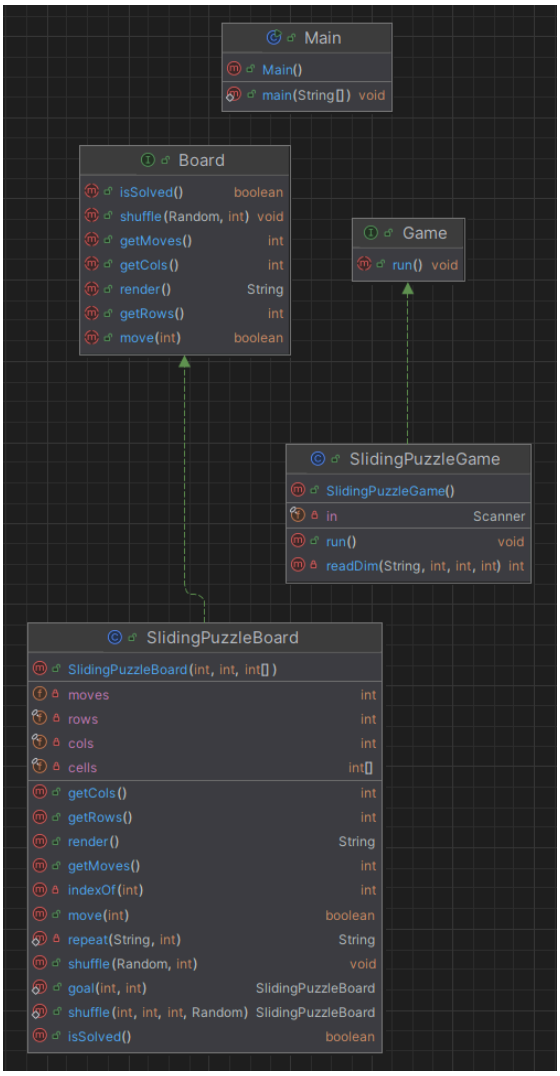

This design is both scalable and extensible because it separates responsibilities and uses interfaces. The Board interface defines a consistent format for any board-based game (move, isSolved, shuffle, render), and Game defines the game loop. This means we can add new games or new board implementations later without changing the interfaces.

Scalability is handled by storing the board as a 1D array with dynamic rows and cols. All logic is derived from these values, so the program works for any m×n board size. Performance also scales well since moves are just index lookups and swaps. I have also accounted for boards up to 5x5, and can easily increase the max size within the code.

Extensibility comes from the clean separation of states and I/O. SlidingPuzzleBoard manages the puzzle rules, while SlidingPuzzleGame handles input and output. This makes it easy to add features like move counters or a timer later by reusing the same board logic. Furthermore, because render() returns a string, the output could be adapted for different games.