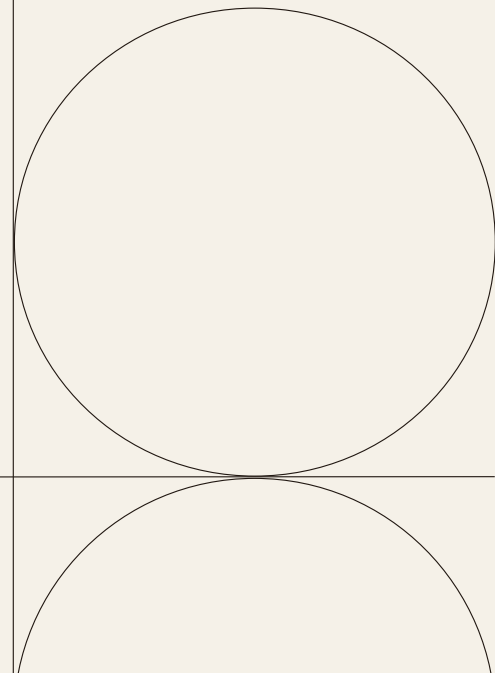


# Design Patterns



## Overview

<i>Singleton pattern</i>	<p>Used in:- GameController logic(inside GameBoard), ScoreManager(“Scoring”), GameState Manger</p> <p>Why:- To maintain a consistent global state, avoid conflicts and manage a centralized state</p>
<i>Strategy pattern</i>	<p>Used in:-ComputerPlayer (AI move strategies)</p> <p>Why:- It allowed ComputerPlayer use a strategy to determine moves. The strategy interface allows different AI b</p>
<i>Observer pattern</i>	<p>Used in:- GameBoard to notify PlayerInfoPanel/UI updates or turn label updates.</p> <p>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.</p>
<i>Factory pattern</i>	<p>Used in:- CellManager(Initializes board with combinations and points)</p> <p>Why:- CellManager could implement a factory method for creating different cell types. Player objects could also be instantiated via a factory for future extensions.</p>
<i>Command pattern</i>	<p>Used in:- MainMenu actions, UI buttons (RollDice, PlaceStone)</p> <p>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.</p>
<i>Memento pattern</i>	<p>Used in:- GameState (for Save/Load)</p> <p>Why:- GameState is used to save and load game state and is serializable. could formalize the Memento structure to increase its encapsulation and extensibility.</p>