项目更新说明

執行 Java 程式

```
java -cp bin App
```

概述

在这个项目中，我们实现了一个支持同时维护三个棋盘的游戏。用户可以选择输入下棋的位置或者棋盘号。以下是我们所做的主要更改：

- **同时维护三个棋盘**：游戏现在支持三个独立的棋盘，每个棋盘可以独立进行游戏。
- **用户输入处理**：用户可以输入下棋的位置（如 "1a"）或棋盘号（如 "1"）。输入棋盘号时，游戏会切换到相应的棋盘。
- **保持棋盘状态**：每次切换棋盘时，保持上一次下棋的状态。
- **默认棋盘初始化**：游戏开始时，默认初始化棋盘为1。
- **输入区分**：通过字符串长度来区分下棋的位置和棋盘号。

代码修改

1. `Board` 类

我们在 `Board` 类中引入了多个棋盘的支持，并添加了一个方法来切换当前棋盘。

```
public static Piece[][][] boards = new Piece[NUM_BOARDS][SIZE][SIZE];
public static int currentBoardIndex = 0;

public static void switchBoard(int boardIndex) {
    if (boardIndex < 0 || boardIndex >= NUM_BOARDS) {
        throw new IllegalArgumentException("Invalid board index");
    }
    currentBoardIndex = boardIndex;
}
```

解释：我们增加了一个三维数组 `boards` 来存储多个棋盘，并通过 `currentBoardIndex` 来跟踪当前活动的棋盘。`switchBoard` 方法用于切换当前棋盘。

2. `GameEngine` 类

在 `GameEngine` 类中，我们引入了一个数组来跟踪每个棋盘的当前玩家。

```
private int[] currentPlayerIndices; // Array to track current player for
each board
```

```
public GameEngine(Player blackPlayer, Player whitePlayer, Scanner scanner)
{
    this.players = new Player[]{blackPlayer, whitePlayer};
    this.scanner = scanner;
    this.currentPlayerIndices = new int[Board.NUM_BOARDS]; // Initialize
to 0 for all boards
}

private void handleTurn(Player player) {
    int[] input = InputUtils.readValidInput(scanner, board,
players[currentPlayerIndices[Board.currentBoardIndex]].pieceType);
    if (input[1] == -1) {
        Board.switchBoard(input[0]);
        System.out.println("Switched to board " + (input[0] + 1));
    } else {
        int row = input[0];
        int col = input[1];
        Board.boards[Board.currentBoardIndex][row][col] =
players[currentPlayerIndices[Board.currentBoardIndex]].pieceType;
    }
}
```

解释：我们引入了 `currentPlayerIndices` 数组来跟踪每个棋盘的当前玩家。`handleTurn` 方法根据用户输入来决定是切换棋盘还是进行下棋操作。

3. `InputUtils` 类

在 `InputUtils` 类中，我们将棋子放置的验证逻辑移到了 `readValidInput` 方法中。

```
public static int[] readValidInput(Scanner scanner, Board board, Piece
piece) {
    while (true) {
        String input = scanner.nextLine().trim();
        try {
            int[] move = parseInput(input);
            if (move[1] == -1) {
                return move;
            }
            int row = move[0];
            int col = move[1];
            if (board.getWhatPiece(row, col) != Piece.EMPTY) {
                System.out.println("This position cannot be placed. Please
try again.");
                continue;
            }
            return move;
        } catch (IllegalArgumentException e) {
            System.out.println("Invalid input format. Please enter a
number (1-8) followed by a letter (A-H), or a board number (1-3).");
        }
```

```
    }
}
```

解释：`readValidInput` 方法现在负责验证用户输入是否可以在棋盘上放置棋子，并根据输入的长度来区分是棋盘号还是下棋位置。

4. `GameSetup` 类

我们创建了一个新的 `GameSetup` 类来处理玩家初始化。

```
public static Player[] initializePlayers(Scanner scanner) {
    Player[] players = new Player[2]; // Two players for all boards

    System.out.print("Please enter the first player name (Using the black
piece ●): ");
    String player1Name = scanner.nextLine().trim();
    players[0] = new Player(player1Name, Piece.BLACK);

    System.out.print("Please enter the second player name (Using the white
piece ○): ");
    String player2Name = scanner.nextLine().trim();
    players[1] = new Player(player2Name, Piece.WHITE);

    return players;
}
```

解释：`GameSetup` 类负责初始化玩家信息，确保所有棋盘使用相同的玩家。

5. `App` 类

在 `App` 类中，我们使用 `GameSetup` 类来初始化玩家。

```
Player[] players = GameSetup.initializePlayers(scanner);
```

解释：`App` 类通过调用 `GameSetup.initializePlayers` 方法来获取玩家信息，并启动游戏。

6. 游戏结束条件关键代码解释

以下代码确保游戏的结束条件，即当所有棋盘都填满时，游戏结束。

```
if (!GameEngine.isGameOver()){
            if (Board.isBoardFull(Board.currentBoardIndex)){
                System.out.print("Board " + (Board.currentBoardIndex + 1)
+ " is full now. Please enter another board number to continue.");
                return;
            }
            System.out.print("Player " + currentPlayer.getName() + ",
```

```
please enter your move or board number:");
        }
```

```
static boolean isGameOver() {
        for (int i = 0; i < Board.NUM_BOARDS; i++){
            if(!Board.isBoardFull(i)){
                return false;
            }
        }return true;
    }
```

```
public static boolean isBoardFull(int boardIndex) {
    for (int i = 0; i < Board.SIZE; i++) {
        for (int j = 0; j < Board.SIZE; j++) {
            if (boards[boardIndex][i][j] == Piece.EMPTY) {
            return false;
            }
        }
    }
    return true;
}
```

解释： `isBoardFull` 遍历单个棋盘，若存在 0（表示可落子），则返回 false，否则返回 true。 `isGameOver` 遍历所有棋盘，只要有一个棋盘未满，则返回 false，否则返回 true，表示游戏结束。这样保证了游戏在所有棋盘填满后才会结束，而非某个棋盘填满时立即终止。

这些更改确保了游戏可以同时维护三个棋盘，并且每个棋盘都有独立的当前玩家状态。用户可以通过输入棋盘号来切换棋盘，游戏会保持每个棋盘的状态。