Design Patterns Used for Focus

Most of the patterns used throughout our project are sort of hybrids of the nine primary GRASP patterns. As some classes may seem to take the form of a design pattern, it is not pure in its implementation. We will discuss some below.

Information Expert: This is probably our most commonly used pattern. Certain classes such as MoveController, AudioController, and LegalController all take care of tasks related to the object's own knowledge. For example, the MoveController takes care of all the move functionality. The AudioController only takes care of tasks related to the background music and sound effects. The LegalController control only deals with generating legal moves.

High Cohesion: Leading in from the Information Expert pattern is high cohesion. The LegalController is a great example of a smaller class with a very specific set of functionality. However, the GUI classes are low in cohesion since they deal with edge cases involving invalid moves. If we had more time, we would try to make use of design patterns which help handle these edge cases such as Pure Fabrication.

Low Coupling: We tried our best to use low coupling. We probably would score somewhere in the middle since some of our classes are low coupled such as GameSession, AIController, and FileManager. Others are highly coupled such as the GameController, the GUI classes (mainly GameGUI) and Board. Given more time, we would make intermediate classes to help reduce coupling between these highly coupled classes.

Creator: All of our GUI classes can be generally classified as Creators. They all create the Swing elements of the game. The Board class, for example, will create the entire board display plus the Stacks (game tokens / pieces) for the board itself. Outside of it being a creator class, it does have some methods which handle the action events of the game. These extra responsibilities take away from it being purely a Creator class.

Controller: we use many session controllers in the design of GameController. Though all actions are done via the GUI classes, each of the controllers (AIController, MoveController, GameController, and LegalController) all each take care of the responses to the action events in the GUI classes. This pattern was the most useful to us in helping manage the input from the user and returning the proper response back to the user.

Pure Fabrication: we have created the FileManager class which is responsible for handling the save / load game functionality. This takes the responsibilities away from the GameGUI, MoveController, AudioController, and the LegalController of saving / loading the game and lets FileManager take care of it.