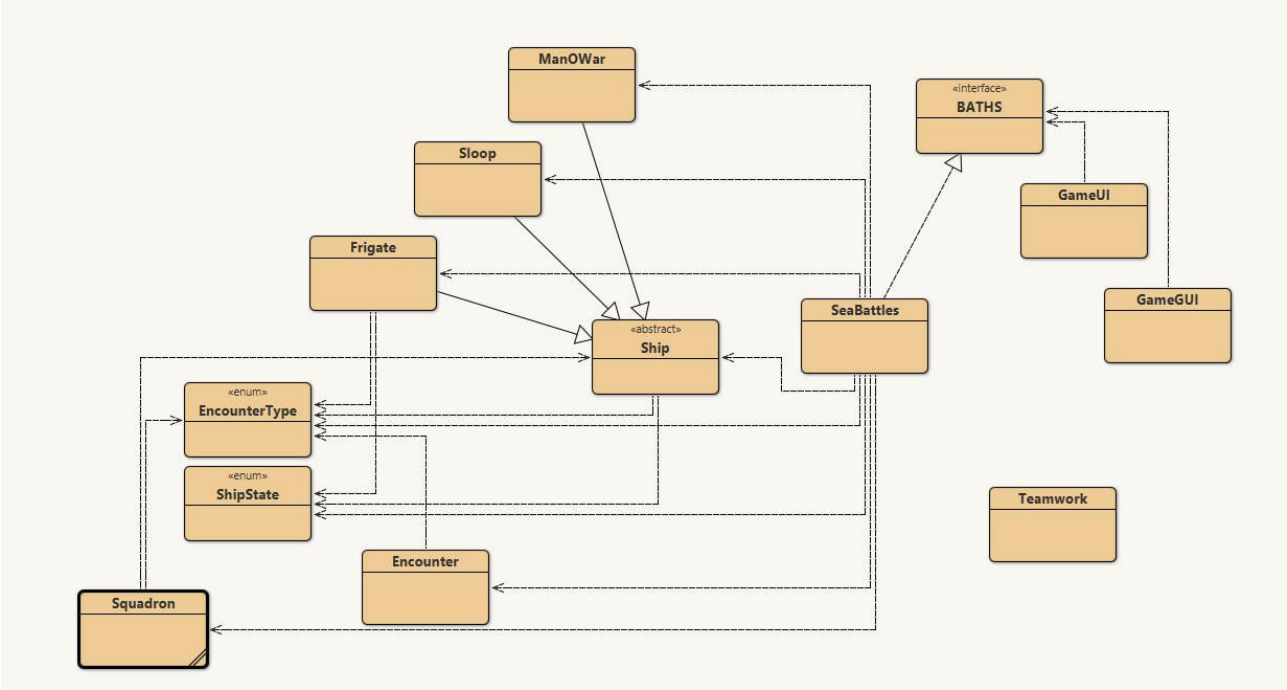
Technical Report CM03

1. Project Overview

The SeaBattles class represents a facade class that manages all our back-end classes. GameUI and GameGUI classes call methods from SeaBattles ultimately allowing to run the game and give the opportunity to test either via terminal or graphics interface. In the game itself by the requirements the following ships have been added as classes Frigates, ManOWars, and Sloops, with respective functionality defined in the case study, following all the requirements. In this project we have tried to follow low coupling and high cohesion paradigms and hopefully we have succeeded.

2. UML Diagram



The UML diagram includes:

- Abstract class Ship with the following subclasses: Frigate, ManOWar, and Sloop.
- Central facade class SeaBattles interacting with every other class combining functionality.
- Enums: ShipState and EncounterType for managing state and combat logic in the back-end.
- GUI interfaces: GameUI, GameGUI, and interface BATHS, which defines methods for the SeaBattles.
- Supporting class: Squadron to group ships logically for the admiral's commissioned ships.

3. Design Decisions

Decision 1: Abstract Class **Ship** with Subclasses

- **Design Decision:**
  The Ship class is abstract, with Frigate, ManOWar, and Sloop as its subclasses. They have their specific logic implementation, so that is why they have been subdivided into different classes.
- **Alternatives:**
  Use a single **Ship** class for ship with condition-based, applied in different methods, behavior for ship types.
- **Pros:**
  - Polymorphism allows us iteratively to assign any type of ship (Sloop, ManOWar, Frigate) to the Ship type objects/variables in the SeaBattles.
  - Improves clarity and maintainability.
- **Cons:**
  - More classes to manage.

Decision 2: **Adding a "Fight" Button and "Decommission Ship" to the GUI**

- **Design Decision:**
  The **GUI** includes a "Fight" button that triggers an encounter and a button that allows players to decommission ships.
- **Alternatives:**
  - Use keyboard shortcuts to initiate fights.
  - Use Terminal or GameUI
- **Pros:**
  - Adds readability compared to the terminal interface provided in the GameUI
- **Cons:**
  - Requires event handlers that are triggered by the buttons and respective calls, and this adds overhead functionality and responsibility to check more edge cases and handle errors.

Decision 3: High Cohesion and Low Coupling

- **Design Decision:**
  The architecture was designed with a strong focus on high cohesion within classes and low coupling between components. For the Project maintainability as per the case study introduction, which would also allow future modifications and scalability.

- **Alternatives:**
    - Design components with broader responsibilities (low cohesion).
    - Allow more direct interaction between unrelated classes (tight coupling).
    - Use global variables or shared states for communication.

- **Pros:**
    - Improves modularity and maintainability.
    - Easier to debug, test, and extend functionality.
    - Facilitates future refactoring or component reuse.
- **Cons:**
    - Slightly more upfront planning and interface design needed.
    - Can lead to more classes or abstraction layers.
- **Rationale:**
  Following object-oriented best practices ensured that each class had a single, well-defined purpose and interacted with others via clear, minimal interfaces. This led to a cleaner, more scalable codebase that can evolve as new features (such as multiplayer or new ship types) are added.

4. Summary

Summing everything up the essential design and architectural decisions were considered and implemented in the project, allowing the application to have readable codebase with possible future scalability. Everything has been organized to the main Facade class SeaBattles where all the calls to the respective classes are being carried out. The UML diagram is the representative of those decisions and can be viewed in the 2.UML Diagram section above in the report. Project gave us an opportunity to work and manage and allocate tasks for each other, ultimately leading us to the result seen in the report and codebase. The introductions above also improved our code, structuring it and making interface more appealing and more usable. Each of these decisions have been examined with alternatives, and best implementations have been chosen.Overall, the project represents balanced and thoughtful design, cooperative work, and priority to develop solid gameplay experiences.