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 and pure fabrication. The class doesn't play any role in the real world domain of the program however it works as an information expert and handles all the options within the game. 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. This class may also represent protected variations as it removes the responsibility from other classes of knowing how or when the options are selected. We can use this same class in the future to support other systems.

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. Example of this would include the MenuCreator, PlayerGrid and Player, SaveGame and SavedState, LoadGame and LoadScreen classes.

The classes SaveGame and LoadGame can both also 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 practised in previous iterations.

The CreateGame class is an example of a creator. When starting a game it initialises all the other objects or assigns responsibility to others.

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. The adapter pattern wouldn't apply due to the project being created from scratch and not specific to any hardware.

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.