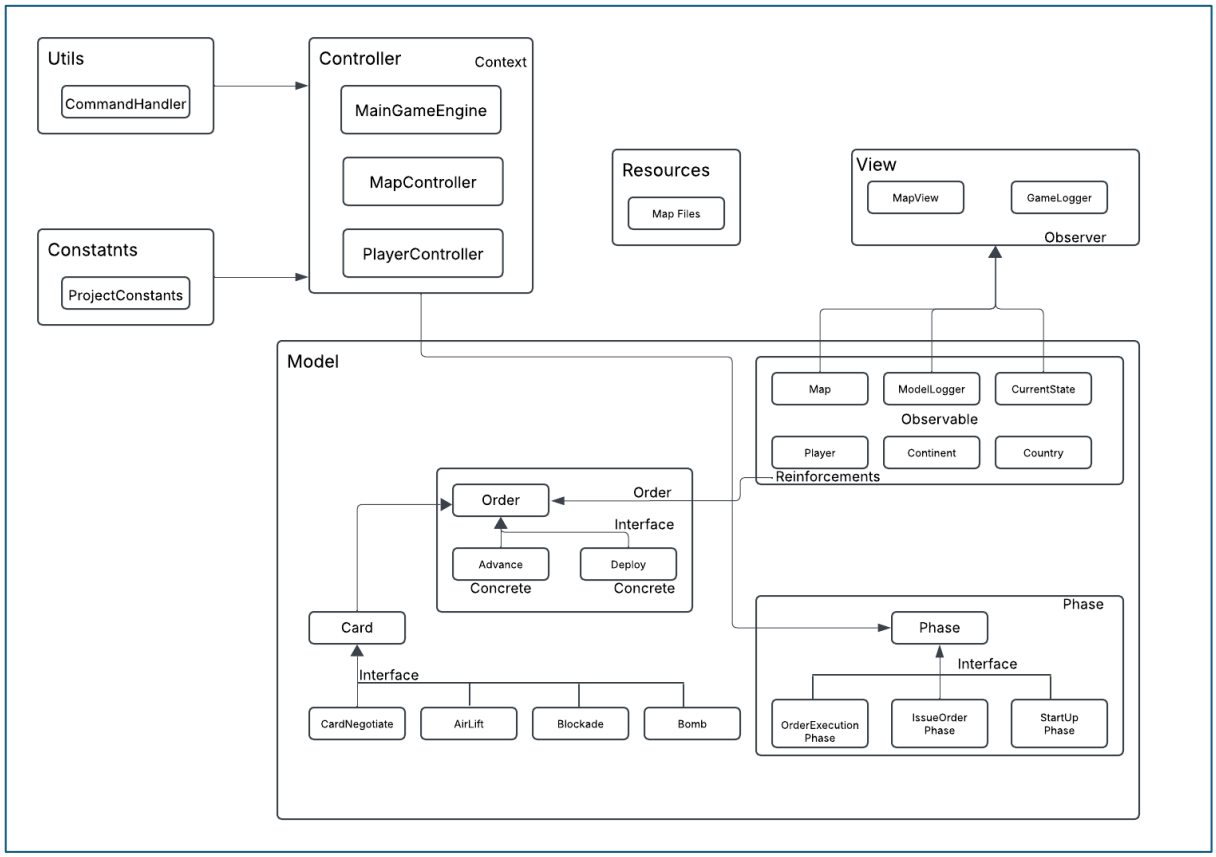
SEON-6441 Advanced Programming Practices

Group ADYST – Project Build 2 Architectural Design Document



Controller

- MainGameEngine: The MainGameEngine class serves as the central controller for managing game phases, states, and transitions. It allows the game to switch between different phases (like "Issue Order Phase" and "Order Execution Phase") and logs these changes. The main method starts the game by initializing the game state and phase, then runs the game's logic by calling the startGame method.
- MapController: MapController class handles loading, editing, and saving maps in a game-like system. It parses map data from files, updates continents and countries, and manages borders between countries. It also provides functionality to edit countries, continents, and their relationships, including adding/removing entries and saving the changes to a file.
- PlayerController: The PlayerController class manages player actions in the game, including assigning countries to players, distributing armies, and handling deployment orders. It ensures proper country ownership, checks for sufficient armies, and tracks unallocated armies and unexecuted orders.

Model

- Map: The Map class that manages a collection of countries and continents, providing functionality to add, remove, and validate countries and continents. It also ensures that countries are connected and that continents form a connected subgraph, along with methods for handling neighboring relationships between countries.
- Continent: The Continent class models a continent, allowing the management of its countries. It provides methods to add or remove countries, set attributes for the continent, and format the continent's details as a string.
- Country: The Country class represents a country, allowing you to manage its neighbours and army count, and access its details like name and continent. It includes methods to add or remove neighbouring countries.
- Player: The Player class in the game is representing a player with attributes like name, unallocated armies, controlled countries, orders, and negotiation history. It includes methods for issuing orders, validating commands, managing player actions, and handling special cards (e.g., bomb, blockade, airlift). The class also tracks the player's log and manages interactions with other players.
- CurrentState: The CurrentState class manages the current state of the game, including the list of players and the game map. It provides methods to add or remove players and retrieve the current list of players and the game map.
- Orders: The Orders interface defines methods for executing, validating, and logging orders in the game. It includes methods to execute the order, validate its validity, and set or get a log of the execution. All order types in the game must implement this interface.
- Advance: The Advance class handles moving armies between countries, executing battles, and conquering territories in a strategy game. It ensures valid execution by checking ownership, army count, and negotiation pacts while updating the game state and logs.
- Card: The Card interface extends Orders and defines a contract for game cards, requiring implementing classes to validate if a card can be legally played based on the game state.
- CardAirLift: The CardAirlift class represents an Airlift card in the game, allowing a player to transfer armies between two owned countries. It implements the Card interface and includes methods for validating the order, executing the airlift, and logging actions. The class ensures the player owns both countries and has enough armies before performing the transfer.
- CardBlockade: The CardBlockade class represents a Blockade card that triples a country's armies and transfers ownership to a Neutral player. It validates ownership, executes the effect, and logs the results.

- CardBomb: The CardBomb class represents a Bomb card that halves the armies in an enemy country. It validates target country existence, checks ownership and adjacency, executes the effect, and logs the results.
- CardNegotiate: The CardNegotiate class represents a negotiation card that allows players to form temporary alliances. It checks if the target player exists, validates the negotiation, and logs the result. When executed, it establishes the alliance by adding both players to each other's negotiation list.
- Deploy: The Deploy class represents a deploy order where a player moves armies to a country they control. It validates if the player owns the target country, and if valid, adds the specified number of armies. If invalid, it logs an error. The class also tracks and logs the execution of the order.
- IssueOrderPhase: The IssueOrderPhase class manages the phase where players issue orders (e.g., deploy, advance, use cards). It processes player commands, validates them, and logs execution results. Invalid commands (like map editing or player management) are restricted. The phase continues until all players have issued orders, then transitions to the next phase.
- ModelLogger: The ModelLogger class is used for logging messages in the game. It extends Observable, allowing observers (like the GameLogger class) to listen for updates. The class sets and formats messages based on their type (command, order, phase, effect, start, or end), and notifies observers whenever a new message is logged.
- OrderExecutionPhase: The OrderExecutionPhase class manages the execution of player orders, checks if the game ends when a player controls all territories, and transitions to the next phase. It rejects invalid commands, prompts players to continue or exit, and adds a neutral player if necessary to handle unclaimed territories.
- Phase: The Phase class is an abstract base for game phases, managing the game engine, state, and controllers for map and player actions. It handles user commands, validates them, and dispatches them to specific phase-related methods like deploy and advance. Invalid commands are logged, and map-related commands are blocked if no map is available. Each phase implements its specific behavior through abstract methods.
- StartUpPhase: The StartupPhase class handles map setup, player management, and command validation during the game’s initial phase. It includes methods for loading, editing, and validating maps, adding players, and displaying game commands, while blocking actions like deploy or advance. Invalid commands are rejected with specific error messages.

Utils

- CommandHandler: The CommandHandler class handles the parsing of commands. It processes a given command string, extracts the main command, and identifies any operations or arguments. It also provides methods to check the validity of operations and their arguments by mapping them into key-value pairs for further processing.

View

- MapView: The MapView class displays the game map, showing continents, countries, their armies, and neighboring countries. It provides an overview of each continent, its countries, and the connections between them.
- GameLogger: The GameLogger class listens for updates from the ModelLogger class (an observable object) and writes any new log messages to a log file (GameLogs.txt). It uses the update method to receive changes and appends the messages to the file using a BufferedWriter.

Constants:

- ProjectConstants: The ProjectConstants class is a utility class that defines a collection of constant strings used throughout the project for error messages, success messages, and command validation. It ensures consistency in messages related to map operations, player management, and game commands.