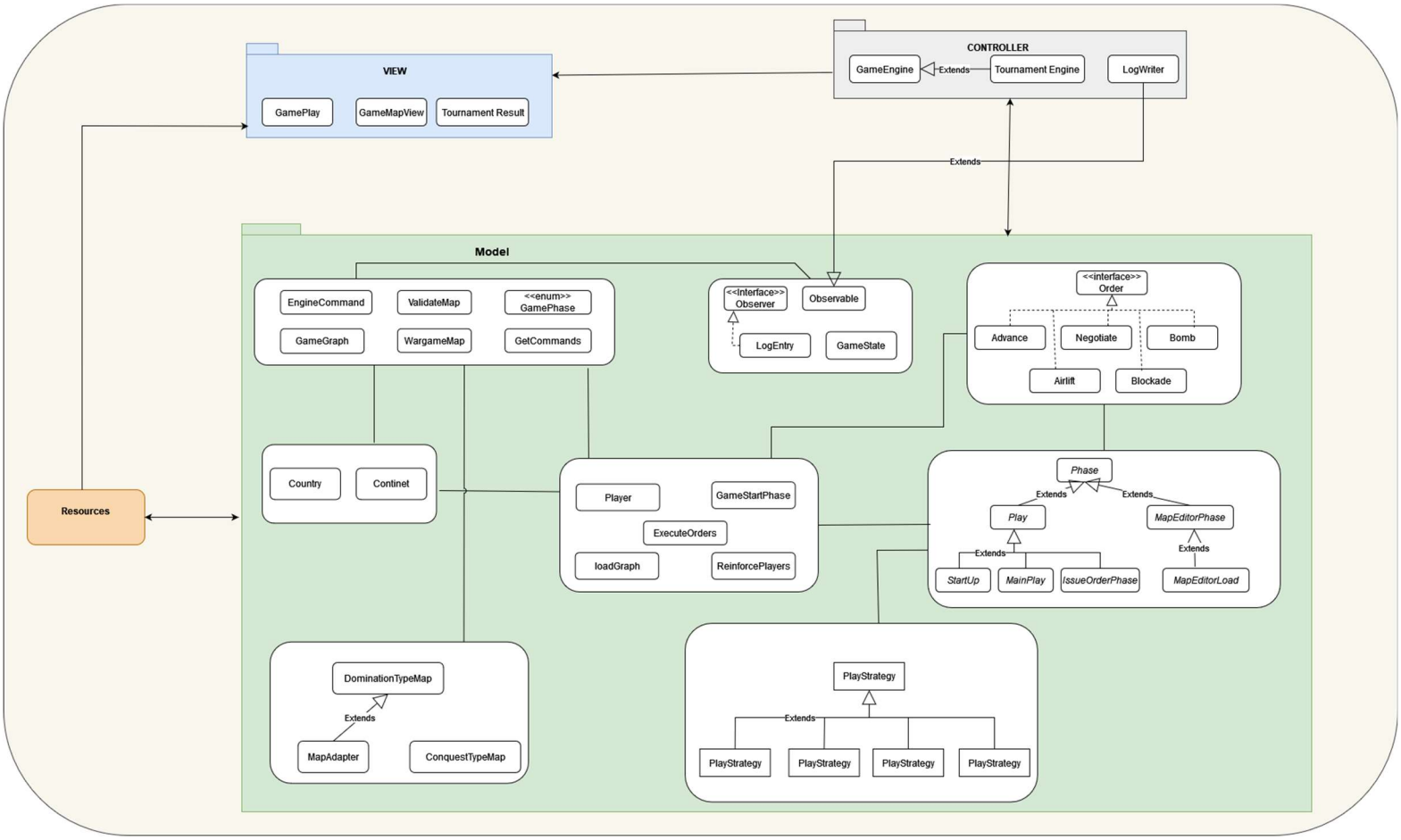
ARCHITECTURE – BUILD 3



The UML package diagram above is the architecture diagram of the wargame command line project created for Build 3. In this build we added additional functionality on top of build1 and build 2 as specified in the build 3 requirements document, we refactored the code to include 2 more design patterns as given below:

- Strategy Pattern
- Adapter Pattern

We also enhanced the game to include 2 game modes:

- Single Game play mode that supports (2-6 players)
- Tournament game mode that is computer based

The game is also enhanced by incorporating Save Game and Load Game features.

Description of the architecture diagram is given below:

BEGINGAME Phase: The `GamePlay.java` file serves as the main entry point for the Risk game, where user can chose to either play in tournament mode or single game play mode. If player choses tournament mode then `TournamentGameEngine.java` class object is initiated and if single game play mode is chosen then it would trigger the `GameEngine.java` class, which parses user commands and calls relevant functions to execute the command.

If Single Game play mode is selected then user is asked to either Load a saved game or create a new game with new or existing map.

The game then follows **State Design Pattern** where depending on users command respective State of the Game is set.

MapEditor State:

1. **editmap:** Opens existing map files or creates a new map. Utilizes `RunCommand.java` to parse and implement commands related to editing continents, countries, and neighbors. Save and load map commands are available to persist changes.
2. **loadmap:** Calls the `loadMap()` function in `RunCommand.java` to load a validated map, transitioning the game to the STARTUP phase.

Play State:

***STARTUP Phase*:** Players are added or removed as per user commands, and countries are allocated. Commands include `showmap`, `gameplayer -add playerName -remove playerName`, and `assigncountries`.

MainPlay State:

***AssignReinforcements*:** After players are added and countries assigned, `AssignReinforcements` calculates armies for each player based on owned countries. The round-robin turn sequence begins for players to deploy armies on their countries.

***TAKETURN* Phase:** Implements round-robin turns for players to issue orders one at a time. After each player's ISSUE_ORDERS phase, `TURNEND` is triggered to move to the next player's turn.

***EXECUTEORDERS* Phase:** Once all orders are collected, the Execution Phase sequentially executes the first order from each player in a round-robin fashion. It checks for a winner if all territories are conquered and removes players if they lose all territories.

ISSUEORDER State:

ISSUE_ORDERS Phase: Players issue orders to the game, with orders collected only if valid. Corresponding army units are reduced upon collection. Players can provide orders and halt to proceed to the Execution phase.

This state also supports **Game Save** command where user just has to enter *savegame <name>* and the game object would get saved in ***Resources/SavedGames*** folder

Adapter Pattern for Map File Formats: Refactor the code to use the Adapter pattern to enable the application to read/write map files in both the "domination" and "conquest" formats. Implemented functionality to decide the file reader to use based on the file type and allow the user to choose the file format for saving. This is implemented by using two classes DominationMap.java and ConquestMap.java class and MapAdapter.java class that extends DominationMap and acts as adapter to convert conquest mpa.

Strategy Pattern

We implemented an abstract class PlayStrategy.java that is extended by following classes:

- An **aggressive** computer player strategy that focuses on centralization of forces and then attack, i.e. it deploys on its strongest country, then always attack with its strongest country, then moves its armies in order to maximize aggregation of forces in one country.
- A **benevolent** computer player strategy that focuses on protecting its weak countries (deploys on its weakest country, never attacks, then moves its armies in order to reinforce its weaker country).
- A **random** computer player strategy that deploys on a random country, attacks random neighboring countries, and moves armies randomly between its countries.
- A **cheater** computer player strategy whose issueOrder() method conquers all the immediate neighboring enemy countries, and then doubles the number of armies on its countries that have enemy neighbors. Note that in order to achieve this, the cheater's strategy implementation will still be called when the issueOrder() method, but will not end up creating