Application design and foundation assignment

Battleships Game

November 7, 2025

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

Contents

1 Application Overview

The project focuses on developing a modern Battleships application—a digital adaptation of the classic board game—designed for a wide range of players. Battleships Game is a cross-platform Battleships game where players place ships on a configurable grid (e,g. 7×7, 10×10) and battle an AI opponent. The frontend is split across different platforms using different programming languages and/or frameworks, all sharing the same Node.js/Express backend.

2 User Requirements

2.1 Target User

Our target users are middle/high school students or casual players who enjoy simple strategy games like Battleships or casual fan of board games who enjoys classic titles and their digital versions. They usually play on mobile phones and prefer quick matches, clean visuals, and minimal setup time.

2.2 Individual User Research

Michal Repčík

To better understand what kind of games players actually enjoy, I did a small survey and short interviews with 8 kids (ages 9–15) who regularly play mobile or browser games (mostly Roblox). These were mostly my own students from programming or game design courses, so the feedback was relaxed and direct.

What They Liked

Players preferred games that are:

- **Quick to start** — no long tutorials or loading screens.
- **Clear and colorful** — easy-to-read games with simple effects.
- **Rewarding** — hits and wins should feel satisfying with sound or animation.
- **Customizable** — being able to change or customize things adds variety.
- **Playable offline** — AI opponents are a must when there's no internet.
- **Short** — matches around 1–2 minutes fit their attention span.

What They Didn't Like

Most complaints were about:

- **Too many ads** — they ruin focus and pacing.

- **Predictable AI** — easy to exploit after a few rounds.
- **Fake multiplayer** — "online" matches that are actually bots.
- **Online-only modes** — can't play offline at all.

Kristián Pribila

I focused my research on players who had no experience with the online version of Battleships and played only the classic board version. Many players appreciate the idea of playing online, mainly for its convenience and availability at any time of day. They requested intuitive controls without unnecessary complications, visual aids that highlight permitted moves, and multiple game modes (against the computer or online against other players).

Martin Kandera

To better understand how players perceive simple web-based strategy games, I conducted several short interviews with university students aged between 18 and 24 who play browser or mobile games occasionally. The main goal was to explore their expectations regarding game design, control simplicity, and accessibility on different devices.

Interview Questions and Summarized Answers:

1. What kind of games do you usually play in your free time? Most respondents mentioned that they play short, casual games such as puzzles, card games, or simple strategy titles. They prefer games that can be started and finished quickly, often during study breaks or short free periods.

2. What makes you keep playing a game? Smooth and responsive controls, quick interactions, and clear visual or sound feedback were repeatedly mentioned as key factors. Players also appreciated when progress is easily visible (for example, animations after a hit or win). A minimalistic interface and dark, readable color palette were preferred.

3. What usually makes you stop playing a game? Respondents agreed that long loading times, confusing menus, or excessive advertisements are the main reasons to quit. Some also mentioned frustration with slow gameplay or unresponsive web interfaces on mobile devices.

4. Do you prefer online or offline play? Most respondents preferred an online browser-based version, mainly because it does not require installation and can be easily accessed from any device. However, they still appreciated the possibility to play against an AI opponent when no human players are available.

5. How important is visual design to you? Participants agreed that clarity and simplicity are more important than complex graphics. They prefer dark backgrounds, simple contrast between the grid and ships, and minimal effects that do not distract from gameplay.

6. Would you like to customize the game (grid size, ship colors, etc.)? Customization was considered an optional enhancement rather than a necessity. Respondents preferred a fast game start with sensible defaults over detailed setup before each match.

7. Would you rather use a web version or install a mobile app? Most participants preferred a browser-based version that is mobile-friendly. They emphasized the advantage of quick access without installation and smooth operation even on slower connections.

Summary of Key Insights:

- Players expect quick access and simple setup within a web browser.
- The interface must be responsive and optimized for both desktop and mobile devices.
- Clear feedback (hit/miss animation, sound) increases engagement.
- Clean and minimal design with dark theme and visible contrast is preferred.
- AI mode is desirable as an alternative to online multiplayer.
- Customization options add value but should not delay gameplay.

Conclusion: Based on the collected feedback, users value accessibility, responsiveness, and clarity. They prefer online games that are lightweight, quick to start, and visually consistent across different devices. The interface should remain simple, intuitive, and easy to navigate. Including an option to play against an AI opponent and maintaining clear visual feedback for player actions were identified as key factors for a positive user experience.

Adam Veselý

I asked casual gamers to describe their ideal Battleships app. They all shared the desire to have a minimalistic strategy game with no crazy colors, simple controls and a responsive interface. They also made it very clear that advertisements were not welcome. No in-app purchases either. Offline mode was requested as well. They also want the game to start quickly, no long loading screens.

Summary

- One window. Two grids. Nothing else.
- Black background, dark colors, easier on the eyes. No rainbow.
- Click = shoot. No drag, no right-click menu.
- Start button → game. No login, no setting, no long loading screen.
- Works offline — AI opponent required.
- Real time UI updates.
- No ads or in-app purchases.
- Minimalistic, easy and quick to play game.

2.3 Summary of Key User Requirements

The application should let users access the game quickly without complicated settings. Controls must be comfortable and intuitive.

- Be quick and easy to start playing.
- Offer an offline mode.
- Have simple but colorful graphics and clear hit/miss effects.
- Allow customization.
- Avoid intrusive ads or unnecessary menus.
- Optionally track stats (accuracy, wins, streaks, etc.).

3 Research on Existing Battleship Games

To put this list together, we looked at several different sources — mostly Google Play reviews, Reddit discussions, and YouTube gameplay videos where players shared their experiences. This helped to get a feel for what people liked or disliked about each game.

By combining online feedback with our own short play-testing, we got a good overview of what features work well and what players usually complain about.

3.1 Sea Battle (Byril)

Author: Byril
Downloads: 10M+ (Google Play)
Rating: 4.7 (750K reviews)

Overview: Classic Battleship with online multiplayer, Bluetooth play, AI opponent, and customizable themes.

Positives:

- Engaging online multiplayer
- Hand-drawn graphics
- Free with optional IAP
- Offline AI mode
- Regular updates

Negatives:

- Intrusive ads
- Occasional multiplayer bugs

- IAP may feel necessary
- Limited customization

Alternatives:

App Store: Sea Battle Online

Desktop: Emulatable (MEmu, LDPlayer)

Web: Playhop / Yandex Games

3.2 Sea Battle 2: Warship Online (Byril)

Author: Byril
Downloads: 50M+ (Google Play)
Rating: 4.5 (1.58M reviews)

Overview: Sequel adding city-building, arenas, ranks, and customizable fleets to the core Battleship gameplay.

Positives:

- City-building & arenas
- Vibrant hand-drawn style
- Real-time online PvP
- Frequent content updates
- Offline AI mode

Negatives:

- Heavy ads & IAP
- Unbalanced matchmaking
- High battery usage
- Occasional server lag

Alternatives:

App Store: Sea Battle 2

Desktop: Emulatable

Web: Yandex Games / Playgama

3.3 Fleet Battle (Smuttlewerk Interactive)

Author: Smuttlewerk Interactive
Downloads: 10M+ (Google Play)
Rating: 4.6 (216K reviews)

Overview: Modern Battleship with high-quality graphics, ranked multiplayer, chat, and varied AI difficulty.

Positives:

- Excellent graphics & sound
- Ranked & private matches
- In-game chat and leaderboards
- Regular balance updates

Negatives:

- Frequent ads
- Matchmaking delays
- Cosmetic IAP
- Rare crashes on older devices

Alternatives:

App Store: Fleet Battle: Sea Battle Game

Desktop: Google Play Games for PC

Web: `smuttlewerk.de/fleetbattlegame`

3.4 Battleship: The Board Game (Marmalade Game Studios)

Author: Marmalade Game Studio
Downloads: 100K+ (Google Play)
Rating: 3.6 (3.12K reviews)

Overview: Official Hasbro adaptation with Classic and Commanders (special abilities) modes and online multiplayer.

Positives:

- Faithful to the original board game
- Commanders mode with abilities
- Cross-platform multiplayer

- High-quality 3D visuals
- Family-friendly

Negatives:

- Paywall for extra content
- No salvo mode
- Multiplayer bugs
- Higher price

Alternatives:

App Store: BATTLESHIP

Desktop: Steam (Hasbro's BATTLESHIP)

Web: None

3.5 Warships Online (Coolplay GmbH)

Author: Coolplay GmbH
Downloads: 100K+ (Google Play)
Rating: 4.7 (2.1K reviews)

Overview: Real-time naval PvP with chat, friends list, leaderboards, and three AI difficulty levels.

Positives:

- Sleek, modern UI
- Chat & friends system
- Quick matchmaking
- Free with optional ads
- Frequent stability updates

Negatives:

- Intrusive ads
- Occasional crashes
- Limited ship variety
- Small player base

Alternatives:

App Store: Warships Online

Desktop: Emulatable only

Web: None

3.6 Battleships (Joe Blake B)

Author: Joe Blake B (Joe Baker)
Downloads: 100+ (Google Play)
Rating: Not rated

Overview: Minimal single-player Battleship vs AI, built in Kotlin.

Positives:

- Lightweight, no ads/IAP
- Saves state on rotation
- Free & partly open-source
- Quick offline play

Negatives:

- No multiplayer
- Basic graphics
- Single AI opponent
- Few updates

Alternatives:

App Store: None

Desktop: Source on `joeblakeb.com`

Web: None

3.7 BattleShip Pro (Luigi Ferioli)

Author: Luigi Ferioli
Downloads: 10000+ (Microsoft Store)
Rating: 4.1/5 (83 reviews)

Overview: Single-player and LAN multi-player Battleship app.

Positives:

- Lightweight

- Free to download and play
- Quick offline play
- LAN multiplayer

Negatives:

- No online matchmaking
- Few updates
- Can be considered too bright

Alternatives:

App Store: None

Web: None

3.8 BattleShip Online (myadzel)

Author: myadzel
Users: 40000+ (Chrome Web Store)
Rating: 4.6/5 (1.4k reviews)

Overview: Online multi-player Battleship website.

Positives:

- Lightweight
- Minimalistic graphics
- Free to play
- No downloading required
- Quick online play
- Play with friends by sending a link
- Ability to create tournaments

Negatives:

- No offline mode
- Graphics can be considered too simple
- Can be considered too bright
- Website contains ads
- No usernames, difficult to stay acquainted with players

Alternatives:

App Store: None

Desktop: None

3.9 Functional Requirements

- Player can place ships on a grid of configurable size (7×7, 10×10).
- Game supports turn-based battle against AI opponent.
- Visual feedback for hits and misses.
- Restart / reset option for a new match.

3.10 Non-Functional Requirements

- Works on mobile and desktop platforms.
- Fast startup time (2 s) and minimal network dependency.
- Simple UI suitable for touch and mouse input.

Work Division

Each member implements the **same functional GUI** for a **different target device** using a **different programming language or framework**.

- Michal Repčík – Mobile App (Android, iOS) - C# + Avalonia
- Adam Veselý - Dekstop App (Windows) - Rust + Tauri + Svelte
- Kristián Pribila - Web App (Windows) - React
- Maritn Kandera - Web App (mobile devices supported) - Phaser

All frontends consume the **same Node.js/Express backend** (provided TypeScript server).

4 GUI Design

4.1 Mobile Interface (Michal Repčík)

Figma design: Click here to view the Figma prototype

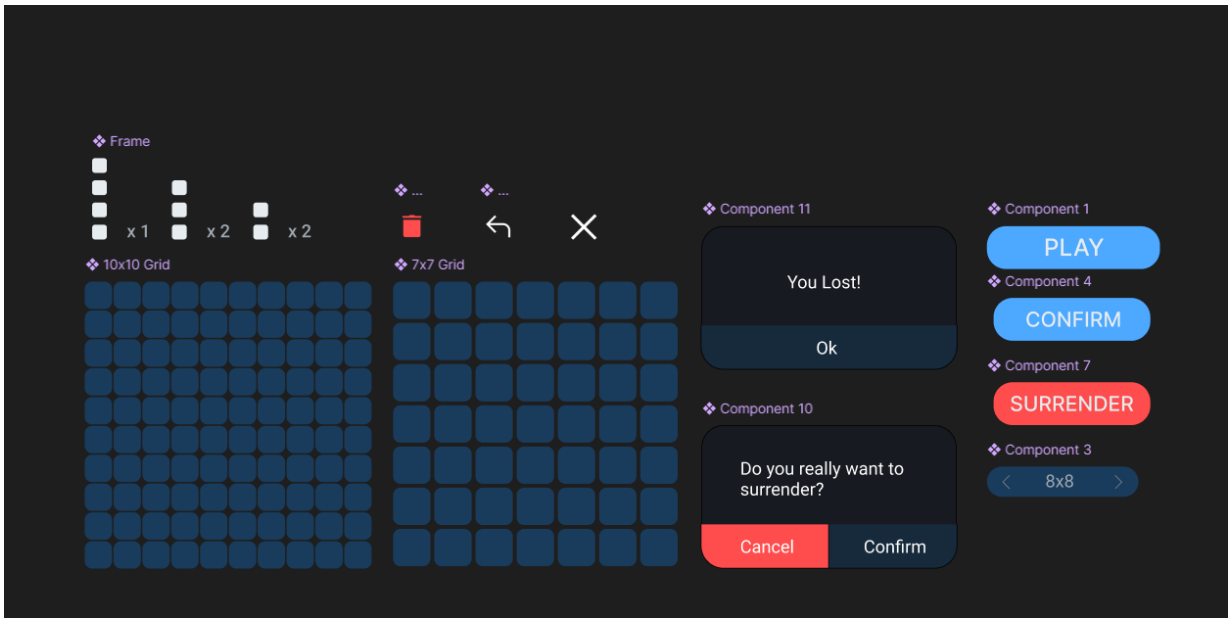
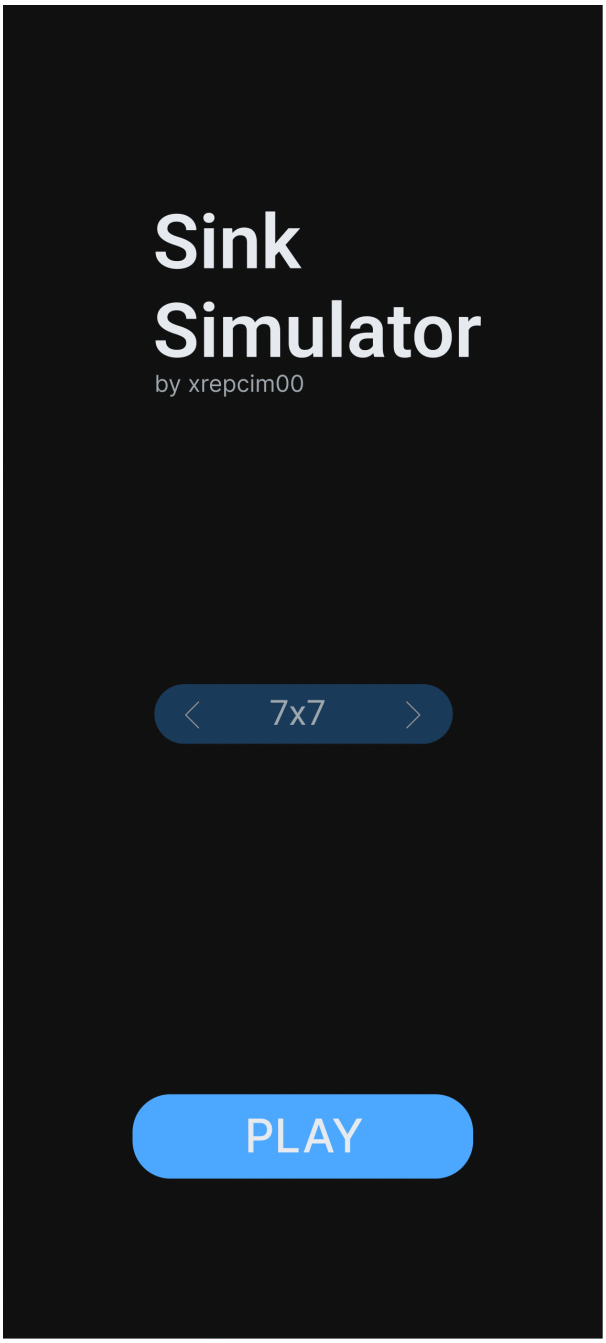
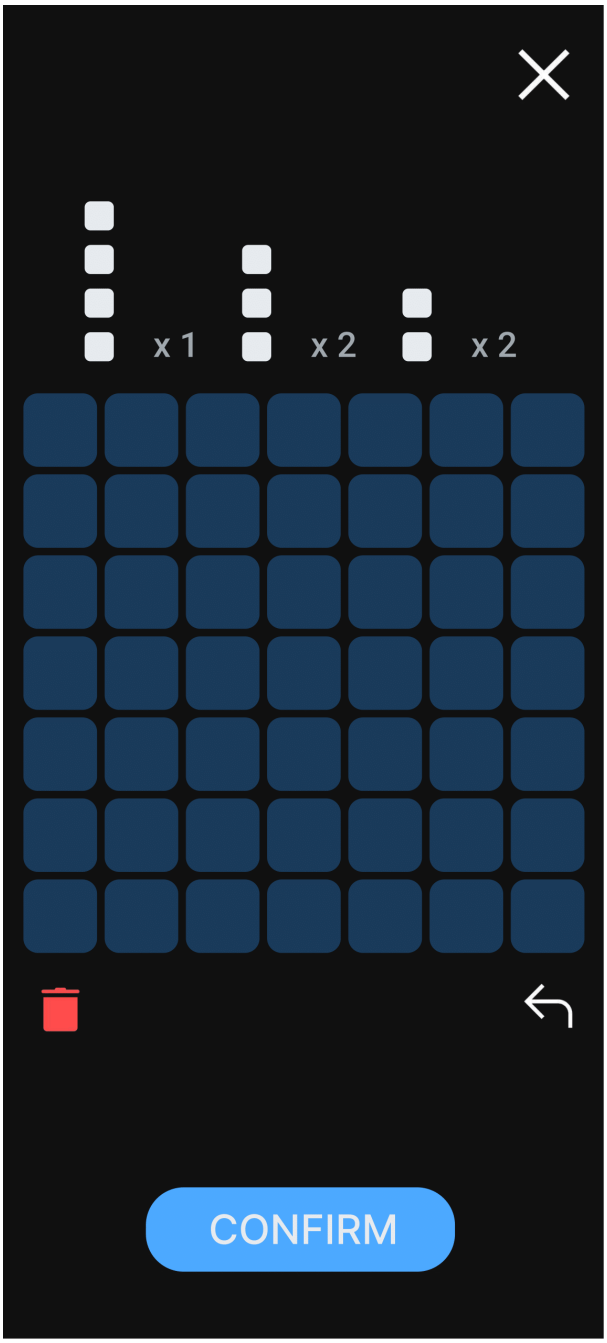


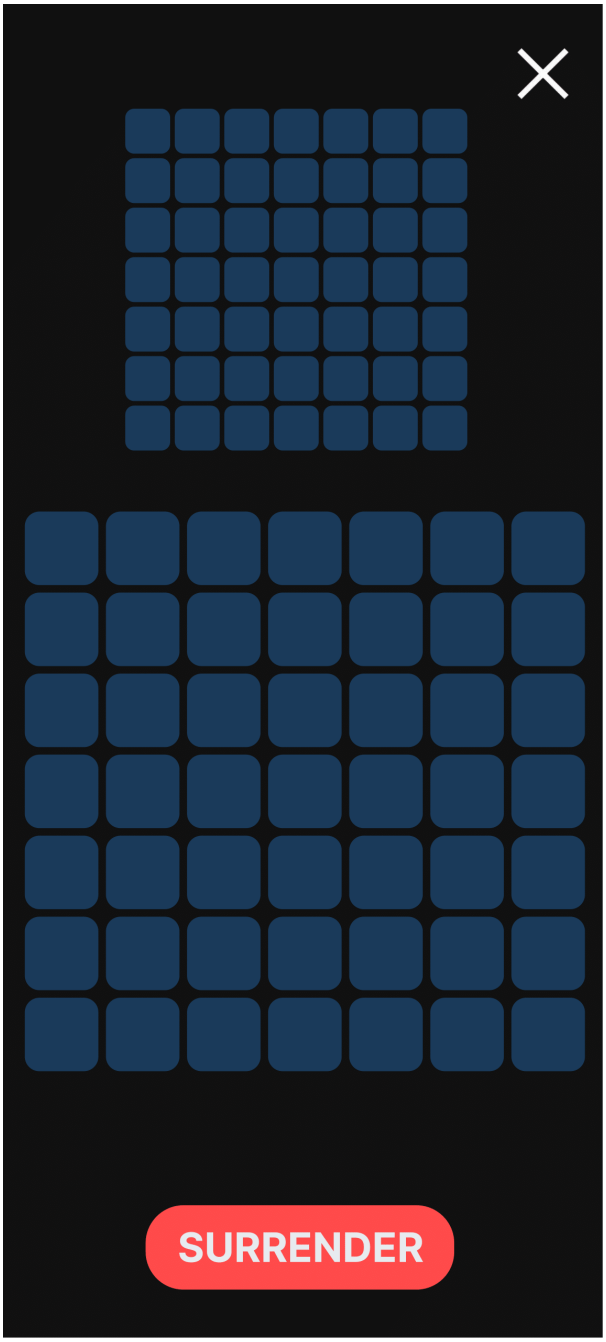
Figure 1: A — Components overview used in the UI design.



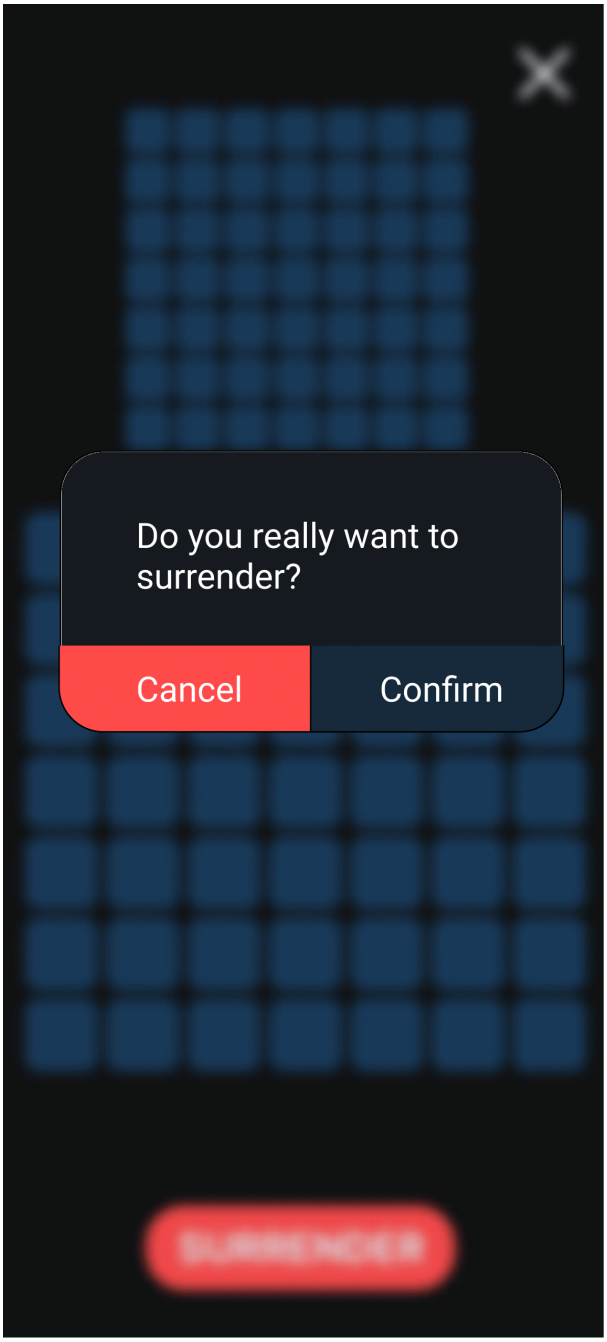
(a) B — Screen 1



(b) C — Screen 2



(a) D — Screen 3



(b) E — Screen 3 with popup

4.2 Web (Kristián Pribila)

Figma design: Click here to view the Figma prototype

Figure 4: Main page

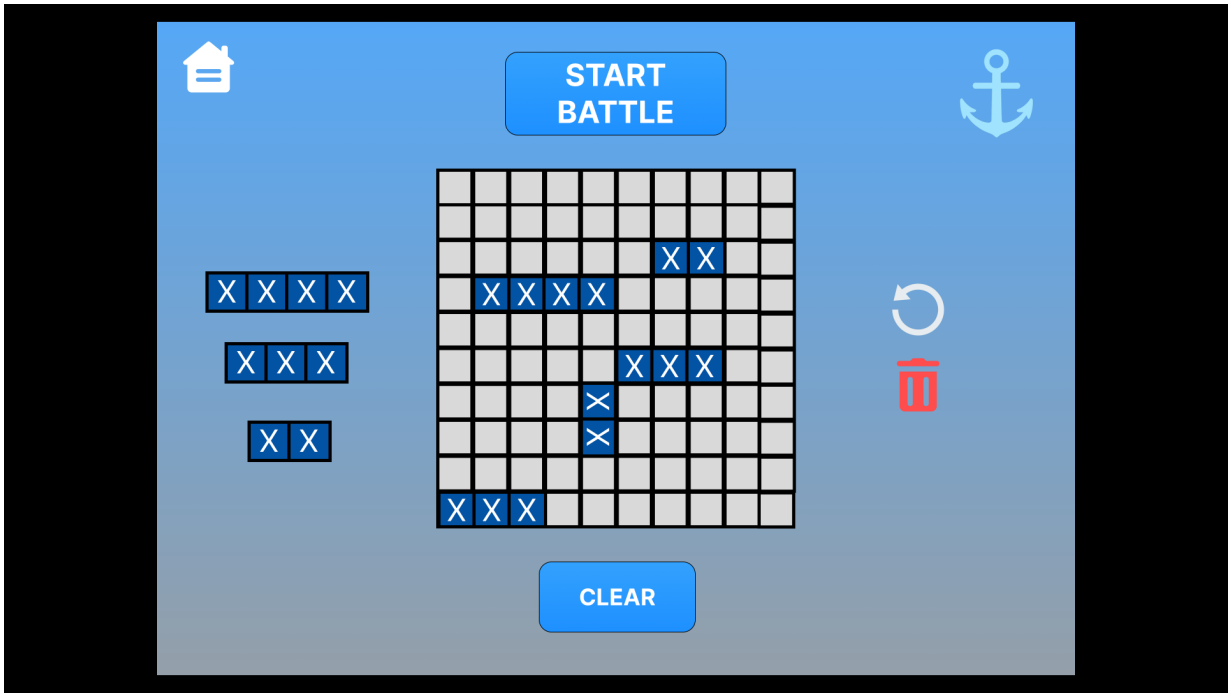


Figure 5: Planning page.

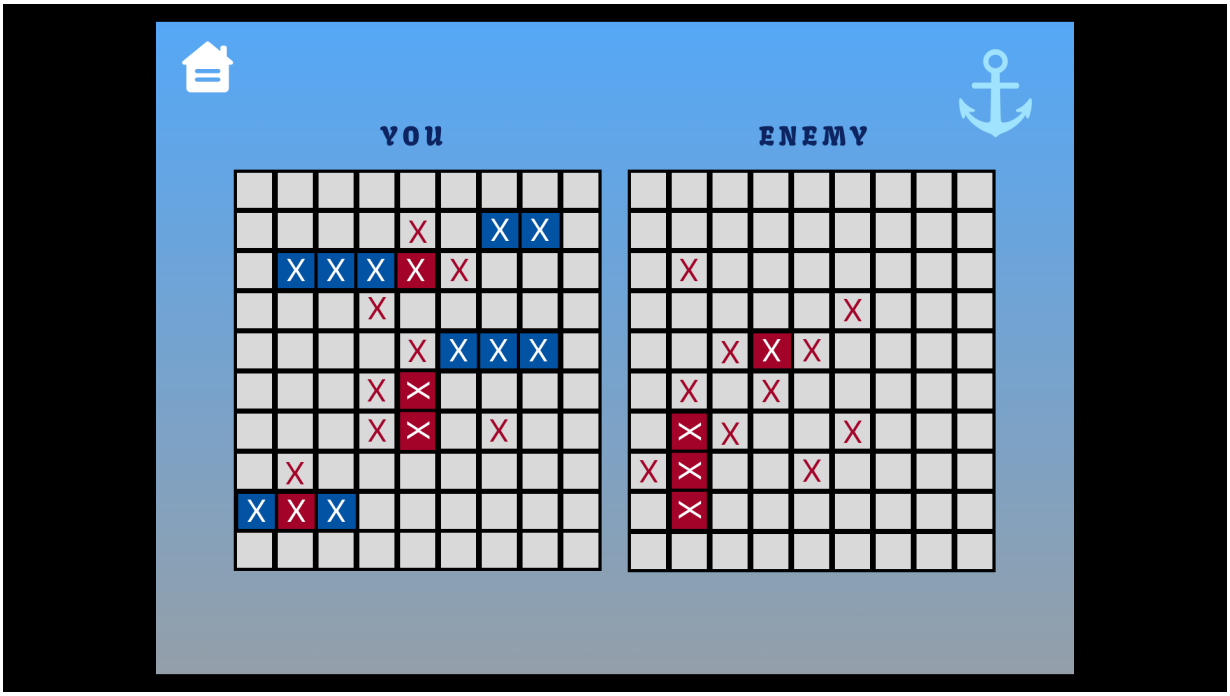


Figure 6: Game page

4.3 Desktop App (Adam Veselý)

Figma design: Click here to view the Figma prototype

Figure 7: Main page

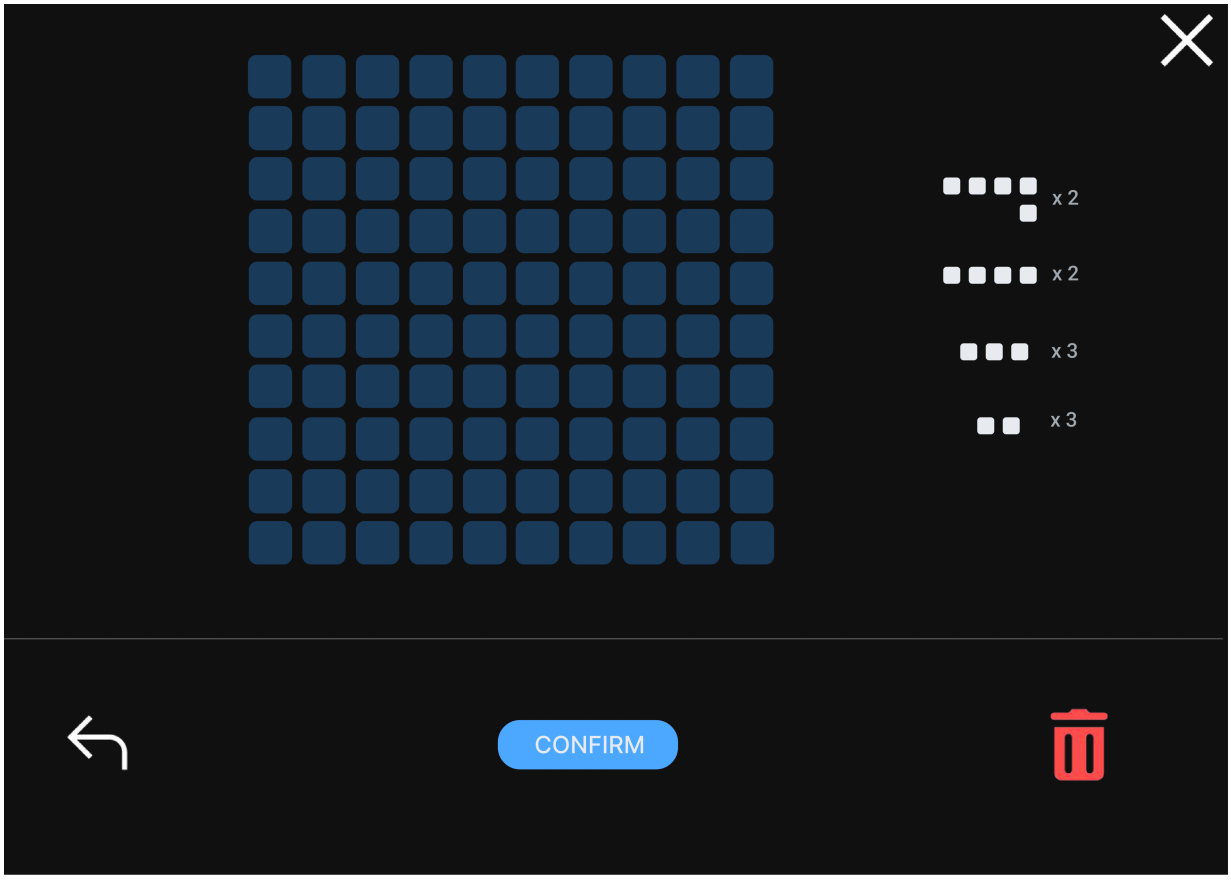


Figure 8: Planning page.

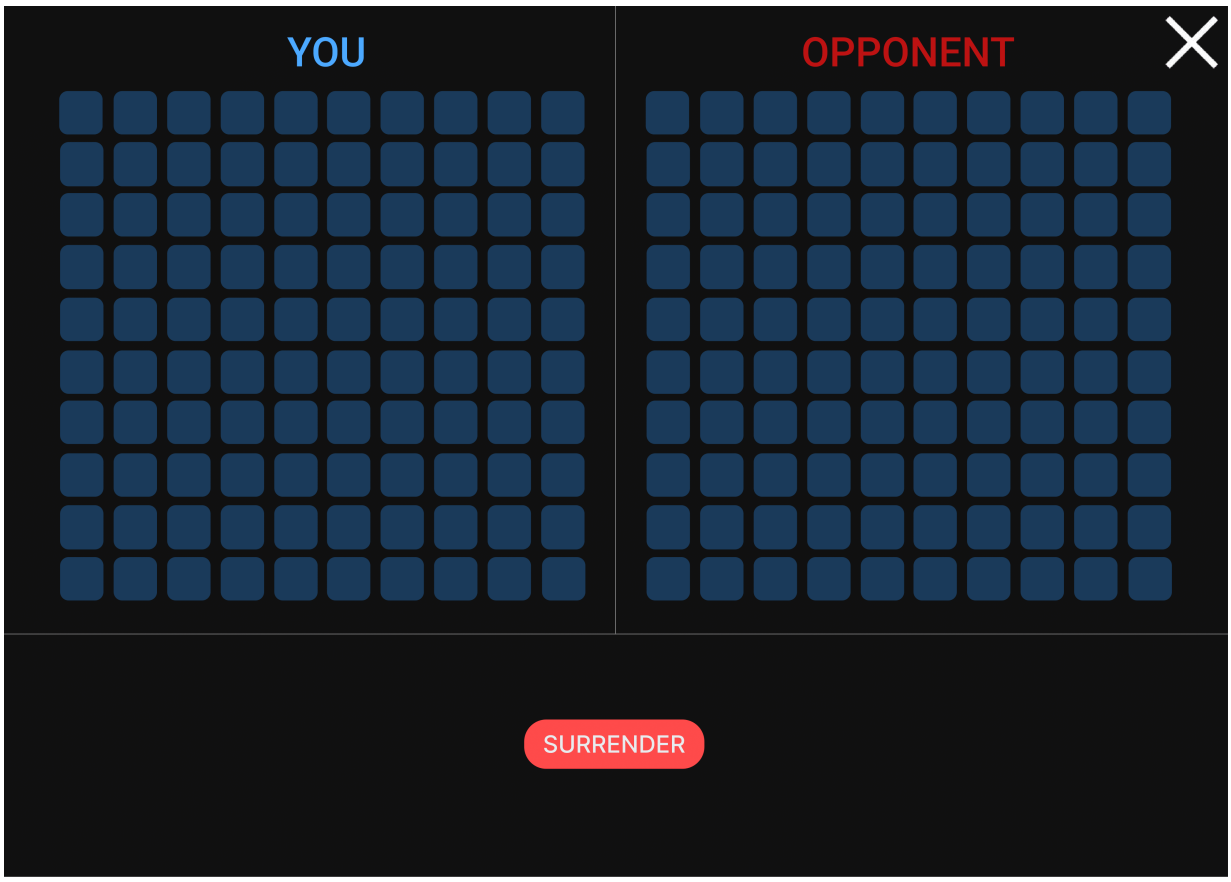


Figure 9: Game page

4.4 Web app (Martin Kandera)

Figma design: Click here to view the Figma prototype

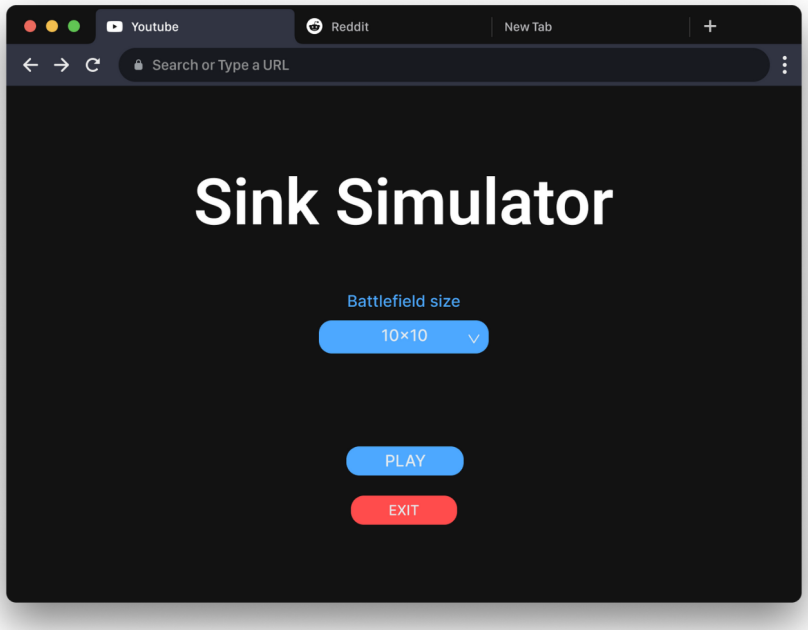


Figure 10: Main page (PC/tablet)

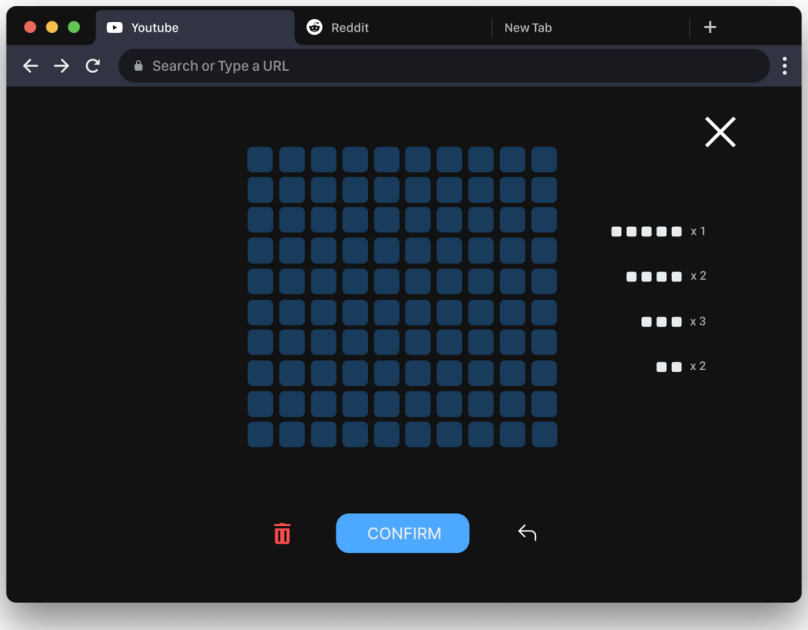


Figure 11: Planning page (PC/tablet)

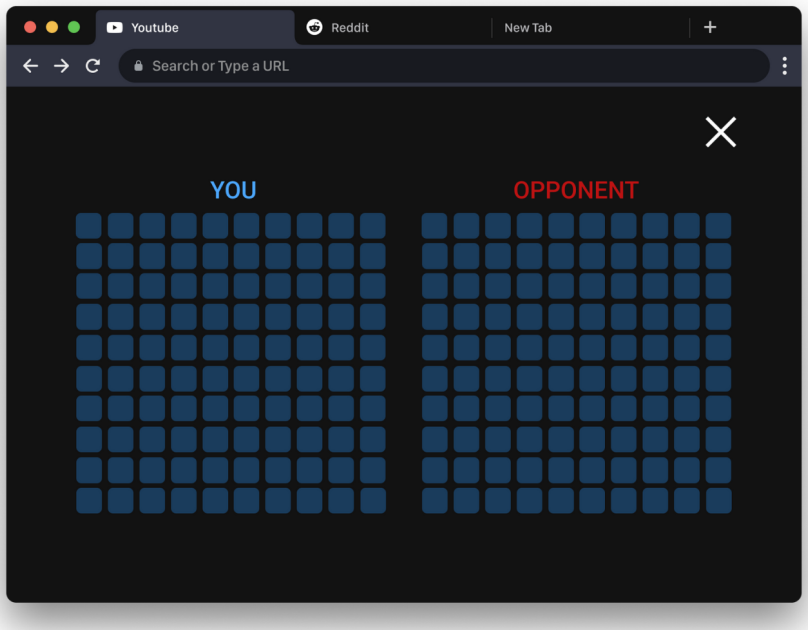


Figure 12: Game page (PC/tablet)

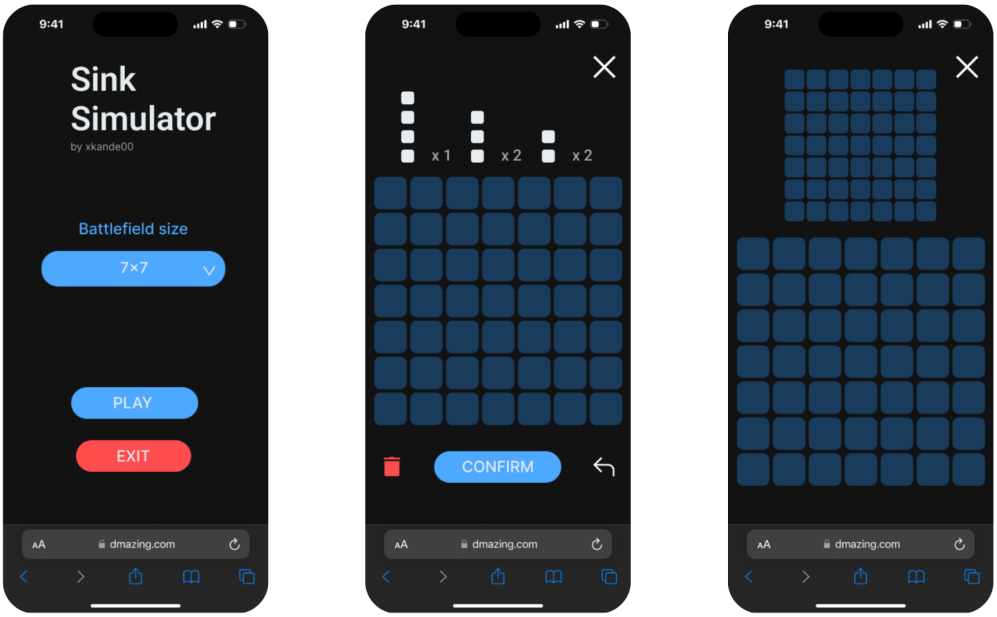


Figure 13: Mobile web app

5 API Design

Backend: Node.js + Express (REST, JSON file persistence).

5.1 Key data structures (TypeScript)

```
interface IShip {
    id:string;
    size:number;
    color:string;
    rotation:0|90;
    name:string;
}
interface IPlacedShip extends IShip {
    row:number;
    col:number;
}
interface Grid {
    gridSize:number;
    tiles:string[][];
}
```

5.2 Core endpoints

- `GET /api/screen` - current screen name
- `POST /api/screen` {current_screen}
- `GET /api/player-grid`, `POST /api/player-grid`
- `GET /api/pc-grid`, `POST /api/pc-grid`
- `GET /api/planning` - full planning state
- `POST /api/planning/placed-ships` - place/move
- `POST /api/planning/rotate-active-ship`
- `POST /api/planning/clear-grid`

6 Frontend - Backend Communication

6.1 Architecure Patterns

6.1.1 MVVM (C# + Avalonia)

MVVM (Model-View-ViewModel)

- **View** – Avalonia XAML
- **ViewModel** – `ObservableObject` with `RelayCommand`
- **Service** – `ApiService.cs` (singleton HTTP client)

Download Data (Pull)

```
var data = await _api.GetPlanningDataAsync();
PlayerGrid = data.player_grid;
AvailableShips = data.available_ships;
```

Binds to grid UI.

Upload Data (Push)

```
await _api.AddPlacedShipAsync(ship with { Row = row, Col = col });
```

Server updates `planning.json`.

Dynamic Interaction

```
[RelayCommand]
private async Task Attack(int row, int col)
{
    var grid = await _api.UpdateGridCellAsync(row, col, "pc", PcGrid
       );
    PcGrid = grid;
    if (_api.CheckForWin(grid)) ShowVictory();
}
```

Click → API → UI update in real time.

6.1.2 Tauri + Svelte (Reactive Declarative)

- **View + ViewModel** – `App.svelte` file
- **Model** – plain JS objects (no classes)
- **Service** – `fetch()` (native)

Download Data (Pull)

```
let planning = await fetch("http://localhost:5000/api/planning")
                    .then(r => r.json());

$: playerGrid  = planning.player_grid;
$: shipsLeft   = planning.available_ships;
$: activeShip  = planning.active_ship;
```

Svelte redraws the second any line changes.

Upload Data (Push) – one ship

```
await fetch("http://localhost:5000/api/planning/placed-ships", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ ship, row, col, rotated })
});
```

Server writes `planning.json` instantly.

Upload Data (Push) – rotate or clear

```
await fetch("http://localhost:5000/api/planning/rotate-active-ship",
    {method:"POST"});
await fetch("http://localhost:5000/api/planning/clear-grid",
            {method:"POST"});
```

Dynamic Interaction

```
async function attack(row: number, col: number) {
  if (!myTurn) return;
  const res = await fetch("http://localhost:5000/api/pc-grid", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ row, col })
  });
  const grid = await res.json();
  opponentGrid = grid;
  myTurn = (await (await fetch("/api/screen")).json()) === "player-
     turn";
  if (grid.flat().filter(c=>c==="ship").length === 0)
    alert("Victory!");
}
```

Click → API → real-time UI update.

Screen flow

```
let screen = await (await fetch("/api/screen")).text();
$: document.title = "Battleship     " + screen;
```

6.2 Cross-Platform Consistency

All four frontends share the **same** `ApiService` contract:

- Identical method signatures.
- JSON deserialization to shared POCOs.

All frontends use identical `ApiService` interface and POCOs. Only XAML and input handling differ.