109-2 基礎物件導向程式設計

期末專案報告

小朋友下樓梯

刑浩霆 b06703084 蔡昭信 b07902017
簡宏曄 b07902023 陳玉恆 b07902026

supervised by
Prof. HT Lin

July 1, 2021

1 專案構想

我們將在本章節敍述我們專案的成果。

1.1 靈感來源

本次專題題目的靈感來自《 NS-Shaft 》。這是一個在 1990 年由 NAGI-P SOFT 公司所開發的遊戲。《 NS-Shaft 》的遊戲畫面如 Figure 1。遊戲的目標是藉由踩在下方的平台，盡可能地往越下層走。這個由下在台灣被稱作「 小朋友下樓梯 」，是我們兒時回憶中的 flash 小遊戲之一。

Figure 1: 《 NS-Shaft 》的遊戲畫面

1.2 與原版的差異

在原版的遊戲中，可以讓主角站立的平台分爲五種：普通地面、陷阱地板、彈簧板、輸送帶和釘板。其中翻面的地板在被踩踏之後會翻過來，讓玩家掉下去，彈簧板會讓玩家向上彈，輸送帶會將玩家帶向某個方向，釘板會讓踩到的玩家扣血。

在我們的專案中，視覺上與原版主要的差異在於我們沒有完整的美術圖素材。例如彈簧版在觸法時會由數張圖片連續播放以產生彈簧長度連續改變的視覺效果，但我們因爲只有一張圖片所以無法呈現。另外，我們也因此把陷阱地板改爲踩踏後一段時間會消失。

1.3 遊戲玩法

當只有一個玩家時，使用鍵盤方向的左和右來操控主角，有第二個玩家時則是使用 A 和 D。主角會一直往下掉，下面有各式各樣的平台可以供主角站立。請以地底最深處爲目標，避開陷阱和釘板，踩著平台努力往下。

2 軟體架構

我們將在這個章節解釋我們遊戲的軟體架構。我們主要是基於 W13 助教課 source code並做修改。以下列出幾點重點:

1. 與助教的 knight 遊戲有的 x-y 軸的世界，我們世界是 x-z 軸方向。玩家透過鍵盤可以控制 child 左右方向的移動，並隨時受到向下的加速度影響。
2. 因爲在此遊戲中只有 child 會與其他物件 (sprite) 碰撞，我們在 sprite 裡面加上了 collision handle 的 abstract method，並讓不同牆壁，階梯等實作與 child 發生碰撞 (location 重疊) 時的處理。
3. 每個 sprite 裡面有 update 的 method，代表在 Game Loop 中每次回圈所需要做的狀態更新。例如階梯會在此時後往上升，而 child 會根據調整速度並往下掉。

2.1 各物件的介紹

2.1.1 controller

裡面有 Game.java 和 GameLoop.java，用來掌控整體遊戲進行。架構圖如 Figure 3。大圖在此

2.1.2 model

裡面有遊戲中的所有實體物件，包括 Sprite、SpiteShape、HealthPointSprite、Direction、CollisionHandler 和 World。架構圖如 Figure 4。大圖在此

1. Sprite: 所有 world 裡面實體物件 (畫面上看到的東西) 的 super class。
2. SpiteShape: 這個物件規範了每個實體物件的大小和形狀。
3. HealthPointSprite: 從 Sprite 繼承而來，特別用來處理擁有血量的實體，像是玩家。
4. Direction: 處理移動方向的 class，像是玩家的移動，或是 https://www.overleaf.com/project/60d9d7be30371315a949d0db 傳送帶的效果。
5. CollisionHandler: 用來代表有碰撞發生的 interface，會被 child、border、stair 等實體所 implement。
6. World: 代表著整個版面的 class，所有的平台和玩家都存放在這裡，同時也負責所有實體物件的刪除和新增。

2.1.3 border

裡面存有地圖邊界的 class，分別是 Border、Ceiling、Floor 和 Wall。架構圖如 Figure 5。大圖在此

1. Border: 其他三個 class 的 super class，擁有邊界的基本特性。

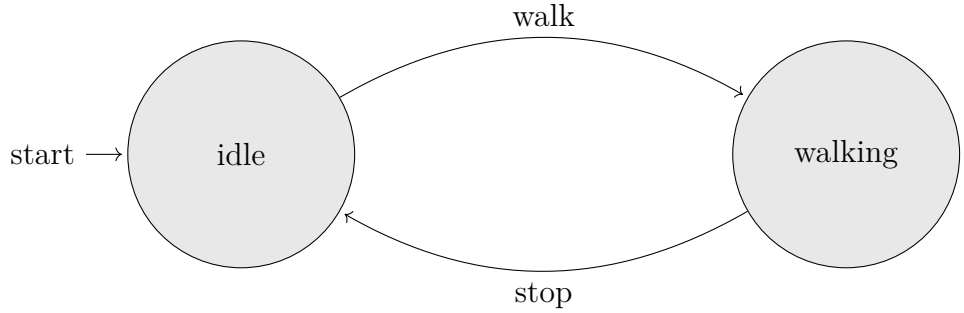

Figure 2: Child 的 fsm

2. Ceiling: 負責上方的邊界，有尖刺的圖層，同時玩家碰到時會觸發特別的 collisionHandle，使玩家扣血。

3. Floor: 負責下方的邊界，實際位置位於可視範圍再下方一點，使得玩家碰到時剛好掉出地圖外，玩家碰到時會直接死亡。

4. Wall: 負責左右的邊界，特性是無法穿越，使玩家可以留在地圖中。

2.1.4 child

裡面有所有和玩家這個實體有關的物件。其中包括 Child、Idle、Walking、ChildImageRenderer 和 HealthPointBar。架構圖如 Figure 6。大圖在此

1. Child: 從 HealthPointSprite 繼承而來，是一個具有血量的實體，動作有被傷害、移動和靜止三個狀態。

2. Idle: 代表 Child 的靜止狀態。

3. Walking: 代表 Child 的移動狀態，玩家的座標會改變，同時　移動時動畫會改變。

4. ChildImageRenderer: 負責 Child 的圖層，在不同狀態下會有所改變。

5. HealthPointBar: 代表 Child 頭上的血條，會隨 Child 血量改變而跟著改變。

2.1.5 fsm

裡面有 State、StateMachine、WaitingPerFrame、Sequence、ImageState、FiniteStateMachine、CyclicSequence。用來除理當遊戲進行時不同狀態間的轉換。

2.1.6 stairs

裡面有所有和平台這個實體有關的物件。其中包括 Stair、NormalStair、Nails、Fake、Conveyor、Trampoline 和 StairGenerator。架構圖如 Figure 7。大圖在此

1. Stair: 是 NormalStair、Nails、Fake、Conveyor 和 Trampoline 的 super class，擁有平台的基本特性。

2. NormalStair: 代表普通的平台，踩在上面不會有特別的事發生。
3. Nails: 代表釘板，玩家踩在上面時會觸發特別的 collisionHandle，使玩家扣血。
4. Fake: 代表陷阱平台，被踩到後會消失。
5. Conveyor: 代表輸送帶，玩家踩在上面時會被帶往指定的方向。
6. Trampoline: 代表彈簧板，玩家踩在上面時會向上彈。
7. StairGenerator: 會在遊戲一開始生成出固定的平台，隨著時間進行，隨機生成新的平台。

2.1.7 views.GameView

views.GameView 負責處理 GUI 介面，包含讀取使用者的滑鼠、鍵盤輸入、與使用者互動，並管理整個遊戲的流程，例如：顯示玩家分數、檢查遊戲是否結束、開起新遊戲等。架構圖如 Figure 8。大圖在此

3 設計分析

在這裡我們要以程式的架構設計來進行分析。

3.1 設計優點

3.1.1 Stair 的設計符合 open-closed principal

在設計的過程中，我們透過繼承 Stair 這個 abstract class 來實作每一種 stair，並透過 StairGenerator 管理每種 stair 的生成。透過這樣的方法，如果這時我們想要增加/移除其中一種 stair，我們只需要在其他物件初始化的地方進行修改。以這邊的例子，我們只需要修改 StairGenerator 的初始化就可以達到任意增加/移除不同種的 stair。

3.1.2 HealthPointSprite 的設計符合 open-closed principal

在設計的過程中，我們先以 HealthPointSprite 繼承 Sprite，再讓 Child 去繼承去繼承 HealthPointSprite。透過這樣的方法，假設我們要增加其他具有血量的實體，像是路上的敵人之類的，我們只需要去繼承 HealthPointSprite，而不用重新寫出一個 class。或是如果我們需要增加其他玩家角色，我們也可以透過繼承 HealthPointSprite 的方式，達到目標。

3.2 設計缺點

3.2.1 程式架構過於複雜

在架構中，我們雖然利用很多 abstract class 和繼承等方式來使我們的設計符合 OCP 原則，但是這樣的設計導致架構變得過於複雜，舉例來說，我們先以 HealthPointSprite 繼承 Sprite，再以 Child 去繼承 HealthPointSprite。HealthPointSprite 這個 class 只有在這裡被使用到，在尚未有其他具有血量的實體前，這個 class 其實是有點累贅的。

4 使用 **package**

- WaterBall 助教的小小騎士
- java.awt
- java.util
- java.io
- javax.imageio
- javax.sound

5 組員分工

刑浩霆 b06703084: 程式碼撰寫
蔡昭信 b07902017: 報告撰寫
簡宏曄 b07902023: 程式碼撰寫、報告撰寫
陳玉恆 b07902026: 程式碼撰寫

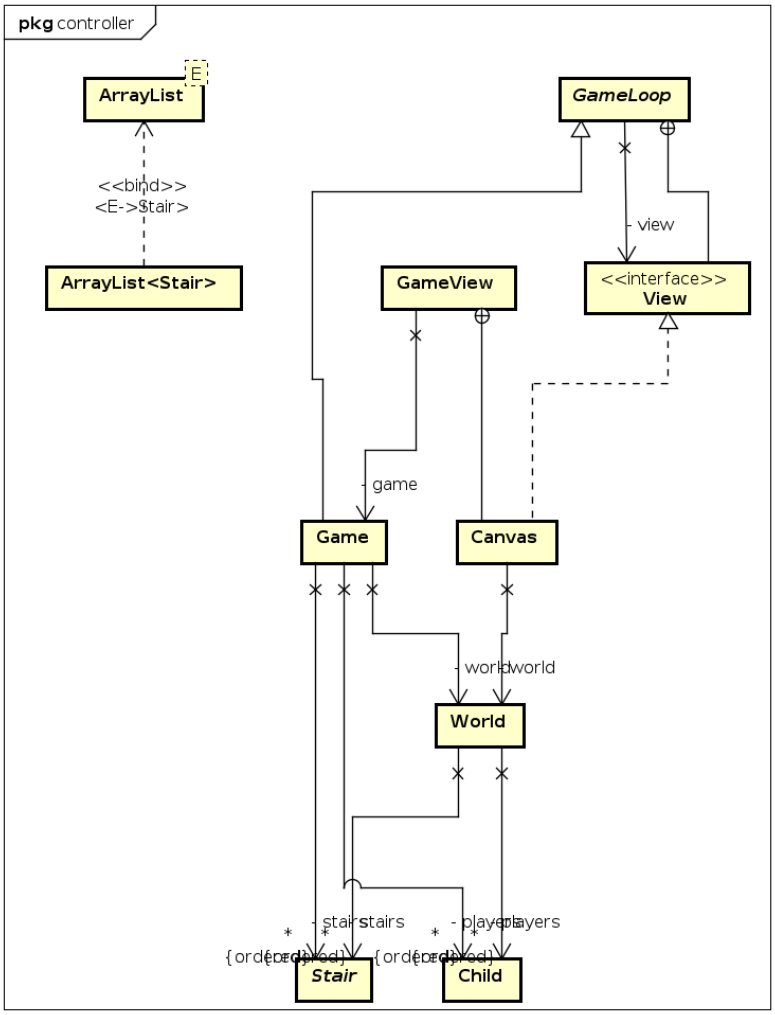

Figure 3: controller 的組織結構圖

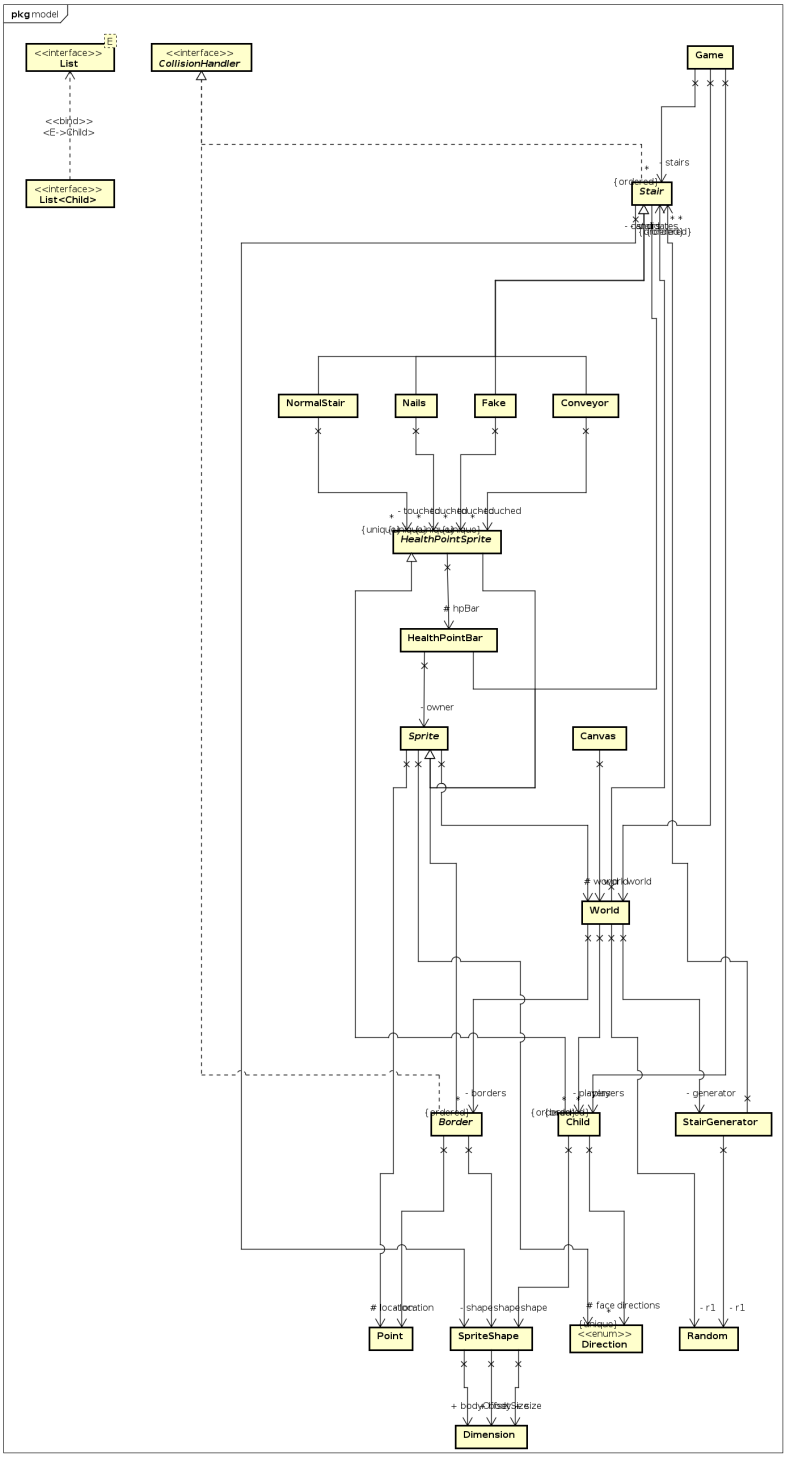

Figure 4: model 的組織結構圖

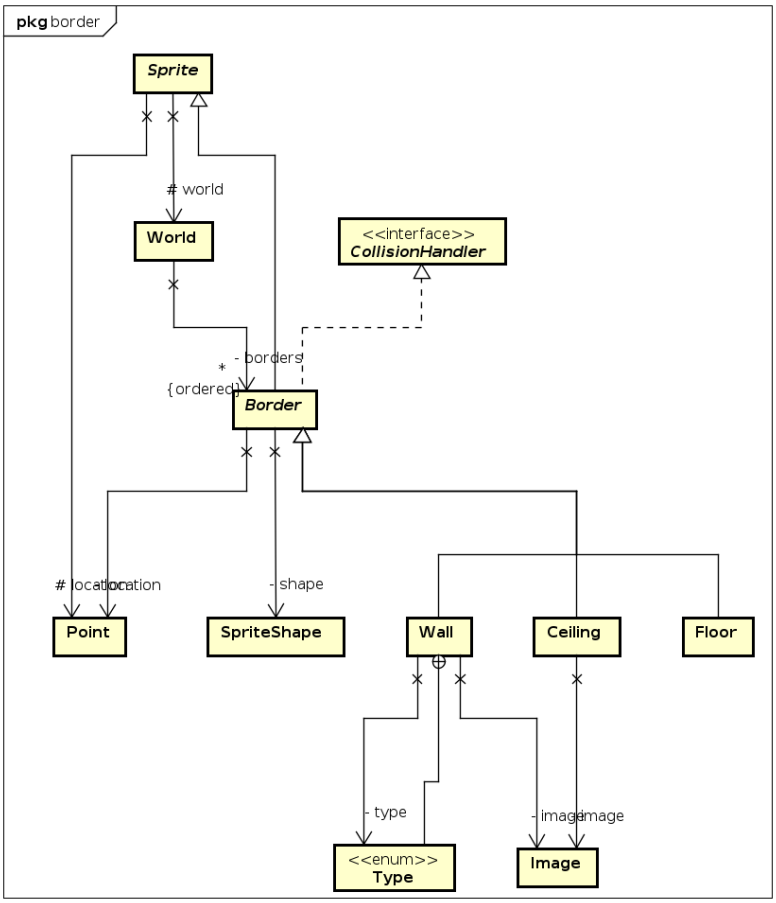

Figure 5: border 的組織結構圖

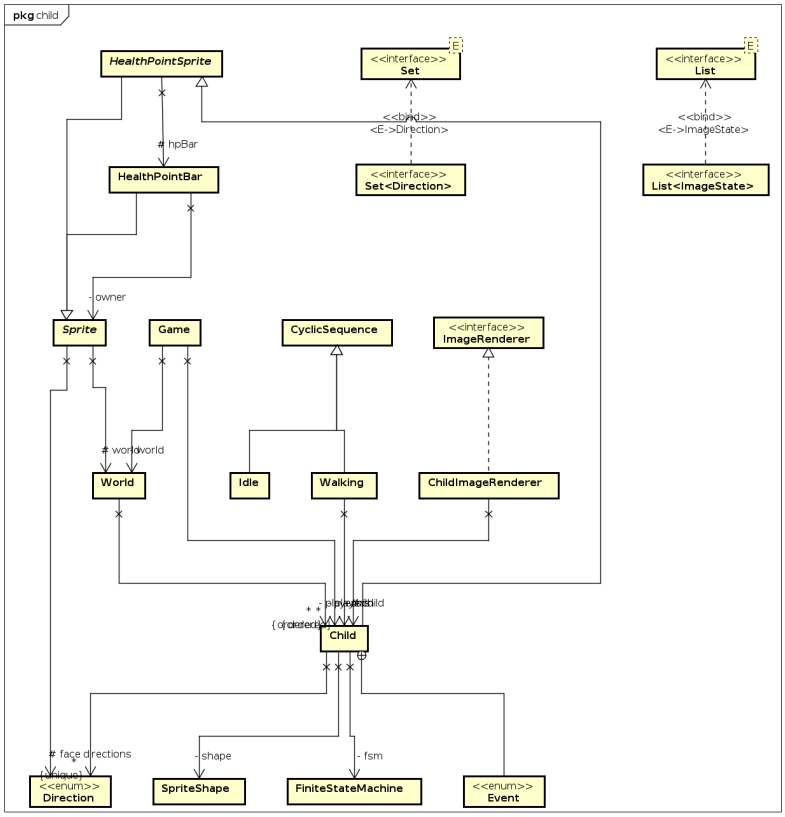

Figure 6: child 的組織結構圖

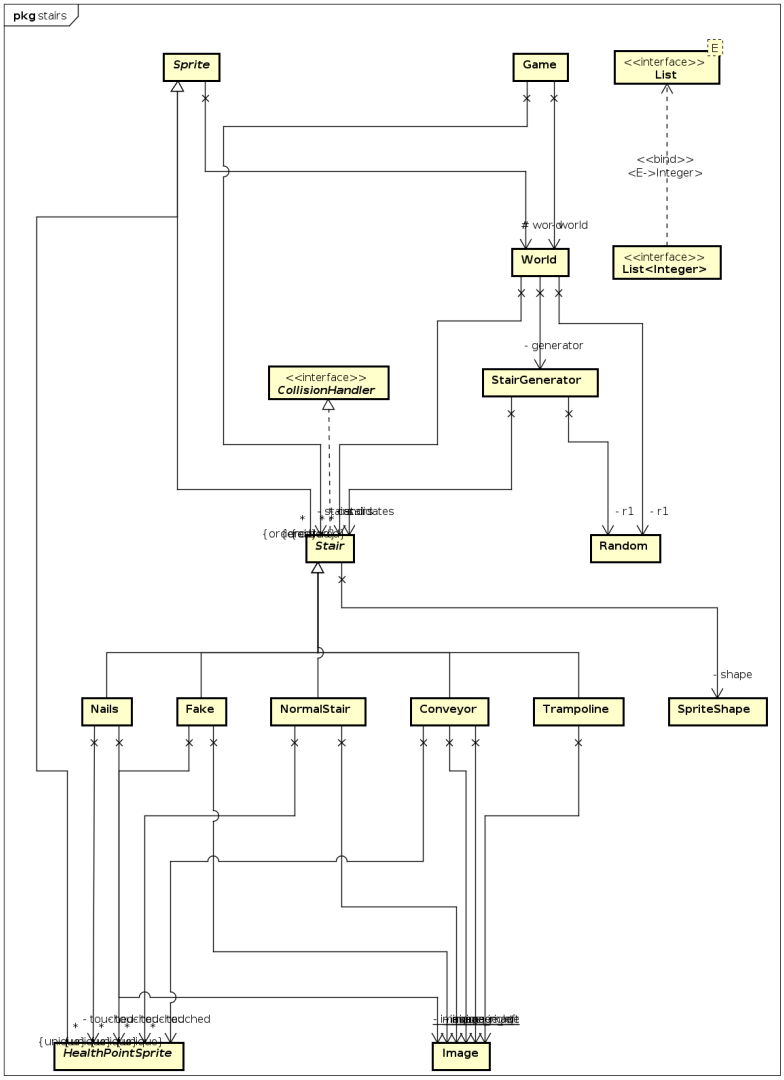

Figure 7: stairs 的組織結構圖

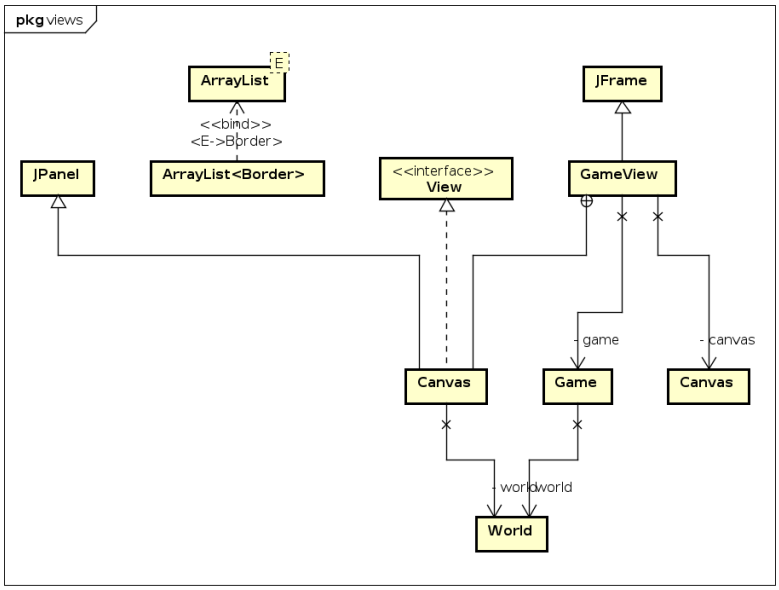

Figure 8: views 的組織結構圖