軟體框架設計

HW1 象棋

姓名：林鈺凱，資訊三甲

學號：D1149371

一、題目

撰寫一個簡單版、非 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 console 介面輸入所要選擇的棋子的位置 (例如 A2, B3)，若該位置的棋子未翻開則翻開，若以翻開則系統要求輸入目的的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

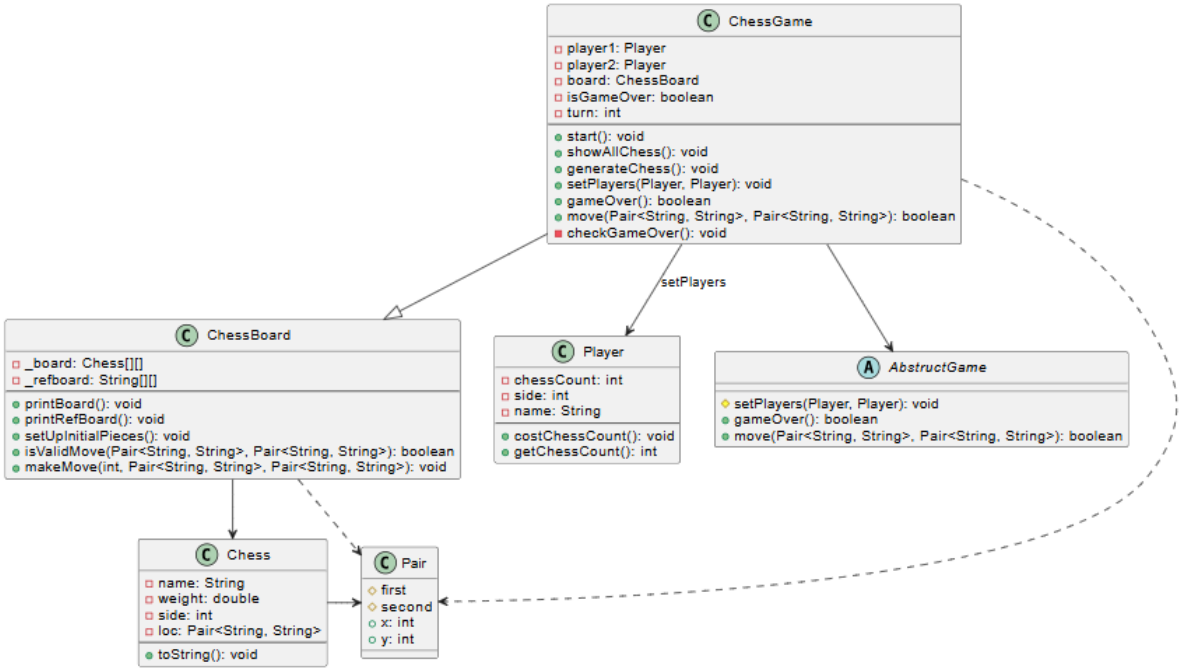
設計 Player、Chess、ChessGame、ChessBoard 作為主要的 class，另外定義 abstruct class 的 AbstructGame，並由 ChessGame 繼承，以 ChessGame 作為進入點，設計兩個 Player attribute，以及 ChessBoard attribute 紀錄棋盤的狀態，而在 ChessBoard 設計 _board attribute 來記錄所有 Chess，不過因為一開始不會知道每一格的 Chess name，所以額外設計 _refboard 紀錄實際上的棋盤佈局，當翻開棋子後才會將 _refboard 紀錄的棋子名稱替換進 _board 對應的位置中。於移動功能，設計兩個參數輸入來源與目的，並以 Pair<String, String> 自定義的結構儲存該變數，如果有棋子被吃掉，就會將對應的 Player 中 chessCount（用來記錄還有多少棋子）減少，並檢查是否其中一個 Player 的棋子都被吃完。以下為完整的 UML。

Class

- ChessGame
- ChessBoard
- Player
- Chess
- Player

Abstruct Class

- AbstructGame



三、程式碼

完整程式碼連結

以下為 Player 和 Chess 的類別，當中將 Player 的屬性封裝，避免意外修改 Player 的屬性

```
class Player {
    private int chessesCount = 16;
    int side; // 0: white, 1: black
    String name;

    Player(String name, int side) {
        this.name = name;
        this.side = side;
    }

    void costChessCount() {
        this.chessesCount--;
    }

    int getChessCount() {
        return this.chessesCount;
    }
}
```

```
class Chess {
    String name;
    double weight;
    int side; // 0: white, 1: black
    Pair<String, String> loc; // column, row

    Chess(String name, double weight, int side, Pair<String, String> loc) {
        this.name = name;
        this.weight = weight;
        this.side = side;
        this.loc = loc;
    }

    @Override
    public String toString() {
        return String.format("棋型: %s\n等級: %s\n陣營: %s\n位置: %s\n", name, weight, side, loc.first + loc.second);
    }
}
```

此為 AbstructGame、ChessGame 的定義，ChessGame 繼承 AbstructGame 並實作出需要的方法，以及定義一場遊戲的關鍵屬性，如 Player、ChessBoard、turn（現在輪到哪方）、isGameOver（是否結束）

```
abstract class AbstructGame {
    protected abstract void setPlayers(Player player1, Player player2);
    abstract boolean gameOver();
    abstract boolean move(Pair<String, String> from, Pair<String, String> to);
}
```

```
class ChessGame extends AbstructGame {
    private Player player1;
    private Player player2;
    private ChessBoard board;
    private boolean isGameOver;
    private int turn = 0;

    // Constructor
    public ChessGame(Player player1, Player player2) {
        this.board = new ChessBoard();
        this.isGameOver = false;
        setPlayers(player1, player2);
        generateChess();
    }

    ...
}
```

此為遊戲開始的主要邏輯，首先打印目前棋盤狀態 → 等待玩家輸入 → 是否有效移動 → 確認是否遊戲結束

```
void start() {
        Scanner scanner = new Scanner(System.in);

        while (gameOver() == false) {
            try {
                showAllChess();
                System.out.println("請輸入棋盤座標（例如：A2 B2）：");
                String[] input = scanner.nextLine().split(" ");
                if (input.length != 2) {
                    System.out.println("Invalid input.");
                    continue;
                }
                move(new Pair<String, String>(input[0].substring(0, 1), input[0].substring(1)),
                        new Pair<String, String>(input[1].substring(0, 1), input[1].substring(1)));
                checkGameOver();
            } catch (Exception e) {
                break;
            }
        }

        System.out.println("================================");
        System.out.println("Game Over!");
        if (player1.getChessCount() == 0 || player1.getChessCount() < player2.getChessCount()) {
            System.out.println("Player 2 wins!");
        } else if (player2.getChessCount() == 0 || player1.getChessCount() > player2.getChessCount()) {
            System.out.println("Player 1 wins!");
        } else {
            System.out.println("Draw!");
        }
        scanner.close();
}
```

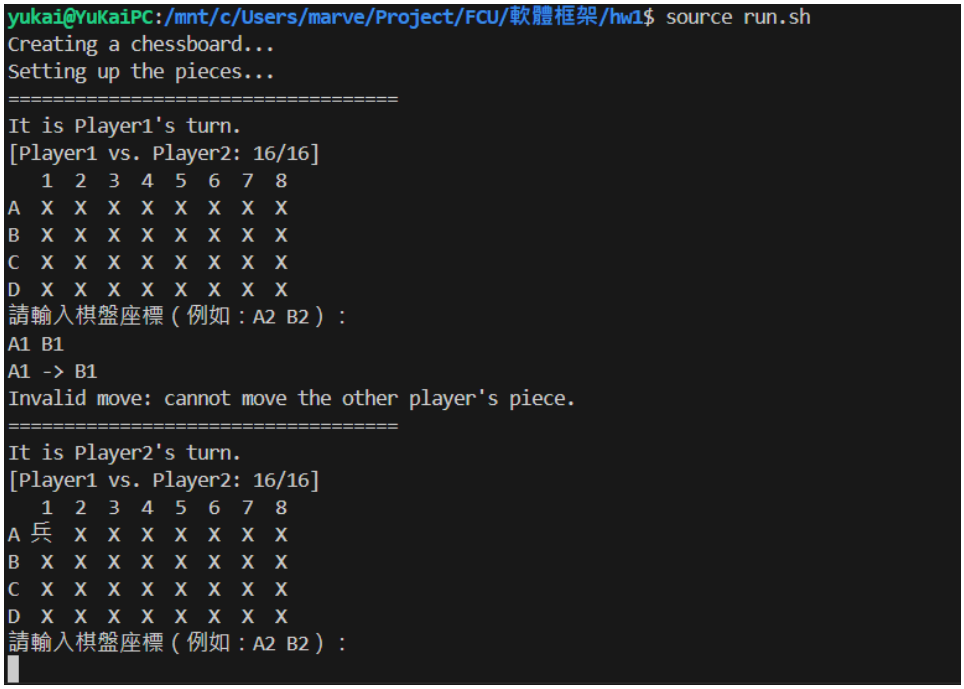
此為棋盤的類別，負責記錄棋盤上所有棋子的狀態，由於一開始會是覆蓋狀態，所以另外寫了 _refboard 來記錄覆蓋下的實際棋子名稱，這樣就不需要額外寫邏輯推出這顆棋子是什麼，另外於實例化時，會開始生成空棋盤，如下 _board 全部的元素為 null，直到被 ChessGame 呼叫 setUpinitialPieces()，才會隨機生成棋子布局

```
class ChessBoard {
    private Chess[][] _board = new Chess[4][8];
    private String[][] _refboard = new String[4][8];

    ChessBoard() {
        System.out.println("Creating a chessboard...");
        for (int i = 0; i < _refboard.length; i++) {
            Arrays.fill(_board[i], null);
        }
    }

    ...
}
```

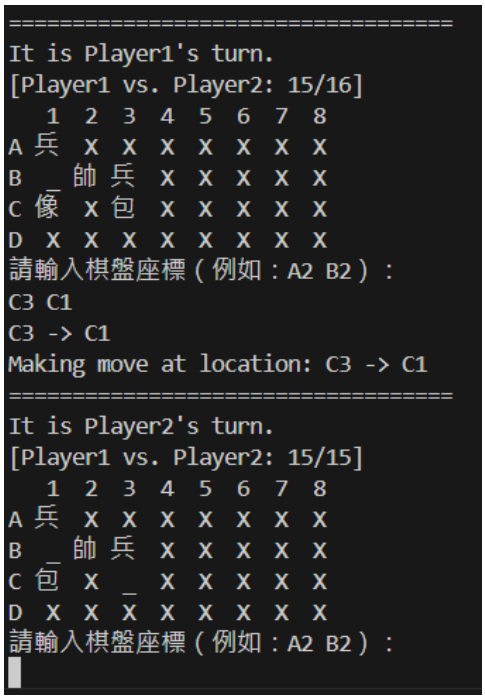
```
void setUpInitialPieces() {
        System.out.println("Setting up the pieces...");
        String[] chessType = { "車", "馬", "象", "士", "將", "士", "象", "馬", "車", "包", "包", "卒", "卒", "卒", "卒", "卒" };
        String[] chessType2 = { "硨", "傌", "像", "仕", "帥", "仕", "像", "傌", "硨", "砲", "砲", "兵", "兵", "兵", "兵", "兵" };
        double[] chessWeight = { 4, 3, 5, 6, 7, 6, 5, 3, 4, 2, 2, 1, 1, 1, 1, 1 };
        int randNum1 = ThreadLocalRandom.current().nextInt(1, 5);
        int randNum2 = ThreadLocalRandom.current().nextInt(1, 9);
        for (int i = 0; i < 16; i++) {
            while (_board[randNum1 - 1][randNum2 - 1] != null) {
                randNum1 = ThreadLocalRandom.current().nextInt(1, 5);
                randNum2 = ThreadLocalRandom.current().nextInt(1, 9);
            }
            _refboard[randNum1 - 1][randNum2 - 1] = chessType[i];
            _board[randNum1 - 1][randNum2 - 1] = new Chess("X", chessWeight[i], 0, new Pair<String, String>(
                    String.valueOf((char) (randNum1 + 64)), String.valueOf(randNum2)));
        }
        for (int i = 0; i < 16; i++) {
            while (_board[randNum1 - 1][randNum2 - 1] != null) {
                randNum1 = ThreadLocalRandom.current().nextInt(1, 5);
                randNum2 = ThreadLocalRandom.current().nextInt(1, 9);
            }
            _refboard[randNum1 - 1][randNum2 - 1] = chessType2[i];
            _board[randNum1 - 1][randNum2 - 1] = new Chess("X", chessWeight[i], 1, new Pair<String, String>(
                    String.valueOf((char) (randNum1 + 64)), String.valueOf(randNum2)));
        }
    }
```

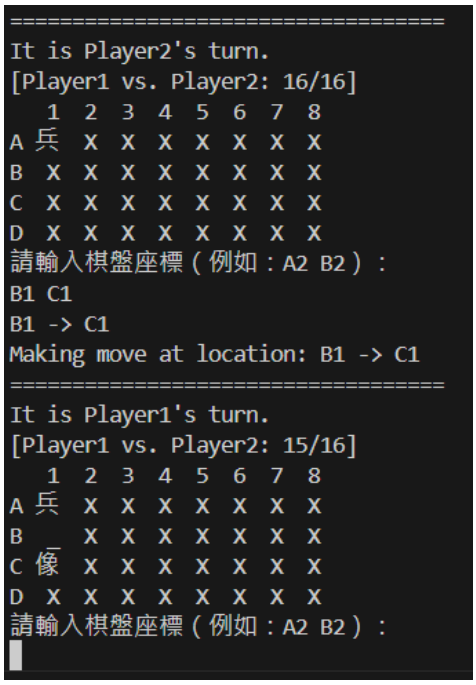
四、執行畫面

此畫面為棋盤第一次生成，所有棋子覆蓋的初始狀態，而 Player1 翻到不是他的棋子，所以打印規則提示

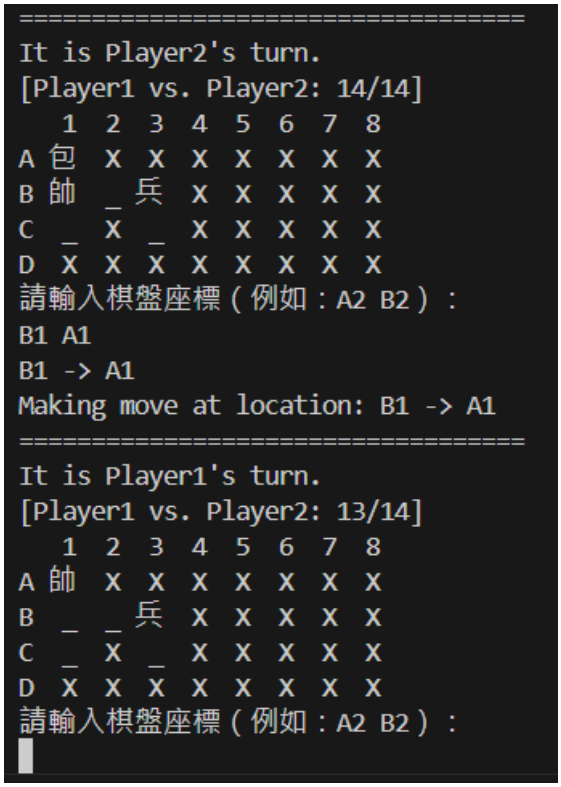


接著，左下這張圖為包可以飛越一顆棋子吃掉敵方棋子的演示，而右下則為 Player2 翻到屬於自己的棋子並吃掉覆蓋的棋子，而這顆棋子屬於 Player1，並且等級比像還低





而此張圖則為 Player2 用帥吃掉包的演示，呈現一般情況下吃棋的過程



五、問題回答

- Q1. 為什麼不將 Chess 記錄到 Player 的屬性當中?
- A. 這樣會導致後面的移動吃棋邏輯更為複雜，使用者輸入上可能需要更多的轉換，例如比較的過程，就還得從 Player 各自的 Chess Array 拉出是哪個棋子，在進一步做比較，但如果將棋子記錄在棋盤上，Player 就只需輸入座標，從一張二維陣列中取出對應的棋子進行比較即可。

六、心得感想

在設計如何隨機分配棋子的部分，一開始想要將棋子記錄在每個玩家當中，但發現這樣會讓棋子移動、棋子比較上實作更為複雜，所以選擇將棋子記錄在棋盤上，以二維方式儲存，這樣就只需要接收兩個座標的參數，就可以對該位置的兩個棋子做比較，簡化邏輯上的實作，而這也讓我了解定義類別間關係的重要性會影響後續的開發上的可讀性與擴展性。於座標的表示，另外定義了 Pair 類別來存取座標的 x、y，這樣就不需要讀取多個變數，來得知要如何移動，而且也在 Pair 當中對 A、B 字符轉換為對應的索引值，這樣就不用將轉換的過程分散在整體程式，方便後續 Debug 與維護。最後我採用例外去處理玩家錯誤的移動方式，並根據情況告訴玩家違反了何項遊玩規則，優化玩家在 UX 上的反饋。總而言之，這次的實作幫助我複習大二時 OOP 的觀念與 Abstruct、Class、Object、泛型上的使用，於之後的開發上讓我可以更有效地根據 Design Pattern 設計出更理想的系統。