Requirements: Explain in one page or less which patterns are used at what points in the code for our project, or where we would put patterns with more time.

The Options class is an example of an information expert we've used. The class is entirely comprised of getters and setters, related to the settings chosen by the user, and it allows other classes to interact with it instead of directly with the UI.

The Piece class is an example of a creator. Unfortunately, most of this class has been hard-coded due to the variety of pieces, but allows those created pieces to be more easily used in other classes.

We've tried to practice low coupling and high cohesion throughout this project. We've also kept each UI screen separate so that most of our UI interactions call on other classes. This way, any change to functionality can be changed in the function call rather than having to rewrite any other code within the UI class.

The classes SaveGame and LoadGame can both be considered session controllers since they work with UI requests and make that information accessible to the main logic classes. These classes also are based on use cases practiced in previous iterations.

Polymorphism was used in that there are 21 distinct piece shapes in Blokus, and we've used the same basis on those for each player, just assigning them to certain players.

Some of the patterns discussed in class would not have applied to this project. A factory would not have been as feasible as Creator classes, provided the specifics of each piece. Adapter wouldn't apply due to the project being created from scratch and not specific to any hardware. Pure fabrication hasn't been needed due to the close relation between domain classes and the necessary implementation classes.

In the creation of this project, more attention could have been paid to using proper software patterns. Had there been knowledge of GRASP done before our implementation, or if more time were provided in the iterations to practice its application, we could have more effectively used these strategies, particularly in the creation of the many pieces and in dividing responsibilities in the GameEngine class.