軟體框架設計

HW2 象棋

姓名：林鈺凱，資訊三甲

學號：D1149371

目錄

一、題目

撰寫一個簡單版、 GUI 介面的象棋翻棋遊戲系統，32 個棋子會隨機的落在 4*8 的棋盤上。透過 Chess 的建構子產生這些棋子並隨機編排位置，再印出這些棋子的名字、位置。

使用者可以在 GUI 介面點擊棋子的位置進行移動，若該位置的棋子未翻開則翻開，若已翻開則可以選擇其他位置進行移動或吃子，如果不成功則系統提示錯誤回到原來狀態。每個動作都會重新顯示棋盤狀態。

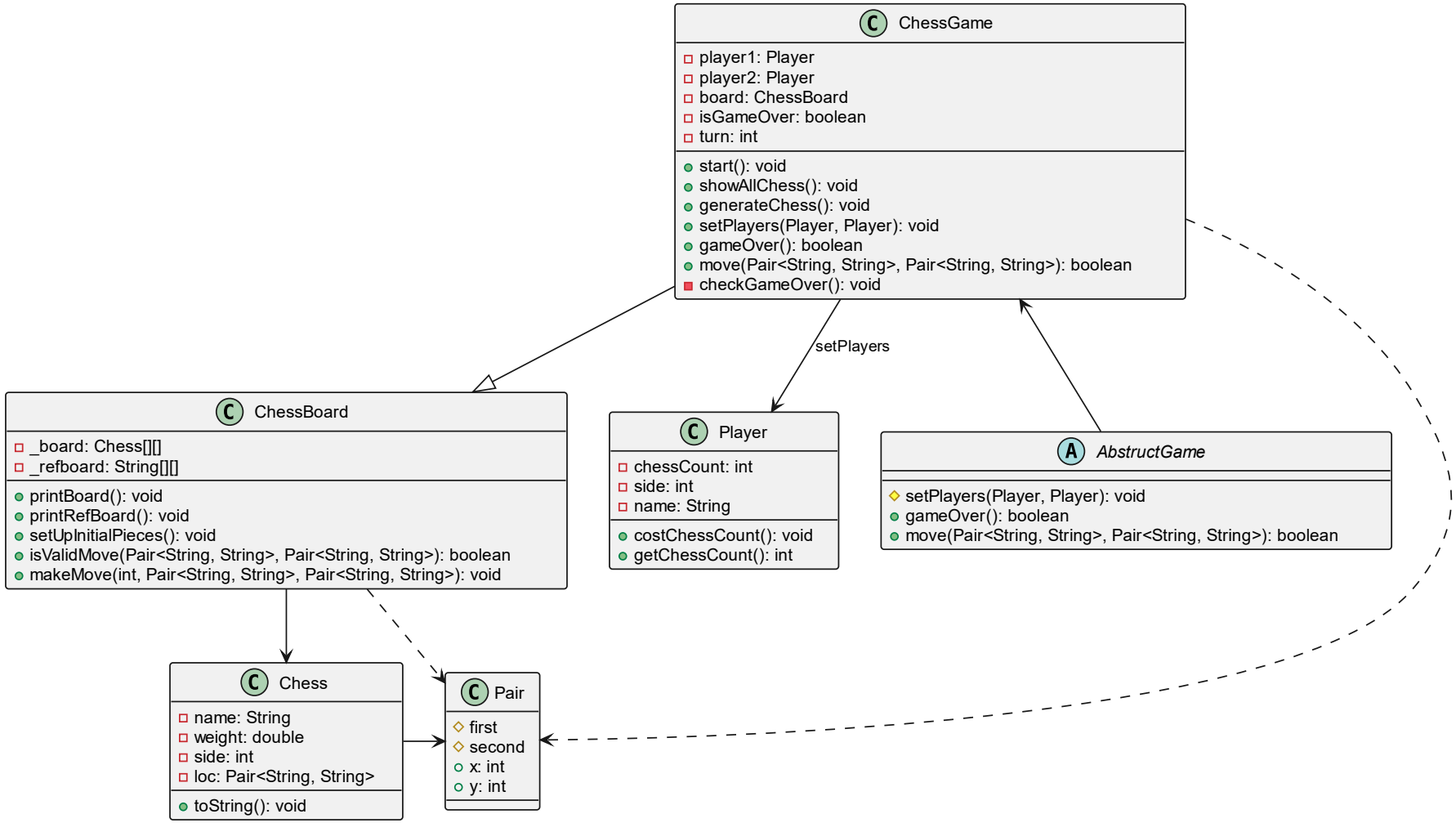
二、設計方法

設計 ChessGameGUI 結合 HW01 的 Player、Chess、ChessGame、ChessBoard class，以 ChessGameGUI 作為進入點，開始遊戲過程。

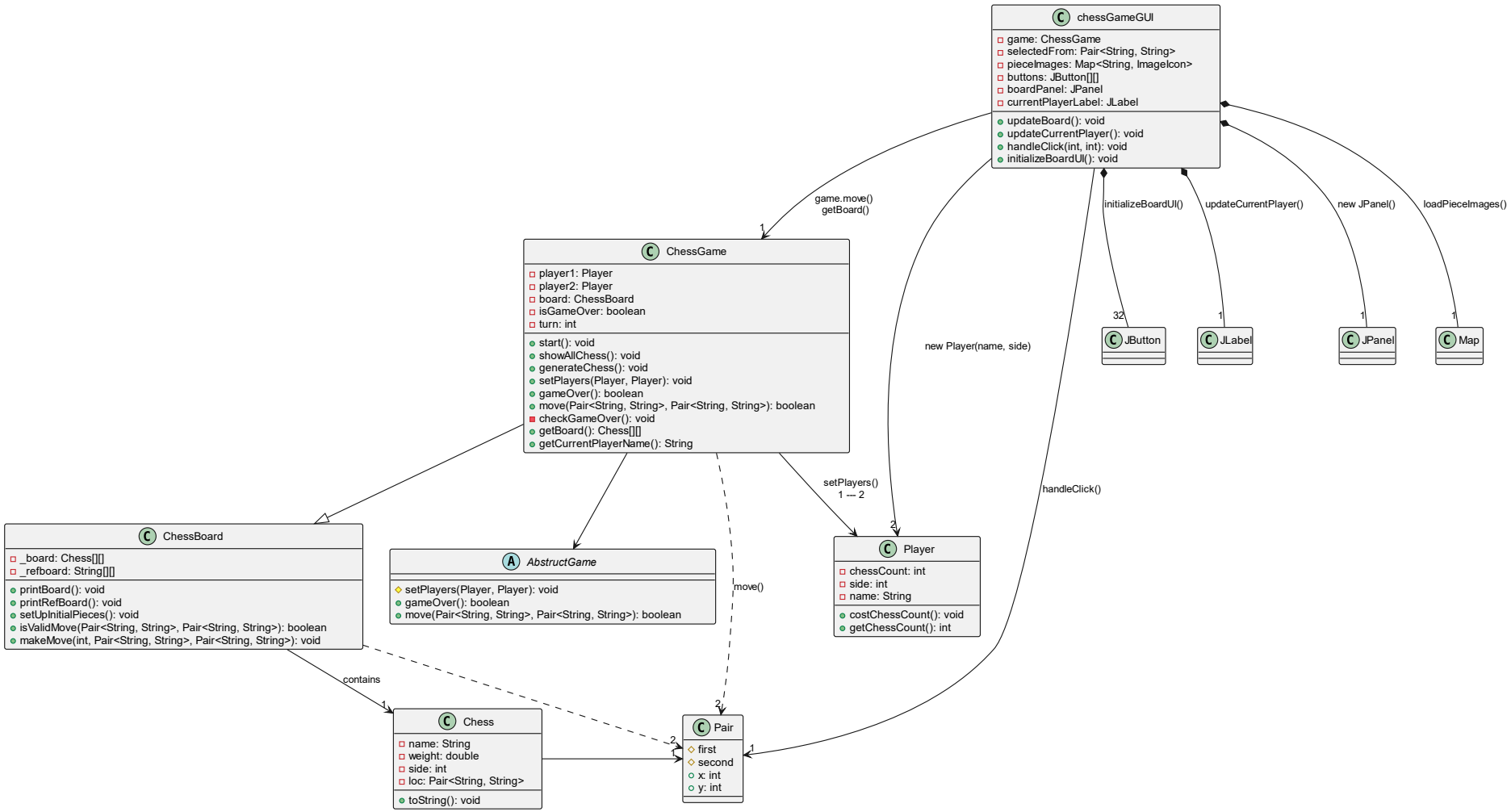
以下為 plantUML 語法

```
class chessGameGUI {
    - game: ChessGame => 紀錄一局遊戲的所有狀態
    - selectedFrom: Pair<String, String> => 被選取棋子的位置
    - pieceImages: Map<String, ImageIcon> => 存放棋子的圖片
    - buttons: JButton[][] => 儲存作為棋子的按鈕 GUI
    - boardPanel: JPanel => 棋盤 GUI
    - currentPlayerLabel: JLabel => 顯示目前玩家的 GUI
    ---
    + updateBoard(): void => 更新棋盤
    + updateCurrentPlayer(): void => 更新現在的玩家
    + handleClick(int, int): void => 處理點擊的事件
    + initializeBoardUI(): void => 初始化棋盤 GUI
}
```

ConsoleChessGame UML



ChessGameGUI UML



三、程式碼

完整程式碼連結

chessGameGUI 的建構子，用來建構棋盤 GUI 需要的物件

```
public chessGameGUI() {

    setTitle("象棋遊戲");
    setSize(800, 400);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setLayout(new BorderLayout());

    Player player1 = new Player("Player 1", 0);
    Player player2 = new Player("Player 2", 1);
    game = new ChessGame(player1, player2);

    boardPanel = new JPanel(new GridLayout(4, 8));
    buttons = new JButton[4][8];

    initializeBoardUI();

    add(boardPanel, BorderLayout.CENTER);
    setVisible(true);
    loadPieceImages();

    currentPlayerLabel = new JLabel();
    currentPlayerLabel.setFont(new Font("Serif", Font.BOLD, 18));
    currentPlayerLabel.setHorizontalAlignment(SwingConstants.CENTER);
    add(currentPlayerLabel, BorderLayout.NORTH);
    updateCurrentPlayer();

}
```

載入棋子需要用的圖片資源

```
private void loadPieceImages() {
    String[] names = { "car", "horse", "elephant", "guard", "king", "cannon", "pawn" };
    for (String name : names) {
        pieceImages.put(name + "_black", new ImageIcon("resources/images/" + name + "_black.png"));
        pieceImages.put(name + "_red", new ImageIcon("resources/images/" + name + "_red.png"));
    }
}
```

更新現在的玩家

```
private void updateCurrentPlayer() {
    String name = game.getCurrentPlayerName();
    currentPlayerLabel.setText("目前輪到: " + name);
}
```

初始化棋盤 GUI，包含棋子按鈕、棋盤寬度與長度

```
private void initializeBoardUI() {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 8; j++) {
            JButton button = new JButton();
            button.setFont(new Font("Serif", Font.PLAIN, 18));
            int row = i;
            int col = j;

            button.addActionListener(e -> handleClick(row, col));

            buttons[i][j] = button;
            boardPanel.add(button);
        }
    }

    updateBoard();
}
```

縮放棋子圖片的 Icon，可以適應棋盤大小

```
private ImageIcon getScaledIcon(ImageIcon icon, int width, int height) {
    Image image = icon.getImage();
    Image scaledImage = image.getScaledInstance(width, height, Image.SCALE_SMOOTH);
    return new ImageIcon(scaledImage);
}
```

處理點擊事件

```
private void handleClick(int row, int col) {
    String rowChar = String.valueOf((char) (row + 'A'));
    String colStr = String.valueOf(col + 1);
    Pair<String, String> clicked = new Pair<>(rowChar, colStr);

    if (selectedFrom == null) {
        selectedFrom = clicked;
    } else {
        game.move(selectedFrom, clicked);
        selectedFrom = null;
        updateBoard();
        updateCurrentPlayer();
        if (game.gameOver()) {
            JOptionPane.showMessageDialog(this, "Game Over!");
        }
    }
}
```

根據棋子的名稱轉換為對應的圖片名稱

```
private String mapChineseNameToKey(String name) {
    switch (name) {
        case "車":
        case "俥":
            return "car";
        case "馬":
        case "傌":
            return "horse";
        case "象":
        case "像":
            return "elephant";
        case "士":
        case "仕":
            return "guard";
        case "將":
        case "帥":
            return "king";
        case "包":
        case "砲":
            return "cannon";
        case "卒":
        case "兵":
            return "pawn";
        default:
            return "pawn"; // fallback
    }
}
```

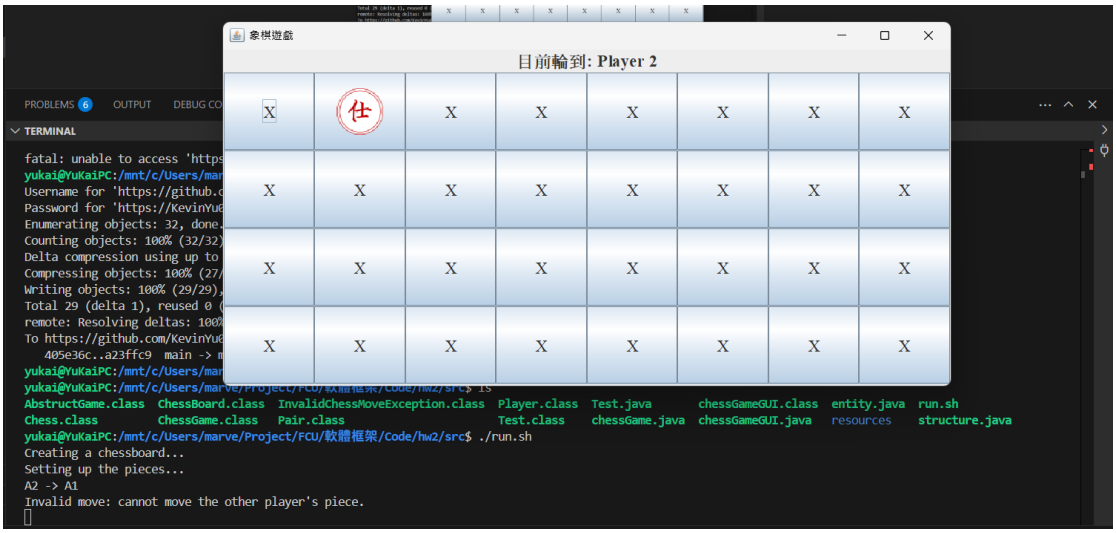
更新棋盤

```
private void updateBoard() {
    Chess[][] board = game.getBoard();
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 8; j++) {
            JButton button = buttons[i][j];
            if (board[i][j] != null) {
                if (board[i][j].name.equals("X")) {
                    // 未翻開的棋子，不顯示圖像
                    button.setIcon(null);
                    button.setText("X");
                } else {
                    String pieceName = board[i][j].name;
                    int side = board[i][j].side;
                    String key = mapChineseNameToKey(pieceName) + (side == 0 ? "_black" : "_red");

                    ImageIcon icon = pieceImages.get(key);
                    if (icon != null) {
                        icon = getScaledIcon(icon, 50, 50); // 控制顯示大小（你可以調整這個數字）
                        button.setIcon(icon);
                        button.setText("");
                    }
                }
            } else {
                button.setIcon(null);
                button.setText("");
            }
        }
    }

}
```

四、執行畫面

遊戲開始畫面，可以看見所有棋子呈現 X，表示朝下覆蓋。如圖

遊玩過程圖，Player1 翻到了紅方的棋，所以無效移動，下方 Console 也打印出這是個 Invalid move，其理由為不能移動其他玩家的棋子。如圖



遊玩過程圖：如圖

五、心得感想

這次使用 Java Swing 的元件製作 GUI 給使用者更好的使用體驗，基本上使用 HW01 的 Class 來建出 ChessGameGUI class，省下需要 Code Refactor 的時間成本，另外使用每張圖片顯示棋子，讓整體的使用體驗更好。在建立 UML 圖與類別間的關係時，能幫助我清楚區分出 Aggregation、Composition 以及 Association 的差異。而透過這次 GUI 的設計，我實際操作了 JLabel、JPanel、JButton 等元件，也體會到介面互動與邏輯分離的重要性。在處理棋盤狀態更新、玩家切換與勝負判斷時，我學會如何將畫面與後端邏輯有效結合，讓整體遊戲流程更順暢