🎮 **Task 7: Sea Battle Game**

**⚓ Welcome to the Sea Battle Code Challenge**

You are provided with a simple, **legacy-style CLI implementation** of the classic **Sea Battle (Battleship)** game: [seabattle.js](https://ventionteamsinc-my.sharepoint.com/:u:/g/personal/mikita_sauko_ventionteams_com/EfprvfrdHwdEm6es4CmTPUIB5i_Iybo-pefDdSbDw_GZmA?e=ajQOPT).  
This version is functional but written using **older JavaScript conventions** (e.g., ES5 var, global variables, etc.).

**🎯 The existing game features:**

* A **10x10 grid**
* **Random ship placement** for both player and CPU
* **Turn-based gameplay** with coordinate input (e.g., 00, 34)
* A basic CPU opponent with **"hunt"** and **"target"** modes
* **Text-based display** of the player's and opponent’s boards

📄 For more details on how the Sea Battle CLI app works, refer to [seabattle.js](https://ventionteamsinc-my.sharepoint.com/:u:/g/personal/mikita_sauko_ventionteams_com/EfprvfrdHwdEm6es4CmTPUIB5i_Iybo-pefDdSbDw_GZmA?e=tRg8ho) and [README.md.](https://ventionteamsinc-my.sharepoint.com/:t:/g/personal/mikita_sauko_ventionteams_com/EfTnAjuJvtlFidXBxjeOXbEB5GrJcZu4pPImsqHdc06Spw?e=gQC3bb)

**📚 Theory**

**🧠 Cursor IDE for Refactoring**

Cursor’s AI helps you **understand and modernize codebases** while keeping behavior intact. It:

* Analyzes complex logic
* Suggests modernizations
* Explains reasoning behind each suggestion
* Supports **separation of concerns** and better **maintainability**

**💡 AI Techniques Used**

* **Understanding Legacy Code**: Ask AI to explain seabattle.js structure and logic before refactoring.
* **Code Modernization & Refactoring**:
  + Upgrade from **ES5 to ES6+** syntax
  + Refactor using **classes**, **modules**, **arrow functions**, and let/const
  + Improve structure (e.g., separate game logic, UI, and utilities)
  + Optionally, **translate** into another language (e.g., Python, TypeScript, Go)
* **Test Generation**: Use AI to generate tests for game states and CPU behaviors.
* **Architectural Guidance**: Discuss possible designs (e.g., **MVC pattern**, component separation).
* **Maintaining Functionality**: Ensure the core mechanics remain unchanged.

**🛠️ Task**

Your challenge is to **modernize and refactor** [**this codebase.**](https://ventionteamsinc-my.sharepoint.com/:u:/g/personal/mikita_sauko_ventionteams_com/EfprvfrdHwdEm6es4CmTPUIB5i_Iybo-pefDdSbDw_GZmA?e=4tP51S) ([seabattle.js](https://ventionteamsinc-my.sharepoint.com/:u:/g/personal/mikita_sauko_ventionteams_com/EfprvfrdHwdEm6es4CmTPUIB5i_Iybo-pefDdSbDw_GZmA?e=4tP51S))

**✅ Objectives**

**1. Modernize & Refactor the Codebase**

* Use **modern ECMAScript (ES6+)** features:
  + Classes, modules, let/const, arrow functions, async/await, etc.
* Or rewrite in a **language of your choice** (e.g., Python, Java, TypeScript)
* Improve structure with:
  + Clear separation of concerns
  + Encapsulation of state and logic
  + Elimination of global variables
* Maintain original functionality:
  + 10x10 grid
  + Turn-based input (e.g., 00, 34)
  + Hit/miss/sunk logic
  + CPU's "hunt" and "target" behavior

**2. Add Unit Tests**

* Implement tests to cover core logic
* Use an appropriate framework (e.g., **Jest** for JS, **Pytest** for Python)
* Ensure **at least 60% test coverage** across core modules

**📌 Requirements**

* The refactored game must fully implement the **original mechanics and rules**
* The code must show:
  + **Clear structure**
  + Use of **modern language features** (ES6+ or equivalent)
* Unit tests must:
  + Cover **core functionality**
  + Achieve **minimum 60% test coverage**