Design Patterns in Can't Stop

Singleton - we used a singleton design pattern, and created a singleton class called Player which provides global access and single access to this class. This class records all the settings selected by the players such as number of players, difficulty and therefore other classes such as WindowDemo can access this single instance and acquire the settings to apply them.

Command – we didn't use command interface and commandHolder interface in our design, but we could have used it for our buttons for Stop, Bust, New Game, Save Game, and so on to fully separate application logic from the UI which would increase cohesion and descrease coupling. Each important button would have its own classes e.g StopButton that would implement the CommandHolder and StopCommand class which would implement command interface.

Controller – we didn't use a controller class, but we could have implemented a facade controller since our project is small and the number of input system events are small as well. Using a controller class would help separate the application logic from the UI, leading to high cohesion and low coupling, increasing modularity. There would be a main controller called gameController and it would handle input system events and delegate actions to others classes such as command classes.

Information Expert – we used information experts to assign responsibilities. For example, the Roll class is the information expert, and has responsibility of rolling. Which allows for low coupling and high cohesion. And other information experts include summary, GridSquare, player_settings which have responsibilities fulfilled with the information necessary in those classes.

Creator – we used creator design patterns to allow classes that aggregate other classes to create instance of them. WindowDemo creates instance of Roll class since it closely uses Roll for making a roll, Summary creates WindowDemo instance since it has initializing data that will be passed to WindowDemo when it is created.