

Design Patterns

Overview

<i>Singleton pattern</i>	Used in:- GameController logic(inside GameBoard), ScoreManager("Scoring"), GameState Manger Why:- To maintain a consistent global state, avoid conflicts and manage a centralized state
<i>Strategy pattern</i>	Used in:-ComputerPlayer (AI move strategies) Why:- It allowed ComputerPlayer use a strategy to determine moves. The strategy interface allows different AI b
<i>Observer pattern</i>	Used in:- GameBoard to notify PlayerInfoPanel/UI updates or turn label updates. Why:- The current system manually updates, but using the Observer pattern would better decouple UI from logic. It also provides loose coupling between backend game logic and frontend UI panels.
<i>Factory pattern</i>	Used in:- CellManager(Initializes board with combinations and points) Why:- CellManager could implement a factory method for creating different cell types. Player objects could also be instantiated via a factory for future extensions.
<i>Command pattern</i>	Used in:- MainMenu actions, UI buttons (RollDice, PlaceStone) Why:- While buttons like "Start Game", and "Roll Dice" call methods directly, they could encapsulate logic in command objects for cleaner UI logic separation. This would help enable undo/redo and clean command encapsulation in the future.
<i>Memento pattern</i>	Used in:- GameState (for Save/Load) Why:- GameState is used to save and load game state and is serializable. could formalize the Memento structure to increase its encapsulation and extensibility.