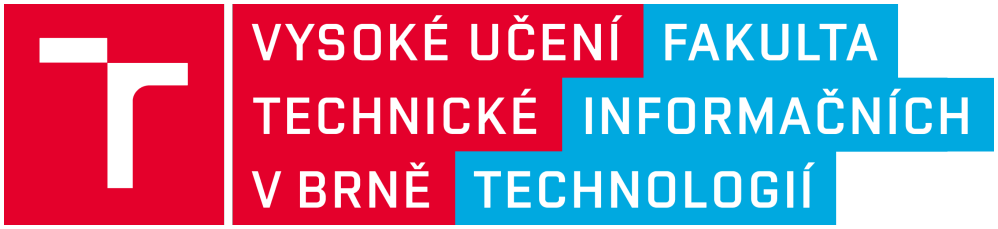

Battleships Game

December 14, 2025

Michal Repčík (xrepcim00)
Kristián Pribila (xpribik00)
Adam Veselý (xvesela00)
Martin Kandera (xkande00)

Contents

1 Project Goal and Assignment

The goal of this project was to implement a fully functional Battleships game with a graphical user interface as part of Phase II of the course assignment.

Each team member implemented their own frontend (FE) version of the application using a different technology stack, while sharing a common Node.js/Express backend provided by the project assignment.

Deviations from Phase I

Compared to the original plan described in Phase I, the following changes were made:

- Multiplayer mode was not included in the final implementation.
- The reason was the need for strict backend synchronization, which became impractical after individual FE-specific backend adjustments.
- The final scope therefore focuses on single-player gameplay against an AI opponent.
- Unified application design

All other functional requirements from Phase I were preserved.

2 Work Division – Final Frontend Implementation

Each team member implemented the same Battleships game logic and functionality, but using a different platform and technology stack. This section describes the individual frontend implementations.

Michal Repčík — Mobile App (C# + Avalonia)

- Implemented a complete mobile frontend for Android and iOS using the Avalonia framework, covering the entire gameplay flow from game setup and ship placement to match completion.
- Designed the application architecture using the MVVM pattern, with XAML-based Views, ViewModels managing observable state and commands, and a separate service layer responsible for communication with the backend.
- Connected UI elements to the backend REST API through a centralized ApiService layer, handling HTTP requests, JSON deserialization, and propagation of updated game state to bound UI components in real time.
- Implemented turn-based gameplay logic on the frontend side, including validation of user actions, enforcement of turn order, and synchronization of moves with the backend.
- Implemented interactive ship placement on a touch-based grid, allowing ships to be placed, removed, and repositioned using direct manipulation.
- Implemented ship rotation logic with explicit user control via UI buttons, ensuring predictable behavior on touch devices.
- Implemented state-driven UI updates so that changes in the game state (grid updates, ship placement, hits, and misses) are immediately reflected in the interface.

Adam Veselý — Desktop App (Rust + Tauri + Svelte)

- Full desktop application using Tauri framework (Rust with Svelte frontend).
- Implemented ship placement interface with click-to-place mechanics.
- Created interactive grid system with hover previews showing ship placement before confirmation.
- Implemented ship rotation (R key or right click).
- Created color-coded ship inventory with visual selection indicators.
- Integrated with Node.js backend via REST API.
- Implemented responsive grid sizing (7x7, 10x10, 13x13) with immediate updates when settings change.
- Added keyboard shortcuts for ship manipulation (R or right click for rotate, Delete/Backspace for remove).

Kristián Pribila — Web App (React + TypeScript + JavaScript)

- Component-Based Architecture with REST API: The application follows a client-server architecture with React component-based frontend communicating with backend through RESTful API endpoints, enabling separation of concerns and modular development.
- Interactive Ship Placement System: Implemented dual input methods including native HTML5 drag-and-drop functionality and click-based placement, allowing users to position ships on grids with rotation (R key) and deletion (D key) keyboard shortcuts for enhanced user experience.
- AI Strategy Implementation with Multiple Difficulty Levels: Developed three distinct AI opponents - Easy (random attacks), Hard (intelligent hunt mode with hint system providing one clue per ship), and Insane (cheating mode with full grid visibility) - featuring state management for strategic decision-making.
- Real-Time Game State Management: Utilized React hooks for local state management, Axios for asynchronous HTTP requests, and file-based JSON storage on backend for persistent game data, enabling smooth gameplay with instant visual feedback for hits, misses, and game outcomes.
- Implemented responsive grid sizing (7x7, 8x8, 9x9 10x10)

Martin Kandera — Web App (Phaser)

- Implemented a browser-based frontend using the **Phaser 3** game framework, focusing on a game-like, canvas-rendered user interface rather than traditional DOM-based layouts.
- Designed the entire application as a set of Phaser scenes, separating the game flow into logical states such as main menu, ship planning phase, and battle phase.
- Implemented an interactive ship placement system based on direct grid interaction, where users select ships from an inventory panel and place them onto the grid using visual preview markers indicating valid placement directions.
- Implemented real-time placement validation on the frontend side, ensuring that ships cannot overlap or exceed grid boundaries before placement is confirmed.
- Implemented undo and reset functionality for ship placement, allowing users to remove the last placed ship or reset the entire board prior to confirmation.

- Integrated the frontend with the shared Node.js backend using REST API calls, handling synchronization of the planning grid, placed ships, and game initialization.
- Implemented responsive layout scaling for different screen resolutions by dynamically computing grid and UI element sizes based on viewport dimensions.
- Focused on precise input handling and visual feedback (selection highlighting, preview indicators, disabled inventory items) to provide a clear and intuitive user experience despite the low-level rendering approach.

3 Incorporation of Feedback

During the control presentation of Phase I, no critical or negative feedback was received regarding the proposed application concept, user interface, or interaction design.

Based on this outcome, the primary goal during Phase II was to preserve the approved design and interaction patterns as closely as possible to the original prototypes. This approach was chosen deliberately in order to avoid unnecessary changes that could introduce new usability issues, inconsistencies between platforms, or deviations from the validated concept.

The implementation therefore focused on faithfully translating the designed prototypes into fully functional frontend applications while ensuring technical correctness, responsiveness, and consistent behavior across all platforms.

Minor adjustments were made only when required by technical constraints of specific frameworks or devices, or when usability issues were discovered during user testing. These changes did not alter the core interaction model or visual structure of the application.

4 Testing

Each team member tested their version of the application with a user from Phase I. The tests followed the required "think-aloud" methodology.

Below are the condensed summaries.

Michal Repčík — Mobile App Test

User Profile: Male, 13 years old, student from a game design class. The user is very technically skilled for his age, with prior experience in Lua, C#, and Roblox Studio.

Test Process: The usability test was conducted remotely using Chrome Remote Desktop. The user shared their screen and controlled the application while verbalizing their thoughts according to the think-aloud methodology.

The user was asked to complete the following tasks without any prior instructions:

- Start a new game
- Place, move, and rotate ships on the grid
- Prepare the board for gameplay

No guidance was provided unless the user became completely blocked.

Key Observations:

1. **Ship Rotation Discoverability:** The user did not immediately understand how to rotate a ship. When attempting to move a ship, the user often unintentionally rotated it instead, which caused confusion and interrupted the placement flow.

2. **Repositioning vs. Rotation Conflict:** The initial interaction design combined ship selection and rotation into a single tap action. This led to accidental rotations when the user's intention was to reposition an already placed ship.

Resulting Design Changes: Based on the observations, ship rotation was moved from direct interaction with the ship to a dedicated rotation button in the UI. Clicking a placed ship now selects it for repositioning instead of rotating it. This separation of actions improved usability and reduced accidental input during ship manipulation.

Adam Veselý — Desktop App Test

User Profile: Male, 22 years old, computer science student, regular gamer.

Test Process: The user was asked to perform the following tasks:

- Configure game settings and start a new game
- Place all ships on the grid
- Play through a complete game against the AI

Key Observations:

1. **Ship Selection Bug:** Initially, the ship selection mechanics were bugged. If the user selected ships in a certain sequence, they could have multiple ships selected at once, creating confusion as to which ship was being manipulated with.
2. **Grid Selection Bug:** Similarly to ship selection, grid selection was also bugged. If the user changed grid sizes in a certain sequence, the actual grid size would change only after clicking the Play button twice.

Both issues were resolved during development based on user feedback.

Kristián Pribila — Web App (React) Test

- Female, 22 years old, architecture student with a general foundation in technology use.
- Male, 19 years old, engineering student, casual computer user, gamer.

To validate the game's outcome and overall design, I utilized a diverse group of **respondents** for testing.

Preparation and Pre-Testing Phase

- **Consultation:** Based on thorough initial preparation and **extensive consultation** with potential players and target audience members, I developed a strong, clear vision of the core mechanics and features that needed to be implemented in the game.
- **Design Focus:** This preparatory phase allowed the design to focus heavily on player-centric elements, aiming for an **intuitive** and engaging experience from the start.

Usability Testing (Uninstructed Playthrough and think-loud methodology)

- **Methodology:** The final testing phase involved multiple subjects who were given **no prior instructions or guidance** regarding the game's controls, objectives, or mechanics. They were simply presented with the game and asked to play.The players verbally described what they saw on the screen, how they felt about the game

- **Key Finding:** A crucial finding was that **all participants** were able to intuitively grasp how to play the game without the need for additional tutorials, tooltips, or supplemental instructions. This directly validated the early design decisions stemming from the consultation phase.
- **Player Enjoyment:** Furthermore, the participants genuinely **enjoyed the game experience**, contributing to a positive outlook on the final product.

Conclusion

The testing process can be summarized as **highly successful** for the following reasons:

1. **Low Iteration Cost:** The positive results meant that only a **minimal number of changes** were necessary to finalize the game.
2. **High Usability Score:** The ability of uninstructed players to immediately understand and engage with the game confirms a high degree of **usability and intuitive design**.

Martin Kandera — Web App (Phaser) Test

User Profiles:

- Male, 22 years old, computer science student and regular PC gamer, with no prior knowledge of the project or its internal logic.
- Male, 23 years old, medical student, casual computer user, familiar with standard PC applications but without prior experience with similar games.

Test Process: The usability tests were conducted in a desktop web browser environment and followed the think-aloud methodology. Both users were asked to interact with the application without receiving any explanation of the ship placement mechanics.

The test focused primarily on the ship placement phase and included the following tasks:

- Select ships from the available inventory
- Place ships of different sizes on the grid
- Correctly orient ships in all possible directions

Key Observations (Initial Version):

1. **Manual Placement Was Not Intuitive:** In the initial implementation, ship placement required the user to infer the ship's orientation implicitly through repeated clicks. Both users struggled to predict where the ship would be placed and expressed uncertainty about the underlying placement logic.
2. **Unclear Direction Control:** Neither user immediately understood how to control the ship's orientation. The interaction required experimentation and trial-and-error, which slowed down the placement process and negatively affected the user experience.

Resulting Design Changes:

Based on the initial test results, the ship placement interaction model was redesigned:

- The placement process was simplified into a two-step interaction: the user first selects an anchor cell on the grid and then explicitly chooses the ship's orientation by selecting one of the displayed direction markers.

- This approach removed ambiguity from the placement process and made the orientation choice explicit and visually guided.

Re-testing and Final Evaluation:

After implementing the redesigned placement mechanism, the application was tested again with both users.

During the repeated testing phase, both subjects were able to place ships correctly without any additional instructions. The direction markers were immediately understood, and no confusion regarding ship orientation was observed.

Both users reacted positively to the revised interaction model and confirmed that the placement process felt predictable and intuitive. The redesign successfully eliminated the usability issues identified in the initial testing phase.

5 Summary

The final application meets all requirements of Phase II of the project assignment. All frontend implementations provide consistent gameplay, functional GUI interaction, and validated usability through user testing.