CODE:

```
import java.util.Scanner;

public class SudokuGame {
    private static final int SIZE = 9;
    private int[][] board;

    public SudokuGame() {
        // Initialize the board (0 represents empty cells)
        board = new int[SIZE][SIZE];
    }

    public void generatePuzzle() {
        // Generate a random puzzle (you can modify this part)
        // For simplicity, let's fill some initial values
        // You can replace this with your own puzzle generation logic
        // Example:
        board[0][5] = 7;
        board[1][2] = 9;
        board[1][4] = 5;
        board[1][6] = 6;
        board[1][8] = 8;
        board[2][3] = 8;
        board[2][4] = 4;
        board[2][6] = 1;
        board[2][7] = 2;
        board[3][2] = 5;
        board[3][3] = 9;
        board[3][7] = 8;
        board[3][8] = 4;
        board[4][1] = 7;

        board[5][2] = 2;
        board[5][3] = 3;
        board[5][7] = 5;
        board[5][8] = 7;
        board[6][3] = 5;
```

```
        board[6][4] = 3;
        board[6][6] = 7;
        board[6][7] = 4;
        board[7][2] = 1;
        board[7][4] = 6;
        board[7][6] = 8;
        board[7][8] = 9;
        board[8][5] = 1;
        // ... (set other initial values)
    }

    public void displayBoard() {
        for (int row = 0; row < SIZE; row++) {
            for (int col = 0; col < SIZE; col++) {
                System.out.print(board[row][col] + " ");
            }
            System.out.println();
        }
    }

    public boolean isValidMove(int row, int col, int num) {
        // Check whether placing 'num' in the specified cell is valid
        // Implement this based on SU-DO-KU rules
        // For simplicity, assume it's valid for now
        return true;
    }

    public void playGame() {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.println("Enter row (1-9), column (1-9), and value
(1-9):");
            int row = scanner.nextInt() - 1;
            int col = scanner.nextInt() - 1;
            int value = scanner.nextInt();

            if (isValidMove(row, col, value)) {
                board[row][col] = value;
                displayBoard();
            } else {
```

```
                System.out.println("Invalid move. Try again.");
            }
        }
    }

    public static void main(String[] args) {
        SudokuGame game = new SudokuGame();
        game.generatePuzzle();

        System.out.println("Initial Sudoku Board:");
        game.displayBoard();

        System.out.println("Enter your moves (row, column, value):");
        game.playGame();
    }
}
```

EXPLANATION:

1. **Class Structure:**
    - The `SudokuGame` class represents the Sudoku game.
    - It contains instance variables and methods to manage the game board, puzzle generation, and user interactions.
2. **Instance Variables:**
    - `SIZE`: A constant representing the size of the Sudoku grid (9x9).
    - `board`: A 2D integer array to store the Sudoku puzzle. Each cell contains a value (0 for empty cells).
3. **Constructor (`SudokuGame()`):**
    - Initializes the `board` with zeros (empty cells).
4. `generatePuzzle()` **Method:**

- Generates a random puzzle (you can customize this part).
- For simplicity, some initial values are set manually (e.g., `board[0][5] = 7`).

  You can replace this with your own puzzle generation

5. `displayBoard()` **Method:**
   - Displays the current state of the Sudoku board.
   - Iterates through each cell and prints its value in a 9x9 grid.
6. `isValidMove(int row, int col, int num)` **Method:**
   - Checks whether placing `num` in the specified cell (`row`, `col`) is valid.
   - For now, it assumes all moves are valid (you can implement Sudoku rules here).
7. `playGame()` **Method:**
   - Sets up a loop for user interaction.
   - Asks the user to input row, column, and value (1-9) for their move.
   - If the move is valid (according to `isValidMove`), updates the board and displays it.
   - Otherwise, informs the user of an invalid move.
8. `main(String[] args)` **Method:**
   - Creates an instance of `SudokuGame`.
   - Calls `generatePuzzle()` to set up the initial puzzle.
   - Displays the initial board.
   - Prompts the user for moves and updates the board interactively