta de este. Además, las versiones más recientes, implementan la posibilidad de comunicarse con APIs que implementen el protocolo WebSockets, por lo que se ajusta perfectamente a lo que se necesita en este proyecto.

Figura 7.6. Logo de Postman

En nuestro caso, se ha empleado este programa para probar las distintas formas que tendrán el servidor y los clientes de intercambiar información: tanto para las comunicaciones mediante protocolo HTTP para la identificación, gestión de *tokens* de sesión y transferencia de información de usuarios, como para las que sucedan empleando el protocolo WebSockets, para transmitir información sobre el desarrollo de las partidas. La versión utilizada durante todo el desarrollo ha sido la 9.8.2.

7.3.6 JSON

La Notación de Objetos de JavaScript (JSON, acrónimo del inglés *JavaScript Object Notation*) es un formato de texto sencillo ideado, principalmente, para el intercambio de datos. La especificación del formato JSON se publicó oficialmente en 2006, ideada por Douglas Crockford. A pesar de que es un subconjunto del estándar del lenguaje de programación JavaScript, se considera que es un formato independiente del lenguaje, debido a su gran aceptación como alternativa a XML. Uno de los factores clave para esta aceptación es que es muy fácil de leer y escribir por humanos y, además, también resulta fácil implementar un analizador sintáctico para él.

```json
{
  "nombre": "María",
  "apellidos": "Rodríguez Rodríguez",
  "nacimiento": {
    "año": 1983,
    "mes": "mayo",
    "dia": 7
  },
  "dni": [0, 1, 2, 3, 4, 5, 6, "Z"],
  "mascotas": null,
  "hasCovid": false
}
```

Figura 7.7. Ejemplo de un objeto JSON

El formato está constituido por dos estructuras distintas: una colección de pares nombre-valor (similar a un registro o un diccionario) y una lista ordenada de valores (que se puede entender como un vector). Debido a que estas estructuras son básicas y están presentes, con un nombre u otro, en todos los lenguajes de programación, JSON es un buen candidato para un intercambio de datos independiente del lenguaje.

En este proyecto el software de cliente y servidor está implementado en lenguajes distintos; el cliente está desarrollado utilizando TypeScript y para la lógica del servidor se emplea Java. Por ello, JSON aparece como un formato muy conveniente para el intercambio de datos, tanto entre cliente y servidor (ya sea para procesos de autenticación, o para los datos de las partidas), como entre el servidor y la base de datos.

7.3.7 Navegadores web

En este proyecto, se han empleado varios navegadores distintos durante el desarrollo de la aplicación, así como para probar que las funcionalidades implementadas funcionan como se espera y que la interfaz de usuario se muestra de la forma adecuada. Además, este tipo de aplicaciones serán las que, una vez finalizado el desarrollo, permitirán a los usuarios acceder a la aplicación *web* y utilizarla.

En concreto, la aplicación ha sido probada con Google Chrome, Microsoft Edge, Mozilla Firefox y Opera, versiones 97.0.4692.71, 97.0.1072.69, 95.0.2 y 82.0.4227.32, respectivamente.

7.4 Creación del Sistema

En este apartado se describirán los aspectos relativos a la implementación del código.

7.4.1 Problemas Encontrados

A continuación, se expondrán aquellos problemas encontrados durante el desarrollo del sistema que puedan resultar más interesantes por su naturaleza, complejidad o manera de abordarlos. También se expondrá la solución encontrada para cada uno de ellos.

7.4.1.1 Comunicación de Datos de Partida

Para que un usuario pueda jugar una partida online, tanto contra otro jugador humano, como contra la inteligencia artificial, es necesario que la máquina cliente y el servidor se intercambien información relativa a esta partida: enviar los movimientos realizados por el usuario; recibir aquellos que haga el rival; recibir los datos y el árbol para generar las recomendaciones; etc. Para ello, en un principio, se habían empleado comunicaciones a través de HTTP, ya que, al ser un juego de mesa basado en turnos, el cliente enviaba sus movimientos y recibía el estado del tablero una vez terminado el turno de su rival. Pese a ser una solución funcional, esta presentaba el problema de que, en el caso (común) de que el rival pueda realizar varios movimientos a lo largo de su turno, la información de la que el jugador disponía solo se actualizaba tras el último de ellos (mostrando todos los movimientos al final de golpe, y no uno a uno mientras suceden). El efecto que esto tenía en el jugador era totalmente negativo, generando la sensación de estar esperando durante mucho tiempo y de confusión, si el tablero cambiaba su configuración de forma muy drástica.

Para solucionar esto se decidió emplear el protocolo WebSockets que, como se explica en el Capítulo 3 de este documento, permite establecer una conexión bidireccional y duradera en el tiempo entre cliente y servidor. Mientras esta conexión permanezca activa, ambas partes podrán enviar datos cuando deseen, debiendo implementar la parte contraria la lógica para procesar esa información de la manera adecuada. Así, mediante el uso de esta tecnología, logramos que el servidor pueda enviar las actualizaciones del tablero al jugador en tiempo real, en el momento en que suceden, y una a una.

7.4.1.2 Animaciones Congeladas

Cuando un usuario está jugando una partida contra la IA y con las recomendaciones activadas, dispone de un árbol interactivo desde el que visionar sus movimientos posibles, los datos generados sobre ellos y los posibles caminos que tomará la partida a partir de ellos. En esa vista de árbol, además de otras herramientas, se ofrecen una serie de consultas que permiten buscar distintos tipos de caminos (posibles formas en las que la partida irá evolucionando) o nodos (estados del tablero en un punto concreto de la partida) en el árbol.

Por ello, con el objetivo de que el jugador percibiera que el sistema estaba procesando su petición (sobre todo con aquellas que puedan tardar más segundos), se implementó una sencilla animación de carga. El problema se presentó cuando la ejecución de esas consultas causaba que la animación dejase de actualizarse, ya que el navegador no ejecutaba el código JavaScript y CSS a la vez. Para solucionar esto, se decidió recurrir a los *web workers*, un concepto, similar al de hilo, que implementan la mayoría de navegadores modernos. Estos "trabajadores" permiten que el navegador ejecute código JavaScript en segundo plano. Así, se modificó la aplicación en lo necesario para que la ejecución de las distintas consultas se llevase a cabo en hilos secundarios, permitiendo que el hilo de ejecución principal siguiese animando la interfaz de usuario.

7.4.1.3 Movimientos Rivales no Percibidos por el Jugador

En mitad del desarrollo de la aplicación, cuando únicamente estaban implementadas las partidas contra la IA, se hicieron unas pruebas con usuarios reales para evaluar su aprendizaje. Además, también se pidió que ofreciesen sus opiniones para con la aplicación y posibles puntos de mejora. Varios usuarios se quejaron de que a veces el rival "no movía", esto era, no porque la IA no hiciese movimientos en su turno, sino porque el jugador perdía la atención, el tablero se actualizaba y este no se daba cuenta. Para solucionar esto, se implementó una pantalla de espera durante el turno rival más clara, que deja mejor ver cuándo es el turno del usuario y cuándo no. Además, se decidió que se implementaría un historial de movimientos disponible en cualquier punto de la partida.

7.4.1.4 Error CORS

CORS (*Cross-Origin Resource Sharing* o, en castellano, intercambio de recursos de origen cruzado) es, de forma sencilla, un mecanismo que implementan la mayoría de navegadores *web* para bloquear los accesos a recursos alojados en un servidor desde un origen (dominio) distinto al que pertenece ese servidor, a no ser que el servidor exprese un permiso explícito para ese origen.

Por la estructura del sistema, donde hay una interfaz *web* y un proceso de *backend* diferenciados, esto causaba que las peticiones desde el cliente al servidor fuesen bloqueadas por el navegador. Esto se debe a que, aunque tanto la interfaz como las conexiones con el *backend* se sirven desde la misma máquina, al acceder a cada uno a través de un puerto distinto, el navegador los toma como dos orígenes diferenciados.

Para solucionar esto, se implementó en el código de *backend* un filtro que captura todas las peticiones y añade a la respuesta correspondiente una cabecera para permitir accesos desde orígenes distintos. Durante la fase de desarrollo y pruebas, esa cabecera permite cualquier origen, pero, una vez la aplicación esté desplegada, esto deberá sustituirse por el dominio concreto a través del cual se accede a la interfaz de usuario, para evitar posibles problemas de seguridad.

7.4.1.5 El Servidor es Incapaz de Emparejar a dos Jugadores

En una primera fase del desarrollo de las partidas multijugador, se implementó la lógica que permitía emparejar a distintos jugadores en una misma partida, siempre que buscasen un tipo de partida coincidente. Por una razón ajena, aparentemente, a esta lógica, el servidor no era capaz de unir a dos jugadores en la misma partida. Tras pensarlo detenidamente, me di cuenta de que el problema era que, para cada conexión WebSockets, se creaba una instancia nueva de la clase AlphastarWebSocket, que era donde se almacenaban las partidas disponibles a las que se podían unir los jugadores. Para solucionarlo, simplemente se implementó una clase cuyo fin único era servir como colección de partidas de la cual solo hubiese una instancia en todo el ciclo de ejecución del programa (una clase *singleton*), GamesPool.

7.4.2 Patrones de Diseño Empleados

Aquí se relatarán los patrones de diseño que han surgido a lo largo del diseño o implementación del sistema. Se realizará una breve descripción de cada uno y se explicará en qué punto (o puntos) de la aplicación se ha empleado y con qué objetivos.

7.4.2.1 Singleton

Este patrón sirve para crear una clase de la cual solo existirá una única instancia y que proporciona un punto de acceso a esa instancia.

Como se ha explicado anteriormente, cada vez que un cliente solicita una nueva conexión WebSockets con el servidor, se crea una nueva instancia de la clase encargada de procesar esas comunicaciones (AlphastarWebSocket). Para conseguir que la colección donde se guardan las partidas disponibles sea la misma para todas esas instancias, se ha creado una clase siguiendo el patrón *singleton* que envuelve esa colección de partidas e implementa los métodos necesarios para acceder a ella: GamesPool.

De forma similar, cuando el servidor recibe una petición HTTP, crea una nueva instancia de la clase encargada de procesarla (RegistrationServlet, AuthenticationServlet, LogoutServlet, etc.). Gran parte de esas clases necesitan tener acceso a la colección de *tokens* activos, que debe ser la misma para todo el programa. Como esa colección se guarda en memoria, ya que nunca va a ocupar mucho espacio al ir borrando los *token*s después de 15 minutos de inactividad y será accedida en muchas ocasiones, la clase que la contiene debe ser necesariamente un *singleton*. Esa clase es, en este caso, Authenticator.

Para tener acceso a la base de datos MongoDB a través de su API para Java, se necesita crear una instancia de la clase MongoClient. Esta instancia se ha envuelto dentro de una clase, *DBClient*, que implementa el patrón *singleton*. Esto se ha decidido porque, en la documentación de la propia API, se recomienda que, si es un único proceso el que va a acceder a la base de datos (como en nuestro caso, el proceso de *backend*), se mantenga una única instancia de la clase MongoClient y se realicen a través de ella todos los accesos a la base de

datos. Esto es, según esa documentación, para que la API pueda controlar cómo y cuándo se producen esos accesos de forma efectiva.

7.4.2.2 Strategy

Por su parte, este patrón permite definir una familia de funcionalidades, encapsular cada una de ellas (en una clase distinta) y hacerlas intercambiables. En este proyecto se utilizará principalmente para evitar una explosión de subclases que implementen IServerGame (la interfaz de la que parten todas las clases partida). Así, estas clases se mantendrán en dos en todo el proyecto (PVPGame, para las partidas multijugador, y TrainingGame, para las partidas de entrenamiento), que se convertirían en 14 subclases distintas solo con los tipos de partida que existen actualmente. Además, si en un futuro se decidiesen implementar más tipos de partida, como podrían ser partidas de entrenamiento con tiempo limitado o que afectasen a la puntuación, no sería necesario crear ninguna subclase más gracias al uso del patrón *strategy* para encapsular estas funcionalidades.

De esta forma, se ha empleado este patrón para encapsular la forma en la que se guardan las partidas (clases que implementa IGameSaver), para cómo se calculan las puntuaciones al final de una (IScoreCalculator) y para cómo se deciden los límites de tiempo para realizar movimientos en una partida (IGameTimer).

7.4.2.3 Template

Este patrón sirve para definir en una clase padre el esqueleto de una operación (su comportamiento general y el orden en el que se ejecutan los pasos), dejando libertad a las clases que heredan de ella para reimplementar algunos de esos pasos. Permite extraer la lógica común de una función a la clase padre, mientras las clases hijas se ocupan de sus comportamientos particulares.

Se ha empleado en varias ocasiones a lo largo de este proyecto. Un ejemplo de ello es el método *init()* de la clase AbstractServerGame, que implementa la lógica común a todas las partidas, mientras establece algunos pasos dentro de ese método que deberán ser implementados por las clases hijas, como son *configureExternalParameters()*, *createClientSetUpResponse()*, o *generatePossibleMovements()*.

7.4.2.4 Observer

El patrón *observer* permite definir una dependencia entre el "observado" (*observable* o *publisher*) y el "que observa" (*observer o subscriber*), de forma que cuando el primero sufre un cambio de estado, se notifica al segundo para que ejecute la lógica necesaria. Este patrón se ha empleado en dos partes de este proyecto. Por un lado, los componentes Tool (el árbol interactivo) y BoardDrawPlay (el tablero interactivo) son capaces de notificar al componente Play cuando se realiza un movimiento. De forma similar, las clases que implementan IGameTimer notifican a la partida a la que pertenecen cuando el tiempo de un jugador se ha agotado.

7.4.2.5 Factory Method

Este patrón se empleará, en nuestro caso, para extraer la lógica de creación de partidas de distinto tipo a la clase ServerGameFactory. Las instancias de la clase AlphastarWebSocket recurrirán a esta clase para generar el tipo exactos de partida que el cliente solicita. Para ello, invocarán el método *createServerGame()*, que devuelve una instancia de una clase que implemente IServerGame y recibe como parámetro el tipo de partida que ha llegado desde el cliente a través de una comunicación WebSockets. Ya en ese método, se decide la clase de partida a instanciar y las estrategias concretas que se aplicarán a esa partida.

7.4.3 Descripción Detallada de las Clases

Este es un proyecto que nace mientras yo estaba becado por la FUO (Fundación Universidad de Oviedo) como, en cierta forma, continuación de un proyecto anterior desarrollado por Alba Cotarelo Tuñón. En su trabajo **[1]**, desarrolló el sistema inteligente que se utiliza en esta aplicación para generar los movimientos del rival artificial y los datos que, posteriormente, sirven para definir las recomendaciones que reciben los jugadores. Además, Alba también participó inicialmente en el desarrollo de una parte de este sistema. Por este motivo, dentro de los proyectos de código que se entregarán anexos a este documento, hay partes desarrolladas tanto por ella como por mí. Para clarificar, a continuación, se presentará dos tablas (la primera para el proyecto de Java y la segunda para el de Angular) en las que aparecerán todos los archivos de código (o paquetes) junto con su autor (en caso de que los dos hayamos participado en el desarrollo de algún archivo, aparecerá primero el nombre de aquel cuya contribución a ese archivo concreto sea mayor). En las tablas no se incluirán archivos de configuración ni aquellos archivos autogenerados que no hayan sido modificados por ninguno de los dos.

Paquete	**Autor/autora**
alphastar.core	Alba
alphastar.game	Alba
alphastar.mcts	Alba
alphastar.minimax	Alba
alphastar.neuralnet	Alba
alphastar.play	Alba
*src/test/**	Alba
alphastar.server	Víctor
alphastar.util	Víctor

Archivo de código	**Autor/autora**
src.styles.css	Víctor, Alba
src.app.module.ts	Víctor, Alba
src.app.model.board.ts	Alba, Víctor
src.app.model.iBoard.ts	Alba
src.app.model.iMove.ts	Alba
src.app.model.move.ts	Alba

src.app.components.board.component.ts	Alba
src.app.components.board.component.html	Alba
src.app.components.board.component.css	Alba
src.app.components.help.component.ts	Alba, Víctor
src.app.components.help.component.html	Víctor, Alba
src.app.components.help.component.css	Alba
src.app.components.main-menu.component.ts	Víctor
src.app.components.main-menu.component.html	Víctor
src.app.components.main-menu.component.css	Víctor
src.app.components.main-menu.login-register.component.ts	Víctor
src.app.components.main-menu.login-register.component.html	Víctor
src.app.components.main-menu.login-register.component.css	Víctor
src.app.components.main-menu.profile.component.ts	Víctor
src.app.components.main-menu.profile.component.html	Víctor
src.app.components.main-menu.profile.component.css	Víctor
src.app.components.main-menu.rank-list.component.ts	Víctor
src.app.components.main-menu.rank-list.component.html	Víctor
src.app.components.main-menu.rank-list.component.css	Víctor
src.app.components.moves-history.component.ts	Víctor
src.app.components.moves-history.component.html	Víctor
src.app.components.moves-history.component.css	Víctor
src.app.components.play.component.ts	Víctor
src.app.components.play.component.html	Víctor
src.app.components.play.component.css	Víctor
src.app.components.play.iupdateable-play-mode.ts	Víctor
src.app.components.play.abstract-tree-play-mode.ts	Víctor
src.app.components.play.game-ended.component.ts	Víctor
src.app.components.play.game-ended.component.html	Víctor
src.app.components.play.game-ended.component.css	Víctor
src.app.components.tool.d3Tree.ts	Víctor, Alba
src.app.components.tool.component.ts	Víctor
src.app.components.tool.component.html	Víctor, Alba
src.app.components.tool.component.css	Víctor, Alba
src.app.components.tool.tree.webworker.worker.ts	Víctor
src.app.components.tool.tool-table-expert.component.ts	Víctor
src.app.components.tool.tool-table-expert.component.html	Víctor, Alba
src.app.components.tool.tool-table-expert.component.css	Víctor, Alba
src.app.components.tool.tool-table-not-expert.component.ts	Alba, Víctor
src.app.components.tool.tool-table-not-expert.component.html	Alba, Víctor
src.app.components.tool.tool-table-not-expert.component.css	Alba, Víctor
src.app.components.util.boardDraw.component.ts	Alba, Víctor
src.app.components.util.board-draw-play.component.ts	Alba, Víctor
src.app.components.util.board-draw-play.component.html	Víctor
src.app.components.util.board-draw-play.component.css	Víctor
src.app.components.util.enums.ts	Víctor
src.app.components.util.iBoardDraw.ts	Alba, Víctor

Las tablas siguientes servirán para documentar las clases Java que pertenecen a este proyecto (las que se encuentran en los paquetes *alphastar.server* y *alphastar.util* y sus subpaquetes). No aparecen documentadas las clases que desarrolló Alba en el proyecto anterior.

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
AlphastarWebSocketServer	public	Clase que se encarga de desplegar el servidor de *backend* de la aplicación.	
Responsabilidades			
Número	**Descripción**		
1	Desplegar el servidor en la IP y puerto de acceso definidos.		
2	Establecer las configuraciones y las distintas rutas de acceso al servidor.		
3	Detener el servidor si sucede alguna excepción que no haya podido ser capturada.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public static	void	main	args: String[]
public static	void	startServer	port: int
public static	void	stopServer	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private	static final	int	DEFAULT_PORT
private	static final	Logger	LOGGER
private	static	Undertow	server
private	static	DeploymentManager	deploymentManager
Observaciones			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
AlphastarWebSocket	public	Recibe y procesa las conexiones a través del protocolo WS para jugar una partida.	
Responsabilidades			
Número	**Descripción**		
1	Recibir y ejecutar la lógica correspondiente cuando se reciba una comunicación WebSockets a través de la ruta establecida.		
2	Ocuparse de unir a jugadores a partidas del tipo que soliciten (a una que esté disponible o creando una nueva, si no hay).		
3	Servir de intermediario para la comunicación entre el cliente y la partida en ambas direcciones.		
4	Detectar si una comunicación ha sido cancelada.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	void	open	gameKind: String

			session: Session
public	void	handleMessage	message: String session: Session
public	boolean	sendResponseToClient	session: Session response: JsonObject
public	void	close	session: Session
public	void	onError	e: Throwable
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private	static final	String	PARAM_ACCESS_TOKEN_NAME
private		Authenticator	authenticator
private		GamesPool	games
Observaciones			
La clase está decorada con la anotación @javax.websocket.server.ServerEndpoint que establece que será desplegada y será accesible en un servidor como *websocket*. Algunos de sus métodos también están decorados con sendas anotaciones para que sean invocados e n los momentos correctos de la comunicación con un cliente.			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
CORSFilter	public	Un filtro que añade a las respuestas de todas las peticiones que llegan al servidor el tratamiento CORS adecuado.	Implementa javax.servlet.Filter
Responsabilidades			
Número	**Descripción**		
1	Capturar todas las peticiones HTTP que llegan al servidor.		
2	Añadir a las respuestas a esas peticiones las cabeceras correspondientes para establecer los dominios de origen, métodos y cabeceras permitidas.		
3	Volver a soltar esas peticiones, con las cabeceras añadidas, para que sean procesadas de forma normal.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	void	init	filterConfig: FilterConfig
public	void	doFilter	servletRequest: ServletRequest servletResponse: ServletResponse filterChain: FilterChain
public	void	destroy	
Observaciones			
La clase está decorada con la anotación @javax.servlet.annotation.WebFilter que declara que es un filtro para los distintos *servlets* y permite que el servidor lo aplique durante el despliegue.			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
WebSocketServerConstants	public interface	Guarda constantes que deben de	

		estar disponibles para todo el programa.	
Responsabilidades			
Número	**Descripción**		
1	Guardar la ruta al archivo de configuración.		
2	Guardar la ruta al archivo del certificado SSL.		
3	Guardar los nombres que tienen distintas propiedades relativas al certificado en el archivo de configuración.		
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
public	static final	String	PID
public	static final	String	PROPERTY_SSL_KEYSTORE
public	static final	String	PROPERTY_SSL_KEYSTORE_TYPE
public	static final	String	PROPERTY_SSL_PASSWORD
public	static final	String	CONFIG_FILE_PATH
public	static final	String	CERT_FILE_PATH
Observaciones			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
LeaveGameServlet	public	Recibe y procesa las peticiones HTTP de los clientes que quieren abandonar una partida.	Hereda de javax.servlet.http.HttpServlet
Responsabilidades			
Número	**Descripción**		
1	Recibe las peticiones HTTP realizadas a través de la ruta establecida y las procesa.		
2	Averigua si el usuario está en una partida y, en caso afirmativo, advierte a la partida de que el usuario se ha salido y corta la conexión entre ambos.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
protected	void	doPost	req: HttpServletRequest resp: HttpServletResponse
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Authenticator	authenticator
private		GamesPool	games
Observaciones			

La clase está decorada con la anotación @javax.servlet.annotation.WebServlet que declara que es un *servlet* para que esté disponible en el servidor.

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
PagedUsersServlet	public	Recibe las peticiones HTTP para obtener el *ranking* de usuarios paginado y las procesa.	Hereda de javax.servlet.http.HttpServlet
Responsabilidades			
Número	**Descripción**		
1	Recibe las peticiones HTTP realizadas a través de la ruta establecida y las procesa.		
2	Pide el *ranking* de jugadores (una página concreta y de un tamaño máximo especificado, en realidad) a la clase encargada de las comunicaciones con la BD relativas a usuarios.		
3	Construye la respuesta con esta información y en el formato adecuado.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
protected	void	doGet	req: HttpServletRequest resp: HttpServletResponse
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private	static final	int	DEFAULT_PAGE_NUMBER
private	static final	int	DEFAULT_PAGE_SIZE
private		UsersCRUD	usersCRUD
Observaciones			
La clase está decorada con la anotación @javax.servlet.annotation.WebServlet que declara que es un *servlet* para que esté disponible en el servidor.			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
ServerPlayer	public	Guarda toda la información sobre un jugador que necesita una partida.	
Responsabilidades			
Número	**Descripción**		
1	Guarda toda la información de un usuario de cara al servidor y ofrece métodos para acceder a ella.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	ServerPlayer	ServerPlayer	session: Session user: UserDTO
public	Session	getSession	

public	EPlayer	getPlayingAs	
public	void	setPlayingAs	playingAs: EPlayer
public	UserDTO	getUser	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Session	session
private		EPlayer	playingAs
private		UserDTO	userDTO
Observaciones			
EPlayer es una enumeración que simplemente representa si el usuario está jugando como el jugador uno (EPlayer.P1) o como el juagdor2 (EPlayer.P2). No aparece aquí documentada ya que es una de las clases que pertenecen al proyecto anterior.			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
AccessTokenFilter	public	Un filtro que bloquea las peticiones que no contengan un *token* de sesión válido.	Implementa javax.servlet.Filter
Responsabilidades			
Número	**Descripción**		
1	Capturar todas las peticiones HTTP que llegan al servidor a través de la URL que permite solicitar partidas.		
2	Comprueba si estas peticiones contienen un *token* de sesión, que este esté activo y que no haya sido empleado para solicitar una partida en curso.		
3	Si se cumplen los requisitos anteriores, libera la petición para que sea procesada. De lo contrario, lo bloquea y envía al cliente un mensaje de error.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	void	init	filterConfig: FilterConfig
public	void	doFilter	servletRequest: ServletRequest servletResponse: ServletResponse filterChain: FilterChain
public	void	destroy	
private	void	returnForbiddenError	response: HttpServletResponse message: String
Observaciones			
La clase está decorada con la anotación @javax.servlet.annotation.WebFilter que declara que es un filtro para los distintos *servlets* y permite que el servidor lo aplique durante el despliegue.			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
AuthenticationServlet	public	Recibe las peticiones HTTP para identificarse mediante credenciales y las procesa.	Hereda de javax.servlet.http.HttpServlet

Responsabilidades			
Número	**Descripción**		
1	Recibe las peticiones HTTP realizadas a través de la ruta establecida y las procesa.		
2	Comprueba que las credenciales existan en la base de datos y que no haya una sesión iniciada para esas credenciales a través de la clase Authenticator.		
3	Si se cumplen los requisitos, genera un nuevo *token* para la cuenta identificada. Además, construye una respuesta con el *token* de sesión y los datos del usuario autenticado en el formato correcto.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
protected	void	doPost	req: HttpServletRequest resp: HttpServletResponse
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Authenticator	authenticator
Observaciones			
La clase está decorada con la anotación @javax.servlet.annotation.WebServlet que declara que es un *servlet* para que esté disponible en el servidor.			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
LogoutServlet	public	Recibe las peticiones HTTP para extinguir una sesión y las procesa.	Hereda de javax.servlet.http.HttpServlet
Responsabilidades			
Número	**Descripción**		
1	Recibe las peticiones HTTP realizadas a través de la ruta establecida y las procesa.		
2	De estar presente, invalida el *token* de sesión que aparece en la petición.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
protected	void	doPost	req: HttpServletRequest resp: HttpServletResponse
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Authenticator	authenticator
Observaciones			
La clase está decorada con la anotación @javax.servlet.annotation.WebServlet que declara que es un *servlet* para que esté disponible en el servidor.			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
ReValidateAccessTokenServlet	public	Recibe las peticiones HTTP	Hereda de javax.servlet.http.HttpServlet

		para volver a validar un *token* y las procesa.	
Responsabilidades			
Número	**Descripción**		
1	Recibe las peticiones HTTP realizadas a través de la ruta establecida y las procesa.		
2	Si el *token* que se envía con la petición aparece en una colección de *tokens* de sesión eliminados recientemente (desaparecen definitivamente a los 20 segundos de ser invalidados) vuelve a convertirlo en un *token* de sesión válido.		
	Si se ha revalidado algún *token*, construye una respuesta con él y los datos del usuario autenticado en el formato correcto.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
protected	void	doPost	req: HttpServletRequest resp: HttpServletResponse
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Authenticator	authenticator
Observaciones			
La clase está decorada con la anotación @javax.servlet.annotation.WebServlet que declara que es un *servlet* para que esté disponible en el servidor.			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
RegistrationServlet	public	Recibe las peticiones HTTP para registrar nuevos usuarios y las procesa.	Hereda de javax.servlet.http.HttpServlet
Responsabilidades			
Número	**Descripción**		
1	Recibe las peticiones HTTP realizadas a través de la ruta establecida y las procesa.		
2	Comprueba que las credenciales no existan en la base de datos y que los datos introducidos son válidos.		
3	Si se cumplen los requisitos, inserta el nuevo usuario en la base de datos y genera un nuevo *token* para la cuenta identificada. Además, construye una respuesta con el *token* de sesión y los datos del usuario autenticado en el formato correcto.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
protected	void	doPost	req: HttpServletRequest resp: HttpServletResponse
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Authenticator	authenticator
Observaciones			

La clase está decorada con la anotación @javax.servlet.annotation.WebServlet que declara que es un *servlet* para que esté disponible en el servidor.

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
Authenticator	public	Se encarga de toda la lógica relativa a identificación de usuarios y *tokens* de sesión.	
Responsabilidades			
Número	**Descripción**		
1	Mantiene un caché de *tokens* de sesión válidos y se encarga de que se invaliden tras 15 minutos de inactividad.		
2	Mantiene un caché de *tokens* invalidados recientemente. Esos *tokens* desaparecen de forma definitiva a los 20 segundos de ser invalidados.		
3	Mantiene una relación uno a uno entre tokens y usuarios.		
4	Valida, de las formas establecidas durante la fase de identificación de requisitos, los datos introducidos para acceder con credenciales o registrarse.		
5	Encripta las contraseñas y guarda los nuevos usuarios en la base de datos del sistema.		
6	Se encarga de mantener una única estancia de su clase durante el ciclo de ejecución del programa y permite el acceso a ella por parte del resto del sistema.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
private	Authenticator	Authenticator	
public static	Authenticator	getInstance	
public	boolean	userHasActiveAccessToken	email: String
public	boolean	checkCredentials	credentials: Credentials
public	String	generateAccessToken	email: String
public	void	invalidateToken	token: String
public	boolean	reValidateToken	token: String
public	Optional <UserDTO>	getUserFromValidToken	accesToken: String
public	void	registerNewCredentials	credentials: RegistrationCredentials
private	String	hashPasswordPBKDF2	password: String salt: byte[]
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private	static	Authenticator	authenticatorInstance
private	final static	String	EMAIL_REG_EXP
private	final static	int	USERNAME_MAX_LENGTH
private	final static	int	PASSWORD_MIN_LENGTH
private	final static	int	PASSWORD_MAX_LENGTH
private		UsersCRUD	usersCRUD
private		Cache<String, UserDTO>	accessToken
private		Cache<String, UserDTO>	temporarilyInvalidatedAccessTokens
Observaciones			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
Credentials	public	Su única función es servir como objeto para la transferencia de datos dentro del sistema.	
Responsabilidades			
Número	**Descripción**		
1	Guarda toda la información relativa a unas credenciales de acceso y ofrece métodos para acceder a ellas.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	Credentials	Credentials	
public	String	getEmail	
public	void	setEmail	email: String
public	String	getPassword	
public	void	setPassword	password: String
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		String	email
private		String	password
Observaciones			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
RegistrationCredentials	public	Su única función es servir como objeto para la transferencia de datos dentro del sistema.	Hereda de Credentials
Responsabilidades			
Número	**Descripción**		
1	Guarda toda la información relativa a unas credenciales de registro y ofrece métodos para acceder a ellas.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	RegistrationCredentials	RegistrationCredentials	
public	String	getRepeatPassword	
public	void	setRepeatPassword	repeatPassword: String
public	String	getUsername	
public	void	setUsername	username: String
public	boolean	hasSucceded	

public	void	setSucceded	succeded: boolean
public	String	getErrorMessage	
public	void	setErrorMessage	message: String
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		String	repeatPassword
private		String	username
private		boolean	succeded
private		String	errorMessage
Observaciones			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
DBClient	public	Guarda una instancia del cliente de la base de datos.	
Responsabilidades			
Número	**Descripción**		
1	Guarda la única instancia del cliente de la base de datos que habrá en el ciclo de ejecución del programa y ofrece acceso a ella.		
2	Se encarga de que solo haya una única instancia de su clase y esté accesible al resto del programa.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
private	DBClient	DBClient	
public static	DBClient	getInstance	
public	MongoClient	getClient	
public	MongoDatabase	getDatabase	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private	static final	String	DB_USER
private	static final	String	DB_PASSWORD
private	static final	String	DB_NAME
private	static	DBClient	dbClientInstance
private		MongoClient	client
Observaciones			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
UsersCRUD	public	Es la clase que implementa todas las comunicaciones relativas a usuarios entre el resto de la aplicación y la BD.	
Responsabilidades			
Número	**Descripción**		
1	Implementa los métodos necesarios para todas las operaciones de BD relativas a usuarios que se necesitan en el sistema.		

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	UsersCRUD	UsersCRUD	
public	boolean	insertUser	newUser: UseDTO
public	boolean	updateUser	user: UserDTO
public	UserDTO	findUserByUsername	username: String
public	UserDTO	findUserByEmail	semail: String
public	List<UserDTO>	gatPagedUsersOrderedByScore	pageNumber: int pageSize: int
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private	static final	String	USERS_COLLECTION_NAME
private		DBClient	client
Observaciones			

Nombre	Tipo	Descripción	Implementa | Hereda de…
UserDTO	public	Sirve para la transferencia de datos de usuarios entre distintas pardes del sistema.	
Responsabilidades			
Número	**Descripción**		
1	Guardar todos los datos de un usuario y ofrece los métodos necesarios para acceder a ellos.		
2	Ser capaz de servir sus datos en formato JSON.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	UserDTO	UserDTO	_id: ObjectId username: String encryptedPassword: String salt: byte[] email: String score: int
public	UserDTO	UserDTO	username: String encryptedPassword: String salt: byte[] email: String
public	ObjectId	get_id	
public	void	set_id	_id: ObjectID
public	String	getUsername	
public	void	setUsername	username: String
public	String	getEntryptedPassword	
public	void	setEncryptedPassword	encryptedPassword: String
public	byte[]	getSalt	
public	void	setSalt	salt: byte[]

public	String	getEmail	
public	void	setEmail	email: String
public	int	getScore	
public	void	setScore	score: int
public	boolean	isInGame	
public	void	setInGame	inGame: Boolean
public	String	toJsonString	
public	String	toJsonStringUsernameScore	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		ObjectId	_id
private		String	username
private		String	encryptedPassword
private		byte[]	salt
private		String	email
private		int	score
private		boolean	inGame
Observaciones			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
UserGenerated GameDTO	public	Clase que guarda la información de una partida y sirve para transmitirla entre puntos distintos del sistema.	
Responsabilidades			
Número	**Descripción**		
1	Guardar toda la secuencia de estados de una partida en orden.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	UserGenerated GameDTO	UserGenerated GameDTO	_id: ObjectId states: List<GameStateDTO>
public	UserGenerated GameDTO	UserGenerated GameDTO	states: List<GameStateDTO>
private	void	addStates	states: List<GameStateDTO>
public	ObjectId	get_id	
public	void	set_id	_id: ObjectId
public	List<GameStateDTO>	getStates	
public	void	setStates	states: List<GameStateDTO>
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		ObjectId	_id
private		List<GameStateDTO>	states
Observaciones			

Nombre	Tipo	Descripción	Implementa | Hereda de…
GameStateDTO	public	Guarda la información de un estado dentro de una partida.	
Responsabilidades			
Número	**Descripción**		
1	Guarda el estatus de ese estado y la configuración del tablero en ese momento y ofrece métodos para acceder a esta información.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	GameStateDTO	GameStateDTO	board: Board
public	GameStateDTO	GameStateDTO	boardValues: List<Integer> status: Integer
public	List<Integer>	getBoardValues	
public	void	setBoardValues	boardValues: List<Integer>
public	Integer	getStatus	
public	void	setStatus	status: Integer
public	void	addBoardData	board: Board
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		List<Integer>	boardValues
private		Integer	status
Observaciones			

Nombre	Tipo	Descripción	Implementa | Hereda de…
GamesPool	public	Mantiene la única colección de partidas disponibles que deberá poder ser accedida desde todo el programa.	
Responsabilidades			
Número	**Descripción**		
1	Guarda un mapa de partidas, asociadas cada una a la sesión WebSocket del usuario que la está jugando.		
2	Implementa los métodos necesarios para que el resto del sistema acceda y manipule esta colección.		
3	Se encarga de que solo haya una única instancia de su clase y esté accesible al resto del programa.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
private	GamesPool	GamesPool	

public static	GamesPool	getInstance	
public	void	putGame	session: Session game: IServerGame
public	void	removeGame	session: Session
public	IServerGame	getGame	session: Session
public	Entry<Session, IServerGame>	findGameByPlayerEmail	email: String
public	Optional<IServerGame>	getAvailableGameOfKind	gameKind: String
public	int	size	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private	static	GamesPool	gamesPoolInstance
private		Map<Session, IServerGame>	games
Observaciones			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
ServerGameFactory	public abstract	Sirve para la creación de partidas de un tipo determinado.	
Responsabilidades			
Número	**Descripción**		
1	Implementa un método fábrica que recibe un parámetro con el tipo de juego que se desea y devuelve una instancia de la clase correcta con las estrategias concretas.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public static	IServerGame	createServerGame	gameKind: String websocket: AlphastarWebSocket
Observaciones			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
EServerGameKind	public enum	Guarda todos los tipos de partida válidos.	
Responsabilidades			
Número	**Descripción**		
1	Contiene todos los tipos de partida válidos que un cliente puede solicitar en forma de valores de una enumeración.		
Valores			
TRAINING_GAME			
REGULAR_PVP_GAME_TIME_PER_GAME			
RANKED_PVP_GAME_TIME_PER_GAME			
REGULAR_PVP_GAME_TIME_PER_MOVEMENT			

RANKED_PVP_GAME_TIME_PER_MOVEMENT
REGULAR_PVP_GAME_NO_TIME_LIMIT
RANKED_PVP_GAME_NO_TIME_LIMIT
Observaciones

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
ERivalDifficulty	public enum	Guarda los distintos valores de dificultad que puede tener una partida de entrenamiento.	
Responsabilidades			
Número	**Descripción**		
1	Contiene todos los valores correctos que un cliente puede solicitar para la dificultad del rival virtual en una partida de entrenamiento.		
2	Para cada uno de estos valores, guarda un número de iteraciones del algoritmo inteligente y el nombre de usuario que se mostrará.		
3	En caso de que esté disponible, cogerá ese número de iteraciones para cada nivel de dificultad del archivo de configuración.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
private	ERivalDifficulty	ERivalDifficulty	iterations: int mockName: String
private	void	init	
public	int	getIterations	
public	String	getMockName	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private	static final	Logger	LOGGER
private	static	Properties	prop
private		int	iterations
private		String	mockName
Valores			
EASY			
MEDIUM			
HARD			
PRO			
Observaciones			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
IServerGame	public interface	Establece una interfaz que deben implementar todas las clases partida.	
Responsabilidades			
Número	**Descripción**		

1	Establece algunos métodos que las distintas clases partidas deben implementar.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	void	init	
public	boolean	addPlayer	player: ServerPlayer
public	boolean	performMovement	session: Session data: JsonObject
public	boolean	needPlayers	
public	List<ServerPlayer>	getPlayers	
public	EServerGameKind	getGameKind	
public	boolean	hasPlayerWithEmail	email: String
public	void	playerLeft	email: String
public	void	notifyTimeOverForPlayer	player: EPlayer
Observaciones			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
AbstractServerGame	public abstract	Implementa la lógica común a todas las partidas.	Implementa IServerGame
Responsabilidades			
Número	**Descripción**		
1	Implementa la lógica que todas las partidas deben ejecutar y establece su orden de ejecución.		
2	Dispone métodos abstractos para que las clases que las clases hijas puedan implementar ciertas partes concretas de su lógica.		
3	Guarda los atributos que son comunes a todas las clases partida.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
protected	AbstractServer Game	AbstractServerGame	gameKind: EServerGameKind websocket: AlphastarWebSocket savingStrategy: IGameSaver scoringStrategy: IScoreCalculator timingStrategy: IGameTimer
public	void	init	
public	boolean	performMovement	session: Session data: JsonObject
protected	void	gameEnded	saveGame: Boolean
public	void	notifyTimeOverForPlayer	player: EPlayer
protected	void	createResponse	sessions: Session[]

			context: ServerGameContext movesNext: Object millisLeft: long
protected	void	createResponse	sessions: Session[] context: ServerGameContext movesNext: EPlayer
protected	JsonObject	createResponseJson	s: Session gameStatus: EGameStatus movesNext: Object teeJSON: JsonValue movesHistoryJSON: JsonValue movesHistory List<INode> millisLeft: long
protected	JsonObject	createResponseJson	s: Session gameStatus: EGameStatus movesNext: Object teeJSON: JsonValue movesHistoryJSON: JsonValue movesHistory List<INode>
protected	void	updateContext	context: ServerGameContext
public	EServerGameKind	getGameKind	
public	void	playerLeft	email: String
protected abstract	void	configureExternal Parameters	
protected abstract	void	setInitialBoardState	
protected abstract	void	createClientSetUp Response	
protected abstract	void	generatePossible Movements	context: ServerGameContext
protected abstract	ServerPlayer	getPlayerBySession	s: Session
protected abstract	Session[]	getSessionsArray	
protected abstract	EPlayer	getMovesNext	s: Session data: JsonObject
protected abstract	JsonObject	getPlayersNamesAsJson	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
protected	final static	String	GAME
protected		UCB	ucb
protected	final	String	URL
protected	final	boolean	USE_NN
protected		int	PlayerIterations

protected	final static	String	PARAM_BY_ID
protected	final static	String	PARAM_BY_BOARD
protected	final static	String	PARAM_BY_NAME
protected	final static	String	PARAM_DATA_NAME
private		EServerGameKind	gameKind
protected		AlphastarWebSocket	webSocket
protected		IGameSaver	savingStrategy
protected		IScoreCalculator	scoringStrategy
protected		IGameTimer	timignStrategy
protected		ServerGameContext	context
protected		boolean	hasEnded
Observaciones			
En la parte de métodos de esta tabla no serán excluidos los que ya aparezcan en la interfaz que implementa, ya que esta es una clase abstracta y podría no implementar algunos de esos métodos.			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
PVPGame	public	Contiene la lógica de las partidas multijugador.	Hereda de AbstractGameServer
Responsabilidades			
Número	**Descripción**		
1	Implementar toda la lógica necesaria para que se pueda jugar una partida entre dos usuarios humanos.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	PVPGame	PVPGame	gameKind: EServerGameKind webSocket: AlphastarWebSocket saveStrategy: IGameSaver scoringStrategy: IScoreCalculator timingStrategy: IGameTimer
public	boolean	addPlayer	player: ServerPlayer
public	boolean	needPlayers	
public	List<ServerPlayer>	getPlayers	
public	boolean	hasPlayerWithEmail	email: String
protected	void	generatePossibleMovements	context: ServerGameContext
protected	ServerPlayer	getPlayerBySession	session: Session
protected	Session[]	getSessionsArray	
protected	void	configureExternalParameters	
protected	void	setInitialBoardState	
protected	void	createClientSetUpResponse	
protected	EPlayer	getMovesNext	s: Session

			data: JsonObject
protected	void	gameEnded	saveGame: Boolean
public	void	playerLeft	email: String
protected	JsonObject	getPlayersNamesAsJson	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
protected	static	Logger	log
private		ServerPlayer	player1
private		ServerPlayer	player2
Observaciones			
En esta tabla aparecen todos los métodos implementados en esta tabla, para que se pueda saber cuáles de ellos se sobrescriben en esta clase.			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
TrainingGame	public	Contiene la lógica de las partidas de entrenamiento.	Hereda de AbstractGameServer
Responsabilidades			
Número	**Descripción**		
1	Implementar toda la lógica necesaria para que se pueda jugar una partida entre un usuario humano y el sistema inteligente actuando como rival.		
2	Generar, recurriendo al algoritmo inteligente basado en MCTS y redes neuronales, los datos que posteriormente serán usados para producir las recomendaciones que se presentan al usuario para cada movimiento.		
3	Configurarse en base a los parámetros que llegan a través de la conexión WS y a los que hay en el archivo de configuración del servidor (dificultad del rival, quién juega primero e iteraciones del algoritmo MCTS para las recomendaciones).		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	TrainingGame	TrainingGame	gameKind: EServerGameKind webSocket: AlphastarWebSocket saveStrategy: IGameSaver
public	boolean	addPlayer	player: ServerPlayer
public	boolean	needPlayers	
public	List<ServerPlayer>	getPlayers	
public	boolean	hasPlayerWithEmail	email: String
private	vodi	performRivalMovements	context: ServerGameContext sessions: Session[]
protected	void	generatePossibleMovements	context: ServerGameContext
protected	ServerPlayer	getPlayerBySession	session: Session
protected	Session[]	getSessionsArray	
protected	void	configureExternalParameters	

protected	void	setInitialBoardState	
protected	void	createClientSetUpResponse	
protected	EPlayer	getMovesNext	s: Session data: JsonObject
protected	JsonObject	getPlayersNamesAsJson	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
protected	static	Logger	log
private	final static	String	PARAM_PLAYING_AS_NAME
private	final static	String	PARAM_DISABLE_RECOMMENDATIONS_NAME
private	final static	String	PARAM_RIVAL_DIFFICULTY_NAME
private		String	startNodePath
private		int	playerIterationsWithRecommendations
private		boolean	disableRecommendations
private		ERivalDifficulty	rivalDifficulty
private		ServerPlayer	player
private		EPlayer	rivalPlayingAs
Observaciones			
De nuevo, en esta tabla aparecen todos los métodos implementados en esta tabla, para que se pueda saber cuáles de ellos se sobrescriben en esta clase.			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
ServerGameContext	public	Clase auxiliar que emplean las clases partida para guardar y transmitir datos.	
Responsabilidades			
Número	**Descripción**		
1	Guardar una serie de datos del contexto de una partida que reflejan su situación en el momento anterior a realizar un movimiento.		
2	Ofrecer los métodos necesarios para acceder a sus atributos.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	ITree	getTree	
public	void	setTree	tree: ITree
public	INode	getNode	
public	void	setNode	node: INode
public	INode	getStartingStateNode	
public	void	setStratingStateNode	startingStateNode: INode
public	List<INode>	getMovesNodes	
public	void	setMovesNodes	movesNodes: List<INode>
public	IMCTS	getMcts	

public	void	setMcts	mcts: IMCTS
public	int	getMovesNum	
public	int	incMovesNum	
public	void	setMovesNum	movesNum: int
public	boolean	getNotRecentConquer	
public	void	setNotRecentConquer	notRecentConquer: Boolean
public	String[][]	getLast	
public	void	setLast	last: String[][]
public	EPlayer	getPlayer	
public	void	setPlayer	player: EPlayer
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		ITree	tree
private		INode	node
private		INode	stringStateNode
private		List<INode>	movesNodes
private		IMCTS	mcts
private		int	movesNum
private		boolean	notRecentConquer
private		String[][]	last
private		EPlayer	player
Observaciones			
IMCTS es la interfaz que define los métodos públicos del algoritmo basado en árboles de búsqueda de Monte Carlo.			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
IGameSaver	public interface	Establece la base que deben implementar las clases que representen variantes distintas de la estrategia.	
Responsabilidades			
Número	**Descripción**		
1	Definir los métodos que deben implementar las distintas estrategias de guardado de partidas.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	boolean	saveGame	
Observaciones			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
DBGameSaver	public	Define la estrategia de guardado de partidas en	Implementa IGameSaver

		base de datos.	
Responsabilidades			
Número	**Descripción**		
1	Implementar la lógica necesaria para poder guardar partidas en la BD.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	boolean	saveGame	context: ServerGameContext
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		UserGeneratedGamesRepo	gamesRepo
Observaciones			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
JSONFileGameSaver	public	Define la estrategia de guardado de partidas en un archivo con formato JSON.	Implementa IGameSaver
Responsabilidades			
Número	**Descripción**		
1	Implementar la lógica necesaria para poder guardar partidas en un archivo con formato JSON.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	boolean	saveGame	context: ServerGameContext
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private	static final	String	USER_GENERATED_DATASET_PATH
Observaciones			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
IScoreCalculator	public interface	Establece la base que deben implementar las clases que representen variantes distintas de la estrategia.	
Responsabilidades			
Número	**Descripción**		
1	Definir los métodos que deben implementar las distintas estrategias de cálculo de puntuaciones al final de una partida.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y**

			Tipos
public	ScoreDifferenceDTO	calculateScoreDifferences	status: EGameStatus players: List<ServerPlayer>
Observaciones			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
RankedScoreCalculator	public	Define la estrategia de cálculo de puntuaciones para partidas clasificatorias.	Implementa IScoreCalculator
Responsabilidades			
Número	**Descripción**		
1	Implementar la lógica necesaria para poder calcular las puntuaciones al final de una partida clasificatoria.		
2	Calcular esas puntuaciones en base a la diferencia de puntos de los jugadores y empleando, si es posible, los parámetros presentes en el archivo de configuración del servidor.		
3	Actualizar las nuevas puntuaciones en la BD.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	RankedScore Calculator	RankedScore Calculator	
public	ScoreDifference DTO	calculateScoreDifferences	status: EGameStatus players: List<ServerPlayers>
private	int	calculateDifference	winnerScore: int loserScore: int
private	void	init	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private	static	Logger	log
private		int	baseScorePerGame
private		int	minScorePerGame
private		int	maxScorePerGame
private		int	minScore
private		double	scoreDifferenceRatio
private		UsersCRUD	usersCRUD
private		ScoreDifferenceDTO	scores
Observaciones			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
NullScoreCalculator	public	Define una estrategia	Implementa

		vacía de cálculo de puntuaciones.	IScoreCalculator
Responsabilidades			
Número	**Descripción**		
1	Recibir las peticiones que le lleguensin generar ningún error ni modificar nada.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	ScoreDifference DTO	calculateScoreDifferences	status: EGameStatus players: List<ServerPlayers>
Observaciones			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
ScoreDifference DTO	public	Clase para el intercambio de información entre clases distintas.	
Responsabilidades			
Número	**Descripción**		
1	Guardar información sobre las puntuaciones que se ha calculado que se deben sumar y restar a los jugadores.		
2	Implementar métodos para dar acceso a estos datos.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	ScoreDifference DTO	ScoreDifferenceDTO	scoreDifferenceEnabled: Boolean p1ScoreDifference: int p2ScoreDifference: int
public	bolean	isScoreDifferenceEnabled	
public	void	setScoreDifferenceEnabled	scoreDifferenceEnabled: boolean
public	int	getP1ScoreDifference	
public	void	setP1ScoreDifference	p1ScoreDifference: int
public	int	getP2ScoreDifference	
public	void	setP2ScoreDifference	p2ScoreDifference: int
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		boolean	scoreDifferenceEnabled
private		int	p1ScoreDifference
private		int	p2ScoreDifference
Observaciones			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**

IGameTimer	public interface	Establece la base que deben implementar las clases que representen variantes distintas de la estrategia.	
Responsabilidades			
Número	**Descripción**		
1	Definir los métodos que deben implementar las distintas estrategias de limitación de tiempos para los jugadores en una partida.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	void	subscribe	game: IServerGame
public	long	getMillisecondsLeftAfterMovementPerformed	player: EPlayer movesAgain: Boolean
public	void	unsubscribe	game: IServerGame
public	IServerGame	getSubscriber	
Observaciones			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
PerGameTimer	public	Define la estrategia para limitar el tiempo máximo que tiene cada jugador para una partida.	Implementa IGameTimer
Responsabilidades			
Número	**Descripción**		
1	Implementar la lógica necesaria para poder calcular el tiempo restante de cada jugador en un momento determinado.		
2	Notificar a la partida correspondiente cuando el tiempo de un jugador se acaba.		
3	Establecer el tiempo máximo por partida y jugador en función de lo que se establezca en el archivo de configuración.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	PerGameTimer	PerGameTimer	
public	void	subscribe	game: IServerGame
public	long	getMillisecondsLeftAfterMovementPerformed	player: EPlayer movesAgain: boolean
public	void	unsubscribe	game: IServerGame
public	IServerGame	getSubscriber	
private	void	init	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private	static	Logger	log

private		long	millisecondsPerGame PerPlayer
private		long	millisecondsTime Margin
private		IServerGame	subscriber
private		Timer	p1Timer
private		Timer	p2Timer
private		long	p1ElapsedMillis
private		long	p2ElapsedMillis
private		long	p1LastStarted Countdown
private		long	p2LastStarted Countdown
Observaciones			

Nombre	**Tipo**	**Descripción**	**Implementa | Hereda de…**
PerMovement Timer	public	Define la estrategia para limitar el tiempo máximo que tiene cada jugador por cada movimiento.	Implementa IGameTimer
Responsabilidades			
Número	**Descripción**		
1	Implementar la lógica necesaria para notificar a la partida correspondiente cuando a un jugador se le acaba el tiempo.		
2	Establecer el tiempo máximo por movimiento en función de lo que se establezca en el archivo de configuración del servidor.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	PerMovementTimer	PerMovementTimer	
public	void	subscribe	game: IServerGame
public	long	getMillisecondsLeftAfter MovementPerformed	player: EPlayer movesAgain: boolean
public	void	unsubscribe	game: IServerGame
public	IServerGame	getSubscriber	
private	void	init	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private	static	Logger	log
private		long	millisecondsPerMovement
private		long	millisecondsTime Margin
private		IServerGame	subscriber
private		Timer	p1Timer
private		Timer	p2Timer
Observaciones			

Nombre	Tipo	Descripción	Implementa | Hereda de…
NullGameTimer	public	Define la estrategia para las partidas sin límite de tiempo alguno.	Implementa IGameTimer
Responsabilidades			
Número	**Descripción**		
1	Recibir las peticiones de las partidas sin lanzar ningún error ni notificar en ningún momento que el tiempo se ha acabado.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public	void	subscribe	game: IServerGame
public	long	getMillisecondsLeftAfterMovementPerformed	player: EPlayer movesAgain: boolean
public	void	unsubscribe	game: IServerGame
public	IServerGame	getSubscriber	
Observaciones			

Nombre	Tipo	Descripción	Implementa | Hereda de…
JsonUtil	public abstract	Implementa utilidades para traducir ciertos elementos a formato JSON.	
Responsabilidades			
Número	**Descripción**		
1	Implementar métodos estáticos para convertir distintos objetos a formato JSON. Ofrecer esos métodos de utilidad en todas las partes del programa que los necesiten.		
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y Tipos**
public static	boolean	jsonToFile	filePath: String mcts: IMCTS
public static	boolean	gameMovesToJSONFile	filePath: String moves: List<INode>
private static	StringBuffer	gameMovesToJSONString	moves: List<INode>
private static	String	jsonFormat	json: JsonObject options: String…
private static	Map<String, Boolean>	buildConfig	options: String[]
public static	INode	jsonToDoubleArrayGame	game: String JSONFilePath: String
Observaciones			
Esta clase añade utilidades para traducir a JSON y de JSON algunas clases implementadas en el proyecto en que se desarrolló el algoritmo MCTS.			

Capítulo 8. Desarrollo de las Pruebas

En este capítulo se relatará cómo se han desarrollado las distintas pruebas empleadas para asegurar, en la medida de lo posible, el buen funcionamiento del sistema.

8.1 Pruebas Unitarias

Como el desarrollo de este sistema se ha llevado a cabo de forma incremental, cada nueva funcionalidad ha sido probada tras ser implementada. Tras cada prueba fallada, se ha depurado la parte correspondiente del sistema para localizar el defecto en el código. En ningún caso se ha procedido a añadir funcionalidades nuevas a la aplicación sin que las anteriores superasen todas las pruebas establecidas.

Las pruebas de este tipo se han centrado, principalmente, en aspectos de la interfaz de usuario; en el módulo de código que produce las recomendaciones y ejecuta las consultas sobre el árbol interactivo (probando con numerosos árboles generados por el algoritmo MCTS, más de 30); en probar el buen funcionamiento de la lógica de *tokens* de sesión, y en asegurarse de que la lógica implementada para las partidas multijugador y de entrenamiento funciona como se espera.

Además, por cada iteración del desarrollo del sistema, se han probado no solo las nuevas funcionalidades implementadas, sino también las anteriores, para evitar así cualquier posible fallo por un defecto en el código nuevo o modificado.

8.2 Pruebas de Integración y del Sistema

Aquí se mostrarán los resultados obtenidos al ejecutar las pruebas funcionales y de integración diseñadas anteriormente, en el Apartado 6.6.2.

Caso de Uso 1: Acceder Mediante Credenciales (Log In)	
Prueba	**Resultado Esperado**
El usuario introduce unas credenciales válidas.	El sistema crea una nueva sesión para ese usuario, generando un *token* de autenticación que enviará al cliente del usuario.
	Resultado Obtenido
	Efectivamente, el sistema añade al caché un nuevo *token* de sesión y se lo envía a la interfaz de navegador, que procede a mostrar la vista de usuario identificado.
Prueba	**Resultado Esperado**
El usuario deja vacío alguno de los campos obligatorios.	El sistema no genera el *token*, y advierte al usuario no identificado de su error.
	Resultado Obtenido
	No se añade ningún *token* a la colección y la interfaz *web*, además de no cambiar la vista, muestra un error al usuario advirtiéndole de que debe completar todos los campos.
Prueba	**Resultado Esperado**
El usuario introduce unas credenciales que no están presentes en el sistema.	Al contrastarlas con la base de datos, el sistema comprueba que esas credenciales no son válidas. También le muestra al usuario un mensaje de error y no genera ningún nuevo *token*.
	Resultado Obtenido
	El módulo correspondiente del sistema detecta que las credenciales no son correctas. No se genera un nuevo *token*. La interfaz gráfica no cambia de pantalla y muestra un mensaje de error que expone que la combinación email-contraseña empleada no es correcta.
Prueba	**Resultado Esperado**
El usuario introduce unas credenciales válidas, pero con una sesión activa.	El sistema comprueba en la base de datos que las credenciales son válidas, pero, tras buscar en la colección de *tokens* activos, encuentra que esa cuenta ya tiene un *token* asignado. No se genera ningún *token* y se advierte al usuario con un mensaje específico.
	Resultado Obtenido
	El sistema detecta, efectivamente, que las credenciales son válidas, pero ya hay una sesión activa con esas credenciales. No se genera ningún *token* nuevo. La pantalla que muestra la aplicación *web* no cambia y muestra el mensaje de error correspondiente.
Prueba	**Resultado Esperado**
El usuario cancela la operación.	No se aplica ningún cambio al sistema.
	Resultado Obtenido
	Efectivamente, no se aplica ningún cambio al sistema: ni a la colección de *tokens* en memoria, ni a la base de datos. Además, la interfaz de usuario pasa a mostrar el menú principal de acceso.

Prueba	**Resultado Esperado**
El usuario introduce credenciales válidas, pero la conexión con el servidor falla.	No se aplica ningún cambio al sistema.
	Resultado Obtenido
	Como se esperaba, no hay cambios en el sistema. La interfaz de navegador tampoco cambia de pantalla.
Prueba	**Resultado Esperado**
El usuario emplea las herramientas de desarrollador del navegador, por ejemplo, para intentar inhabilitar las validaciones.	El sistema realizará todas las validaciones igualmente, al estar presentes también en el servidor. Se abrirá una sesión nueva y se generará un *token* solo si la validación es exitosa, las credenciales son correctas y el usuario no tiene ya una sesión iniciada.
	Resultado Obtenido
	Las validaciones se llevan a cabo de igual forma en la parte del servidor. Si alguno de los requisitos establecidos no se cumple, no se genera ningún *token* y la interfaz no cambia de pantalla, además de mostrar el mensaje de error adecuado. De lo contrario, si los datos pasan todas las validaciones, se genera un nuevo *token* de sesión y la GUI pasa a mostrar las opciones de usuarios identificado.
Prueba	**Resultado Esperado**
La conexión con el servidor falla.	El sistema permanecerá sin cambios.
	Resultado Obtenido
	Efectivamente, no hay ningún cambio en el sistema.

Caso de Uso 2: Registrarse	
Prueba	**Resultado Esperado**
El usuario rellena todos los campos con datos correctos.	El sistema, tras validar esos datos, inserta al nuevo usuario en la base de datos. Tras esto, crea una sesión para él, generando un *token* nuevo y enviándoselo.
	Resultado Obtenido
	El sistema valida los datos, inserta el usuario en la BD y genera un nuevo *token*. La vista de la aplicación de navegador pasa a ser la de un usuario identificado.
Prueba	**Resultado Esperado**
El usuario deja vacío alguno (o varios) de los campos obligatorios.	Esto es detectado durante el proceso de validación, por lo que el sistema no crea ninguna cuenta nueva ni genera un *token* de sesión. Además, advierte al usuario de su error.
	Resultado Obtenido
	Efectivamente, no hay cambios en el sistema: ni en memoria ni en BD. La interfaz de usuario no cambia de pantalla y muestra el mensaje de error adecuado.
Prueba	**Resultado Esperado**
El usuario introduce datos incorrectos, al fallar alguno de los requisitos de validación establecidos previamente.	De nuevo, el sistema es capaz de detectar el error y no generará una nueva cuenta ni un *token* de sesión. También mostrará al usuario un mensaje de error adecuado a cada situación.

	Resultado Obtenido
	Como se espera, el sistema detecta el error. No se genera ningún *token* ni se inserta al usuario en la BD. No cambia la vista de la aplicación de navegador y se muestra un mensaje acorde con la situación.
Prueba	**Resultado Esperado**
El usuario cancela la operación.	El sistema no aplica ningún cambio.
	Resultado Obtenido
	Efectivamente, no aparecen cambios en el sistema. La interfaz gráfica pasa a mostrar el menú principal de inicio de sesión.
Prueba	**Resultado Esperado**
El usuario se sirve de las herramientas de desarrollador del navegador para intentar inhabilitar las validaciones.	El sistema realizará todas las validaciones igualmente, al estar presentes también en el servidor. Solo se insertará el nuevo usuario si los datos introducidos pasan todas las comprobaciones.
	Resultado Obtenido
	El sistema lleva a cabo las validaciones de igual forma. Si los datos cumplen todos los requisitos, sucede lo mismo que en la prueba en la que se introducen datos válidos. De lo contrario, si algún requisito no se cumple, lo que pasa es lo que ya se ha visto en la prueba de introducir datos incorrectos.
Prueba	**Resultado Esperado**
La conexión con el servidor falla.	El sistema permanecerá sin cambios.
	Resultado Obtenido
	El sistema no aplica ninguna modificación y la pantalla que muestra la interfaz *web* no cambia.

Caso de Uso 3: Cerrar Sesión	
Prueba	**Resultado Esperado**
El usuario identificado selecciona la opción correspondiente en el menú principal.	El sistema cancela el *token* de sesión perteneciente a ese usuario, lo que hará posible que acceda de nuevo con sus credenciales.
	Resultado Obtenido
	El *token* de ese usuario es invalidado. Ese *token*, efectivamente, ya no sirve para hacer peticiones al servidor. Al usuario se le muestra el menú de acceso, donde puede iniciar sesión de nuevo como se esperaba.
Prueba	**Resultado Esperado**
El usuario identificado pulsa la opción correcta, pero la conexión con el servidor falla.	La aplicación *web* muestra al usuario el menú de acceso. El sistema no cancelará el *token* de sesión, por lo que para entrar con las mismas credenciales deberá esperar a que se agote el tiempo de inactividad. Sí podrá entrar con otras credenciales o registrarse.
	Resultado Obtenido

	Sucede lo que se espera. La pantalla que muestra la interfaz es la del menú de acceso. El usuario no puede acceder con la cuenta cuya sesión acaba de cerrar, pero sí con otras credenciales o registrarse. Una vez se ha consumido lo que restase del tiempo de inactividad del *token*, podrá iniciar sesión de nuevo con la cuenta inicial.

Caso de Uso 4: Ver Ranking de Jugadores	
Prueba	**Resultado Esperado**
El usuario identificado selecciona la opción para ver el *ranking* de forma normal.	El sistema obtendrá la primera página del *ranking* de jugadores (con un máximo de 10 entradas) y se lo mostrará en la pantalla adecuada. Además, permitirá que el usuario se mueva a la página previa (si no se encuentra ya en la primera) y a la siguiente (si no se encuentra en la última). No se realizarán cambios en el sistema.
	Resultado Obtenido
	Efectivamente, la interfaz *web* solicita al servidor la información de la primera página del *ranking* con 10 jugadores máximo. Esta información se muestra en la tabla. Si el usuario avanza o retrocede de página, se vuelve a pedir la información de la página correspondiente y se muestra de forma correcta.
Prueba	**Resultado Esperado**
El usuario identificado pulsa la opción correspondiente, pero la conexión con el servidor falla.	La aplicación *web* muestra al usuario la pantalla de ver *ranking*, aunque vacía, al no haber podido recuperar la información. No se efectuará ningún cambio en el sistema.
	Resultado Obtenido
	La interfaz de navegador solicita al servidor los datos correspondientes, pero, como la conexión falla, este no puede responder. Se muestra la tabla de *ranking* de jugadores vacía.
Prueba	**Resultado Esperado**
El usuario identificado modifica los parámetros de la petición (número de página o entradas por página) empleando las herramientas para desarrolladores.	El sistema será capaz de mostrar igualmente el *ranking*, adaptándose a los parámetros. Si los parámetros se sitúan fuera de los límites de la colección de jugadores, el *ranking* aparecerá vacío. Si los parámetros son eliminados, el sistema dispondrá de unos por defecto (primera página y 10 entradas por página). El sistema permanecerá sin cambios.
	Resultado Obtenido
	De la forma esperada, si los nuevos parámetros son válidos, el servidor responde con la información solicitada y esta se muestra en la tabla de *ranking* de jugadores. Si los parámetros se sitúan fuera de la colección de usuarios, el servidor responderá con una lista vacía y la tabla aparecerá igualmente vacía. Si se eliminan los parámetros o no son enteros válidos, el sistema aplica los que aparecen por defecto (primera página y un máximo de 10 entradas), devolviendo los datos correspondientes. Estos datos aparecen en la tabla como se espera.

Caso de Uso 5: Ver Perfil Propio	
Prueba	**Resultado Esperado**
El usuario identificado abre su perfil desde el menú principal.	El sistema le mostrará en un cuadro de diálogo la información de su cuenta. No se realizarán cambios en el sistema.
	Resultado Obtenido
	La GUI muestra el cuadro de diálogo correspondiente al perfil con la información del usuario identificado. El sistema no realiza ningún cambio ni en memoria ni en BD.
Prueba	**Resultado Esperado**
El usuario identificado pulsa la opción correspondiente, pero la conexión con el servidor falla.	El sistema mostrará el cuadro de diálogo correspondiente, pero con datos por defecto. No se efectuarán cambios en el sistema.
	Resultado Obtenido
	De nuevo, la interfaz muestra el cuadro de diálogo del perfil, pero con los datos por defecto, si no se han podido recuperar los del usuario identificado.

Caso de Uso 6: Buscar Partida de Entrenamiento	
Prueba	**Resultado Esperado**
El usuario identificado selecciona la opción correspondiente.	El sistema comprobará que cuenta con un *token* activo y creará una partida nueva de entrenamiento para él. Tras un breve tiempo de carga, empezará la partida para el jugador.
	Resultado Obtenido
	Efectivamente, el sistema comprueba que el *token* es válido y crea una nueva partida de entrenamiento, en la que mete al usuario identificado. Después, envía a la máquina cliente la información de la partida iniciada y la interfaz de navegador la muestra.
Prueba	**Resultado Esperado**
El usuario identificado pulsa la opción correspondiente, pero la conexión con el servidor falla.	No se creará ninguna partida para el jugador. El sistema le mostrará un mensaje advirtiendo de que no ha sido posible establecer la conexión.
	Resultado Obtenido
	No se crea ninguna partida. La aplicación *web* no cambia la pantalla mostrada y emite un mensaje de error advirtiendo del fallo.
Prueba	**Resultado Esperado**
El usuario identificado pulsa la opción correspondiente, pero no cuenta con un *token* activo o este ha expirado.	No se creará ninguna partida para el jugador. El sistema le advertirá del error y le recomendará acceder de nuevo con sus credenciales.
	Resultado Obtenido
	De la forma esperada, el sistema detecta que no hay un *token* válido. De nuevo, la aplicación *web* no cambia la pantalla mostrada y emite un mensaje de error advirtiendo del fallo.

Prueba	**Resultado Esperado**
El usuario identificado modifica los parámetros de configuración que se envían junto con la petición, empleando las herramientas de desarrollador.	Si los parámetros introducidos representan opciones de configuración válidas, el sistema creará la partida de la forma solicitada. Si esos parámetros no son válidos, el sistema contará con una configuración por defecto y le creará una partida con esta configuración.
	Resultado Obtenido
	Si estos parámetros están dentro del conjunto de valores válidos, el sistema los interpreta de la forma adecuada. De lo contrario, la partida se configura con parámetros por defecto (solo se usarán los valores por defecto en aquellos parámetros que no puedan ser recuperados de la petición). Tras esto, la partida se inicia y la información es enviada al cliente.
Prueba	**Resultado Esperado**
El usuario identificado cancela la partida durante la pantalla de carga.	El sistema lo detecta y le lleva de nuevo al menú principal.
	Resultado Obtenido
	El cliente notifica al servidor de esto. La partida se elimina y la interfaz de navegador pasa a mostrar el menú principal de usuario identificado.

Caso de Uso 7: Configurar Partida de Entrenamiento	
Prueba	**Resultado Esperado**
El usuario identificado selecciona no modifica ningún parámetro.	La aplicación de navegador guardará y mostrará las configuraciones por defecto, para luego enviarlas a la hora de solicitar una partida.
	Resultado Obtenido
	Los parámetros que se guardan y se envían posteriormente al servidor son los mismos que aparecen en la pantalla por defecto.
Prueba	**Resultado Esperado**
El usuario identificado modifica algún parámetro (o varios, desde la interfaz).	La aplicación rastreará estos cambios y los guardará para luego usarlos para solicitar una partida. La configuración que guarda la aplicación se corresponderá con la que le muestra al usuario, aunque cambie de pantalla y luego vuelva.
	Resultado Obtenido
	La aplicación *web* detecta los cambios realizados. Estos cambios se mantienen y se envían al servidor de esta forma, aunque el usuario navegue entre menús.
Prueba	**Resultado Esperado**
El usuario identificado varía las opciones de configuración mediante las herramientas para desarrolladores del navegador.	Si estas son válidas, la interfaz las mostrará. Si son inválidas, no se puede predecir el comportamiento de la interfaz *web*, ya que el usuario puede, potencialmente, modificarlo todo. De todas formas, estas configuraciones se validarán en el servidor cuando se solicite una partida, donde habrá una configuración por defecto para estos casos.
	Resultado Obtenido

	La aplicación muestra en la pantalla de configuración las opciones si son válidas. Estas configuraciones se envían al servidor al buscar partida y, si alguna de ellas no está dentro del conjunto de valores válidos, se sustituye por el valor por defecto.

Caso de Uso 8: Buscar Partida Multijugador Normal	
Prueba	**Resultado Esperado**
El usuario identificado selecciona la opción correspondiente.	El sistema comprobará que cuenta con un *token* activo y creará una partida multijugador normal nueva para él, o lo introducirá en una ya creada del mismo tipo. Cuando se encuentre un rival que busca el mismo tipo de partida, empezará la partida para el jugador.
	Resultado Obtenido
	La validez del *token* es comprobada en el servidor. Si en la colección de partidas existe una del tipo solicitado que esté esperando por jugadores, el usuario se mete en esta. De lo contrario, se crea una nueva. La partida se inicia la información se envía al cliente para que se muestre en la interfaz.
Prueba	**Resultado Esperado**
El usuario identificado pulsa la opción correspondiente, pero la conexión con el servidor falla.	No se creará ninguna partida para el jugador. El sistema le mostrará un mensaje advirtiendo de que no ha sido posible establecer la conexión.
	Resultado Obtenido
	Efectivamente, no se crea ninguna partida. La aplicación *web* no cambia la pantalla mostrada y emite un mensaje de error advirtiendo del fallo.
Prueba	**Resultado Esperado**
El usuario identificado pulsa la opción correspondiente, pero no cuenta con un *token* activo o este ha expirado.	No se creará ninguna partida para el jugador. El sistema le advertirá del error y le recomendará acceder de nuevo con sus credenciales.
	Resultado Obtenido
	De la forma esperada, el sistema detecta que no hay un *token* válido. De nuevo, la aplicación *web* no cambia la pantalla mostrada y emite un mensaje de error advirtiendo del fallo y recomendando que inicie sesión de nuevo.
Prueba	**Resultado Esperado**
El usuario identificado intenta solicitar una partida cuando ya se encuentra jugando una partida multijugador.	El sistema lo detecta y le mostrará un mensaje de error diciéndole que no es posible jugar varias partidas multijugador de forma simultánea.
	Resultado Obtenido
	Cuando se envía la petición al servidor, este detecta que el *token* empleado está asociado a una partida en curso. La interfaz de navegador no cambia de pantalla y muestra un mensaje advirtiendo de que no se puede jugar varias partidas

	multijugador simultáneamente.
Prueba	**Resultado Esperado**
El usuario identificado modifica los parámetros de configuración que se envían junto con la petición, empleando las herramientas de desarrollador.	Si los parámetros introducidos representan opciones de configuración válidas, el sistema creará la partida de la forma solicitada. Si los parámetros no son válidos, la partida multijugador no se creará y el usuario permanecerá en el menú principal.
	Resultado Obtenido
	Si estos parámetros representan un tipo de partida válido (un tipo de partida que esté implementado), la partida se crea tal como se ha solicitado. De lo contrario, la partida no se crea y la GUI no cambia de pantalla.

Caso de Uso 9: Buscar Partida Multijugador Clasificatoria	
Prueba	**Resultado Esperado**
El usuario identificado selecciona la opción correspondiente.	El sistema comprobará que cuenta con un *token* activo y creará una partida multijugador clasificatoria nueva para él o, si hay una partida del mismo tipo esperando por otro jugador, lo introducirá en ella. Cuando se encuentre un rival que busca el mismo tipo de partida, empezará la partida para el jugador.
	Resultado Obtenido
	La validez del *token* es comprobada en el servidor. Si en la colección de partidas existe una del tipo solicitado (multijugador clasificatoria y con la limitación de tiempo especificada) que esté esperando por jugadores, el usuario se mete en esta. De lo contrario, se crea una nueva. La partida se inicia la información se envía al cliente para que se muestre en la interfaz.
Prueba	**Resultado Esperado**
El usuario identificado pulsa la opción correspondiente, pero la conexión con el servidor falla.	No se creará ninguna partida para el jugador. El sistema le mostrará un mensaje advirtiendo de que no ha sido posible establecer la conexión.
	Resultado Obtenido
	De la forma esperada, no se crea ninguna partida. La aplicación *web* no cambia la pantalla mostrada y emite un mensaje de error advirtiendo del fallo.
Prueba	**Resultado Esperado**
El usuario identificado pulsa la opción correspondiente, pero no cuenta con un *token* activo o este ha expirado.	No se creará ninguna partida para el jugador. El sistema le advertirá del error y le recomendará acceder de nuevo con sus credenciales.
	Resultado Obtenido
	Efectivamente el sistema detecta la no presencia de un *token* válido. De nuevo, la aplicación *web* no cambia la pantalla mostrada y emite un mensaje de error advirtiendo del fallo y recomendando que inicie sesión de nuevo.
Prueba	**Resultado Esperado**

El usuario identificado intenta solicitar una partida cuando ya se encuentra jugando una partida multijugador.	El sistema lo detecta y le mostrará un mensaje de error diciéndole que no es posible jugar varias partidas multijugador de forma simultánea.
	Resultado Obtenido
	Cuando se envía la petición al servidor, este detecta que el *token* empleado está asociado a una partida en curso (siempre que la partida sea de tipo multijugador). La interfaz de navegador no cambia de pantalla y muestra un mensaje advirtiendo de que no se puede jugar varias partidas multijugador simultáneamente.
Prueba	**Resultado Esperado**
El usuario identificado modifica los parámetros de configuración que se envían junto con la petición, empleando las herramientas de desarrollador.	Si los parámetros introducidos representan opciones de configuración válidas, el sistema creará la partida de la forma solicitada. Si los parámetros no son válidos, la partida multijugador no se creará y el usuario permanecerá en el menú principal.
	Resultado Obtenido
	Efectivamente, si estos parámetros representan un tipo de partida válido (uno de los que están implementados), la partida se crea tal como se ha solicitado. De lo contrario, la partida no se crea y la GUI no cambia de pantalla.

Caso de Uso 10: Escoger límite de tiempo	
Prueba	**Resultado Esperado**
El usuario identificado escoge una de las opciones disponibles en la interfaz.	Cuando busque partida, el sistema deberá asegurarse de que entrará en una partida del tipo exacto seleccionado.
	Resultado Obtenido
	Esta selección se envía al servidor. Efectivamente, el servidor se encarga de meter al usuario siempre en una partida del tipo especificado.
Prueba	**Resultado Esperado**
El usuario cambiar, empleando las herramientas de desarrollador, el tipo de partida que va a pedir.	Si el tipo de partida no es válido, no se creará ninguna partida para el jugador. Si el tipo de partida es válido, el sistema se asegurará de nuevo de que entre en una partida del mismo tipo que ha elegido.
	Resultado Obtenido
	El servidor solo mete al usuario identificado en una partida si el parámetro de tipo recibido coincide con uno de los tipos de partida implementados. Si no, el usuario no entra en partida y la pantalla mostrada en la interfaz *web* no cambia.

Caso de Uso 11: Jugar Partida	
Prueba	**Resultado Esperado**
El jugador realiza todos los movimientos de forma	El sistema garantizará que el juego se desarrolle como debe, controlando toda la lógica de turnos, puntuaciones, reglas del

correcta, empleando el tablero y el árbol interactivo (si está disponible), y sin agotar la limitación de tiempo si la hubiera.	juego, etc. Cuando no sea posible realizar más movimiento, el sistema decidirá quien ha ganado en función de las cajas cerradas, mostrando al ganador y al perdedor los mensajes de victoria y derrota, respectivamente. Estos mensajes irán acompañados de la puntuación perdida o ganada, si se trata de una partida clasificatoria. La secuencia de movimientos de esta partida se guardará en la base de datos.
	Resultado Obtenido
	El sistema ejecuta la lógica de la partida de la forma esperada. Controla en todo momento quién mueve y qué movimientos puede realizar. Si es una partida de entrenamiento con recomendaciones activadas, también genera los datos para producir esas recomendaciones. Una vez no hay más movimientos disponibles, el sistema se encarga de ejecutar la lógica de final de partida correspondiente al tipo de juego. Efectivamente, la secuencia de estados de la partida se guarda en la BD.
Prueba	**Resultado Esperado**
El jugador realiza los movimientos de forma correcta, pero agota su tiempo por partida o por movimiento.	El sistema será capaz de detectar esto, ejecutando la lógica correspondiente a cada tipo de partida para cuando esta acaba. El proceso se realizará de forma normal, repartiendo puntuaciones si procediese. La secuencia de movimientos no se guardará en la base de datos.
	Resultado Obtenido
	El sistema detecta que el tiempo de un jugador se ha acabado y ejecuta la lógica correspondiente al tipo de partida de forma normal. La partida no se guarda en la BD.
Prueba	**Resultado Esperado**
El jugador abandona la partida antes de que termine empleando la opción de salir al menú principal.	El sistema será capaz de detectar esto, ejecutando la lógica correspondiente a cada tipo de partida para cuando esta acaba. El proceso se realizará de forma normal, repartiendo puntuaciones si procediese. La secuencia de movimientos no se guardará en la base de datos. Por otra parte, la sesión del usuario no se cerrará y este podrá volver a jugar otra partida desde el menú principal.
	Resultado Obtenido
	Como se esperaba, el sistema detecta que el jugador ha abandonado la partida y ejecuta la lógica de final de partida de forma normal. La partida no se guarda en la BD. La interfaz de usuario pasa a mostrar el menú principal y, desde allí, el usuario puede solicitar otra partida.
Prueba	**Resultado Esperado**
El jugador abandona la partida antes de que termine cerrando el navegador.	El sistema será capaz de detectar esto, ejecutando la lógica correspondiente a cada tipo de partida para cuando esta acaba. El proceso se realizará de forma normal, repartiendo puntuaciones si procediese. La secuencia de movimientos no se guardará en la base de datos. Por otra parte, la sesión del usuario se cerrará y su *token* será cancelado. Si vuelve a abrir la aplicación, podrá acceder con sus credenciales de nuevo.
	Resultado Obtenido
	Como antes, el sistema detecta que el jugador ha

	abandonado la partida y ejecuta la lógica de final de partida de forma normal. La partida no se guarda en la BD. Además, el servidor se encarga de invalidar el *token* de sesión del usuario, que puede volver a acceder con sus credenciales si reabre la página.
Prueba	**Resultado Esperado**
El usuario pierde la conexión con el servidor.	El sistema será capaz de detectar esto, ejecutando la lógica correspondiente a cada tipo de partida para cuando esta acaba. El proceso se realizará de forma normal, repartiendo puntuaciones si procediese. La secuencia de movimientos no se guardará en la base de datos. Si recupera la conexión antes de que se supere el tiempo de inactividad de un *token* (15 minutos) podrá seguir con su sesión iniciada, aunque habrá perdido la partida.
	Resultado Obtenido
	El sistema lo detecta y entiende que el jugador ha abandonado la partida. Tras esto, ejecuta la lógica de final de partida de forma normal. La partida no se guarda en la BD. El *token* de sesión no se invalida.
Prueba	**Resultado Esperado**
El jugador intenta hacer trampas utilizando las herramientas para desarrolladores o un software externo. Podrá intentar realizar movimientos ilegales o decidir los de su rival.	El sistema será capaz de discernir qué movimientos son reglamentarios y de donde proceden. Si un jugador intenta realizar un movimiento que va contra las reglas del juego o intenta mover fuera de su turno, el sistema lo penalizará con la derrota. De nuevo, ejecutará la lógica correspondiente a cada tipo de partida para cuando esta acaba. El proceso se realizará de forma normal, repartiendo puntuaciones si procediese. La secuencia de movimientos no se guardará en la base de datos.
	Resultado Obtenido
	Efectivamente, el sistema es capaz de distinguir qué movimientos son válidos y están hechos por el jugador que debería, y cuáles no. Si un jugador intenta hacer trampas, es declarado perdedor y se ejecuta la lógica de final de partida de forma normal. La partida no se guarda en la BD. El *token* de sesión no se invalida.

Caso de Uso 12: Modificar Archivo de Configuración	
Prueba	**Resultado Esperado**
El administrador del sistema modifica uno, varios o todos los parámetros correctamente.	El servidor será capaz de extraer estos parámetros del archivo de configuración y adecuarse a todos ellos.
	Resultado Obtenido
	El servidor se configura atendiendo a los nuevos parámetros.
Prueba	**Resultado Esperado**
El administrador del sistema introducirá errores en uno, varios o todos los parámetros configurables.	El servidor será capaz de extraer del archivo todos aquellos parámetros que estén bien formados y adaptarse a ellos. Para los que, por alguna razón, no puedan ser leídos, se aplicarán las configuraciones por defecto establecidas en el código de

	los distintos subsistemas. El sistema avisará del error y se extraerá el mensaje a un archivo de registro.
	Resultado Obtenido
	El servidor utiliza aquellos parámetros que pueda extraer y utiliza los valores por defecto para el resto. También muestra el mensaje de error.
Prueba	**Resultado Esperado**
El administrador borrará o cambiará la localización del archivo de configuración.	El servidor detectará el fallo, empleando las configuraciones por defecto en todos los casos. El sistema avisará del error y se extraerá el mensaje a un archivo de registro.
	Resultado Obtenido
	El servidor se configura con los parámetros por defecto y avisa del error de la forma prevista.

Caso de Uso 13: Modificar el Contenido de la Base de Datos	
Prueba	**Resultado Esperado**
El administrador del sistema accederá a la interfaz *web* de MongoDB para insertar un usuario.	El sistema también permitirá acceder con las credenciales de esta nueva cuenta.
	Resultado Obtenido
	Las credenciales introducidas se pueden utilizar para iniciar sesión, de la misma forma que se puede con una cuenta registrada por la vía normal (a través de la aplicación).
Prueba	**Resultado Esperado**
El administrador del sistema accederá a la interfaz *web* de MongoDB para modificar un usuario.	Los datos que mostrará el sistema para esta cuenta serán los nuevos datos introducidos, tanto en el perfil como en el *ranking* de jugadores. Además, de haber cambiado las credenciales, las anteriores quedarán inservibles y solo será posible acceder con las nuevas.
	Resultado Obtenido
	Efectivamente, los datos que muestra la aplicación y que utiliza para el acceso son los que el administrador ha introducido. Si se cambian email o nombre de usuario, estos pasan a estar disponible a la hora de registrarse.
Prueba	**Resultado Esperado**
El administrador del sistema accederá a la interfaz *web* de MongoDB para borrar un usuario.	El sistema aplicará los cambios: las credenciales de esta antigua cuenta ya no podrán usarse para iniciar sesión en la aplicación y, tanto su nombre de usuario como su email, volverán a estar disponibles para usar durante un proceso de registro.
	Resultado Obtenido
	Las credenciales del usuario borrado no se pueden usar para iniciar sesión, y su nombre y email pueden ser empleados a la hora de registrase en la aplicación. Esta cuenta borrada tampoco aparece en el *ranking* de jugadores.

8.3 Pruebas de Usabilidad y Accesibilidad

Esta sección servirá para exponer los resultados de las pruebas determinadas para evaluar la usabilidad, la funcionalidad, el aspecto y la accesibilidad de la aplicación. En esta sección se incluyen tanto pruebas con usuarios reales de distintas tipologías como una serie de verificaciones técnicas.

8.3.1 Pruebas de Usabilidad

En este apartado se mostrarán los resultados de las pruebas con usuarios reales, definidas por los cuestionarios y procedimientos ya fijados en el Apartado 6.6.3.

Además de estas pruebas definidas en un inicio, se realizaron otras a mitad del desarrollo, cuando únicamente estaban implementadas las partidas contra el rival artificial. El objetivo de estas pruebas fue comprobar si la herramienta (las partidas de entrenamiento, la exploración el árbol y las recomendaciones) ayudaba al aprendizaje de los usuarios humanos y, de paso, preguntar a los probadores qué aspectos se podrían mejorar con respecto a la usabilidad.

Durante esas pruebas, los usuarios se quejaron, principalmente, de que no se habían dado cuenta de algunos de los movimientos del rival. Para solucionar esto, se decidió implementar un historial de movimientos y, además, una pantalla de espera más clara, para cuando es el turno del rival. También en ese momento percibí que algunos usuarios se frustraban, ya que no eran capaces de ganar a la IA sin disponer de recomendaciones. Para solucionar esto, se ha añadido una dificultad configurable para el rival virtual. Para información más detallada sobre estas pruebas, se puede consultar el artículo publicado al respecto **[14]**.

A continuación, se mostrarán los resultados de las pruebas con usuarios reales ya previstas en el capítulo de diseño.

8.3.1.1 Usuario 1

8.3.1.1.1 Preguntas de carácter general

Este usuario es una persona joven, acostumbrada al manejo de ordenadores, y jugador online habitual.

¿Usa un ordenador frecuentemente?
1. **Todos los días** 2. Varias veces a la semana 3. Ocasionalmente 4. Nunca o casi nunca
¿Cómo describiría su experiencia cuando usa un ordenador?
1. Tediosa 2. Indiferente

3. Amena **4. Entretenida**
¿Juega a juegos de mesa?
1. Sí, varias veces a la semana 2. Sí, varias veces al mes **3. Con muy poca frecuencia** 4. No, nunca
¿Juega juegos online?
5. Sí, varias veces a la semana 6. Sí, varias veces al mes 7. Con muy poca frecuencia 8. No, nunca
¿Qué busca Vd. Principalmente en un programa?
5. Que sea fácil de usar **6. Que sea intuitivo** 7. Que sea rápido 8. Que tenga todas las funciones necesarias

8.3.1.1.2 Actividades guiadas

Estas son las actividades que realizará el usuario:

- Registrarse en la aplicación
- Cerrar sesión
- Volver a iniciar sesión con sus credenciales
- Buscar su perfil para ver cuántos puntos tienen
- Jugar una partida de entrenamiento contra el rival virtual, intentando aprender de las recomendaciones
- Jugar una partida, del tipo que elijan, online
- Acceder al *ranking* de usuarios y localizar cuánta puntuación tienen tras la partida

8.3.1.1.3 Preguntas Cortas sobre la Aplicación y Observaciones

Facilidad de Uso	Siempre	Frecuentemente	Ocasionalmente	Nunca
¿Sabe dónde está dentro de la aplicación?	X			
¿Existe ayuda para las funciones en caso de que tenga dudas?		X		
¿Le ha resultado de utilidad esa ayuda?	X			
¿Le resulta sencillo el uso de la aplicación?		X		

¿Ha sabido identificar las distintas opciones disponibles a lo largo de la aplicación?	X			
¿Le ha resultado fácil, una vez dentro, jugar una partida?	X			
¿Le ha resultado fácil entender las recomendaciones de la aplicación?		X		
Funcionalidad	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Funciona cada tarea como Vd. Espera?	X			
¿El tiempo de respuesta de la aplicación es muy grande?			X	
¿Ha sentido que la aplicación respondía a sus comandos como debería?	X			
¿Ha encontrado difícil de batir al rival virtual?		X		

Calidad del Interfaz				
Aspectos gráficos	**Muy Adecuado**	**Adecuado**	**Poco Adecuado**	**Nada Adecuado**
El tipo y tamaño de letra es			X	
Los iconos e imágenes usados son		X		
Los colores empleados son	X			
El dibujo del tablero es		X		
La disposición del árbol es		X		
Diseño de la Interfaz		**Si**	**No**	**A veces**
¿Le resulta fácil de usar?		X		
¿El diseño de las pantallas es claro y atractivo?				X
¿Cree que el programa está bien estructurado?		X		
¿Cree que los menús están bien diseñados?		X		
¿Es adecuada la cantidad de opciones en cada menú?		X		
¿Le ha resultado agradable el diseño de la pantalla de juego?		X		

Observaciones
Cualquier comentario del usuario El usuario opina que se debería aumentar el tamaño de letra 1 punto. También cree que mejoraría la usabilidad usar el icono de "ir atrás" en lugar del de "cerrar" para el botón que permite abandonar una partida en curso.

8.3.1.1.4 Cuestionario para el Responsable de las Pruebas

Aspecto Observado	**Notas**
El usuario comienza a usar las distintas funcionalidades rápidamente	Sí.

Tiempo en realizar cada tarea	En los menús muy breve y sin necesidad de explicaciones. En la pantalla de juego muy breve.
Errores leves cometidos	Intentar usar la información y consultas del árbol sin apoyarse en pantalla de ayuda. En un principio, no se dio cuenta de que tanto el árbol como el historial cuentan con un *zoom* que permite examinarlos más fácilmente.
Errores graves cometidos	-
El usuario comprende las recomendaciones	Sí.
El usuario se atreve a explorar el árbol	Sí, aunque sin detenerse a buscar información en la pantalla de ayuda.
El usuario ha disfrutado la experiencia de juego	Sí. Sobre todo, las partidas contra otro jugador, ya que son más rápidas y dinámicas.

8.3.1.2 Usuario 2

8.3.1.2.1 Preguntas de carácter general

El siguiente usuario es una persona joven, acostumbrada a trabajar con ordenadores, pero jugador online ocasional.

¿Usa un ordenador frecuentemente?

1. **Todos los días**
2. Varias veces a la semana
3. Ocasionalmente
4. Nunca o casi nunca

¿Cómo describiría su experiencia cuando usa un ordenador?

1. Tediosa
2. Indiferente
3. Amena
4. **Entretenida**

¿Juega a juegos de mesa?

1. Sí, varias veces a la semana
2. **Sí, varias veces al mes**
3. Con muy poca frecuencia
4. No, nunca

¿Juega juegos online?

1. Sí, varias veces a la semana
2. **Sí, varias veces al mes**
3. Con muy poca frecuencia
4. No, nunca

¿Qué busca Vd. Principalmente en un programa?
1. Que sea fácil de usar 2. Que sea intuitivo 3. Que sea rápido **4. Que tenga todas las funciones necesarias**

8.3.1.2.2 Actividades guiadas

Estas son las actividades que realizará el usuario:

- Registrarse en la aplicación
- Cerrar sesión
- Volver a iniciar sesión con sus credenciales
- Buscar su perfil para ver cuántos puntos tienen
- Jugar una partida de entrenamiento contra el rival virtual, intentando aprender de las recomendaciones
- Jugar una partida, del tipo que elijan, online
- Acceder al *ranking* de usuarios y localizar cuánta puntuación tienen tras la partida

8.3.1.2.3 Preguntas Cortas sobre la Aplicación y Observaciones

Facilidad de Uso	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Sabe dónde está dentro de la aplicación?	X			
¿Existe ayuda para las funciones en caso de que tenga dudas?			X	
¿Le ha resultado de utilidad esa ayuda?	X			
¿Le resulta sencillo el uso de la aplicación?	X			
¿Ha sabido identificar las distintas opciones disponibles a lo largo de la aplicación?		X		
¿Le ha resultado fácil, una vez dentro, jugar una partida?	X			
¿Le ha resultado fácil entender las recomendaciones de la aplicación?	X			
Funcionalidad	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Funciona cada tarea como Vd. Espera?		X		
¿El tiempo de respuesta de la aplicación es muy grande?		X		
¿Ha sentido que la aplicación respondía a sus comandos como debería?		X		
¿Ha encontrado difícil de batir al rival virtual?		X		
Calidad del Interfaz				

Aspectos gráficos	Muy Adecuado	Adecuado	Poco Adecuado	Nada Adecuado
El tipo y tamaño de letra es		X		
Los iconos e imágenes usados son		X		
Los colores empleados son		X		
El dibujo del tablero es		X		
La disposición del árbol es	X			
Diseño de la Interfaz		**Si**	**No**	**A veces**
¿Le resulta fácil de usar?		X		
¿El diseño de las pantallas es claro y atractivo?				X
¿Cree que el programa está bien estructurado?				X
¿Cree que los menús están bien diseñados?		X		
¿Es adecuada la cantidad de opciones en cada menú?		X		
¿Le ha resultado agradable el diseño de la pantalla de juego?		X		
Observaciones				
El botón para abandonar la partida debería preguntarte si de verdad quieres salir, para no irte sin querer.				

8.3.1.2.4 Cuestionario para el Responsable de las Pruebas

Aspecto Observado	Notas
El usuario comienza a usar las distintas funcionalidades rápidamente	Sí. Ha entendido las funcionalidades del menú y de la pantalla de juego sin necesidad de ayuda.
Tiempo en realizar cada tarea	Muy breve.
Errores leves cometidos	-
Errores graves cometidos	Salir de una partida de entrenamiento sin querer empleando el botón para abandonar partidas.
El usuario comprende las recomendaciones	Sí.
El usuario se atreve a explorar el árbol	Sí. Además, ha decidido buscar en la ventana de ayudas para entender ciertas informaciones.
El usuario ha disfrutado la experiencia de juego	Sí. De nuevo, la parte que más entretiene es jugar contra otros usuarios humanos. Aunque también tiene ganas de jugar contra la IA e intentar ganar.

8.3.1.3 Usuario 3

8.3.1.3.1 Preguntas de carácter general

Esta es una persona adulta, jubilada y que casi nunca usa ordenadores, pero emplea regularmente un *smartphone* en su vida cotidiana.

¿Usa un ordenador frecuentemente?

1. Todos los días 2. Varias veces a la semana 3. Ocasionalmente **4. Nunca o casi nunca**
¿Cómo describiría su experiencia cuando usa un ordenador?
1. Tediosa **2. Indiferente** 3. Amena 4. Entretenida
¿Juega a juegos de mesa?
1. Sí, varias veces a la semana 2. Sí, varias veces al mes 3. Con muy poca frecuencia **4. No, nunca**
¿Juega juegos online?
1. Sí, varias veces a la semana 2. Sí, varias veces al mes 3. Con muy poca frecuencia **4. No, nunca**
¿Qué busca Vd. Principalmente en un programa?
1. Que sea fácil de usar 2. Que sea intuitivo 3. Que sea rápido 4. Que tenga todas las funciones necesarias

8.3.1.3.2 Actividades guiadas

Estas son las actividades que realizará el usuario:

- Registrarse en la aplicación
- Cerrar sesión
- Volver a iniciar sesión con sus credenciales
- Buscar su perfil para ver cuántos puntos tienen
- Jugar una partida de entrenamiento contra el rival virtual, intentando aprender de las recomendaciones
- Jugar una partida, del tipo que elijan, online
- Acceder al *ranking* de usuarios y localizar cuánta puntuación tienen tras la partida

8.3.1.3.3 Preguntas Cortas sobre la Aplicación y Observaciones

Facilidad de Uso	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Sabe dónde está dentro de la aplicación?		X		
¿Existe ayuda para las funciones en caso de que tenga dudas?		X		
¿Le ha resultado de utilidad esa ayuda?		X		
¿Le resulta sencillo el uso de la aplicación?	X			
¿Ha sabido identificar las distintas opciones disponibles a lo largo de la aplicación?	X			
¿Le ha resultado fácil, una vez dentro, jugar una partida?	X			
¿Le ha resultado fácil entender las recomendaciones de la aplicación?			X	
Funcionalidad	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Funciona cada tarea como Vd. Espera?	X			
¿El tiempo de respuesta de la aplicación es muy grande?			X	
¿Ha sentido que la aplicación respondía a sus comandos como debería?	X			
¿Ha encontrado difícil de batir al rival virtual?	X			

Calidad del Interfaz				
Aspectos gráficos	**Muy Adecuado**	**Adecuado**	**Poco Adecuado**	**Nada Adecuado**
El tipo y tamaño de letra es		X		
Los iconos e imágenes usados son	X			
Los colores empleados son	X			
El dibujo del tablero es	X			
La disposición del árbol es		X		
Diseño de la Interfaz		**Si**	**No**	**A veces**
¿Le resulta fácil de usar?		X		
¿El diseño de las pantallas es claro y atractivo?		X		
¿Cree que el programa está bien estructurado?		X		
¿Cree que los menús están bien diseñados?		X		
¿Es adecuada la cantidad de opciones en cada menú?		X		
¿Le ha resultado agradable el diseño de la pantalla de juego?		X		

Observaciones
-

8.3.1.3.4 Cuestionario para el Responsable de las Pruebas

Aspecto Observado	Notas
El usuario comienza a usar las distintas funcionalidades rápidamente	Sí, ha necesitado algo de ayuda porque no habla inglés. De todas formas, ha entendido la mayoría de las opciones de los menús sin problema.
Tiempo en realizar cada tarea	No ha tardado en entender las distintas funcionalidades.
Errores leves cometidos	-
Errores graves cometidos	-
El usuario comprende las recomendaciones	Sí.
El usuario se atreve a explorar el árbol	Sí, aunque poco, de forma tímida.
El usuario ha disfrutado la experiencia de juego	Sí. De hecho, ha mostrado ganas de competir contra la IA.

8.3.1.4 Usuario 4

8.3.1.4.1 Preguntas de carácter general

El último usuario es tiene el perfil de persona adulta, en activo y que usa ordenadores de forma regular en su trabajo.

¿Usa un ordenador frecuentemente?
1. Todos los días 2. Varias veces a la semana 3. Ocasionalmente 4. Nunca o casi nunca
¿Cómo describiría su experiencia cuando usa un ordenador?
1. Tediosa 2. Indiferente 3. Amena **4. Entretenida**
¿Juega a juegos de mesa?
1. Sí, varias veces a la semana 2. Sí, varias veces al mes **3. Con muy poca frecuencia** 4. No, nunca
¿Juega juegos online?
1. Sí, varias veces a la semana 2. Sí, varias veces al mes

3. Con muy poca frecuencia 4. No, nunca
¿Qué busca Vd. Principalmente en un programa?
1. Que sea fácil de usar 2. Que sea intuitivo 3. Que sea rápido 4. Que tenga todas las funciones necesarias

8.3.1.4.2 Actividades guiadas

Estas son las actividades que realizará el usuario:

- Registrarse en la aplicación
- Cerrar sesión
- Volver a iniciar sesión con sus credenciales
- Buscar su perfil para ver cuántos puntos tienen
- Jugar una partida de entrenamiento contra el rival virtual, intentando aprender de las recomendaciones
- Jugar una partida, del tipo que elijan, online
- Acceder al *ranking* de usuarios y localizar cuánta puntuación tienen tras la partida

8.3.1.4.3 Preguntas Cortas sobre la Aplicación y Observaciones

Facilidad de Uso	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Sabe dónde está dentro de la aplicación?		X		
¿Existe ayuda para las funciones en caso de que tenga dudas?		X		
¿Le ha resultado de utilidad esa ayuda?		X		
¿Le resulta sencillo el uso de la aplicación?	X			
¿Ha sabido identificar las distintas opciones disponibles a lo largo de la aplicación?	X			
¿Le ha resultado fácil, una vez dentro, jugar una partida?	X			
¿Le ha resultado fácil entender las recomendaciones de la aplicación?	X			
Funcionalidad	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Funciona cada tarea como Vd. Espera?	X			
¿El tiempo de respuesta de la aplicación es muy grande?				X
¿Ha sentido que la aplicación	X			

respondía a sus comandos como debería?				
¿Ha encontrado difícil de batir al rival virtual?	X			

Calidad del Interfaz				
Aspectos gráficos	**Muy Adecuado**	**Adecuado**	**Poco Adecuado**	**Nada Adecuado**
El tipo y tamaño de letra es		X		
Los iconos e imágenes usados son	X			
Los colores empleados son	X			
El dibujo del tablero es	X			
La disposición del árbol es		X		
Diseño de la Interfaz		**Si**	**No**	**A veces**
¿Le resulta fácil de usar?		X		
¿El diseño de las pantallas es claro y atractivo?		X		
¿Cree que el programa está bien estructurado?		X		
¿Cree que los menús están bien diseñados?		X		
¿Es adecuada la cantidad de opciones en cada menú?		X		
¿Le ha resultado agradable el diseño de la pantalla de juego?		X		
Observaciones				
-				

8.3.1.4.4 Cuestionario para el Responsable de las Pruebas

Aspecto Observado	**Notas**
El usuario comienza a usar las distintas funcionalidades rápidamente	Sí, de nuevo sin ayuda.
Tiempo en realizar cada tarea	No ha tardado en entender las distintas funcionalidades.
Errores leves cometidos	-
Errores graves cometidos	-
El usuario comprende las recomendaciones	No ha mostrado problemas para entenderlas.
El usuario se atreve a explorar el árbol	Sí. Ha intentado entender las distintas posibilidades a través del árbol.
El usuario ha disfrutado la experiencia de juego	Sí. Ha disfrutado tanto competir contra otro humano como contra la IA.

8.3.1.5 Conclusiones

En general, la reacción de los usuarios frente a la aplicación ha sido aceptablemente positiva. Han coincidido mayoritariamente en que su uso es sencillo y es atractiva a la vista. Además, se les ha visto bastante involucrados en el juego a la hora de competir entre ellos principalmente,

aunque también contra la inteligencia artificial. Todos han tenido una actitud positiva hacia las recomendaciones, valorando que eran útiles y que, en ciertos momentos, les han mostrado estrategias de las que no se habrían dado cuenta de otra forma (lo que les ayuda a mejorar su habilidad en el juego). Por otra parte, el árbol, al ser algo más complejo, no atrae tanto la atención de los usuarios. Pese a esto, tras leer la ventana de ayuda, la mayoría son capaces de entender toda la información sin problema, aunque pocos se aventuran a explorarlo en profundidad en un uso ligero de la aplicación.

Ninguno de los usuarios ha cometido muchos errores, todos han entendido de forma rápida las distintas funcionalidades que la herramienta ofrecen. Para atajar el primero de los errores que se ha detectado en uno de los usuarios (que no se percata de que el árbol tiene zoom e intenta analizar la información sin el apoyo de la ventana de ayuda) quizás sería interesante implementar un breve cuadro de diálogo que "chivase" a los usuarios estos aspectos al entrar en una partida. Respecto al segundo error, el de abandonar una partida por pulsar el botón sin querer, podría añadirse otro pequeño diálogo que advirtiese de lo que eso significa y se asegure de que el usuario de verdad quiere salir.

8.3.2 Pruebas de Accesibilidad

A continuación, se detallarán las pruebas realizadas para asegurarnos de que la aplicación cumple los estándares de accesibilidad en la medida de lo posible. Por ejemplo, al ser un juego, gran parte de esta aplicación se basa en información visual, por lo que no tiene sentido intentar usarla a través de navegadores por voz o herramientas similares. Tampoco se probará la aplicación en escala de grises porque es prácticamente necesario usarla en pantallas con color, ya que ofrece gran cantidad de información, que sería imposible explicar sencillamente de otra forma, mediante códigos de colores.

8.3.2.1 Revisión Manual

La aplicación *web* ha sido probada empleando cuatro navegadores distintos: Chrome, Edge, Firefox y Opera. Las versiones empleadas han sido, respectivamente, la 97.0.4692.71, la 97.0.1072.69, la 95.0.2 y la 82.0.4227.32

Para este proceso, se ha establecido una muestra de pantallas representativas de la aplicación: el formulario de registro, el menú principal de usuarios identificado, el *ranking* de jugadores y la pantalla de juego, tanto con la vista de tablero interactivo como con la del árbol de posibles movimientos.

Para cada una de estas pantallas distintas se han realizado las siguientes pruebas:

- Aumentar y reducir el *zoom* del navegador.
- Usar el teclado (*tab*, *shitf+tab* e *intro*) para navegar por los menús.
- Poner el navegador a pantalla completa.
- Probar distintas resoluciones: 800x600, 1024x768, 1152x864, 1280x720, 1280x768, 1280x960, 1280x1024, 1600x1200, 1920x1080 (FHD), 2048x1080 (2K) y 3840x2160 (4K).
- Emplear pantallas de 15, 17.3 y 28 pulgadas.
- Aumentar y disminuir el tamaño de la ventana numerosas veces.

La interfaz *web* de la aplicación ha superado todas las pruebas anteriores de manera satisfactoria. Únicamente se requerirá un *scroll*, siempre vertical, para los menús en aquellas pantallas que cuenten con una de las dos resoluciones inferiores probadas (800x600 y 1024x768). Para evitar que el tablero se vea raro, ya que no es un elemento HTML, sino que se dibuja con un tamaño fijo en un *canvas*, se ha implementado que el propio tablero calcule el espacio que puede ocupar en función del tamaño de la ventana de navegador en la que se muestre. También se ha hecho que sea capaz de recalcular su tamaño y redibujarse cada vez que se cambian las dimensiones de la ventana. En la vista de árbol, se ha implementado un *zoom* propio de la aplicación para poder tener una vista más general o acercarse a elementos concretos. El usuario podrá usarlo en cualquier momento.

8.3.2.2 Revisión Automática

En este paso, se han empleado la extensión que WAVE ofrece para Chrome y Lighthouse, otra extensión, de código abierto y desarrollada por la comunidad, que permite evaluar varios aspectos de una *web*, entre ellos, la accesibilidad.

Lighthouse ofrece, para todas las pantallas de la aplicación que deja probar (básicamente, los distintos menús), una puntuación de 100/100 en cuanto a accesibilidad, como se puede ver en la Figura 8.1.

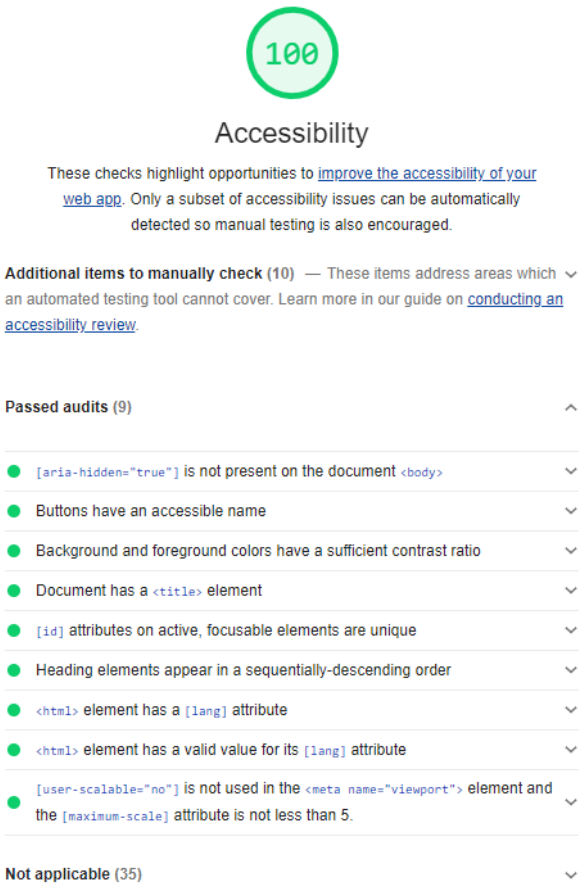


Figura 8.1. Informe de Lighthouse sobre la accesibilidad de nuestra aplicación web

A continuación, se mostrarán los resultados obtenidos con WAVE y, tras ello, se explicarán y se describirán las acciones tomadas para corregir los distintos errores encontrados. Nótese que la disposición de los elementos en cada página que se ve en las imágenes siguientes no es exactamente la que les aparece a los usuarios, ya que está modificada por los elementos que añade WAVE.

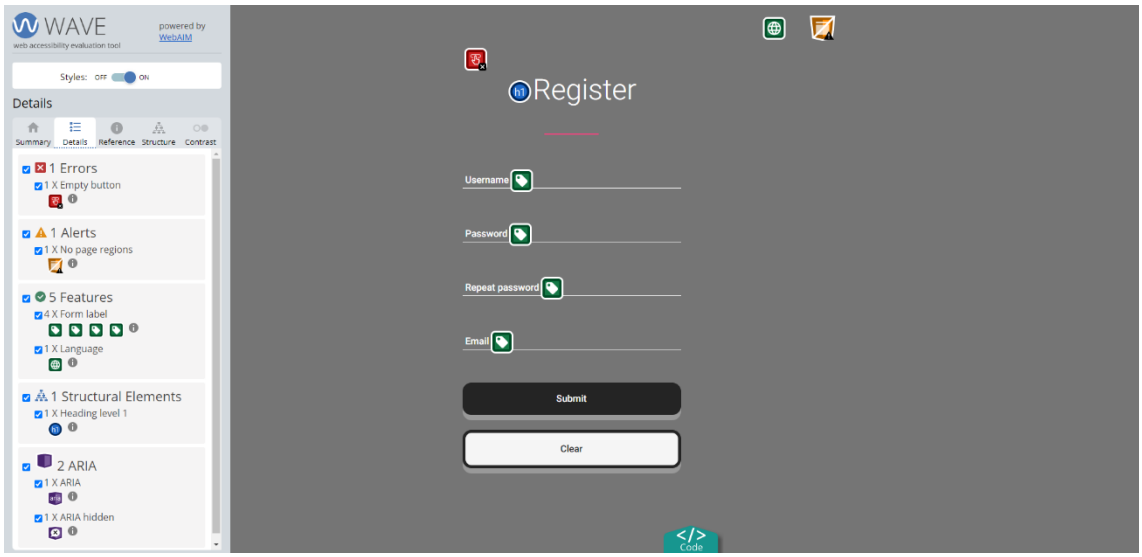


Figura 8.3. Informe de WAVE sobre la accesibilidad del formulario de registro

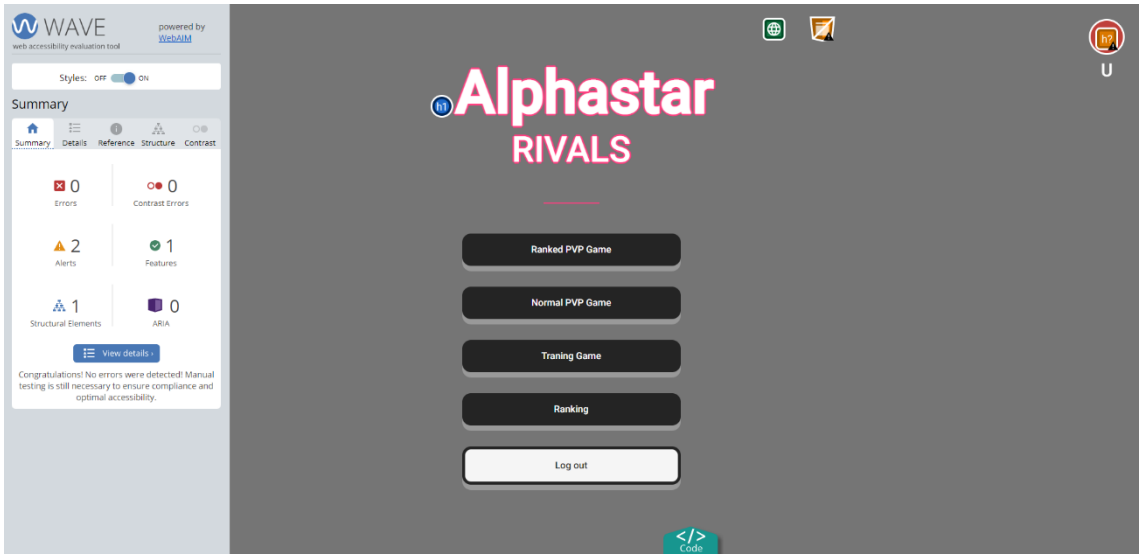


Figura 8.2. Informe de WAVE sobre la accesibilidad del menú principal para usuarios identificados

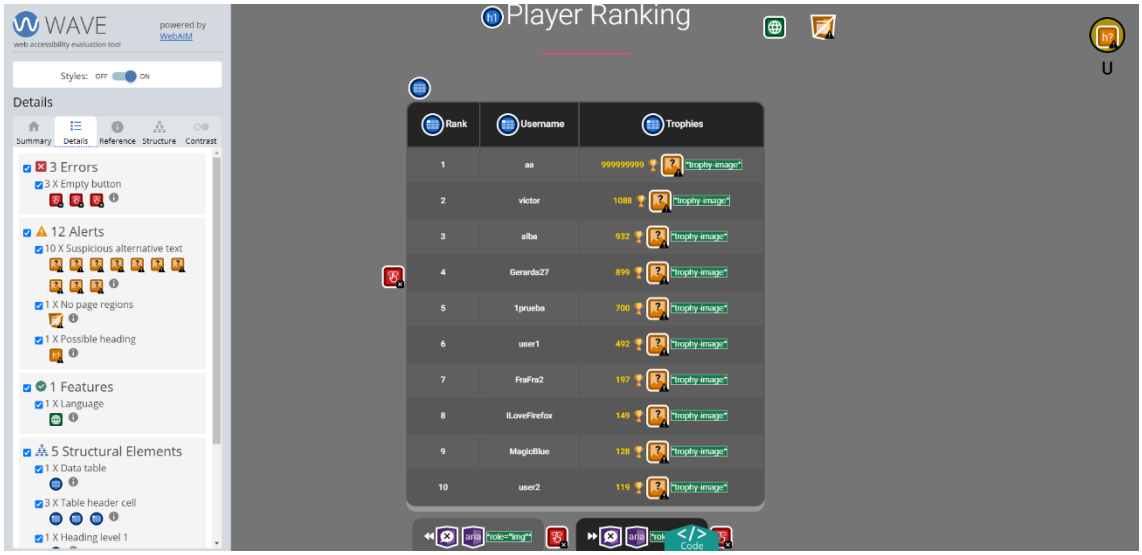


Figura 8.5. Informe de WAVE sobre la accesibilidad del menú ranking de jugadores

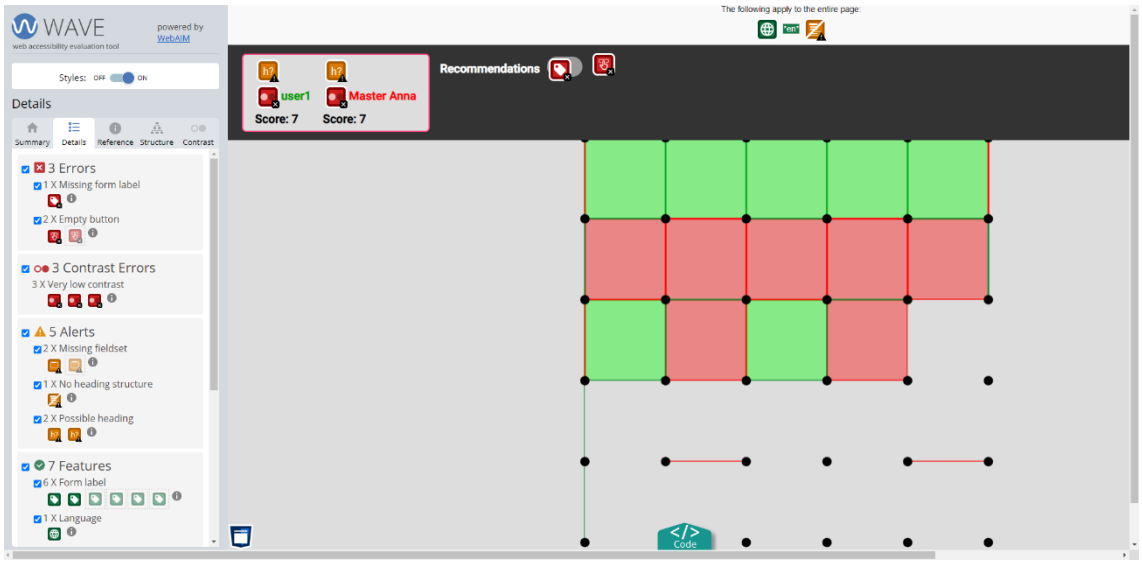


Figura 8.4. Informe de WAVE sobre la accesibilidad de la pantalla de juego con la vista de tablero

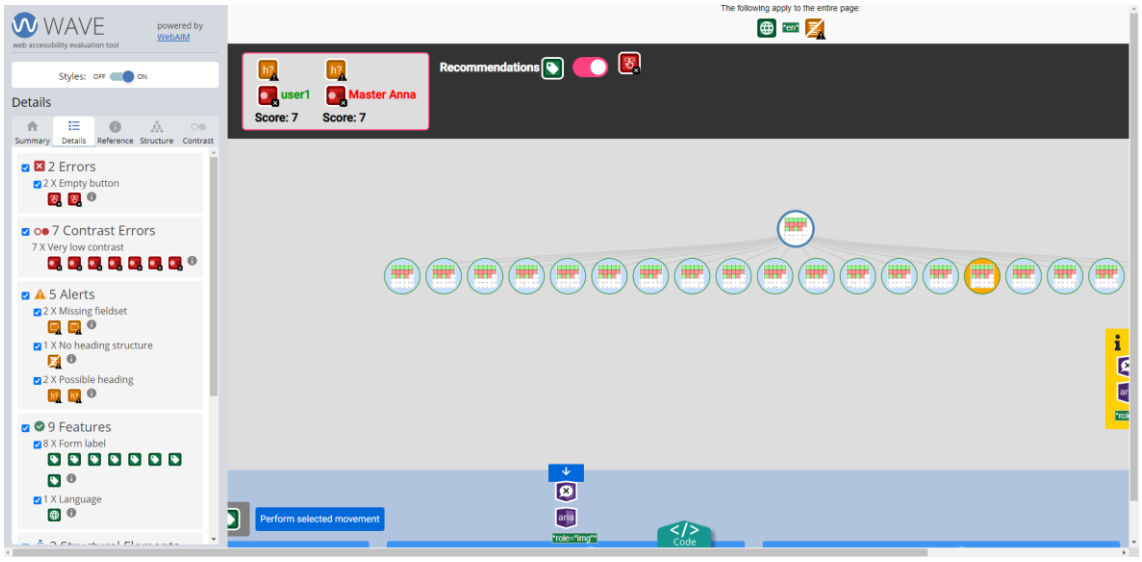


Figura 8.6. Informe de WAVE sobre la accesibilidad de la pantalla de juego con la vista de árbol

Todos los errores que aparecen en las anteriores imágenes son de tres únicos tipos:

- Elementos de formulario sin etiqueta. Este fallo estaba provocado porque las etiquetas correspondientes no tenían añadido el atributo *for* que las vincula a un elemento de tipo *input*. Para solucionarlo, simplemente se ha añadido el atributo.
- Elementos con poco contraste de color entre ellos. Este tipo de errores han sido subsanados simplemente cambiando algunos de los colores escogidos en la interfaz. Se ha logrado un nivel AA del WCAG 2.1 o superior en este aspecto para todos los textos.
- Botones vacíos. La extensión etiqueta todos los botones que contienen un icono como "vacíos". Se ha decidido dejar esto como está ya que estos botones implementan funcionalidades muy sencillas (ir atrás, salir…) que están bien representadas por los iconos correspondientes. Añadirles texto empeoraría, al menos en mi opinión, es aspecto y usabilidad de la aplicación.

8.3.2.3 Accesibilidad con Dispositivos Móviles

La aplicación no está especialmente desarrollada para dispositivos móviles ya que, para algunas de sus funcionalidades (como las consultas en el árbol interactivo), se requiere que el dispositivo del cliente cuente con suficiente capacidad de procesado. Por eso, no se realizarán pruebas extensas sobre su accesibilidad en dispositivos móviles.

A pesar de ello, se han empleado dos dispositivos distintos para probar, de forma superficial, si la aplicación se puede usar utilizando este tipo de máquinas como cliente. El resultado ha sido que todas las funcionalidades están disponibles y que la mayoría de la aplicación se ve bien. La única excepción es el árbol interactivo que, aunque ofrece todas sus funcionalidades de forma normal, ocupa un tamaño muy reducido en la pantalla, lo que lo hace prácticamente inutilizable.

Los dispositivos empleados para estas pruebas han sido un Huawei Mate 20 Lite y un Redmi Note 10 Pro. Dispositivos considerados de gama media-baja y gama media, respectivamente.

8.4 Pruebas de Rendimiento

En esta fase, no se han realizado exhaustivas pruebas de carga sobre el servidor, ya que el *hardware* que se está empleando dista enormemente de ser el idóneo para servir este tipo de aplicaciones al requerir bastante potencia de computación para ejecutar el algoritmo inteligente. De todas formas, se ha probado a jugar diez partidas de entrenamiento (las que más recursos consumen, ya que requieren que el algoritmo genere tanto los movimientos del rival como las recomendaciones) distintas de forma simultánea y no se han podido apreciar diferencias muy notables en el tiempo de respuesta del servidor.

En la siguiente tabla se compara el tiempo que tarda el sistema en generar cada movimiento del rival cuando se le solicitan diez de forma simultánea en contraposición a lo que tarda cuando se le piden de forma individual. En ella podemos ver que el tiempo máximo por movimiento ha aumentado un 26% con diez peticione simultáneas respecto a cuando le llega una sola. También es relevante el hecho de que el tiempo por cada movimiento se vuelve menos consistente, aumentando la diferencia entre unos y otros.

	Peticiones simultáneas	
	10 peticiones	**1 petición**
	Tiempo de respuesta (ms)	
M1	12889	9390
M2	13004	8858
M3	11624	8671
M4	10483	8890
M5	10532	8765
M6	12071	9187
M7	12584	8828
M8	10005	8515
M9	9820	8374
M10	8778	8678
Media	11179	8815.6
Dev. Est.	1383	283

En la Figura 8.7 se puede observar el porcentaje de uso de cada procesador lógico de la CPU. La primera mitad corresponde a un período en el que el sistema debe generar los diez movimientos del rival al mismo tiempo. La segunda mitad de cada gráfico indica el uso cuando el algoritmo está produciendo los movimientos de uno en uno.

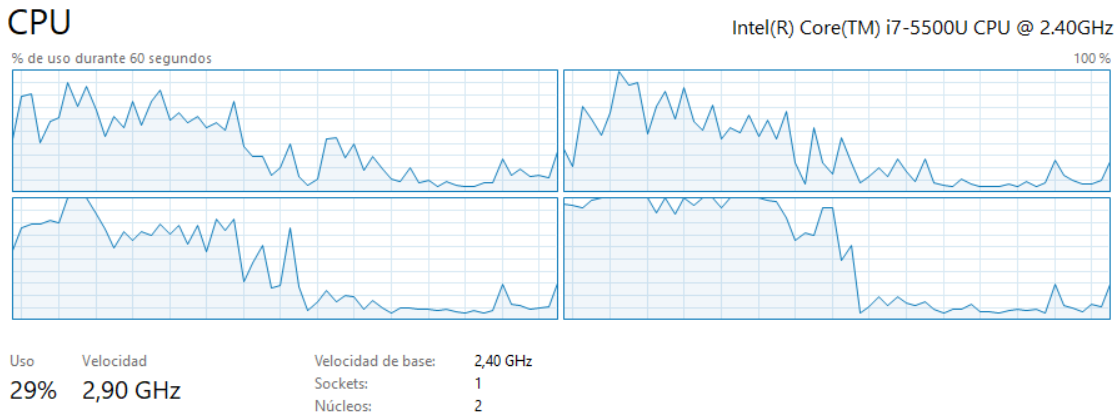


Figura 8.7. Gráfico de porcentaje de uso de la CPU

Capítulo 9. Manuales del Sistema

Este capítulo está dedicado a explicar distintos aspectos de la aplicación desarrollada. Cada explicación está enfocada a uno de los distintos perfiles que pueden tener contacto con nuestro sistema: administradores del sistema, usuarios finales y programadores.

9.1 Manual de Instalación

En esta sección se explicará cómo instalar y configurar todo el *software* necesario para poder, después, iniciar los dos procesos distintos que comprenden el sistema.

9.1.1 Instalar Java

En este caso, se mostrarán los pasos de cómo instalar el JDK 13.0.2 (Java 13) en Windows 10. El proceso es similar si se utilizan otros sistemas operativos, como puede ser Linux. También se puede instalar otra versión de Java, siempre que sea más reciente que la 13.0.2.

El primer paso será acceder, mediante navegador, al repositorio de Oracle para esta versión del JDK. Una vez allí, se encontrará una tabla (Figura 9.1) en la que deberá seleccionar el archivo de instalación que se adecúe a su máquina y sistema operativo. Para iniciar la descarga, se le pedirá que acepte los términos de uso y que cree y valide una nueva cuenta (o, si ya la tiene, que inicie sesión con ella).

Java SE Development Kit 13.0.2

This software is licensed under the Oracle Technology Network License Agreement for Oracle Java SE

Product / File Description	File Size	Download
Linux Debian Package	155.72 MB	jdk-13.0.2_linux-x64_bin.deb
Linux RPM Package	162.66 MB	jdk-13.0.2_linux-x64_bin.rpm
Linux Compressed Archive	179.41 MB	jdk-13.0.2_linux-x64_bin.tar.gz
macOS Installer	173.3 MB	jdk-13.0.2_osx-x64_bin.dmg
macOS Compressed Archive	173.7 MB	jdk-13.0.2_osx-x64_bin.tar.gz
Windows x64 Installer	159.83 MB	jdk-13.0.2_windows-x64_bin.exe
Windows x64 Compressed Archive	178.99 MB	jdk-13.0.2_windows-x64_bin.zip

Figura 9.1. Página web con los distintos archivos de instalación para Java 13

Una vez el archivo de instalación ejecutable esté descargado, puede simplemente hacer doble *click* sobre él y, tras darle los permisos para modificar el sistema, se abrirá el asistente de instalación. Ya en este asistente, como se ve en la Figura 9.2, se le pedirá que escoja la localización en el sistema de archivos donde quiere que se instale esta versión de Java (puede

dejar la localización que aparece por defecto y recuerde el lugar elegido). Y, a partir de ahí, comenzará la instalación.

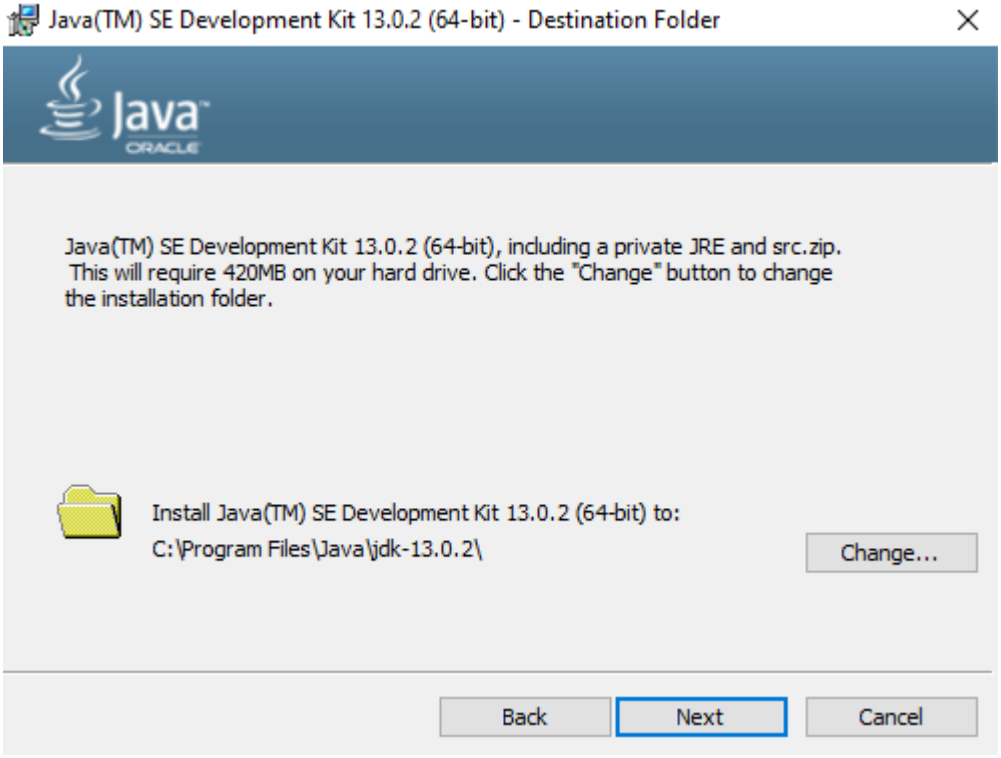


Figura 9.2. Pantalla del asistente donde puede elegir la localización de la instalación de Java

Una vez el producto esté instalado, busque en la barra de búsqueda de Windows (si está en el escritorio, abajo a la izquierda) "Este equipo" y pulse el botón derecho del ratón sobre él. En el submenú que emergerá, seleccione "Propiedades". Esto abrirá una pantalla de configuración como la que se observa en la Figura 9.3. En su parte derecha, descienda hasta encontrar la opción "Configuración avanzada del sistema" y acceda a ella.

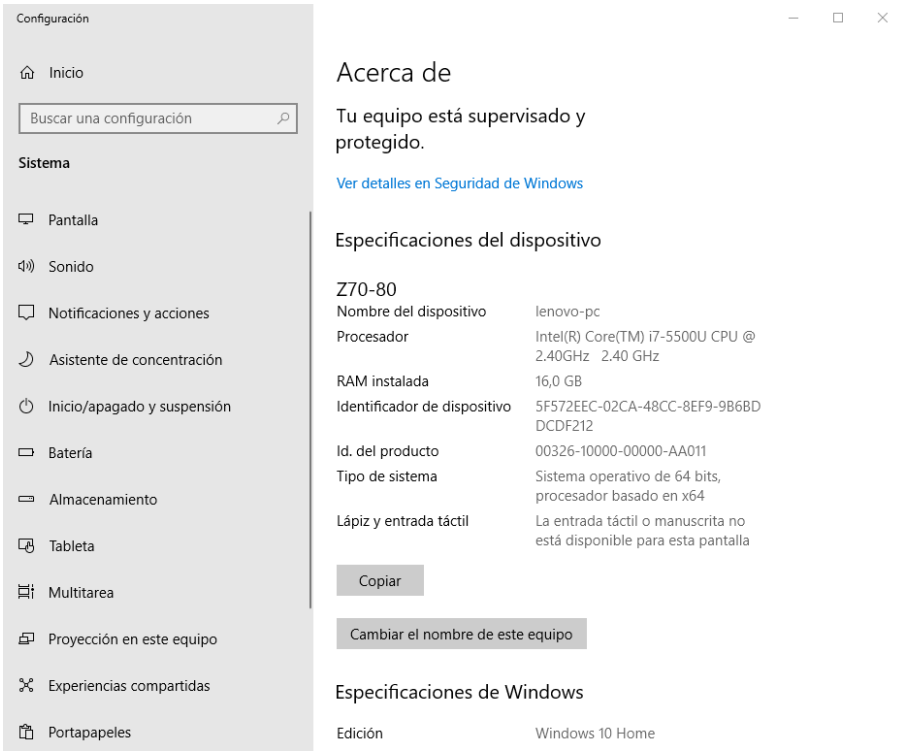


Figura 9.3. Configuración de Propiedades del equipo

Ya en la pantalla de configuración avanzada, pulse "Variables de entorno…", en la parte inferior derecha. Esto abrirá una nueva ventana de configuración similar a la de la Figura 9.4.

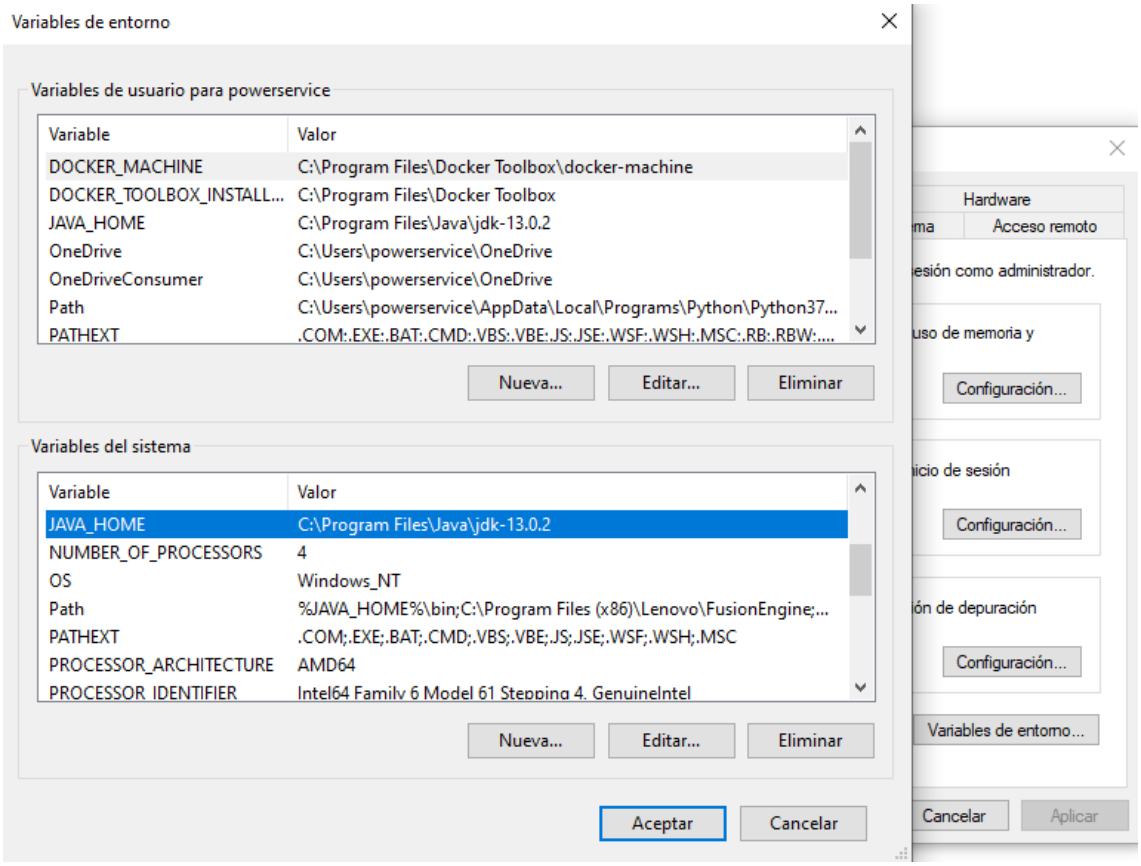


Figura 9.4. Configuración de Variables de entorno

Localice la opción "Nueva..." en la pantalla de configuración de variables de entorno, abajo a la derecha. Como se ve en la Figura 9.5, cree una nueva variable cuyo nombre sea JAVA_HOME y en "Valor" introduzca la ruta a la instalación de Java que acaba de hacer (la misma ruta que aparecía en el asistente de instalación.

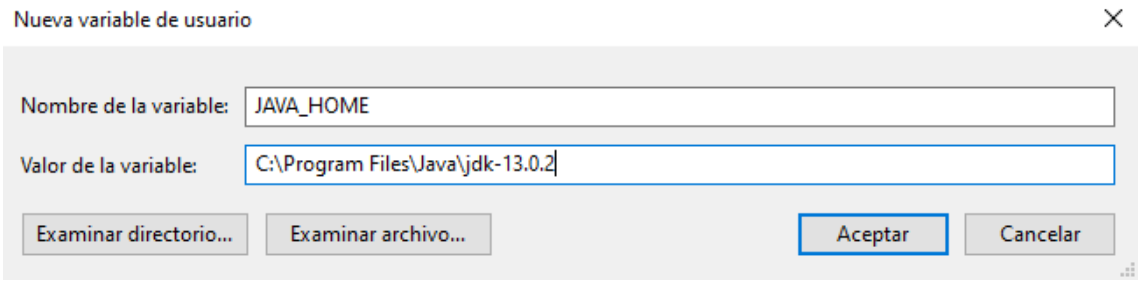


Figura 9.5. Crear nueva variable de entorno

A continuación, localice en la parte inferior de la pantalla de configuración de variables de entorno (la que puede observar en la Figura 9.4) la variable con nombre "Path". Edite esa variable (en la ventana de configuración que se ve en la Figura 9.7) y añada la una nueva línea (opción "Nuevo) con el siguiente texto: %JAVA_HOME%\bin.

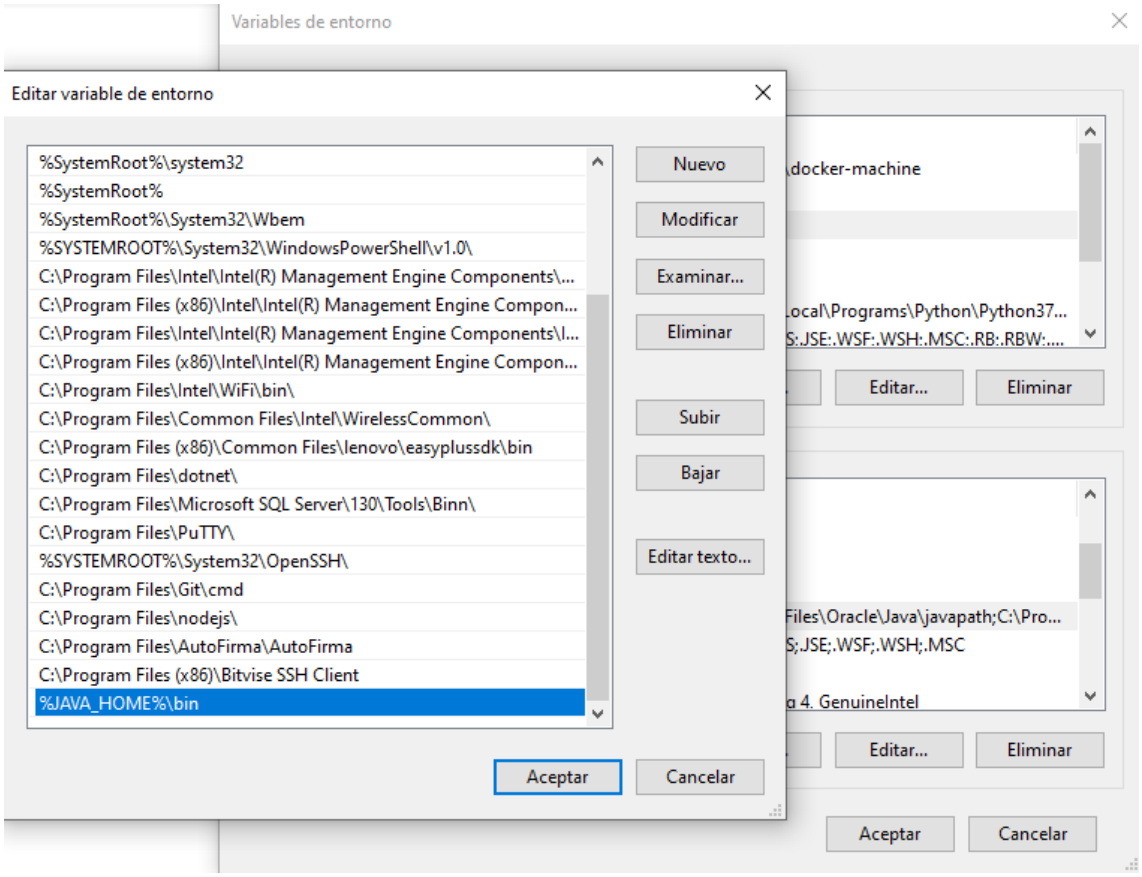


Figura 9.7. Configuración de la variable "Path"

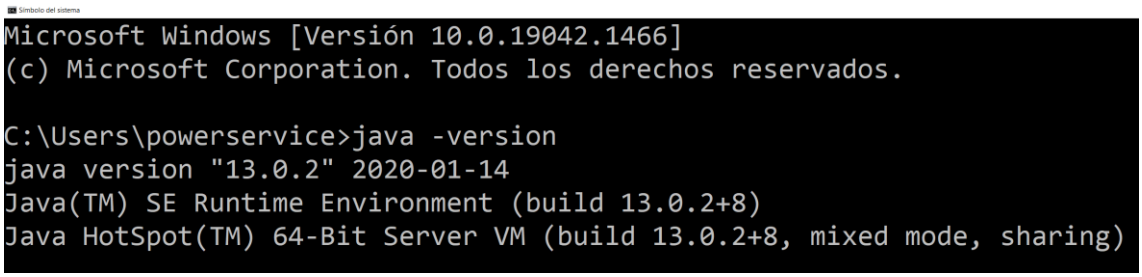


Figura 9.6. Comprobación de que Java 13 se ha instalado correctamente

Para asegurarse de que la versión de Java se ha instalado y configurado correctamente, puede abrir una nueva instancia de la consola de Windows (escriba "cmd" en la barra de búsqueda) y ejecutar el comando "`java -version`".

9.1.2 Instalar Angular

En este apartado podrán verse los pasos a seguir para instalar Angular en Windows 10. De nuevo, el proceso sería similar en Linux o macOS.

Lo primero es descargar NPM (Node Package Manager), *software* a través del cual se realizará la instalación de Angular posteriormente. Para ello, debe descargar el instalador de Node.js de su página *web* (Fig. 9.8). Una vez allí seleccione la versión (le recomendamos que sea la versión LTS, siglas en inglés de Soporte a Largo Plazo) y escoja el archivo de instalación que se adecúe al sistema operativo y arquitectura de su máquina.

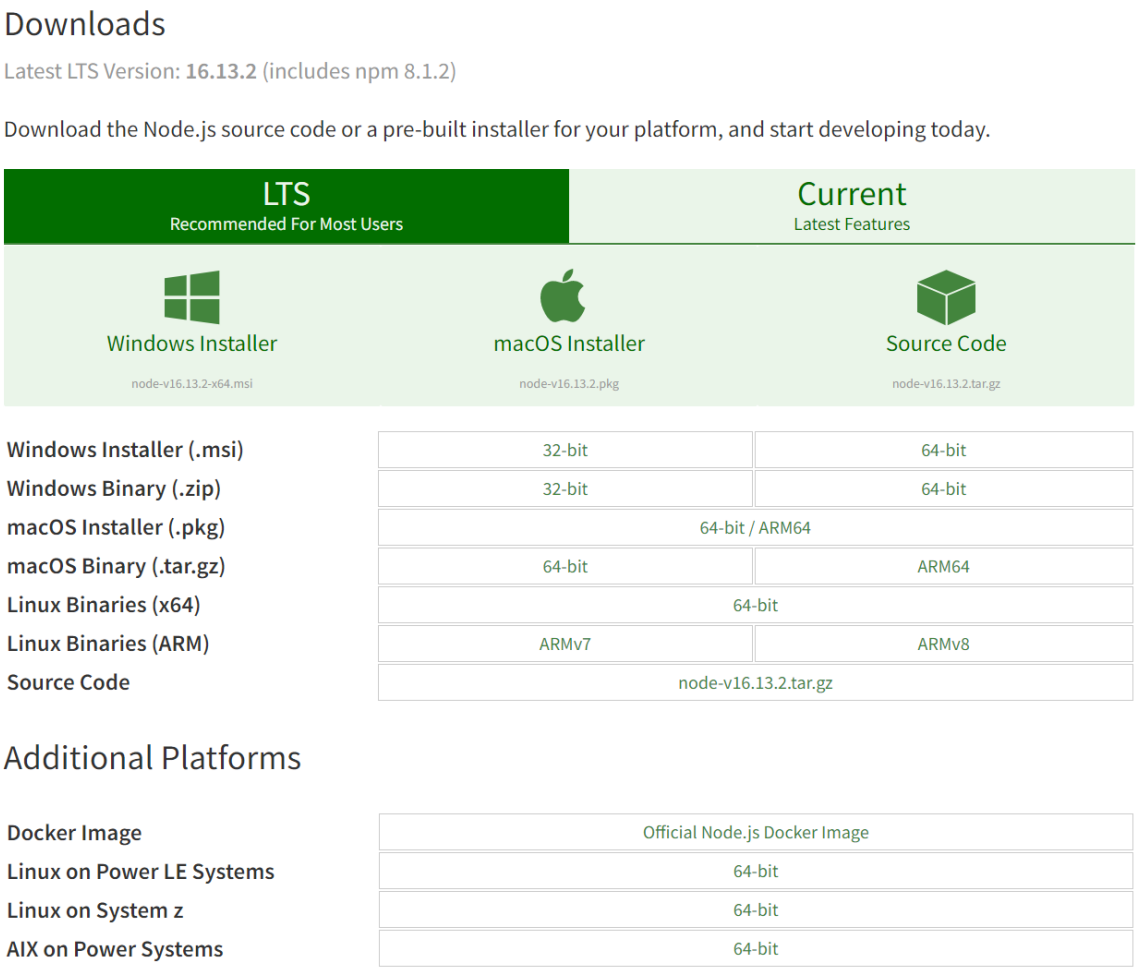


Figura 9.8. Página web con los distintos archivos de instalación para Node.js

Cuando el archivo de instalación esté descargado, ejecútelo dando doble *click* sobre él. Esto abrirá el asistente de instalación. Tras aceptar los términos y condiciones para usar el *software*, deberá elegir dónde quiere que se realice la instalación (Fig. 9.9).

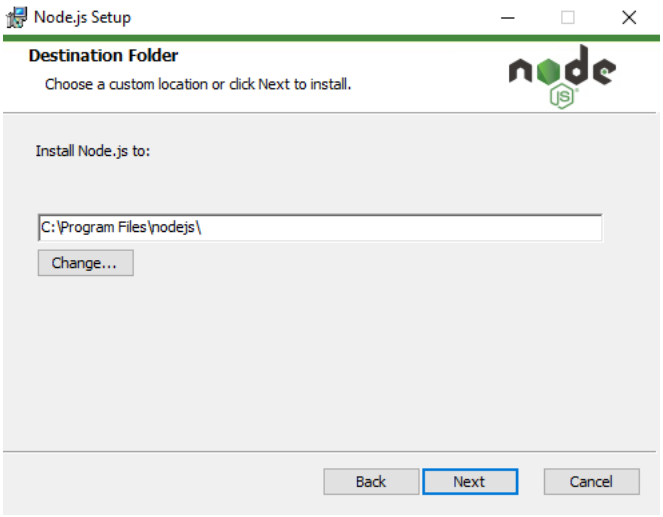


Figura 9.9. Pantalla del asistente donde puede elegir la localización de la instalación de Node.js

Después, verá una pantalla que le permitirá modificar la instalación de Node.js: si desea modificarla, asegúrese de que la opción de "npm package manager" sí será instalada, como se observe en la Figura 9.10. En la pantalla siguiente, seleccione directamente la opción "Next", ya que las funcionalidades que ahí se pueden habilitar no son obligatorias. Y, finalmente, pulse el botón "Install" (Fig. 9.11).

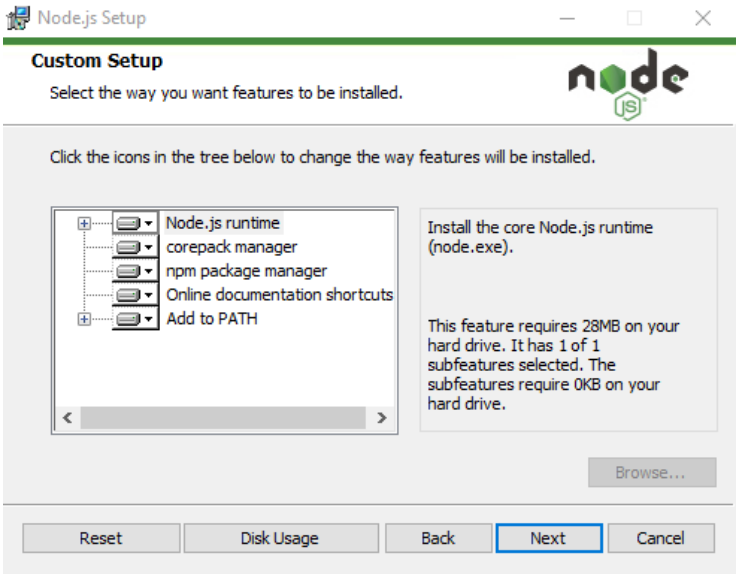


Figura 9.10. Pantalla del asistente donde puede configurar qué módulos de Node.js serán instalados

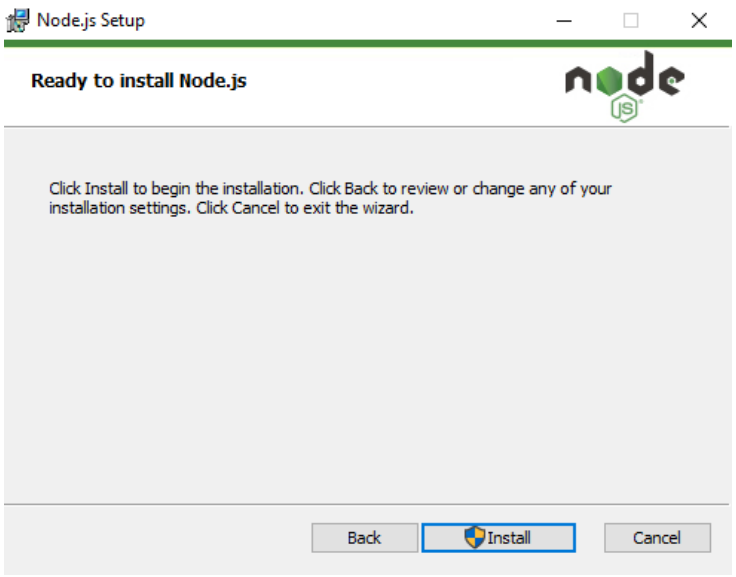


Figura 9.11. Pantalla del asistente que permite iniciar la instalación de Node.js

A continuación, se le pedirá que otorgue permisos para realizar modificaciones en el sistema y, tras aceptar, comenzará la instalación. Una vez el asistente de instalación le notifique de que la instalación ha acabado, pulse el botón "Finish". De forma similar a con la instalación de Java, puede asegurarse de que Node.js y NPM han sido instalados correctamente abriendo la consola de Windows e introduciendo los comandos "`node -v`" y "`npm -v`".

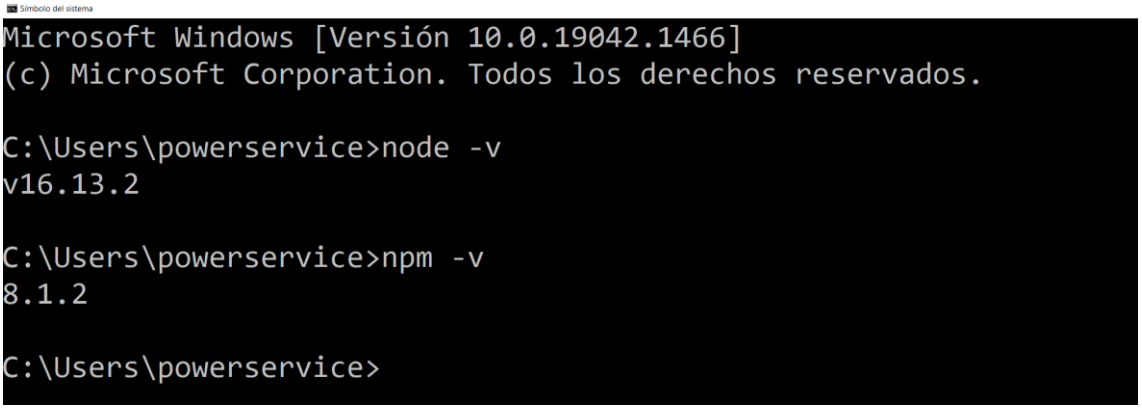


Figura 9.12. Comprobación de que Node.js y NPM se han instalado correctamente

El siguiente paso requerido será instalar Typescript, el lenguaje de programación en el que se ha desarrollado la parte de cliente de la aplicación *web*. Ahora que cuenta con NPM, esto no tiene mayor dificultad: abra la consola de Windows e introduzca lo siguiente: "`npm install -g typescript`". Opcionalmente, puede comprobar la versión instalada a través del comando "`tsc -v`".

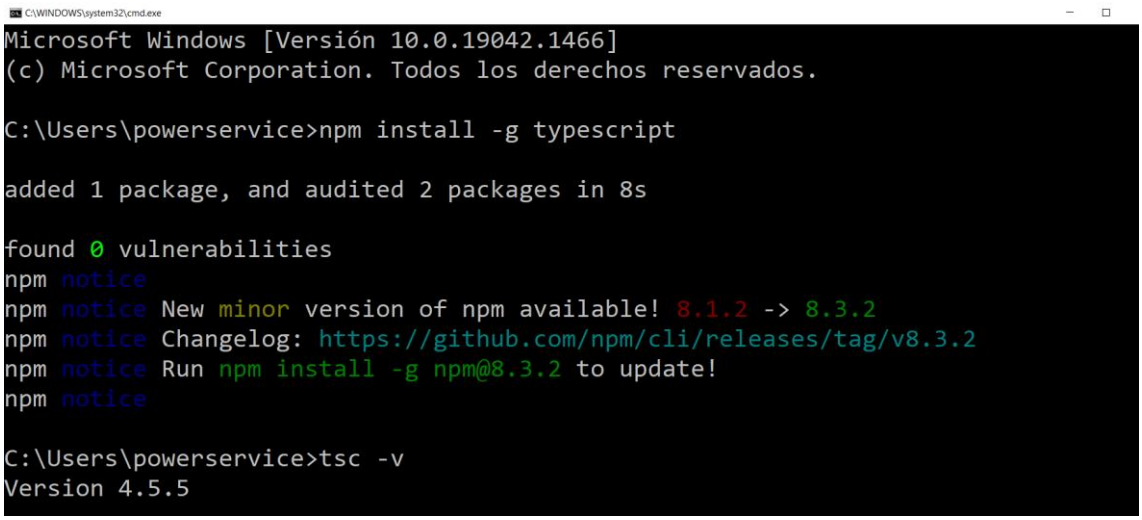


Figura 9.13. Comandos para instalar Typescript

A continuación, instale Angular. Esta operación es idéntica a la anterior: "`npm install -g @angular/cli`" para instalar y, opcionalmente, "`ng --version`" para saber qué versión se ha instalado.

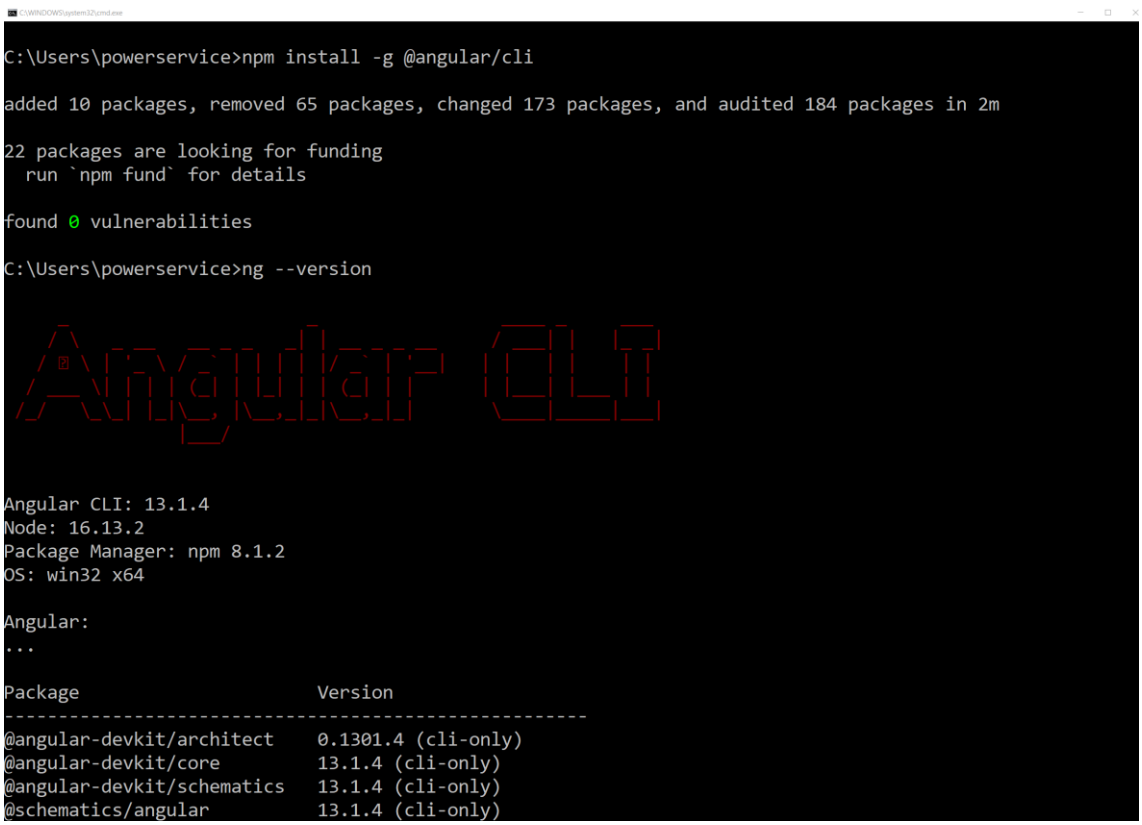


Figura 9.14. Comandos para instalar Angular

Como último paso de la instalación de la parte desarrollada en Angular, abra el explorador de archivos de Windows y sitúese en el directorio raíz de la carpeta donde tiene el proyecto de

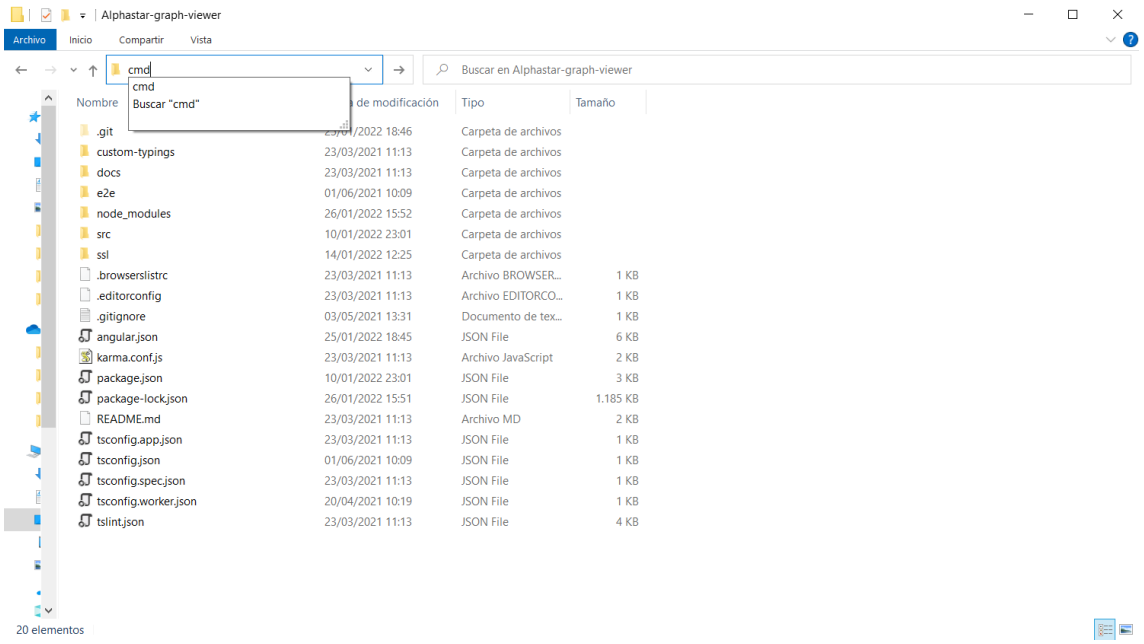


Figura 9.15. Carpeta que contiene el proyecto de Angular

Angular (Fig. 9.15). Una vez allí, pulse en la barra de búsqueda superior izquierda e introduzca "cmd". Esto abrirá una terminal, en ella deberá ejecutar el comando "`npm install`" para instalar todas las dependencias que requiere el proyecto.

Ahora deberá explicitar en el proyecto Angular cuál es la dirección del servidor de *backend*. Para ello, acceda al directorio raíz de este proyecto (el que se ve en la Figura 9.14) y lo calice el archivo *globalVariableConfig.ts*. Debe asignar como valor de la variable *alphastarServerIP* la dirección IP o el dominio del servidor de *backend* sin incluir el protocolo de comunicación. Por ejemplo: "192.168.1.39", "localhost" o www.midominio.com.

9.1.3 Configurar MongoDB

Para poder servir la aplicación de forma efectiva desde una máquina, es necesario que esta pueda conectarse con la base de datos MongoDB. Para ello, se debe registrar la IP que usa la máquina en la lista de direcciones que se pueden comunicar con la base de datos del sistema. Esta lista está restringida por motivos de seguridad: de esta manera, únicamente se puede modificar el contenido de la BD desde el servidor de la aplicación.

A la hora de probar la aplicación, no será necesario realizar este paso. Por comodidad, cuando entregue la documentación, código y ejecutables, configuraré la base de datos para que admita conexiones desde cualquier dirección IP. Esto habría que revertirlo a la hora de desplegar el sistema ya que, si alguien consiguiese las credenciales requeridas en cada petición a MongoDB, podría modificar la base de datos desde cualquier conexión. De todas formas,

esto no debería suceder, ya que estas credenciales solo están en el servidor y las comunicaciones con la BD se hacen a través de HTTPS.

Para modificar la lista mencionada, abra en un navegador *web* la página de MongoDB. Una vez allí, debe iniciar sesión (botón "Sign In") con las credenciales de la cuenta de administrador registrada en MongoDB. Tras eso, le aparecerá un panel de configuración con varias opciones presentes en el lateral izquierdo (Fig. 9.16). Seleccione la opción "Network Access" y pulse, a la derecha de la página, el botón "ADD IP ADDRESS". En el cuadro de diálogo que emergerá,

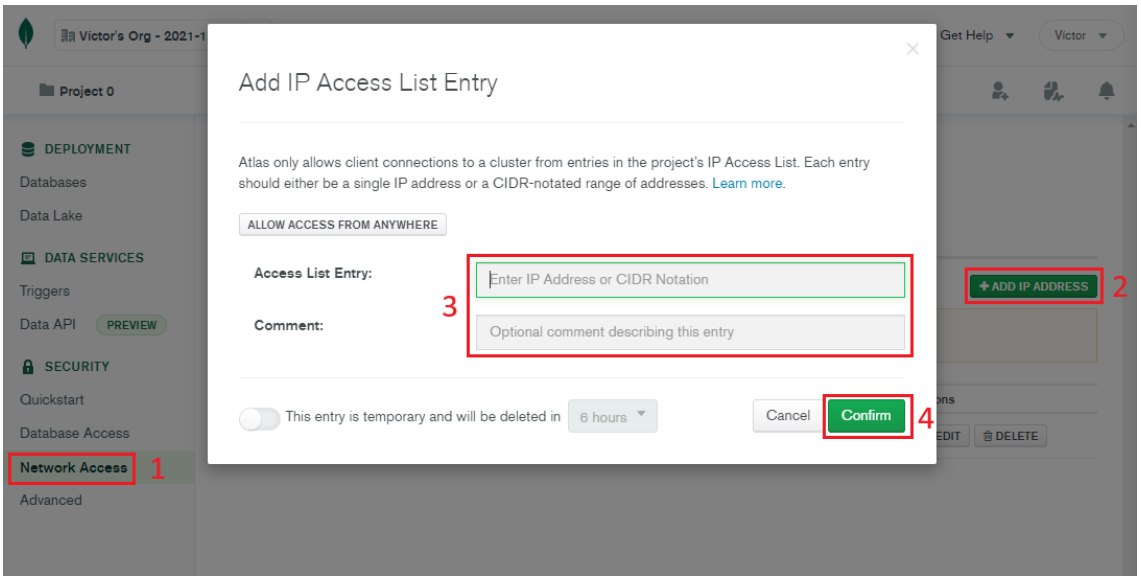


Figura 9.16. Secuencia de pasos para autorizar una nueva IP en MongoDB

introduzca la IP que desea autorizar junto con una breve descripción. Confirme los datos ingresados y habrá acabado la configuración necesaria. Puede ver la secuencia de pasos en la Figura 9.15.

9.2 Manual de Ejecución

Una vez completadas las instalaciones y configuraciones contempladas en la Sección 9.1, el despliegue del sistema no tiene dificultad alguna.

El primer paso será acceder mediante el explorador de archivos a la carpeta donde se encuentra el archivo ejecutable del servidor (alphastarServer.jar). Una vez allí, abra una nueva terminal ("cmd" en la barra de búsqueda superior izquierda del explorador de archivos) y ejecute el siguiente comando:

```
java -Djdk.tls.client.protocols=TLSv1.2 -jar alphastarServer.jar
```

Una vez lea en la terminal "Application deployed", el servidor estará corriendo correctamente. Para el proceso que sirve la interfaz de cliente, lo que debe hacer es algo similar. Localice el directorio raíz de la carpeta que contiene el proyecto de Angular y abra otra terminal. En ella, introduzca lo siguiente:

```
ng serve --host 0.0.0.0 [--port 80]
```

El texto que aparece entre corchetes es opcional y sirve para modificar el puerto en el que se servirá la aplicación *web* (el que se usa por defecto es el 4200). Cuando en la terminal aparezca "Compiled successfully", la aplicación ya estará siendo servida en la IP por defecto y el puerto escogido.

Tras estas acciones, la aplicación ya estará funcionando y podrá acceder a ella a través de la dirección en la que se está sirviendo el cliente.

Para parar el sistema deberá detener cada uno de los procesos por separado. La forma más simple es dirigirse a cada una de las terminales y pulsar la combinación de teclas **Ctrl + C**. Además, deberá confirmar la detención del proceso introduciendo una "S" en la terminal y pulsando la tecla **Enter**.

9.3 Manual de Usuario

Aquí se detallarán las formas en las que los usuarios pueden usar la aplicación, y las distintas herramientas y funcionalidades que tienen a su alcance. Estas explicaciones estarán acompañadas de diversas capturas para facilitar su comprensión.

9.3.1 Acceso

Para acceder a la aplicación, simplemente es necesario tener un navegador con interfaz gráfica y JavaScript activado, y escribir en la barra de búsquedas la dirección (URL o IP) en la que se esté sirviendo. Una vez aquí, el usuario puede identificarse de dos maneras: empleando credenciales de una cuenta ya existente (*Log in*) o registrándose (*Register)*. Cada una de estas opciones le llevará a un formulario distinto donde podrá introducir sus datos y, si estos superan las distintas validaciones, se iniciará su sesión en la aplicación. De lo contrario, se le mostrará un mensaje advirtiéndole del fallo y comunicándole la causa concreta de este (credenciales que no existen en el sistema, registro con nombre de usuario ya existente, email inválido, etc.). Además, una vez cuente con una sesión iniciada, no podrá acceder de nuevo con las mismas credenciales hasta que la cierre (por ejemplo, con el botón de "Log out" o cerrando la pestaña del navegador).

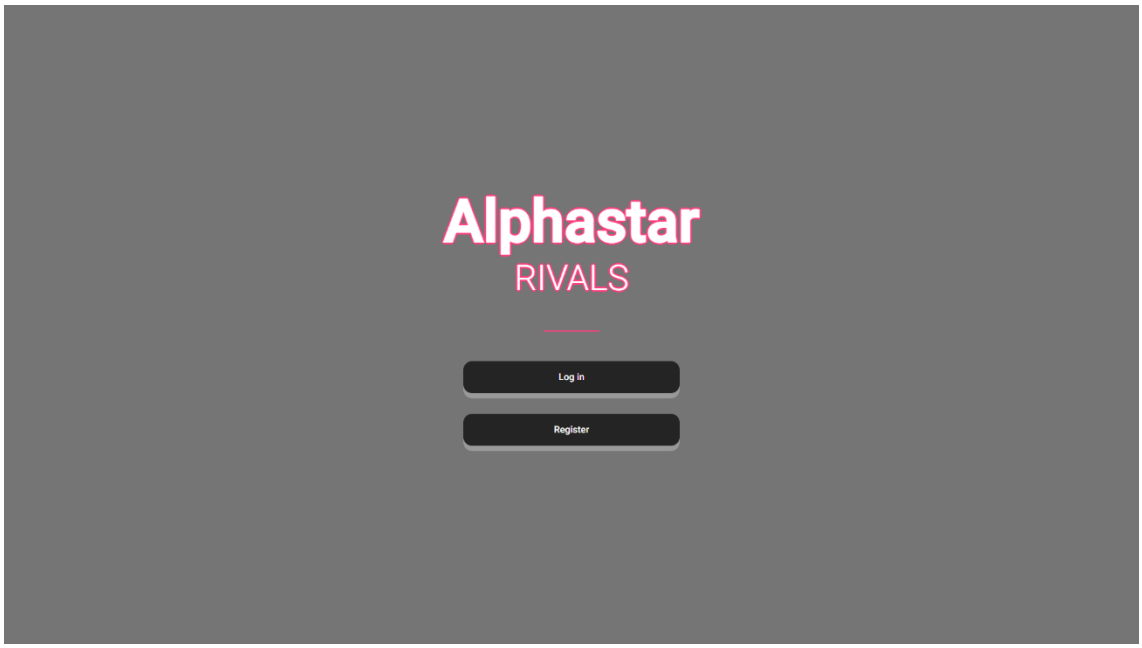


Figura 9.17. Menú de acceso a la aplicación

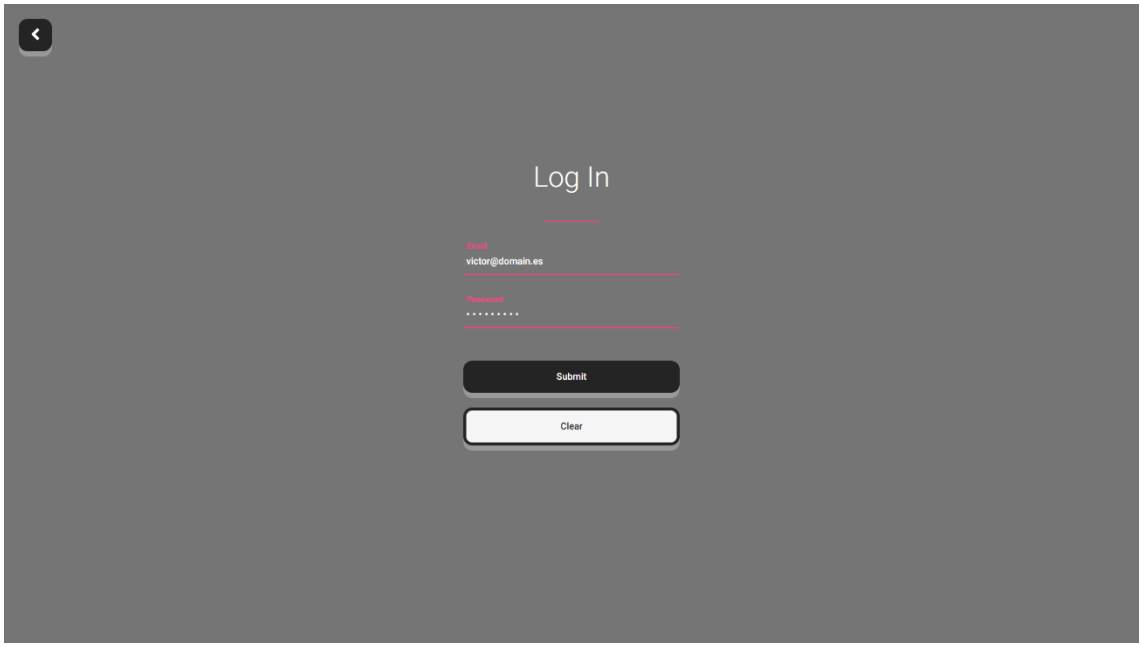


Figura 9.19. Formulario de inicio de sesión con credenciales

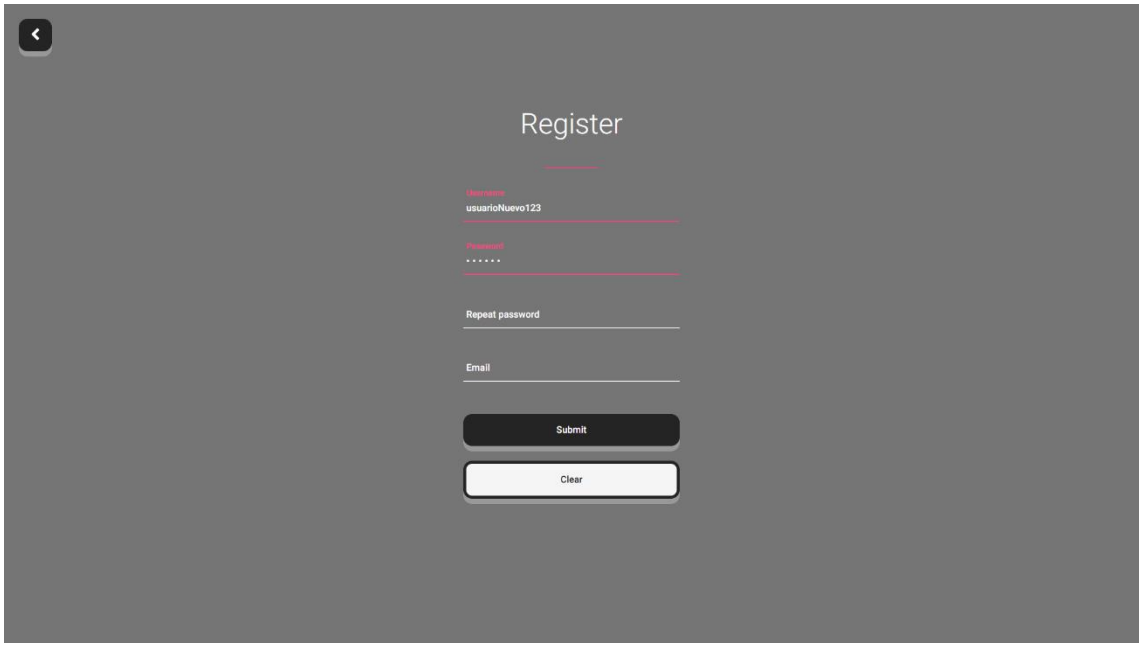


Figura 9.18. Formulario de registro

9.3.2 Menú Principal

Una vez haya entrado en la aplicación mediante cualquiera de las opciones ya mencionadas, se le mostrará un menú con las distintas opciones que tiene disponibles (Fig. 9.20), de arriba a abajo:

- **Partida multijugador clasificatoria**. Esta opción le llevará a un submenú (Fig. 9.22) donde podrá seleccionar qué tipo de partida multijugador clasificatoria quiere jugar: tiempo limitado por partida y jugador; tiempo limitado por movimiento, o sin limitación de tiempo (no recomendado, ya que el jugador rival podría hacerle esperar indefinidamente). Tras seleccionar una de estas opciones, el sistema lo emparejará con otro jugador humano que busque una partida de iguales características y podrán empezar el juego. Cuando estas partidas acaban, la aplicación realiza una repartición de puntos (los puntos del ganador suben y los del perdedor bajan) en función de la diferencia de puntuaciones que exista entre ambos jugadores.
- **Partida multijugador normal**. Esta opción es similar a la anterior y tiene los mismos subtipos. La diferencia principal radica en que las partidas de este tipo no afectan de ninguna forma a las puntuaciones de ninguno de los dos jugadores.
- **Partida de entrenamiento**. Si pulsa este botón, accederá a un nuevo menú (Fig. 9.21) en el que podrá configurar una partida de entrenamiento contra una inteligencia artificial. Cuando haya escogido la configuración que le parezca adecuada, podrá empezar la partida.
- ***Ranking* de jugadores**. Esto le permite ver una lista (Fig. 9.23) de todos los jugadores ordenados por puntuación, en orden descendente (aquel con más puntuación aparece en la parte de arriba de la tabla). Esta lista se le presentará dividida en páginas, con diez entradas cada una, y podrá moverse libremente entre ellas.
- **Cerrar sesión**. Mediante esta función podrá cancelar la sesión que tiene iniciada e irá, de nuevo, al menú de acceso (Fig. 9.17), donde podrá volver a identificarse o registrar una nueva cuenta.

Además de estas opciones, en la esquina superior derecha de este menú principal y todos sus submenús, tendrá disponible un botón redondo con la primera letra de su nombre de usuario en el centro. Si lo pulsa, aparecerá un cuadro de diálogo mostrándole la información de la cuenta con la que ha accedido: nombre, *email* y puntuación (Fig. 9.22). Para cerrarlo, simplemente haga *click* en cualquier punto de este cuadro de diálogo.

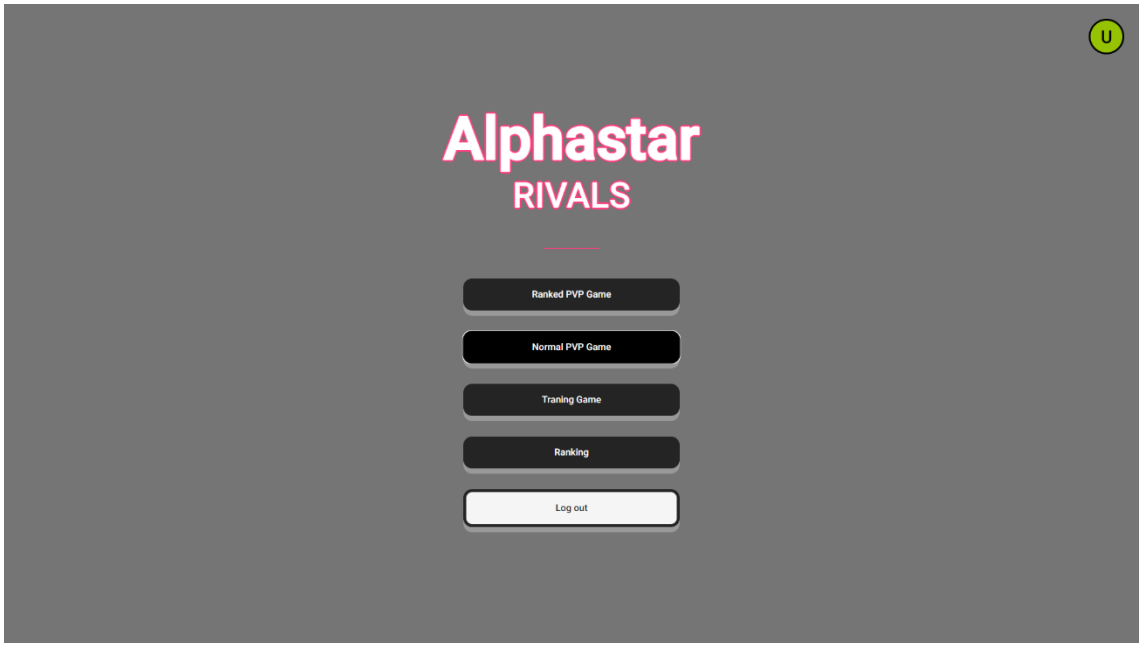


Figura 9.20. Menú principal de la aplicación

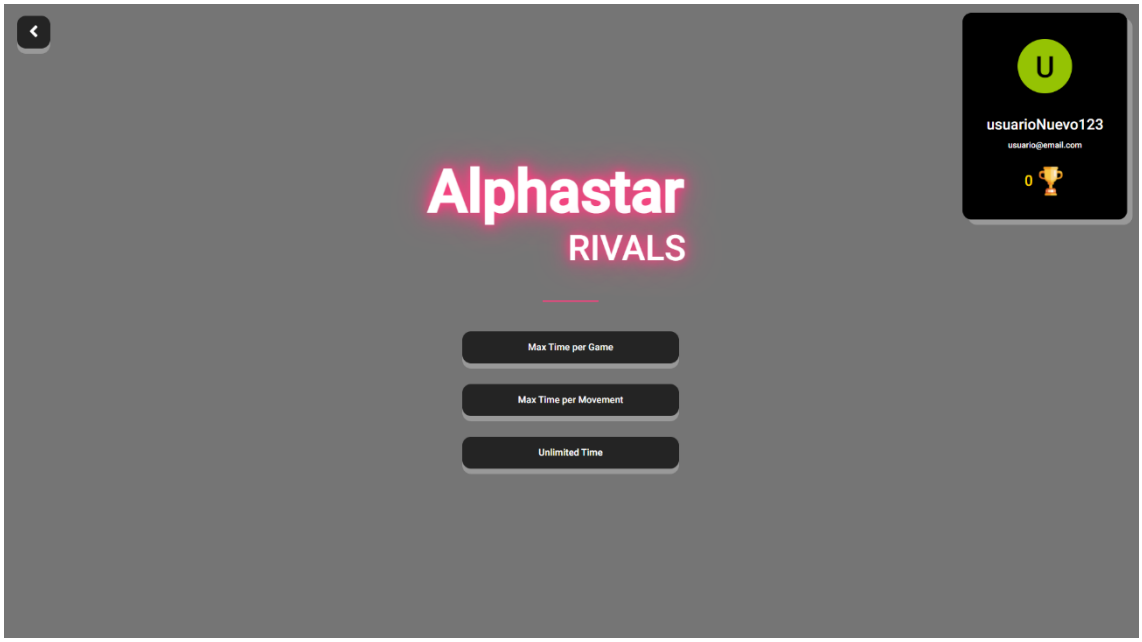


Figura 9.22. Submenú para escoger tipo de partida multijugador

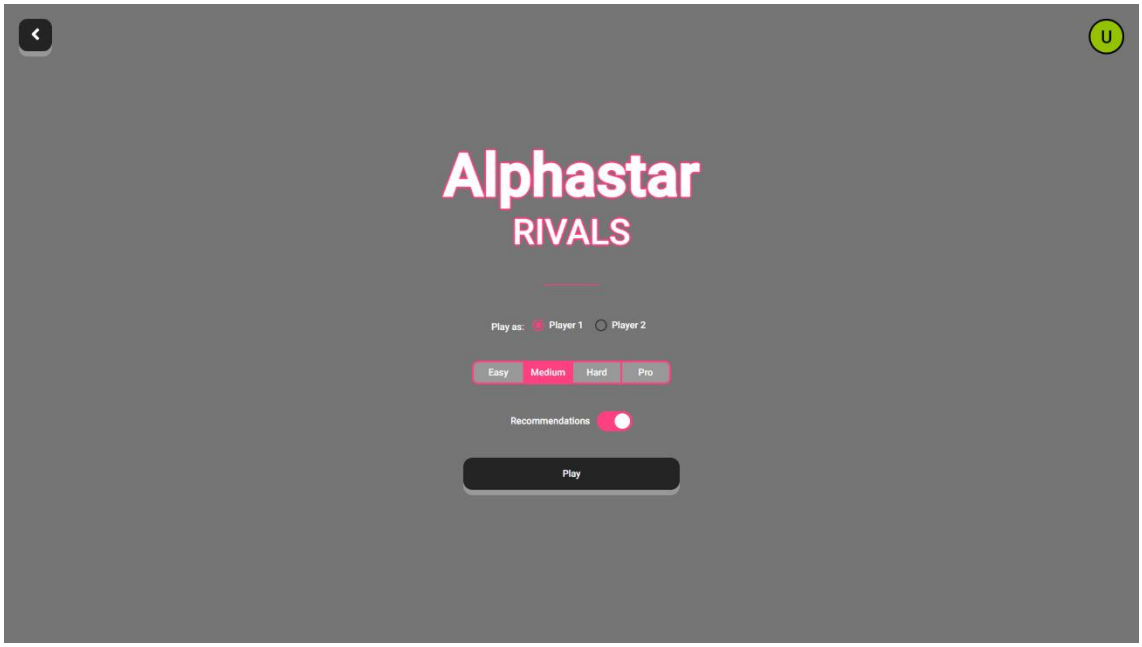


Figura 9.21. Submenú para configurar una partida de entrenamiento

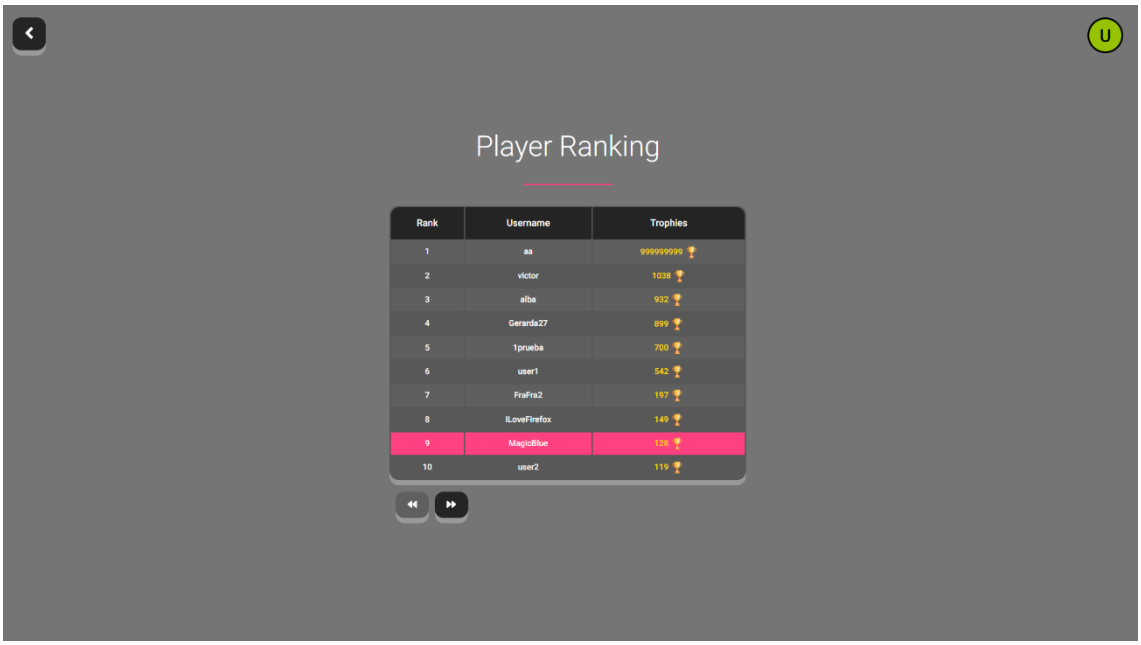


Figura 9.23. Ranking de jugadores

9.3.3 Partidas Multijugador

Cuando escoja el tipo de partida que quiere según su limitación de tiempo (a través del menú que se observa en la Figura 9.22), da igual que sea una partida normal o una clasificatoria, le aparecerá una pantalla de carga con una animación circular hasta que se encuentre un rival apropiado (Fig. 9.24). Mientras se encuentre en esa pantalla de búsqueda de rival, puede abandonar mediante la cruz de la esquina superior derecha o cerrando la ventana del navegador y no será penalizado de ninguna forma, ni se le restarán puntos en caso de que haya buscado una partida clasificatoria.

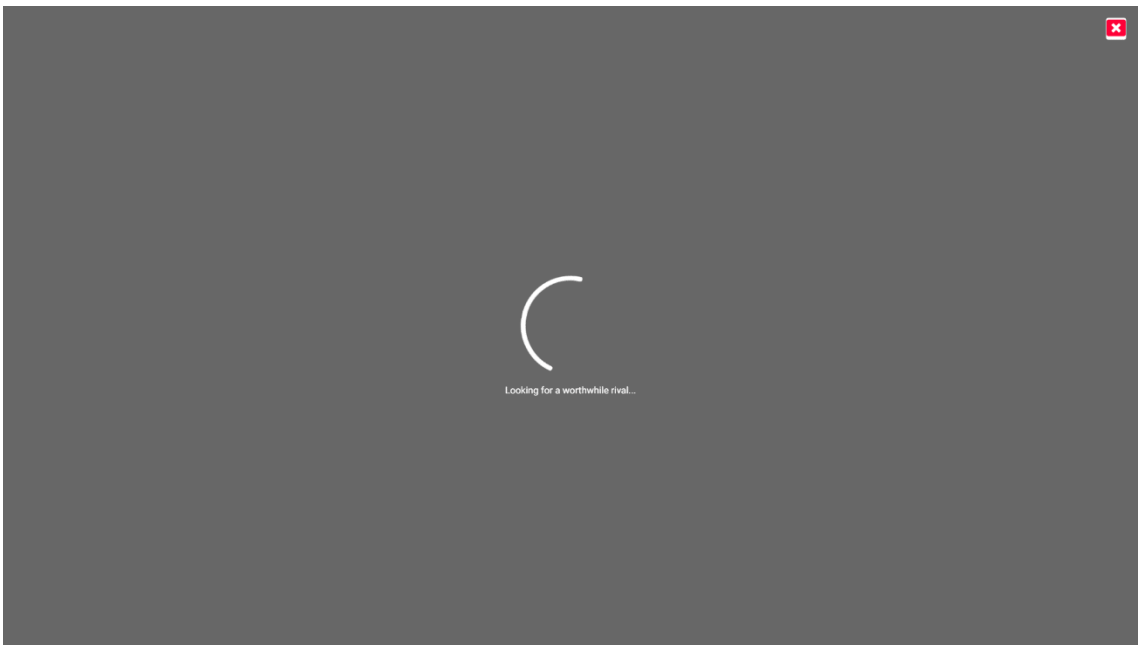


Figura 9.25. Pantalla de búsqueda de rival

Figura 9.24. Pantalla de juego

Una vez de haya encontrado a otro jugador que esté buscando el mismo tipo de partida, el juego empezará. A partir de aquí, ambos jugadores irán dibujando líneas alternadamente (excepto si alguien consigue cerrar una caja, en cuyo caso le toca volver a dibujar). Realizará el primer movimiento aquel jugador al que se le haya asignado el color verde. La partida se jugará desde la pantalla que aparece en la Figura 9.25. Como se puede observar, la sección principal de esta está ocupada por el tablero interactivo sobre el que los jugadores realizarán sus movimientos. En la parte superior aparecerán los nombres de ambos jugadores, del color que se les haya sido asignado, junto con su puntuación. Al lado también aparecerá una cuenta

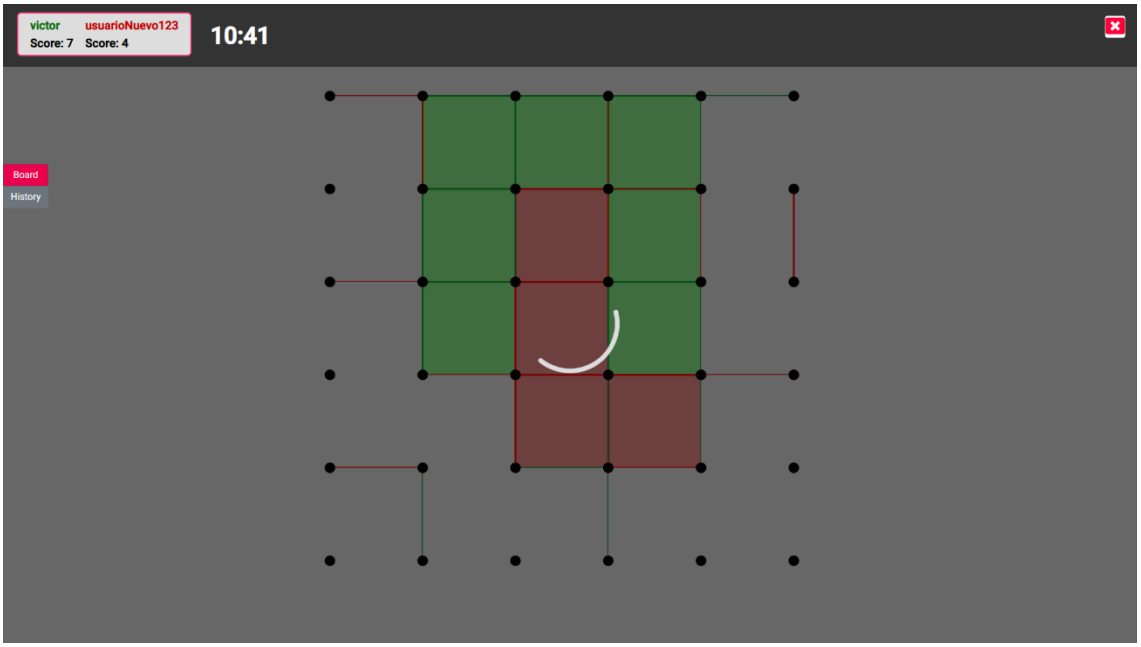


Figura 9.26. Pantalla de juego durante los turnos del rival

atrás que muestra el tiempo restante del que dispone el jugador que, evidentemente, solo se verá reducido durante su turno. Cuando le toque mover al rival, aparecerá en la pantalla una animación de espera como la que se puede ver en la Figura 9.26.

Además, en cualquier momento de la partida (independientemente de que sea su turno o el del rival, o de que la partida haya finalizado), ambos jugadores tendrán acceso a un historial de movimientos (Fig. 9.27) al que podrán acceder a través del menú lateral de la izquierda. En este historial aparecerán todos los movimientos ordenados, desde el estado inicial de la partida, hasta el momento presente, del color del jugador que los ha realizado. Este historial

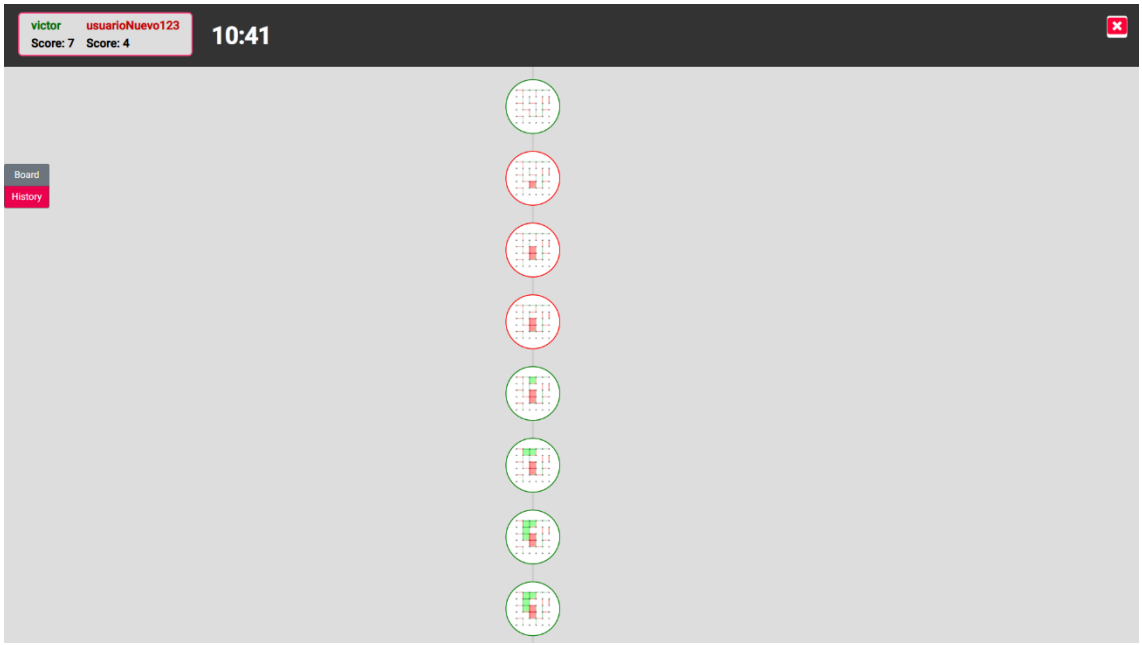


Figura 9.27. Historial de movimientos de una partida

también implementa un *zoom* que permite tanto tener una vista general del trascurso de la partida como fijar el foco en un movimiento (o grupo de movimientos) concreto.

Una vez la partida haya finalizado, se mostrará una pantalla de victoria (Fig. 9.29) y otra de derrota (Fig 9.28) al vencedor y al perdedor, respectivamente. En caso de que la partida jugada

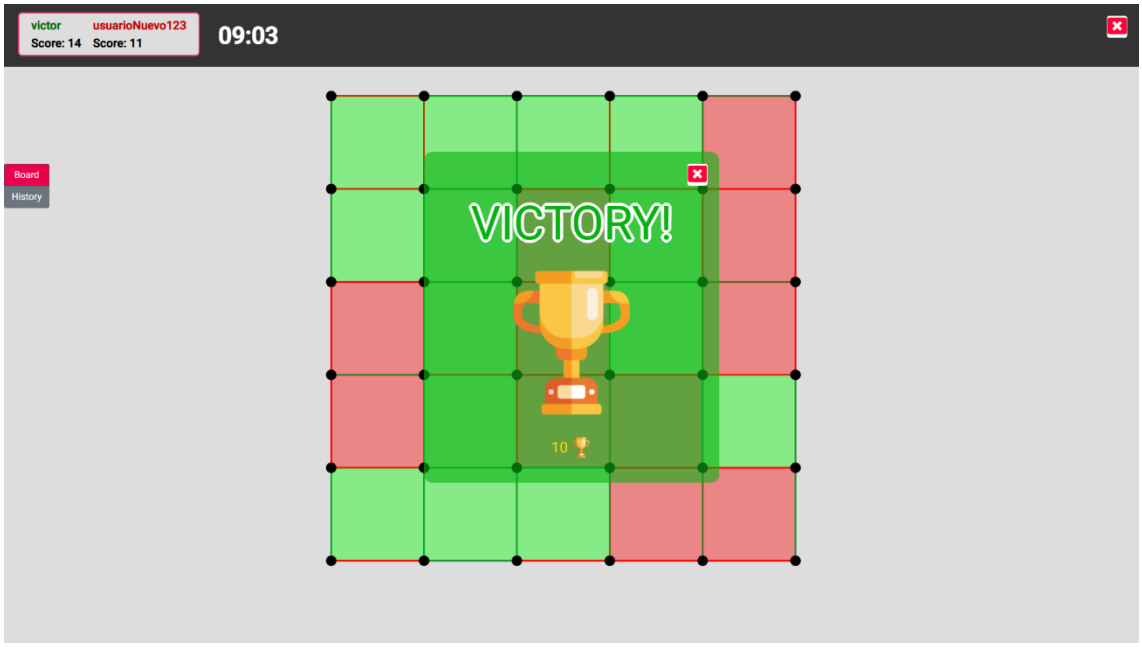


Figura 9.29. Cuadro de diálogo de victoria

haya sido una partida de entrenamiento o una multijugador normal, aparecerán los mismos cuadros de diálogo, pero se obviará el texto donde se informa de los puntos ganados o perdidos.

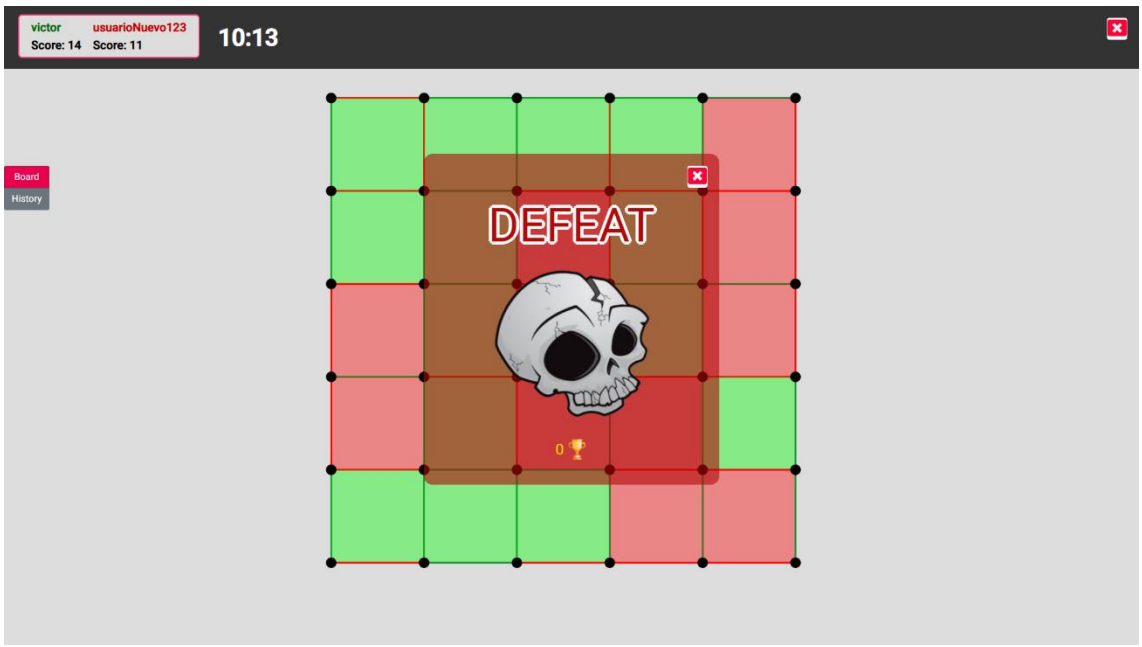


Figura 9.28. Cuadro de diálogo de derrota

Como se puede observar en las Figuras 9.29 y 9.28, al ganador solo se le han sumado 10 puntos, mientras que el perdedor no ha perdido ninguno. Esto es debido a que los puntos al final de una partida clasificatoria se calculan en base a la diferencia que exista entre los jugadores: cuanta más puntuación tenga el ganador sobre el perdedor, menos puntos ganará, y viceversa (siempre entre un mínimo de 10 y un máximo de 50). En las imágenes 9.22 y 9.23 se puede observar que las puntuaciones del ganador y del perdedor, son respectivamente, 1038 y 0 puntos, lo que explica que al ganador se le haya sumado el mínimo posible por partida. Además, en una situación normal, al perdedor se le resta lo mismo que se le suma al ganador. Esto no sucede cuando el perdedor tiene una puntuación inferior a 100, en cuyo caso no se le quita ningún punto, como en la situación mostrada. Esto es algo que implementan, de una u otra forma, todos los juegos con mecanismos de clasificación basados en puntos y sirve para ir aumentando el número de puntos totales que existen en el sistema.

La partida se considerará finalizada si sucede uno de los siguientes supuestos:

- No hay más movimientos disponibles. Cuando el tablero se completa y no quedan líneas por dibujar, habrá ganado aquel jugador que haya conseguido cerrar más cajas durante la partida.
- A un jugador se le agota el tiempo. Si la partida cuenta con limitación de tiempo, cuando este llegue a cero para alguno de los jugadores, la aplicación será capaz de detectarlo. Evidentemente, aquel jugador que haya agotado su tiempo será el perdedor.
- Un jugador abandona la partida. Ya sea mediante el botón rojo de la esquina superior derecha que permite volver al menú principal o cerrando la pestaña del navegador, el sistema lo sabrá, otorgando la victoria al rival del jugador que haya abandonado.
- Uno de los usuarios intenta hacer trampas. Si alguno intentase enviar al servidor un movimiento inválido o un movimiento del rival, el servidor se daría cuenta y asignaría la derrota al tramposo.

Una vez la partida haya acabado, aparecerán los cuadros de diálogo de derrota y victoria ya explicados. El jugador podrá seguir examinando la información de la partida hasta que decida salir al menú principal (botón rojo con una cruz) o cerrar la pestaña del navegador.

9.3.4 Partidas de Entrenamiento

En este tipo de partidas, un jugador humano se batirá contra la IA del sistema actuando como rival virtual. Antes de iniciar una partida de este tipo, el jugador puede configurar varios parámetros (Fig. 9.21):

- Si quiere jugar como *Player 1* o como *Player 2*. Si escoge la primera opción, el jugador humano realizará el primer movimiento. De lo contrario, empezará su rival.
- El nivel de dificultad del rival virtual. Aquí el jugador puede elegir entre cuatro niveles distintos: *Easy*, *Medium*, *Hard* y *Pro* (fácil, medio, difícil y profesional).
- *Recommendations*. Este "interruptor" sirve para que el jugador pueda indicar a la aplicación si desea recibir recomendaciones e información producida por la IA, o no.

En caso de que más adelante desee cambiar esta elección, podrá hacerlo mientras esté jugando la partida. El contenido de estas recomendaciones se explicará a continuación.

Una vez haya configurado su partida, podrá iniciarla mediante el botón "Play". Tras esto, verá una pantalla de juego (Fig. 30) muy similar a la que vería en una partida multijugador. Las principales diferencias son que, en el cuadro de información superior, aparecerá el interruptor que permite activar o desactivar la generación de recomendaciones (que permite, siempre en su turno, decidir si quiere recibir estas ayudas para los movimientos siguientes). Además, si las recomendaciones están activadas, en el menú de la izquierda dispondrá de una nueva opción:

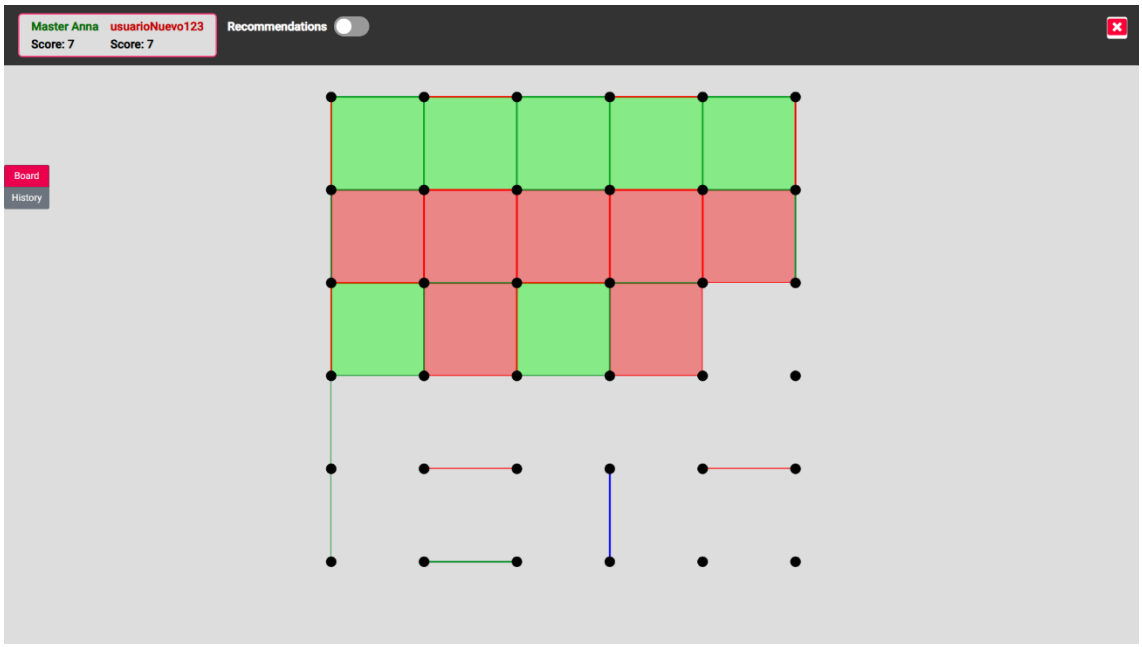


Figura 9.30. Pantalla de juego de una partida de entrenamiento: tablero

"Tree". Aquí es donde podrá explorar toda la información generada por el algoritmo inteligente. El jugador podrá, en todo momento, cambiar entre las vistas de tablero, árbol e historial, libremente y sin afectar a la partida hasta que desee ejecutar un movimiento.

Figura 9.31. Pantalla de juego de una partida de entrenamiento: vista de árbol

En esta vista de árbol, aparecerán todos los posibles movimientos (cada uno representado por un nodo) a partir del estado inicial, y un subconjunto de los posibles estados que seguirían a esos movimientos. El movimiento que tiene más posibilidades de resultar en victoria para el jugador aparece destacado en naranja. Pulsando en los distintos nodos del árbol, el usuario podrá expandirlos o colapsarlos (aquellos con el fondo azul claro, aún tienen hijos por mostrar), además de realizar su movimiento usando el botón "Perform selected movement". Evidentemente, solo podrá realizar movimientos que sean hijos directos del estado actual (nodos de profundidad uno). Situando el cursor sobre cualquier nodo, el usuario podrá ver la siguiente información relativa a él: estado del tablero, números de cuadrados cerrados por cada jugador (*Score*), probabilidad de victoria para cada jugador (*Percentage of victory*) y número de visitas del algoritmo para ese nodo (*Visits*). Se puede ver un ejemplo de esto en la Figura 9.31.

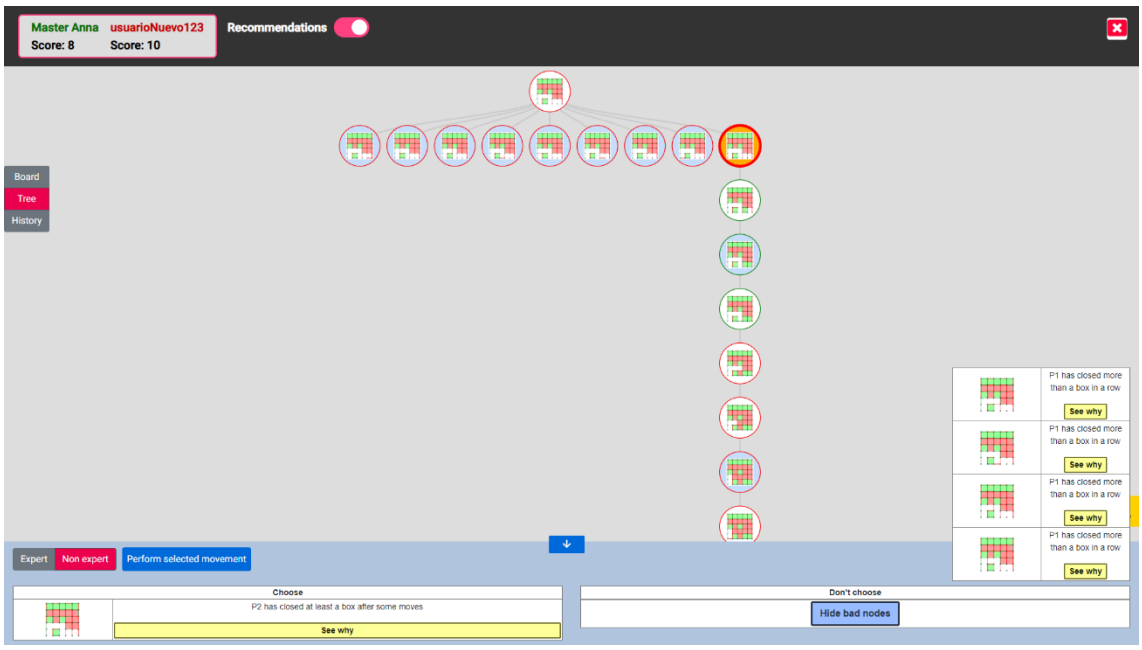


Figura 9.32. Pantalla de juego de una partida de entrenamiento: vista de árbol en modo no experto

Además, el usuario dispondrá de dos modos principales para explorar sus posibilidades: jugador no experto (Fig. 32) y jugador experto (Fig. 33), entre los que podrá cambiar en cualquier momento. El primer modo está enfocado a aquellos jugadores que saben menos del juego o que no desean explorar los datos de los que disponen en profundidad. Aquí, la herramienta simplemente muestra un movimiento recomendado y una lista de aquellos movimientos que deberían evitarse, cada uno de ellos acompañado de una breve explicación textual y un botón "See why" que mostrará la evolución más probable (según los resultados obtenidos del algoritmo) de la partida tras ese movimiento. Estas recomendaciones se calculan a través de una fórmula que tiene en cuenta el porcentaje de victoria calculado por el algoritmo MCTS para cada movimiento y su número de visitas respecto al total. Este último valor vendría a ser un medidor de la "confianza" que podemos tener en los resultados obtenidos para un movimiento (a más visitas tiene un nodo, más reforzada está la previsión del algoritmo para el movimiento que ese nodo representa).

Por su parte, el modo experto está destinado a aquellos jugadores que tienen la capacidad y desean analizar sus posibilidades de una forma más exhaustiva. Aquí el jugador dispone de una serie de consultas para obtener distintos tipos de caminos (secuencias de movimientos) a partir del nodo que tenga seleccionado (este es el último que haya pulsado, se marca con un borde más grueso):

- Caminos que llevan a la victoria del Jugador 1 (verde). Contará con cuatro consultas distintas para obtener caminos de este tipo: todos, el más rápido, el más visitado por el algoritmo y el que tiene mayor porcentaje de victoria para este jugador.
- Caminos que llevan a la victoria del Jugador 2 (rojo). Similares a los anteriores, pero respecto al jugador 2.
- Caminos que llevan a empate. También dispondrían de los mismos cuatro tipos de consultas. Estas consultas están implementadas por si en un futuro se modifica el

tamaño del tablero, ya que en un tablero 5×5, no se pueden dar empates (el número de cuadrados a cerrar es 25, no divisible de forma exacta entre dos).

- Caminos con mejor porcentaje para el Jugador 1 o el Jugador 2. Esta consulta permite obtener las secuencias de movimientos que serían más ventajosas para cada uno de los jugadores.
- Camino más visitado. Esta permite visualizar una simulación del curso que es más posible que tome la partida (lo que sucedería si cada jugador realizas, en cada turno, el movimiento que el algoritmo cree que es más beneficioso para él).

Como se puede ver en la Figura 9.33, los botones de las consultas que no vayan a dar ningún resultado (las de empate, en este caso), aparecen directamente desactivados. También en esta consulta se muestra el resultado de la consulta que muestra todos los caminos generados que llevan a victoria del Jugador 2.

Además de estas consultas, en el modo experto, el jugador puede configurar el número máximo de nodos a expandir y su profundidad relativa máxima (respecto al nodo que expanda) cada vez que pulse un nodo.

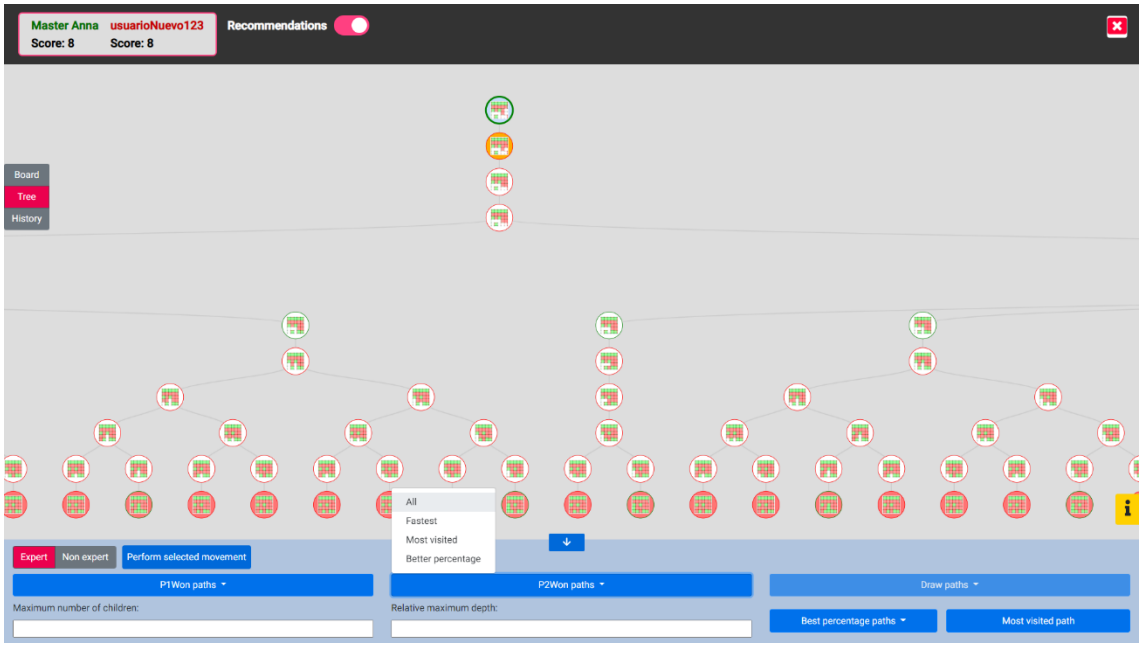


Figura 9.33. Pantalla de juego de una partida de entrenamiento: vista de árbol en modo experto

Además, para ayudar a los jugadores a entender y emplear las herramientas implementadas en la vista de árbol, existe una ventana de ayuda a la que acceder mediante el botón amarillo que aparece abajo a la derecha. En ella se explica todo lo relativo a la navegación por el árbol y la información de sus nodos (Fig. 9.35); al modo experto (Fig. 9.34), y al modo no experto (Fig. 9.36).

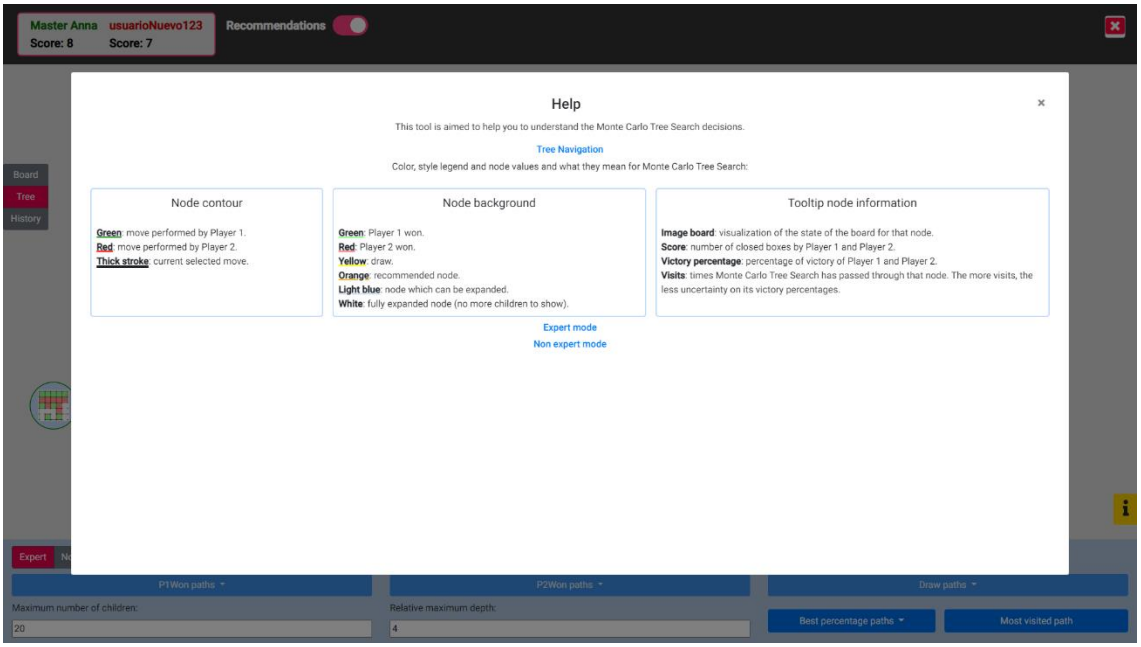


Figura 9.35. Ventana de ayuda: navegación por el árbol

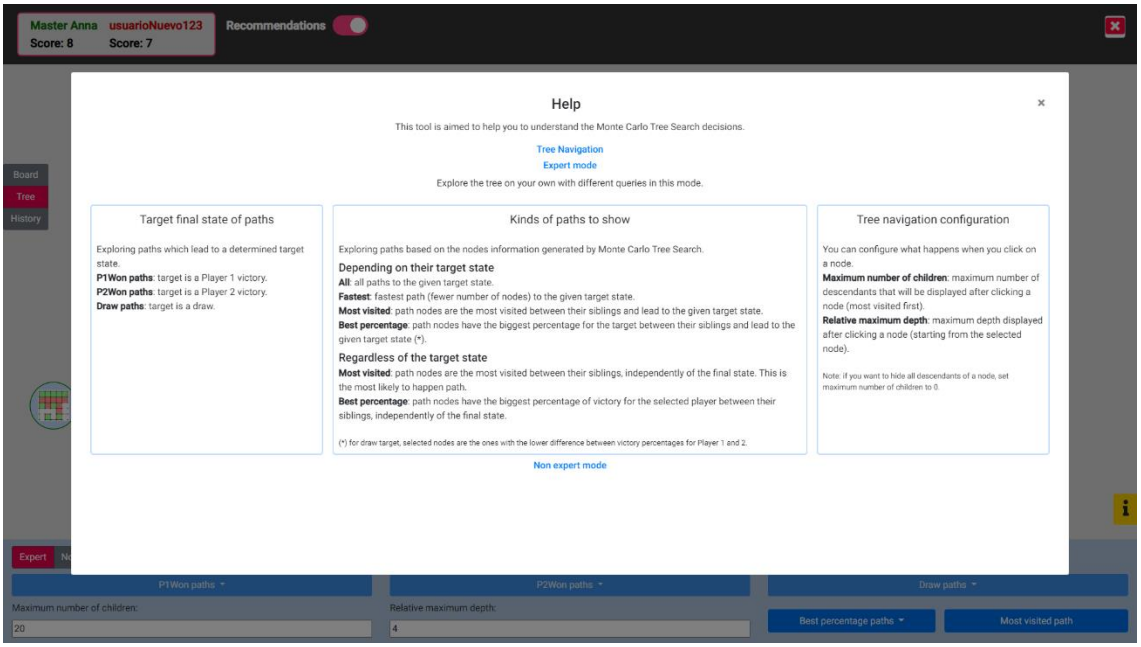


Figura 9.34. Ventana de ayuda: modo experto

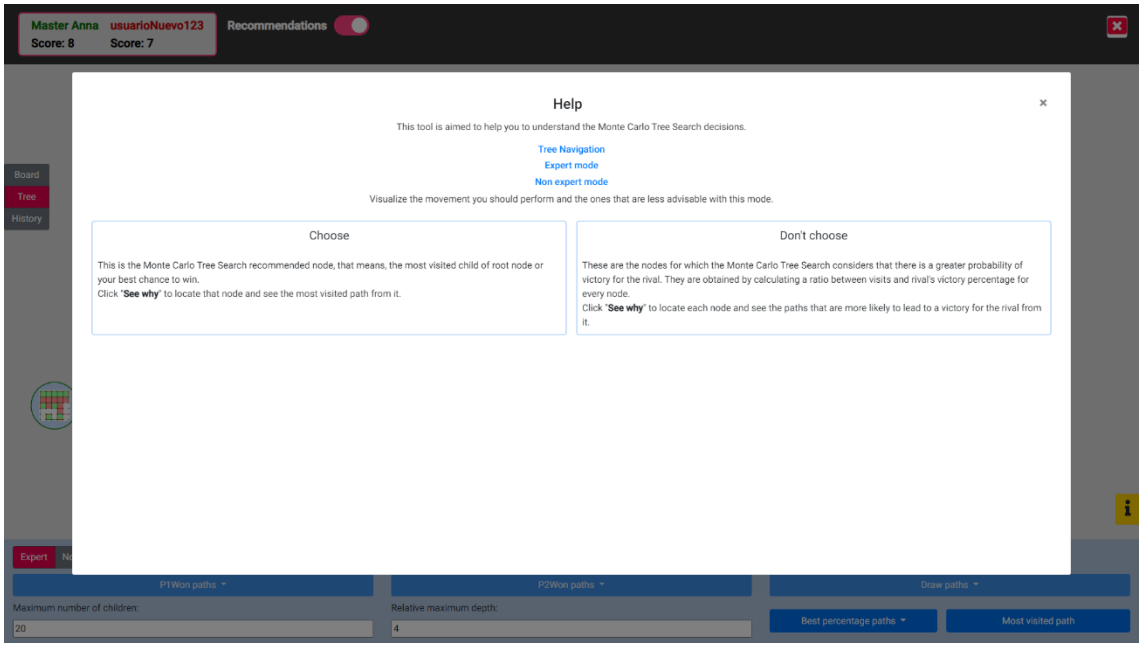


Figura 9.36. Ventana de ayuda: modo no experto

9.4 Manual del Programador

En esta sección se resumirá la estructura general del sistema para facilitar, si fuese necesario, su modificación o ampliación.

9.4.1 Interfaz de Navegador

En caso de querer crear una nueva pantalla, la opción más recomendable es emplear el comando "`ng generate <nombre-componente>`" situándose en el directorio en el que se quiera generar el nuevo componente. Esto creará todos los archivos necesarios para definir un componente. Además, es necesario declararlo en el archivo de código *app.module.ts*.

Para modificar las comunicaciones relativas a la identificación o registro de nuevos usuarios, esto debe hacerse en el componente *main-menu*. Si se desea cambiar la lógica de comunicación para jugar una partida, esta se encuentra en el componente *play*. Y, en el caso de querer cambiar o añadir nuevas herramientas para el árbol, toda su lógica está en los componentes *tree*, *tool-table-expert* y *tool-table-not-expert*, y en los archivos de código *tree.webworkers.worker.ts* y *D3Tree.ts*.

9.4.2 Tipos de Partida

Hay ciertos tipos de partidas que se podrían implementar simplemente añadiendo una línea en la clase ServerGameFactory, como podrían ser partidas de entrenamiento con tiempo limitado o partidas de entrenamiento que afectasen a la puntuación del jugador.

Para implementar nuevas formas de limitar el tiempo que tiene un jugador en cada partida simplemente sería necesario crear otra clase que implementase la interfaz IGameTimer. Así se podrían crear, por ejemplo, partidas con tiempo limitado por movimiento, pero acumulable (es decir, que si se realiza un movimiento en menos tiempo del que se tiene disponible, el restante se añadirá al tiempo que se tendrá para el próximo movimiento). También se pueden añadir nuevas formas de calcular las puntuaciones al final de una partida o de guardar sus secuencias de movimientos creando clases que implementen, respectivamente, IScoreCalculator e IGameSaver.

En caso de querer generar partidas distintas a estas, habría que añadir clases que hereden de AbstractGame. Esto permitiría, por ejemplo, crear partidas con más jugadores, o por equipos.

9.4.3 Peticiones al Servidor

En este servidor, cada petición HTTP está implementada en un *servlet* distinto, que encapsula su lógica, e interpreta y emite la información que se devolverá al cliente. Para crear una nueva ruta HTTP, se puede emplear como plantilla uno de los *servlets* sencillos ya existentes, como podría ser LogoutServlet. Además, si se crea una nueva clase de este tipo, es necesario explicitarla en el método *startServer()* de la clae AlphastarWebSocketServer. De forma similar, se puede crear un filtro que afecte a ciertas peticiones que llegan al servidor para que, por ejemplo, bloquee las peticiones que no contienen algún parámetro, o que no mandan datos correctos. Un ejemplo de filtro es la clase AccessTokenFilter. Igual que con los *servlets*, si se crea un filtro, es necesario registrarlo en el servidor a través del método ya mencionado.

Si se desea modificar el tratamiento de las comunicaciones WebSockets que permiten jugar las partidas, la lógica de ese intercambio de información está implementada en la clase AlphastarWebSocket.

9.4.4 Generar Ejecutable

Para generar un ejecutable del código del servidor (*backend*, lógica de partidas, etc.) se recomienda emplear el IDE de Eclipse para Java. La forma más sencilla es hacer *click* derecho encima de la clase que contiene el método *main()* (AlphastarWebSocketServer), y exportar como Jar ejecutable, marcando la opción que permite extraer todas las librerías necesarias a una subcarpeta al lado del JAR generado. Antes de esto, es necesario copiar la carpeta META-INF (situada dentro del proyecto en *src/main/resources*) a la carpeta *src/main/java* para que aparezca en el lugar adecuado dentro del JAR. Esta carpeta contiene archivos de configuración del servidor que emplea nuestro sistema.

Capítulo 10. Conclusiones y Ampliaciones

En este capítulo se realizará una breve revisión del trabajo desarrollado y se propondrán una serie de posibles cambios y ampliaciones que mejorarían el sistema de una u otra forma.

10.1Conclusiones

Personalmente, tengo la sensación de que este proyecto ha sido el culmen perfecto de mi trayectoria a lo largo del grado. Con él he conseguido demostrar (a mí mismo, sobre todo) que, efectivamente, estos años y todo el esfuerzo han servido para algo. He aprendido a programar, a diseñar y construir sistemas desde cero, sí; pero creo que, principalmente, he aprendido a abordar problemas en los que puedo no tener ninguna experiencia; a informarme sobre ellos, probar y corregir errores, y solucionarlos de forma satisfactoria.

Para conseguir desarrollar el sistema que estoy presentando, he tenido que leer y aprender sobre temas que, previamente, eran absolutamente desconocidos para mí: árboles de búsqueda de Monte Carlo, aprendizaje automático, fases del aprendizaje en humanos, diseño de interfaces *web*, etc. También he tenido que aprender a utilizar tecnologías totalmente nuevas para mí sin ayuda, como podrían ser Angular o el protocolo WebSockets para comunicaciones a través de red. Y he tenido que estudiar el algoritmo desarrollado por Alba Cotarelo *et al.* **[1]** para comprender cómo utilizarlo, interpretar sus resultados y, finalmente, aplicarlo al sistema.

El planteamiento inicial era desarrollar una herramienta (aplicada, en este caso, al juego de *Dots and* Boxes) que fuese capaz de presentar los resultados del algoritmo (que, a final, es un "simple" número, las visitas, asignado a cada nodo) a un usuario sin conocimientos específicos en inteligencia artificial para que pudiese aprender de ellos de forma autónoma y sin ayuda humana. He visto cómo ha evolucionado el sistema desde ese planteamiento hasta su estado actual, donde se ha convertido en una aplicación *web* usable y atractiva, que podría ser desplegada para que usuarios sin ningún conocimiento experto la empleasen para aprender y entrenar de diversas formas.

Además de esto, me ha resultado especialmente interesante ver cómo el hacer pruebas con usuarios reales y ajenos al desarrollo puede ayudar enormemente a mejorar la calidad del producto final. Ni siquiera es necesario que estos probadores sean expertos en ningún ámbito concreto: el contar con una visión distinta puede destapar fallos y defectos que quien ha desarrollado el programa no es capaz de ver.

10.2Ampliaciones

En esta sección se definirán cambios o funcionalidades previstos para el sistema, y cómo influirán en él y lo mejorarán.

10.2.1 Mejoras del Algoritmo

Aunque, como ya he mencionado, este no ha sido desarrollado por mí, sino que ha sido parte de un proyecto previo de Alba Cotarelo Tuñón, hay varias opciones con potencial para mejorar su rendimiento y resultados. Una forma de reducir los tiempos del algoritmo recortando el número de iteraciones cuando uno de los posibles movimientos se viese claramente favorito, por ejemplo, cuando el proceso de ejecución del algoritmo hubiese llegado a la mitad.

También sería interesante conseguir que el algoritmo aprendiese de los propios usuarios. Ya que emplea redes neuronales, podría reentrenarse con las partidas generadas por jugadores humanos si la aplicación se publicase. De hecho, sería relativamente fácil implementar que el servidor siguiese entrenando al algoritmo de forma periódica o cada vez que se consiguiese registrar un número de nuevas partidas de usuarios.

Otra opción que podría, potencialmente, mejorar los resultados del algoritmo, sería conseguir que comprendiese las simetrías dentro de un tablero (por ejemplo, el estado de una partida es el mismo, aunque gires 90º el tablero). La forma más directa de hacer esto sería lograr una representación numérica del estado del tablero que fuese implícitamente simétrica. Aunque esto, de ser posible, requeriría mucho trabajo.

10.2.2 Mejoras en la Interfaz de Usuario

Esto requeriría estudiar las críticas y propuestas obtenidas en las últimas pruebas de usabilidad con usuarios reales y llevarlas al sistema. Se podrían realizar cambios como el tamaño de letra, modificación de algunos iconos o de la paleta de colores empleada, o incluir un diálogo de advertencia antes de abandonar una partida, por ejemplo.

Otra posible mejora que se le podría hacer a la aplicación sería internacionalizarla, por ejemplo, añadiendo una traducción al español.

10.2.3 Nuevas Funcionalidades

Sería interesante que los usuarios dispusiesen de un historial de partidas jugadas. En él podrían ver contra quién jugaron, cuánto duró, las cajas cerradas por cada uno, el estado final, la puntuación que ganaron o perdieron, o incluso el historial de movimientos de cada partida.

También estaría bien añadir nuevos modos de juego que hiciesen la aplicación más divertida: partidas de más de dos jugadores, partidas por equipos, partidas con *power ups* (por ejemplo, podría haber cuadrados que valiesen doble o que restasen tiempo al rival), etc.

Capítulo 11. Planificación del Proyecto y Presupuesto finales

En este capítulo se explicará la planificación real seguida para el desarrollo del proyecto y se realizará una evaluación de los costes del mismo, estableciendo un presupuesto final.

11.1Planificación Final

En esta sección podrá verse la planificación final seguida, que se ha ido registrando a medida que se desarrollaba el trabajo. Se ha empleado un calendario de trabajo que establece una media de cinco horas diarias en todos los días laborables (de lunes a viernes exceptuando festivos). Esta planificación se presentará en forma de tabla primero, y luego en un diagrama de Gantt (dividido en dos imágenes para mejorar su legibilidad).

En medio de la planificación hay un período (del 01/06/2021 al 01/10/2021) que aparece vacío. Eso se debe a que no pude dedicar tiempo a este trabajo durante ese tiempo, ya que estuve plenamente centrado en otras tareas relacionadas con la beca de la FUO.

EDT	Nombre de Tarea	Duración	Comienzo	Fin
1	**Trabajo Fin de Grado**	**163,25 days**	**Thu 01/04/21**	**Mon 31/01/22**
1.1	**Trabajo previo**	**14 days**	**Thu 01/04/21**	**Tue 27/04/21**
1.1.1	Formación en IA	10 days	Thu 01/04/21	Tue 20/04/21
1.1.2	Estudio de tecnologías	2 days	Tue 20/04/21	Thu 22/04/21
1.1.3	Estudio de estado del arte	2 days	Fri 23/04/21	Tue 27/04/21
1.2	**Primer conjunto de sprints**	**17,75 days**	**Tue 27/04/21**	**Mon 31/05/21**
1.2.1	**Análisis**	**1,88 days**	**Tue 27/04/21**	**Fri 30/04/21**
1.2.1.1	Definición del sistema	2 hrs	Tue 27/04/21	Tue 27/04/21
1.2.1.2	Requisitos del sistema	4 hrs	Wed 28/04/21	Wed 28/04/21
1.2.1.3	Identificación de subsistemas	1 hr	Wed 28/04/21	Wed 28/04/21
1.2.1.4	Análisis de clases preliminar	3 hrs	Wed 28/04/21	Thu 29/04/21
1.2.1.5	Análisis de casos de uso y escenarios	4 hrs	Thu 29/04/21	Thu 29/04/21
1.2.1.6	Análisis de la interfaz de usuario	1 hr	Fri 30/04/21	Fri 30/04/21
1.2.2	**Diseño**	**2,13 days**	**Fri 30/04/21**	**Tue 04/05/21**
1.2.2.1	Arquitectura del sistema	3 hrs	Fri 30/04/21	Fri 30/04/21
1.2.2.2	Diseño de clases	4 hrs	Fri 30/04/21	Mon 03/05/21
1.2.2.3	Diseño de la interfaz	2 hrs	Mon 03/05/21	Mon 03/05/21

1.2.2.4	Especificación del Plan de pruebas	8 hrs	Mon 03/05/21	Tue 04/05/21
1.2.3	**Implementación**	**7,75 days**	**Wed 05/05/21**	**Wed 19/05/21**
1.2.3.1	Exportar las decisiones del algoritmo desde Java	3 hrs	Wed 05/05/21	Wed 05/05/21
1.2.3.2	Crear la aplicación que es capaz de representar el árbol	15 hrs	Wed 05/05/21	Fri 07/05/21
1.2.3.3	Implementar consultas	8 hrs	Mon 10/05/21	Tue 11/05/21
1.2.3.4	Implementar recomendaciones	6 hrs	Tue 11/05/21	Wed 12/05/21
1.2.3.5	**Desarrollar partidas de entrenamiento**	**3,75 days**	**Wed 12/05/21**	**Wed 19/05/21**
1.2.3.5.1	Implementar pantalla de juego	10 hrs	Wed 12/05/21	Thu 13/05/21
1.2.3.5.2	Implementar lógica de servidor y comunicaciones	20 hrs	Fri 14/05/21	Wed 19/05/21
1.2.4	**Pruebas**	**13,38 days**	**Wed 05/05/21**	**Mon 31/05/21**
1.2.4.1	Pruebas unitarias	9,5 days	Wed 05/05/21	Mon 24/05/21
1.2.4.2	Pruebas de integración	9,5 days	Mon 10/05/21	Wed 26/05/21
1.2.4.3	Pruebas de accesibilidad y usabilidad con usuarios	6 days	Wed 19/05/21	Mon 31/05/21
1.3	**Segundo conjunto de sprints**	**47 days**	**Fri 01/10/21**	**Tue 28/12/21**
1.3.1	**Análisis**	**3,38 days**	**Fri 01/10/21**	**Thu 07/10/21**
1.3.1.1	Definición del sistema	2 hrs	Fri 01/10/21	Fri 01/10/21
1.3.1.2	Requisitos del sistema	8 hrs	Fri 01/10/21	Mon 04/10/21
1.3.1.3	Identificación de subsistemas	2 hrs	Mon 04/10/21	Mon 04/10/21
1.3.1.4	Análisis de clases preliminar	8 hrs	Tue 05/10/21	Wed 06/10/21
1.3.1.5	Análisis de casos de uso y escenarios	6 hrs	Wed 06/10/21	Thu 07/10/21
1.3.1.6	Análisis de la interfaz de usuario	1 hr	Thu 07/10/21	Thu 07/10/21
1.3.2	**Diseño**	**4,75 days**	**Thu 07/10/21**	**Fri 15/10/21**
1.3.2.1	Arquitectura del sistema	4 hrs	Thu 07/10/21	Fri 08/10/21
1.3.2.2	Diseño de clases	12 hrs	Fri 08/10/21	Tue 12/10/21
1.3.2.3	Diseño de la BD	2 hrs	Tue 12/10/21	Tue 12/10/21
1.3.2.4	Diseño de la interfaz	8 hrs	Tue 12/10/21	Wed 13/10/21
1.3.2.5	Especificación del Plan de pruebas	12 hrs	Wed 13/10/21	Fri 15/10/21
1.3.3	**Implementación**	**34 days**	**Fri 15/10/21**	**Mon 20/12/21**
1.3.3.1	Solucionar los problemas detectados en las pruebas con usuarios	15 hrs	Fri 15/10/21	Wed 20/10/21
1.3.3.2	Implementar las mejoras derivadas de las	25 hrs	Wed	Tue 26/10/21

	pruebas con usuarios		20/10/21	
1.3.3.3	Implementar comunicaciones entre servidor y BD	4 hrs	Tue 26/10/21	Wed 27/10/21
1.3.3.4	**Implementar partidas multijugador**	**7,75 days**	**Wed 27/10/21**	**Wed 10/11/21**
1.3.3.4.1	Implementar la lógica de juego y comunicaciones en el servidor	45 hrs	Wed 27/10/21	Fri 05/11/21
1.3.3.4.2	Implementar lógica de límite de tiempo	10 hrs	Fri 05/11/21	Tue 09/11/21
1.3.3.4.3	Implementar lógica de puntuaciones	7 hrs	Tue 09/11/21	Wed 10/11/21
1.3.3.5	Añadir guardado de partidas	6 hrs	Wed 10/11/21	Thu 11/11/21
1.3.3.6	Implementar lógica de sesiones y registro	40 hrs	Thu 11/11/21	Mon 22/11/21
1.3.3.7	Implementar aplicación de navegador	15 days	Mon 22/11/21	Mon 20/12/21
1.3.4	**Pruebas**	**36,88 days**	**Wed 20/10/21**	**Tue 28/12/21**
1.3.4.1	Pruebas unitarias	33 days	Wed 20/10/21	Tue 21/12/21
1.3.4.2	Pruebas de integración	30 days	Wed 27/10/21	Wed 22/12/21
1.3.4.3	Pruebas de accesibilidad y usabilidad con usuarios	4 days	Mon 20/12/21	Mon 27/12/21
1.3.4.4	Pruebas de rendimiento	6 hrs	Mon 27/12/21	Tue 28/12/21
1.4	Documentación del proyecto	65 days	Fri 01/10/21	Mon 31/01/22

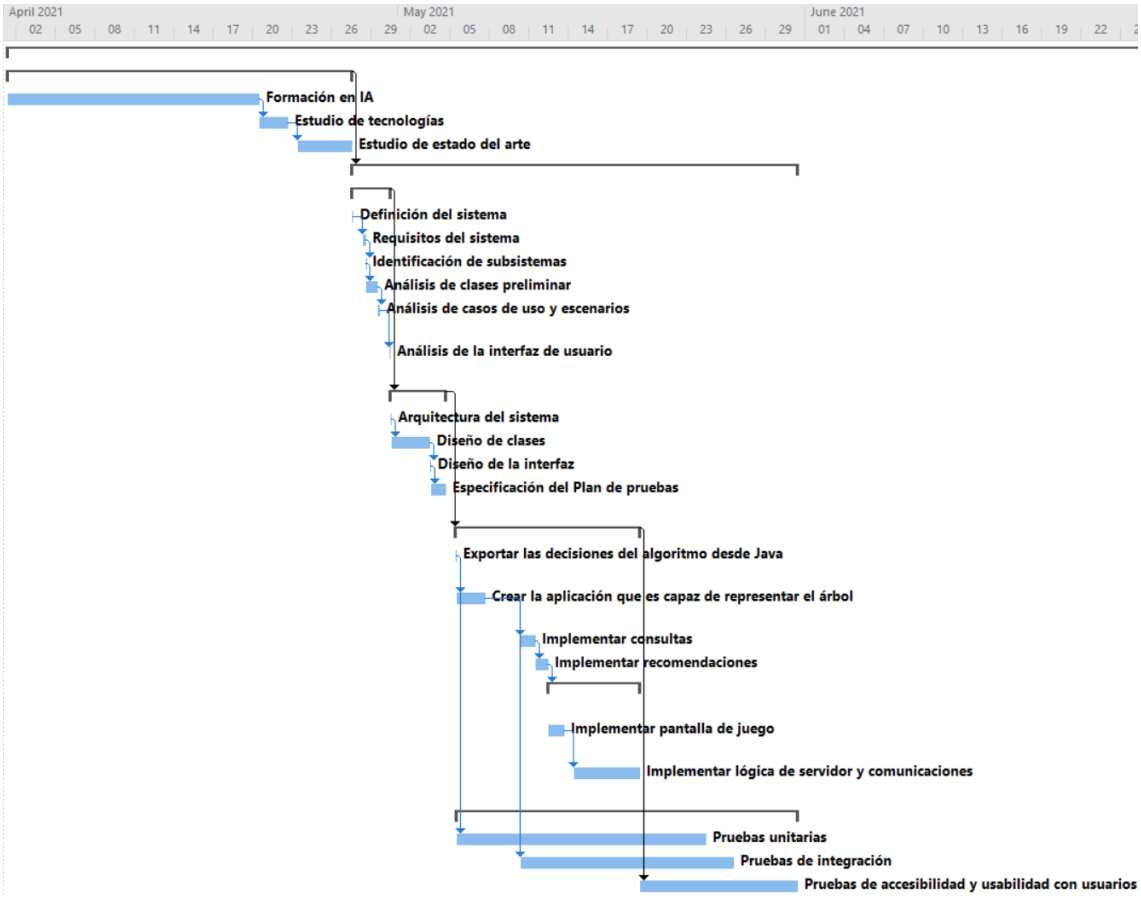


Figura 11.1. Diagrama de Gantt de la planificación final. Parte 1

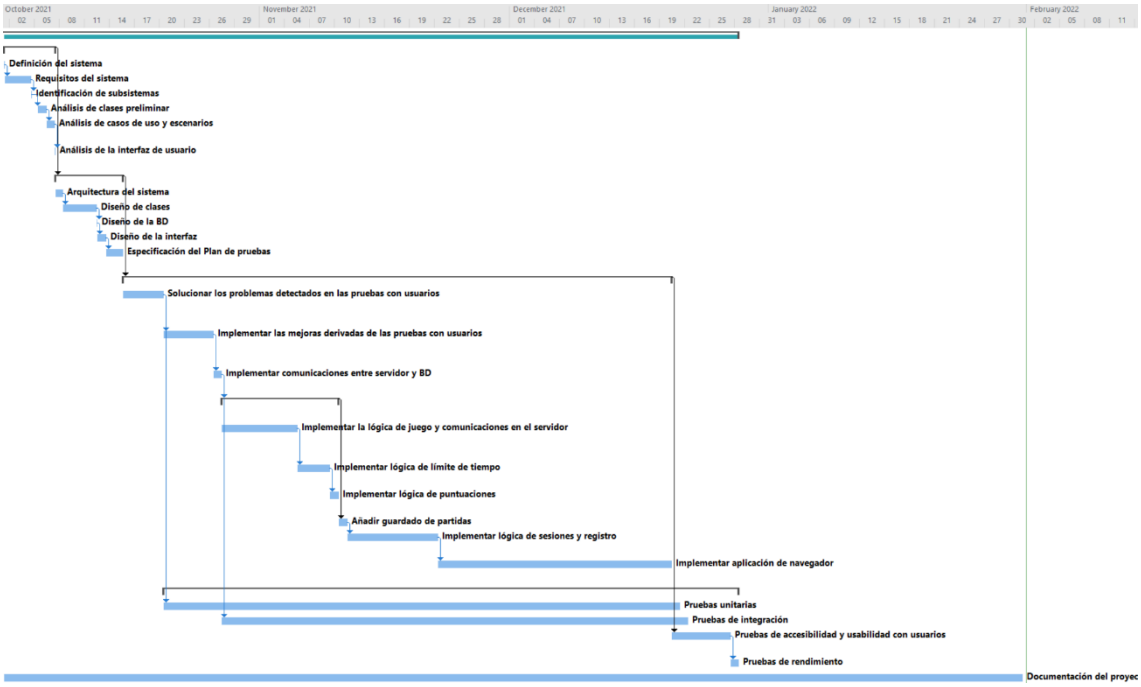


Figura 11.2. Diagrama de Gantt de la planificación final. Parte 2

11.2 Presupuesto Final

La siguiente tabla mostrará el presupuesto final interno del proyecto. Como el desarrollo no se ha llevado a cabo por módulos, sino que se ha dividido en dos conjuntos de *sprints* claramente diferenciados, los conceptos se han establecido en base al trabajo realizado en cada uno de ellos.

Ítem	Concepto	Cantidad	Amortización	Precio Unitario o mensual (€)	Total (€)
1	*Investigación Inicial*				
1.1	Formación en IA	-	100%	750	750
1.2	Estudio de tecnologías	-	100%	150	150
1.3	Estudio de estado del arte	-	100%	150	150
2	*Recursos Software*				
2.1	Microsoft Windows 10 Pro	2	10%	259	51.9
2.2	Microsoft 365 Empresa Estándar	6	100%	10.5	63
2.3	Visual Studio Code	1	100%	0	0
2.4	Eclipse IDE	1	100%	0	0
2.5	Java EE 13	1	100%	0	0
2.6	Project Plan 3	6	100%	25.3	151.8
3	*Recursos Hardware*				
3.1	Portátil Lenovo	1	10%	999	99.9
3.2	Portátil Dell	1	10%	325	32.5
4	*Módulos del proyecto*				
4.1	Análisis iteración 1	1	100%	141	141
4.2	Diseño iteración 1	1	100%	159.75	159.75
4.3	Implementación iteración 1	1	100%	581.25	581.25
4.4	Pruebas iteración 1	1	100%	735	735
4.5	Análisis iteración 2	1	100%	253.5	253.5
4.6	Diseño iteración 2	1	100%	356.25	356.25
4.7	Implementación iteración 2	1	100%	2550	2550
4.8	Pruebas iteración 2	1	100%	1335	1335
4.9	Documentación del proyecto	1	100%	1218.75	1218.75
5	*Otros*				
5.1	Internet 600MB	6	100%	30	180
5.2	Electricidad	6	100%	40	240
Subtotal					9199.6
Subtotal + Beneficio (10%)					10119.56
IVA (21%)					2125.11
TOTAL					**12244.67**

11.2.1 Presupuesto Final (Cliente)

En este breve apartado se muestra el desglose del presupuesto que se enviaría al cliente. Se puede observar una desviación de poco más de 1€ (de aumento) con respecto al total que se había calculado para el presupuesto interno. Esto es debido a que, a la hora de imputar la parte proporcional del beneficio a cada concepto, se ha decidido poner un número de horas con un máximo dos decimales.

Concepto	Cantidad	Precio Unitario	Coste Total Concepto
Horas de Investigación y Formación	70.00	17.00 €	1,190.00 €
Horas de Análisis y Diseño	60.70	17.00 €	1,031.90 €
Horas de Implementación	197.20	17.00 €	3,352.40 €
Horas de Pruebas	138.00	17.00 €	2,346.00 €
Horas de Documentación	81.25	17.00 €	1,381.25 €
Internet 600MB Mensual	6.00	30.00 €	180.00 €
Electricidad Mensual	6.00	40.00 €	240.00 €
Portátil Lenovo	0.10	999.00 €	99.90 €
Portátil Dell	0.10	325.00 €	32.50 €
Microsoft Windows 10 Pro	0.20	259.00 €	51.90 €
Microsoft 365 Empresa Estándar Mensual	6.00	10.50 €	63.00 €
Project Plan 3 Mensual	6.00	25.30 €	151.80 €
Visual Studio Code	1.00	0.00 €	0.00 €
Eclipse IDE	1.00	0.00 €	0.00 €
Java EE 13	1.00	0.00 €	0.00 €
		Subtotal	10,120.65 €
		IVA (21%)	2,125.34 €
		TOTAL	**12,245.99 €**

Capítulo 12. Referencias Bibliográficas

12.1Libros y Artículos

[1] Cotarelo, A.; García-Díaz, V.; Núñez-Valdez, E.R.; González García, C.; Gómez, A.; Chun-Wei Lin, J. "Improving Monte Carlo Tree Search with Artificial Neural Networks without Heuristics". Applied Sciences. 2021.

[2] Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T.; Simonyan, K.; Hassabis, D. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm". Arxiv.org. 2017. https://arxiv.org/abs/1712.01815

[3] Lucas, E. "L'Arithmetique Amusante". 1889. ISBN 9780274852116.

[4] González García, C.; Núñez-Valdez, E.R.; García-Díaz, V.; Pelayo G-Bustelo, B.C.; Cueva Lovelle, J.L. "A Review of Artificial Intelligence in the Internet of Things". IJIMAI Journal. 2019.

[5] El Naqa, I.; Murphy, M.J. "What Is Machine Learning?". Machine Learning in Radiation Oncology. 2015.

[6] Mamoshina, P.; Vieira, A.; Putin, E.; Zhavoronkov, A. "Applications of Deep Learning in Biomedicine". Molecular Pharmaceutics. 2016.

[7] El Naqa, I.; Ruijiang, L.; Murphy, M.J. "Machine Learning in Radiation Oncology". Springer. 2015. ISBN 978-3-319-18304-6.

[8] Lin, L; Liu, J.; Zhang, X.; Liang, X. "Automatic translation of spoken English based on improved machine learning algorithm". Journal of Intelligent and Fuzzy Systems. 2021.

[9] Soni, A.; Dharmacharya, D.; Pal, A.; Srivastava, V.K.; Shaw, R.N.; Ghosh, A. "Design of a Machine Learning-Based Self-driving Car". Machine Learning for Robotics Applications. 2021.

[10] Sharif, M.; Bhagavatula, S.; Bauer, L.; Reiter, M.K. "Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition". CCS 16: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 2016.

[11] Wagner-Mohnsen, H.; Altermatt, P.P. "A Combined Numerical Modeling and Machine Learning Approach for Optimization of Mass-Produced Industrial Solar Cells". IEEE Journal of Photovoltaics. 2020.

[12] Jean-Bernard Chaslot, G.M. "Monte-Carlo Tree Search". Researchgate. 2010. ISBN 978-90-8559-099-6.

[13] Baudiš, P. "Balancing MCTS by Dinamically Adjusting the Komi Value". ICGA. 2011.

[14] Gonzalo-Cristóbal, V.; Núñez-Valdez, E.R.; García-Díaz, V.; González García, C.; Cotarelo, A.; Gómez, A. "Monte Carlo Tree Search as a Tool for Self-Learning and Teaching People to Play Complete Information Board Games". Electronics. 2021.

[15] Dreyfus, S.E. "The Five-Stage Model of Adult Skill Acquisition". Bulletin of Science, Technology & Society. 2004.

12.2Referencias en Internet

[16] WebSockets – Referencias de la API Web.
https://developer.mozilla.org/es/docs/Web/API/WebSockets_API

[17] Carlos Herrera. "Ventajas de Angular para crear aplicaciones web". 2020.
https://blogueropro.com/blog/ventajas-de-angular-para-crear-aplicaciones-web

[18] Foro de MongoDB. https://www.mongodb.com/community/forums/

[19] Wikipedia – Java (programming language).
https://en.wikipedia.org/wiki/Java_(programming_language)

[20] Introducción a JSON. https://www.json.org/json-es.html

[21] Wikipedia - MongoDB. https://en.wikipedia.org/wiki/MongoDB

[22] Visual Studio Code - Code Editing. Redefined. https://code.visualstudio.com/

[23] Wikipedia – Visual Studio Code. https://es.wikipedia.org/wiki/Visual_Studio_Code

[24] Eclipse desktop & web IDEs | The Eclipse Foundation. https://www.eclipse.org/ide/

[25] PAe - Métrica v.3.
https://administracionelectronica.gob.es/pae_Home/pae_Documentacion/pae_Metodolog/pae_Metrica_v3.html

[26] Manuel Cillero. "MÉTRICA 3". https://manuel.cillero.es/doc/metodologia/metrica-3/

[27] Scott W. Ambler. "UML 2 Use Case Diagrams: An Agile Introduction".
http://www.agilemodeling.com/artifacts/useCaseDiagram.htm

[28] Scott W. Ambler. "Robustness Diagrams: An Agile Introduction".
http://www.agilemodeling.com/artifacts/robustnessDiagram.htm

[29] Scott W. Ambler. "User Interface Flow Diagrams (UI Storyboards): An Agile Introduction".
http://www.agilemodeling.com/artifacts/uiFlowDiagram.htm

[30] UML 2 Tutorial - Package Diagram
https://sparxsystems.com/resources/tutorials/uml2/package-diagram.html

[31] Scott W. Ambler. "UML 2 Package Diagrams: An Agile Introduction".
http://www.agilemodeling.com/artifacts/packageDiagram.htm

[32] Component diagram. https://www.visual-paradigm.com/VPGallery/diagrams/Component.html

[33] Scott W. Ambler. "UML 2 Component Diagrams: An Agile Introduction".
http://www.agilemodeling.com/artifacts/componentDiagram.htm

[34] Scott W. Ambler. "UML 2 Deployment Diagrams: An Agile Introduction". http://www.agilemodeling.com/artifacts/deploymentDiagram.htm

[35] What is scrum? https://www.scrum.org/resources/what-is-scrum

Capítulo 13. Apéndices

13.1 Glosario y Diccionario de Datos

- **API**: Siglas en inglés de Interfaz de Programación de Aplicaciones. Es el conjunto de procesos, funciones y métodos que ofrece un módulo o sistema a través de los cuales agentes externos podrán comunicarse con él.
- **Backend**: En informática y programación, se refiere a la parte de un sistema a la que el usuario no accede de forma directa. Generalmente, es el responsable de la lógica y la manipulación de datos.
- **Canvas**: Lienzo en inglés. En este contexto, un elemento HTML que permite dibujar gráficos de manera programática y dinámica.
- **Cluster**: Grupo en inglés. Se refiere a un conjunto de máquinas, físicas o virtuales, que están interconectadas entre sí y actúan como un único ordenador.
- **CRUD**: Siglas en inglés de Crear, Leer, Actualizar y Borrar. En este caso, se refiere a un módulo que implementa estas funcionalidades.
- **Framework**: Marco en inglés. En programación, se refiere a una plataforma que ofrece unas bases para implementar un producto *software*.
- **Frontend**: En informática, hace referencia a aquellas partes de un sistema o aplicación con las que el usuario interacciona directamente. Por ejemplo, una interfaz gráfica de usuario.
- **HTML**: Siglas de *Hipertext Markup Language*. Es un lenguaje de marcado que se emplea para el desarrollo de páginas *web*.
- **HTTP**: Siglas de *Hipertext Transfer Protocol*. Forma parte del conjunto de protocolos de internet y es el protocolo de comunicación más común entre cliente y servidor a través de red.
- **Servlet**: En Java, es una clase empleada para aumentar las capacidades de un servidor que ofrece una aplicación a la que se accede a través de un modelo de solicitud-respuesta. Estas clases son generalmente empleadas en servidores *web*.
- **Scroll**: En este caso, la herramienta que ofrecen los navegadores y permite desplazar el contenido de una pantalla de ordenador para ver material nuevo.
- **Token**: En este caso, un objeto que representa el derecho a realizar cierta operación. Concretamente, una cadena de caracteres que permite acceder a las funcionalidades reservadas para usuarios identificados.
- **URL**: Siglas en inglés de Localizador Uniforme de Recursos. Es una referencia a un recurso *web*. La aplicación más común es la referencia a una página *web* (coloquialmente, su dirección *web*).

13.2Contenido Entregado en el Archivo adjunto

13.2.1 Contenidos

En este apartado se explicará el contenido del entregable. De nuevo, quiero aclarar que no todo el código entregado aquí está desarrollado por mí (el autor de este TFG), para información más en detalle, pueden revisar las primeras tablas del Apartado 7.4.3.

Como el archivo adjunto a este TFG supera los 150 MB, se ha subido al siguiente enlace: https://unioviedo-my.sharepoint.com/:u:/g/personal/uo264074_uniovi_es/EVj6VF6WckhPjEyANWaEr2EB-rPlsQ-EauLNz66so8btYA?e=pEwD6E

13.2.1.1 Estructura General de Directorios del Archivo Adjunto

Directorio	Contenido
./ Directorio raíz del Archivo adjunto	Contiene un fichero leeme.txt explicando toda esta estructura.
./alphastar	Contiene toda la estructura de directorios de la parte Java del proyecto para desarrollo. **Ver la tabla Estructura de Directorios de Alphastar (Java).**
./alphastar-rivals	Contiene toda la estructura de directorios de la parte Angular del proyecto para desarrollo. También es el directorio base desde el que ejecutar el proceso Angular. **Ver la tabla Estructura de Directorios de Alphastar-Rivals (Angular).**
./alphastar-server-ejecutable	Contiene todo lo necesario para ejecutar la parte Java del sistema. Únicamente se necesita tener instalado el JDK 13.0.2 o superior.
./instalacion	Ficheros utilizados para la instalación del proyecto.
./documentacion	Contiene toda la documentación asociada al proyecto. Contiene tanto un *.pdf* como un *.docx*.
./documentacion/img	Directorio que contiene las imágenes utilizadas en la documentación (incluidos diagramas) en formato *.png*.
./documentacion/uml	Ficheros generados por la herramienta web Diagrams.net (formato *.drawio*).
./herramientas/desarrollo	Ficheros de instalación de las herramientas utilizadas en el desarrollo. También son necesarias las incluidas en el directorio *./instalación*, que no se han incluido aquí para no hacer la entrega más pesada con archivos redundantes.

13.2.1.2 Estructura de Directorios de Alphastar (Java)

Directorio	Contenido
./alphastar	Contiene los ficheros de proyecto del IDE utilizado, el archivo con los parámetros de la red neuronal, una Java KeyStore y el archivo POM propio de Maven.
./.settings	Contiene las configuraciones del IDE de Eclipse.
./src/main/java	Contiene los archivos de código del proyecto (paquetes *alphastar.server* y *alphastar.util*). También contiene el código del algoritmo MCTS. También contiene algunos archivos de configuración relevantes: • *config.properties*: para los parámetros configurables del servidor de los que hemos ido hablando a lo largo del proyecto. • *log4j.properties*: para los archivos de registro.
./src/main/resources *./src/main/webapp*	Contienen algunos archivos de configuración que el servidor necesita.
./src/test	Contiene los archivos de código para probar el algoritmo MCTS.
./start_nodes	Guarda la representación de varios estados de tablero desde los que podría empezar una partida de entrenamiento. Ahora mismo se emplea *startNode.json*.
./target	Si se emplea Maven, es la carpeta donde se guardan los contenidos del proyecto compilados.

13.2.1.3 Estructura de Directorios de Alphastar-Rivals (Angular)

Directorio	Contenido
./alphastar-rivals	Contiene los ficheros de configuración propios de Angular, Node.js y NPM. También contiene el archivo *globalBariablesConfig.ts*, donde se puede configurar la dirección del servidor a la hora de desplegar el sistema.
./src	Un poco de código del proyecto y *styles.css*, donde se guardan los estilos que deben estar disponibles para todos los componentes.
./src/app	Contiene el código base de la aplicación.
./src/app/components	Contiene la mayoría del código de la aplicación de navegador. Es donde se guardan todos los componentes.
./src/app/model	Aquí se define el modelo de algunos objetos.
./src/assets	Aquí se guardan ciertos recursos, como

	pueden ser imágenes o iconos.
./src/enviroments	En esta carpeta se guardan los distintos *enviroments* que puede utilizr este proyecto Angular.

13.2.2 Código Ejecutable e Instalación

Para iniciar el sistema, que consta de dos procesos distintos, es necesario tener instalado Angular 11.0.3, o superior, y Java 13.0.2, o superior. En los apartados 9.1.1 y 9.2.1 hay dos manuales que muestran como instalar, respectivamente, Java y Angular.

El primer paso es averiguar qué IP está usando su máquina dentro de la red, esto puede hacerlo a través del comando "`ipconfig`", si está usando Windows. Deberá usar esta IP como valor de la variable *alphastarServerIP*, contenida en el archivo *globalVariablesConfig.ts* que, a su vez, podrá localizar en el directorio raíz del proyecto Angular. En caso de que solo desee probar la aplicación en su máquina, y no desplegarla en su red, puede asignar el valor "localhost" a esta variable. En ese mismo directorio, abra una terminal de Windows y ejecute "`npm install`", para instalar las dependencias necesarias.

A continuación, abra una terminal en el directorio raíz del código de Angular y ejecute el comando "`ng serve --host 0.0.0.0 --port 80`". Para finalizar, acceda al directorio en el que se encuentra el archivo alphastarServer.jar y ejecute el comando "`java -Djdk.tls.client.protocols=TLSv1.2 -jar alphastarServer.jar`".

Una vez haya hecho todo esto, podrá acceder a la aplicación a través de un navegador con interfaz gráfica, poniendo en la barra de búsqueda "localhost" o la dirección IP de su máquina dentro de la red que esté usando.

13.2.3 Ficheros de Configuración

Para el servicio desarrollado en Angular, los archivos de configuración necesarios son los que ya se encuentran dentro de la carpeta de código del proyecto, y son propios de Angular y npm. El único archivo que se deberá modificar es *globalVariablesConfig.ts*, situado en el directorio raíz del proyecto. En él podrá encontrar una variable, *alphastarServerIP*, a la cual tendrá que asignar como valor la dirección IP o dominio de la máquina que esté sirviendo el código de *backend*.

Para el proceso de Java, es necesario que, en el mismo directorio en el que se encuentra el JAR ejecutable (alphastarServer.jar) y a su mismo nivel, estén el archivo *network_big_115.eg*, y las carpetas *start_nodes* y *src/main/java*. La primera de las carpetas debe contener un JSON con nombre *startNode*, mientras que la segunda contiene los archivos *cert.pem*, *config.properties* y *log4j.properties*. A continuación, se explicará brevemente el contenido de cada archivo:

- ***network_big_115.eg***. Este archivo contiene los parámetros y configuraciones de la red neuronal que emplea el algoritmo inteligente.

- ***startNode.json***. Guarda la representación, en forma de vector, del estado de tablero desde el que empezarán las partidas de entrenamiento.
- ***cert.pem***. Contiene un certificado autofirmado.
- ***config.properties***. Este es el archivo de configuración del servidor, contiene todos los parámetros configurables de los que se ha hablado a lo largo de la documentación, además de otros relativos al algoritmo inteligente.
- ***log4j.properties***. Aquí se guardan las configuraciones de usadas por la librería Log4j, que el sistema usa para generar archivos de registro.

13.3Código Fuente

Los archivos de código del proyecto Java pueden encontrarse en la carpeta *alphastar*, bajo el directorio raíz de esta entrega. Por otra parte, el código fuente de la parte desarrollada con Angular se sitúa en la carpeta *alphastar-rivals*, bajo el mismo directorio.

Para más detalle de como se estructuran las dos carpetas, puede ver las tablas de los Apartados 13.2.1.2 y 13.2.1.3.

13.4 Propiedad intelectual y licencias

La mayoría de las dependencias y librerías de ambos proyectos, tanto del proyecto Angular como del de Java, son de uso libre y no aparecen incrustadas en el código de este sistema.

La única excepción es el archivo de código D3Tree, creado por Rob Schmuecker y modificada para nuestro sistema concreto. Para usar y redistribuir ese código, se pide que se incluya el texto siguiente, tanto en el propio código como en la documentación del producto.