To support game selection and player count selection, we created a new class called VariableButton. This class extends the Button class and primarily acts the same as a Button, but the most substantial difference is that, in the ViewFacade class, the register() method is overloaded to set the remote ID of said item to its label. This allows for a more predictable way of getting the information needed for which button was pressed instead of having a different event for each player count or game identifier. In turn, when we receive a SocketMessage, this is passed to the correct event class where it creates an event with the information specified in its remoteId acquired from the SocketMessage. On top of this, we created the PlayerCountButton and SelectGameButton classes that are used to trigger a SetQuorumEvent and a SelectGameEvent respectively. These buttons also contain a kSelectorBase static variable that acts as a prefix to the selector passed to the superclass and is specific to the button class. The full selector will be kSelectorBase appended with the label of the button.

The events triggered by both of these Button classes now implement the EventFactory interface, which means they must have a createEvent() method. This means they depend on SocketMessage as they receive a SocketMessage to return their respective event. Also, when a SetQuorumMove is triggered, and the ViewFacade is passed in, it will look at the ScreenState class we created. ViewFacade has a ScreenState instance variable, and this acts as an iterator that shows the currently displayed objects as well as the objects displayed on the next screen. Upon apply(), SetQuorumMove will get the next screen. On the other hand, SelectGameEvent will handle this in a different way. The next screen for this is dependent on whether the game being played is single-player or multiplayer, so it will instead determine this based on whether a quorum is already set. When a quorum has been set that will indicate that this is a single-player game and go straight for the deal screen, but when it has not, it will instead go to the next screen. All of the needed screens will be created in the GameController's apply() method for SelectGameEvent.

We will also need to modify the way several currently existing methods operate. Currently, GameController's apply method for SelectGameEvent creates a Quorum, but we don't want this done yet outside of single-player games, so we will remove it from this position. Multiplayer games will utilize PlayerCountButtons, but single-player games will get a default quorum added in their SelectGameEvent with a minimum and maximum player count of 1 so will still have it added here. Similarly, in the ConnectEvent, it will no longer create a new SelectGameEvent as the game is not until a button is pressed.