



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Proyecto de Fin de Grado en Ingeniería Informática

**DESARROLLO DEL VIDEOJUEGO HUNDIR  
LA FLOTA PARA RASPBERRY PI  
UTILIZANDO SENSE HAT**

ALEJANDRO ÁVALOS PÉREZ

Dirigido por: MIGUEL ROMERO HORTELANO

Curso: CURSO 2020/2021





# DESARROLLO DEL VIDEOJUEGO HUNDIR LA FLOTA PARA RASPBERRY PI UTILIZANDO SENSE HAT

Proyecto de Fin de Grado en Ingeniería Informática  
de modalidad *específica*

Realizado por: ALEJANDRO ÁVALOS PÉREZ

Dirigido por: MIGUEL ROMERO HORTELANO

Fecha de lectura y defensa: 8 de julio de 2021



## **Agradecimientos**

Quiero agradecer a la Universidad Nacional de Educación a Distancia por estos años de aprendizaje. A todos los profesores que he tenido por todo lo que me han enseñado. En especial a mi director de proyecto, Miguel, por su guía durante el desarrollo del mismo.

A mis compañeros de la UNED.

A mis amigos, por su paciencia y comprensión con mis largas ausencias, entendiendo que para mí esta carrera tenía más prioridad que muchos eventos importantes. Por su apoyo y ánimos, en especial a Silvia, Ana, Antonio, José, Gema, Gabriel, Pedro y su padre, que en paz descanse.

A mi pareja, Venus, por todas las horas de apoyo, comprensión, ayuda y paciencia. Por ayudarme a creer en mí en los momentos más difíciles y soportar tantas horas de ausencia, o de hablar sólo sobre los estudios. A ella quiero dedicarle el proyecto. Agradecer también al joven y prometedor Imad, y al resto de su familia por sus ánimos y paciencia.

A mi familia, por todo el apoyo recibido durante toda mi vida. A mi abuela, Fina. A quienes ya no están, el proyecto va dedicado a todos ellos.

A mi hermano José Antonio, por todo lo que siempre ha hecho por mí, nunca habría podido soñar un hermano mejor. A su mujer, Lourdes, y al pequeño Alonso.

A mis padres, José Antonio y Fina, que lo han dado todo por mí, y me han dado todo lo que tengo. Por creer siempre en mí, ayudarme con cualquier problema, y entender cualquier cosa que me pasara, en especial a mi madre, la mejor informática que conozco, siempre dispuesta a ayudarme a probar programas.

## **Desarrollo del videojuego Hundir la Flota para Raspberry Pi utilizando Sense Hat**

### **Resumen**

Los videojuegos se han convertido desde hace años en la industria del entretenimiento más potente del mundo. Cada día son jugados por más usuarios simultáneamente mediante el juego en red.

La tecnología influye directamente en la experiencia de los usuarios con los videojuegos, que cada vez demandan mejores prestaciones y precios por parte de los dispositivos que se utilizan para jugar con los videojuegos.

El desarrollo del hardware de bajo costo y la mejora en las librerías de libre distribución para lenguajes como Python posibilitan la creación de videojuegos por parte de la comunidad de usuarios.

Este proyecto desarrolla un videojuego clásico en un dispositivo de bajo coste mediante el uso del paradigma de la computación distribuida, con la que los dispositivos pueden compartir recursos eficientemente en una red.

## **Development of the video game Hundir la flota for Raspberry Pi using Sense Hat**

### **Abstract**

Video games have become the most powerful entertainment industry in the world for years. Every day they are played by more users simultaneously through network play.

Technology directly influences the user experience with video games, which is increasingly demanding better features and prices from the devices used to play video games.

The development of low-cost hardware and the improvement in free distribution libraries for languages such as Python make it possible for the user community to create videogames.

This project develops a classic video game on a low-cost device by using the paradigm of distributed computing, with which devices can efficiently share resources on a network.

## **Palabras Clave**

**Sistemas distribuidos, Hundir la Flota, Raspberry Pi, Sense Hat, videojuegos, multiplataforma**

## **Keywords**

**Distributed systems, BattleShip, Raspberry Pi, Sense Hat, video games, multi-platform**



## Contenido

|       |   |    |
|-------|---|----|
| 1     | Introducción .....                                    | 1  |
| 1.1   | Contexto .....  | 1  |
| 1.1.1 | Ámbito.....   | 1  |
| 1.1.2 | Tema y sus particularidades .....                     | 2  |
| 1.2   | Motivación .....                                      | 3  |
| 1.2.1 | Pertinencia del proyecto .....                        | 3  |
| 1.2.2 | Originalidad.....                                     | 4  |
| 1.3   | Objetivos .....                                       | 4  |
| 1.3.1 | Generales .....                                       | 4  |
| 1.3.2 | Específicos .....                                     | 4  |
| 1.4   | Aplicaciones posibles .....                           | 5  |
| 1.4.1 | Aplicabilidad del trabajo .....                       | 5  |
| 1.4.2 | Ámbitos relacionados .....                            | 5  |
| 1.4.3 | Estructura del trabajo .....                          | 6  |
| 2     | Estado del arte .....                                 | 7  |
| 2.1   | La industria del videojuego .....                     | 7  |
| 2.1.1 | Los videojuegos.....                                  | 8  |
| 2.1.2 | Videojuegos en red.....                               | 9  |
| 2.1.3 | Videojuegos multiplataforma .....                     | 10 |
| 2.2   | La escena Indie.....                                  | 11 |
| 2.3   | Los sistemas distribuidos .....                       | 11 |
| 2.4   | Las API gráficas y los motores para videojuegos ..... | 12 |
| 2.5   | Inteligencia Artificial .....                         | 13 |
| 3     | Tecnologías hardware y software utilizadas .....      | 15 |
| 3.1   | Hardware .....  | 15 |

|        |   |    |
|--------|---|----|
| 3.1.1  | Raspberry pi 3B.....  | 15 |
| 3.1.2  | Sense Hat.....  | 16 |
| 3.1.3  | Redes de computadores. Arquitecturas .....                                | 16 |
| 3.2    | Software .....  | 17 |
| 3.2.1  | Sistemas Operativos. Raspbian, Windows, Ubuntu. ....                      | 17 |
| 3.2.2  | Lenguajes de programación. Python. El Shell.....                          | 17 |
| 3.2.3  | Los IDE y los editores de código. PyCharm y Visual Studio Code .....      | 18 |
| 3.2.4  | Herramientas para diseño y modelado de software. Astah UML.....           | 20 |
| 3.2.5  | Pyro5. API middleware para sistemas distribuidos.....                     | 21 |
| 3.2.6  | Las API para GUI y los motores para videojuegos. Tkinter y Pygame.....    | 22 |
| 3.2.7  | API para bases de datos SQL. SQLite3.....                                 | 23 |
| 3.2.8  | Simulador de placa Sense Hat .....  | 23 |
| 3.2.9  | Wireshark .....   | 23 |
| 3.2.10 | Seguridad. Protocolo SSL/TLS y certificados auto firmados.....            | 23 |
| 4      | Construcción de la solución .....   | 27 |
| 4.1    | Proceso de desarrollo software y modelos de proceso .....                 | 27 |
| 4.2    | Modelos de proceso prescriptivo.....                                      | 27 |
| 4.2.1  | Modelo de proceso incremental .....                                       | 28 |
| 4.2.2  | Modelos de proceso evolutivo.....   | 28 |
| 4.2.3  | El Proceso Unificado.....   | 29 |
| 4.2.4  | Modelo de proceso personal.....   | 29 |
| 4.2.5  | Elección de la metodología .....  | 30 |
| 4.3    | Fases del desarrollo de la aplicación .....                               | 30 |
| 4.3.1  | Especificación de requisitos. Funcionales y no funcionales .....          | 30 |
| 4.3.2  | Análisis del problema. Diagrama de Casos de Uso.....                      | 32 |
| 4.3.3  | Modelado de datos. Diagrama de Modelo de Dominio.....                     | 42 |
| 4.3.4  | Diseño de la solución. Diagramas de Interacción y Diagrama de Clases..... | 44 |

|       |  |     |
|-------|--|-----|
| 4.3.5 | Codificación de la solución. Modelo de Implementación .....                    | 53  |
| 4.3.6 | Pruebas del producto. Caja blanca y caja negra. Testing avanzado .....         | 67  |
| 4.3.7 | Despliegue del producto. Entorno virtual y dependencias .....                  | 68  |
| 4.3.8 | Mantenimiento. Posibilidad de cambio. Acoplamiento y cohesión .....            | 69  |
| 5     | Estudio del comportamiento.....  | 73  |
| 5.1   | Pruebas de caja blanca y de caja negra.....                                    | 73  |
| 5.2   | Casos de uso .....   | 74  |
| 5.2.1 | Base de datos.....   | 75  |
| 5.2.2 | Servidor .....   | 76  |
| 5.2.3 | Cliente .....  | 78  |
| 6     | Planificación y presupuesto.....   | 91  |
| 6.1   | Planificación.....   | 91  |
| 6.1.1 | Actividades.....   | 91  |
| 6.1.2 | Diagrama de Gantt .....  | 92  |
| 6.2   | Presupuesto .....  | 93  |
| 6.2.1 | Estimación del esfuerzo en unidades de persona/mes .....                       | 93  |
| 6.2.2 | Estimación del coste final del proyecto.....                                   | 95  |
| 7     | Conclusiones y trabajo futuro .....  | 97  |
| 7.1   | Conclusiones .....   | 97  |
| 7.2   | Dificultades encontradas y modificaciones realizadas al diseño inicial.....    | 100 |
| 7.2.1 | Paso de Pyro4 a Pyro5.....   | 100 |
| 7.2.2 | Las conexiones de red local.....   | 101 |
| 7.3   | Trabajo futuro.....  | 101 |
| 7.3.1 | Nuevas interfaces de usuario.....  | 101 |
| 7.3.2 | Nuevas IA.....   | 102 |
| 7.3.3 | Otros adaptadores para bases de datos, renders, o computación distribuida..... | 102 |
| 7.3.4 | Más usos para SenseHat.....  | 102 |

|       |  |     |
|-------|--|-----|
| 7.3.5 | Mejorar la arquitectura para que soporte más juegos .....          | 103 |
| 7.3.6 | Portarlo a más Sistemas Operativos .....                           | 103 |
| 8     | Bibliografía .....   | 105 |
| 9     | Anexos.....  | 107 |
| I.    | Creación de un certificado SSL auto firmado mediante openssl ..... | 107 |

|   |    |
|---|----|
| Ilustración 1: Entorno integrado de desarrollo PyCharm .....                          | 19 |
| Ilustración 2: Editor de código Visual Studio Code .....                              | 20 |
| Ilustración 3: Herramienta de modelado Astah UML .....                                | 21 |
| Ilustración 4: Captura de datos sin SSL .....   | 25 |
| Ilustración 5: Captura de datos con SSL .....   | 26 |
| Ilustración 6: Diagrama de casos de uso .....   | 34 |
| Ilustración 7: Diagrama de modelo de dominio .....                                    | 43 |
| Ilustración 8: Diagrama de secuencia del CU Registrarse .....                         | 45 |
| Ilustración 9: Diagrama de secuencia del CU Autenticarse .....                        | 45 |
| Ilustración 10: Diagrama de secuencia del CU Salir del sistema .....                  | 46 |
| Ilustración 11: Diagrama de secuencia del CU Consultar datos del jugador .....        | 46 |
| Ilustración 12: Diagrama de secuencia del CU Crear partida .....                      | 47 |
| Ilustración 13: Diagrama de secuencia del CU Listar partidas en espera .....          | 47 |
| Ilustración 14: Diagrama de secuencia del CU Entrar en partida .....                  | 48 |
| Ilustración 15: Diagrama de secuencia del CU Jugar partida.....                       | 49 |
| Ilustración 16: Diagrama de secuencia del CU Configurar aplicación .....              | 50 |
| Ilustración 17: Diagrama de secuencia del CU Monitorizar servicios del servidor ..... | 50 |
| Ilustración 18: Diagrama de secuencia del CU Consultar partidas en juego.....         | 51 |
| Ilustración 19: Diagrama de secuencia del CU Monitorizar servicios de la BD.....      | 51 |
| Ilustración 20: Diagrama de secuencia del CU Consultar jugadores registrados.....     | 51 |
| Ilustración 21: Diagrama de clases.....   | 52 |
| Ilustración 22: Prueba de caja negra para testear la creación de un barco .....       | 74 |



## 1 Introducción

Este documento es la memoria del Proyecto de Fin de Grado del Grado en Ingeniería Informática “Desarrollo del videojuego Hundir la Flota para Raspberry Pi utilizando Sense Hat”.

El objetivo del mismo es evaluar el trabajo realizado durante el curso universitario, en el que se aplican los conocimientos adquiridos durante la titulación para resolver un problema de ingeniería, tratando de aportar un enfoque original y aplicable en el mundo comercial.

### 1.1 Contexto

En este apartado se introducirá el contexto del proyecto, explicando el ámbito general en el que se enmarca dentro de la informática. Después se detallará el tema dentro del ámbito y las particularidades que lo caracterizan.

#### 1.1.1 Ámbito

El proyecto se desarrolla en varios ámbitos asociados a los videojuegos: sistemas de comunicación y control, multimedia, sistemas operativos, bases de datos y seguridad.

Los videojuegos pueden ser sistemas complejos formados por servidores, bases de datos, clientes, sistemas de pago, etcétera. Al estar formados por distintos sistemas, se hace necesaria la aplicación de los fundamentos teóricos de los sistemas de comunicación y control, pues esta rama se centra en los distintos modos y arquitecturas que se pueden realizar entre distintos dispositivos pertenecientes a una red.

No se puede entender un videojuego sin imagen y sonido, siendo estos dos componentes los fundamentos del estudio de la multimedia. Se necesitarán conceptos de informática gráfica, así como conceptos físicos y electrónicos sobre el sonido.

Un videojuego moderno se ejecuta en un sistema operativo. Si se quiere implementar con éxito un videojuego o aplicación sobre uno o varios sistemas operativos, será necesario comprender a grandes rasgos, y particularmente en muchas ocasiones, cómo funciona un sistema operativo.

En los videojuegos en red, jugados por miles de usuarios, se hace necesario utilizar un sistema que permita la gestión permanente de los datos de éstos. Este es el ámbito de las bases de datos, la organización y gestión eficiente de los datos.

La seguridad es importante en los videojuegos, y cada día más, ya que el juego en red es cada día más habitual. En la rama de la seguridad en la informática se estudian en detalle los protocolos de comunicación por red, así como el uso de sistemas de seguridad por parte de los sistemas operativos, como los cortafuegos. La seguridad se centra también en la organización interna y análisis de las redes informáticas.

### 1.1.2 Tema y sus particularidades

El tema específico es la implementación de un videojuego utilizando computación distribuida. La computación distribuida se utiliza para compartir recursos entre varios equipos conectados a una red de forma transparente para cada equipo.

En un sistema distribuido se ha de diseñar una arquitectura subyacente a la red en la que los equipos están conectados. Existen varios modelos que se pueden utilizar como base, como son el modelo cliente-servidor, el modelo cliente a cliente (P2P). Esta arquitectura que permite la conexión de varios equipos de forma transparente para los usuarios es conocida como middleware. Mediante el uso de librerías que implementan el middleware para una aplicación, el programador no tiene que lidiar con protocolos de conexión en la red. Esto traslada una preocupación al programador, que debe elegir con cuidado qué servicio tiene la responsabilidad de la lógica de una aplicación, pudiéndose hacer ciertos cálculos o comprobaciones tanto en un cliente, como en un servidor.



Además, como particularidades se tendrán la gestión de un sistema de registro de usuarios, el uso de dispositivos electrónicos, la seguridad en el envío de información por la red entre los equipos que forman parte del sistema distribuido, etc. Una particularidad importante es el atractivo visual, si bien no es el único atractivo que tiene un videojuego para los usuarios. La inteligencia artificial y el diseño de niveles es pieza fundamental en el diseño de un videojuego que busca ser atractivo.

## 1.2 Motivación

Esta sección trata sobre la justificación del tema elegido para el proyecto, discutiendo la pertinencia del mismo a día de hoy y reflexionando sobre la originalidad de la solución propuesta.

### 1.2.1 Pertinencia del proyecto

En estos días, la tecnología juega un factor determinante en la experiencia de juego de los usuarios. Una mala conexión a Internet puede ocasionar que no se muestren correctamente en pantalla los eventos, debido a los retrasos que se puedan dar en el envío de la información. Este retraso se conoce como latencia, y puede reducir considerablemente la calidad del producto para el usuario. Por otro lado, un equipo con características técnicas modestas no será capaz de ejecutar con la misma fluidez un videojuego que otro de características superiores. Habitualmente los usuarios deben invertir dinero en mejorar sus dispositivos o adquirir otros más potentes.

Para la industria del videojuego la tecnología también juega un papel fundamental. Las compañías más grandes invierten dinero en adquirir más servidores para poder prestar servicio a un número creciente de usuarios.

Se hace necesaria una solución tecnológica para los usuarios y la industria del videojuego, que evite la necesidad de mejorar las prestaciones del servicio de juego en red a costa de la inversión en equipos cada vez más caros.

### 1.2.2 Originalidad

La industria de los videojuegos está bien establecida tras décadas de progreso. El juego en red es mantenido por muchas compañías que mantienen en su mayoría un modelo de cliente-servidor con un servidor centralizado. Una variante es tener multitud de servidores centrales, repartidos en ciertas partes del mundo, con lo que jugadores de distintas regiones no tienen por qué llegar a coincidir en una partida.

Este proyecto desarrolla la idea de que la computación distribuida en el ámbito de los videojuegos mejora la calidad del servicio para los usuarios, que pueden compartir los recursos y la carga del mantenimiento de la red en la que juegan. Esta solución permite aumentar el número de usuarios simultáneos de forma natural por la propia tecnología que utiliza, sin tener que invertir dinero en la compra de más servidores para mejorar la capacidad de cómputo. Los jugadores pueden ejecutar aplicaciones con muchos menos recursos técnicos, con lo que pueden jugar en ordenadores de bajo consumo y bajo coste.

## 1.3 Objetivos

Esta sección plantea los objetivos generales que busca cumplir el proyecto, así como los objetivos específicos. Se expresarán en forma esquemática y reducida. El orden en que aparecen no es relevante.

### 1.3.1 Generales

Desarrollo de un videojuego.

Uso de hardware de bajo precio y bajo consumo.

Uso de la tecnología de computación en sistemas distribuidos.

### 1.3.2 Específicos

Utilizar una base de datos permanente.

Utilizar un servicio de registro para los usuarios.

Garantizar la seguridad de los datos de los usuarios.

## 1.4 Aplicaciones posibles

En esta sección se discuten las posibles aplicaciones para las que está pensado el producto del desarrollo del proyecto, así como otros ámbitos relacionados en los que pudiese tener aplicación.

### 1.4.1 Aplicabilidad del trabajo

El producto tiene salida directa al mercado, mediante la posibilidad de publicarlo para su descarga en una o varias plataformas virtuales para tal propósito en la red, o bien mediante la distribución física de los archivos del programa. Al ser multiplataforma, el potencial mercado de venta es mucho mayor.

Puede utilizarse como base para construir un proyecto de otro tipo, aprovechando la arquitectura y cambiando la funcionalidad. Se podría utilizar como base para un servicio de registro y autenticación de usuarios en una base de datos en un sistema distribuido.

### 1.4.2 Ámbitos relacionados

Un ámbito relacionado es el de la educación. El proyecto puede utilizarse en el ámbito de la enseñanza de la informática o la electrónica en institutos o universidades.

Puede resultar interesante para la escena de programadores que utilizan las mismas placas que el proyecto y las empresas que comercian con ellas, dado que siempre buscan proyectos basados en su tecnología como medio de difusión de sus productos.

Los videojuegos comienzan a tener aplicación en otros ámbitos, como la salud o la empresa, y se continúa estudiando sobre la aplicación de los videojuegos en diferentes ámbitos de la cultura y la sociedad (Johannes, Vuorre, & Przybylski, 2020).

### 1.4.3 Estructura del trabajo

En primer lugar, se presentará el estado del arte, esto es; se hará un estudio de otros proyectos similares y se analizarán sus puntos fuertes y débiles, intentando comparar este proyecto con ellos.

En segundo lugar, se introducirá el marco teórico necesario junto con las tecnologías utilizadas para llevar a cabo la implementación del proyecto.

En tercer lugar, se detallará la construcción de la solución junto con el proceso seguido durante ésta.

En cuarto lugar, se explicarán las pruebas sobre el comportamiento realizadas y los resultados obtenidos, junto con la metodología llevada a cabo para efectuar tales pruebas.

En quinto lugar, se detallará la planificación que se ha seguido durante la etapa de construcción de la solución, junto con el presupuesto detallado del proyecto.

En sexto lugar, se extraerán las conclusiones sobre el proyecto, objetivos cumplidos respecto a la planificación y sobre las posibles futuras mejoras que se podrían realizar.

## 2 Estado del arte

Esta sección muestra el estado actual del tema central del proyecto, los videojuegos, así como el estado de algunas tecnologías que se van a utilizar en el desarrollo de esta solución.

### 2.1 La industria del videojuego

La industria del videojuego ha crecido en las dos últimas décadas hasta convertirse en la mayor industria del entretenimiento del planeta, superando desde 2010 a las industrias de la música y el cine juntas, como se indica en el artículo de En.Digital (En.Digital, 2018). En el mismo artículo se pueden consultar cifras como las de Grand Theft Auto 5, videojuego que vendió 11.21 millones de copias en el día de su lanzamiento, generando 817.5 millones de dólares, llegando a los 1000 millones en 3 días.

El crecimiento espectacular de los videojuegos en la plataforma de los teléfonos móviles está suponiendo todo un terremoto dentro de la industria, ya que permite la entrada masiva de nuevos jugadores, que pueden utilizar equipos muy modestos para jugar, comparados con otras plataformas como las videoconsolas o el PC. Según el anterior artículo para 2021 ya ocupan el 60% de la industria, habiendo pasado los ingresos de videojuegos en PC y consolas desde 2012 a 2021 de 57900 millones de dólares a 73840 millones de dólares en 2021. En el caso de los juegos para móviles desde 2012 a 2021 han pasado de generar 12700 millones de dólares a 106260 millones de dólares.

De acuerdo con el artículo sobre el videojuego en España de la Asociación Española de Videojuegos (AEVI, 2018), España es el décimo país en consumo de videojuegos, quinto de Europa. El Gobierno de España reconoció la importancia del videojuego como factor estratégico en el futuro de la economía.

### 2.1.1 Los videojuegos

Los videojuegos comenzaron su historia en la década de 1950. Durante mucho tiempo ha sido difícil identificar el primer videojuego de la historia. La Facultat d'Informàtica señala en un artículo (RetroInformàtica, 2002) que el primer videojuego fue Nought and crosses, también llamado OXO, desarrollado por Alexander S. Douglas en 1952. Según el artículo, los primeros videojuegos nunca salieron del ámbito universitario, siendo el primer videojuego para dos jugadores humanos Tennis for two, creado por William Higginbotham en 1958. El videojuego doméstico nacería en 1966 con el videojuego Fox and Hounds, de la mano de Ralph Baer, junto a Albert Maricon y Ted Dabney.

En 1972, el videojuego Pong de la compañía Atari acabaría confirmando a la industria de los videojuegos como una industria estable, si bien en esa misma década se acabaría experimentando una crisis en el sector norteamericano.

La llegada de los sistemas de 8 bits y las compañías japonesas revitalizarían la industria, llegando primero Nintendo y llegando a convertirse en un icono cultural junto a su videojuego más icónico, Super Mario Bros. Como puede verse en varias películas de la década de 1980, la palabra Nintendo se utiliza como sinónimo de videojuego.

La posterior llegada al mercado de Sega, junto a la nueva generación de videoconsolas de 16 bits acabaría cambiando para siempre a la industria, dado que, en su apuesta por abrirse paso en el mercado, Sega protagonizaría junto a Nintendo una campaña de marketing como no se había visto en este sector, llegando a jugadores de más edad y abriendo la puerta de los videojuegos a otras industrias del entretenimiento como las películas o la música. De esta época son las primeras películas basadas en videojuegos. En esta época nacerían las videoconsolas portátiles, cuyo más conocido representante es la videoconsola Game Boy.

Las máquinas arcade fueron fundamentales para la expansión del videojuego, con títulos que ahora tenían un impacto global como fue el caso de Street Fighter 2: World

Warrior. Este juego sentaría las bases de uno de los géneros más populares, los juegos de lucha. En casa, los usuarios comenzaban a poder jugar versiones domésticas de los juegos más populares de los salones arcade.

Un momento crítico sería el paso a las 3 dimensiones en un mundo poligonal, con la incursión en la industria de la compañía Sony y su videoconsola PlayStation, que rompería nuevos récords, convirtiéndose en el sinónimo moderno de la palabra videoconsola. Su influencia cultural fue enorme.

Con el paso del tiempo los videojuegos han ido ganando en realismo, jugabilidad y calidad en general, mejorando además facetas artísticas como la historia, la música, el ritmo, o las dinámicas de juego. Pero, con todo, los últimos adelantos en las telecomunicaciones están cambiando la forma en la que consumimos los videojuegos, dejando cada vez más atrás el formato de juego en solitario y en un hardware, en favor de un formato de juego multijugador y multiplataforma, con cada vez menos dependencia sobre el hardware.

### 2.1.2 Videojuegos en red

Un videojuego en red es un videojuego en el que se hace uso de Internet para jugar. Normalmente el objetivo es el de permitir el juego multijugador entre usuarios que no están en la misma red local. De esta forma un usuario puede jugar con otros usuarios de cualquier parte del mundo.

El juego en red tardó varias décadas en desarrollarse en la industria, si bien una vez logró cristalizar gracias a la mejora en la velocidad y capacidad de las telecomunicaciones, ha acabado logrando una importancia capital en el significado mismo de videojuego. A día de hoy la mayoría de videojuegos incluyen un modo de juego en red o contenido accesible disponible en la red, incluso muchos videojuegos no tienen sentido sin el modo multijugador en red.

Los videojuegos en red permiten la creación de comunidades en torno al videojuego en sí y a sus usuarios, añadiendo un valor social a su uso (Gómez, Pernía, & Lacasa, 2010).

Otro aspecto a tener en cuenta acerca de la dimensión social de los juegos en red es la cada vez más influyente creación y retransmisión (streaming) de contenido basado en los videojuegos. En el caso de una de las compañías más grandes del planeta, Twitch, se tienen los datos de 6300 millones de horas vistas por los usuarios, frente a los 1060 millones de horas de Facebook Gaming y los 1370 millones de Youtube Gaming (EuropaPress, 2021).

Una muestra más del impacto del juego en red es el juego competitivo, que origina multitud de eventos y competiciones de carácter internacional basado en uno o varios juegos, incluyendo modalidades como campañas benéficas basadas en el streaming de speedruns, que son partidas competitivas en las que el objetivo es romper la marca mundial de tiempo en completar un determinado juego.

### 2.1.3 Videojuegos multiplataforma

Se conoce comúnmente como plataforma al dispositivo en el que se ejecuta un videojuego, pudiendo ser varias las plataformas en las que este se ejecuta.

Cuando un videojuego es capaz de ejecutarse en múltiples plataformas, es denominado multiplataforma, y por extensión el juego multiplataforma es aquella modalidad de juego que implica el soporte multiplataforma.

Aunque a lo largo de su historia los videojuegos fueron diseñados para ejecutarse en hardware específico, desde muy pronto se realizaron versiones conocidas como *ports* de videojuegos de un sistema a otro. Desde hace algunos años la mejora en las prestaciones tecnológicas de las redes y la computación han hecho posible la proliferación de juegos multiplataforma, que cada vez se van asentando más en la industria. Un caso notable es el de Fortnite, que permite jugar a usuarios con distintas plataformas,



incluyendo el PC. Se estima que Fortnite cuenta con una media de 250 a 300 millones de jugadores, generando entre 1500 y 2000 millones de dólares al año (Algobia, 2021).

## 2.2 La escena Indie

Desde hace unos años y con la bajada de precio del hardware unida a la mejora en técnicas de software y librerías de videojuegos, se ha facilitado la posibilidad del desarrollo de un videojuego por parte de pequeñas compañías, o incluso por parte de una sola persona. Estos videojuegos, hechos por compañías independientes a la gran industria del videojuego, son llamados indie.

La calidad de los juegos indie suele ser más baja en comparación con los juegos con mayor presupuesto que desarrollan las compañías más grandes, si bien algunas veces el resultado de un proyecto indie alcanza una calidad profesional. A veces se realizan campañas de financiación para lograr establecer un proyecto de este tipo mediante las donaciones de la comunidad interesada en su mantenimiento.

Cada vez más aparecen más juegos indie, aportando frescura en la industria, que habitualmente tiene periodos de repetición creativa, con escasas mejoras en sagas anuales aparte del nivel de detalle gráfico.

Muchos medios de la prensa especializada del videojuego incluyen secciones exclusivas para este tipo de juegos.

## 2.3 Los sistemas distribuidos

Como una especie de evolución en los juegos de red, viene apareciendo en los últimos años la demanda por parte de los usuarios de videojuegos distribuidos totalmente en red, sin la dependencia del hardware específico que dé soporte a los requisitos completos del videojuego.

Un ejemplo es el juego en la nube, en el que los usuarios descargan un cliente ligero, que se encarga de recibir las acciones introducidas por el usuario en el juego y se las envía a un servidor que es responsable de todo el cómputo que el juego pueda necesitar, eliminando la necesidad por parte del jugador de invertir en equipo.

Esta demanda no pasa desapercibida para las empresas, que en los últimos años muestran una tendencia a convertirse en empresas de servicios, en lugar de empresas de productos. Esto se puede notar en la proliferación de plataformas de descarga de videojuegos, que poco a poco van haciendo prescindir del formato físico, en favor del formato digital de descarga.

Es el caso de la plataforma de Google, Stadia (Stadia, 2021). Se prevé que varias compañías se conviertan en distribuidoras de un catálogo de juegos que se podrán ejecutar en diversos dispositivos, en lugar de contar con versiones compatibles para cada dispositivo específico.

## 2.4 Las API gráficas y los motores para videojuegos

Existen multitud de herramientas que facilitan la labor de desarrollar un videojuego, desde librerías en las que poder apoyarse, como pueden ser las librerías gráficas o una librería que gestione la reproducción de audio por parte del sistema operativo.

Un tipo de software más enfocado a la creación de videojuegos son los motores de videojuegos. Habitualmente estos motores incluyen varios tipos de subsistemas o motores como el gráfico, el de sonido, o incluso físicas. Uno de los más famosos es Unreal Engine, siendo utilizado en multitud de videojuegos profesionales, indie o amateur, incluyendo proyectos de ingeniería.

Otro tipo de motor de videojuegos son los específicos para un género de videojuego, como el popular RPG Maker, que incluye una basta cantidad de elementos con los que poder empezar, como diseños artísticos, escenarios, o personajes.

## 2.5 Inteligencia Artificial

La inteligencia artificial es un factor determinante en el análisis de un videojuego. Si bien es cierto que la inteligencia artificial de un jugador virtual puede ser realmente competitiva, o incluso ser superior a cualquier jugador humano, lo cierto es que la inteligencia artificial que se implementa en un videojuego no hace uso de los más avanzados paradigmas de la inteligencia virtual, como pueden ser las redes neuronales artificiales, o los algoritmos genéticos.

Existe gran diferencia en este aspecto entre géneros de videojuegos ya que, por ejemplo, en ajedrez sí se utilizan módulos avanzados que implementan este tipo de paradigmas.

Sin embargo, en la gran mayoría de videojuegos el enfoque que se le da a la implementación de la IA es algorítmico, y esto es así por motivos de eficiencia, si bien esto podría cambiar en el futuro, y se prevé que cambie gracias a los modelos de juego en red o juego en la nube, ya que un ordenador con gran capacidad de cómputo en el caso del juego en la nube, podría ser capaz de jugar contra varios jugadores, tal como hacen los grandes jugadores de ajedrez en los eventos en los que juegan partidas contra muchos usuarios al mismo tiempo.

Otros juegos utilizan una variante matemática del enfoque algorítmico. Por ejemplo, en nuestro caso, podríamos implementar un algoritmo que generase distintos mapas (tableros) con densidades de probabilidad asociadas a la posibilidad de una casilla de un barco de encontrarse en una determinada casilla del tablero, dada la información obtenida hasta ese turno de la partida y la información de la entropía del sistema.



### 3 Tecnologías hardware y software utilizadas

En esta sección se detallan las tecnologías hardware y software utilizadas. El hardware se refiere a las placas utilizadas, no se especifican otros dispositivos utilizados, como pantallas, teclado o ratón.

#### 3.1 Hardware

El hardware es el conjunto de elementos físicos que componen un sistema informático. La base del proyecto es un microordenador de bajo coste y bajo consumo con soporte para sistema operativo y capacidad de expansión mediante otras placas electrónicas.

##### 3.1.1 Raspberry pi 3B

La placa base que se ha utilizado para el proyecto. Se trata de una placa base ampliamente conocida en todo el mundo, que se utiliza en universidades y otros centros de enseñanza para el estudio de la electrónica digital, la programación, o los sistemas operativos, entre otros ámbitos de la informática y la electrónica. Dada su popularidad, existen multitud de proyectos basados en ella, y existe una gran comunidad que colabora entre sí para mejorar y mantener los proyectos que utilizan esta placa. La especificación de la placa es la siguiente (Raspberry, <https://www.raspberrypi.org/>, 2021):

- CPU de 4 núcleos a 1,2 GHz Broadcom BCM2837 de 64 bits
- 1 GB de RAM
- BCM43438 LAN inalámbrica y Bluetooth de baja energía (BLE)
- 100 Ethernet base
- GPIO extendido de 40 pines
- 4 puertos USB 2
- Salida estéreo de 4 polos y puerto de video compuesto
- HDMI de tamaño completo

- Puerto de cámara CSI para conectar una cámara Raspberry Pi
- Puerto de pantalla DSI para conectar una pantalla táctil Raspberry Pi
- Puerto micro SD para cargar su sistema operativo y almacenar datos
- Fuente de alimentación micro USB conmutada mejorada hasta 2.5A

### 3.1.2 Sense Hat

Se trata de una placa que se puede añadir a la placa Raspberry Pi (un add-on) fabricada expresamente para la misión Astro Pi, pero que ahora está a la venta para los usuarios. La misión Astro Pi permite que estudiantes de todo el mundo puedan realizar programas que se ejecutarán en una placa que se encuentra en la Estación Espacial Internacional.

Contiene una matriz de LED 8x8 RGB, un joystick de 5 botones e incluye los siguientes sensores (Raspberry, <https://www.raspberrypi.org/>, 2021):

- Giroscopio
- Acelerómetro
- Magnetómetro
- Temperatura
- Presión barométrica
- Humedad

Todos estos sensores ofrecen una gran versatilidad para su uso en robótica, domótica, así como en física y otras ramas de las ciencias naturales.

### 3.1.3 Redes de computadores. Arquitecturas

La red de computadores y de Internet utilizada durante el desarrollo del proyecto para comunicar distintos equipos informáticos es una red local básica, la que se suele instalar por parte de las compañías de telecomunicaciones en los hogares que contratan sus servicios. Básicamente se trata de un router que proporciona una salida hacia otras

redes de Internet, y de un switch, que proporciona la capacidad de crear una red local interna.

## 3.2 Software

El software es la parte lógica que compone un sistema informático. Se diferencia entre varios tipos de niveles según su lectura sea más o menos comprensible para el hombre.

### 3.2.1 Sistemas Operativos. Raspbian, Windows, Ubuntu.

El sistema mínimamente soportado es Raspberry Pi OS (anteriormente llamado Raspbian), el sistema operativo que se suele utilizar en la placa Raspberry Pi. Se trata de una distribución de GNU/Linux basada en Debian, con modelo de desarrollo de código abierto (Software libre).

Dado que es posible instalar Windows 10 en la placa Raspberry Pi, se ha utilizado este sistema operativo durante la fase de construcción, y se le ha dado soporte en el sistema final.

Se ha utilizado otra distribución de GNU/Linux, Ubuntu, con objeto de realizar pruebas de compatibilidad en otros sistemas.

### 3.2.2 Lenguajes de programación. Python. El Shell.

El lenguaje de programación utilizado ha sido principalmente Python (Python Software Foundation, 2021). Se trata de un lenguaje de programación multiparadigma (soporta programación funcional, orientada a objetos, e imperativa), fuertemente tipado y de tipado dinámico, interpretado, y multiplataforma. Esta última propiedad es muy conveniente para el interés del proyecto. La filosofía del lenguaje es la claridad y legibilidad del código que permite escribir.

Python tiene licencia de código abierto, la Python Software Foundation License, administrada por la fundación Python Software Foundation.

Una de las razones más importantes para su elección era el hecho de tener experiencia previa con el lenguaje. Otra de las razones es que se trata del lenguaje más comúnmente utilizado en los proyectos realizados con la placa Raspberry Pi.

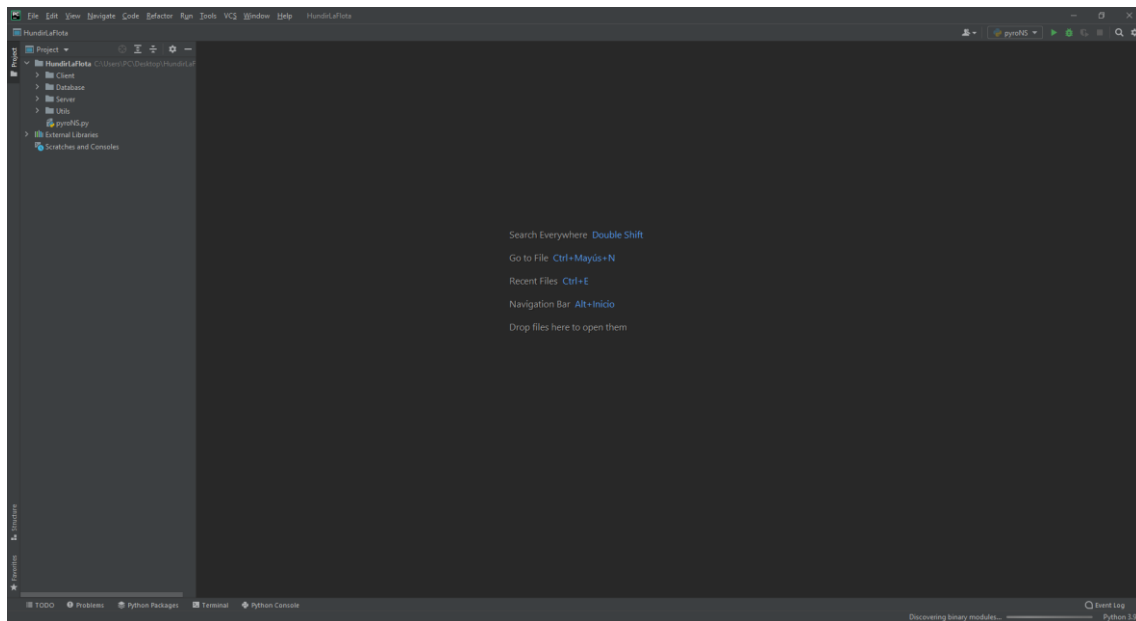
En todos los sistemas operativos utilizados se ha hecho uso con frecuencia del terminal (consola, en el caso de Windows) para tareas de programación y pruebas, mediante scripts de Shell cuando ha sido preciso.

### 3.2.3 Los IDE y los editores de código. PyCharm y Visual Studio Code

Un IDE (Integrated Development Environment) es una aplicación informática que facilita la tarea de programar, gestionando ficheros de un proyecto, librerías que debe importar, coloreado de texto, así como muchas otras funciones para depurar, compilar, interpretar o ejecutar código fuente.

Existen muchos IDE que ofrecen soporte para la programación con Python, aunque en este proyecto se utilizará principalmente PyCharm. PyCharm cuenta con útiles características para la gestión de un proyecto, como pueden ser la gestión integral de los archivos y directorios que forman parte de él, herramientas de asistencia para la programación, coloreado de la sintaxis del código, etc.

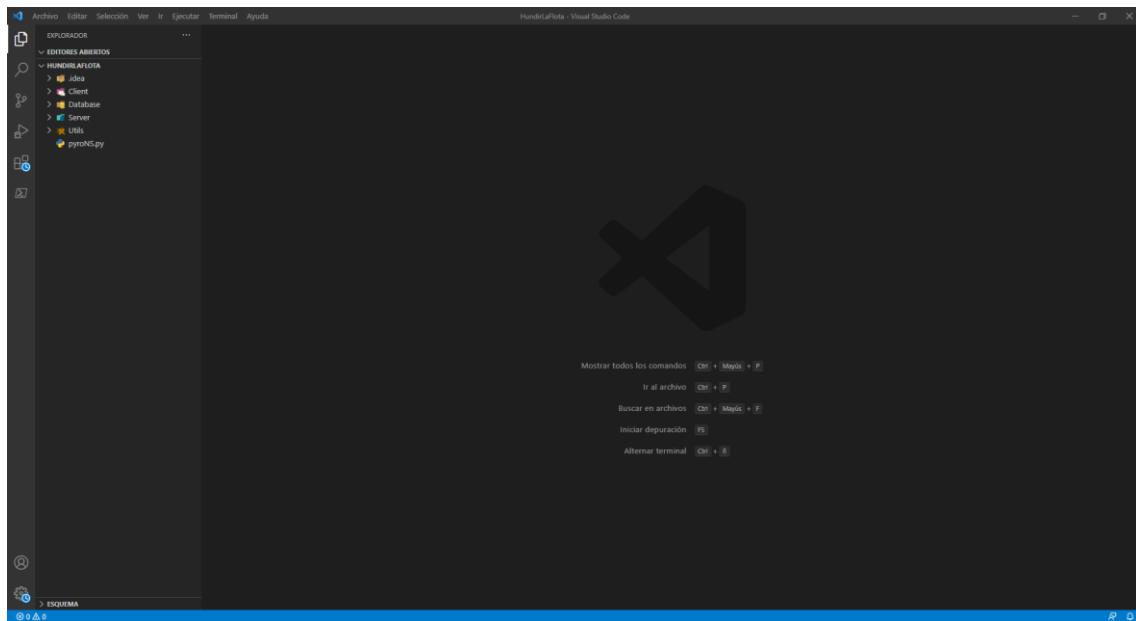




*Ilustración 1: Entorno integrado de desarrollo PyCharm*

Un editor de código fuente es una aplicación informática diseñada para editar ficheros de código fuente. Normalmente forman parte de un IDE, y suelen tener menos herramientas que un IDE promedio.

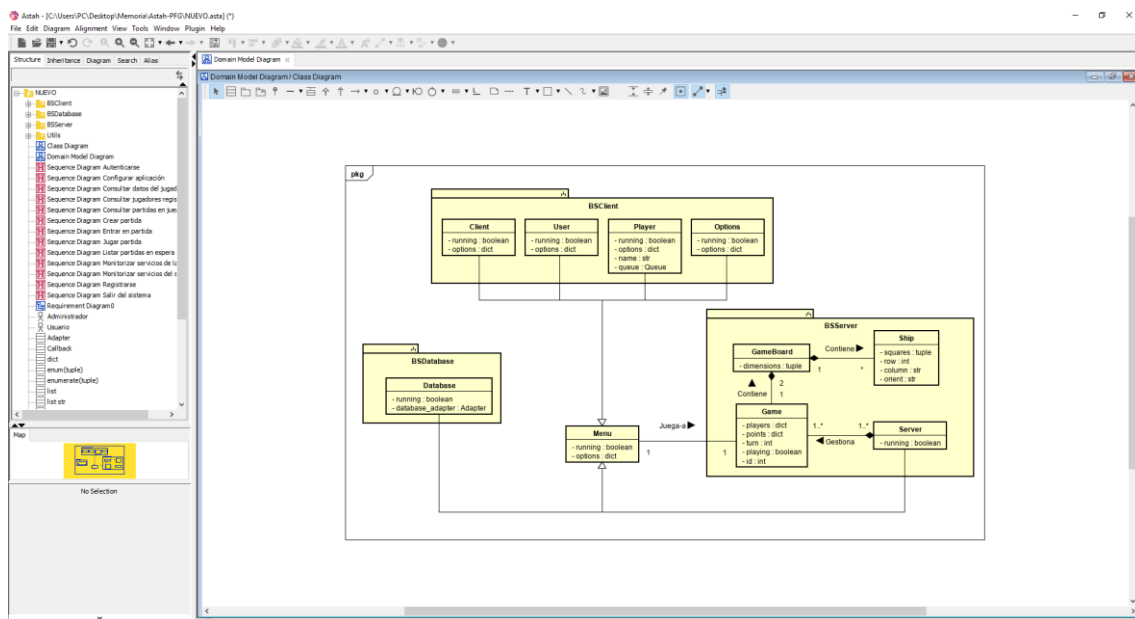
En este proyecto se hace uso del editor de código fuente Visual Code Studio. Sus características son su eficiencia en el uso de recursos unido a una gran cantidad de herramientas que facilitan la tarea de programar.



*Ilustración 2: Editor de código Visual Studio Code*

#### 3.2.4 Herramientas para diseño y modelado de software. Astah UML

Para la elaboración de los artefactos de diseño en cada una de las etapas del desarrollo del proyecto se ha utilizado Astah UML, una herramienta que permite modelar diversos diagramas que se usan en el diseño del software para la comunicación de ideas entre desarrolladores o entre clientes y desarrolladores.



*Ilustración 3: Herramienta de modelado Astah UML*

### 3.2.5 Pyro5. API middleware para sistemas distribuidos

Una API (Application Programming Interfaces) consiste en un conjunto de funciones, definiciones y protocolos necesarios para desarrollar aplicaciones que necesitan comunicación con otros módulos o librerías.

Pyro5 es un módulo de Python que será pieza fundamental en el proyecto. Esta librería proporciona elementos para poder implementar un sistema distribuido, sin tener que lidiar con los aspectos de su estructura a bajo nivel. A este tipo de software se le conoce como middleware. La siguiente es una lista de algunas de sus características (Jong, 2021):

- Escrito 100% en Python
- Funciona entre diferentes arquitecturas de sistemas y sistemas operativos.
- Capaz de comunicarse entre diferentes versiones de Python de forma transparente.
- Admite diferentes serializadores (serpent, json, marshal, msgpack).
- Puede utilizar sockets de dominio IPv4, IPv6 y Unix.

- Uso de conexiones seguras opcionales a través de SSL / TLS (cifrado, autenticación e integridad), incluida la validación del certificado en ambos extremos (SSL bidireccional).
- Servidor de nombres que realiza un seguimiento de las ubicaciones reales de sus objetos para que pueda moverlos de forma transparente.
- Soporte para reconexión automática a servidores en caso de interrupciones.
- Código de comunicación de red estable que ha funcionado de manera confiable en muchas plataformas durante más de una década.
- Confiable y establecido: construido sobre más de 20 años de historia existente de Pyro, con soporte y desarrollo continuos.

### 3.2.6 Las API para GUI y los motores para videojuegos. Tkinter y Pygame

Una GUI (Graphical User Interface) es un programa informático que establece una interfaz con el usuario, usualmente mediante ventanas, que permite una comunicación sencilla y amigable para el usuario con el sistema operativo.

Existen varias API para programar una GUI, como son wxPython, PyQt, PyGTK, etc. Nosotros elegiremos Tkinter, dado que es parte de la librería estándar de Python, es multiplataforma, y se cuenta con experiencia previa con ella.

Un motor de videojuegos (game engine), consiste en un conjunto de programas y herramientas orientadas al desarrollo de videojuegos.

En Python existen varios motores de juego, y en este proyecto se utilizará Pygame por el amplio soporte que tiene, por ser multiplataforma, y por ofrecer resultados de calidad.

El juego está pensado para ejecutarse en un entorno en 2 dimensiones.

### 3.2.7 API para bases de datos SQL. SQLite3

SQLite es un sistema gestor de bases de datos relacionales con licencia de dominio público. Se diferencia de otros sistemas gestores de bases de datos en que no se ejecuta como un proceso independiente al proceso principal que se comunica con él, sino que es enlazado con el mismo. Esto conlleva una menor latencia en el acceso a la base de datos. La base de datos se crea como un único archivo en la máquina host.

En Python existe el módulo `sqlite3`, que es una interfaz para SQLite, y que forma parte de la librería estándar, motivo principal de su elección frente a otras API y por contar con experiencia previa con el módulo.

### 3.2.8 Simulador de placa Sense Hat

Se trata de una página web que permite escribir código para la placa Sense Hat y ejecutar una simulación para comprobar los resultados del programa escrito (<https://trinket.io/sense-hat>, 2021).

### 3.2.9 Wireshark

Se trata de un analizador de tráfico de red. Se utiliza para analizar redes de comunicaciones, así como datos y protocolos.

Nosotros lo utilizaremos para capturar el tráfico en nuestra red y comprobar cómo funcionan los mecanismos de seguridad que implantaremos.

### 3.2.10 Seguridad. Protocolo SSL/TLS y certificados auto firmados

El uso de la librería `Pyro5` posibilita el uso de los protocolos SSL (Secure Socket Layer) y TLS (Transport Layer Security) para las comunicaciones entre los distintos equipos y servicios.

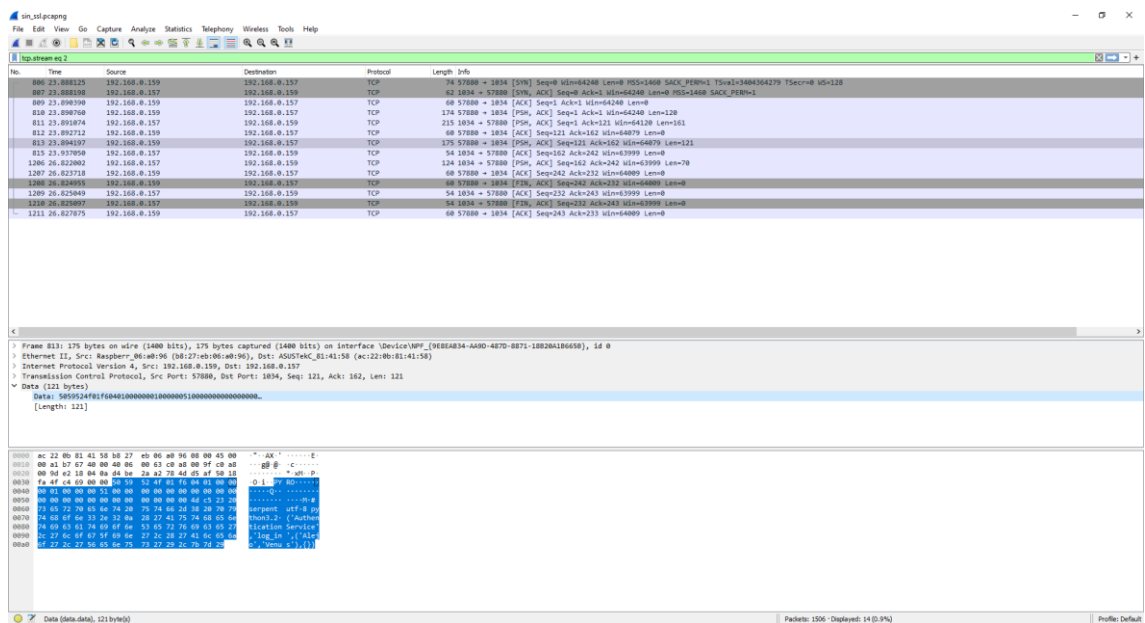
Ambos protocolos se utilizan para garantizar que la información transmitida por una red sea cifrada y segura, garantizando la integridad y privacidad de emisores y receptores. Para el cifrado de los datos se utilizan algoritmos de encriptación simétrica y asimétrica, pudiendo negociar las partes interesadas el algoritmo a utilizar.

Para poder utilizar este protocolo se tiene que contar con un certificado, que es el que verifica la identidad del equipo que lo contiene. Estos certificados se emiten por instituciones y empresas dedicadas a tal fin, y se consideran fiables los que estén firmados por ellas.

Aunque se puede crear un certificado para que sea firmado por una de estas instituciones, existe la opción de crear un certificado auto firmado, que no tiene validez al no estar firmado por una compañía de confianza, pero que puede ser utilizado para realizar pruebas de identificación y encriptar las comunicaciones.

En este proyecto se han utilizado certificados auto firmados, que tienen caducidad. Estos certificados han sido creados mediante el software openssl, que viene incluido en las distribuciones de Linux.

La siguiente captura se tomó con el software Wireshark, mostrando cómo es posible capturar una contraseña en el producto de desarrollo de este proyecto cuando no se utiliza el protocolo SSL:



*Ilustración 4: Captura de datos sin SSL*

Se puede observar cómo ha sido posible capturar el nombre y la contraseña de un usuario, pudiendo suplantar su identidad a partir de este momento.

Si configuramos los certificados y el uso del protocolo SSL para encriptar las comunicaciones entre los subsistemas de la aplicación y volvemos a analizar el tráfico de la red obtendremos lo siguiente:





## 4 Construcción de la solución

A lo largo de esta sección se detalla el modelo de proceso seguido durante el desarrollo del proyecto, así como las fases en las que se ha descompuesto.

### 4.1 Proceso de desarrollo software y modelos de proceso

El Proceso del software es la definición de un conjunto de actividades dirigidas a concluir con éxito el desarrollo de un producto software. Estas actividades se clasifican en estructurales y sombrilla.

Las actividades estructurales están relacionadas con la comunicación, planificación, modelado, construcción y despliegue de un producto.

Las actividades sombrilla complementan a las estructurales, y se centran en el seguimiento y control del proyecto, la administración del riesgo, el aseguramiento de la calidad, las revisiones técnicas, medición de parámetros del producto, administración de la configuración software, administración de la reutilización, además de la preparación y producción del producto de trabajo.

### 4.2 Modelos de proceso prescriptivo

Proponen un flujo predecible para el trabajo. Entre ellos se encuentran el modelo en cascada o el modelo en V.

El modelo en cascada es el modelo más antiguo de todos. Se basa en definir claramente los requisitos y cada etapa del proceso, para luego recorrer cada etapa secuencialmente, sin pasar a la siguiente etapa hasta no haber completado la anterior. Entre sus ventajas se encuentra su idoneidad para llevar a cabo un proyecto en el que los requisitos no cambian o están perfectamente claros desde el principio, y entre sus desventajas se encuentran que es complicado que un proyecto se ajuste realmente a una planificación secuencial, que habitualmente los requisitos son ambiguos dado que el

cliente puede tener dificultad para identificarlos o explicarlos, o que hasta la fase final del proyecto no se tendrá una versión funcional del producto.

El modelo en V es una variante del modelo en cascada en la que se diferencian dos etapas de desarrollo, una considerada descendente y que finaliza con una versión ejecutable del producto, y otra ascendente de pruebas para cada una de las anteriores fases descendentes.

#### 4.2.1 Modelo de proceso incremental

En este tipo de modelo se producen una serie de incrementos, que cada vez van incluyendo más funcionalidad. Estos incrementos se van entregando al cliente para que vaya probándolos. Su utilidad es mayor en casos en los que no se cuenta con suficiente personal para cumplir con los plazos de entrega, cuando existe urgencia por parte de los usuarios finales en contar con funcionalidad en etapas tempranas del desarrollo, o cuando los requisitos están bien definidos, pero no es posible aplicar un proceso lineal.

#### 4.2.2 Modelos de proceso evolutivo

El objetivo de estos modelos es desarrollar software de calidad de forma iterativa o incremental. Cuando no se puede utilizar un enfoque secuencial porque los requisitos cambian durante el desarrollo del producto, se puede utilizar este tipo de modelo. En él se va generando una versión final del producto en iteraciones sucesivas, que van aumentando la funcionalidad. Este enfoque permite flexibilidad y velocidad de desarrollo.

El modelo en espiral, el desarrollo de prototipos, o los modelos concurrentes, son variantes de los modelos de proceso evolutivos.

En el modelo de desarrollo de prototipos se utiliza cada prototipo para identificar requisitos del software, para ir aproximándose al sistema que se desea implementar. Algunas de sus desventajas del uso de prototipos son las malas decisiones en términos de

mantenibilidad del código, ya al tener la obligación de que el prototipo sea funcional, se toman ciertos compromisos que empeoran la calidad del producto a largo plazo.

El modelo en espiral produce una mezcla del modelo iterativo de prototipos junto con la secuencialidad del modelo en cascada. Consiste en dividir el proceso de desarrollo en actividades estructurales, y se planifican entregas evolutivas del producto, en cada una de las cuales se marcan puntos de referencia, combinando las fases del desarrollo con los productos elaborados. Este modelo puede utilizarse durante todo el desarrollo del producto, y es uno de los más realistas para el desarrollo de sistemas y software a gran escala. Permite una mejor reacción a los problemas que se dan durante el desarrollo.

Los modelos concurrentes permiten definir una red de eventos y actividades que existen concurrentemente. Un equipo software puede definir eventos y transiciones hacia otros eventos y gestionar eficientemente el estado actual del proyecto. Es aplicable a todo tipo de proyectos de desarrollo.

#### 4.2.3 El Proceso Unificado

El Proceso Unificado intenta fusionar las mejores características de los modelos del proceso del software, implementando principios de desarrollo ágil.

Utiliza un lenguaje de modelado, UML (Unified Modeling Language), que expresa de una forma potente el paradigma de la orientación a objetos.

Permite una orientación en el desarrollo en base a casos de uso, que son descripciones sobre interacciones del usuario con el sistema que proporcionen un valor observable para el usuario.

#### 4.2.4 Modelo de proceso personal

El modelo de proceso personal es el que aplica cada desarrollador para medir las características de su trabajo y de la calidad de éste. Define las actividades de planeación,

diseño de alto nivel, revisión del diseño de alto nivel, desarrollo y post mórtem. Se enfoca en registrar y analizar los fallos cometidos con objeto de desarrollar estrategias que lo eliminen.

#### 4.2.5 Elección de la metodología

Para el desarrollo del proyecto, se va a utilizar el Proceso Unificado, de forma iterativa y siguiendo un modelo de proceso personal. De esta forma se irán sucediendo iteraciones en las que se irán mejorando los artefactos UML que modelan la solución junto con el programa en cada caso de uso.

Se ha decidido así porque, aunque se tienen los requisitos desde el primer momento y esto habría aconsejado un modelo de desarrollo en cascada, se ha considerado como muy probable la aparición de nuevos requisitos, con lo que la elección final es una mezcla del enfoque de ambos modelos de desarrollo.

### 4.3 Fases del desarrollo de la aplicación

Se describen en las siguientes secciones las fases seguidas durante el desarrollo de la solución, con una breve introducción sobre los fundamentos utilizados en cada fase. Se incluyen los artefactos esenciales producidos en cada fase.

#### 4.3.1 Especificación de requisitos. Funcionales y no funcionales

Los requisitos para la aplicación se dividen en 2 tipos: requisitos funcionales, y requisitos no funcionales.

Los requisitos funcionales definen la funcionalidad del sistema, lo que el sistema debe hacer.

- Permitir a un usuario registrarse en el sistema
- Permitir a un usuario autenticarse en el sistema

- Permitir a un usuario autenticado consultar sus datos
- Permitir a un usuario autenticado crear una partida nueva contra otro usuario o contra la CPU
- Permitir a un usuario autenticado ver una lista de partidas iniciadas a la espera de contrincante
- Permitir a un usuario autenticado unirse a una partida creada a la espera de contrincante
- Permitir a un usuario autenticado cerrar la sesión activa
- Permitir a un administrador consultar los datos de los servicios del servidor
- Permitir a un administrador consultar el estado de las partidas en juego
- Permitir a un administrador consultar los datos de los servicios de la base de datos
- Permitir a un administrador consultar los datos de los usuarios registrados en la base de datos

Los requisitos no funcionales definen los atributos necesarios para el funcionamiento del sistema, que no tienen que ver directamente con su funcionalidad.

- Permitir una conexión segura entre los sistemas utilizando los protocolos TCP y SSL
- Se debe utilizar una base de datos para garantizar la permanencia de los datos de los usuarios
- Cuando un jugador se desconecte durante una partida, el servidor debe gestionar el final de la partida y cerrar la sesión del jugador
- El diseño debe separar la interfaz de usuario de la lógica de la aplicación
- El diseño debe ser extensible, pudiendo añadir distintas interfaces de usuario
- El diseño debe estar orientado a los datos, para mejorar la mantenibilidad
- El diseño debe permitir realizar test unitarios
- Debe usarse la placa Sense Hat para la comunicación con el usuario

#### 4.3.2 Análisis del problema. Diagrama de Casos de Uso

Sobre la especificación de requisitos se realizan diagramas de casos de uso, que reflejan acciones entre los usuarios y el sistema que devuelvan un determinado valor para el usuario.

En estos diagramas, se representan a la izquierda a los actores, que generalizan a usuarios, administradores, etc. Habitualmente con el símbolo de una persona, si bien estos actores pueden ser otros sistemas, en cuyo caso se situarán en la parte derecha del diagrama. El símbolo utilizado para denotar a un sistema es un círculo con un segmento a modo de base.

En el centro del diagrama se ubica una caja que representa al sistema, y dentro de él se ubican los casos de uso mediante elipses en las que se indica el nombre del caso de uso, normalmente una descripción corta de una acción. En los casos en los que el sistema cuente con varios módulos, estos se representan con el símbolo de una carpeta.

Cada actor se conecta mediante una línea con los casos de uso que tiene a su disposición, y si el caso de uso se conecta con otro actor o sistema, tendrá representadas las conexiones.

La opción <<include>> señala una opción obligatoria para el caso de uso del que procede, mientras que la opción <<extend>> indica una extensión opcional de un caso de uso hacia otro al que apunta. Aunque el uso de este tipo de notación no se recomienda de no ser estrictamente necesario, necesitamos indicar que el usuario necesita estar autenticado para realizar determinados casos de uso, indicando la situación mediante el uso de <<include>>. También necesitamos indicar que el usuario puede buscar una partida y finalmente no acabar accediendo a ella, o comenzar la creación de una partida, pero no acabar jugándola. Para estos dos últimos casos utilizaremos la opción <<extend>>.

La flecha que incide en el usuario desde el administrador indica que tienen una relación de generalización, ya que el administrador puede realizar todos los casos de uso propios y además todos los de un usuario.

Es importante reseñar lo que se considerará un caso de uso. Un caso de uso es una acción que devuelve un valor al usuario que lo efectúa. No se puede considerar un caso de uso una acción como manipular una ventana de la aplicación, o elegir la dificultad del juego contra la CPU. Por ese motivo los casos de uso se inspiran en los requisitos, tratando de encontrar el valor que proporciona cada acción al usuario en base al requisito que se busca satisfacer.

Partiendo de la información esquematizada en el diagrama, se construyen casos de uso formales, que son descripciones formales del flujo de acciones en cada caso de uso. Aunque la definición sea formal, debe evitar el lenguaje técnico, ya que estos artefactos sirven para comunicar al cliente información sobre la funcionalidad evaluada a partir de los requisitos.

The diagram illustrates the functional requirements of the BattleShip system, organized into three main components: Client, Server, and Database.

- Actors:**
  - Usuario (User):** Interacts with the Client use cases.
  - Administrador (Administrator):** Interacts with the Server and Database use cases.
- Client Use Cases:**
  - Registrar
  - Salir del sistema
  - Consultar datos del jugador
  - Crear partida
  - Listar partidas en espera
  - Entrar en partida
  - Configurar aplicación
  - Autenticarse
  - Jugar partida
- Server Use Cases:**
  - Monitorizar servicios del servidor
  - Consultar partidas en juego
- Database Use Cases:**
  - Monitorizar servicios de la BD
  - Consultar jugadores registrados
- External Services:**
  - Servicio de datos:** Interacts with Registrar, Consultar datos del jugador, Consultar partidas en juego, Consultar jugadores registrados, and Jugar partida.
  - Servicio de autenticación:** Interacts with Autenticarse.
  - Servicio gestor:** Interacts with Crear partida, Listar partidas en espera, Entrar en partida, and Monitorizar servicios del servidor.
  - Servicio Callback Jugador:** Interacts with Jugar partida.
- Relationships:**
  - Include Relationships (dashed lines with open arrowheads):**
    - Salir del sistema includes Registrar.
    - Consultar datos del jugador includes Autenticarse.
    - Crear partida includes Autenticarse.
    - Listar partidas en espera includes Autenticarse.
    - Entrar en partida includes Autenticarse.
    - Configurar aplicación includes Autenticarse.
    - Jugar partida includes Autenticarse.
  - Extend Relationships (dashed lines with open arrowheads):**
    - Autenticarse extends Jugar partida.
    - Consultar datos del jugador extends Jugar partida.
    - Crear partida extends Jugar partida.
    - Listar partidas en espera extends Jugar partida.
    - Entrar en partida extends Jugar partida.
    - Configurar aplicación extends Jugar partida.

34



Se muestran ahora las descripciones de los casos de uso, en formato esencial y a doble columna:

|   |  |
|---|--|
| <b>Caso de uso</b>  | Registrarse  |
| <i>Formato completo (variante 'a dos columnas'), estilo esencial</i>  |  |
| <b>Evolución típica de los acontecimientos</b>  |  |
| <b>Acciones del actor</b>   | <b>Respuesta del sistema</b>   |
| 1. El caso de uso comienza cuando el usuario selecciona la operación “Registrarse”.   | 2. El sistema solicita los datos necesarios para el registro: nombre y contraseña. |
| 3. El usuario proporciona los datos.  | 4. El sistema verifica los datos.  |
|   | 5. El sistema informa al usuario del resultado.                                    |
| 6. El cliente confirma el resultado.  | 7. El sistema vuelve al menú principal de usuario.                                 |
| <b>Alternativas</b>   |  |
| 2. El servidor no está disponible.<br>2.1. El sistema informa al usuario de que el servidor no está disponible.<br>4. El nombre ya está registrado.<br>4.1. El sistema informa al usuario de que no se ha podido realizar la operación.<br>4.2. El usuario confirma el resultado.<br>4.3. El sistema vuelve al menú principal de usuario. |  |

|   |   |
|---|---|
| <b>Caso de uso</b>  | Autenticarse  |
| <i>Formato completo (variante 'a dos columnas'), estilo esencial</i>          |   |
| <b>Evolución típica de los acontecimientos</b>                                |   |
| <b>Acciones del actor</b>   | <b>Respuesta del sistema</b>  |
| 1. El caso de uso comienza cuando el usuario selecciona la operación “Login”. | 2. El sistema solicita los datos necesarios para la autenticación: nombre y contraseña. |
| 3. El usuario proporciona los datos.  | 4. El sistema verifica los datos.   |
|   | 5. El sistema registra al usuario como autenticado.                                     |
|   | 6. El sistema informa al usuario del resultado.   |
| 7. El usuario confirma el resultado.  | 8. El sistema vuelve al menú principal de usuario.                                      |

|  |
|--|
| <b>Alternativas</b>  |
| 2. El servidor no está disponible.<br>2.1. El sistema informa al usuario de que el servidor no está disponible.<br>4. Existe un problema con los datos.<br>4.1. El nombre ya está autenticado.<br>4.1.1. El sistema informa al usuario de que no se ha podido realizar la operación.<br>4.1.2. El usuario confirma el resultado.<br>4.1.3. El sistema vuelve al menú principal de usuario.<br>4.2. El usuario no existe en el sistema.<br>4.2.1. El sistema informa al usuario de que no se ha podido realizar la operación.<br>4.2.2. El usuario confirma el resultado.<br>4.2.3. El sistema vuelve al menú principal de usuario. |

|  |   |
|--|---|
| <b>Caso de uso</b>   | Salir del sistema   |
| <i>Formato completo (variante 'a dos columnas'), estilo esencial</i>   |   |
| <b>Evolución típica de los acontecimientos</b>   |   |
| <b>Acciones del actor</b>  | <b>Respuesta del sistema</b>                                    |
| 1. El caso de uso comienza cuando el usuario selecciona la operación "Logout".   | 2. El sistema elimina al usuario del registro como autenticado. |
|  | 3. El sistema vuelve al menú principal de usuario.              |
| <b>Alternativas</b>  |   |
| 1. En cualquier momento el usuario cierra la ventana de su programa.<br>1.1. El sistema elimina al usuario del registro como autenticado.<br>2. El servidor no está disponible.<br>2.1. El sistema informa al usuario de que el servidor no está disponible. |   |

|  |   |
|--|---|
| <b>Caso de uso</b>   | Consultar datos del jugador   |
| <i>Formato completo (variante 'a dos columnas'), estilo esencial</i>   |   |
| <b>Evolución típica de los acontecimientos</b>   |   |
| <b>Acciones del actor</b>  | <b>Respuesta del sistema</b>  |
| 1. El caso de uso comienza cuando el usuario autenticado selecciona la operación "Ver puntuación histórica". | 2. El sistema presenta al usuario un resumen con los datos del jugador. |

|  |  |
|--|--|
| 3. El usuario confirma el resultado.   | 4. El sistema vuelve al menú de usuario autenticado. |
| <b>Alternativas</b>  |  |
| 1. En cualquier momento el usuario cierra la ventana de su programa.<br>1.1. El sistema solicita elimina al usuario del registro como autenticado.<br>2. Existe un problema en el servidor o la base de datos.<br>2.1. El servidor no está disponible.<br>2.1.1. El sistema informa al usuario de que el servidor no está disponible.<br>2.2. La base de datos no está disponible.<br>2.2.1. El sistema informa al usuario de que la base de datos no está disponible. |  |

|  |   |
|--|---|
| <b>Caso de uso</b>   | Crear partida   |
| <i>Formato completo (variante 'a dos columnas'), estilo esencial</i>   |   |
| <b>Evolución típica de los acontecimientos</b>   |   |
| <b>Acciones del actor</b>  | <b>Respuesta del sistema</b>  |
| 1. El caso de uso comienza cuando el usuario autenticado selecciona la operación “Crear partida”.  | 2. El sistema solicita al usuario los datos necesarios para la operación: número de barcos, tamaño de cada barco, y tipo de rival (humano o CPU). |
| 3. El usuario proporciona los datos.   | 4. El sistema verifica los datos, crea una partida e introduce en ella al usuario.  |
|  | 5. El sistema registra la partida como partida en espera.   |
|  | 6. El sistema informa al usuario del resultado.   |
| <b>Alternativas</b>  |   |
| 1. En cualquier momento el usuario cierra la ventana de su programa.<br>1.1. El sistema elimina al usuario del registro como autenticado.<br>2. El servidor no está disponible.<br>2.1. El sistema informa al usuario de que el servidor no está disponible.<br>3. El usuario cancela la operación<br>3.1. El sistema vuelve al menú de usuario autenticado. |   |

|  |                           |
|--|---------------------------|
| <b>Caso de uso</b>   | Listar partidas en espera |
| <i>Formato completo (variante 'a dos columnas'), estilo esencial</i> |                           |

| Evolución típica de los acontecimientos  |  |
|--|--|
| Acciones del actor   | Respuesta del sistema  |
| 1. El caso de uso comienza cuando el usuario autenticado selecciona la operación “Ver partidas a la espera de contrincante”.   | 2. El sistema presenta al usuario un resumen con las partidas creadas a la espera de contrincante. |
| 3. El usuario confirma el resultado.   | 4. El sistema vuelve al menú de usuario autenticado.   |
| <b>Alternativas</b>  |  |
| 1. En cualquier momento el usuario cierra la ventana de su programa.<br>1.1. El sistema elimina al usuario del registro como autenticado.<br>2. El servidor no está disponible.<br>2.1. El sistema informa al usuario de que el servidor no está disponible. |  |

| <b>Caso de uso</b>  | Entrar en partida  |
|---|--|
| <i>Formato completo (variante ‘a dos columnas’), estilo esencial</i>  |  |
| Evolución típica de los acontecimientos   |  |
| Acciones del actor  | Respuesta del sistema  |
| 1. El caso de uso comienza cuando el usuario autenticado selecciona la operación “Unirse a una partida creada”.   | 2. El sistema solicita al usuario el identificador de la partida.  |
| 3. El usuario proporciona el identificador.   | 4. El sistema verifica los datos, elimina la partida del registro de partidas a la espera de contrincante y registra la partida como partida en juego. |
|   | 5. El sistema introduce en la partida al usuario e informa al usuario del resultado.   |
| <b>Alternativas</b>   |  |
| 1. En cualquier momento el usuario cierra la ventana de su programa.<br>1.1. El sistema solicita elimina al usuario del registro como autenticado.<br>2. El servidor no está disponible.<br>2.1. El sistema informa al usuario de que el servidor no está disponible.<br>3. El usuario cancela la operación<br>3.1. El sistema vuelve al menú de usuario autenticado. |  |

|  |   |
|--|---|
| <b>Caso de uso</b>   | Jugar partida   |
| <i>Formato completo (variante 'a dos columnas'), estilo esencial</i>   |   |
| <b>Evolución típica de los acontecimientos</b>   |   |
| <b>Acciones del actor</b>  | <b>Respuesta del sistema</b>  |
| 1. El caso de uso comienza cuando el usuario autenticado ha seleccionado la operación "Crear partida".   | 2. El sistema notifica al usuario que el juego ha comenzado.  |
|  | 3. El sistema solicita al usuario que introduzca los datos necesarios para la operación: la posición de sus barcos. |
| 4. El usuario proporciona los datos.   | 5. El sistema almacena la información de los barcos en la partida creada.   |
|  | 6. El sistema solicita al usuario los datos necesarios para la operación: la posición del disparo.                  |
| 7. El usuario proporciona los datos.   | 8. El sistema verifica los datos e informa al usuario del resultado del disparo.                                    |
|  | 9. El sistema informa al usuario de que debe esperar el turno de disparo del rival.                                 |
|  | 10. El sistema verifica el resultado del disparo del rival y notifica el resultado al usuario.                      |
| Se repiten los pasos 6 a 10 hasta acabar la partida.   |   |
|  | 11. El sistema actualiza la puntuación del usuario.   |
|  | 12. El sistema notifica al usuario el resultado de la partida.  |
| 13. El usuario confirma el resultado.  | 14. El sistema vuelve al menú de usuario autenticado.   |
| <b>Alternativas</b>  |   |
| 1. En cualquier momento el usuario cierra la ventana de su programa.<br>1.1. El sistema solicita elimina al usuario del registro como autenticado. |   |
| 1. El usuario seleccionó la operación "Unirse a una partida creada".<br>1.1. El sistema notifica al usuario que el juego ha comenzado.             |   |

|   |
|---|
| 1.2. El sistema solicita al usuario que introduzca los datos necesarios para la operación: la posición de sus barcos.<br>1.3. El usuario proporciona los datos.<br>1.4. El sistema almacena la información de los barcos en la partida creada.<br>1.5. Continúa a partir del paso 9.<br>2. El servidor no está disponible.<br>2.1. El sistema informa al usuario de que el servidor no está disponible. |
|---|

|  |   |
|--|---|
| <b>Caso de uso</b>   | Configurar aplicación   |
| <i>Formato completo (variante 'a dos columnas'), estilo esencial</i>   |   |
| <b>Evolución típica de los acontecimientos</b>   |   |
| <b>Acciones del actor</b>  | <b>Respuesta del sistema</b>  |
| 1. El caso de uso comienza cuando el usuario autenticado selecciona la operación “Cambiar opciones”.   | 2. El sistema muestra al usuario las opciones que puede cambiar.  |
|  | 3. El sistema solicita al usuario los datos necesarios para la operación: la opción a cambiar y el nuevo valor. |
| 4. El usuario proporciona los datos.   | 5. El sistema verifica los datos y realiza los cambios en la configuración.                                     |
|  | 6. El sistema vuelve al menú de usuario autenticado.  |
| <b>Alternativas</b>  |   |
| 1. En cualquier momento el usuario cierra la ventana de su programa.<br>1.1. El sistema solicita elimina al usuario del registro como autenticado.<br>2. El usuario cancela la operación<br>2.1. El sistema vuelve al menú de usuario autenticado. |   |

|  |   |
|--|---|
| <b>Caso de uso</b>   | Monitorizar servicios del servidor  |
| <i>Formato completo (variante 'a dos columnas'), estilo esencial</i>                                       |   |
| <b>Evolución típica de los acontecimientos</b>   |   |
| <b>Acciones del actor</b>  | <b>Respuesta del sistema</b>  |
| 1. El caso de uso comienza cuando el administrador selecciona la operación “Ver información del servidor”. | 2. El sistema presenta al administrador un resumen con la información del servidor. |

|   |  |
|---|--|
| 3. El administrador confirma el resultado.  | 4. El sistema vuelve al menú principal de administrador. |
| <b>Alternativas</b>   |  |
| 2. El servidor no está disponible.<br>2.1. El sistema informa al administrador de que el servidor no está disponible. |  |

|   |  |
|---|--|
| <b>Caso de uso</b>  | Consultar partidas en juego  |
| <i>Formato completo (variante 'a dos columnas'), estilo esencial</i>  |  |
| <b>Evolución típica de los acontecimientos</b>  |  |
| <b>Acciones del actor</b>   | <b>Respuesta del sistema</b>   |
| 1. El caso de uso comienza cuando el administrador selecciona la operación “Ver partidas en juego”.                   | 2. El sistema presenta el administrador un resumen con el estado de las partidas en juego. |
| 3. El administrador confirma el resultado.  | 4. El sistema vuelve al menú principal de administrador.                                   |
| <b>Alternativas</b>   |  |
| 2. El servidor no está disponible.<br>2.1. El sistema informa al administrador de que el servidor no está disponible. |  |

|   |  |
|---|--|
| <b>Caso de uso</b>  | Monitorizar servicios de la BD   |
| <i>Formato completo (variante 'a dos columnas'), estilo esencial</i>  |  |
| <b>Evolución típica de los acontecimientos</b>  |  |
| <b>Acciones del actor</b>   | <b>Respuesta del sistema</b>   |
| 1. El caso de uso comienza cuando el administrador selecciona la operación “Ver información de la base de datos”.               | 2. El sistema presenta al administrador un resumen con la información de la base de datos. |
| 3. El administrador confirma el resultado.  | 4. El sistema vuelve al menú principal de administrador.                                   |
| <b>Alternativas</b>   |  |
| 2. La base de datos no está disponible.<br>2.1. El sistema informa al administrador de que la base de datos no está disponible. |  |

|   |  |
|---|--|
| <b>Caso de uso</b>  | Consultar jugadores registrados  |
| <i>Formato completo (variante 'a dos columnas'), estilo esencial</i>  |  |
| <b>Evolución típica de los acontecimientos</b>  |  |
| <b>Acciones del actor</b>   | <b>Respuesta del sistema</b>   |
| 1. El caso de uso comienza cuando el administrador selecciona la operación “Ver jugadores registrados”.                         | 2. El sistema presenta al administrador un resumen de los jugadores registrados. |
| 3. El administrador confirma el resultado.  | 4. El sistema vuelve al menú principal de administrador.                         |
| <b>Alternativas</b>   |  |
| 2. La base de datos no está disponible.<br>2.1. El sistema informa al administrador de que la base de datos no está disponible. |  |

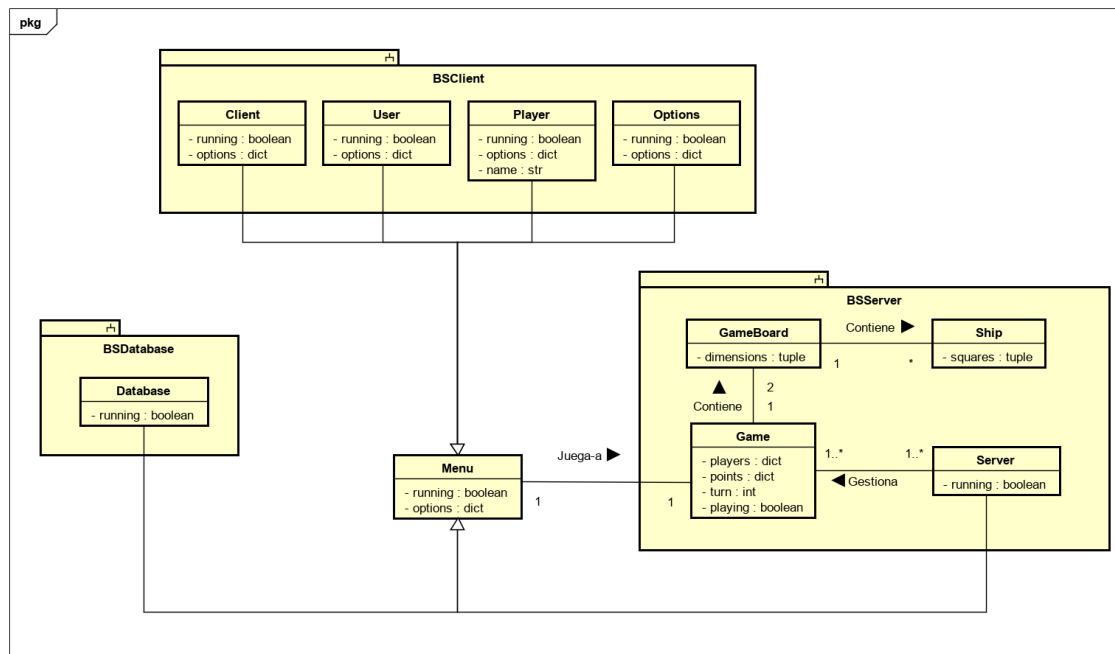
#### 4.3.3 Modelado de datos. Diagrama de Modelo de Dominio

Mediante los diagramas de modelo de dominio se representa el modelo de datos que va a utilizar la aplicación.

Esta es una visión estática de la aplicación y define las entidades pertenecientes al mundo en el que vive la aplicación, sin prestar atención a objetos software puros, como puede ser una base de datos. Se definen sólo los atributos de las entidades que luego inspirarán las clases del diseño orientado a objetos. El modelado del comportamiento de las clases inspiradas en este diagrama se perfilará en siguientes artefactos del proceso.



El siguiente es el diagrama de modelo de dominio de la aplicación:



*Ilustración 7: Diagrama de modelo de dominio*

Aunque aparecen nombres como Server o Database, se trata de objetos conceptuales, que son especializaciones del objeto Menu. De esta forma se representan las diferentes pantallas del videojuego, modelando en ellas un atributo para gestionar su puesta en marcha y parada (running), y otro atributo que representará las distintas opciones que incluya el menú. Un caso particular es el del objeto que modela a los jugadores (Player), que contiene además un atributo que modela el identificador del jugador.

Se ha decidido también la estructura que tendrán las partidas (Game), cuyos atributos son los nombres y puntuación de los jugadores, además del turno y un atributo que indica si se está jugando la partida (playing).

Las partidas contienen tableros (GameBoard), que almacenan las dimensiones de la matriz de casillas.

Los tableros contienen barcos (Ship), que almacenan las casillas que ocupa un barco en particular.

#### 4.3.4 Diseño de la solución. Diagramas de Interacción y Diagrama de Clases

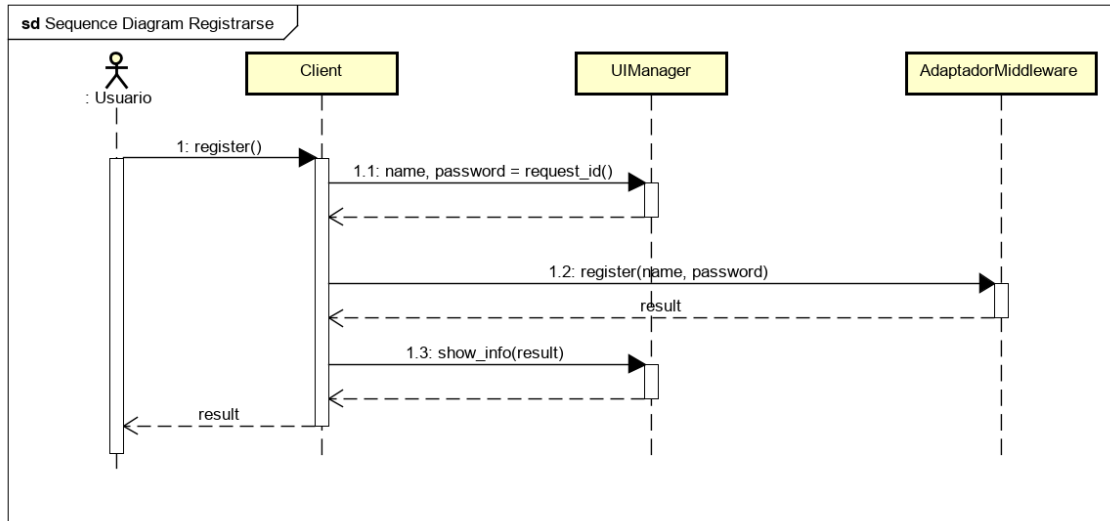
En esta sección se presentan los diagramas de interacción y los diagramas de clases.

Los diagramas de interacción aportan una visión dinámica de la funcionalidad requerida. Modelan interacciones entre objetos del modelo de dominio reflejando las responsabilidades en la creación de cada objeto por parte de cada objeto. Se utilizan para modelar los métodos que finalmente tendrán las clases de la aplicación.

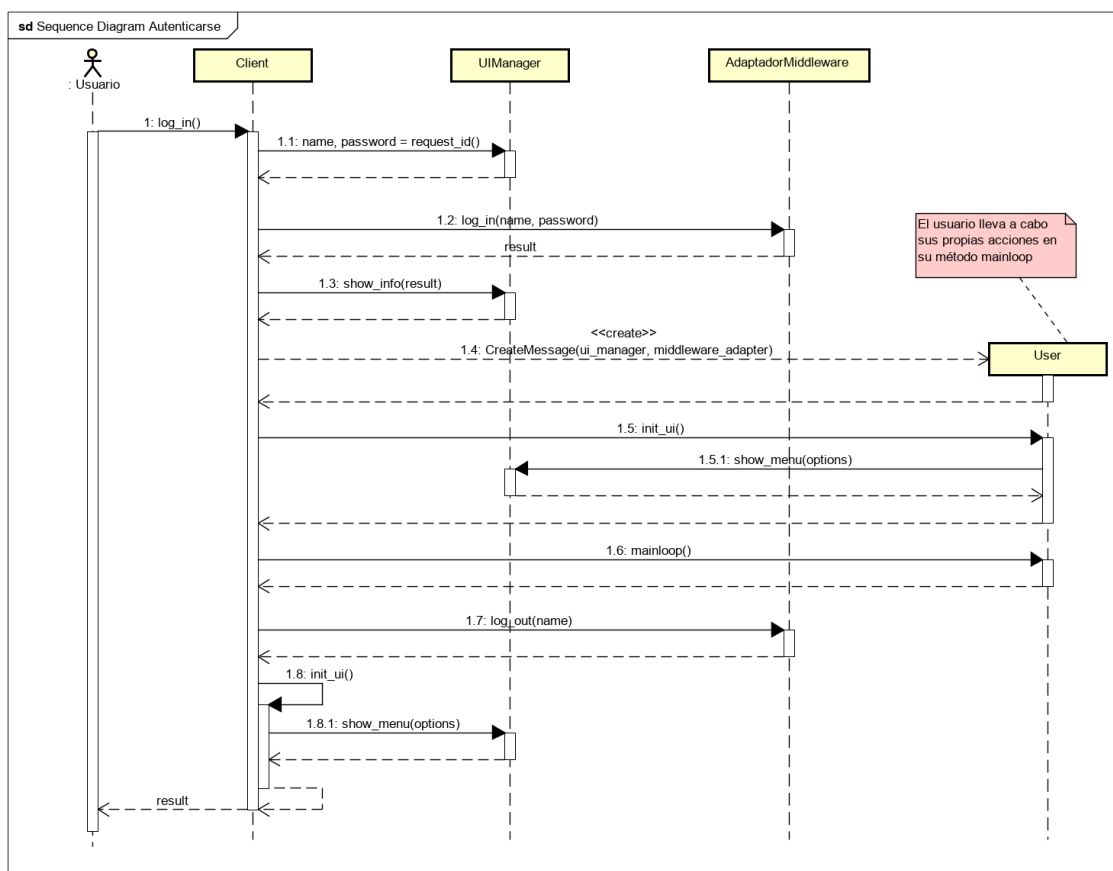
Existen de dos tipos, diagramas de colaboración y diagramas de secuencia. Ambos son equivalentes, teniendo los diagramas de colaboración una dimensión más compacta, mientras que los de secuencia modelan con más detalle las interacciones respecto al tiempo en el flujo de ejecución. Nosotros utilizaremos los diagramas de secuencia.

En estos diagramas se hace uso de una línea temporal de vida de cada objeto, y se representan los mensajes (llamadas a métodos) entre ellos. En los mensajes se indican los parámetros con los que se llama al método, y se suele indicar el tipo de éstos. La representación de un mensaje es una flecha dirigida hacia la línea de vida del objeto al que se envía el mensaje. La representación de los mensajes de retorno se indica con una flecha desde el objeto llamado al objeto que realizó la llamada, indicando el valor retornado.

Por cada caso de uso se realizará un diagrama de secuencia:



*Ilustración 8: Diagrama de secuencia del CU Registrarse*



*Ilustración 9: Diagrama de secuencia del CU Autenticarse*

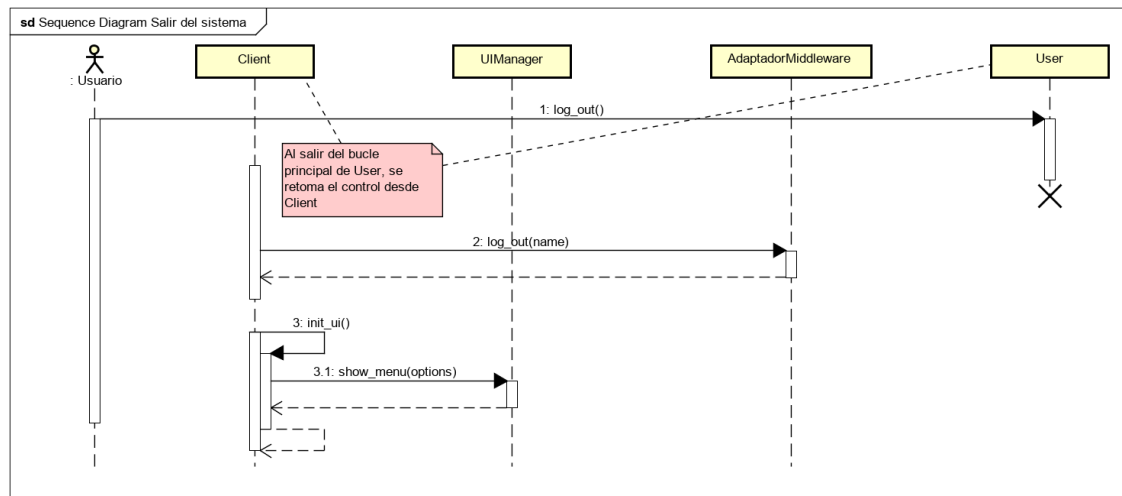


Ilustración 10: Diagrama de secuencia del CU Salir del sistema

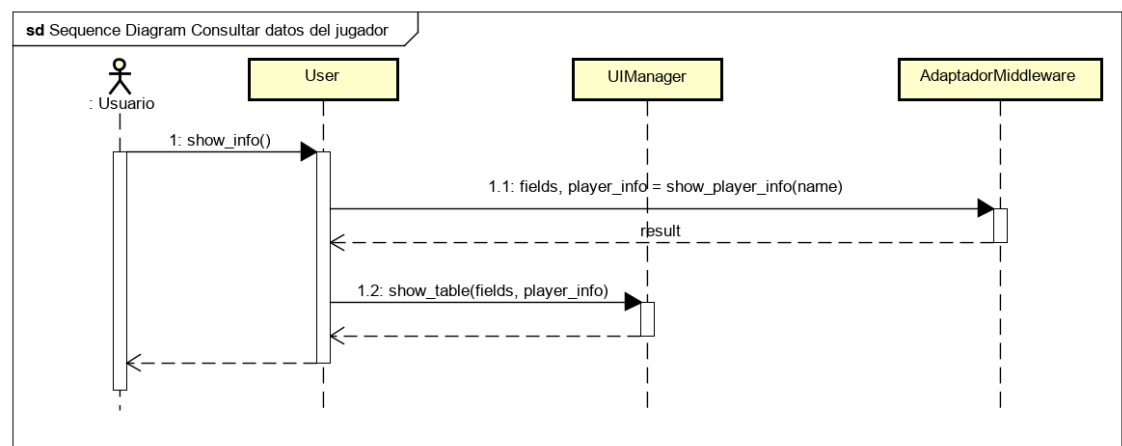


Ilustración 11: Diagrama de secuencia del CU Consultar datos del jugador

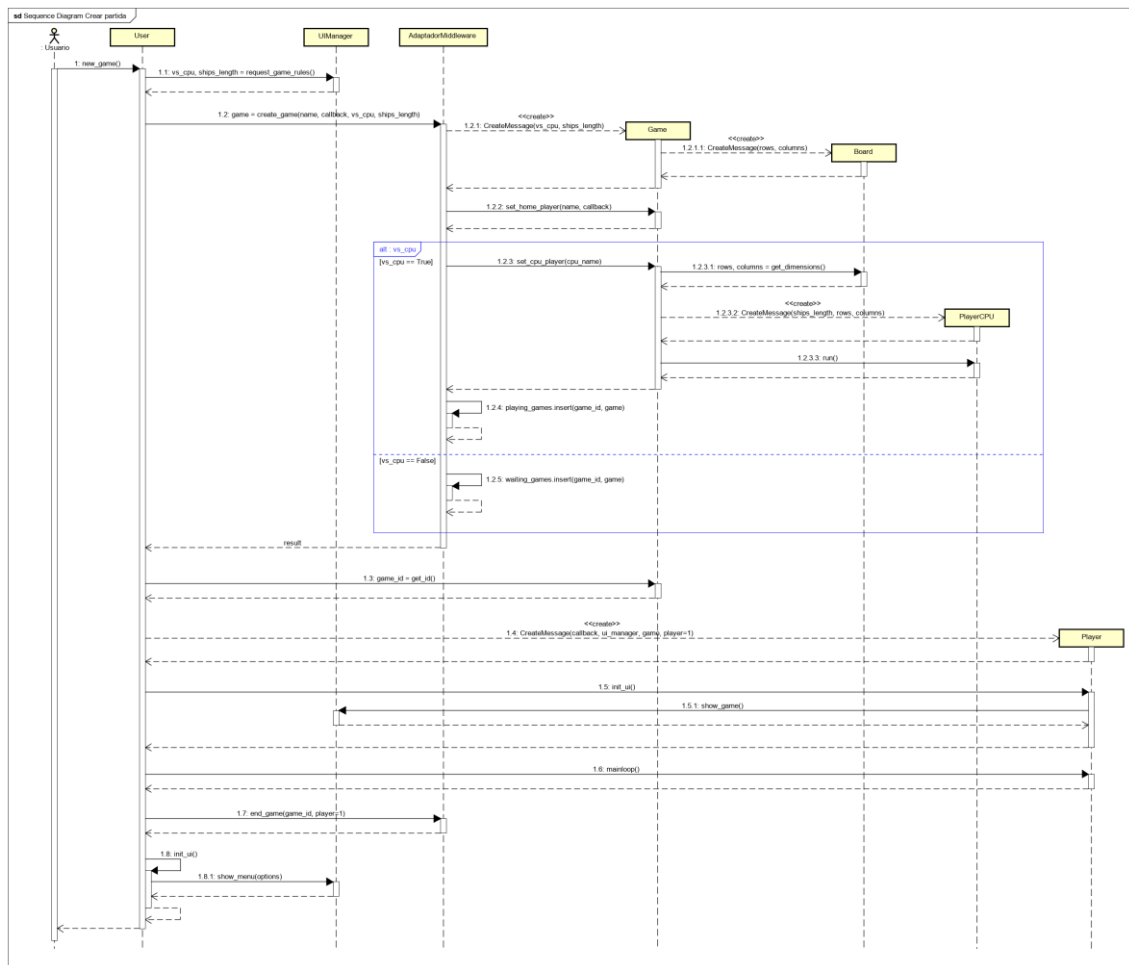


Ilustración 12: Diagrama de secuencia del CU Crear partida

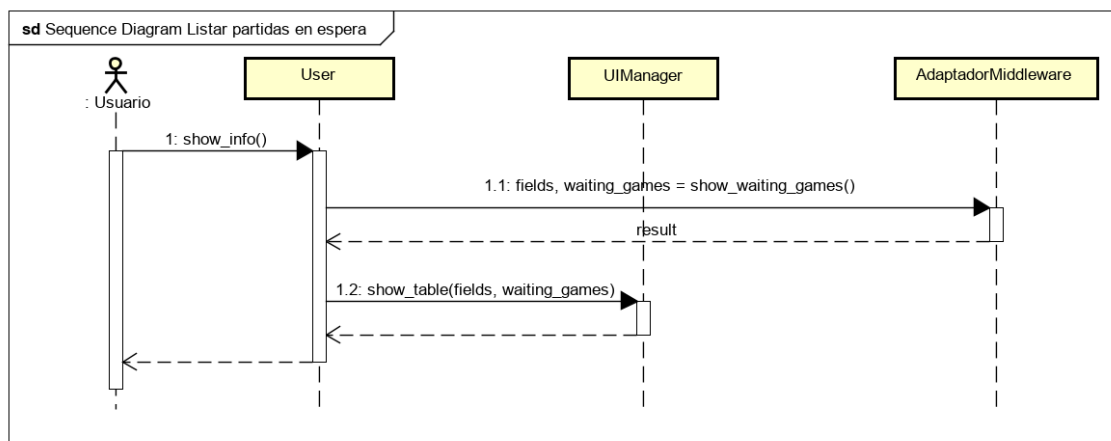


Ilustración 13: Diagrama de secuencia del CU Listar partidas en espera

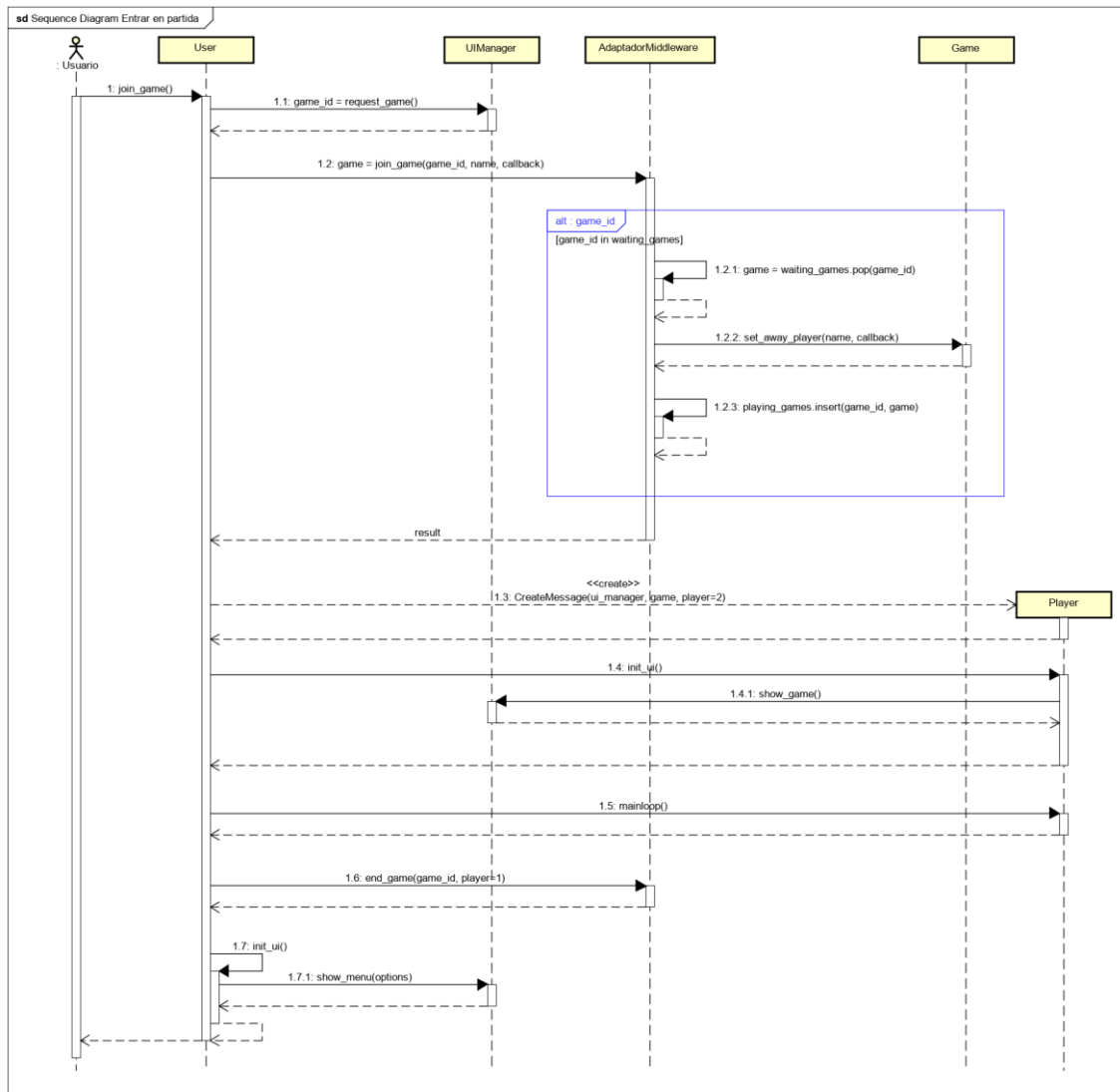


Ilustración 14: Diagrama de secuencia del CU Entrar en partida

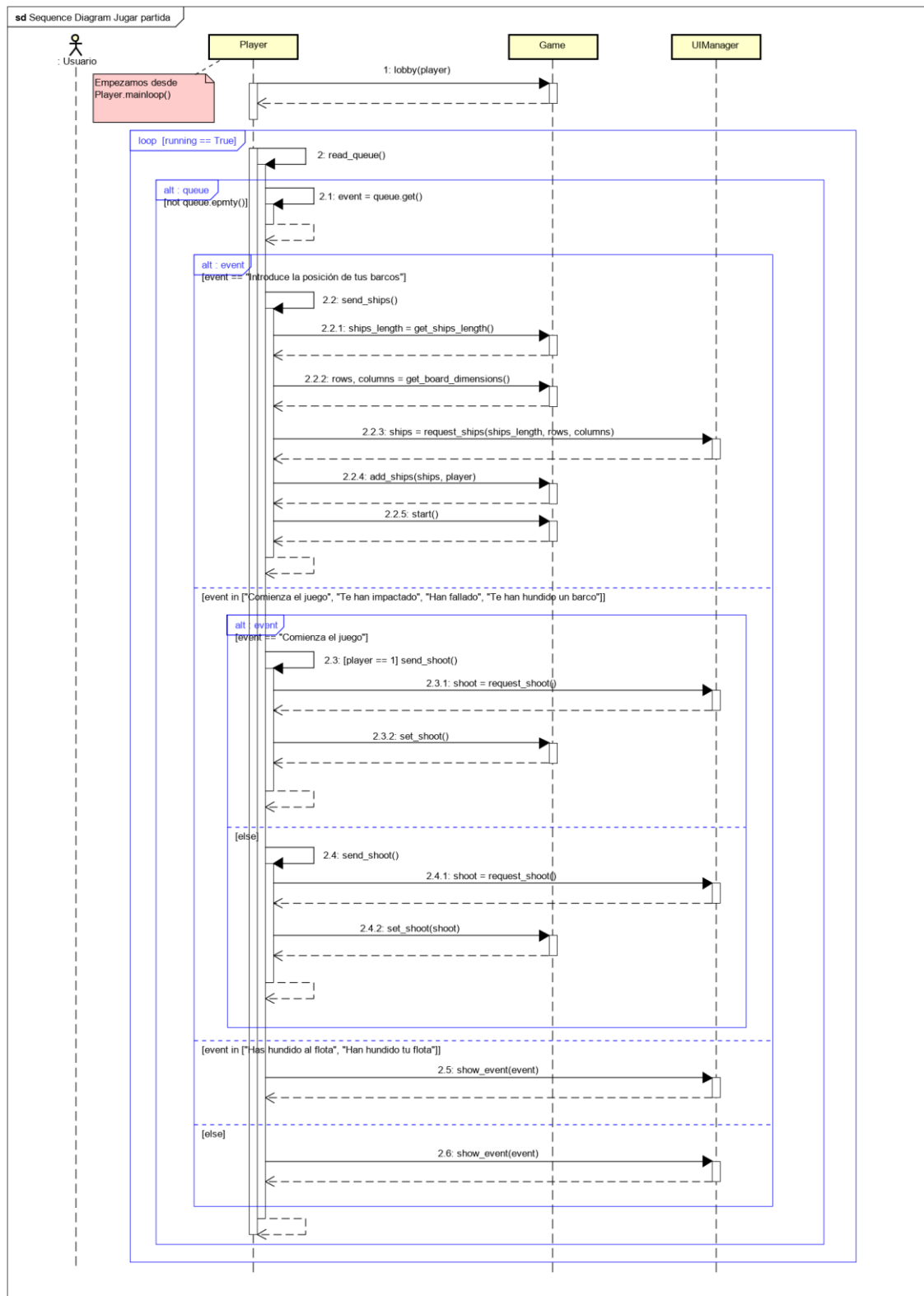


Ilustración 15: Diagrama de secuencia del CU Jugar partida

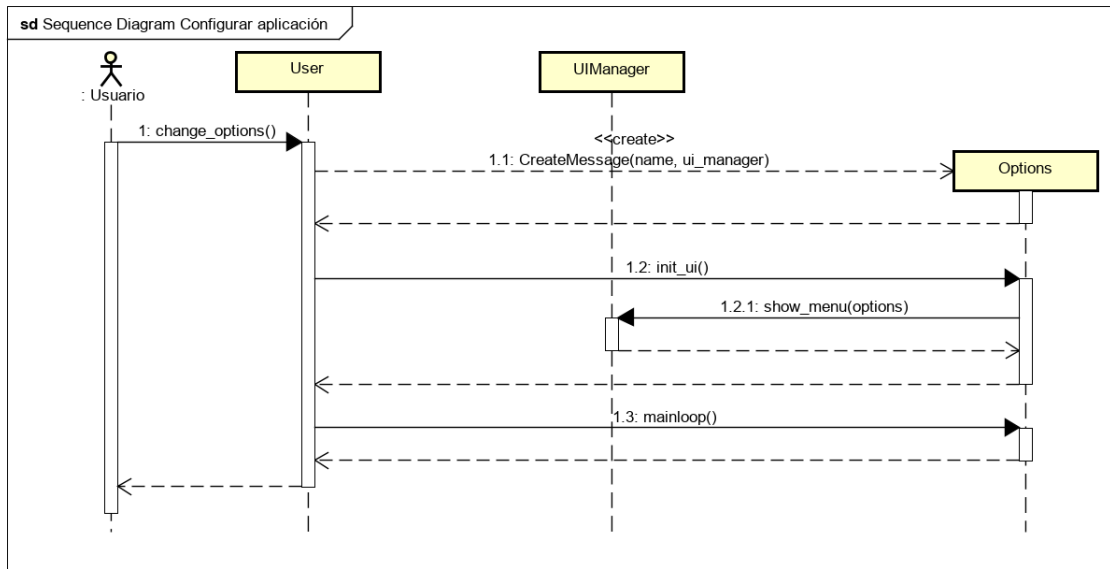


Ilustración 16: Diagrama de secuencia del CU Configurar aplicación

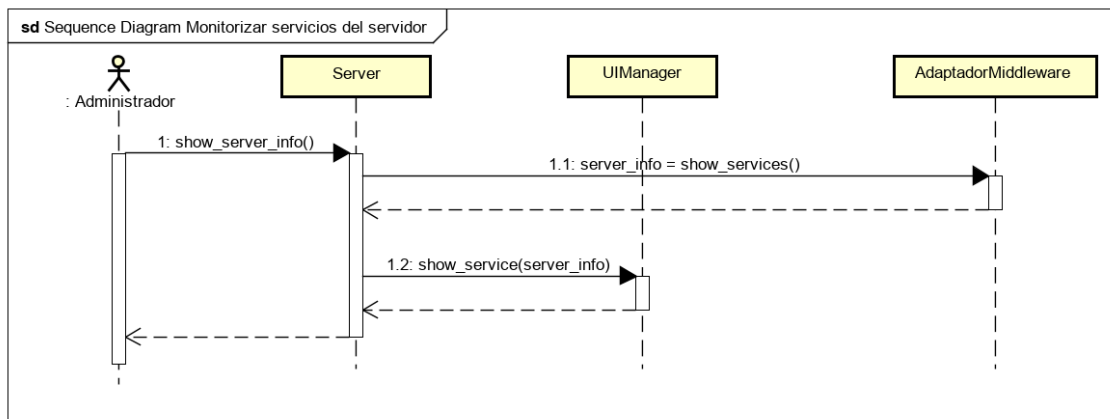
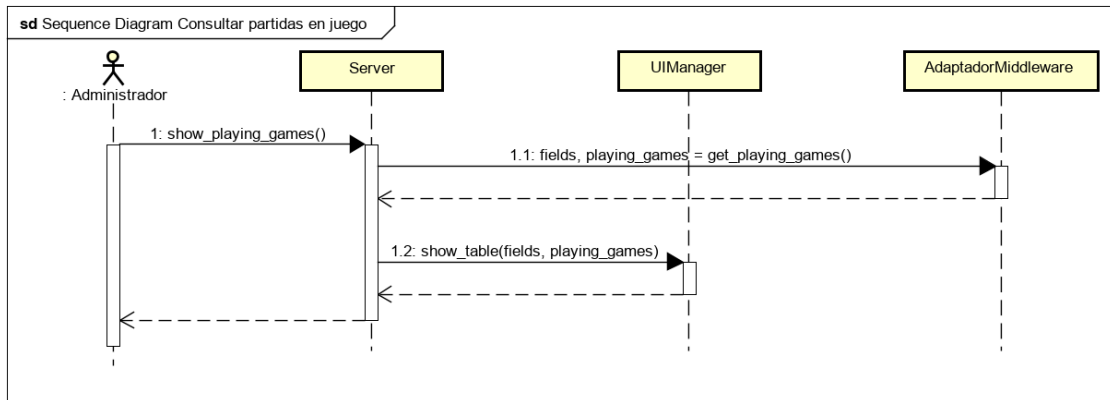
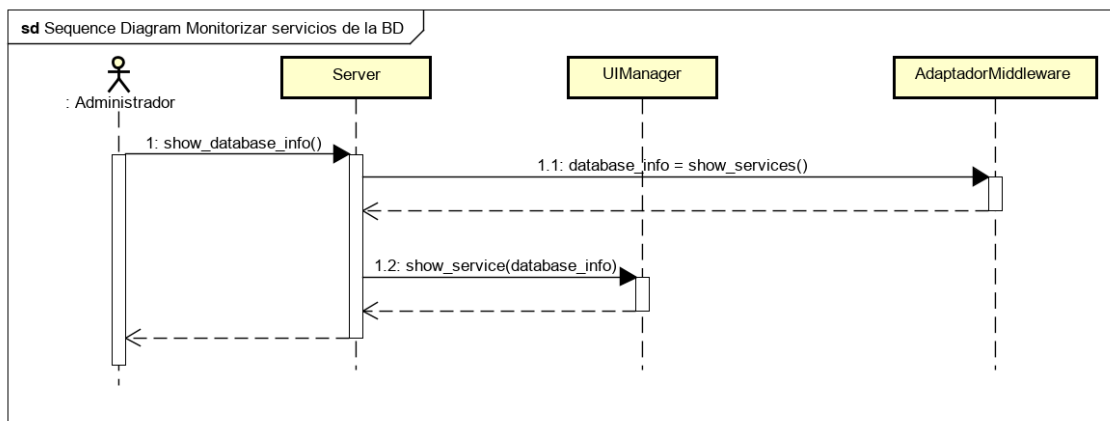


Ilustración 17: Diagrama de secuencia del CU Monitorizar servicios del servidor

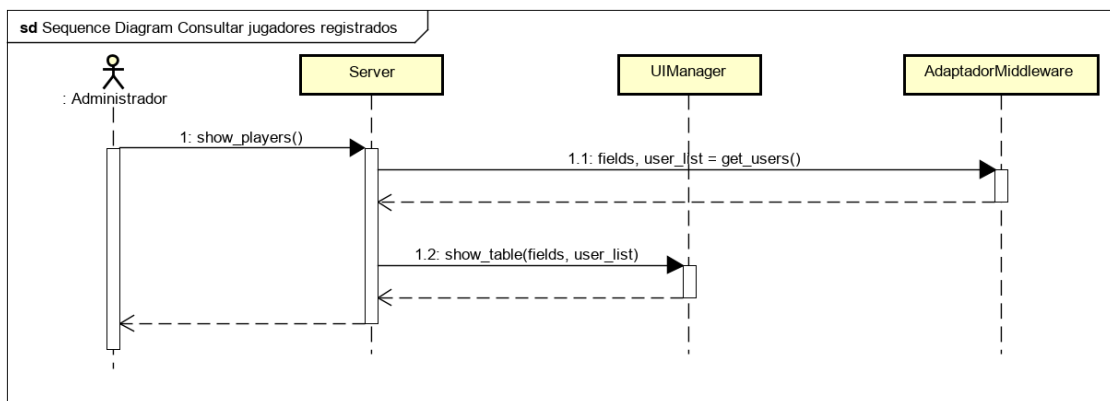




*Ilustración 18: Diagrama de secuencia del CU Consultar partidas en juego*



*Ilustración 19: Diagrama de secuencia del CU Monitorizar servicios de la BD*



*Ilustración 20: Diagrama de secuencia del CU Consultar jugadores registrados*

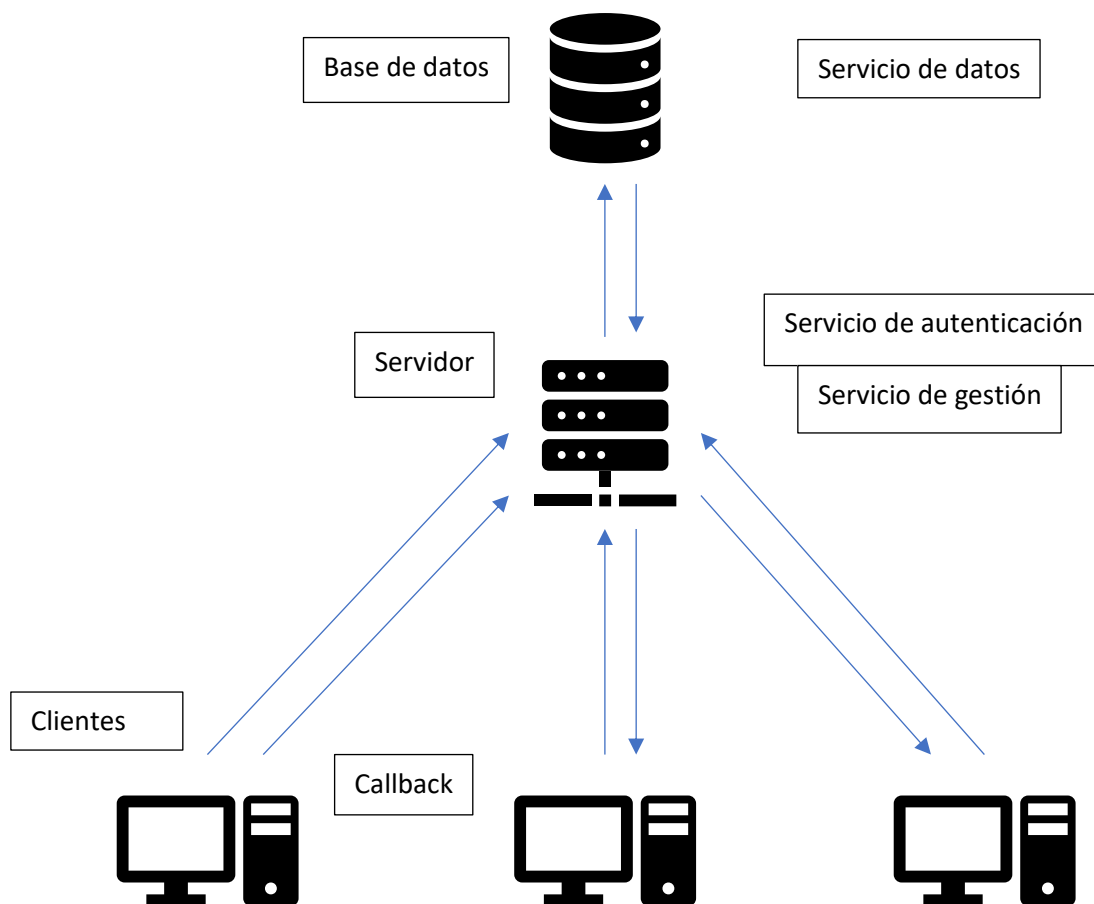


#### 4.3.5 Codificación de la solución. Modelo de Implementación

Una vez que se tienen los diagramas de interacción y de clases, se comienza con la adaptación de estos al lenguaje concreto que se va a utilizar, en nuestro caso Python. En este momento se tienen en cuenta detalles técnicos, como pueden ser el uso de una u otra base de datos, el uso de una u otra API gráfica, y demás requisitos no funcionales.

Con la información obtenida hasta el momento, se realiza un catálogo de clases y funciones, con una descripción funcional detallada de su comportamiento. Este catálogo se organiza jerárquicamente por módulos, clases y funciones.

Antes de detallar el catálogo, se va a presentar una vista arquitectónica del sistema distribuido, poniendo el foco sobre los sistemas que componen el proyecto.



## Catálogo funcional de la aplicación

### Subsistema de la base de datos:

#### Paquetes:

- Database/Services/Data/: Paquete que incluye el servicio de datos de la base de datos de la aplicación. Incluye el fichero Data.py.

#### Módulos:

- Database/Database.py: Fichero principal de la base de datos.

Clase Database: La clase principal de la base de datos. Hereda de la clase Menu del paquete Utils/Menu.py. Su propósito es iniciar y gestionar el servicio de datos.

#### Métodos:

- *show\_database\_info()*: Muestra los datos de los servicios de la base de datos relativos al middleware.
- *show\_players()*: Muestra la información relativa a los jugadores registrados en el sistema.

Database/Services/Data/Data.py: Fichero principal del servicio de datos de la base de datos de la aplicación.

Clase Data: La clase que implementa el servicio de datos de la base de datos de la aplicación.

#### Métodos:

- *create\_user(name, password)*: Recibe como parámetros el nombre (que se utilizará como identificador único) y la contraseña de un usuario, y lo registra en el sistema.
- *read\_user(name)*: Recibe como parámetro el identificador de un usuario. Busca su información en la base de datos y la devuelve.
- *update\_users(players\_to\_update)*: Recibe una lista de usuarios con sus puntuaciones y los actualiza.
- *get\_users()*: Busca los datos de los jugadores en la base de datos y los muestra.

Archivos:

Database/Players: Este archivo es la base de datos permanente en el disco de almacenamiento.

### **Subsistema del servidor:**

Paquetes:

- Server/Services/Authentication/: Paquete que incluye el servicio de autenticación del servidor de la aplicación. Su objetivo es manejar la información del registro de usuarios en la base de datos de la aplicación. Incluye el fichero Authentication.py.
- Server/Services/Management/: Paquete que incluye el servicio de gestión del servidor de la aplicación. Su objetivo es manejar la información de las partidas que juegan los usuarios. Incluye el fichero Management.py.
- Server/Services/Management/GameBoard/: Paquete que incluye las estructuras necesarias para el funcionamiento del servicio de gestión del servidor. Incluye los ficheros Game.py, Board.py, PlayerCPU.py y Ship.py.

Módulos:

Server/ Server.py: Fichero principal del servidor.

Clase Server: La clase principal del servidor. Hereda de la clase Menu del paquete Utils/Menu.py. Su propósito es iniciar y gestionar los servicios de autenticación y gestión del servidor.

Métodos:

- *show\_server\_info()*: Muestra los datos de los servicios del servidor relativos al middleware.
- *show\_playing\_games()*: Muestra la información relativa a las partidas en juego en el sistema.

Server/Services/Authentication.py: Fichero principal del servicio de autenticación del servidor de la aplicación.

Clase Authentication: La clase que implementa el servicio de autenticación del servidor de la aplicación.

Métodos:

- *register(name, password)*: Recibe como parámetros el identificador de un usuario y una contraseña, y se los pasa al servicio de datos de la base de datos de la aplicación para que los introduzca.
- *log\_in(name, password)*: Recibe como parámetros el identificador de un usuario y una contraseña, y se los pasa al servicio de datos de la base de datos de la aplicación para que los valide. Si el usuario no existe o la contraseña no es la correcta, devuelve el valor falso, en caso de que el usuario exista y la contraseña sea la correcta devuelve el valor verdadero, y lo introduce en su registro de jugadores autenticados.
- *log\_out(name)*: Recibe como parámetro el identificador de un usuario, comprueba que se encuentre en el registro de usuarios autenticados y lo elimina del registro.

Server/Services/Management/Management.py: Fichero principal del servicio de gestión del servidor de la aplicación.

Clase Management: La clase que implementa el servicio de gestión del servidor de la aplicación.

Métodos:

- *show\_player\_info(name)*: Recibe como parámetros el identificador de un usuario, busca su información en el servicio de datos de la base de datos de la aplicación y la devuelve.
- *show\_waiting\_games()*: Muestra la información relativa a las partidas creadas a la espera de contrincante en el sistema.
- *create\_game(name, callback, vs\_cpu, ships\_length)*: Recibe como parámetros el identificador de un usuario, su callback, el valor que representa su voluntad de jugar contra la CPU, y una lista con las longitudes de los barcos que desea utilizar, y se los pasa al constructor de una partida. Registra la partida en el sistema middleware y la devuelve.
- *join\_game(game\_id, player, callback)*: Recibe como parámetros el identificador de una partida, el identificador de un usuario y su callback, y llama al método *set\_away\_player* de la clase *Game*, para introducir al usuario como rival en una partida creada a la espera de contrincante.

- *get\_playing\_games()*: Muestra la información relativa a las partidas que se están jugando en el sistema.
- *end\_game(game\_id, player)*: Recibe como argumentos el identificador de una partida, y el jugador que la finaliza. Actualiza en la base de datos de la aplicación las puntuaciones de los jugadores que hayan puntuado, utilizando el servicio de datos.

Server/Services/Management/GameBoard/Board.py: Fichero principal de la clase Board, que modela un tablero de juego.

Clase Board: Implementa el tablero de juego de la partida.

Métodos:

- *get\_dimensions()*: Obtiene las dimensiones del tablero de juego y las devuelve.
- *ships\_ready()*: Comprueba si los barcos de los dos jugadores han sido introducidos.
- *add\_ships(ships, player, ships\_length)*: Recibe como parámetros el número del jugador, una lista de longitudes de barcos, y una lista de barcos en formato de 3 letras; column, fila y orientación. Guarda sus casillas en diccionarios.
- *check\_shoot(shoot, player)*: Recibe como parámetros el número del jugador, y un disparo en el formato de 2 letras; column y fila. Comprueba si el disparo ha impactado en un barco rival y devuelve el resultado.
- *player\_defeated(player)*: Recibe como parámetro el número del jugador y comprueba si le quedan barcos o por el contrario ha sido derrotado.

Server/Services/Management/GameBoard/Game.py: Fichero principal de la clase Game, que modela una partida del juego.

Clase Game: Implementa una partida.

Métodos:

- *get\_id()*: Devuelve el identificador de la partida.
- *is\_against\_cpu()*: Devuelve verdadero si el jugador eligió jugar contra la CPU. Devuelve falso en caso contrario.
- *is\_running()*: Devuelve verdadero si la partida se está jugando. Devuelve falso en caso contrario.
- *get\_status()*: Si la partida ha comenzado, devuelve la información relativa a su estado: nombre de los jugadores y sus puntuaciones, dimensiones del tablero y

tamaños de barcos utilizados. Si la partida no está en juego, devuelve los datos relativos al único jugador presente en la partida.

- *get\_ships\_length()*: Devuelve una lista con los tamaños en casillas de los barcos elegidos en la creación de la partida.
- *get\_board\_dimensions()*: Devuelve las dimensiones del tablero de juego; valores de las filas y las columnas.
- *set\_home\_player(name, callback)*: Recibe como parámetros el identificador del jugador que crea la partida y su callback, y los almacena en un diccionario.
- *set\_away\_player(name, callback)*: Recibe como parámetros el identificador del jugador que se une a la partida y su callback, y los almacena en un diccionario.
- *set\_cpu\_player(name)*: Recibe como parámetro el identificador creado en el servidor para un jugador virtual, crea al nuevo jugador y lo almacena en un diccionario. Ejecuta el método *run* del jugador virtual.
- *lobby(player)*: Recibe como parámetro el número del jugador en la partida. Si es el jugador que creó la partida y juega contra la CPU, da comienzo a la partida. Si el jugador eligió jugar contra otro jugador humano, le pondrá a la espera indicando el identificador de su partida. Si el jugador que ejecuta el método es el jugador rival, se informa al creador de que un jugador se ha unido a su partida y solicita a los jugadores la posición de sus barcos mediante el método *ask\_ships()*.
- *ask\_ships()*: Envía a los jugadores el evento correspondiente para que introduzcan la posición de sus barcos.
- *add\_ships(ships, player)*: Recibe como parámetro una lista de barcos en formato de 3 letras (columna, fila y orientación) y el número del jugador en la partida. Introduce los barcos en la partida mediante el método *add\_ships(ships, player)* de la clase *Board*.
- *start()*: Fija el estado de la partida al estado de ejecución, comprueba que los barcos de ambos jugadores estén listos, y da comienzo a la partida notificando a ambos jugadores el evento correspondiente.
- *set\_shoot(shoot)*: Recibe como parámetro la posición del disparo del jugador en formato de 2 letras (columna y fila), comprueba el resultado del disparo, notifica a ambos jugadores el resultado y pasa el turno.



- *end()*: Elimina la partida del registro del sistema middleware y fija el estado de la partida a no ejecutando.

Server/Services/Management/GameBoard/PlayerCPU.py: Fichero principal de la clase PlayerCPU, que modela un jugador virtual.

Clase PlayerCPU: Clase principal del módulo PlayerCPU. Implementa un jugador virtual para el juego contra la CPU.

Métodos:

- *send\_ships()*: Genera una lista de barcos de acuerdo a las reglas de la partida. Valida que no tengan casillas solapadas. El formato de los barcos es de 3 letras (columna, fila, orientación).
- *send\_shoot()*: Genera un disparo de acuerdo a las dimensiones del tablero de la partida. El formato del disparo es de 2 letras (columna, fila).
- *run()*: Crea un hilo de ejecución paralelo para que el jugador virtual pueda jugar en la partida.
- *mainloop()*: Entra en un bucle mientras la partida esté jugándose, lee eventos de su cola de mensajes, y reacciona a ellos.

Clase CallbackCPU: Implementa el callback del jugador virtual, para poder recibir eventos y actuar frente a ellos, ejecutando métodos de la clase PlayerCPU.

Métodos:

- *notify(event)*: Recibe un evento de la partida y lo introduce en la cola de mensajes que comparte con la clase PlayerCPU.

Server/Services/Management/GameBoard/Ship.py: Fichero principal de la clase Ship, que modela un barco del tablero de juego de una partida.

Clase Ship: Implementa un barco del tablero de juego de una partida.

Métodos:

- *get\_squares()*: Devuelve la lista de casillas que ocupa el barco. El formato de las casillas es de 2 letras (columna, fila).
- *get\_touched\_squares()*: Devuelve la lista de las casillas que han recibido un impacto de disparo. El formato de las casillas es de 2 letras (columna, fila).
- *touched(shoot)*: Recibe como parámetro un disparo en formato de 2 casillas (columna, fila) y devuelve verdadero en caso de existir una casilla en el barco en

la ubicación del disparo, y añade la casilla a la lista de casillas tocadas por un disparo. Devuelve falso en caso contrario.

- *sunken()*: Devuelve verdadero si todas las casillas del barco han sido tocadas. Devuelve falso en caso contrario.

### **Subsistema del cliente:**

#### Paquetes:

Client/Services/: Paquete que incluye el servicio de callback del cliente de la aplicación. Su objetivo es manejar los eventos de las partidas que juegan los usuarios, introduciéndolos en una cola que comparte con el cliente. Incluye el fichero Callback.py.

#### Módulos:

Client/Client.py: Fichero principal del cliente.

Clase Client: La clase principal del cliente. Hereda de la clase Menu del paquete Utils/Menu.py. Su propósito es conectar al cliente con el servicio de autenticación del servidor.

#### Métodos:

- *register()*: Comienza el proceso de registro de un usuario en el sistema. Llama al método *request\_id()* de la interfaz de usuario para obtener el nombre y password del usuario, para después llamar al método *register(name, password)* del servicio de autenticación.
- *log\_in()*: Comienza el proceso de autenticación de un usuario en el sistema. Llama al método *request\_id()* de la interfaz de usuario para obtener el nombre y password del usuario, para después llamar al método *log\_in(name, password)* del servicio de autenticación. Finalmente crea una instancia de la clase User, la inicializa llamando a su método *init\_ui()* y ejecuta su método *mainloop()*. Llama al método *log\_out(name)* del servicio de autenticación al acabar.
- *on\_close()*: Comienza el proceso de limpieza de datos en caso de que el cliente cierre la aplicación.

Client/User.py: Fichero principal del usuario (cliente autenticado).

Clase User: La clase principal del usuario. Hereda de la clase Menu del paquete Utils/Menu.py. Su propósito es conectar al usuario con el servicio de gestión del servidor.

Métodos:

- *show\_info()*: Muestra la información relativa al jugador, su identificador y su puntuación histórica.
- *new\_game()*: Inicia la creación de una nueva partida, seleccionando las reglas de juego (número de barcos y sus longitudes y juego contra CPU o jugador) y llamando al método *create\_game(name, callback, vs\_cpu, ships\_length)* del servicio de gestión del servidor. Finalmente crea una instancia de la clase Player, la inicializa llamando a su método *init\_ui()* y ejecuta su método *mainloop()*.
- *show\_waiting\_games()*: Muestra la información relativa a las partidas que creadas a la espera de un contrincante.
- *join\_game()*: Inicia el proceso de entrada a una partida creada a la espera de contrincante. Solicita el identificador de la partida y llama al método *join\_game(game\_id, name, callback)* del servicio de gestión. Crea una instancia de la clase Player, la inicializa llamando a su método *init\_ui()* y ejecuta su método *mainloop()*.
- *change\_options()*: Solicita al jugador que indique un cambio en la interfaz, y realiza el cambio.
- *log\_out()*: Termina la ejecución del método *mainloop()*.
- *on\_close()*: Comienza el proceso de limpieza de datos en caso de que el usuario cierre la aplicación.

Client/Player.py: Fichero principal del jugador (usuario jugando una partida).

Clase Player: La clase principal del jugador. Hereda de la clase Menu del paquete Utils/Menu.py. Su propósito es conectar al jugador con el servicio de gestión del servidor y con la partida.

Métodos:

- *init\_ui()*: Inicia la interfaz de usuario, llamando al método *show\_game()* de su interfaz de usuario.
- *send\_ships()*: Inicia el proceso para que el usuario introduzca la posición de sus barcos mediante el método *request\_ships()* de su interfaz de usuario. Finalmente envía a la partida las posiciones mediante el método *add\_ships(ships, player)* de

la clase Game. Los barcos se recogen en el formato de 3 letras (columna, fila, orientación).

- *send\_shoot()*: Inicia el proceso para que el usuario introduzca la posición de su disparo mediante el método *request\_shoot()* de su interfaz de usuario. Finalmente envía a la partida la posición mediante el método *set\_shoot(shoot)* de la clase Game. El disparo se recoge en el formato de 2 letras (columna, fila).
- *read\_queue()*: Saca un evento de su cola de eventos compartida con el callback, y reacciona en consecuencia.
- *show\_event(event)*: Llama al método *show\_event(event)* de la interfaz de usuario.
- *on\_close()*: Comienza el proceso de limpieza de datos en caso de que el usuario cierre la aplicación.
- *mainloop()*: Inicia un bucle mientras la partida se esté jugando en el que se llama al método *read\_queue()* para procesar los mensajes que vayan llegando procedentes de la partida al callback del jugador, que los va introduciendo en la cola.

Client/Options.py: Fichero principal de las opciones de usuario.

Clase Options: Implementa el menú de opciones configurables del usuario. Hereda de la clase Menu del paquete Utils/Menu.py. Su propósito es permitir al usuario realizar cambios en la interfaz de usuario.

Métodos:

- *change\_theme()*: Permite realizar un cambio de tema en la GUI seleccionada.

Client/Services/Callback.py: Fichero principal del servicio callback del cliente de la aplicación.

Clase Callback: La clase que implementa el servicio callback del cliente de la aplicación

Métodos:

- *notify(msg)*: Recibe como parámetro un mensaje, y lo introduce en la cola de mensajes que comparte con la clase Player.

## **Subsistema de útiles:**

Paquetes:

- Utils/Adapters/: Paquete que incluye los objetos software adaptadores para las librerías middleware y las librerías de acceso a bases de datos. Su objetivo es independizar la lógica del programa principal del uso de una librería u otra. Incluye los ficheros Sqlite3Adapter.py y Pyro5Adapter.py
- Utils/Testing/: Paquete que incluye clases para testing de la aplicación. Su objetivo es proporcionar una forma alternativa al middleware para probar la trazabilidad del programa. Incluye los ficheros FakeService.py y FakeGame.py
- Utils/UI/: Paquete que incluye las clases necesarias para la interfaz del usuario. Su objetivo es independizar la lógica del programa principal del uso de una librería o paradigma de representación. Incluye los ficheros UIManager.py, GUI.py y TUI.py.

Módulos:

Utils/Menu.py: Fichero principal de la clase Menu.

Clase Menu: Clase base para las clases Server, Database, Client, User, Player y Options, que implementa un menú de opciones ejecutable. Su objetivo es independizar el tratamiento de la funcionalidad de una clase con su representación.

Métodos:

- *init\_ui()*: Crea una instancia de la clase UIManager si no recibe una como parámetro de inicialización. Comprueba que su estado sea de ejecución, y si es el caso llama al método *show\_menu(options)* de su manager de interfaz de usuario.
- *mainloop()*: Llama al método *mainloop()* de su interfaz de usuario.
- *on\_close()*: Fija su estado a no ejecutando, y llama al método *quit()* de su interfaz de usuario.
- *debug(method)*: Recibe un método e imprime por pantalla (en la consola) la clase que realiza el método y su nombre.

Utils/Adapters/Sqlite3Adapter.py: Fichero principal de la clase Sqlite3Adapter.

Clase Sqlite3Adapter: Implementa un adaptador a la librería sqlite3 de la librería estándar de Python. Su objetivo es independizar el resto de la aplicación del uso de una u otra librería.

Métodos:

- *create\_user(name, password)*: Recibe como parámetros el identificador único de un usuario y su contraseña. Si el usuario no existe previamente en la base de datos, lo inserta. Devuelve verdadero en caso de haber realizado la inserción, y falso en caso contrario.
- *read\_user(name)*: Recibe como parámetro el identificador único de un usuario, recupera sus datos y los devuelve si se encuentra en la base de datos. En caso contrario devuelve una lista vacía.
- *update\_users(users\_to\_update)*: Recibe como parámetro un diccionario de usuarios con sus nuevas puntuaciones y los actualiza en la base de datos.
- *get\_users()*: Devuelve una lista de los datos de los usuarios registrados en la base de datos, a excepción de las contraseñas.

Utils/Adapters/Pyro5Adapter.py: Fichero principal de la clase Pyro5Adapter.

Clase Pyro5Adapter: Implementa un adaptador a la librería Pyro5, que no pertenece a la librería estándar de Python. Su objetivo es independizar el resto de la aplicación del uso de una u otra librería.

Métodos:

- *get\_daemon()*: Devuelve el proceso encargado de registrar servicios en el servidor de nombres de Pyro5.
- *register\_callback(service)*: Recibe como parámetro el callback de un jugador y lo registra como objeto Pyro, que implementa un objeto remoto.
- *register(name, password)*: Recibe como parámetros el identificador único de un usuario y su contraseña, y llama al método *register(name, password)* del servicio de autenticación del servidor. Devuelve verdadero si se realizó el registro, en caso contrario devuelve falso.
- *log\_in(name, password)*: Recibe como parámetros el identificador único de un usuario y su contraseña, y llama al método *log\_in(name, password)* del servicio de autenticación del servidor.
- *log\_out(name)*: Recibe como parámetro el identificador único de un usuario y llama al método *log\_out(name)* del servicio de autenticación del servidor.
- *show\_player\_info(name)*: Recibe como parámetro el identificador único de un usuario y llama al método *show\_player\_info(name)* del servicio de gestión del

servidor. Devuelve una lista de atributos de un jugador, y una tupla con los valores de éstos.

- *create\_game(name, callback, vs\_cpu, ships\_length)*: Recibe como parámetros el identificador único de un usuario, su callback, su elección sobre jugar contra la CPU, y una lista con las longitudes en casillas de sus barcos. Llama al método *create\_game(name, callback, vs\_cpu, ships\_length)* del servicio de gestión del servidor.
- *show\_waiting\_games()*: Llama al método *show\_waiting\_games()* del servicio de gestión del servidor. Devuelve una lista de atributos de una partida, y una lista de tuplas con los valores de las partidas creadas a la espera de un contrincante.
- *join\_game(game\_id, name, callback)*: Recibe como parámetros el identificador único de una partida, el identificador único de un usuario, y el callback del usuario. Llama al método *join\_game(game\_id, name, callback)* del servicio de gestión del servidor. Devuelve la partida a la que se accede.
- *end\_game(game\_id, player)*: Recibe como parámetros el identificador único de una partida, y el número del jugador. Finaliza la partida y actualiza las puntuaciones de los jugadores.
- *register\_service(service, name)*: Recibe como parámetros un servicio y su identificador único, y los registra en el servidor de nombres de Pyro5.
- *show\_services()*: Devuelve un diccionario con información relativa al servidor de nombres de Pyro5, así como el proceso demonio utilizado para los registros de servicios.
- *get\_users()*: Llama al método *get\_users()* del servicio de datos de la base de datos. Devuelve una lista de atributos de un jugador, y una lista de tuplas con los valores de los jugadores registrados en la base de datos.

Utils/UI/UIManager.py: Fichero principal de la clase UIManager.

Clase UIManager: Implementa un manager para la interfaz de usuario. Su objetivo es independizar el resto de la aplicación del uso de una u otra librería gráfica.

Métodos:

- *show\_game()*: Muestra la interfaz de usuario de la partida.

- *show\_event(event)*: Recibe como parámetro un evento del servidor y lo muestra en la interfaz de usuario.
- *show\_info(msg)*: Recibe como parámetro un mensaje para el usuario y lo muestra en la interfaz de usuario.
- *show\_waiting\_games(fields, waiting\_games)*: Recibe como parámetros la lista de campos junto con una lista con los datos de las partidas a la espera de contrincante y los muestra en la interfaz de usuario.
- *request\_id()*: Muestra un formulario para que el usuario introduzca su nombre y su contraseña. Devuelve el nombre y la contraseña del usuario.
- *request\_game\_rules()*: Muestra un formulario para que el usuario introduzca los parámetros de creación de una partida, el número de barcos y sus longitudes en casillas, y el modo contra jugador o CPU. Devuelve los parámetros mencionados.
- *request\_ships(ships\_length, rows, columns, player\_names)*: Recibe como parámetros los datos referentes a las longitudes de los barcos, las filas del tablero, las columnas del tablero, y los nombres de los jugadores. Muestra un formulario para que el usuario introduzca las posiciones en las que se situarán sus barcos. Devuelve una lista con las posiciones de los barcos del jugador.
- *request\_shoot()*: Muestra un formulario para que el usuario introduzca la posición de su disparo. Devuelve la casilla a la que el usuario desea disparar.
- *request\_game()*: Muestra un formulario para que el usuario introduzca el identificador de la partida a la que desea acceder. Devuelve el identificador de la partida.
- *change\_theme()*: Muestra un formulario para que el usuario introduzca el tema visual de la interfaz de usuario y la selecciona.
- *game\_over(result)*: Recibe como parámetro el resultado de una partida y muestra al usuario un mensaje indicando el resultado.
- *update\_square(board, value)*: Recibe como parámetro el tablero a actualizar junto con el valor de la casilla que debe actualizar.
- *update\_points(board, value)*: Recibe como parámetro el tablero a actualizar junto con el valor de los puntos que debe actualizar.
- *show\_menu(options, name)*: Recibe como parámetros el nombre de un menú junto con un diccionario de funciones y lo muestra en la interfaz de usuario.



- *show\_service(text, service\_info)*: Recibe como parámetros el nombre de un servicio junto con una lista de sus datos y los muestra en la interfaz de usuario.
- *show\_table(text, fields, data\_enum)*: Recibe como parámetros el título de la información que debe mostrar junto con sus campos y una lista de valores y los muestra en la interfaz de usuario.
- *clear\_screen()*: Limpia la pantalla donde se muestra la interfaz de usuario.
- *set\_title(title)*: Recibe como parámetro el título que debe fijar como el título de la ventana en la que se muestra la interfaz de usuario.
- *mainloop()*: Entra en un bucle para manejar los eventos de la interfaz de usuario.
- *update()*: Actualiza la información en pantalla en la interfaz de usuario.
- *quit()*: Finaliza la ejecución del bucle para el manejo de eventos en la interfaz de usuario.

#### 4.3.6 Pruebas del producto. Caja blanca y caja negra. Testing avanzado

Las pruebas sobre el producto son la siguiente parte del proceso de desarrollo. El objetivo de las pruebas (testing) es encontrar defectos, fallos y errores, así como verificar los requisitos, evaluar la calidad del software o prevenir defectos. Una prueba de software se considera exitosa cuando encuentra un problema de los anteriores.

Un defecto es una diferencia entre el valor de retorno que se espera en un determinado momento por parte de una aplicación, y el resultado real obtenido. Un error es una equivocación por parte de un programador, pudiendo ser un error lógico o de uso del lenguaje. Un fallo es un problema causado por un error, que se manifiesta en tiempo de ejecución.

Las pruebas de software pueden diferenciarse en la granularidad del objeto que se está sometiendo a verificación. Las pruebas unitarias verifican módulos, o clases. Las pruebas de integración prueban grupos de módulos, o sistemas. Finalmente, las pruebas de integración comprueban un sistema completo.

Los enfoques clásicos para las pruebas de software son las pruebas de caja blanca y las de caja negra.

Las pruebas de caja blanca miden la trazabilidad y estructura interna del software que analizan. Buscan recorrer todo el flujo posible de la funcionalidad de la aplicación. Para este proyecto se aplicarán este tipo de pruebas, recorriendo cada menú de la aplicación, en cada sentido en el que sea posible hacerlo.

Las pruebas de caja negra parten desde unas entradas predefinidas de las que se conoce el resultado de aplicar la operación que ejecuta un software particular sobre ellas, para ejecutarlo con las entradas y verificar si la respuesta retornada por la aplicación es correcta. Requieren probar una gran cantidad de casos posibles. En este proyecto se utilizarán en modo unitario para verificar la creación aleatoria de casillas de barcos, reconocimiento de patrones en la información aportada por el usuario, o la interacción con la base de datos permanente.

El testing avanzado trata de lidiar con el enfoque exhaustivo de las pruebas de caja negra, tratando de encontrar soluciones estadísticas y combinatorias ante la imposibilidad de probar todas las combinaciones de condiciones particulares para una aplicación.

#### 4.3.7 Despliegue del producto. Entorno virtual y dependencias

La instalación final por parte del usuario de la aplicación debe ser autocontenida en la medida de lo posible, no siendo admisible no poder ejecutar el producto por la carencia de una falta de librerías utilizadas en el código fuente.

En Python existe la posibilidad de crear un entorno virtual, en el que se pueden instalar las versiones mínimas soportadas por la aplicación de las librerías utilizadas. Incluso se pueden especificar en un archivo de texto para poder instalarlas automáticamente. Esto proporciona la ventaja de empaquetar las dependencias concretas de una aplicación, convirtiéndola en una aplicación portable.

Para crear un ambiente virtual en un directorio, escribiremos en una consola el siguiente comando:

- En Linux: `python3 -m venv /directorio/`
- En Windows (variable PATH configurada): `python -m venv \directorio\`

Para ejecutarlo, ejecutamos el script que crea Python en el directorio que se creó tras el comando anterior:

- En Linux: `source <venv>/bin/activate`
- En Windows: `<venv>\Scripts\activate.bat`

Una vez activado, podemos instalar todas las dependencias de forma local al directorio, utilizando la consola una vez más con el ambiente virtual activado y utilizando el programa pip (pip3 en Linux), que es el instalador de paquetes de Python:

- En Linux: `pip3 install <paquete>`
- En Windows: `pip install <paquete>`

Las dependencias del proyecto se han mantenido en el menor número posible, utilizando en la medida de lo posible la librería estándar de Python. Las librerías utilizadas que no son parte de la librería estándar de Python se detallaron en la sección sobre software utilizado, aunque se enumeran una vez más:

- Pyro5
- Sense Hat

#### 4.3.8 Mantenimiento. Posibilidad de cambio. Acoplamiento y cohesión

Una vez que un producto de software ha sido desplegado y entregado, se debe continuar con el mantenimiento del mismo, debido a la constante evolución de los sistemas operativos, el software y el hardware en los que se ejecuta el producto.

El mantenimiento de un programa puede suponer la reescritura parcial o total de una o varias partes del mismo. De acuerdo al diseño utilizado durante la fase de desarrollo,

el cambio de una parte de un programa puede ser desde difícil a inasumible para la empresa que lo mantiene.

Dos métricas habitualmente utilizadas en el ámbito del diseño del software son el acoplamiento y la cohesión.

El acoplamiento mide el grado en que un módulo depende de otros módulos para realizar sus tareas. Aunque se puede profundizar en el acoplamiento y distinguir entre distintos tipos (entre funciones, entre parámetros, entre atributos, etc) entenderemos aquí el acoplamiento como el nivel de conocimiento innecesario en un módulo para la realización de sus tareas. Lo preferible es mantener un acoplamiento bajo, de forma que un cambio en un módulo no afecte a un gran número de módulos que dependían de él, o un cambio en uno de los varios módulos de una aplicación no afecte siempre al mismo módulo, que depende de casi todos los demás.

La cohesión mide el grado de funcionalidad de una clase con respecto a los datos que maneja. Una vez más se puede profundizar en los tipos de cohesión del software, pero aquí entenderemos la cohesión como la eficiencia de funcionalidad de un módulo, que sabe manejar perfectamente sus datos, y sólo sus datos. Lo preferible es mantener un nivel alto de cohesión en todas las clases de una aplicación, de forma que se favorezca la reutilización y la estructura subyacente del dominio de una aplicación se vea reflejada en la sencillez del modelo de dominio.

Una de las claves para el éxito en el mantenimiento del software es diseñarlo teniendo en cuenta el equilibrio de las dos métricas anteriores. Un bajo acoplamiento conlleva normalmente una cohesión alta, y viceversa. Mantener el equilibrio entre estas dos métricas facilita la introducción de cambios en un programa, como pueden ser los cambios de escalabilidad (aumento de los recursos necesarios para la ejecución) o la extensibilidad (capacidad de añadir nueva funcionalidad).

Los patrones de diseño son parejas de problemas con su solución, presentados de forma genérica por ser comunes en casi todos los ámbitos del diseño del software.

En el desarrollo de este proyecto se ha tenido cuidado para mantener el código con una cohesión alta y un acoplamiento bajo, utilizando patrones de diseño para aumentar la extensibilidad. Se utilizan managers (objetos software que actúan como interfaz entre clases disminuyendo el acoplamiento), o adaptadores (objetos software que permiten la utilización de un servicio o librería sin tener que preocuparse por los protocolos asociados a su uso) con el objetivo de conseguir mantenibilidad.



## 5 Estudio del comportamiento

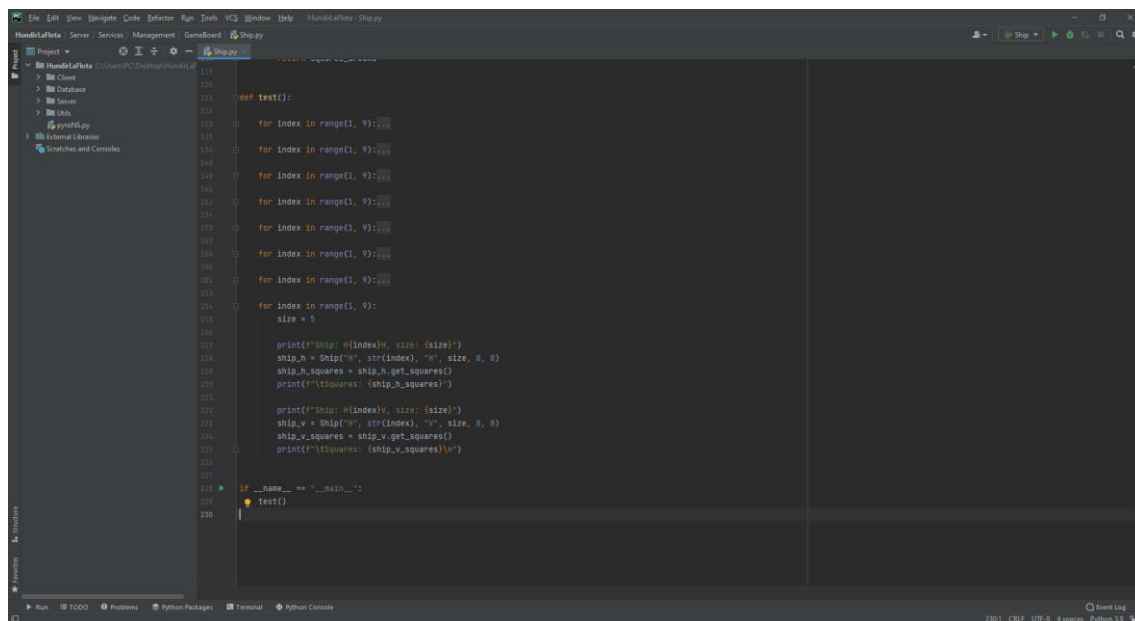
En esta sección se va a estudiar el comportamiento de la aplicación mediante las pruebas descritas en el apartado 4.3.6, concretamente pruebas de caja blanca y de caja negra. Se recorrerán los casos de uso y se presentarán capturas de pantalla de la aplicación.

### 5.1 Pruebas de caja blanca y de caja negra

El uso de pruebas de caja negra es sencillo en Python, gracias a la diversidad de módulos que se pueden encontrar en Internet, diseñados para tal efecto.

En nuestro caso aprovecharemos la capacidad de importar o ejecutar módulos en python como módulo principal, o módulo importado. De esta forma podemos importar como principal un módulo que normalmente sería importado por el módulo principal, ejecutando funciones o instrucciones que sólo se ejecutarán en ese caso.

Incluiremos test en varios archivos, en los que generaremos una respuesta conocida para un problema en concreto, y comprobaremos si la solución que genera el programa coincide con la respuesta conocida previamente. Esto es útil para, por ejemplo, comprobar que una función genera correctamente las casillas que debe ocupar un barco en un tablero.



*Ilustración 22: Prueba de caja negra para testear la creación de un barco*

Las pruebas de caja blanca que realizaremos, y que se detallarán en el siguiente apartado, consistirán en recorrer los distintos casos de uso diseñados comprobando que el flujo de ejecución de la aplicación es el correcto.

## 5.2 Casos de uso

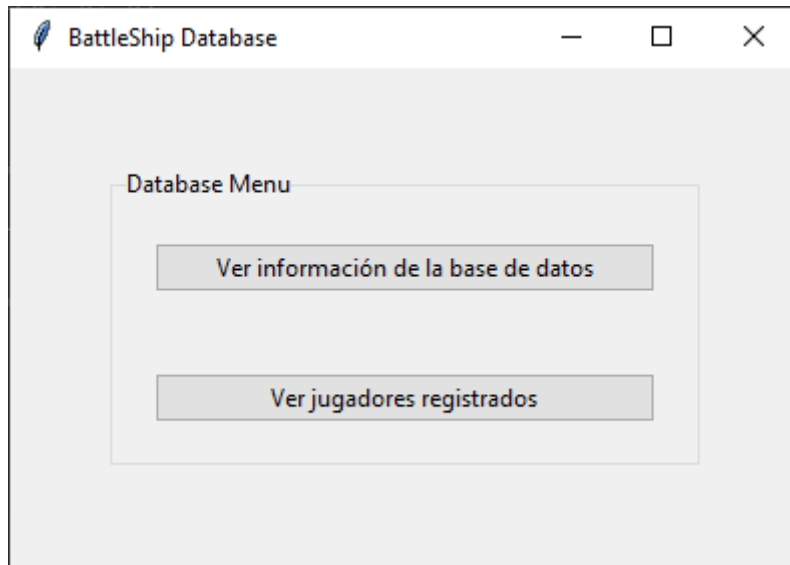
Dividiremos los casos de uso en subsistemas, teniendo el subsistema de la base de datos, el subsistema del servidor, y el subsistema del cliente.

Para los dos primeros comprobaremos los monitores asociados a estos, mientras que para el caso del cliente ejecutaremos el resto de funcionalidad no usada en el caso de los monitores de la base de datos y del servidor, pues el cliente es el encargado de solicitar estas operaciones.

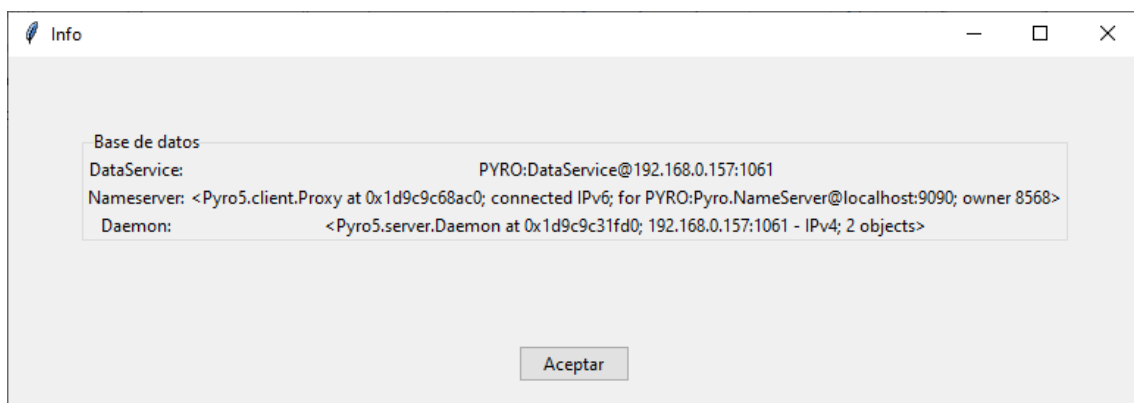


### 5.2.1 Base de datos

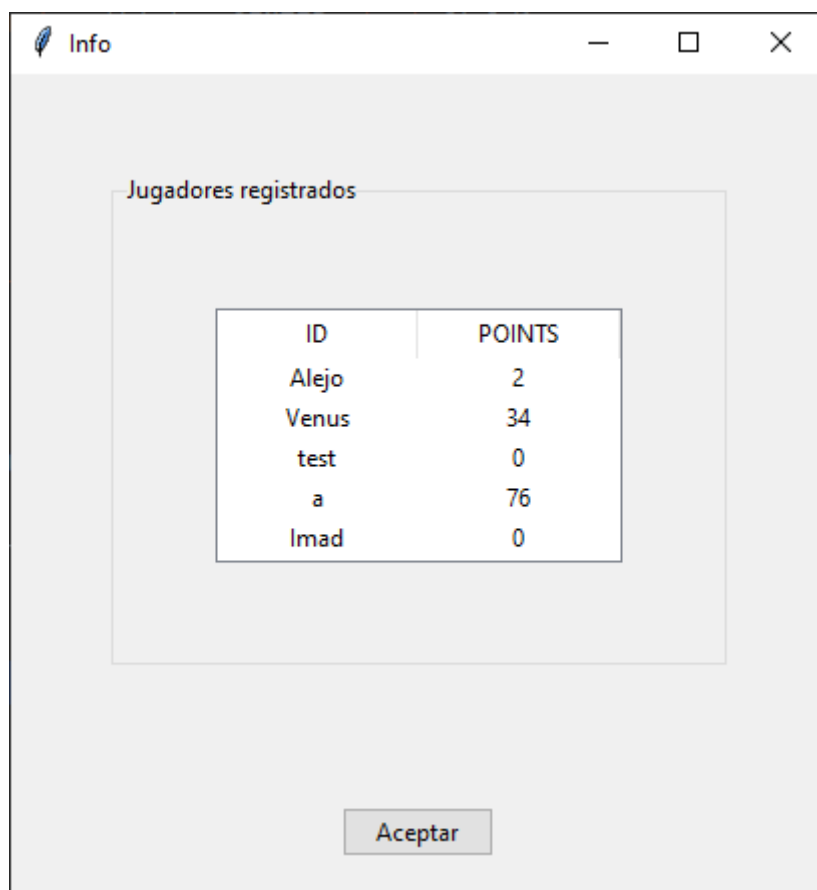
Comenzaremos con los casos de uso del subsistema de la base de datos. Comprobamos el flujo de ejecución para el monitor de la base de datos, que contiene los casos de uso *Consultar información de la base de datos*, y *Ver jugadores registrados*.



Primero veremos el caso de uso *Ver información de la base de datos*. Si accedemos a la opción “Ver información de la base de datos”, se abre la siguiente ventana, mostrando la información:

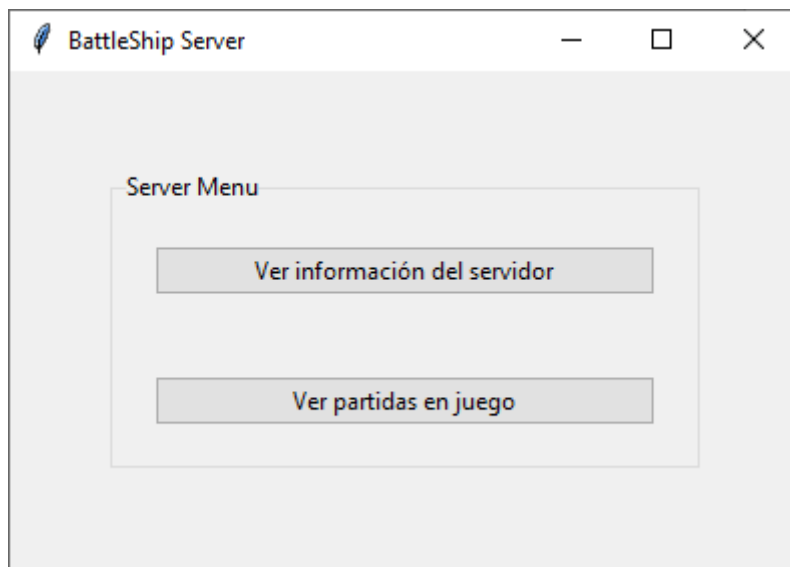


Para el caso de uso *Ver jugadores registrados*, accedemos a la opción “Ver jugadores registrados”, tras lo cual se abre la siguiente ventana, en la que podemos comprobar los datos:

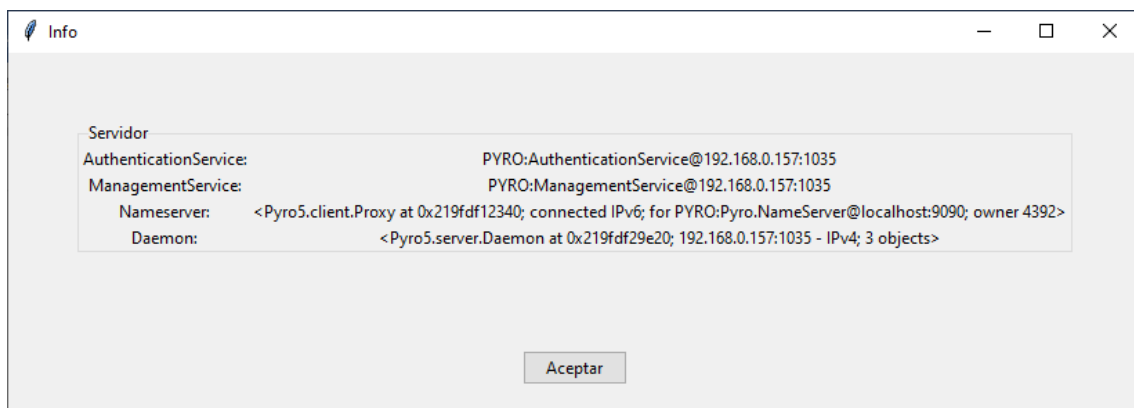


### 5.2.2 Servidor

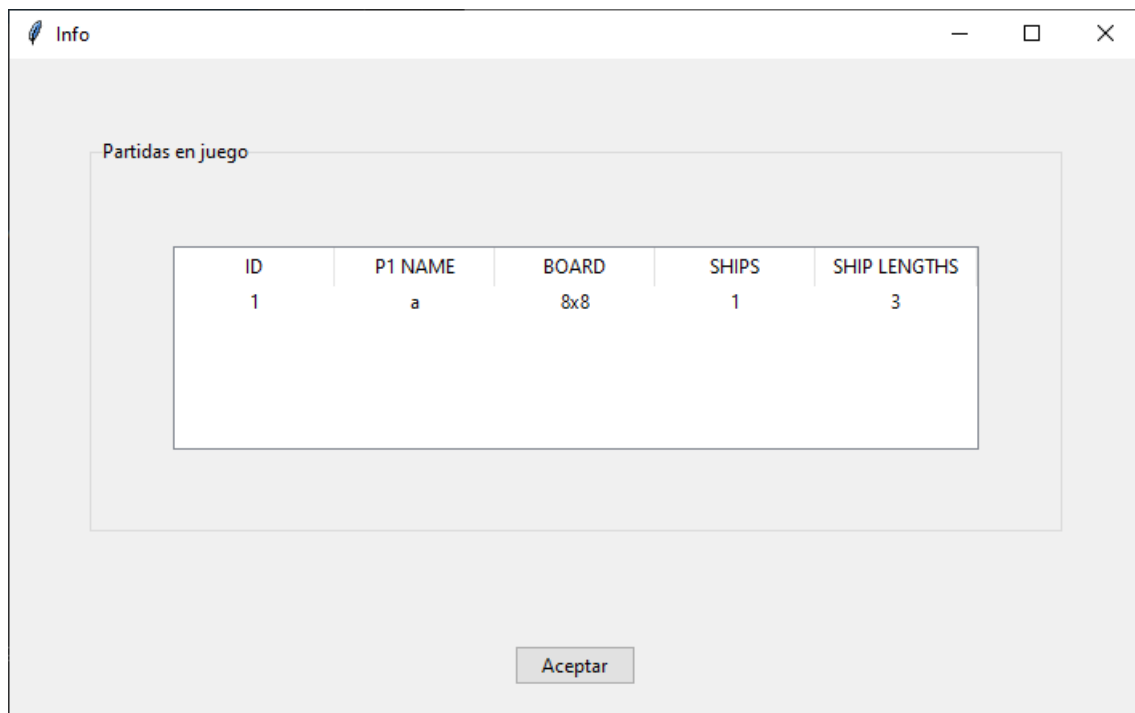
Seguiremos con los casos de uso del subsistema del servidor. Comprobamos el flujo de ejecución para el monitor del servidor, que contiene los casos de uso *Consultar información del servidor*, y *Ver partidas en juego*.



Comprobamos el caso de uso *Ver información del servidor* al acceder a la opción “Ver información del servidor”. Al hacerlo aparece la siguiente ventana con la información del servidor:

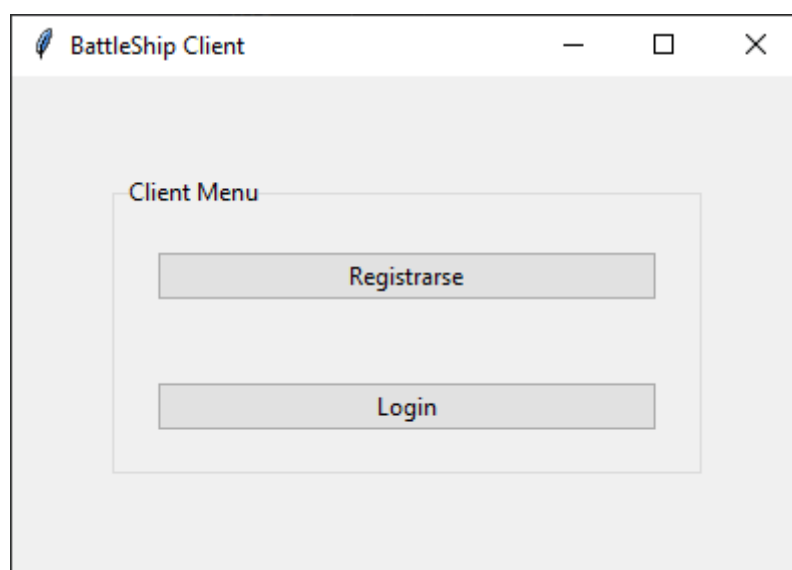


Para el caso de uso *Ver partidas en juego*, primero creamos una partida (más adelante se verá cómo hacerlo), accedemos a la opción “Ver partidas en juego”, tras lo cual se abre la siguiente ventana, en la que podemos comprobar los datos:

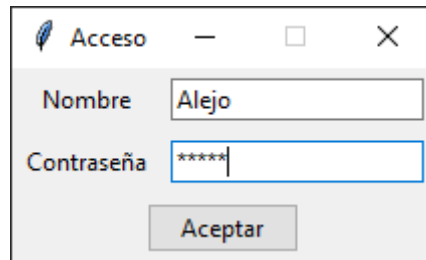


### 5.2.3 Cliente

Finalmente realizaremos las pruebas para la aplicación del cliente. Inicialmente tenemos las siguientes opciones, que representan los casos de uso *Registrarse* y *Autenticarse*.

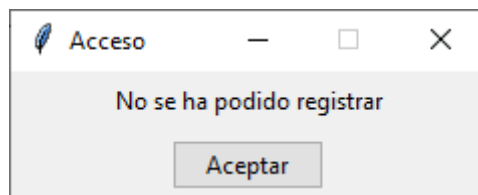


Comprobamos el caso de uso *Registrarse* al acceder a la opción “Registrarse”. Al hacerlo aparece la siguiente ventana solicitando nuestra información:



A screenshot of a Windows-style dialog box titled "Acceso". It contains two text input fields: "Nombre" with the text "Alejo" and "Contraseña" with five asterisks "\*\*\*\*\*". Below the fields is a button labeled "Aceptar".

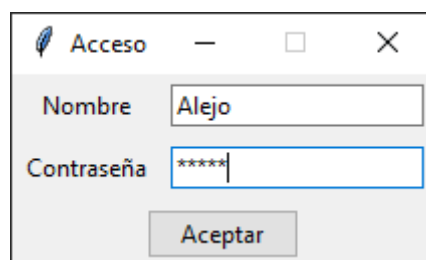
Introducimos información de un usuario ya registrado, con lo cual aparece la siguiente ventana:



A screenshot of a Windows-style dialog box titled "Acceso". The main area contains the text "No se ha podido registrar" in red. Below this text is a button labeled "Aceptar".

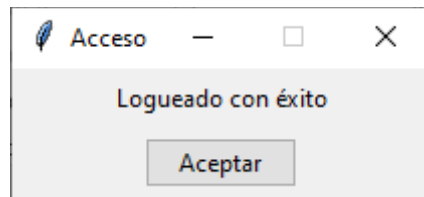
Tras esto se llega a la ventana principal de nuevo, independientemente del éxito o fracaso del registro.

Comprobaremos ahora el caso de uso *Autenticarse* seleccionando la opción “Login”. Tras hacerlo nos volverá a aparecer la ventana en la que se nos solicitan nuestros datos de inicio de sesión:

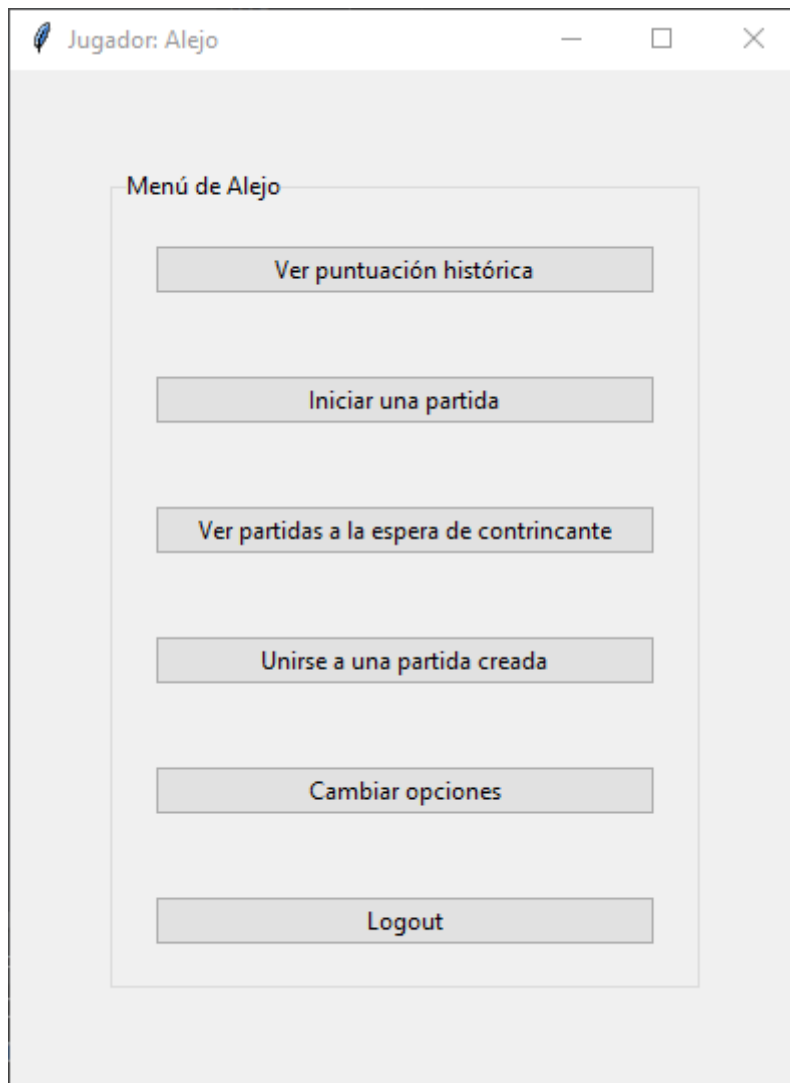


A screenshot of a Windows-style dialog box titled "Acceso". It contains two text input fields: "Nombre" with the text "Alejo" and "Contraseña" with five asterisks "\*\*\*\*\*". Below the fields is a button labeled "Aceptar".

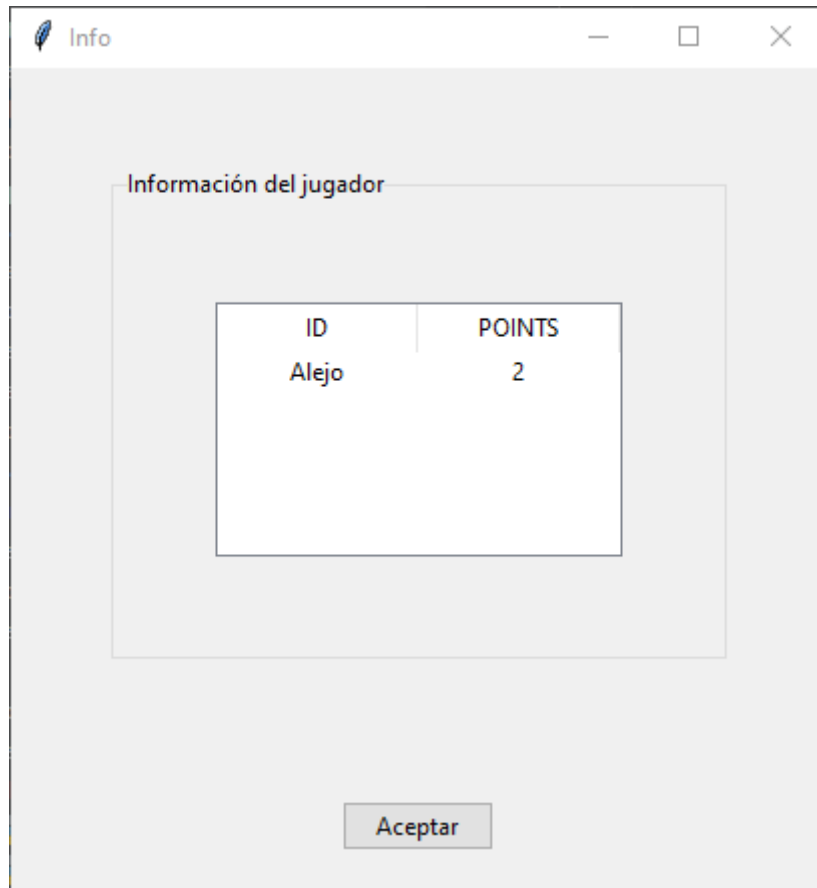
Esta vez hemos introducido los datos correctos de un usuario registrado en el sistema, con lo cual nos aparece la siguiente ventana:



Tras aceptar, accedemos al menú de usuario registrado, en el que tenemos las siguientes opciones, que representan los casos de uso *Ver puntuación histórica*, *Iniciar una partida*, *Ver partidas a la espera de contrincante*, *Unirse a una partida creada*, *Cambiar opciones*, y *Logout*:

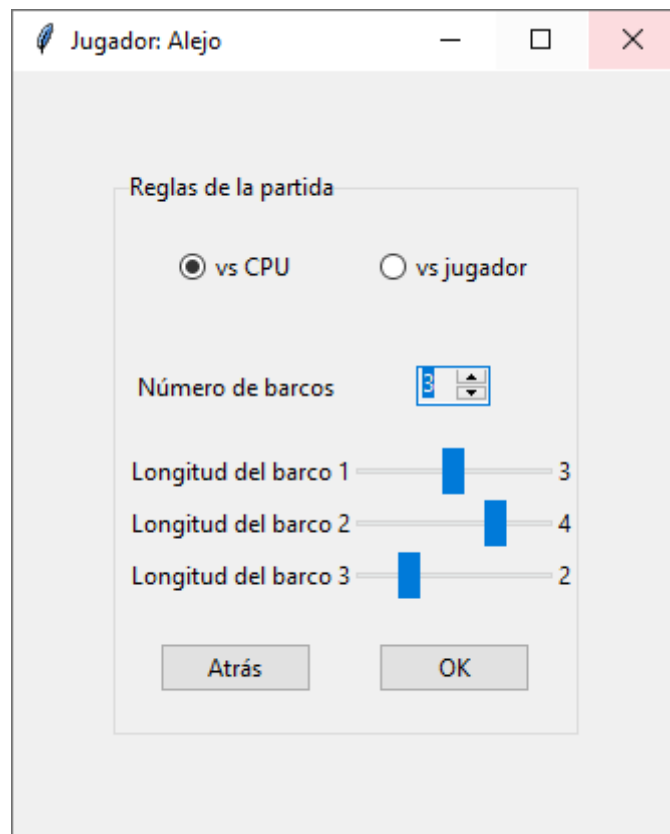


Comprobamos el caso de uso *Ver puntuación histórica* al acceder a la opción “Ver puntuación histórica”. Al hacerlo aparece la siguiente ventana con la información:



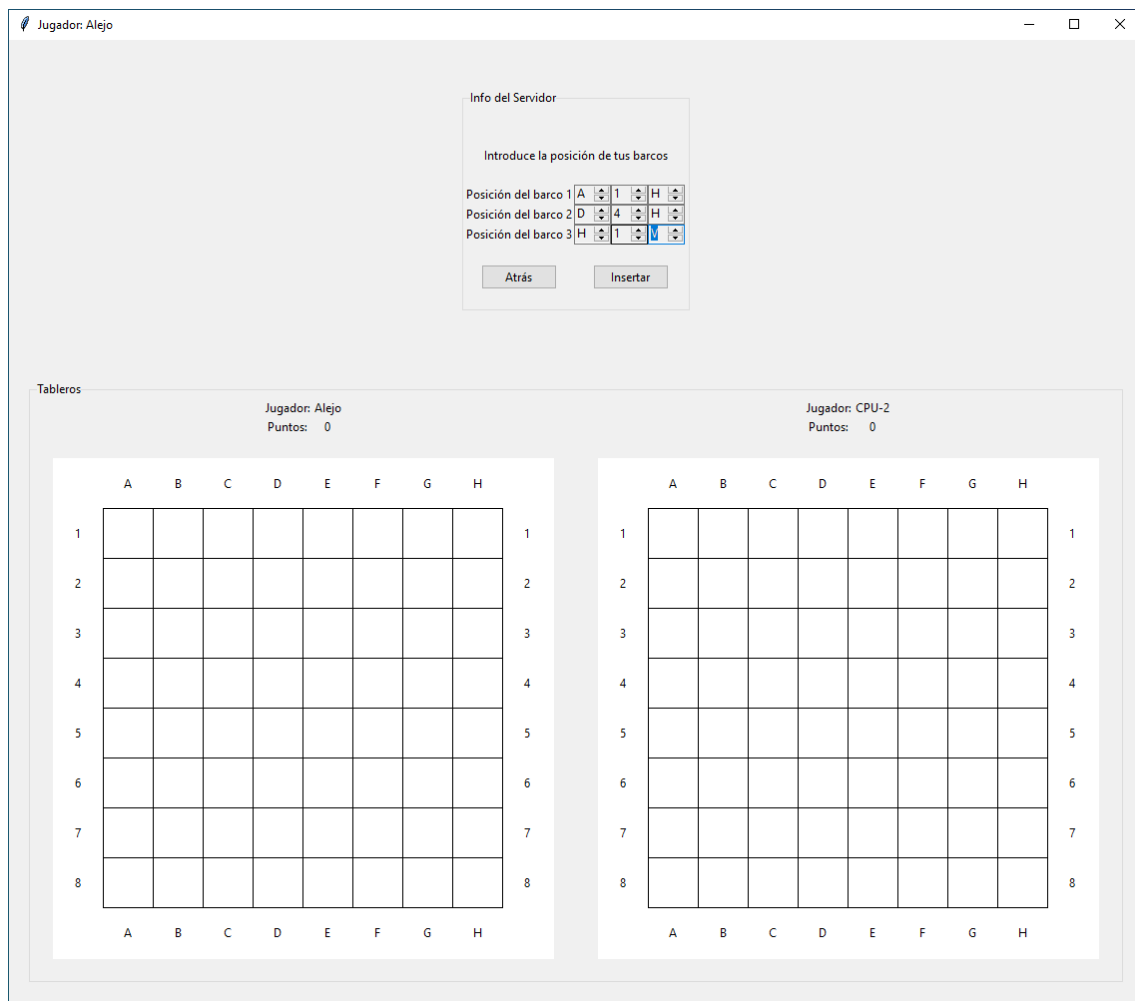
Tras aceptar se llega a la ventana principal de usuario autenticado de nuevo.

Comprobaremos ahora el caso de uso *Iniciar una partida* seleccionando la opción “Iniciar una partida”. Tras hacerlo nos aparece el menú de selección de reglas de la partida:

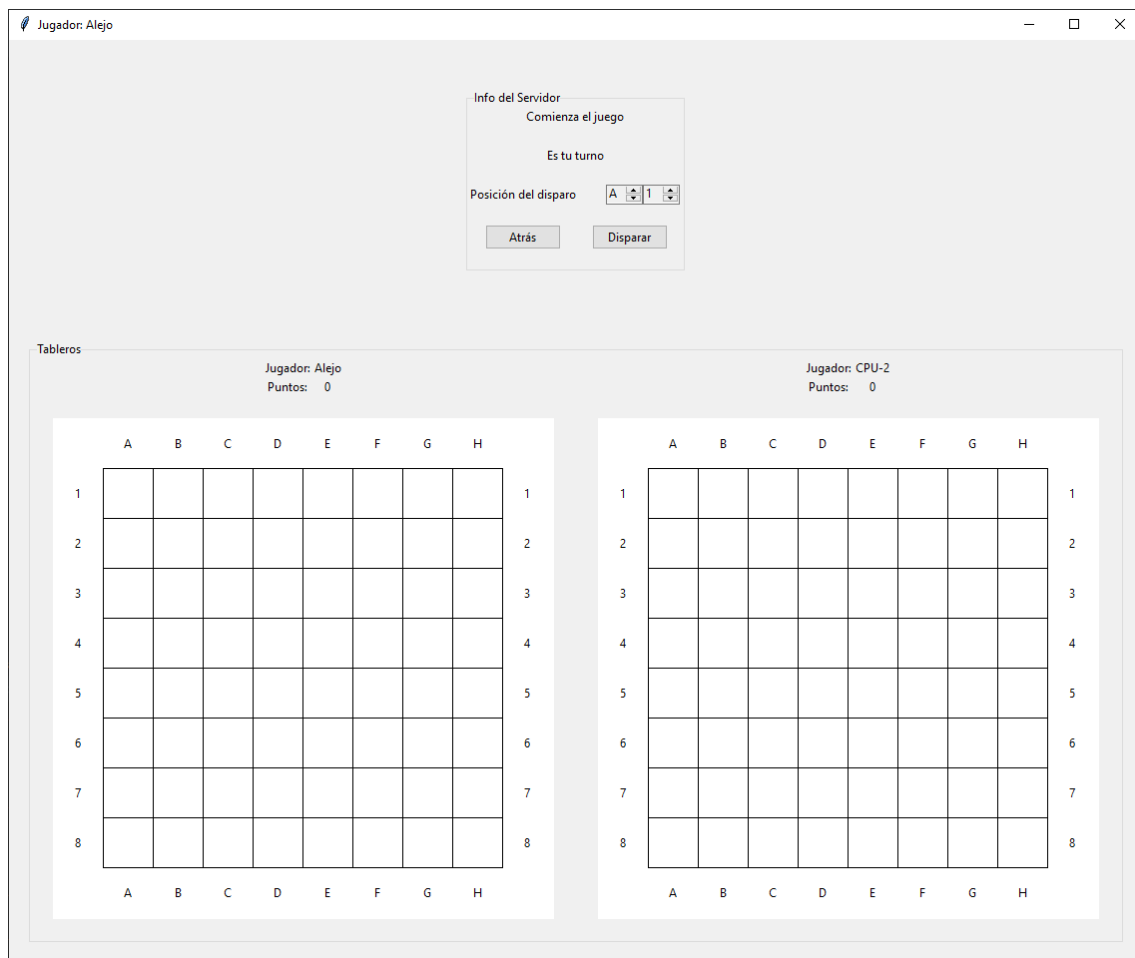


En este menú, podemos elegir si deseamos jugar contra la CPU o contra un jugador humano, así como el número de barcos, y la longitud en casillas que ocupará cada barco. Tras configurar las opciones y aceptar se llega a la siguiente pantalla:

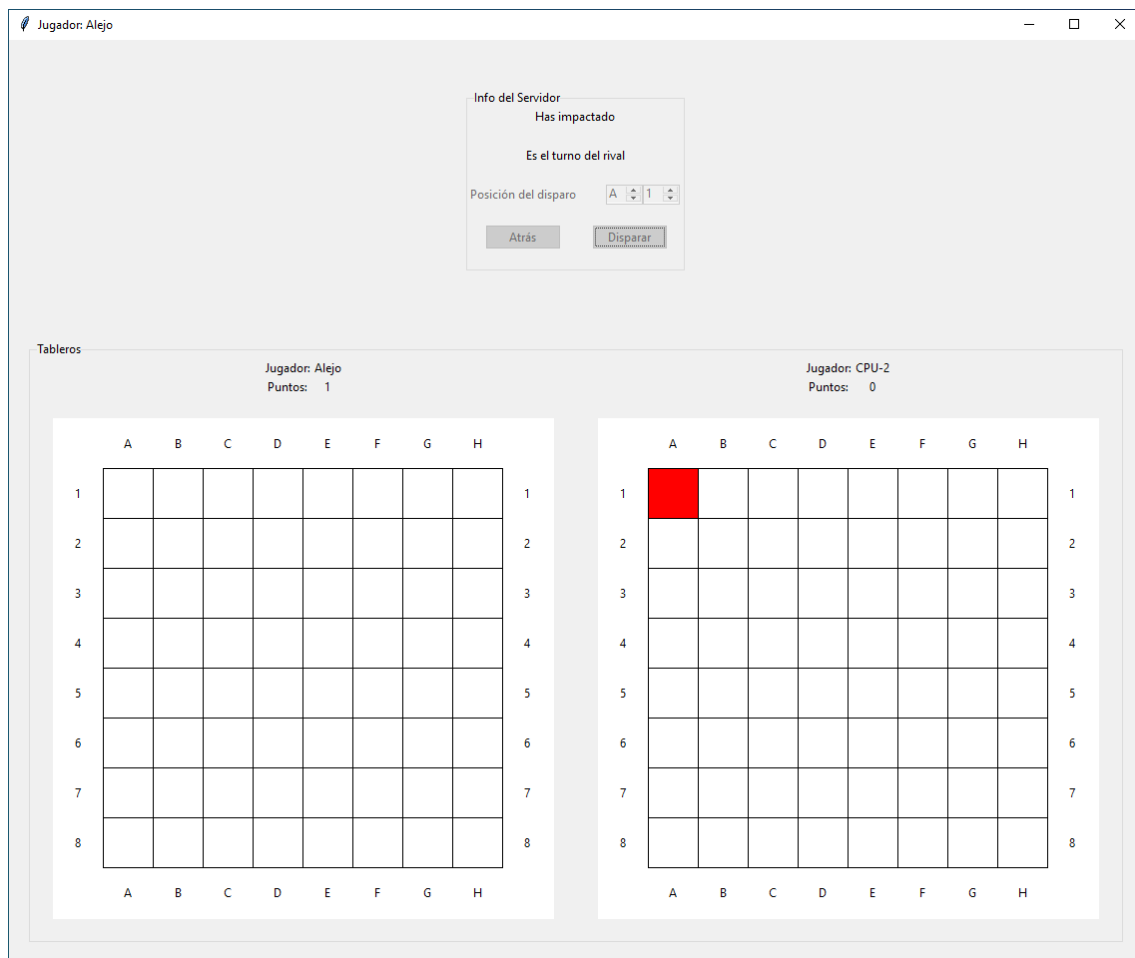




En esta pantalla, podemos elegir la ubicación de nuestros barcos. Tras aceptar se llega a la siguiente pantalla:



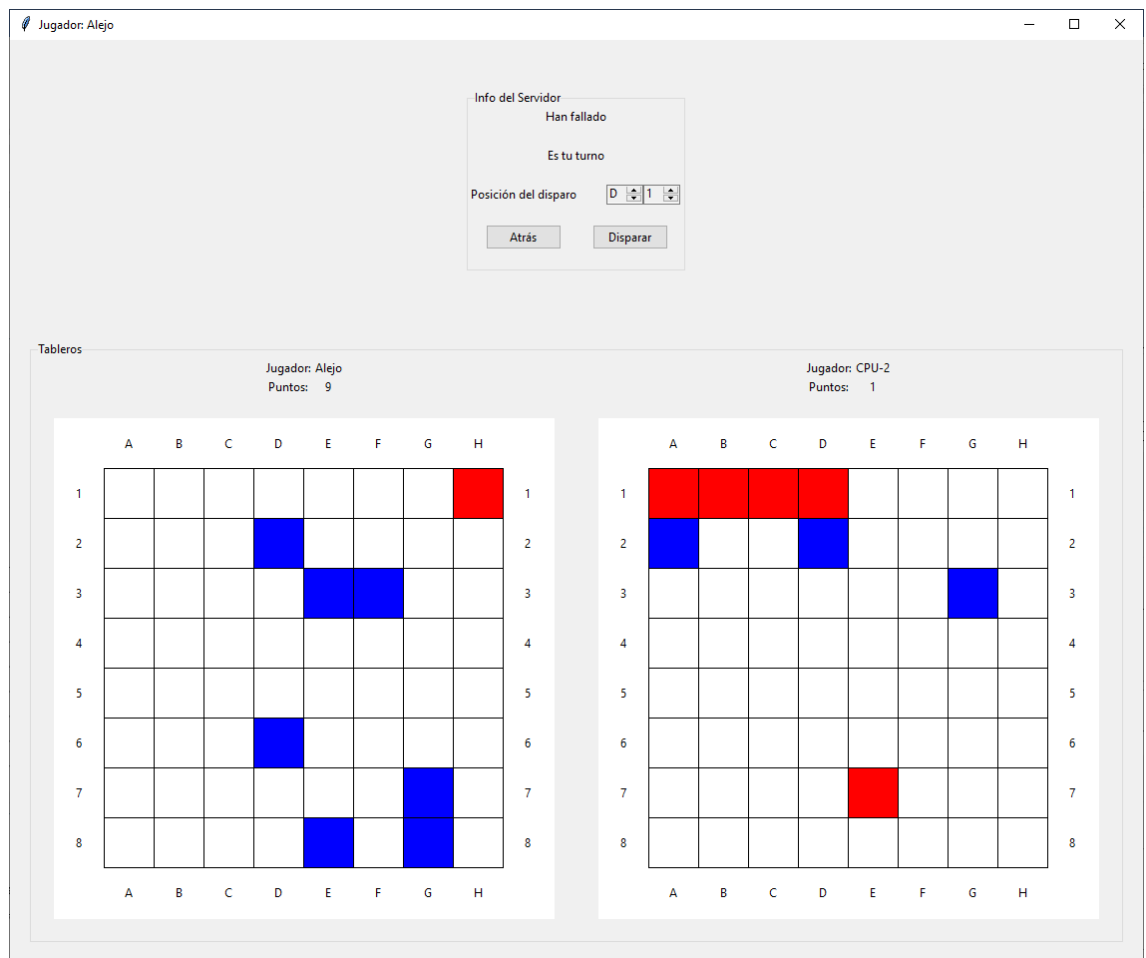
En esta pantalla, podemos elegir la ubicación de nuestro disparo. Tras aceptar se llega a la siguiente pantalla:



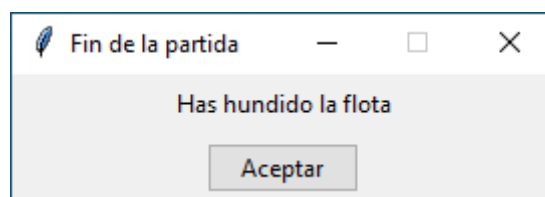
Esta vez la pantalla se quedará esperando a que el rival introduzca la posición de su disparo. Una vez el rival haya enviado su disparo, se nos mostrará la siguiente pantalla:



La siguiente captura muestra la partida en una ronda más avanzada:

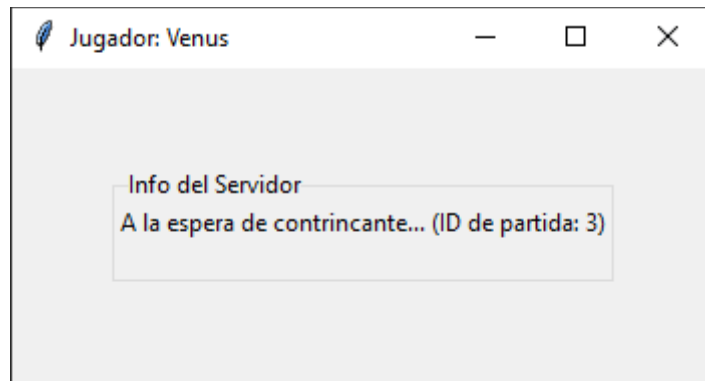


Tras varios turnos, se acaba llegando a la pantalla de fin de partida, tras lo cual se abre una ventana que nos informa del resultado:

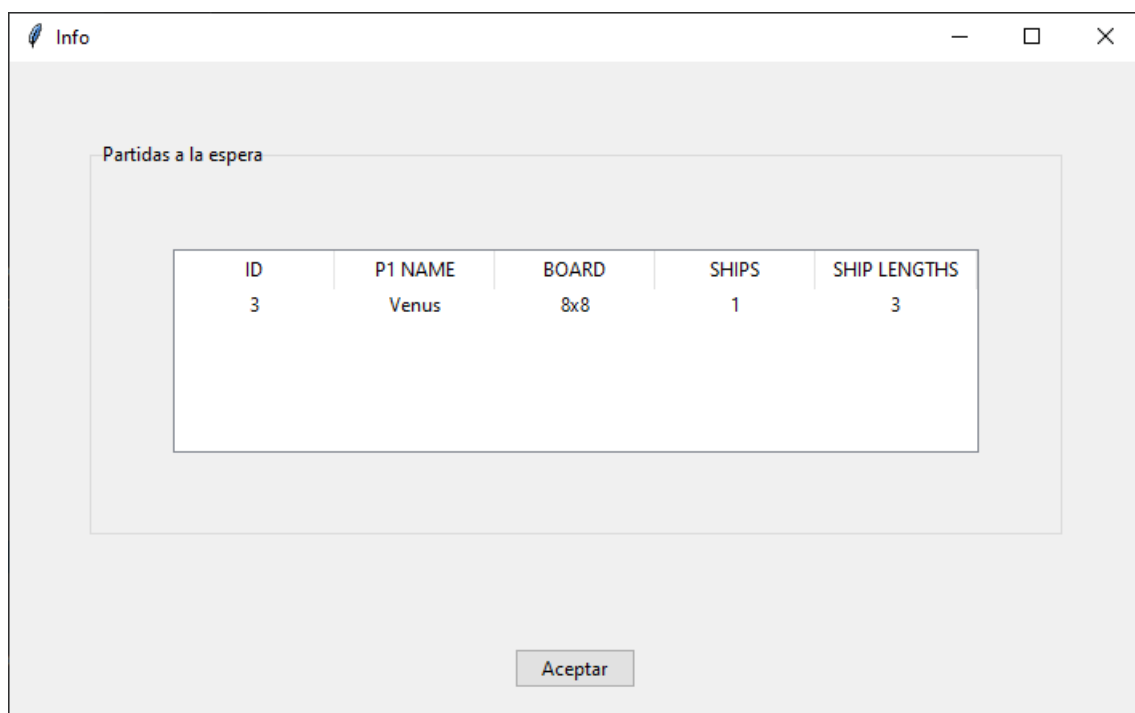


Tras aceptar se llega a la ventana principal de usuario autenticado de nuevo.

Comprobaremos ahora el caso de uso *Listar partidas a la espera de contrincante*. Primero crearemos una partida con otro jugador humano, repitiendo el caso de uso anterior, pero seleccionando esta vez el modo de juego contra un jugador humano. Tras hacerlo se abrirá la siguiente ventana:

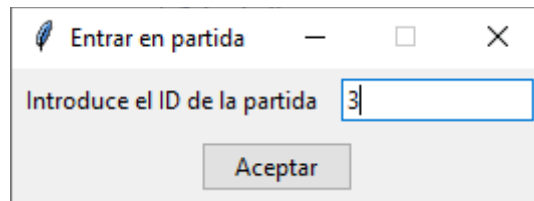


Seleccionando ahora con el primer jugador la opción “Listar partidas a la espera de contrincante”, se abre una ventana con información sobre las partidas a la espera:



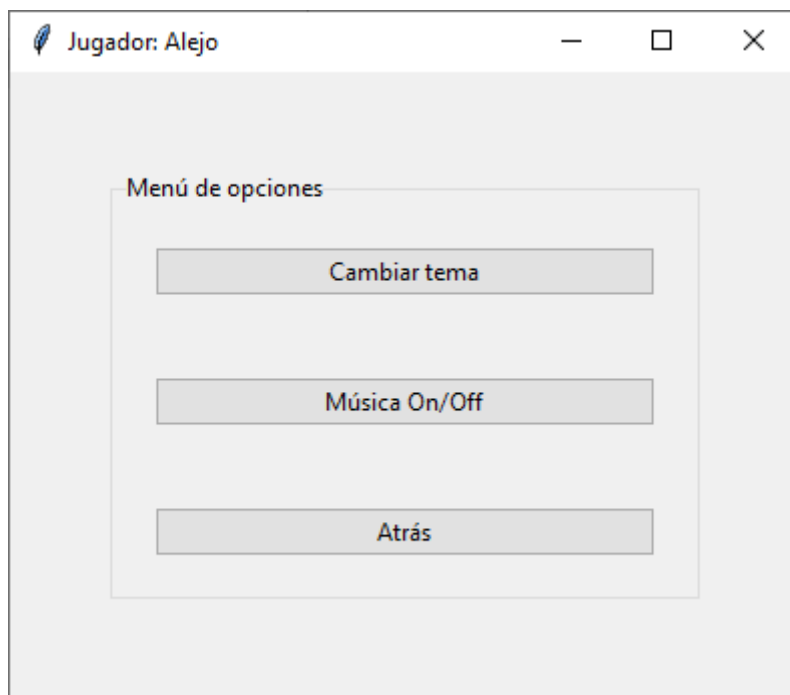
Tras aceptar se llega a la ventana principal de usuario autenticado de nuevo.

Comprobaremos ahora el caso de uso *Entrar en partida* seleccionando la opción “Entrar en partida”. Tras hacerlo nos aparece una ventana solicitando el identificador de la partida en la que se quiere entrar:

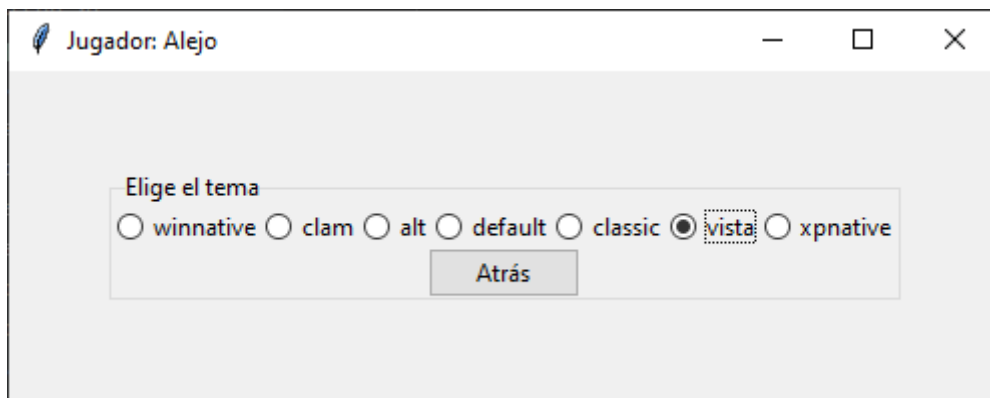


Tras aceptar se llega a la ventana de juego vista en anteriores casos de uso. Tras concluir la partida se llega al menú de usuario autenticado.

Comprobaremos ahora el caso de uso *Configurar aplicación* seleccionando la opción “Entrar en partida”. Tras hacerlo nos aparece el siguiente menú con las opciones disponibles:



Si pulsamos en la opción “Cambiar tema”, aparece el siguiente menú en el que podremos cambiar el aspecto de la aplicación:



Tras pulsar en la opción “Atrás”, volvemos al menú anterior. Si pulsamos la opción “Música On/Off” desactivaremos la música. Pulsando nuevamente la opción “Atrás” volveremos al menú de usuario autenticado.

Comprobaremos ahora el caso de uso *Salir del sistema* seleccionando la opción “Logout”. Tras hacerlo volvemos al menú principal de usuario no autenticado.



## 6 Planificación y presupuesto

En esta sección se detalla la planificación seguida en el proyecto, y que se diseñó en el modelo de anteproyecto anterior al desarrollo de la solución. En base a esa planificación se hará una estimación del presupuesto necesario para su realización.

### 6.1 Planificación

Se detallan las actividades seguidas durante el desarrollo de la solución y se crea un cronograma (diagrama de Gantt) para representarlas.

#### 6.1.1 Actividades

La planificación original estimada en el documento de modelo de anteproyecto consistió en las siguientes actividades:

- Investigar las librerías a utilizar: Investigación de la documentación y uso de las librerías que se pueden utilizar en el proyecto con objeto de elegir las que se utilizarán.
- Pruebas con el simulador Sense Hat: La placa Sense Hat dispone de un emulador web en el que poder probar el código que se programa para ella. Se realizan pruebas con el simulador.
- Preparación del entorno de desarrollo: Se obtienen las librerías necesarias y los programas necesarios para el desarrollo de la solución.
- Análisis del problema a resolver: Se toman los requisitos de manera formal y se investiga sobre la posibilidad de añadir nuevos a los iniciales.
- Diseño de la solución: Se diseña la solución en base a los análisis y los requisitos. Se generan varios artefactos de diseño de software que se utilizarán para la siguiente fase.
- Codificación de la solución: Se codifica la solución especificada en los diseños.
- Pruebas de la solución: Se diseñan pruebas a partir de los requisitos y se realizan las pruebas con la solución codificada.

- Valoración del proyecto: Se valoran los resultados obtenidos en base a los requisitos especificados.
- Redacción de la memoria del proyecto: Se reúne la documentación, se clasifica y se sintetiza. Se realiza este documento.

### 6.1.2 Diagrama de Gantt

El diagrama de Gantt muestra las actividades necesarias para la realización de un proyecto. Se muestra como una tabla ordenada en el tiempo con las actividades en forma secuencial.

| Actividades                           | Fechas       |                |                |            |              |            |            |           |            |
|---------------------------------------|--------------|----------------|----------------|------------|--------------|------------|------------|-----------|------------|
|                                       | Octubre 2020 | Noviembre 2020 | Diciembre 2020 | Enero 2021 | Febrero 2021 | Marzo 2021 | Abril 2021 | Mayo 2021 | Junio 2021 |
| Investigar las librerías a utilizar   |              |                |                |            |              |            |            |           |            |
| Pruebas con el simulador Sense Hat    |              |                |                |            |              |            |            |           |            |
| Preparación del entorno de desarrollo |              |                |                |            |              |            |            |           |            |
| Análisis del problema a resolver      |              |                |                |            |              |            |            |           |            |
| Diseño de la solución                 |              |                |                |            |              |            |            |           |            |
| Codificación de la solución           |              |                |                |            |              |            |            |           |            |
| Pruebas de la solución                |              |                |                |            |              |            |            |           |            |
| Valoración del proyecto               |              |                |                |            |              |            |            |           |            |
| Redacción de la memoria del proyecto  |              |                |                |            |              |            |            |           |            |

La planificación original estimada no ha sido diferente en cuanto a actividades con respecto a la planificación real seguida, si bien los plazos reales se han solapado más de lo que se estimó, de forma que la fase de codificación ha durado hasta el final, prácticamente.

## 6.2 Presupuesto

En esta sección se hará uso de un modelo de estimación empírico para el presupuesto del proyecto. En concreto se hará uso del modelo constructivo de costos COCOMO (Boehm, 1981). Se utilizará una estimación de 4000 líneas de código (LOC), de forma que el modo que mejor se aproxima a nuestros requisitos es el modo orgánico, que cubre el rango hasta las 50000 LOC. El submodelo utilizado será el intermedio, que incluye una lista de factores de ajuste para mejorar la precisión de la estimación.

### 6.2.1 Estimación del esfuerzo en unidades de persona/mes

Las fórmulas del modelo COCOMO son las siguientes:

$$E = a \cdot KLOC^b \cdot EAF$$

$$T = c \cdot E^d$$

$$P = E/T$$

$$PR = LOC/E$$

Donde:

$$E = \text{Esfuerzo} = \frac{\text{persona}}{\text{mes}}$$

$$a = 3.2$$

$$b = 1.05$$

$$c = 2.5$$

$$d = 0.38$$

$$KLOC = \text{Kilo} - \text{líneas de código}$$

$$EAF = \text{Factor de ajuste}$$

$$T = \text{Tiempo de duración del desarrollo} = \text{meses}$$

$$P = \text{Personal} = \text{personas}$$

$$PR = \text{Productividad} = LOC \cdot \frac{\text{persona}}{\text{mes}}$$

Las constantes a, b, c y d pertenecen al modelo COCOMO en el submodelo intermedio, modo orgánico.

La siguiente tabla muestra los factores de ajuste del submodelo intermedio:

| Tipos de atributos | Atributos | Valor    |      |         |      |          |            |
|--------------------|-----------|----------|------|---------|------|----------|------------|
|                    |           | Muy bajo | Bajo | Nominal | Alto | Muy alto | Extra alto |
| Software           | RELY      | 0.75     | 0.88 | 1.00    | 1.15 | 1.40     |            |
|                    | DATA      |          | 0.94 | 1.00    | 1.08 | 1.16     |            |
|                    | CPLX      | 0.70     | 0.85 | 1.00    | 1.15 | 1.30     | 1.65       |
| Hardware           | TIME      |          |      | 1.00    | 1.11 | 1.30     | 1.66       |
|                    | STOR      |          |      | 1.00    | 1.06 | 1.21     | 1.56       |
|                    | VIRT      |          | 0.87 | 1.00    | 1.15 | 1.30     |            |
|                    | TURN      |          |      | 1.00    | 1.07 | 1.15     |            |
| Personal           | ACAP      | 1.46     | 1.19 | 1.00    | 0.86 | 0.71     |            |
|                    | AEXP      | 1.29     | 1.13 | 1.00    | 0.91 | 0.82     |            |
|                    | PCAP      | 1.42     | 1.17 | 1.00    | 0.86 | 0.70     |            |
|                    | VEXP      | 1.21     | 1.10 | 1.00    | 0.90 |          |            |
|                    | LEXP      | 1.14     | 1.07 | 1.00    | 0.95 |          |            |
| Proyecto           | MODP      | 1.24     | 1.10 | 1.00    | 0.91 | 0.82     |            |
|                    | TOOL      | 1.24     | 1.10 | 1.00    | 0.91 | 0.83     |            |
|                    | SCED      | 1.22     | 1.08 | 1.00    | 1.04 | 1.10     |            |

Significado de los atributos en la tabla:

- RELY: Fiabilidad.
- DATA: Tamaño de la base de datos.
- CPLX: Complejidad.
- TIME: Restricciones de tiempo de ejecución.
- STOR: Restricciones de memoria virtual.
- VIRT: Volatilidad de la máquina virtual.
- TURN: Tiempo de respuesta.

- ACAP: Capacidad de análisis.
- AEXP: Experiencia en la aplicación.
- PCAP: Calidad de los programadores.
- VEXP: Experiencia en la máquina virtual.
- LEXP: Experiencia en el lenguaje.
- MODP: Técnicas actualizadas de programación.
- TOOL: Utilización de herramientas de software.
- SCED: Restricciones de tiempo de desarrollo.

En nuestro caso las únicas puntuaciones que no se considerarán medias serán las del tamaño de la base de datos, experiencia con el tipo de aplicación y con el hardware. Todas ellas se consideran bajas. Entonces, multiplicando las puntuaciones obtenidas, se tiene el siguiente factor de ajuste:

$$EAF = 0.94 \cdot 1.13 \cdot 1.10 = 1.16$$

En nuestro caso:

$$E = 3.2 \cdot 4^{1.05} \cdot 1.16 = 15.91 \frac{\text{personas}}{\text{mes}}$$

$$T = 2.5 \cdot 15.91^{0.38} = 7.15 \text{ meses}$$

$$P = \frac{15.91}{7.15} = 2.22 \text{ personas}$$

$$PR = \frac{4000}{15.91} = 251.41 \text{ LOC} \cdot \frac{\text{persona}}{\text{mes}}$$

Se necesitarían 3 personas trabajando durante 8 meses para realizar el proyecto.

### 6.2.2 Estimación del coste final del proyecto

Según un artículo (universidadeuropea.com, 2019) sobre el sueldo de los ingenieros informáticos en España, un graduado en informática recién acabada su

titulación cobra entre 18000 y 22000 € brutos anuales. Teniendo en cuenta la normativa vigente en España, esto significa que el sueldo mensual es de entre 1000 y 1200 € al mes.

Si multiplicamos el sueldo por las personas que se necesitan para el proyecto, multiplicado todo por el número de meses de duración del desarrollo, se tiene un coste de desarrollo de  $3 \cdot 8 \cdot 1000 = 24000$  € de coste de desarrollo. A este coste se le añade el coste de los dispositivos hardware y licencias de software utilizado.

Precio de los dispositivos hardware utilizados:

| Dispositivo hardware                       | Precio  |
|--|---------|
| Raspberry Pi model 3B                      | 40.95 € |
| Alimentador oficial Raspberry Pi model 3B  | 8.95 €  |
| Tarjeta MicroSD Samsung EVO PLUS de 128 GB | 22.18 € |
| Sense Hat                                  | 36.95 € |

Coste total del hardware = 109.65 €.

| Totales    | Precio   |
|------------|----------|
| Desarrollo | 24000 €  |
| Hardware   | 109.65 € |

Ahora, sumando el coste de desarrollo con el coste total del hardware, se tiene un coste total del proyecto de 24109.65 €.

## 7 Conclusiones y trabajo futuro

Esta sección detalla las conclusiones finales sobre el proyecto. Los objetivos logrados respecto a los propuestos inicialmente, así como las dificultades encontradas durante el desarrollo de proyecto y las modificaciones realizadas respecto del diseño original.

### 7.1 Conclusiones

En cuanto a los objetivos generales se han alcanzado los siguientes logros y conclusiones:

- Desarrollo de un videojuego: Se ha logrado implementar con éxito un videojuego, con interfaz gráfica de usuario, sonido y música. Al estar basado en un clásico, se pueden jugar partidas divertidas.
- Uso de hardware de bajo precio y bajo consumo: El empleo de las placas seleccionadas para la implementación ha garantizado el cumplimiento de este objetivo. El desempeño del hardware y el software es bastante bueno, incluso mejorando Raspbian el que ofrece Windows.
- Uso de la tecnología de computación en sistemas distribuidos: El empleo de la librería para computación distribuida Pyro5 ha garantizado el cumplimiento de este objetivo. La configuración de la aplicación mediante el uso de Pyro5 resulta sencilla y fácil de mantener.

En cuanto a los objetivos específicos se han alcanzado los siguientes logros y conclusiones:

- Utilizar una base de datos permanente: El uso del módulo sqlite3 ha garantizado el cumplimiento de este objetivo. Se ha creado una base de datos residente en memoria, con scripts para realizar copia de seguridad y restauración de la base de datos a partir de una copia de seguridad.

- Utilizar un servicio de registro para los usuarios: El uso del servicio de autenticación del servidor junto con el servicio de datos de la base de datos han logrado cumplir con este objetivo.
- Garantizar la seguridad de los datos de los usuarios: El empleo de certificados SSL auto firmados junto con la librería para computación distribuida Pyro5 han permitido cumplir este objetivo, pues permiten la encriptación de la información compartida entre un cliente y el servidor, pudiendo incluso mejorar la seguridad hasta la verificación mutua, en la que el cliente también debe certificar su identidad.

En cuanto a los requisitos funcionales se han alcanzado los siguientes objetivos y conclusiones:

- Permitir a un usuario registrarse en el sistema: Objetivo cumplido mediante el uso del servicio de autenticación del servidor junto con el servicio de datos de la base de datos.
- Permitir a un usuario autenticarse en el sistema: Objetivo cumplido mediante el uso del servicio de autenticación del servidor junto con el servicio de datos de la base de datos.
- Permitir a un usuario autenticado consultar sus datos: Objetivo cumplido, como se puede comprobar en la sección 5.2.3.
- Permitir a un usuario autenticado crear una partida nueva contra otro usuario o contra la CPU: Objetivo cumplido, como se puede comprobar en la sección 5.2.3.
- Permitir a un usuario autenticado ver una lista de partidas iniciadas a la espera de contrincante: Objetivo cumplido, como se puede comprobar en la sección 5.2.3.
- Permitir a un usuario autenticado unirse a una partida creada a la espera de contrincante: Objetivo cumplido, como se puede comprobar en la sección 5.2.3.
- Permitir a un usuario autenticado cerrar la sesión activa: Objetivo cumplido, como se puede comprobar en la sección 5.2.3.
- Permitir a un administrador consultar los datos de los servicios del servidor: Objetivo cumplido, como se puede comprobar en la sección 5.2.2. El uso de un monitor para el servidor permite consultar sus datos de conexión y servicios.



- Permitir a un administrador consultar el estado de las partidas en juego: Objetivo cumplido, como se puede comprobar en la sección 5.2.2.
- Permitir a un administrador consultar los datos de los servicios de la base de datos: Objetivo cumplido, como se puede comprobar en la sección 5.2.1. El uso de un monitor para la base de datos permite consultar sus datos de conexión y servicios.
- Permitir a un administrador consultar los datos de los usuarios registrados en la base de datos: Objetivo cumplido, como se puede comprobar en la sección 5.2.1.

En cuanto a los requisitos no funcionales, se han alcanzado los siguientes objetivos y conclusiones:

- Permitir una conexión segura entre los sistemas utilizando los protocolos TCP y SSL: Como se ha comentado en la sección de las conclusiones sobre objetivos específicos, Pyro5 permite el uso del protocolo SSL en conjunción con el protocolo TCP para lograr conexiones seguras entre servidor y cliente.
- Se debe utilizar una base de datos para garantizar la permanencia de los datos de los usuarios: Como se ha comentado en la sección de las conclusiones sobre objetivos específicos, se ha logrado satisfacer este requisito mediante el uso del módulo sqlite3 y una base de datos residente en memoria.
- Cuando un jugador se desconecte durante una partida, el servidor debe gestionar el final de la partida y cerrar la sesión del jugador: Este objetivo ha sido cumplido mediante el uso de callbacks (funciones que son pasadas como argumento de otra función para ser usadas en algún momento).
- El diseño debe separar la interfaz de usuario de la lógica de la aplicación: Este objetivo ha sido cumplido mediante el análisis y diseño de la aplicación manteniendo un bajo acoplamiento y una alta cohesión, además de patrones de diseño que facilitan la implementación, como adaptadores o managers.
- El diseño debe ser extensible, pudiendo añadir distintas interfaces de usuario: El anterior requisito fundamenta la satisfacción de este objetivo, para el que se han tenido en cuenta la implementación de una jerarquía de clases en la estructura del código que permitan una sencilla extensión. Tal es el caso de la interfaz de usuario que utiliza Sense Hat, ya que esta hereda de otra interfaz.

- El diseño debe estar orientado a los datos, para mejorar la mantenibilidad: El empleo de python como lenguaje de programación ha permitido satisfacer este objetivo. Se han usado diversas técnicas de la programación orientada a objetos. Se utiliza un patrón de “punteros a delegados” mediante el uso de diccionarios que contienen funciones. De esta forma el código lee datos y compone la lógica a partir de ellos de forma dinámica.
- El diseño debe permitir realizar test unitarios: El uso de python como lenguaje de programación y el uso de su capacidad para realizar test en módulos en función de si son importados o ejecutados como el módulo principal, ha permitido satisfacer este requisito, pudiendo aislar el comportamiento de estos módulos del resto de la aplicación.
- Debe usarse la placa Sense Hat para la comunicación con el usuario: Este requisito ha sido cumplido mediante el uso de la librería para tal efecto, junto con el uso del simulador.

En general, la conclusión es que se han logrado cumplir los objetivos propuestos al inicio del análisis y diseño, si bien se han descubierto otros objetivos que se podrían haber incluido, que se detallan en la siguiente sección.

## 7.2 Dificultades encontradas y modificaciones realizadas al diseño inicial

Durante el desarrollo de un proyecto surgen riesgos, amenazas y dificultades, que alteran el proceso de desarrollo y pueden causar retrasos en las fases del mismo. Se detallan aquí las más importantes.

### 7.2.1 Paso de Pyro4 a Pyro5

Aunque la versión 4 de Pyro es más conocida y está mejor documentada, el programa se encuentra en una fase de transición desde la versión 4 a la versión 5.

Se recomienda utilizar la versión 5 por motivos de seguridad, ya que los algoritmos que utiliza para enviar la información de forma remota se consideran seguros, mientras que en los de la versión 4 no es así.

Las dos versiones no son iguales, lo que implicó una reescritura de una parte del código que ya estaba implementada utilizando Pyro 4.

### 7.2.2 Las conexiones de red local

Configurar la red local para conectar distintos dispositivos físicos dentro de una misma red, o incluso realizar pruebas desde una máquina virtual, han requerido más pruebas y búsqueda de información de la que se había pensado en un inicio.

Esto incluye pruebas con mecanismos de seguridad como el cifrado mediante el protocolo SSL, certificados auto firmados, cortafuegos, así como particularidades de los sistemas operativos utilizados.

## 7.3 Trabajo futuro

Una vez se ha acabado un proyecto y se ha realizado la planificación del mantenimiento del mismo, se pueden planificar mejoras para próximas versiones del producto. En esta sección se detallan algunas de las propuestas.

### 7.3.1 Nuevas interfaces de usuario

El código ha sido diseñado para poder ser extendido con nuevas interfaces de usuario. Se puede heredar de alguna de las disponibles o programar una nueva, que contenga los métodos que tiene la clase UIManager.

Se puede cambiar el diseño de la interfaz añadiendo clases abstractas, de forma que una nueva interfaz que no implemente los métodos prescritos por una clase base

abstracta, no pueda ejecutarse, forzando a los programadores a cumplir con el contrato de la interfaz.

### 7.3.2 Nuevas IA

Aunque por motivos de eficiencia no es lo que se estila en los videojuegos comerciales, se podría implementar una inteligencia artificial con un enfoque más moderno, como puede ser una implementación mediante Redes Neuronales Artificiales, algoritmos genéticos, u otro enfoque matemático dentro del aprendizaje automático.

### 7.3.3 Otros adaptadores para bases de datos, renders, o computación distribuida

Como en el caso de las interfaces de usuario, el código se diseñó para poder ser extendido mediante el soporte de otros tipos de bases de datos, otros renders u otras API middleware. Basta con escribir un nuevo adaptador e incluirlo en el sistema.

En el futuro se podría utilizar un render OpenGL que aportaría un acabado más profesional a la interfaz de usuario, acceso a una base de datos mediante MySQL en lugar de sqlite3, o utilizar un middleware más parecido a Java RMI o basado en otro tipo de tecnología distribuida.

### 7.3.4 Más usos para SenseHat

Sin duda se podrían haber utilizado más sensores que tiene la placa, para controlar un cursor mediante la inclinación de la misma gracias a su giroscopio.

También se podría utilizar el acelerómetro que incorpora para mejorar la usabilidad del programa, ajustando la velocidad de los movimientos de la interfaz gráfica a la velocidad con la que el usuario mueve el dispositivo.

### 7.3.5 Mejorar la arquitectura para que soporte más juegos

Podría mejorarse sin mucho esfuerzo la arquitectura para que soportase algún tipo de juego distinto. Si este fuera un juego de estrategia por turnos, la adaptación sería casi directa. Algunos ejemplos podrían ser el ajedrez, o las damas.

### 7.3.6 Portarlo a más Sistemas Operativos

Si bien se han realizado pruebas en Windows 10, Ubuntu y Raspbian, se desconoce cómo puede comportarse la aplicación en otros sistemas en los que en principio podría funcionar sin cambiar el código.

Una opción muy interesante de cara al futuro es implementar el programa para el sistema operativo Android, pues esto permitiría ejecutar el juego en tabletas y móviles.



## 8 Bibliografía

- AEVI. (2018). <http://www.aevi.org.es/>. Obtenido de <http://www.aevi.org.es/la-industria-del-videojuego/en-espana/>  
Consultado por última vez el día 15/06/2021
- Algobia, A. (2021). <https://www.cope.es/>. Obtenido de <https://www.cope.es/blogs/gamer-cope/2021/02/18/cuantos-jugadores-tiene-fortnite-en-2021-mantiene-las-cifras-de-2020/>  
Consultado por última vez el día 15/06/2021
- Boehm, B. W. (1981). *Economía de la ingeniería de software*. Prentice-Hall.
- En.Digital. (2018). <https://en.digital/>. Obtenido de <https://en.digital/blog/videojuegos-industria-mobile-crecimiento>  
Consultado por última vez el día 15/06/2021
- EuropaPress. (2021). <https://www.europapress.es/>. Obtenido de <https://www.europapress.es/portaltic/videojuegos/noticia-audiencia-twitch-crece-165-primer-trimestre-2021-20210409182637.html>  
Consultado por última vez el día 15/06/2021
- Gómez, S. C., Pernía, M. R., & Lacasa, P. (2010). *Videojuegos y Redes Sociales. El proceso de identidad en Los Sims 3*. Murcia: Universidad de Murcia.  
Consultado por última vez el día 15/06/2021
- <https://trinket.io/sense-hat>. (2021).  
Consultado por última vez el día 19/06/2021
- Johannes, N., Vuorre, M., & Przybylski, A. K. (2020). Videogame playis positively correlated withwell-being. *Royal Society Open Science*.  
Consultado por última vez el día 15/06/2021
- Jong, I. D. (2021). <https://pyro5.readthedocs.io/>. Obtenido de <https://pyro5.readthedocs.io/en/latest/>  
Consultado por última vez el día 19/06/2021
- Python Software Foundation. (2021). <https://www.python.org/>.
- Raspberry. (2021). <https://www.raspberrypi.org/>. Obtenido de <https://www.raspberrypi.org/products/sense-hat/>  
Consultado por última vez el día 19/06/2021
- Raspberry. (2021). <https://www.raspberrypi.org/>. Obtenido de <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>

Consultado por última vez el día 19/06/2021

RetroInformática. (2002). <https://www.fib.upc.edu/>. Obtenido de

<https://www.fib.upc.edu/retro-informatica/historia/videojocs.html>

Consultado por última vez el día 15/06/2021

Stadia. (2021). <https://stadia.google.com/?hl=es>. Obtenido de

<https://stadia.google.com/?hl=es>

Consultado por última vez el día 15/06/2021

*universidadeuropea.com*. (2019). Obtenido de Cuanto cobra un ingeniero informático:

[https://universidadeuropea.com/blog/cuanto-gana-un-ingeniero-](https://universidadeuropea.com/blog/cuanto-gana-un-ingeniero-informatico#:~:text=En%202019%2C%20el%20salario%20de,profesionales%20mejor%20pagados%20en%20Espa%C3%B1a.)

[informatico#:~:text=En%202019%2C%20el%20salario%20de,profesionales%20mejor%20pagados%20en%20Espa%C3%B1a.](https://universidadeuropea.com/blog/cuanto-gana-un-ingeniero-informatico#:~:text=En%202019%2C%20el%20salario%20de,profesionales%20mejor%20pagados%20en%20Espa%C3%B1a.)

Consultado por última vez el día 15/06/2021

Las canciones utilizadas en el videojuego son de libre distribución y han sido obtenidas de la página web [www.zapsplat.com](http://www.zapsplat.com).



## 9 Anexos

### I. Creación de un certificado SSL auto firmado mediante openssl

La creación de un certificado auto firmado se puede realizar desde un sistema operativo Ubuntu, introduciendo en un terminal la siguiente instrucción, con lo que se abrirá un formulario preguntando sobre la información requerida para el certificado:

- `openssl req -new -x509 -days 365 -nodes -out certificado.pem -keyout clave.pem`

Para visualizar el certificado una vez creado, escribimos el siguiente comando en el terminal:

- `openssl x509 -in certificado.pem -noout -text`

Es posible indicar los parámetros de creación en la llamada al programa ssl mediante una instrucción del siguiente tipo:

- `openssl req -new -x509 -days 365 -nodes -out certificado.pem -keyout clave.pem -subj "/C=ES/ST=Madrid/L=Madrid/O=BattleShip/OU=Server/CN=localhost"`

Lo más cómodo es programar un script para crear un cliente y una base de datos. En el siguiente script se utiliza una extensión para añadir distintas IP a un host:

- `openssl req -new -x509 -days 365 -nodes -out certificado.pem -keyout clave.pem -config san.cnf -extensions 'req_ext'`

El anterior script hace uso de un archivo que debe ser creado con anterioridad, “san.cnf”, el cual contiene las siguientes líneas:

```
[ req ]

default_bits = 2048
distinguished_name = req_distinguished_name
req_extensions = req_ext
```

prompt = no

[ req\_distinguished\_name ]

countryName = ES  
stateOrProvinceName = Madrid  
localityName = Madrid  
organizationName = BattleShip  
commonName = localhost

[ req\_ext ]

subjectAltName = @alt\_names

[ alt\_names ]

IP.1 = 192.168.0.157  
IP.2 = 192.168.0.159