University of Applied Sciences Northwestern Switzerland
School of Business

Student Project

Sink & Destroy, Battleship Project

Project documentation: Mobile Applications with Android

Authors: Asia Marti, Dominique Saner, Marcel Senn

Lecturer: Brad Richards

Date: 7th June 2025

Contents

Project Description

This project was developed as part of a student course in Mobile Applications with Android, with a focus on networked multiplayer game development. The objective was to build a modern version of the classic Battleship game while applying essential software engineering principles. The result is an interactive, grid-based multiplayer game featuring a clean, responsive UI with two 10×10 boards—one for the player and one for the opponent. Real-time gameplay and a reliable backend provide an engaging and seamless user experience.

Key Features

- **Interactive Grid UI**: Players interact with two 10×10 grids—one displaying their ships and the other showing hits and misses on the opponent's board.
- **Dynamic Ship Placement**: Players can place five different types of ships with rotation support and real-time feedback for valid or invalid placements.
- **Multiplayer Mode**: Players join a match using a shared game key. The game begins automatically once both participants are ready.
- **Combat System**:
    - Red tiles indicate successful hits; blue tiles show misses
    - Moves are updated in real-time with no need for manual refreshing
    - Turns switch automatically
    - The game detects when a player has won and displays a victory message
- **Robust Networking**:
    - Built on a client-server architecture
    - Automatic retry logic and timeout handling to improve reliability
    - Game state synchronization ensures consistency between both players

Technical Highlights

- **MVVM Architecture**: Clean separation of concerns using ViewModel and LiveData for reactive UI updates.
- **Kotlin Coroutines**: Efficient handling of background operations and network communication.
- **Server Communication**: Powered by Retrofit and long polling for near-instant updates.
- **User Experience Enhancements**:
    - Clear status messages throughout the game
    - Interactive elements are disabled when it's not the player's turn to prevent invalid actions
- **Session Management**: Games are automatically cleaned up after one hour of inactivity to free up resources.

Code Structure

Core Architecture

1. UI Layer (View)

MainActivity

- Acts as the central UI controller
- Hosts two game boards (your grid & opponent's grid)
- Handles user input for ship placement and shot selection
- Observes LiveData from the ViewModel and updates the UI accordingly

BattleshipBoardView

- Custom View component to render the 10×10 game grids
- Visualizes:
    - Ship placements
    - Hit and miss markers
    - Placement previews
- Handles touch input and layout logic for grid interaction

2. ViewModel Layer

BattleshipViewModel

- Core game logic handler and state machine
- Exposes LiveData for:
    - Game phase (SETUP → WAITING → PLAYING → FINISHED)
    - Player and enemy shot lists
    - Turn control and status text
    - Game over condition and sunk ships
- Controls network flow via the Repository
- Implements retries, polling, and error states

3. Repository Layer

BattleshipRepository

- Abstracts all server interactions
- Provides APIs for:
    - Joining a game
    - Firing a shot
    - Listening for opponent moves (long polling)
    - Ping/connection testing

- Handles HTTP requests, JSON parsing, and error responses

4. Model Layer

- GameState: Central state container for the game session
- Ship: Defines ship type, position, and orientation
- Position: Represents grid coordinates
- Shot, JoinGameRequest, FireResponse, etc.: Define API data contracts

Game Flow

1. Setup Phase

- Players input their ID and game key
- Ships are placed on the board with drag and rotate functionality
- Ship validation ensures no overlap or out-of-bound placements

2. Connection Phase

- Server waits for both players to join the same game key
- The game begins when both players are ready

3. Battle Phase

- Players take turns firing at the opponent's board
- The system provides immediate feedback (hit/miss/sunk)
- The turn alternates automatically

4. Victory

- The game ends when one player sinks all enemy ships
- The UI displays victory or defeat messages
- Option to restart the game

Data Flow

[User Action]

↓

[UI Layer] → [ViewModel] → [Repository] → [Server]

↑ ↓ ↑

[State Update] ← [Game Logic] ← [API Response]

- All updates are observed through LiveData, ensuring the UI reflects changes immediately and reactively.

Key Design Concepts

State Management

```
data class GameState(
  val phase: GamePhase,
  val playerShips: List<Ship>,
  val enemyShots: List<Position>,
  val playerHits: List<Position>,
  val playerMisses: List<Position>,
  val isMyTurn: Boolean,
  val isGameOver: Boolean
)
```

- Immutable state model updated via copy()
- Clear phase transitions and centralized game logic

Network Communication

- Retrofit with Moshi for HTTP and JSON
- Coroutine-based asynchronous execution
- Long-polling mechanism to await opponent moves
- Automatic retries with exponential backoff

Gameplay Logic

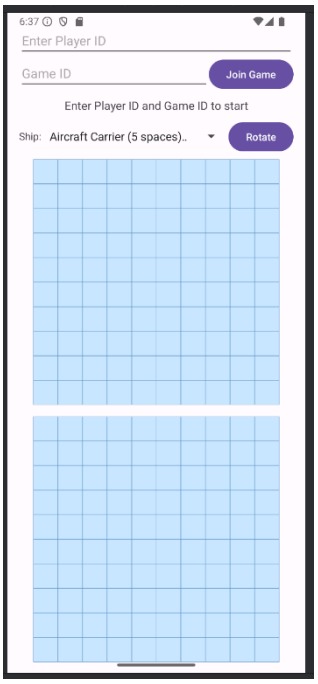
- Ship validation and placement preview
- Hit and miss tracking
- Ship sinking logic based on coordinate matching
- Win condition detection and state finalization

Error Handling

- Server-side errors parsed and reported via _error
- Timeout handling for polling and firing
- Retry logic with capped attempts
- UI feedback for invalid moves or disconnected opponents

Battleship Game - User Guide

Getting Started



1. Starting the Game

1. *Launch the Battleship app*
2. *You'll see two grids:*

- *Top grid: Your board*
- *Bottom grid: Opponent's board*

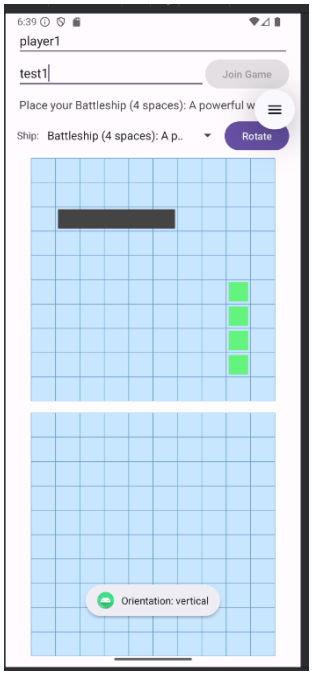
2. Joining a Game

1. *Enter your **Player ID** (minimum 3 characters)*
2. *Enter a **Game Key** (minimum 3 characters)*

- *If you're starting a new game, choose any key*
- *If joining an existing game, use the same key as your opponent*

3. ***Click "Join Game" button***

3. Ship Placement Phase



Available Ships:

- Aircraft Carrier (5 spaces)
- Battleship (4 spaces)
- Destroyer (3 spaces)
- Submarine (3 spaces)
- Patrol Boat (2 spaces)

Placing Ships:

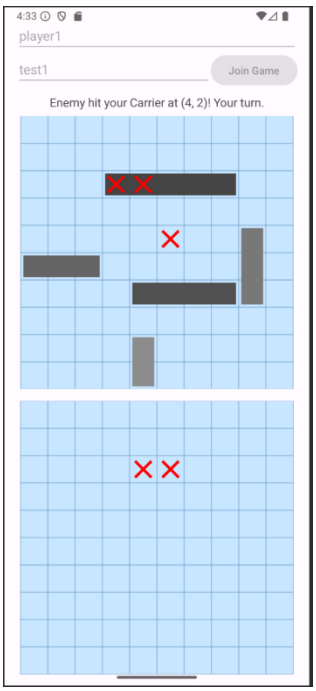
1. Ships must be placed in the order listed above
2. For each ship:

- Use the "Rotate" button to switch between horizontal/vertical orientation
- Hover over your board to see ship placement preview
- Green preview: Valid placement
- Red preview: Invalid placement
- Click to place the ship

3. After placing all ships, click "Start" to begin the game

Playing the Game

Turn Structure



1. The game randomly selects who goes first
2. Players alternate turns
3. On your turn:

- Click a cell on the opponent's board to fire
- Wait for hit/miss confirmation

Shot Indicators

- **Red X**: Hit on enemy ship
- **Blue Circle**: Missed shot
- Your opponent's shots appear on your board with the same indicators

Game Status

The status bar shows:

- Whose turn it is
- Hit/Miss results
- Ships sunk
- Game status messages
- Connection status

Game End Conditions

The game ends when:

1. All ships of one player are sunk
2. Connection is lost and cannot be restored after 5 retries
3. Game expires (after 1 hour of inactivity)

Troubleshooting

Connection Issues

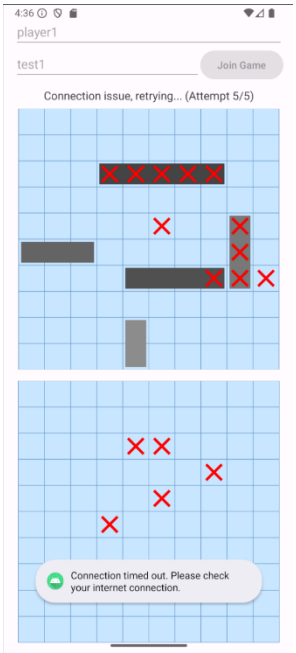
If you experience connection problems:

Check your internet connection

2. Ensure both players are using the same game key
3. Wait for automatic reconnection attempts
4. If problems persist:

- Close the app
- Restart with a new game key

Common Error Messages



- "Not your turn": Wait for opponent to complete their move
- "Already fired here": Choose a different target
- "Invalid placement": Ship is out of bounds or overlapping
- "Game not found": Wrong game key or game expired
- "Connection lost": Network issue, wait for reconnection

Best Practices

1. Use unique game keys to avoid conflicts
2. Place ships strategically:

- Avoid clustering ships
- Use board edges effectively

3. Keep track of your shots:

- Remember hit patterns
- Focus on likely ship locations

Review

We first tried to write our own code based on the Java files from the git repository. Pretty early in our endeavours we realised, that this project will be more challenging than expected. We started looking at the presentations and the lecture information but could not get a working copy. Afterwards we tried turning to AI with multiple calls within the group to check the functionalities and adjust based on the error messages. Our main AI tool we used was the program cursor AI with which we automatically could update our codes while in the call and testing live. While testing there were multiple Issues with the server connectivity so testing the game functionality has been hard. Even though we got our program to work few times we could not get the server connection to run reliably.

When we could not get to run our own program for a final test. Afterwards trying it with the java repository, the connection would fail as well. Other groups we've asked reported the same issue.

Our code to try getting our server connection is heavily influenced by the code shown in the lecture presentation. For our simultaneous work at our code, we used git with different branches when trying out things and pushed and pulled the codes to keep everyone up to date.