1. abstraction:

Using the virtual function in design pattern, strategy pattern, and decorator pattern in order to implement polymorphism. Subclass override the base class functions and after creation, the child object is assigned to a father object that can call child functions directly.

2. inheritance:

Base class Race is inherited by 14 different race classes;

Base class Power is inherited by 20 different power classes;

Base class gamePieces is inherited by 8 different game piece classes;

Base class PlayerStrategy is inherited by 4 different strategy classes;

Base class ViewDecorator is inherited by 3 different decorator classes;

Base class Observer is inherited by BasicView, ViewDecorator, and Player;

Game inherits Subject class.

In Observer Pattern, we have abstract class Observer and Subject, the Game inherits Subject and Player inherits Observer. The game will send a notification to players if the status of players conquered regions are changed.

3. polymorphism

In Race class, it has getTokens () and getType () member functions. The inheritance classes have its own token number and type, and overriding these two getter methods. In Player class, we create a Race vector to stores inheritance race class and call getTokens / getType to get the specific output. Same as Power and gamePiece classes.

In Strategy Pattern, playerStrategy has five virtual functions to play the game, including firstConquestByStrategy, picks\_raceByStrategy, conquersByStrategy, declineComboByStrategy, redeployTokensByStrategy. Each concrete strategy class overridden these functions. In Player class, after setting different strategy to player, player will call the correct function by polymorphism.

4. high cohesion/low coupling

In Game header, all variable belongs to Game are private and have getter and setter so that other classes can access them. In Player header, it has protected members and getters and setters that can be accessible in other classes. Both classes have high cohension.

In Strategy Pattern, we have abstract class PlayerStrategy, and we create a concrete strategy and implement this abstract class. Every player has a PlayerStrategy pointer as its member. In this case, the player and the strategy are low coupling. If we add a new strategy, it will not affect the player at all. And functions related to specific strategy are in the specific strategy and have no relation to another strategy, which is high cohesion.

Even though the players need to access the Map and MapRegions to get information, but the class Player, Map, and MapRegions are low coupling. We singleton the Map and MapRegion, therefore the player can get the information from them even though the Player has no function related to Map and MapRegions. All function and information related to Map are in class Map, and all functions and information related to MapRegions are in class MapRegions, which represent high cohesion.