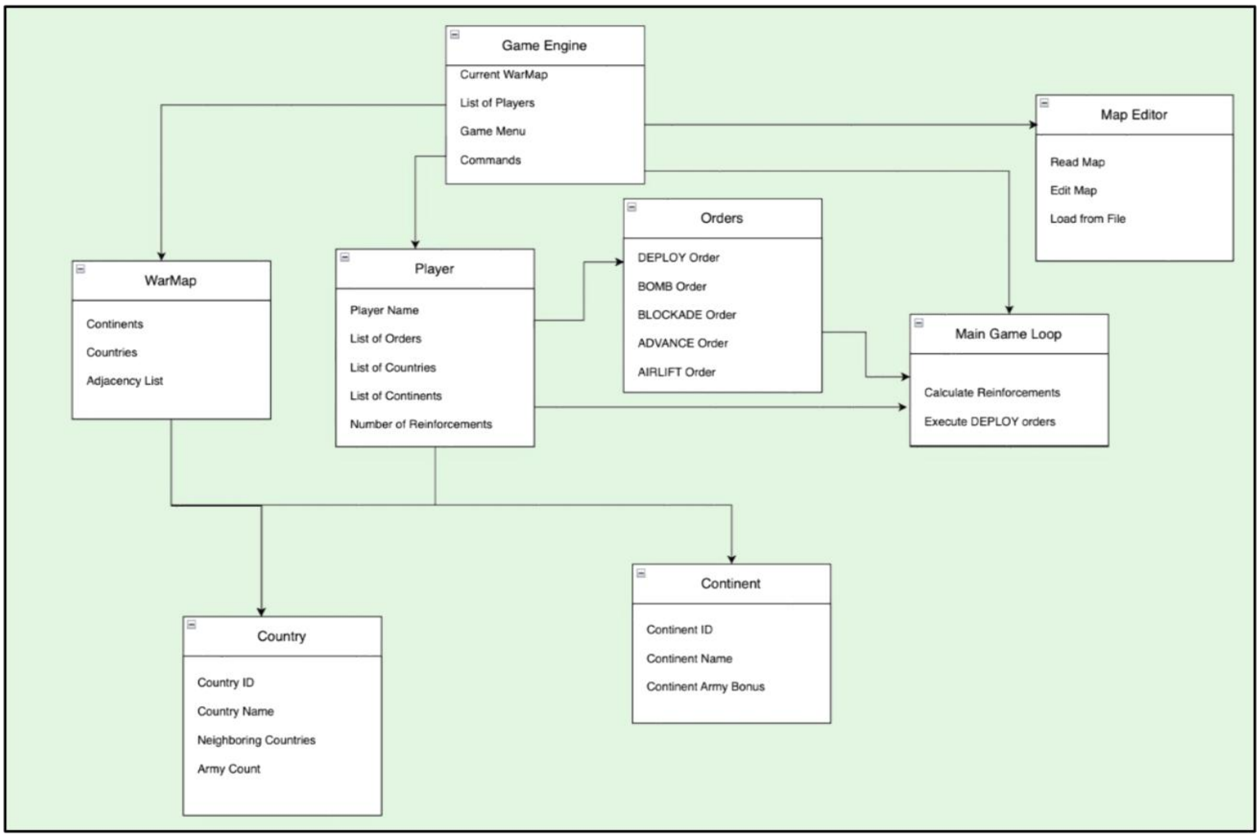
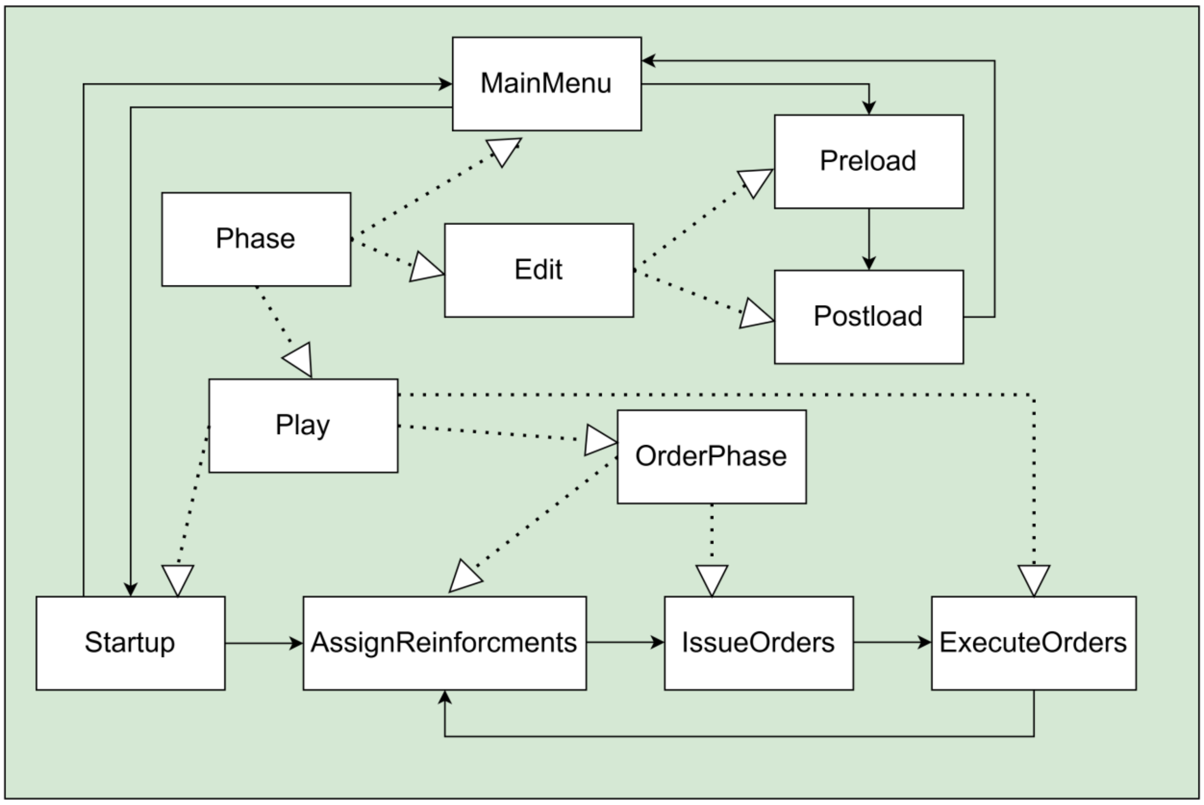
Architectural Design #3

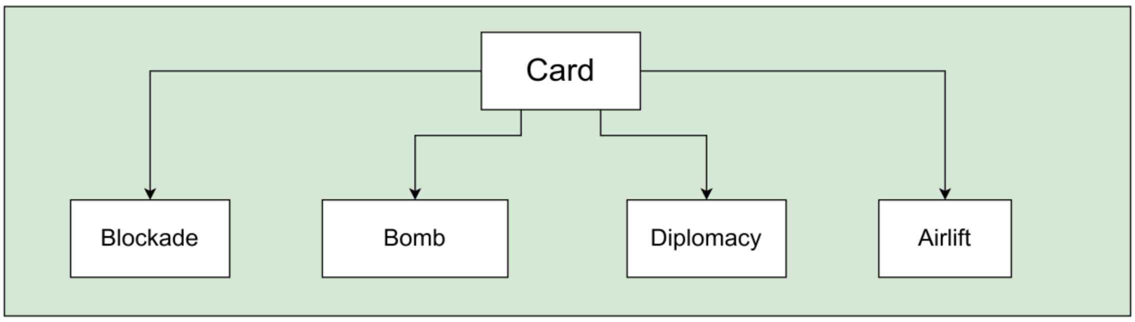
Regarding the main structure of the program given below, GameEngine controls the whole game by holding control of the current WarMap, the list of players, and the current phase of the game. The WarMap has three attributes which are the list of continents, the list of countries and the adjacency List, where the continent is given an ID, name and army bonus and each country is given an ID, name and list of Neighboring countries. The list of players has players attribute which has an ID, name, list of countries, list of orders, and list of cards. This is the high-level structure of our program.

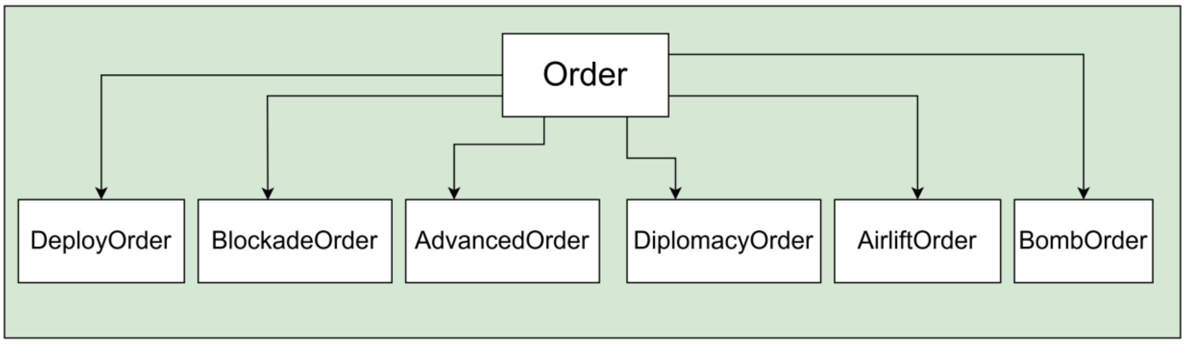


In the GameEngine's phase the state pattern is being used where the MainMenu passes control to either the Edit or Play phases, where the Edit phase is responsible for map editing and the Play phase is responsible for gameplay. In edit phase we separate preload and Postload depending on whether a map has been loaded for editing. The Play phase is divided into Startup, AssignReinforcements, IssueOrders, and ExecuteOrders. In the play phase startup occurs once and the remaining three phases run in a loop, the dotted lines are showing inheritance of phases and solid lines is showing possible flow of phase in the following diagram:

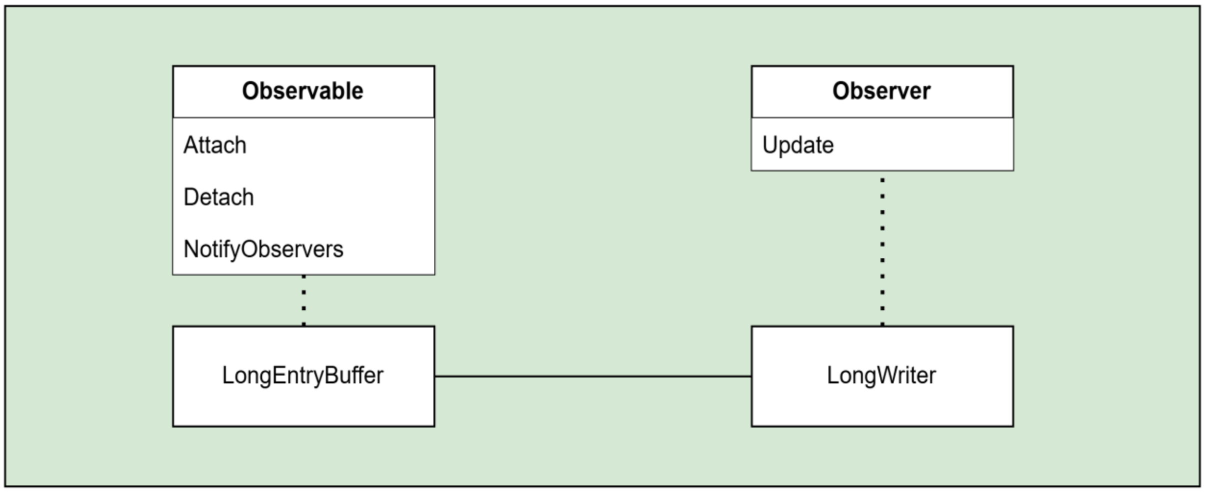


Orders use the command pattern by letting players maintain a list of various order types. This enables a single **issue_order()** function to generate any type of order, determined by user input. The types include deploy, advance, airlift, bomb, barricade, and diplomacy. Similarly, players hold cards in a list, categorized into four types: airlift, bomb, barricade, and diplomacy. The two diagrams illustrate these concepts.

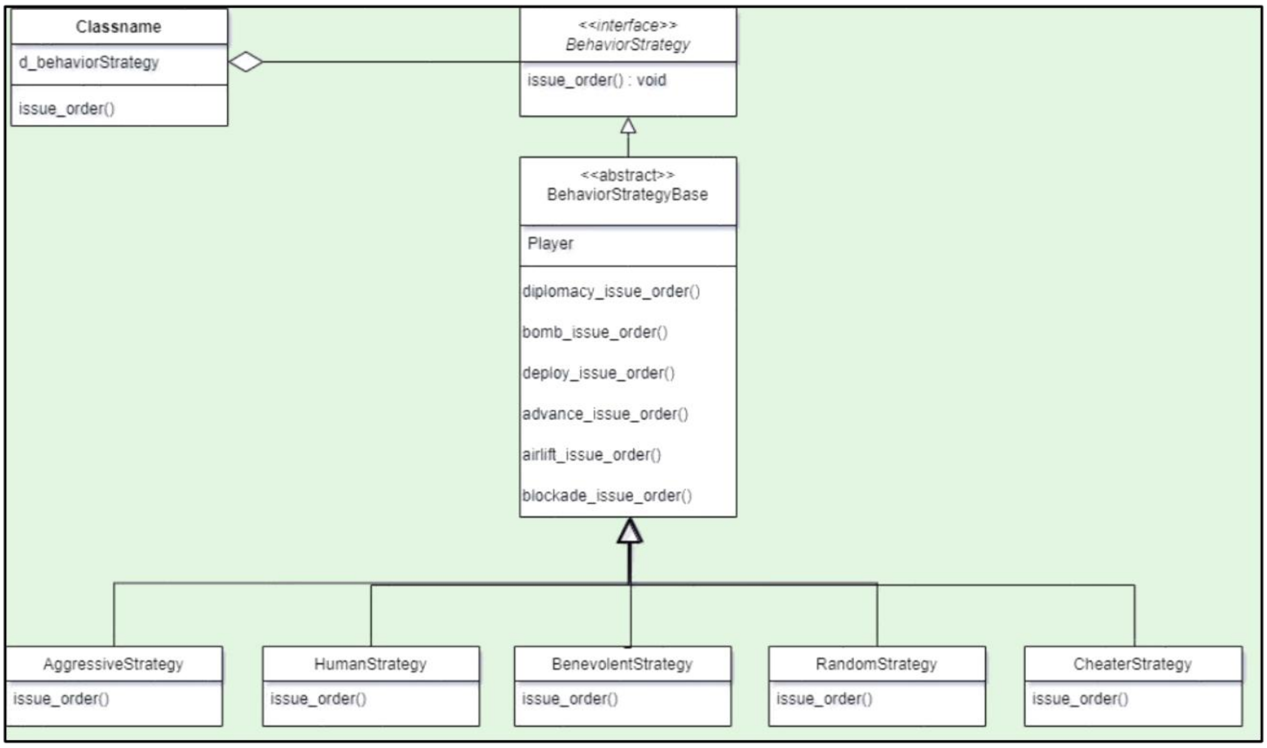




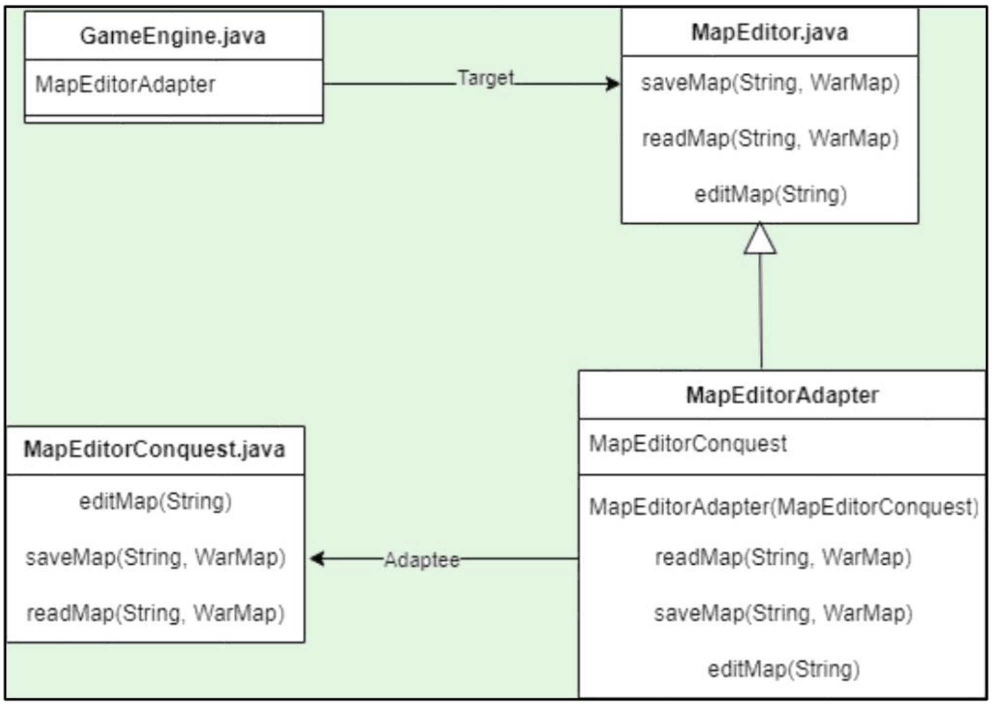
Our program's logging feature uses the observer pattern, with LogEntryBuffer acting as the concrete observable and LogWriter as the concrete observer. Other parts of the program send log messages to the LogEntryBuffer, which then notifies the LogWriter of the new message, prompting it to call the update function and log the message. A diagram illustrating the main relationship of this functionality is shown below.



To use different behavior strategies, the Strategy Pattern is used. The player's issue_order() method calls the issue_order() method from the BehaviourStrategy interface, which is linked to a specific behavior. This behavior is set when the players are created in single-player mode. The control then goes to the specific behavior's issue_order() method, which creates deploy or other orders depending on the current game phase.



To make saving, loading, and editing work with the new conquest map files, we used the Adapter Pattern on the existing MapEditor. This lets us handle conquest files without making the GameEngine worry about the details. We did this by creating a MapEditorAdapter that extends the current MapEditor and passes the method calls to a MapEditorConquest to perform the needed actions.


GameEngine.java
MapEditorAdapter
Target
MapEditor.java
saveMap(String, WarMap)
readMap(String, WarMap)
editMap(String)
MapEditorAdapter
MapEditorConquest
MapEditorAdapter(MapEditorConquest)
readMap(String, WarMap)
saveMap(String, WarMap)
editMap(String)
Adaptee
MapEditorConquest.java
editMap(String)
saveMap(String, WarMap)
readMap(String, WarMap)