OOP design patterns and principles used in this implementation:

* **Abstraction**:
  + The Player class is abstract, providing a common interface for different player types.
  + Why: It allows for different implementations of players while ensuring they all have the necessary methods.
* **Encapsulation**:
  + Each class encapsulates its data and behavior.
  + Why: It helps in maintaining the integrity of the data and provides a clear interface for interaction.
* **Inheritance**:
  + HumanPlayer and ComputerPlayer inherit from the Player class.
  + Why: It promotes code reuse and allows for specialized implementations of different player types.
* **Polymorphism**:
  + The GetMove method is implemented differently in HumanPlayer and ComputerPlayer.
  + Why: It allows for different behavior based on the player type while using a common interface.
* **Single Responsibility Principle**:
  + Each class has a single, well-defined responsibility (e.g., Board manages the game board, Game manages the game flow).
  + Why: It makes the code more modular, easier to understand, and easier to maintain.
* **Open/Closed Principle**:
  + The design allows for easy addition of new player types or game variations without modifying existing code.
  + Why: It makes the system more extensible and reduces the risk of introducing bugs in existing functionality.
* **Strategy Pattern**:
  + Different player types (Human and Computer) implement different strategies for making moves.
  + Why: It allows for easy swapping of player types and addition of new player strategies.
* **Composition**:
  + The Game class composes Board, Player, and Piece objects.
  + Why: It creates a flexible structure that can be easily modified or extended.
* **Factory Method (simplified)**:
  + The Game class creates player objects based on user input.
  + Why: It centralizes object creation and allows for easy modification of the creation process.
* **Iterator Pattern (implicit)**:
  + The use of List<Board> and iterating over it in various methods.
  + Why: It provides a standard way to access elements of the collection without exposing its underlying structure.

These patterns and principles work together to create a flexible, maintainable, and extensible design for the Notakto game. They allow for easy modifications, such as adding new player types or changing game rules, while keeping the core structure intact.