Design Patterns

Kaitlyn Rice & Mikayla Porter

Introduction

Design patterns play a crucial role in software development, providing proven solutions to common design problems. In our Deadwood game implementation, several design patterns were used to ensure a structured and more manageable codebase. In this paper, we will discuss the design patterns used in our Deadwood implementation and their significance in enhancing the overall architecture and functionality of the game.

Singleton

In our Deadwood implementation, the Singleton pattern ensures that designated classes only have one instance throughout gameplay. This ensures consistency and global access to critical game elements. The Gameboard class serves as an eager Singleton, allowing global access to the game board from any part of the code. This allows each player and the game elements to interact with the same instance of the board, preventing inconstancies or conflicts in the game state. The Banker class is another eager Singleton since only one banker is needed for the duration of the game. The banker class interacts with players and sets to determine payments to or from players.

Observer

The Observer pattern is used in our Deadwood implementation to manage game events and update player data establishing a relationship between the subjects and observers. The DeadwoodUI class serves as the observable subject in our implementation. It notifies observers of state changes and events related to the game's user interface. Observers, such as the Board and Set classes, register themselves with the DeadwoodUI class and receive updates whenever the UI state changes. The Player class acts as an observable subject, while Board and Set classes act as observers. The

Player class notifies observers of state changes, such as a change in player location or ranking.

Model-View-Controller (MVC)

The Model-View-Controller (MVC) pattern is used in our Deadwood game by dividing the work into three interconnected components: the model, the view, and the controller. In our implementation, the DeadwoodUI class acts as the view component in the MVC pattern. It is responsible for rendering the game interface, displaying relevant information to the players, and receiving user input. The DeadwoodUI class communicates with the model components, such as the Gameboard, Player, Role, and Scene classes, to retrieve data and update the UI accordingly. The model components control the game's data and logic, providing game states. The model components communicate with the view (DeadwoodUI) through observer notifications, ensuring that any changes in the game state are reflected in the UI. The Action class acts as the controller component in our implementation. It handles user interactions and orchestrates the flow of the game. The controller interacts with the view (DeadwoodUI) to receive user input and updates the model components based on the actions performed by the players.

Conclusion

In conclusion, using design patterns in our Deadwood game implementation has contributed significantly to the game's functionality. These patterns solve common design challenges and serve as valuable tools in creating robust and maintainable software. The Singleton pattern ensures that designated classes have a single instance throughout the game, promoting consistency and global access. The Observer pattern allows for loose coupling and communication between the observable and observers. The Model-View-Controller (MVC) pattern enhances code organization, separating concerns and facilitating maintainability and extensibility. By employing these design patterns, we have created a well-structured and modular codebase for the Deadwood game, enhancing the development process and improving the overall user experience.