Design Patterns in Kivi

Adapter Pattern

The game does not currently use the Adapter pattern, as there are no incompatible interfaces that need bridging. However, if we want to integrate online multiplayer into the game while maintaining the existing KiviBoard structure, we could use the Adapter Pattern to bridge local game logic with network communication.

Factory Method

The Factory Method pattern is partially used in our code. Human and Computer are instantiated via constructors. While not a formal Factory Method implementation, it follows the same concept of creating objects without specifying exact classes. For a better design we could implement a proper PlayerFactory with methods like createHumanPlayer() and createComputerPlayer(). This would centralize player creation logic and make it easier to extend with new player types. Additionally, the Computer class already uses a difficulty enum (EASY, HARD), which could be paired with a Factory Method to create the appropriate turn-handling strategy like EasyTurnStrategy, HardTurnStrategy.

Observer pattern

The Observer Pattern is used in KiviBoard to ensure the GUI components (such as JPanel and JLabel) are updated whenever the game state changes. The board acts as an observer, listening for updates from subjects like the dice roll or turn changes. For example, when a player rolls the dice, the DicePanel notifies KiviBoard, which updates the rolls left label and relevant UI elements. Similarly, when a player's turn ends, the board listens for this event and updates the leaderboard, turn indicators, and score labels. This pattern promotes decoupling, ensuring that game logic and UI updates remain synchronized without tightly coupling them together.

Command Pattern

The Command pattern is not currently used but would be valuable for enhancing functionality. Currently, actions like rolling dice or placing stones are handled procedurally within methods. Encapsulating these as command objects (RollDiceCommand, PlaceStoneCommand) would allow the functionality of reverting a mistaken move (Undo/redo). We could use this for storing a sequence of moves in ComputerTurnController which is useful for networked multiplayer or replay systems.

Controller

The game currently uses a controller layer (ComputerTurnController manages AI logic, while DicePanel and KiviBoard handle input/updates). However, a proper controller pattern would put all game flow like turn transitions, dice rolls, and stone placement into a dedicated GameController class.