Overview of our framework:

**model module**

This module is used for storing data objects like Player, Countries etc. It contains the observable class from observer pattern. It also implements the strategy pattern and command pattern.

**controller module**

This module is responsible for processing commands, game rule etc. It also implements the state pattern for phases and adapter pattern for maps.

**view module**

This module is responsible for interacting with user and executing commands. It also contains the observer class from observer pattern.

Classes in view module:

MainView - This class contains the main function and is used to start the map editing console

ValidateCommandView - This class is used to validate entered command. It will check the syntax and number of parameters passed and then call respective function from phase class.

ExecuteCommandView - This class is responsible for communicating with the controller. It is responsible for calling appropriate functions from controller as per the entered command.

ConsoleView - This view is used to provide console for entire map editing phase

CommandList - This is an enum class which stores all hardcoded commands.

Observer – It is an interface for abstract observer.

FileEntryLogger – It is a subclass of Observer which writes to game log after it update function is called.

Classes in controller module:

GameEngine - This class implements functionality related to the main game engine and also implements console for initial Game play commands like loadmap, gameplayer and assigncountries.

MapController - This class is responsible for maintaining the map. It takes in command and calls appropriate setter/update functions in respective map model class.

MapValidator – This class is responsible for functionality related to checking if a map is valid or not.

ExecuteOrderPhase - This class is just a placeholder.

IssueOrderPhase – This class implements functions related to issue order phase. Each function takes in an order object and adds it to players’ order list.

IntermediateOrderPhase – This class implements all functions that are not valid in issue order and execute order phase.

StartUpPhase – This class implements all functions related to start-up phase.

IntermediateGamePlayPhase – This class implements all functions that are not valid in game play, issue order and execute order phases.

MapEditingPhase – This class implements all functions related to map editing phase.

IntermediateMapEditingPhase – This class implements all functions not valid in map editing phase.

DefaultPhase – This class implements functions related to initializing of phases i.e.: when the game is started this phase allows user to either enter map editing or game play phase.

Phase – The abstract class which defines all functions from all phases.

Classes in model module:

ContinentModel - This class will store information related to specific continent. Its control value and list of country that belong to the continent.

CountryModel - This class will contain information related to specific country. Its current armies count, player who owns it and the continent it belongs to.

CoordinateModel - This model stores coordinates of the country with respect to the entire map.

MapState - This class is used to maintain the list of continents and countries present in the map and their relationship.

Observable – This class is responsible for notifying observer on state change.

LogEntryBuffer – This class inherits Observable and stores the observable data.

Order - This is the abstract class which defines all methods for an Order.

DeployOrder – This class handles validation and execution of deploy order.

AdvanceOrder - This class handles validation and execution of advance order.

BombOrder - This class handles validation and execution of bomb order.

BlockadeOrder - This class handles validation and execution of blockade order.

AirliftOrder - This class handles validation and execution of airlift order.

NegotiateOrder - This class handles validation and execution of negotiate order.

PlayerModel - This class is used to store a players’ information like countries owned and its name. It also stores temporary information like reinforcements and list of orders.

PlayerState - This class is used to maintain the state of the players like add/remove player.

Incorporation of State pattern:

Created Phase abstract class and multiple sub-classes of Phase to indicate different phases. For each command entered by the user/player, we first validate the command and call resp. methods. Then base on the Phase, the method gets executed or it prints “Invalid command for … phase” message.

Incorporation of Command pattern:

Created Order abstract class and multiple sub-classes of Order to indicate different orders. For each command entered by the user/player, we first validate the command and call resp. methods. Then base on the Order, the method creates respective Order object and it gets issued.

Incorporation of Observer pattern:

Created Observable class which stores handle of the Observer class (FileEntryLogger). LogEntryBuffer class inherit Observable and implements functionality related to updating state of the Observable. After every action state in LogEntryBuffer is updated which in-turn notifies the FileEntryLogger, which fetches the current state and adds it to the log.

Incorporation of Adapter pattern:

Created Adapter class which stores handle of the new class (MapLoaderConquest) for reading/writing to map in conquest format. MapLoaderWarzone class inherit Adapter and implements functionality related to reading and saving of maps for domination maps.

Incorporation of Strategy pattern:

Created Strategy class which have a function issueOrder(). Created 5 new classes, all of which inherits Strategy class and have relevant data members. Each player holds an object for Strategy class and based on the behaviour of the player respective class object is created.