The Observer pattern is a way to ensure that several different classes can be notified of an event as it happens. In our project, we have several different views that display various parts of the game. As actions occur and the model and/or state is changed, each view needs to know about the change so that it can updated its view accordingly.

In our project, we will implement the Observer pattern in order to achieve this. Our design is as follows:

1. Create an IListener interface. Every class that needs to be notified of changes made to the game model or class will implement this interface. This basically means that every controller that is linked to a view (to control calculations that need to be done in order to successfully update the view) will implement this interface.

This interface will have several methods that will essentially notify all listeners of some change to the game model. Examples of methods are: roadPlaced(), turnChanged(), playerWon(). If a controller needs update the view as one of these methods is called, then it will implement it accordingly.

2. Create an Observer class. In our case, our ModelFacade class will perform this function. As it controls when and how the game model gets updated (when the Server Poller gets a new model), it will notify all listeners of the updated model.

We accomplish this by creating a list of IListeners, which is contained in the ModelFacade class. All relevant controllers will add themselves to this list right after the contructor is called. Finally, every time the ModelFacade is about to update/change a part of the model, it will call the relevant method in the IListener class. For example, if a player places a road, as the ModelFacade goes to update the new structure, it will call roadPlaced(), and all appropriate controllers will be notified, and change the view accordingly.