Battleship Game

High-Level Design

Overview

Here is the simplest version of Battleship game, that is available as a one player against computer game.

The game consists of two parts:

- **Frontend:** A ReactJS based interface which shows the game board, and sees the results and allows users to shoot.
- **Backend**: A. NET 6 Web API used to orchestrate game logic, including ship placement, shot outcomes, and game state.

System Architecture

- **Frontend (ReactJS UI):**

  This UI consist of 10×10 game grid (dynamically drawn using backend response), here player was able to shoot by clicking on the cells. Depending on whether or not the shot hits a ship, it communicates back with the backend to render our grid. Moreover, the frontend shows game statistics such as remaining hits, sunk ships and win or lost status.

- **Backend (C# Web API):**

  The Backend is primarily responsible for returning important part of the game such as handling the core game logic. The API facilitates to initializes the game, random ship placements on the game grid, processing the player's hits, and returning the game status.

Key Components

- **Frontend (React):**
  - Shows the game board, and manages player input.
  - Request the current game state (e.g. the grid layout and ship locations) from the backend.
  - Report the outcome of each shot they attempt (hit, miss or sink). Update the relevant spots on your board.

- **Backend (C# API):**
    - **Game Grid Initializing:** The server initially creates a 10*10 game grid.
    - **Ship Positioning:** The server initially places lucky ships (1 x 5 squares = battleship and 2 x 4 squares = destroyers) on the board.
    - **Game Logic:** When the player hits the shot fired by the player is checked by the backend if hit or miss or the ship sunk; The player has 20 hits only to play. This will also know of the game state which persists and sends updates back to the frontend.

Game Flow

- **Click on "Start New Game":** The user clicks on start game at the frontend. The backend starts a new game by randomly placing ships on the grid and sending the initial state back to the frontend.
- **Player Fire**: When a shot is fired, the frontend sends coordinates to the backend to check if it hit or missed a ship. Comes back to the frontend which refreshes the display with new game state.
- **Winning/losing the game:** To win the game the player must sink all the ships within 20 hits. Otherwise, the player will lose the game.

Technologies Used

- **Frontend:** ReactJS is used for UI interaction and HTML, CSS for layout and styles.
- **Backend:** C# NET 6 Web API to handle game logic and API endpoint.
- **Hosting:** IIS for hosting the Application.

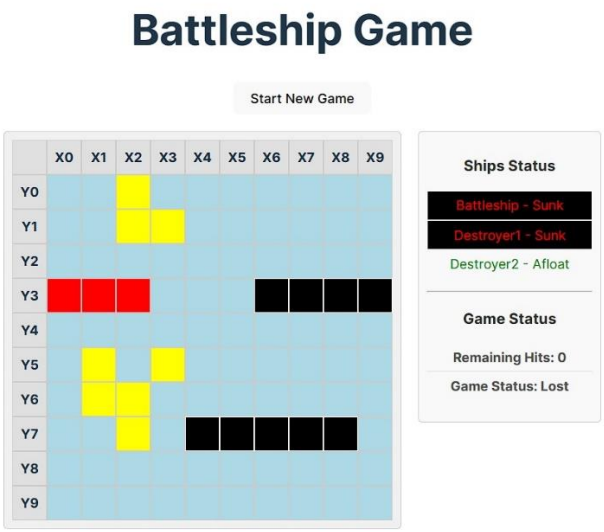

Figure 01