Write-Up

Singleton Pattern

The ***Board*** class implements the Singleton pattern.

- Consistency: This design ensures that there is only one instance of the game board throughout the application, preventing inconsistencies in the game state across different parts of the program. This is crucial because the game board is a central component that must remain synchronized across various operations like moving pieces and building towers.
- Memory Efficiency: It also prevents unnecessary allocation of memory and processing power that would come with maintaining multiple instances of a complex object like the game board.

Observer Pattern

The ***Game*** utilizes the Observer pattern via the ***Observer*** class to handle updates and notifications throughout the game flow. For instance, when a player makes a move or when the game status changes (e.g., a player wins), the Observer is notified and updates all listeners (in this case, outputs to the console).

- Decoupling: The pattern decouples the game state management from the user interface or any other system that needs to respond to game state changes. This separation enhances the system's maintainability.
- Flexibility: New observers can be easily added without modifying the existing subjects or other observers, making the system more adaptable to future enhancements such as new ways of displaying the game state or integrating additional logging.

Memento Pattern

To facilitate undo and redo functionality, the ***History*** class employs the Memento pattern. This class stores state snapshots at each turn, allowing players to revert to a previous state or reapply a state if needed. This feature enhances user experience by allowing exploration of different strategies without permanent consequences on the game's progress.

Template Pattern

The ***Player*** class serves as a template for different types of players (human, random computer, heuristic computer). Each subclass implements the player_turn method, which defines the logic specific to each type of player while adhering to the template set in the base class

- Code Reusability: By defining a skeleton of operations in the base class, the Template pattern allows subclasses to implement these operations without altering their structure. This pattern promotes code reuse and polymorphism, making it easier to introduce new player types without altering existing code significantly. This leads to a reduction in code redundancy and increases code reusability.
- Standardization: It sets a standard procedure (template) that all player types must follow, ensuring consistency across different implementations, which simplifies understanding and maintaining the code.

Command Pattern

The ***Piece*** class demonstrates the Command pattern, encapsulating all commands a game piece can perform, such as moving and building. Each command modifies the game state through methods like move and build, which can be executed, undone, and redone.

- Command Encapsulation: Commands related to game moves and builds are encapsulated in the Piece class, making it easy to extend or change the command without affecting the caller. This encapsulation also aids in adding new commands easily.
- Undo/Redo Integration: Integrates seamlessly with the Memento pattern to provide undo/redo functionality by keeping the command execution history. This is crucial for allowing players to revert and redo their moves.

Iterator Pattern (Enumerate Moves/Builds)

The methods ***enumerate_all_moves*** and ***enumerate_all_builds*** in the Piece class can be seen as custom iterators that provide a specific iteration over possible moves or builds for a piece.

- Simplification of Complex Iterations: Custom iterators simplify the logic required to navigate complex structures (like a game board). This makes the code more readable and maintainable.
- Encapsulation of Navigation Logic: By encapsulating the navigation logic within the iterator, the rest of the application is shielded from the complexity of the underlying data structure (the grid layout of the board in this case).