# SOEN 6441 - Advanced Programming Practices

## Winter 2024

## Warzone Risk Game

## Team 11

**Submitted by**

**Tania Sanjid (40255010)**
**Tanzina Nasrin (40235506)**
**Khandaker Rifah Tasnia (40276078)**
**Masum Newaz (40292704)**
**Shehzar Aurangzeb Abbasi (40291795)**

# Architectural Design Document for Warzone Risk Game

## 1. Introduction

This document outlines the architectural design for the Warzone game project, a strategy game inspired by the classic board game Warzone Risk. This document presents an architectural design for the Warzone Risk Game, building upon the previous MVC-based architecture to include the State, Command, and Observer design patterns. These additions aim to improve system flexibility, facilitate easier maintenance, and support more scalable development by allowing for well-defined behavior encapsulation, command abstraction, and decoupled state management.

## 2. System Overview

The Warzone Risk game allows players to engage in strategic territorial conquests. It features a dynamic map editor for creating, modifying, and playing on custom maps. Core gameplay mechanics include deploying armies, attacking territories, and fortifying positions to achieve global domination.

## 3. High-Level Architecture

The application's architecture integrates the State, Command, and Observer patterns alongside the foundational MVC (Model-View-Controller) architecture, further dividing the system into cohesive modules for improved scalability and maintainability.

**Model-View-Controller (MVC) Pattern**

The system is structured around the MVC architecture, segregating the application into three interconnected components:

**Model:** Manages the game data and logic, including entities such as maps, continents, countries, players, orders, and game phases.

**View:** Presents data to the user and gathers user input, with classes like MapView and PlayerView.

**Controller:** Handles user input, interacts with the model to update the game state, and updates the view. It includes the GameEngine for overall game flow and MapEditor for map editing functionalities.

**State Pattern:** Used to manage game phases (Startup, IssueOrder, ExecuteOrder), allowing the game to change its behavior when its state changes.

**Command Pattern:** Implemented through various Order classes (e.g., DeployOrder, AdvanceOrder), encapsulating all information needed to perform actions or trigger events.

**Observer Pattern:** Applied in the logging system, where LogEntryBuffer (Observable) notifies LogEntryWriter (Observer) of changes to log game actions.

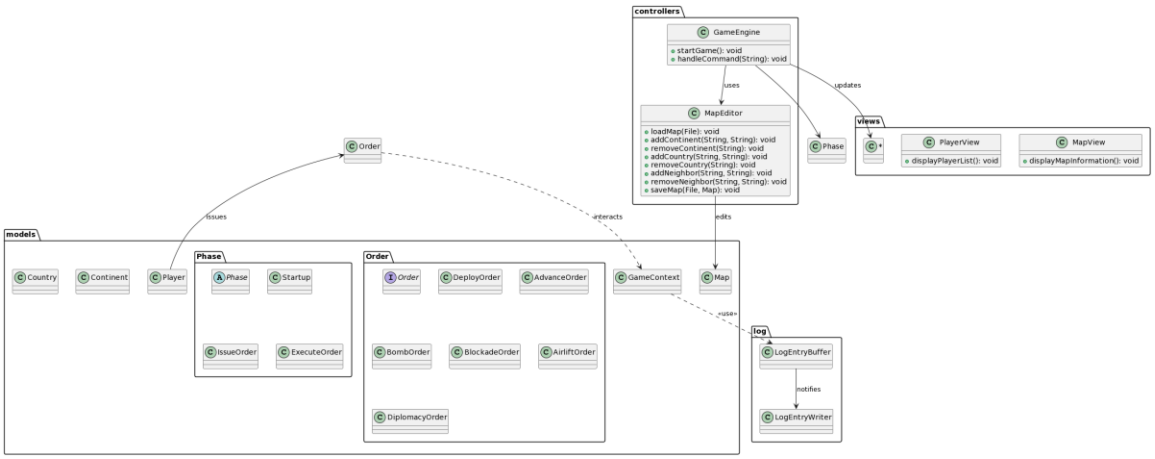


**Fig 1: Architectural Diagram of Warzone**

## Design Pattern Integration

**State Pattern:** Facilitates the transition between different game states without tight coupling, managed by a central game controller that triggers state changes based on gameplay progress.

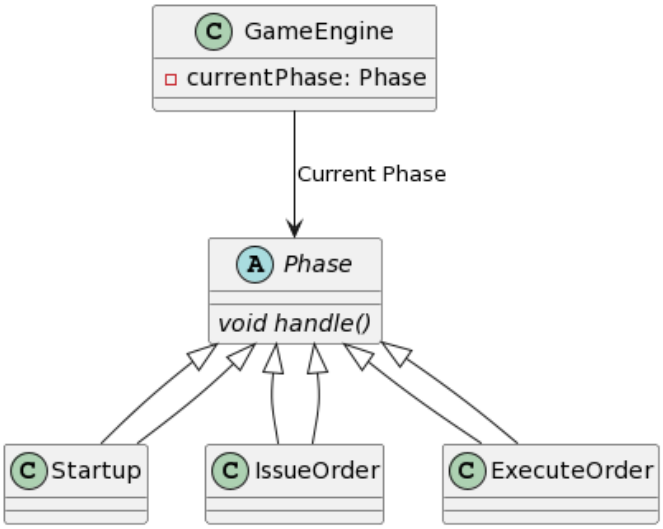


**Fig 2: State Pattern Diagram**

**Command Pattern:** Centralizes game commands (e.g., attack, move, deploy) into executable objects, managed by a command processor that handles execution and undo functionalities, enhancing modularity and testability.

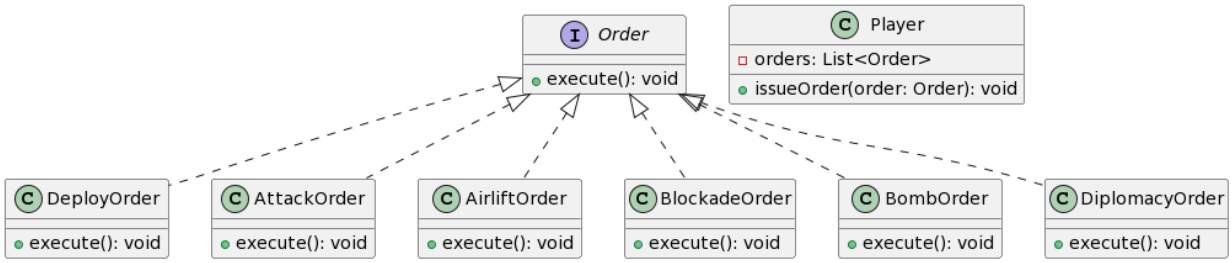


**Fig 3: Command Pattern Diagram**

**Observer Pattern:** Enables dynamic update mechanisms for game views and logs, where game entities notify observers of state changes, adhering to the "publish-subscribe" model for loose coupling and high cohesion.

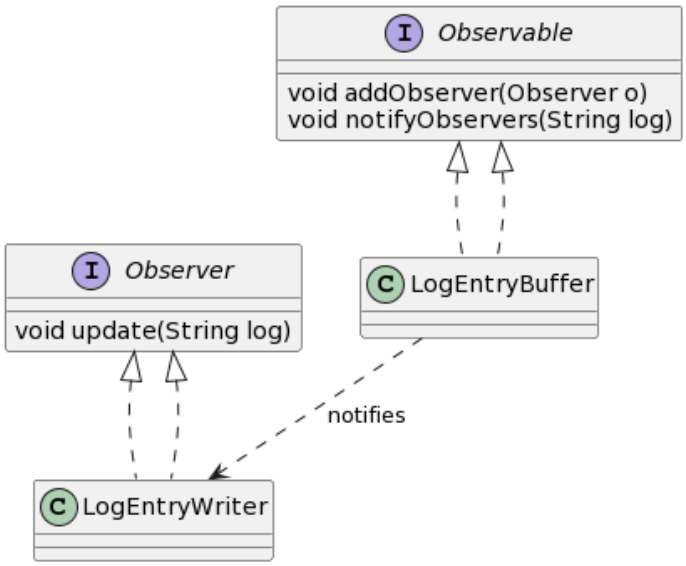


**Fig 4: Observer Pattern Diagram**

## 4.1 Class Diagram

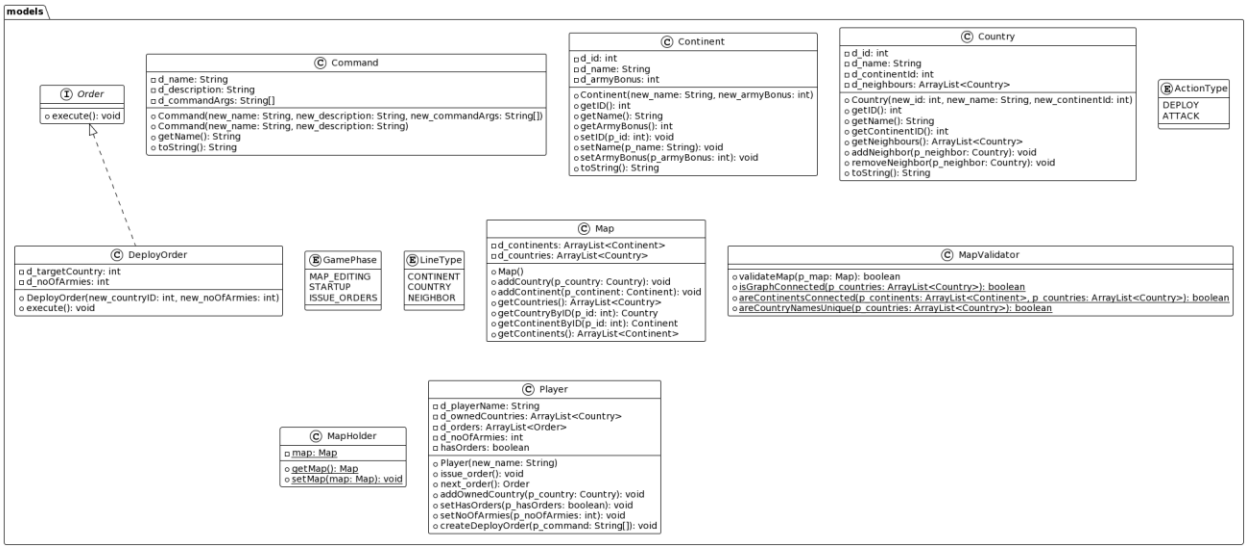


**Fig 5: Models Package Overview**

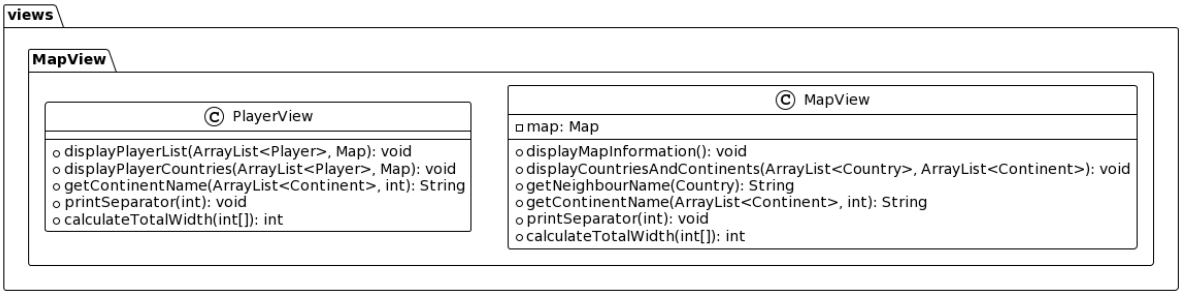


**Fig 6: View Package Overview**

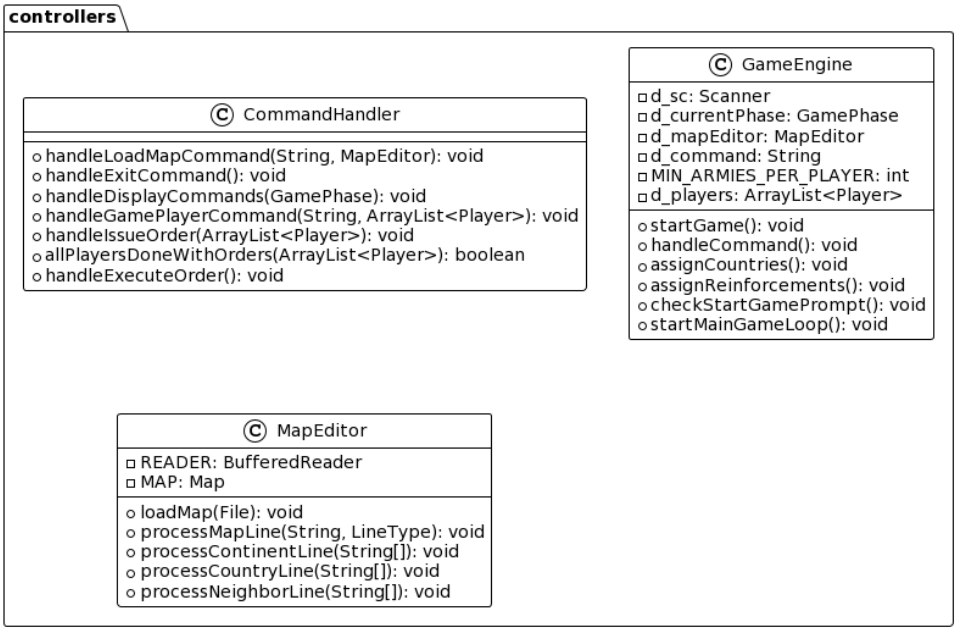


**Fig 7: Controller Package Overview**


**Fig 8: Constants Package Overview**

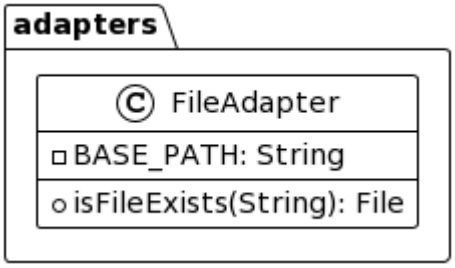


**Fig 9: Adapter Package Overview**

## 5. Testing Strategy

The testing strategy for the Warzone game project encompasses unit testing, integration testing, and system testing to ensure comprehensive coverage of all functionalities. Focus

on individual components within the models, controllers, and views packages. For models, test the integrity of game logic and state changes. For controllers, ensure that commands are processed correctly. For views, verify that information is displayed as expected. Tests are structured parallel to the application's main components, allowing for targeted validation of each module's functionality.

## 6. Conclusion

This document provides a detailed overview of the architectural design for the Warzone Game Project. By adhering to the MVC pattern and incorporating additional support components, the design aims to achieve a flexible, maintainable, and scalable system. The outlined testing strategy ensures that the game's critical functionalities are rigorously validated, contributing to a robust and reliable gaming experience.