Explanation of Design Patterns

The design patterns we used:

Information Expert: We assigned the GameGUI class as an information expert. The GameGUI contains both the SettingsController and BidController. It also updates the players' scores and announces who is the winner at the end of the game.

Creator: The creator pattern has been used for several classes. For instance; the Robot objects, Token objects, and Barrier objects are created & contained in the GameBoard class. And Bid objects are created & contained in the BidController. These are some examples of the classes in our program that follow the creator design pattern.

Low Coupling: By using classes such as BidController and ColorPalette, we are lowering the responsibilities of the GameGUI. This reassignment of responsibilities allows the GameGUI to be changed without affecting related classes. This is a clear example where the low coupling design pattern was implemented in our design.

High Cohesion: We created several classes for game pieces. Also, assigned each class of the model with specific responsibilities in order to keep objects strongly related and focused to a particular function. Hence, high cohesion is maintained.

Controller: We have used a controller, which controls all the settings input and bid inputs from the players and responses accordingly.

The design patterns we could use:

Indirection: We could use indirection which would allow us to create an intermediate object to assign specific responsibility. This would help us in the long run during maintenance and re-use without losing low coupling and high cohesion.

Command: We could use the command pattern. This would allow us to assign responsibility for responses to small distinct user commands such as selecting object, starting a new game, and loading a saved game etc. This would help us to maintain high cohesion and low coupling during maintenance afterwards.