Software Design 113 Fall - Final Project
Relay Race System

NM6121030 余振揚

https://github.com/LittleFish-Coder/relay-race-system

Outline

- 引用來源 (Resource)
- 專案背景 (Background)
- 問題分析 (Code Smell)
- 重構設計 (Refactor)
- 具體實作 (Implementation)
- 重構效益分析 (Refactor Benefit)
- 測試框架 (Testing)
- 測試效益分析 (Testing Benefit)
- 總結心得 (Feedback)

引用來源

本專案來源於作者: 電機113丙班 - 顏鈺蓁(E24096857)

- Java程式設計 HW3
- 該專案整體document可於該連結參考: https://github.com/LittleFish-Coder/relay-race-system/tree/master/reference
- 重構專案已取得作者同意

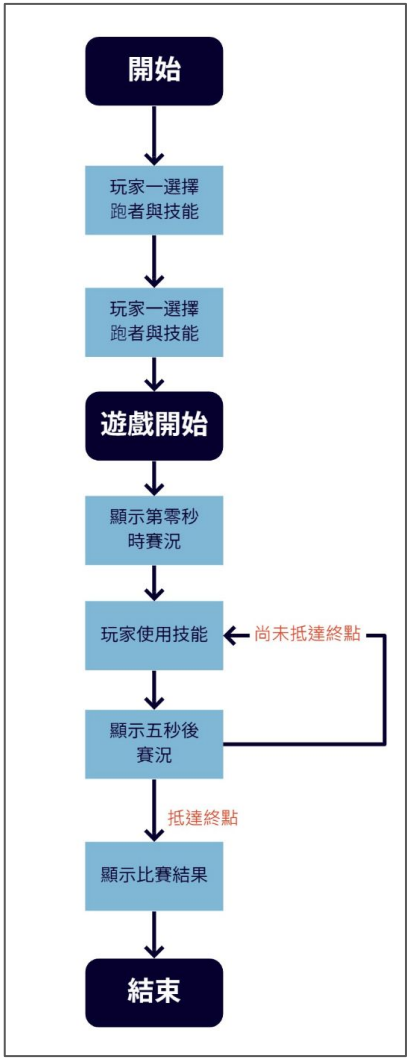
專案背景與目標

概述:

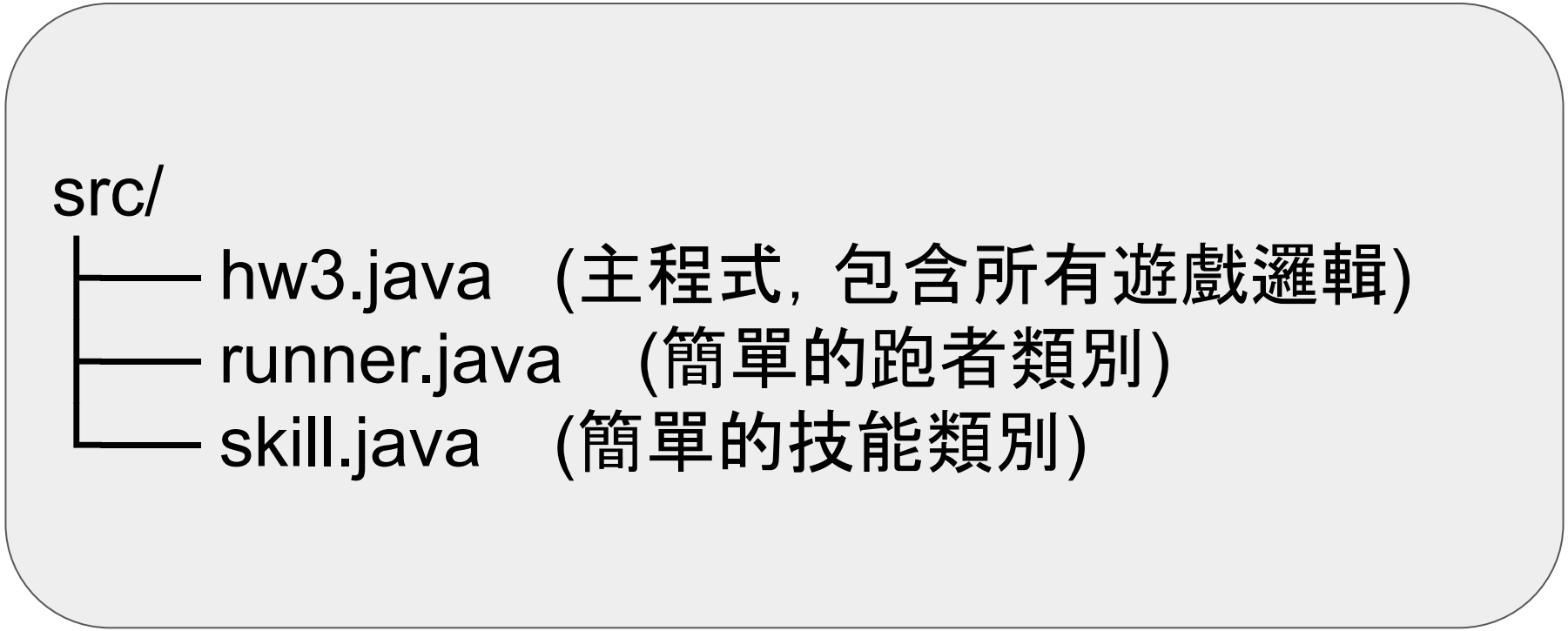
本專案為超能力接力賽跑模擬系統，設計一個回合制的遊戲環境，讓兩位玩家進行虛擬的超能力接力賽跑對決。每位玩家需要策略性地配置五名跑者並運用特殊技能來爭取勝利。

核心功能特色:

- 玩家可選擇兩種類型跑者:
  力量型(每秒10公尺)與敏捷型(每秒20公尺)
- 技能系統包含:
  1. 冰凍技能(暫停對手移動)
  2. 力場技能(降低敏捷型跑者速度)
- 採用回合制設計，每5秒為一回合
- 完整的比賽狀態顯示與即時戰況回饋



問題分析


src/
hw3.java (主程式，包含所有遊戲邏輯)
runner.java (簡單的跑者類別)
skill.java (簡單的技能類別)

Large Class / Long Method

```
// hw3.java
public static void main(String args[]) {
    // 超過500行的main方法, 包含:
    // - 遊戲初始化
    // - 玩家輸入處理
    // - 技能處理
    // - 移動計算
    // - 結果顯示
}
```

Duplicate Code

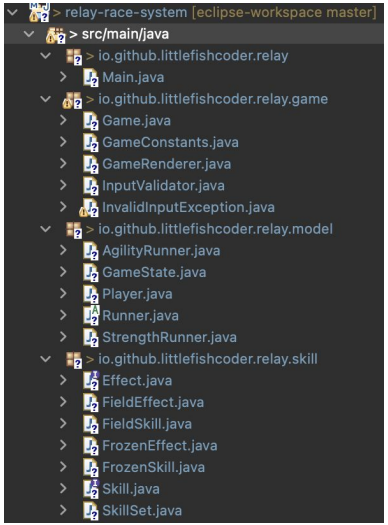
```
// 玩家狀態更新重複
if (skill2.skillInf1 == 1 && skill2.skillInf2 == 0) {
    if (P1.equals("str")) {
        speed1 = P1runner.setSpeed("str");
        System.out.println("受冰凍技能影響");
        t1 = t1 - 5;
    }
}

// 幾乎相同的程式碼用於玩家二
if (skill1.skillInf1 == 1 && skill1.skillInf2 == 0) {
    if (P2.equals("str")) {
        speed2 = P2runner.setSpeed("str");
        System.out.println("受冰凍技能影響");
        t2 = t2 - 5;
    }
}
```

Primitive Obsession (基本型態濫用)

```
String[] runnerP1 = new String[5];  // 應該使用專門的Runner類別
double distance1 = 100;             // 應該封裝在Runner類別中
double speed1 = 0;                  // 應該封裝在Runner類別中
```

重構設計 (MVC)

```
src/
├── hw3.java   (主程式，包含所有遊戲邏輯)
├── runner.java          (簡單的跑者類別)
└── skill.java   (簡單的技能類別)
```




```
io.github.littlefishcoder.relay/
├── game/          // Controller層
│   ├── Game.java            // 主要控制邏輯
│   ├── GameConstants.java   // 常數定義
│   ├── GameRenderer.java    // 遊戲顯示介面
│   └── InputValidator.java  // 輸入驗證
│
├── model/         // Model層
│   ├── Runner.java          // 跑者抽象類別
│   ├── StrengthRunner.java  // 力量型跑者
│   ├── AgilityRunner.java   // 敏捷型跑者
│   ├── Player.java          // 玩家模型
│   └── GameState.java       // 遊戲狀態
│
└── skill/         // 技能系統模組
    ├── Skill.java           // 技能介面
    ├── Effect.java          // 效果介面
    └── ...                  // 具體實現類別
```

具體實作 (Strategy Pattern)

```
// 抽象跑者類別定義移動策略
public abstract class Runner {
    public abstract double move(double time);
}

// 具體實現不同移動策略
public class StrengthRunner extends Runner {
    @Override
    public double move(double time) {
        // 力量型跑者的移動邏輯
    }
}
```

具體實作 (Model)

```
// 遊戲狀態管理
public class GameState {
    private Player player1;
    private Player player2;
    private double gameTime;

    public boolean isGameFinished() {
        // 遊戲結束條件判斷
    }
}

// 玩家狀態管理
public class Player {
    private Runner[] runners;
    private int currentRunnerIndex;
    private int frozenSkills;
    private int fieldSkills;
}
```

具體實作 (異常處理優化)

```
public class InputValidator {
    public Runner[] validateAndCreateRunners(String input)
        throws InvalidInputException {
        if (input.length() != 19) {
            throw new InvalidInputException("指令長度應為19");
        }
        // 更多驗證邏輯..
    }
}
```

重構效益分析

1. 模組化設計

- 清晰的職責分離
- 降低程式碼耦合度
- 提高程式碼重用性

2. 檔案結構

- 按照功能分類程式碼
- 統一的命名規範
- 清晰的檔案結構

測試框架

```
test/
├── game/
│   ├── GameStateTest.java     // 遊戲狀態測試
│   └── GameTest.java          // 遊戲邏輯測試
├── model/
│   ├── PlayerTest.java        // 玩家邏輯測試
│   ├── StrengthRunnerTest.java // 力量型跑者測試
│   └── AgilityRunnerTest.java  // 敏捷型跑者測試
└── skill/
    ├── SkillTest.java         // 技能效果測試
    └── SkillSetTest.java      // 技能組合測試
```

測試Model

重點驗證：

- 跑者切換機制
- 跑者類型判定
- 玩家狀態管理

```java
    @Test
    void testCurrentAndNextRunner() {
        Runner[] runners = new Runner[5];
        runners[0] = new StrengthRunner();
        runners[1] = new AgilityRunner();
        runners[2] = new StrengthRunner();
        runners[3] = new AgilityRunner();
        runners[4] = new StrengthRunner();

        player.setRunners(runners);

        assertTrue(player.getCurrentRunner() instanceof StrengthRunner,
            "第一位跑者應該是力量型");
        assertEquals(1, player.getCurrentRunnerIndex(),
            "當前跑者索引應為1");

        player.nextRunner();
        assertTrue(player.getCurrentRunner() instanceof AgilityRunner,
            "第二位跑者應該是敏捷型");
        assertEquals(2, player.getCurrentRunnerIndex(),
            "當前跑者索引應為2");
    }
```

測試Game

重點驗證：

- 遊戲時間更新
- 距離計算準確性
- 狀態轉換邏輯

```java
@Test
void testUpdateGameState() {
    // 設置初始狀態
    Player p1 = gameState.getPlayer1();
    Player p2 = gameState.getPlayer2();

    // 設置跑者
    Runner[] runners = new Runner[5];
    for (int i = 0; i < 5; i++) {
        runners[i] = new StrengthRunner();
    }
    p1.setRunners(runners);
    p2.setRunners(runners.clone());

    // 執行更新
    game.updateGameState();

    // 驗證時間增加
    assertEquals(5.0, gameState.getGameTime());

    // 驗證跑者移動
    double expectedDistance = 100 - (10 * 5); // 力量型跑者速度 * 時間
    assertEquals(expectedDistance, p1.getCurrentRunner().getDistance(), 0.01);
    assertEquals(expectedDistance, p2.getCurrentRunner().getDistance(), 0.01);
}
```

測試Skill

重點驗證：

- 技能效果正確性
- 不同類型跑者反應
- 技能疊加效果

```java
@Test
void testFrozenSkillOnAgilityRunner() {
    Skill frozenSkill = new FrozenSkill();
    Runner runner = new AgilityRunner();

    // 記錄原始移動距離
    double originalDistance = runner.getDistance();
    double time = 5.0;

    // 使用冰凍技能
    frozenSkill.apply(runner);

    // 移動並檢查距離
    double movedDistance = runner.move(time);
    assertEquals(0.0, movedDistance, "被凍結的跑者不應該移動");
    assertEquals(originalDistance, runner.getDistance(), "被凍結的跑者距離不應改變");
}
```

測試效益分析

1. 品質保證

- 自動化測試確保功能正確
- 及早發現潛在問題
- 防止重構引入新的錯誤

2. 開發效率

- 快速驗證修改
- 減少手動測試時間
- 提升除錯效率

3. 設計改進

- 促進模組化設計
- 提高程式碼可測試性
- 降低組件耦合度

總結心得

通過這次專案，了解到良好的程式設計不僅是實現功能，更重要的是建立一個可持續發展、易於維護的系統架構。此堂課也讓我學習到:

技術成長

- 學習識別Code Smells和解決方案
- 掌握設計模式的實務應用
- 理解測試驅動開發的重要性

設計思維

- 從單體架構到模組化設計
- 重視程式碼可維護性
- 培養系統架構的宏觀思維

Thanks

https://github.com/LittleFish-Coder/relay-race-system