Use of Design Patterns in the Game of Blokus

Design patterns are an integral part of the modern software development process. They can be incredibly useful when used properly and intently. In this course, we've formally learned about General Responsibility Assignment Software Patterns (GRASP) and have tried our best to implement them in the design of our Blokus application. Those patterns we used made our code notably more efficient to update between iterations.

The first design pattern that we implemented, without having properly learned of it yet, was that of the Information Expert. We created the class diagram and divided the responsibilities to each class, reflecting on which responsibilities would best fit in each class given the information available to those classes and the concepts they represent. Most classes can fulfill their tasks using their own attributes and methods. For example, the responsibility of the SettingsGUI class is to create a Settings interface for the Blokus game and to open a game board as per the settings chosen by the player.

Two other patterns that we focused on are Low Coupling and High Cohesion. As mentioned above, our game is implemented many classes designed to be reasonably independent. We tried to avoid class dependency as much as possible so that alterations to one class would not widely upset the greater design. This allowed us to manage the code base easily and made the code re-usable. With regards to High Cohesion, we tried our level best to keep the Cohesion as high as possible, however, we could still work on it as there're still rooms to improve at this point.

Another pattern that we used is Controller by designing an interface called Player to implements two classes, namely HumanPlayer and Computer Player. By doing this we are not only keeping the player classes logic separate from the interface, it also makes the maintainability of the code easier. We have also used Singleton pattern by making a class named Main, the task of which is to instantiate a class named Blokus.

We believe we could have applied more design patterns, both if given more time and with better use of the time we had. We could've used Pure Fabrication Pattern to save and load the game. We could've also the Polymorphism pattern better. One area in which our code could be noticeably improved is through the use of Protected Variations. Especially noticeable in the Strategey class' easyStrategey() method, some methods in the code feature high object coupling, which leads to fragile design where the alteration of different classes could potentially effect these methods without alteration. Given more time, fixing this would be a priority.