DESIGN

1. ARCHITECTURE

This Ivanhoe game is implemented primarily using the Model-View-Controller (MVC) architecture.

2. PATTERNS

Besides the overall MVC pattern, very few patterns were used. There are two reasons for this. First of all, there was no indication that the requirements (such as the networking style or the game rules) would undergo future changes. Making it easy to swap classes only created needless abstraction and confusion. While this is unrealistic for real software, in the case of this particular project adding more patterns would have been adding patterns for the sake of patterns.

Secondly, many of the classes are necessarily tightly coupled. Rules for playing cards are highly dependent on all other aspects of the game – the tournament color, the other players' display, the cards previously played, etc. Almost all the data members access and change each other. In additions, action cards have very individualized behavior, and trying to abstract away their behavior (for example, through a Strategy pattern, or multiple Controllers) would have created a new array of problems.

--- FACTORY PATTERN

The AppClient uses a Factory pattern to create its View object. The ViewFactory is an interface implemented by its subclasses – MockViewFactory, GUIViewFactory, and TextViewFactory. This makes it possible to switch between a GUI, a TextView, and a MockView (for testing) depending on the requirements, and would make it easy to add more views for different platforms (such as mobile).

--- FAÇADE PATTERN

You could argue that the AppServer is a façade class to access the Controller, since all client interactions must pass through here. Additionally, the AppServer hides the complex, tightly coupled interactions of the Controller class.

3. CHANGES FROM ITERATION 1

There were minimal changes from Iteration 1. The controller was separated into smaller methods, each one with a more precise responsibility.

Where possible, code duplication was removed by adding additional methods (instead of through patterns). This can be seen in the Ivanhoe Controller, and in the addition of the ITM (IntegrationTestMethods) which contains generalized methods for writing integration tests.

4. PROS & CONS

--- PROS

- The Controller manages all of the game information – player turns, tournament, connected players, etc. As such, there are no synchronization issues.

- The Controller is the only object that changes model values, and the Controller can access all of its data members directly. This makes for good performance.

--- CONS

- At the moment, all classes are highly coupled with each other.

- The Controller class (Ivanhoe controller) is somewhat convoluted as it involves multiple states and handlers. However, since it contains all of the model information, breaking the Controller into multiple classes would have led to other problems.