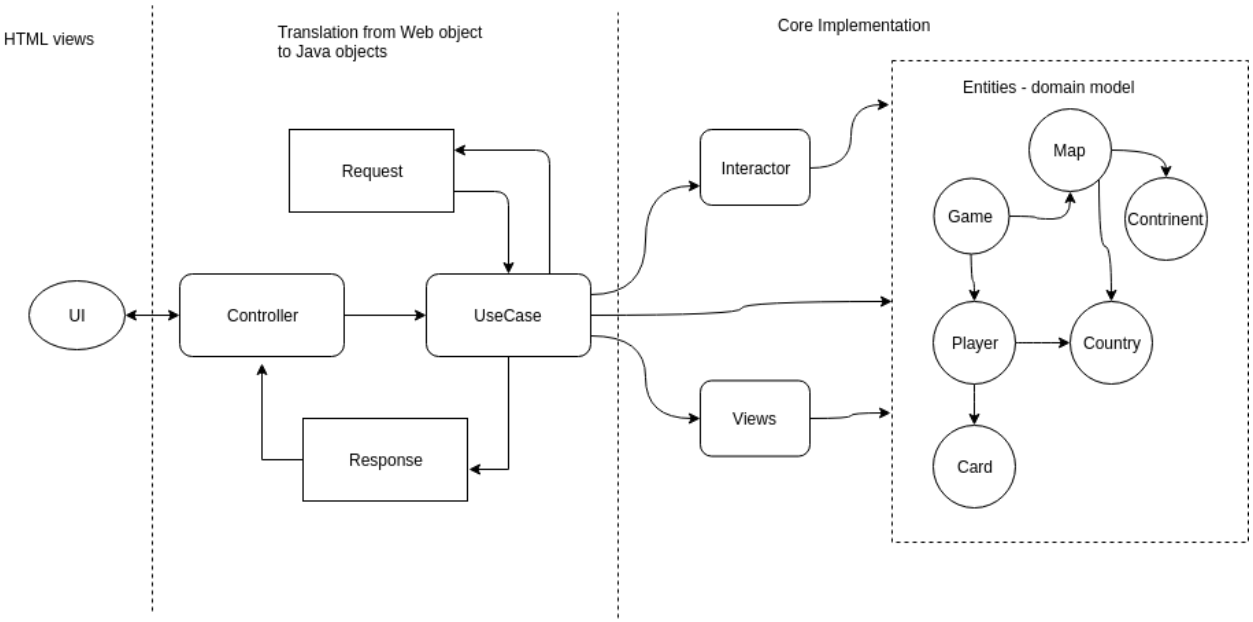
Architecture design diagram



Modules - Controller, Usecase, Request, Response, Interactor, Views, Entity

Architecture style - Object oriented style.
Each module operate on their own data to deliver the data in abstract form to minimize the dependencies between the modules.

All interactions to the system is done through the *ApiController*. ApiController expose the functionalities which are available to the UI to carry out operations and fetch views. APIs are tightly coupled with the views to deliver HTML pages to the client.

Against each API, there are *usecases* which help the user interactions to get converted into useful actions. Each use case can interact with the core layer via *interactor*.

Interactor is responsible to make *entities* interact and get the operation done. So, use cases depends on multiple interactors to achieve a single task.

Entities are the domain model of the application which store data and have business logic to implement the different phases.

Observers are responsible to observe the entities and update the views object whenever there is a change in the entity. The views (PhaseView, CardExchangeView and DominationView) are picked up by the response module to deliver the HTML web pages to the client.

Request layer follow the objective of DataMapper to link the HTML based string input to data structure based java objects to allow the use case to work on them.

Entities

1. Game - is the instance created which loads the map and stores the player along with their current phase and current player to take turn.
2. Map - is responsible to hold all the countries and continents. This class has nothing to do with the involvement of player.
3. Country - is responsible to hold the army count.
4. Continent - since, during the game the geography does not changes, so continent and country objects don't know about the player instances.
5. Player - has the meta-data related to the player info, which is responsible to drive the business logic in the form of its implementation methods.

Interactor

In order to make the game available throughout the session, a singleton pattern has been implemented to store the game state. This singleton is named as GamePlay which is also the front facing class to the Usecases to execute any operation.

GamePlay is responsible to attach the observers as it has to maintain the views in session which will get updated as and when the entity notifies the observers.

Observer

Currently the views are updated by the observer, to remove the dependency of model with the views. Views are responsible to deliver content to the HTML pages, thus, they have all the objects in string and int format.

1. PhaseView - it is responsible to fetch the updates from the game instance, to fetch the current player and current phase.
2. DominationView - since, it is responsible to show both the army counts and share of each player in terms of countries owned, thus, DominationView observes both *player* and *country* objects.
3. CardExchangeView - holds the cards held by each player. This view observes the player class as the cards are held by the player class.