Our deadwood board game follows the Model-View-Controller (MVC) design pattern, which offers numerous benefits such as modularity, reusability, maintainability, flexibility, and collaboration. In our implementation of the game, we have three major models: Player, Board, and GameState.

The Board model is responsible for storing data extracted from the parsers for cards.xml and board.xml. It holds information about the game sets, and to achieve a well-structured design, we have implemented an abstract class called Location, with three children classes: Trailer, CastingOffice, and SetLocation. Additionally, the SetLocation class contains a Roles class to manage role information. This approach ensures high cohesion and low coupling within the class hierarchy.

Similarly, the Player class stores information about the players, while the GameState class manages the overall progress of the game. These classes provide methods to determine and update player and game states. By focusing on specific purposes and minimizing dependencies, these classes exhibit high cohesion and low coupling.

Moving on to the controllers, we have implemented several components. The ActionProvider controller primarily focuses on the player's actions and validates their validity. The DisplayController handles player inputs other than actions, while the GameInitializer is responsible for creating player objects and ensuring unique names. By assigning each controller a single responsibility, we maintain high cohesion, and as they are independent in terms of changes and modifications, we achieve low coupling.

All controllers can communicate with the Display class to output information about the current player and game status. Overall, by adhering to the MVC design pattern, our implementation demonstrates high cohesion and low coupling, resulting in a well-structured and flexible architecture.