**Requirements Specification**

**Functional Requirements**

1. Grid-based Game

* Game takes place on a grid
* Each cell on the grid can be empty, have Christopher Columbus ship, pirate ship, island, treasure or sea monster

1. Player

* Player controls Columbus ship, starting at top-left cell (0,0).
* Columbus moves up, down, left, right with buttons or keys.
* Columbus cannot move off the grid, or onto an island.

1. Islands

* Islands are placed at random positions on grid.
* Columbus ship and pirate ships cannot move onto islands.

1. Pirate Ships

* There are different types of pirate ships (slow/fast).
* They are placed at random positions on grid.
* Pirate Ships chase Columbus using a movement strategy.
* If Columbus moves onto a pirate ship, or vice versa, game ends (Columbus ship hijacked).

1. Sea Monsters

* Sea monsters are placed randomly on grid.
* They move randomly on a grid or can move with a strategy.
* If Columbus ship moves onto a sea monster, or vice versa, Columbus is sent back to starting point (cell [0,0]).

1. Treasure

* Treasure is placed randomly on the grid.
* Columbus wins if he reaches the treasure before being caught by pirates.

1. User Interface

* Game has a welcome/start page, having a play button.
* Main grid represents all entities.
* Game page has buttons to play the game.
* Notification appears when collision with pirate ship, monster or treasure happens, showing corresponding message.

1. Game State Management

* The backend manages all entity positions and handles move requests.
* Game state is fetched from backend and rendered in frontend.
* All logic for movement, collisions, win/lose conditions, happens on server.

1. API Endpoints

* POST /api/start – Resets/starts the game
* POST /api/move(direction) – Moves Columbus ship and updates state
* GET /api/state – Returns current state of game (positions of all entities)

1. No Invalid Moves

* No invalid moves can happen.
* Columbus cannot move onto an island or off the grid.
* Pirate ships and monsters avoid moving onto islands, and each other.
* Any invalid move onto an island should show an alert.

**Non-Functional Requirements**

1. UI should be visually clear, friendly and responsive.
2. Code should be modular, using at least five design patterns.
3. Code should be easy to extend.
4. System should handle errors gracefully without crashing