# 黑白棋游戏系统实验报告

## 项目介绍

本项目实现了一个支持两种游戏模式的黑白棋游戏系统，包括Peace模式（简单棋子放置）和Reversi模式（经典黑白棋）。系统支持两个游戏棋盘之间的切换，并实现了完整的游戏逻辑与界面交互。

## 源代码文件及主要功能

| 文件名 | 主要功能 |

|------|--------|

| `Piece.java` | 棋子枚举类，定义了黑子、白子和空位，并提供对应的符号 |

| `Player.java` | 玩家类，包含玩家名称和棋子颜色信息 |

| `Board.java` | 棋盘类，实现8×8棋盘的数据结构和基本操作 |

| `Game.java` | 游戏抽象基类，定义了游戏的共同属性和方法 |

| `PeaceGame.java` | Peace模式游戏类，实现简单棋子放置规则 |

| `ReversiGame.java` | Reversi模式游戏类，实现完整的黑白棋规则 |

| `GameManager.java` | 游戏管理器，负责管理两种游戏模式和游戏切换 |

| `ChessGame.java` | 主应用类，包含界面显示和用户交互逻辑 |

## 关键代码及设计思路

#### 1. 棋子与玩家设计

```java

// Piece.java - 使用枚举表示棋子类型

public enum Piece {

    BLACK("●"),

    WHITE("○"),

    EMPTY("·");

    private final String symbol;

    Piece(String symbol) {

        this.symbol = symbol;

    }

    public String getSymbol() {

        return symbol;

    }

}

// Player.java - 玩家信息封装

public class Player {

    private String name;

    private Piece piece;

    // 构造函数和getter方法...

}

```

**设计思路**：

- 使用枚举类型表示棋子，便于类型安全和状态管理

- 将棋子与其显示符号关联，简化界面显示逻辑

- 将玩家信息封装为单独类，包含名称和棋子颜色，便于扩展和管理

2. 游戏抽象与多态设计

```java

// Game.java - 游戏抽象基类

public abstract class Game {

    protected int gameId;

    protected String gameType;

    protected Board board;

    protected Player player1;

    protected Player player2;

    protected Player currentPlayer;

    protected boolean gameOver;

    // 构造函数和通用方法...

    // 抽象方法，由具体游戏模式实现

    public abstract boolean placePiece(int row, int col);

    public abstract boolean isGameOver();

    public abstract List<int[]> getValidMoves();

}

// PeaceGame.java和ReversiGame.java分别实现这些抽象方法

```

**设计思路**：

- 使用抽象类定义游戏的通用属性和行为

- 利用多态实现不同游戏模式的特定规则

- 关键抽象方法包括：落子逻辑、游戏结束判断和有效落子位置计算

- 共享代码放在基类中，特定逻辑在子类中实现

#### 3. Reversi模式的核心算法

```java

// ReversiGame.java - 黑白棋规则实现

public boolean isValidMove(int row, int col, Piece piece) {

    if (!board.isInBoard(row, col) || !board.isEmpty(row, col)) {

        return false;

    }

    Piece opponentPiece = (piece == Piece.BLACK) ? Piece.WHITE : Piece.BLACK;

    // 检查8个方向

    for (int[] dir : DIRECTIONS) {

        int dr = dir[0];

        int dc = dir[1];

        int r = row + dr;

        int c = col + dc;

        boolean hasOpponentPiece = false;

        // 沿着当前方向移动

        while (board.isInBoard(r, c) && board.getPiece(r, c) == opponentPiece) {

            hasOpponentPiece = true;

            r += dr;

            c += dc;

        }

        // 如果这个方向上有对手的棋子，且最后是自己的棋子，则是合法落子位置

        if (hasOpponentPiece && board.isInBoard(r, c) && board.getPiece(r, c) == piece) {

            return true;

        }

    }

    return false;

}

// 实现棋子翻转逻辑

private void flipPieces(int row, int col) {

    Piece currentPiece = currentPlayer.getPiece();

    Piece opponentPiece = (currentPiece == Piece.BLACK) ? Piece.WHITE : Piece.BLACK;

    // 检查8个方向，翻转夹在中间的对手棋子

    for (int[] dir : DIRECTIONS) {

        // 方向搜索和翻转逻辑...

    }

}

```

**设计思路**：

- 使用方向数组表示8个搜索方向，简化代码结构

- 落子有效性判断：检查是否能夹住对手棋子

- 翻转算法：先搜索可能需要翻转的棋子，再执行翻转

- 传统黑白棋规则的完整实现，包括pass机制

#### 4. 游戏管理器设计

```java

// GameManager.java - 管理两种游戏模式

public class GameManager {

    private List<Game> games;

    private int currentGameIndex;

    public GameManager() {

        games = new ArrayList<>();

        // 初始化两个游戏，只有这两个

        games.add(new PeaceGame(1)); // 游戏1: Peace模式

        games.add(new ReversiGame(2)); // 游戏2: Reversi模式

        // 确保默认进入模式1

        currentGameIndex = 0;

    }

    public boolean switchGame(int gameId) {

        // 只支持1和2两个游戏编号

        if (gameId == 1) {

            currentGameIndex = 0;

            return true;

        } else if (gameId == 2) {

            currentGameIndex = 1;

            return true;

        }

        return false;

    }

    // 其他方法...

}

```

**设计思路**：

- 使用列表存储两种游戏模式的实例

- 通过索引跟踪当前活动游戏

- 提供简单的切换机制，直接通过游戏ID切换

- 委托调用当前游戏的方法，实现统一接口

#### 5. 用户界面与交互

```java

// ChessGame.java - 用户界面与交互

private void displayGame() {

    // 获取当前游戏和状态

    Game currentGame = gameManager.getCurrentGame();

    Board board = currentGame.getBoard();

    List<int[]> validMoves = currentGame.getValidMoves();

    // 显示棋盘、游戏状态和可用命令

    // ...

    // 在Reversi模式中，显示合法落子位置

    for (int j = 0; j < Board.SIZE; j++) {

        if (currentGame.getGameType().equals("reversi") && isValidMove(validMoves, i, j)) {

            System.out.print("+ ");

        } else {

            System.out.print(board.getPiece(i, j).getSymbol() + " ");

        }

    }

    // 显示其他游戏信息...

}

public void start() {

    // 游戏主循环，处理用户输入

    while (!quit) {

        clearScreen();

        displayGame();

        // 获取用户输入并处理...

        // 处理各种命令：落子、切换游戏、pass、退出等

    }

}

```

**设计思路**：

- 三列排列,简洁直观

- 清屏无缝切换

## 设计模式应用

1. **策略模式**：使用Game抽象类和具体游戏实现类，可以无缝切换不同的游戏规则

2. **状态模式**：游戏状态的管理，包括当前玩家、游戏是否结束等

3. **单一职责原则**：每个类都有明确的职责，如Board负责棋盘，Game负责规则

4. **开闭原则**：系统对扩展开放，可以方便地添加新的游戏模式

## 项目收获与总结

1. **面向对象设计实践**：通过抽象、封装和多态实现了灵活的系统架构

2. **算法实现**：实现了黑白棋的翻转算法和游戏规则判断

3. **用户界面设计**：设计了清晰、易用的控制台界面

4. **错误处理**：实现了良好的错误处理和用户提示机制

本项目通过合理的类设计和游戏逻辑实现，成功构建了一个完整的黑白棋游戏系统，实现了两种游戏模式的支持和无缝切换，满足了所有需求规格。

## 附:运行截图:

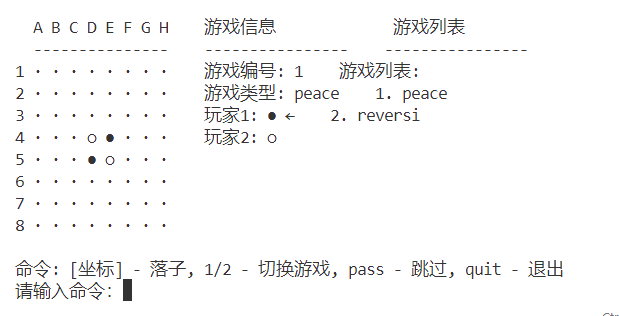


表 1初始

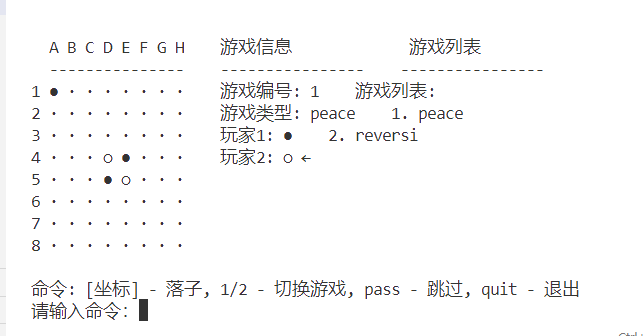


表 2输入1a

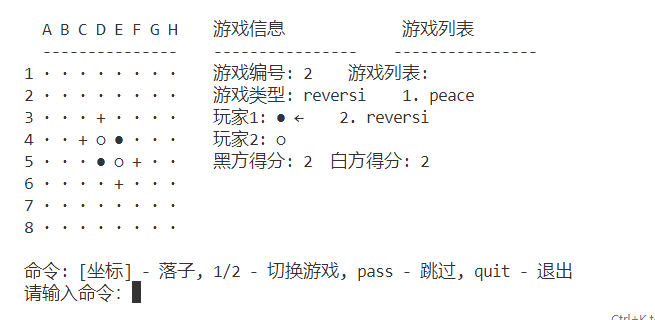


表 3 输入2

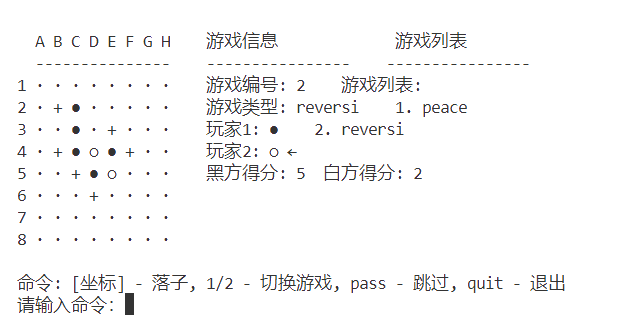


表 4 游戏规则正常运行

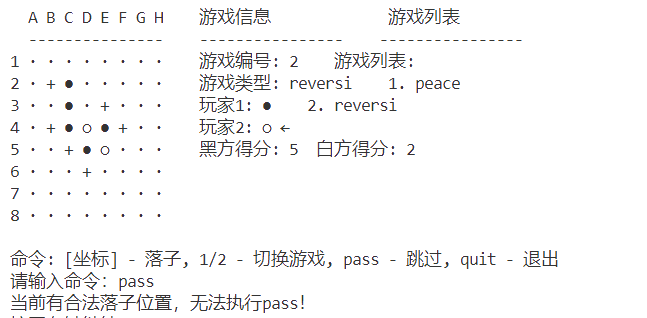


表 5 pass

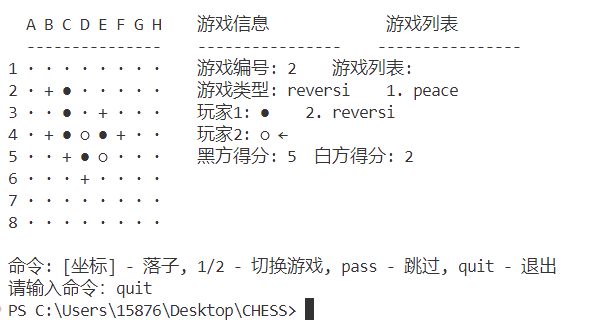


表 6 quit

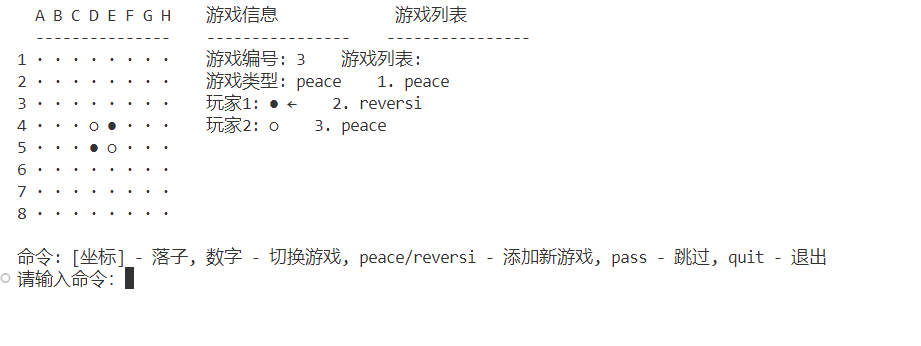


表 7 输入peace

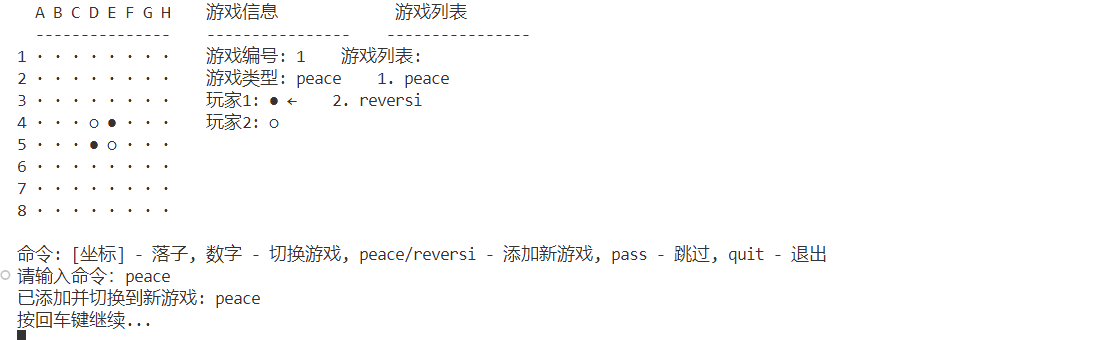


表 8 成功添加

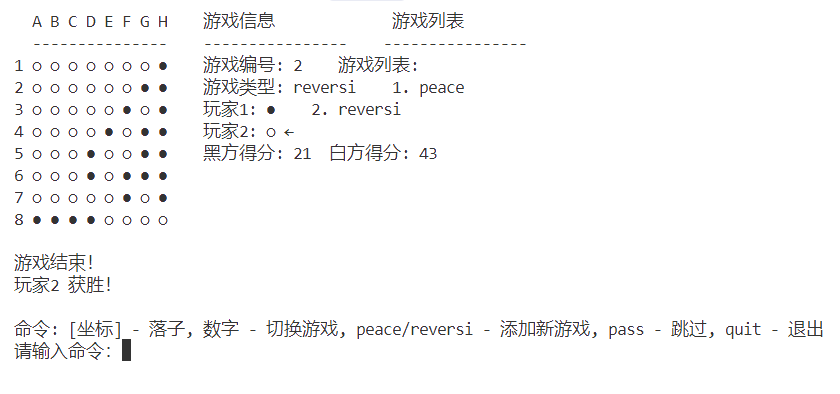


表 9 2.游戏结束

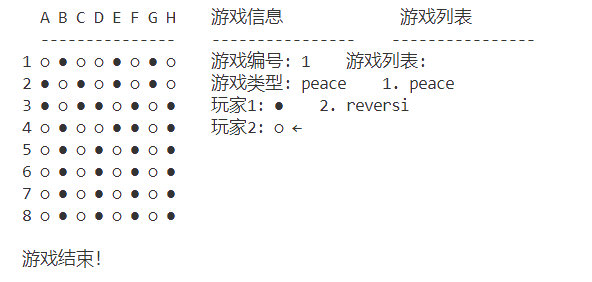


表 10 1.游戏结束