CSCI 345: Design Patterns Write-up

Author: Daniel Wertz

June 03, 2023

In our Deadwood project, we have leveraged the power of design patterns to create a well-structured and maintainable program. By implementing the Model-View-Controller (MVC) pattern, we ensure a clear separation of concerns and promote extensibility. The Singleton design pattern guarantees the creation of a single instance of two essential classes. Additionally, we have incorporated elements of the Observer pattern to capture user actions and update the program accordingly. Together, these design patterns enhance the overall architecture and functionality of our Deadwood project.

Our program follows the MVC design pattern, ensuring a clear separation of responsibilities and promoting maintainability and extensibility. The ViewHandler class serves as the view component, responsible for presenting the graphical user interface and interacting with the user. The view communicates exclusively with the Controller class, which acts as the central controller component. The Controller class serves as the controller component and orchestrates the flow of data between the view and various model classes. The remaining classes make up the model and a few of them interact with the controller. By keeping the view separate from the model, the MVC pattern promotes a clean separation of concerns.

The incorporation of the MVC pattern offers several benefits to our program. Firstly, it enhances maintainability by providing a clear structure and separation of responsibilities. The different components can be developed, tested, and modified independently without impacting the others. This modular approach allows for the introduction of new features or enhancements without major modifications to the existing codebase. Overall, the implementation of MVC in our program enhances code organization, promotes reusability, and simplifies maintenance and future development.

We also made use of the Singleton design pattern to ensure the creation of only one instance of the Board and Controller classes. This Singleton approach allows for multiple classes throughout the application to access and utilize the same shared instance of the Board and Controller, promoting consistency, coordination, and efficient resource management. Incorporating the Singleton pattern gives us a global point of access to these classes, simplifying their management and control, and eliminating the need for multiple instances, thus reducing complexity and potential synchronization issues.

Our ViewHandler class also uses design principles from the Observer design pattern. The JButton's are dynamically created in the ViewHandler based on function calls made by the Controller. They use an ActionListener to capture user actions and notify the Controller class. Controller then determines what to do with this information and updates the model accordingly. Similarly, the paintComponent method in ViewHandler acts like an observer, automatically triggered by the framework to repaint the GUI when needed.