Memoria del Proyecto: Hundir la Flota

Título: Hundir la Flota

Nombre: Lois Figueira Castiñeiras

Módulo y curso: PSP | 2ºDAM

Fecha de entrega: 20/02/2026

Índice

1. Introducción

El proyecto consiste en la creación de un videojuego desarrollado en Kotlin y Compose con conceptos básicos de comunicación en red con sockets. El juego es el clásico hundir la flota con una interfaz moderna gracias a Compose y que permite jugar a varios jugadores a la vez, ver sus estadísticas y jugar contra una IA.

2. Protocolo de comunicación

Para el desarrollo del proyecto, el juego utiliza un sistema de mensajería asíncrona sobre WebSockets, enviando objetos JSON. Estos objetos actuarán como las acciones con toda la información que el sistema necesita para manejar los datos y estados correctamente.

Ejemplos de cómo podrían ser las acciones son (estas son más simples, solo son meros ejemplos):

Acción	Origen	Propósito	Payload Ejemplo
JOIN_LOBBY	Cliente	Entrar en cola o sala	{"mode": "PVP", "code": "AX32"}
MATCH_FOUND	Servidor	Notifica inicio de partida	{"config": {"boardSize": 14}, "opponent": "Bot"}
SEND_ATTACK	Cliente	Enviar coordenadas	{"x": 5, "y": 8}
GAME_STATE	Servidor	Actualización de tableros	{"turn": "player1", "hit": true, "gameOver": false}

3. Diagrama de arquitectura

El proyecto se basa en una arquitectura Cliente-Servidor asíncrona diseñada para soportar concurrencia (múltiples partidas simultáneas) y reactividad en tiempo real.

A. Modelo Cliente-Servidor (Multi-instancia):

- El servidor actúa como un Hub Central. No es una simple respuesta a peticiones, sino un sistema de comunicación bidireccional constante mediante WebSockets.
- El Servidor utiliza Corrutinas para gestionar cada conexión de forma independiente sin bloquear el hilo principal.
- El Cliente utiliza el patrón MVI (Model-View-Intent), donde la UI reacciona a los cambios de estado que llegan del servidor.

B. Diagrama de Bloques Funcionales:

- Capa de Red (WebSockets): Gestiona la entrada/salida de paquetes JSON. Convierte los bytes de la red en objetos.
- Gestor de Salas (RoomManager): Es el cerebro del servidor. Mantiene un registro de quién está en qué sala, gestiona los códigos de invitación y empareja a los jugadores con la IA o con otros humanos.
- Motor de Lógica (GameSession): Aquí reside la lógica del juego. Valida si un disparo es acierto o fallo, gestiona el cambio de turnos y verifica las condiciones de victoria.
- Capa de Persistencia: Un módulo independiente que lee/escribe en records.json de forma segura (usando Mutex para evitar que dos partidas escriban al mismo tiempo).

C. Ciclo de Vida de una Partida (Diagrama de Secuencia):

Para entender cómo interactúan los componentes, este es el flujo lógico:

- Fase de Negociación (Handshake): El cliente se conecta -> El servidor envía las estadísticas globales -> El cliente solicita una Sala.
- Fase de Sincronización: El servidor dicta la configuración (ej. 30s) -> Ambos clientes configuran sus tableros locales.

- Fase de Bucle de Juego:
    - Jugador A envía Action.Attack.
    - Servidor valida, actualiza el mapa y cambia el turno.
    - Servidor emite GameStateUpdate a ambos jugadores.
    - Cliente B detecta el cambio de turno y habilita su UI.

D. Escalabilidad y Reactividad:

La arquitectura destaca por dos puntos clave:

- Desacoplamiento: La UI no sabe la lógica de juego, solo dibuja lo que el servidor le envía. Esto permite que pasar configuraciones sea posible simplemente cambiando un valor en el servidor.
- Gestión de Concurrencia: Gracias a las Corrutinas de Kotlin, el servidor puede gestionar varias salas simultáneas con un consumo de memoria mínimo, ya que no crea un hilo físico por cada jugador.

4. Manual de usuario

4.1 Abrir desde un IDE

1. Para hacer uso del proyecto desde un IDE como IntelliJ IDEA, se deberá clonar el repositorio en el enlace proporcionado en el anexo. Desde un terminal se haría “git clone enlace_al_repo”.

2. Las instrucciones de ejecución están en INFO.md.

4.2 Instalar en Windows

1. Se accede al repositorio y en el apartado releases se descarga el .msi.

2. Luego se sigue el proceso de instalación.

3. Por último, se busca la carpeta, donde se encuentra la aplicación.

4.3 Instalar en Linux/Mac

1. Se accede al repositorio y en el apartado releases se descarga el .deb.

2. Una vez hecho se localiza y se ejecuta:
   “sudo dpkg –i rutaAlDirectorio/ProcessMonitor_Package_Linux.deb”.

3. Una vez hecho, la aplicación ya aparecerá entre las aplicaciones y en la tienda.

4.4 Manual de uso

4.4.1 Inicio y Configuración

Conexión Inicial

- Al iniciar la aplicación, se te solicitará un Nombre de Usuario. Este nombre será tu identidad en el servidor y se utilizará para registrar tus victorias en el Salón de la Fama.

Ajustes del Juego

Antes de lanzarte a la batalla, puedes personalizar la experiencia en el menú de Configuración:

- Tamaño del Tablero: Configurable desde 7x7 hasta 14x14. (El sistema reescalará automáticamente la interfaz).
- Número de Rondas: Elige entre partidas únicas o "Mejor de 3/5".
- Tiempo de Turno: Ajusta el límite de segundos por disparo (60s por defecto).

4.4.2 Modos de Juego

Jugador contra Entorno (PvE)

- Enfréntate al Bot Hunter, una IA diseñada para aprender de tus movimientos.
- Nivel de dificultad: Puedes elegir entre EASY, MEDIUM o HARD.

- Uso: Ideal para probar nuevas configuraciones de flota o practicar puntería.

Multijugador Online (PvP)

El corazón de Hundir la Flota. Tienes dos formas de jugar con personas:

- Crear Sala Privada: El sistema te proporcionará un Código de Sala de 4 dígitos. Compártelo con un amigo.
- Partida rápida: Busca una partida rápida contra otros jugadores en matchmaking.

4.4.3 Mecánicas de la Partida

Fase 1: Despliegue de la Flota

Dispones de una flota de 10 barcos con diferentes capacidades (también puedes configurar la flota clásica):

- 1 Portaaviones (5 casillas)
- 2 Acorazados (4 casillas)
- 3 Cruceros (3 casillas)
- 4 Destructores (2 casillas)

Instrucciones:

- Arrastra los barcos a la rejilla.
- Usa el botón Girar para alternar entre posición horizontal o vertical.
- El botón Confirmar solo se activará cuando todos los barcos estén colocados legalmente.

Fase 2: La Batalla

- El objetivo es hundir toda la flota enemiga antes de que el oponente hunda la tuya.
- Tu turno: Haz clic en una celda del tablero enemigo (radar principal).
- Impacto (Rojo): ¡Has tocado un barco!
- Agua (Azul): Has fallado. El turno pasa automáticamente al oponente.

- Abandono: Si sales de la partida antes de que termine, se te contará como una derrota automática y el oponente recibirá la victoria por abandono.

4. Récords y Estadísticas

Toda tu actividad se registra en el servidor. En la pantalla de Récords podrás consultar:

- Partidas Jugadas/Ganadas: Tu porcentaje de efectividad.
- Racha de Victorias: Cuántas partidas has ganado consecutivamente.
- Ranking: Clasificación global de jugadores por “win rating”.

5. Anexos

5.1 Repositorio GitHub

Enlace al repositorio de GitHub donde se aloja el proyecto y los instaladores.

loisfigueira/Hundir-la-flota

5.2 Uso de IA

Resumen de uso de IA

En este proyecto, la Inteligencia Artificial (Gemini/ChatGPT) ha actuado como un consultor y “vibe coding”. Su uso se centró en tres pilares:

- **Arquitectura de Red:** Diseño de la mensajería asíncrona mediante WebSockets y gestión de salas concurrentes.
- **Lógica Matemática y Algoritmia:** Desarrollo de la IA del "Bot Hunter" (sistema de disparo y búsqueda) y el algoritmo de validación de flotas.
- **Interfaz Reactiva:** Resolución de problemas de escalado dinámico en pantallas de alta densidad y adaptación de la cuadrícula de batalla.

La IA permitió reducir drásticamente el tiempo de depuración en errores de sincronización entre el servidor Ktor y el cliente Compose.

Ejemplos de Interacción

Prompt: "Necesito que el servidor gestione múltiples partidas a la vez sin que los mensajes de una se mezclen con otra."

Respuesta: Diseñó una estructura basada en un Map<RoomId, GameSession>. Sugirió el uso de Corrutinas de Kotlin y MutableSharedFlow para que cada sesión de juego fuera un proceso aislado y seguro (Thread-safe).

Prompt: "¿Cómo guardo los récords en un archivo JSON sin que se corrompa si dos jugadores ganan al mismo tiempo?"

Respuesta: Recomendó la implementación de un Mutex (Exclusión Mutua) en el StatsManager. Proporcionó el código para serializar la lista de jugadores con Kotlinx.Serialization, asegurando que la escritura en disco sea atómica.

Prompt: "La IA del Bot Hunter se detiene cuando implemento las salas online. ¿Cómo le doy contexto de sala?"

Respuesta: Identificó que la IA había quedado "desacoplada" del estado. Propuso inyectar el RoomId en el motor de la IA y usar un sistema de triggers por turno, donde el servidor invoca automáticamente la función de disparo si el siguiente turno pertenece a un Player marcado como isBot.

Conclusión del anexo

El uso de la IA no sustituyó la lógica de negocio ni el diseño original, sino que sirvió para optimizar la implementación técnica. El desarrollador validó cada sugerencia, adaptando el código generado a las necesidades específicas del protocolo de comunicación y garantizando la estabilidad del sistema final.