一、 概覽

首先，需介紹這次專題與目標。這次的專題是工學院的跨領域專題，我們的目標是要做出一款遊戲，並且加上我們自製的搖桿，希望可以打破第四道牆，讓搖桿除了操控與震動外還可以有更多的回饋。

整個遊戲檔案被我分成以下的樣子：

```
|------game
    |------core
        |------__init__.py
        |------map
        |------charater
        |-----scene
        |------utilities
        |------updater
            |------__init__.py
```

每個資料夾各為一系統，例：戰鬥、物理與碰撞，將其分的這麼細緻則是希望可以提高程式重複使用率。每個系統皆為許多子系統組合而成，如 updater 是指遊戲更新處理系統，但一個遊戲更新包含許多不同方面，包含物理、狀態與碰撞。我們將這幾個子系統分開，就可依據不同要求包裝出不同系統，像是戰鬥系統包含碰撞檢測、物理引擎即狀態更新，如果將這幾個子系統分開撰寫，那之後想生出新系統就可以快速地用這邊寫好的東西快速寫出來。

1.1 map

地圖系統負責一件簡單的事，地圖場景地形處理。在我的設想中，地形會被這裡所處理，並渲染至畫面。這邊也會負責定義所有地圖會用到的物件，像土、水與草地等地圖組成元素，這些都會在這一邊定義。

另外，如果未來地圖變得過大，這邊也會負責 Chunk 系統的功能，讓一次刷新的壓力不要那麼大。

1.2 charater

顧名思義，這裡負責定義所有角色。每個角色都會繼承一個父物件，並提供 update method 來被遊戲控制流程呼叫，每個角色都會有自己的一套更新邏輯，並儲存自己的狀態。

至於 render 則是交由其他事物去 render。

1.3 scene

這是負責定義所有場景，所有場景皆有 update 與 render method 可供呼叫。實際上的顯示與遊戲介面上的按鈕與控制邏輯每個場景都不一樣。另外，場景還需提供 change method，只要呼叫這個 method 就可以把當修改前場景。

1.4 utilities

這是負責處理支援功能的，例如渲染位置，我們習慣會以圖片的中心點作為基準點，但 pygame 是以圖片的左上角為基準點，這時我們就可以寫一個 function 去處理這個轉換，而不需用人腦心算。

1.5 updater

updater 是專門處理遊戲中的各種更新的，例如物理引擎、戰鬥引擎，碰撞偵測與狀態更新。這些全部都被放在這個檔案夾理面，並依照需求來裝配出所需的引擎可供使用。