UML Design for Battleship Game in C#

Jennifer Vicentes

Homework 2: Object-Oriented Programming

1 Introduction

This report explains the design and justification of the UML class diagram created for a Battleship game written in C#. The game is designed to place ships on a grid and allow a player (or players) to guess the locations of those ships, aiming to sink them. The UML diagram reflects the interactions and relationships between various classes and their respective responsibilities as derived from the provided code. For this class diagram, I used the website PlantUML, and you can check the code I used in the *Class Diagram.txt* file.

2 UML Class Diagram

Below is the UML class diagram representing the structure of the Battleship game:

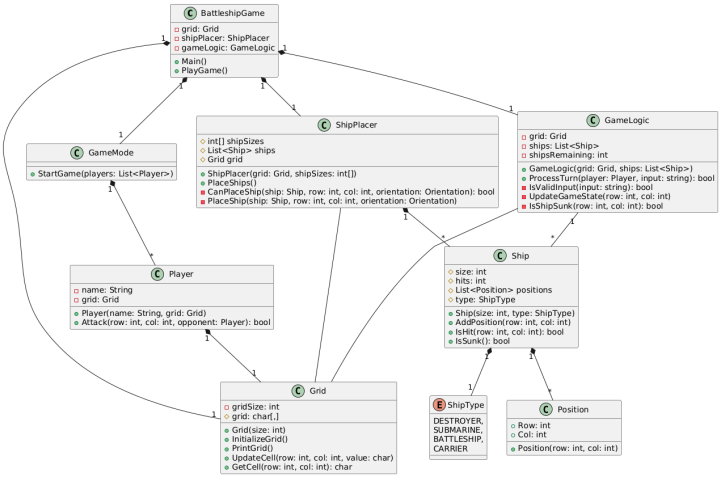

Figure 1: UML Class Diagram for the Battleship Game

3 Explanation of the UML Diagram

The UML diagram consists of several classes that collaborate to implement the core functionality of the Battleship game. Each class has a specific role, following the principles of object-oriented design, such as encapsulation and separation of concerns.

3.1 BattleshipGame Class

The **BattleshipGame** class is the main entry point for the game. It contains references to the **Grid**, **ShipPlacer**, and **GameLogic** classes. The methods `Main()` and `PlayGame()` are responsible for initiating the game and controlling the gameplay loop.

3.2 Grid Class

The **Grid** class manages the game's grid where ships are placed. It stores the grid size and grid data in a 2D array of characters. This class provides methods for initializing the grid, printing it, updating cells, and retrieving specific cell values.

3.3 Ship Class

The **Ship** class represents individual ships. It stores the ship's size, number of hits, and a list of **Position** objects, which denote the ship's location on the grid. Each ship has a type, defined by the **ShipType** enum, and methods to handle ship positioning and damage tracking.

3.4 ShipType Enum

The **ShipType** enum defines predefined ship types such as `DESTROYER`, `SUBMARINE`, `BATTLESHIP`, and `CARRIER`. These ship types are assigned to ships and help categorize them by size and shape, emulating the standard rules of the Battleship game.

3.5 ShipPlacer Class

The **ShipPlacer** class is responsible for placing ships on the grid. It ensures that ships are placed validly and do not overlap. This class contains methods for checking if a ship can be placed at a given position and for placing the ship either horizontally or vertically.

3.6 GameLogic Class

The **GameLogic** class handles the core mechanics of the game, such as processing player turns, validating input, updating the game state, and checking

whether a ship is sunk. It interacts with both the **Grid** and **Ship** classes to track hits and ship statuses.

3.7 Player Class

The **Player** class represents each player in the game. Each player has a name and their own grid. The class includes methods for attacking the opponent's grid, taking in a row and column for the attack, and interacting with the game logic to determine the outcome.

3.8 GameMode Class

The **GameMode** class determines how the game is played, whether in single-player or two-player mode. The class initializes the game for one or more players, ensuring that each player has their own grid and ships.

3.9 Position Class

The **Position** class represents the coordinates (row and column) of each part of a ship on the grid. It is used by the **Ship** class to track where each ship is located.

4 Class Relationships

The diagram includes various relationships between classes:

- **Composition**: The **BattleshipGame** class has a composition relationship with the **Grid**, **ShipPlacer**, and **GameLogic** classes, indicating that these objects are essential components of the game.
- **Aggregation**: The **GameMode** class aggregates multiple **Player** objects, indicating that each game mode can support one or more players.
- **Association**: The **ShipPlacer** and **GameLogic** classes both associate with the **Grid** class, showing that they interact with the grid during gameplay.
- **Dependency**: The **Ship** class depends on the **ShipType** enum for determining the type of ship it represents.

5 Access Modifiers and Symbol Explanation

In the UML diagram, different symbols are used to denote the visibility of attributes and methods:

- **Public (+)**: Denotes public visibility, meaning the field or method is accessible from any other class.

- **Private (-)**: Denotes private visibility, meaning the field or method is only accessible within the class it is declared.
- **Protected (#)**: Denotes protected visibility, meaning the field or method is accessible within the class it is declared and its subclasses.

6 Conclusion

The UML diagram effectively models the structure of the Battleship game, illustrating how the various classes interact to manage the game's logic, grid, and ship placement. By adhering to the principles of object-oriented design, including encapsulation and separation of concerns, this diagram provides a clear and maintainable structure for the game's codebase.