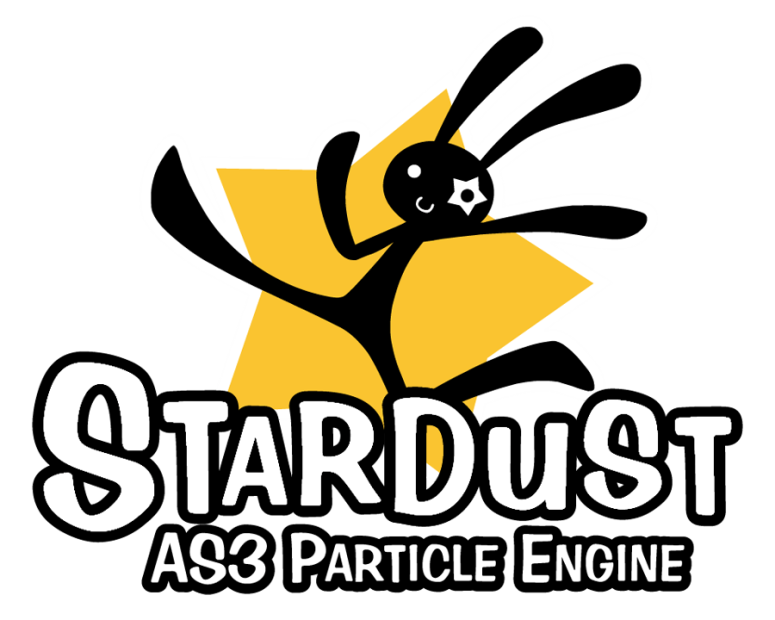
**軟體設計模式期末專題**

**星塵粒子引擎**



組別： 16

系級： 電機四

組員： 周明倫(B95901008)

林昭瑋(B95901038)

指導教授： 陳俊良

**目次**

[Stardust Particle Engine的由來 3](#_Toc250929237)

[Stardust目前的發展狀況 4](#_Toc250929238)

[Google Code Project 4](#_Toc250929239)

[PDF Manual & Video Tutorials 4](#_Toc250929240)

[Feedbacks 4](#_Toc250929241)

[Stardust Particle Engine的特色 5](#_Toc250929242)

[擴充容易 5](#_Toc250929243)

[支援2D與3D特效 5](#_Toc250929244)

[可設計複雜的粒子行為 5](#_Toc250929245)

[支援XML序列化 5](#_Toc250929246)

[光碟附加檔案說明 6](#_Toc250929247)

[檔案與資料夾結構 6](#_Toc250929248)

[檔案類型說明 7](#_Toc250929249)

[Stardust類別繼承關係 8](#_Toc250929250)

[Stardust元素 9](#_Toc250929251)

[StardustElement類別 9](#_Toc250929252)

[Emitter類別 9](#_Toc250929253)

[Initializer類別 10](#_Toc250929254)

[Action類別 10](#_Toc250929255)

[Clock類別 10](#_Toc250929256)

[Renderer類別 11](#_Toc250929257)

[Particle類別 11](#_Toc250929258)

[Stardust使用流程 12](#_Toc250929259)

[生成Emitter物件 12](#_Toc250929260)

[指派Initializer物件 12](#_Toc250929261)

[指派Action物件 12](#_Toc250929262)

[生成Renderer物件 13](#_Toc250929263)

[持續呼叫主迴圈 13](#_Toc250929264)

[Stardust主迴圈運作流程 14](#_Toc250929265)

[生成新粒子 14](#_Toc250929266)

[更新粒子 14](#_Toc250929267)

[移除死亡粒子 14](#_Toc250929268)

[視覺化呈現 14](#_Toc250929269)

[Stardust與設計模式 15](#_Toc250929270)

[使用自己想要的Initializer與Action 15](#_Toc250929271)

[Command Pattern 15](#_Toc250929272)

[Initializer與Action的執行優先順序變更 16](#_Toc250929273)

[Observer Pattern 16](#_Toc250929274)

[利用Clock來決定粒子生成頻率 16](#_Toc250929275)

[Strategy Pattern 16](#_Toc250929276)

[將多個元素打包 17](#_Toc250929277)

[Composite Pattern 17](#_Toc250929278)

[粒子物件生成 18](#_Toc250929279)

[Factory Method Pattern 18](#_Toc250929280)

[粒子集合ParticleCollection類別 19](#_Toc250929281)

[Iterator Pattern 19](#_Toc250929282)

[Strategy Pattern 20](#_Toc250929283)

[Singleton Pattern 20](#_Toc250929284)

[從Particle Pool中提取粒子物件 21](#_Toc250929285)

[Singleton Pattern 21](#_Toc250929286)

[Renderer的視覺呈現 22](#_Toc250929287)

[Observer Pattern 22](#_Toc250929288)

[MVC Pattern 22](#_Toc250929289)

[XML序列化 24](#_Toc250929290)

[Visitor Pattern 24](#_Toc250929291)

[Stardust效能檢討 25](#_Toc250929292)

[課程心得 27](#_Toc250929293)

[周明倫 27](#_Toc250929294)

[林昭瑋 28](#_Toc250929295)

[相關連結 30](#_Toc250929296)

[參考資料 30](#_Toc250929297)

# Stardust Particle Engine的由來

　　星塵粒子引擎 (Stardust Particle Engine) 是我 (周明倫) 於2009年暑假自修完Head First Design Patterns與GoF Design Patterns之後，為了自我練習設計模式而開發的實驗性質作品。在這之前我已經有開發另外一套粒子系統Emitter，由於當時沒有很強的設計模式知識與觀念，類別與介面設計不佳，導致維護極度不容易，Stardust也可以算是Emitter的後繼者吧。

　　以我個人的定義來說，粒子就是大量「行為、外觀相似」但「互相存在差異」的個體。粒子特效即是利用粒子製作的視覺特效，在電影、電玩是不可或缺的特效；例如火焰、煙霧、雨滴、雪花等，都算是粒子特效。現在網路上Flash網站與遊戲已非常流行，Flash Player (Flash的虛擬機器)全球的普及率超過95%，是一塊非常值得投資的領域。自從開發Stardust以後，我便一直以把Stardust推廣到全世界為目標，希望大家能夠接受Stardust，並且使用Stardust來開發Flash網站與遊戲的視覺特效。

　　粒子引擎最重要的就是效能，因為要即時處理大量的粒子物件、更新它們的屬性與參數，並且將其視覺化於螢幕上。另外，粒子特效也需要一些行為控制，例如亂流、群體移動、重力模擬等。Stardust將繁瑣的演算法包裝起來，提供一套簡單上手的介面，讓程式設計師不用費心去撰寫粒子特效相關的底層記憶體管理與演算法等，而能夠專注於粒子行為的設計上。

　　這學期上了設計模式的課，讓我們對改善Stardust架構躍躍欲試，於是我們這學期決定好好研究一下Stardust的架構、對其架構作優化、盡量把我們這學期所學到的知識應用在上面。



▲一位日本ActionScript 3.0程式設計師用Stardust製作的蝴蝶粒子特效

　(ActionScript 3.0為Flash的程式語言)

# Stardust目前的發展狀況

## Google Code Project

　　目前Stardust已經發展成一個稍具規模的Project，目前使用Google Code提供的SVN Repository版本控管系統來儲存所有相關檔案。

Stardust首頁：<http://code.google.com/p/stardust-particle-engine>

首頁上提供各版本的原始檔、範例、編譯過的Bytecode下載。

## PDF Manual & Video Tutorials

　　為了幫助大家學習Stardust，首頁上除了提供用ASDoc自動生成的Documentation以外，還提供了PDF格式的英文自修手冊；另外，於YouTube也有提供一系列的Stardust教學影片。這些教學資源於Stardust專案首頁上皆可找到連結。

## Feedbacks

　　Stardust剛推出的時候使用者並沒有很多，但是最近開始有人於Twitter與WonderFL上流傳Stardust的消息，於是Stardust的使用者人數開始快速增加。已經有一些網友於Blog上面撰寫Stardust的介紹與教學，Stardust也被WonderFL收錄為可用API。

WonderFL首頁：<http://wonderfl.net>

* WonderFL為一個日本ActionScript 3.0專業交流網站，提供一個簡單的文字編輯器和線上編譯服務，讓大家可以從沒有Flash編譯器的電腦直接於網頁上撰寫Flash程式、即時觀看編譯後的結果、以及分享和觀賞其他使用者的作品；目前全世界已經有許多知名的ActionScript 3.0程式設計師加入WonderFL的社群。WonderFL收錄許多第三方API，讓程式設計師撰寫程式的時候有更多的API選擇，而Stardust也有被收錄其中。

# Stardust Particle Engine的特色

## 擴充容易

　　Stardust將粒子的初始化與行為委派給Initializer和Action兩個類別，將視覺化處理委派給Renderer類別，並且已經有非常多的內建Initializer、Action、與Renderer類別供選擇。如果使用者想要自己擴充出自己的自訂粒子初始化、行為、或者視覺化方式，可以自行繼承這些類別來辦到。

## 支援2D與3D特效

　　Stardust的粒子模擬系統有兩套model，一套是2D粒子特效，而另外一套是3D的。使用者可以視需求採用2D或3D的model來作粒子模擬，然後用**Renderer**類別將此model視覺化。Stardust提供內建的2D與3D Renderer，也提供支援一些其他的ActionScript 3.0第三方3D引擎的Renderer，例如ZedBox(我自己寫的2.5D引擎)、Papervision3D、與ND3D。

## 可設計複雜的粒子行為

　　Action Trigger是一種特別的Action，同時也是一個Composite Action，它能夠指定其Component Action在「特定條件」狀況下觸發，藉此設計行為更加複雜與多樣化的粒子特效。例如「當一個粒子死亡時，原地生出多個粒子」可以製作煙火特效，以集「當兩個粒子互相碰撞實，蹦出火花粒子」可以製作物體撞擊火花特效。

## 支援XML序列化

　　Stardust最重要的特色之一，就是它支援XML序列化。Stardust是現行的ActionScript 3.0粒子引擎中唯一支援XML序列化功能的。藉由序列化，使用者可以將一個粒子系統的行為設計與參數轉換成一個外部XML檔案，以供Flash應用程式動態載入；這個功能在開發大型Flash應用程式的時候很有用：只要修改並且儲存該外部XML檔案，然後重新執行Flash應用程式即可套用新的粒子特效參數，而不用重新編譯。

# 光碟附加檔案說明

## 檔案與資料夾結構

files ┬ ***Flash Player.exe***

├ src (Stardust原始檔)

│ └ ***Stardust.as3proj***

├ FlashDevelop

│ └ ***FlashDevelop.exe***

├ docs (Stardust documentation)

│ └ ***index.html*** (documentation首頁)

├ manual (Stardust PDF英文說明書)

│ ├ ***Stardust Particle Engine Manual.pdf***

│ └ examples

│ └ 說明書範例

└ demos

├ ***Stardust Demos.as3proj***

├ 2D (2D demos)

│ └ <任意資料夾>

│ ├ FLA檔

│ ├ AS檔

│ └ SWF檔

└ 3D (3D demos)

└ <任意資料夾>

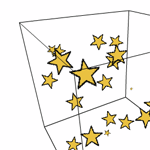
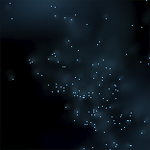
├ FLA檔

├ AS檔

└ SWF檔

* src資料夾中包含完整的Stardust project原始檔，其中的Stardust.as3proj是免費第三方編輯器FlashDevelop的專案檔。
* FlashDevelop資料夾中的FlashDevleop.exe為FlashDevleop的stand-alone版本主程式，可以用來開啟AS3PROJ專案檔。請注意，FlashDevelop需要於支援.NET Framework 2.0的執行環境下才可執行。
* docs資料夾中包含透過Adobe提供的documentation生成器ASDocs所自動產生的API說明文件。
* manual資料夾中有我為Stardust撰寫的PDF英文使用手冊，是於Stardust首頁下載得到的。
* demos資料夾中是一些Stardust的特效展示，包含原始檔與編譯過後的SWF檔案，執行方式請見下頁的檔案說明。

以下為一些Demo的截圖：

## 檔案類型說明

* FlashDevelop為一個使用.NET Framework的免費ActionScript編輯器，其功能完整、使用者眾多，是最熱門的免費ActionScript編輯器。
* AS3PROJ檔為FlashDevleop的ActionScript 3.0 Project專案檔，用FlashDevleop開啟，即可於Project Panel中瀏覽專案。
* Flash Player.exe是Flash Player的stand-alone版本，是可獨立執行的虛擬機器，可以用來開啟SWF檔。
* FLA檔是Flash IDE的原始檔，若沒有Flash IDE無法開啟該檔案並無所謂，因為這些FLA檔只包含視覺元件的外觀設計資訊，真正的程式碼都寫在AS檔裡。
* 一個AS檔相當於一個JAVA檔，其中包含一個class或者interface的定義，與FLA檔同名的AS檔為main class。
* SWF檔相當於Java的CLASS檔，是FLA檔與AS檔編譯之後產生的Bytecode檔，將其拖曳到Flash Player視窗中即可執行。

# Stardust類別繼承關係

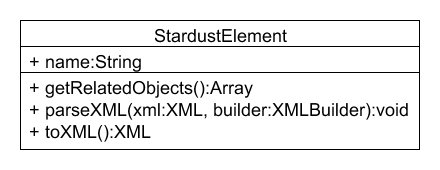
　　為了不要讓畫面充滿了擁擠的UML diagram，我先直接將Stardust的PDF英文說明書中的插圖在此重現，之後再用UML diagram詳細介紹各個類別。

inheritance diagram.png

legends.png

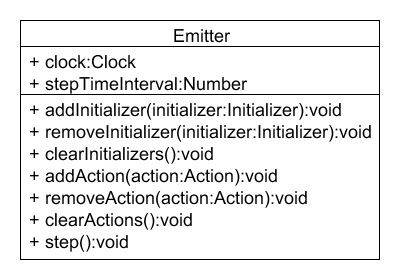
# Stardust元素

## **StardustElement**類別



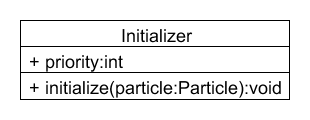
　　除了Value Object (VO)類別以外，Stardust的所有類別都繼承自StardustElement這個類別。這個類別中定義了XML序列化所需的method與property，諸如name、toXML()與parseXML()。

## **Emitter**類別



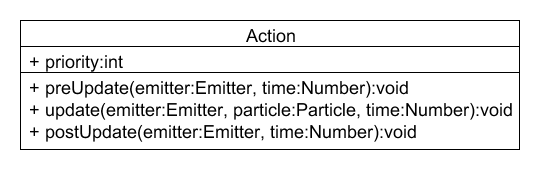
　　Emitter是粒子系統的噴發器，新粒子的生成與初始化、粒子的行為都是由它管理。使用者透過addInitializer()與addAction()兩個method來賦予Emitter想要使用的粒子初始化與粒子行為。step()為Stardust的主迴圈，使用者需持續呼叫這個method以維持粒子模擬的運作。stepTimeInterval的數值是呼叫一次主迴圈所模擬的時間間隔，例如2即代表粒子模擬的速度為兩倍。clock決定每一次呼叫主迴圈，Emitter生成多少新的粒子。

## **Initializer**類別



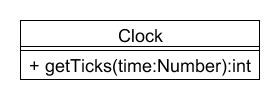
　　此類別是用來初始化新生粒子的，新生成的粒子物件會被當作參數傳入Initializer的initialize() method而被初始化。priority是Initializer的執行優先順序。

## **Action**類別



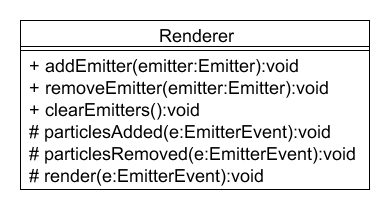
　　此類別是用於每一次呼叫主迴圈時，更新粒子物件的屬性用的，例如Move Action會依照粒子物件的速度來更新位置。preUpdate()會於開始更新粒子時呼叫一次，用以設定好會用到的資源；update()會用於更新所有的粒子，每個粒子都會被當作參數傳入該method一次；postUpdate()則是於更新完粒子屬性之後被呼叫一次，是用來釋放資源用的。priority是Action的執行優先順序。

## **Clock**類別



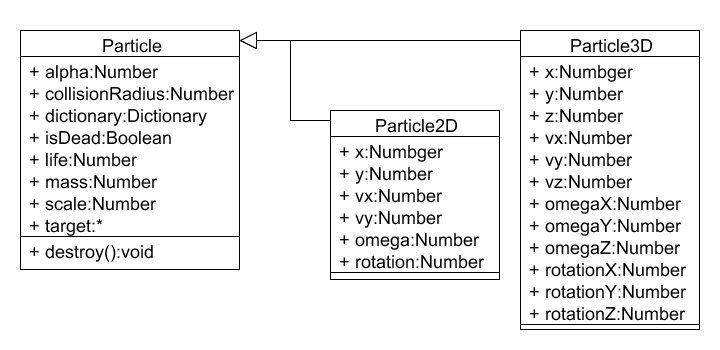
　　Clock類別用於決定Emitter於一次主迴圈中，會生成多少個新粒子物件：的回傳值即代表新粒子的生成個數。

## **Renderer**類別



　　Renderer負責將Emitter的粒子model視覺化，透過addEmitter()可以開始監控Emitter的粒子生成、死亡、與主迴圈的呼叫。particlesAdded()會於新粒子生成時被觸發，particlesRemoved()會於死亡粒子從模擬中移除時被觸發，render()則會於Emitter.step()呼叫完成時觸發，以將主迴圈結束時的model視覺化。

## **Particle**類別



　　此為VO類別。一個Particle物件代表一個粒子，此類別定義了2D與3D粒子共用的屬性，2D與3D粒子特有的屬性則分別定義於Particle2D與Particle3D類別中。

# Stardust使用流程

## 生成Emitter物件

　　Emitter是Stardust的根本，要創造粒子特效就需要生成Emitter物件。生成Emitter物件時可從其Constructor傳入一個Clock物件，以指定生成粒子的頻率。

以下程式碼會生成一個Emitter物件，並且指派一個SteadyClock物件，表示每一次呼叫主迴圈時，創造三個新的粒子。

|  |
| --- |
| var emitter:Emitter= new Emitter2D(new SteadyClock(3)); |

* Emitter2D類別是用來創造2D粒子特效的Emitter子類別，而Emitter3D則用來創造3D粒子特效。

## 指派Initializer物件

　　Initializer用來初始化新生粒子的屬性；使用者可依自己需求，指派想要使用的Initializer物件給Emitter物件。

以下程式會碼指派初始化粒子尺寸的Scale Initializer物件，並且於其Constructor中傳入一個UniformRandom物件，導致新生粒子的尺寸位於20.5的範圍內。

|  |
| --- |
| emitter.addInitializer(  new Scale(new UniformRandom(2, 0.5))); |

## 指派Action物件

　　Action用來於主迴圈中更新粒子的屬性；同Initializer，使用者可以依自己的需求，指派要使用的Action物件給Emitter物件

以下程式碼會指派一個Move Action物件給Emitter物件，導致每一次呼叫主迴圈時，粒子都會依照其速度而更新位置座標。

|  |
| --- |
| emitter.addAction(new Move()); |

## 生成Renderer物件

　　以上的步驟都將是只會修改與更新粒子模擬的Model而已，要將其以視覺呈現在畫面上，需要使用Renderer物件。視覺呈現的方式有很多種，使用者可依自己的需求使用不同的Renderer。

以下程式碼會生成一個DisplayObjectRenderer物件、指定該Renderer視覺化的目標Container、並且指派一個Emitter物件給它，每一次呼叫完主迴圈，Renderer會自動更新畫面。

|  |
| --- |
| var renderer:Renderer  = new DisplayObjectRenderer(container);  renderer.addEmitter(emitter); |

## 持續呼叫主迴圈

　　以上皆為前置作業，要開始模擬粒子特效，需要持續呼叫主迴圈函式Emitter.step()。以下範例程式碼監聽Flash每一次更新Frame的事件，於這個時候呼叫主迴圈。

|  |
| --- |
| addEventListener(Event.ENTER\_FRAME, mainLoop);  function mainLoop(e:Event):void {  emitter.step();  } |

# Stardust主迴圈運作流程

以下行為依照一次呼叫主迴圈的發生順序列出：

## 生成新粒子

　　Emitter會呼叫其被指派的Clock物件的getTicks()函式，該函式的回傳值決定本次主迴圈生成的新粒子數量；另外，新粒子的各項參數將會被Initializer.initialize()函式初始化。初始化完畢後，Emitter將會通知Renderer作新生粒子的相關處理。

## 更新粒子

　　每一個粒子的各項參數將會被Action.update()函式更新。於這些Action的更新流程中，某些Action物件有可能會把粒子標示為「死亡」 (將Particle.isDead屬性設為true)，這些死亡的粒子接下來會被移除。

## 移除死亡粒子

　　根據Particle.isDead屬性，Emitter會將死亡的粒子從模擬中移除，並且通知Renderer做死亡的相關處理。

## 視覺化呈現

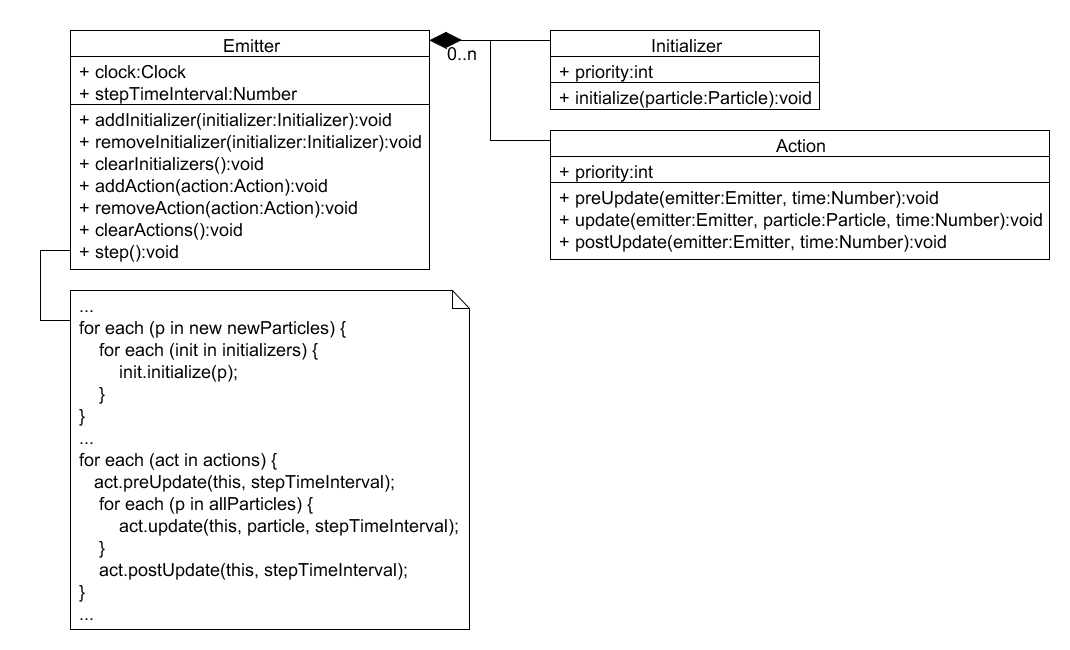
　　主迴圈中修改Model的部分到此為止，接下來Emitter會通知Renderer依照主迴圈結束前的Model來視覺化呈現粒子特效。

# Stardust與設計模式

## 使用自己想要的Initializer與Action

### Command Pattern

　　使用者可依照自己需求使用不同組合的Initializer與Action物件，此為Command Pattern。Emitter只管呼叫Initializer.initialize()來初始化新生粒子，以及呼叫Action.update()來更新粒子屬性；真正的initialize()與update()行為是由Initializer與Action子類別來Override。Initializer.initialize()與Action.update()即相當於Command Pattern中Command Object的execute()函式。

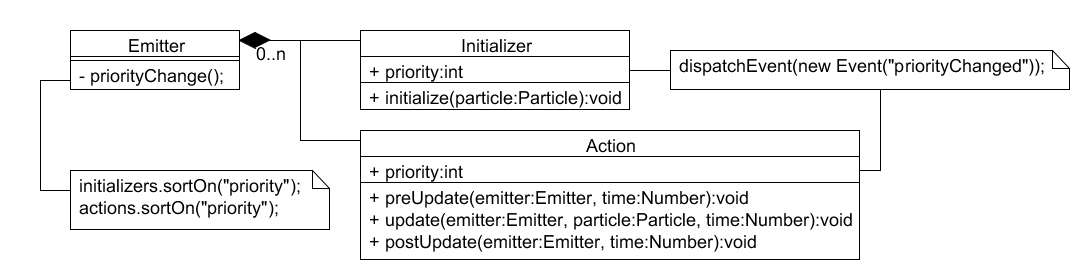


## Initializer與Action的執行優先順序變更

### Observer Pattern

　　Initializer與Action都有定義一個priority屬性，代表這些物件的使用優先順序，Emitter於使用Initializer與Action物件時，將會先使用priority值較大者。當一個Initializer或Action物件的priority屬性變更時，將會透過Observer Pattern通知Emitter，導致Emitter重新依照優先順序來排序物件；Initializer與Action物件為Subject，Emitter的角色則是Observer。

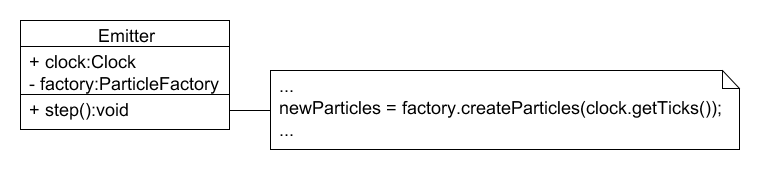
(此功能使用ActionScript 3.0的Event API來實作Observer Pattern)



## 利用Clock來決定粒子生成頻率

### Strategy Pattern

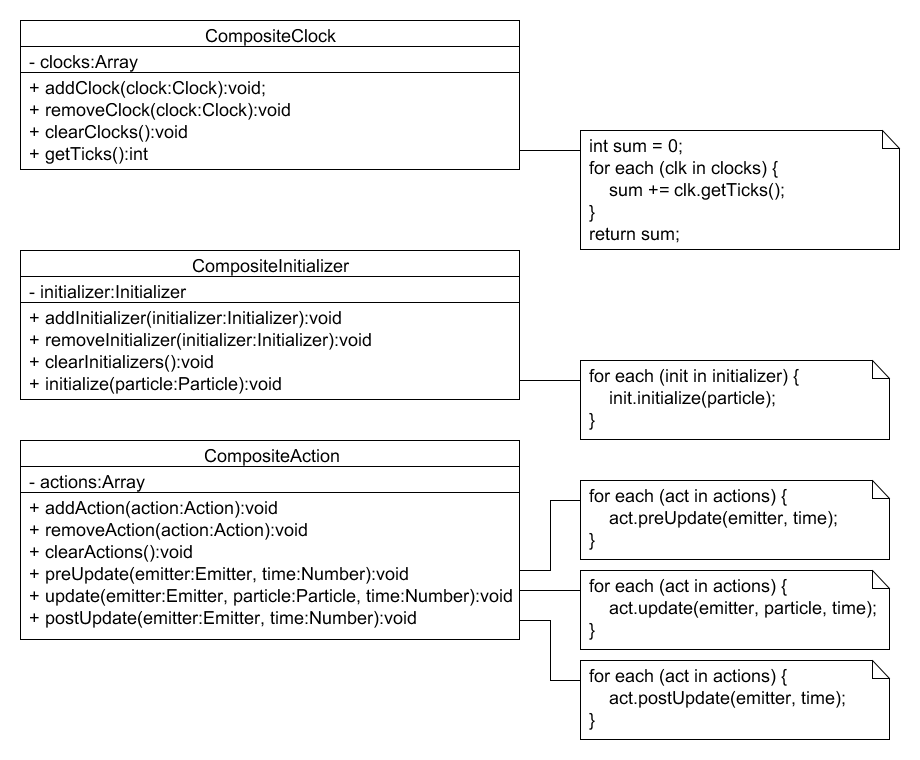
　　指派給Emitter不同的Clock，即可有不同的粒子生成頻率，例如SteadyClock代表穩定生成頻率，RandomClock代表隨機生成頻率。此為Strategy Pattern，Clock物件即為Strategy/Behavior Object。



## 將多個元素打包

### Composite Pattern

　　使用者可以使用CompositeClock、CompositeInitializer、和CompositeAction類別，分別將多個Clock、Initializer、或Action物件打包成集合，以方便管理，此為Composite Pattern。Clock、Initializer和Action為Component，CompositeClock、CompositeInitilizer和CompositeAction則是Composite。

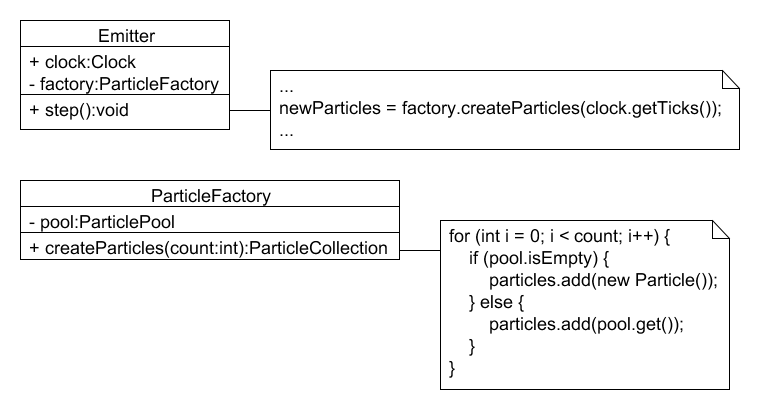


## 粒子物件生成

### Factory Method Pattern

　　為了增進Stardust效能，代表死亡粒子的Particle物件會從Emitter被移除，然後放到一個暫存物件池(Object Pool)；當需要生成新的Particle物件時，若Object Pool還有庫存，則會直接從庫存中取得一個Particle物件，以省去生成新Particle物件的資源消耗。

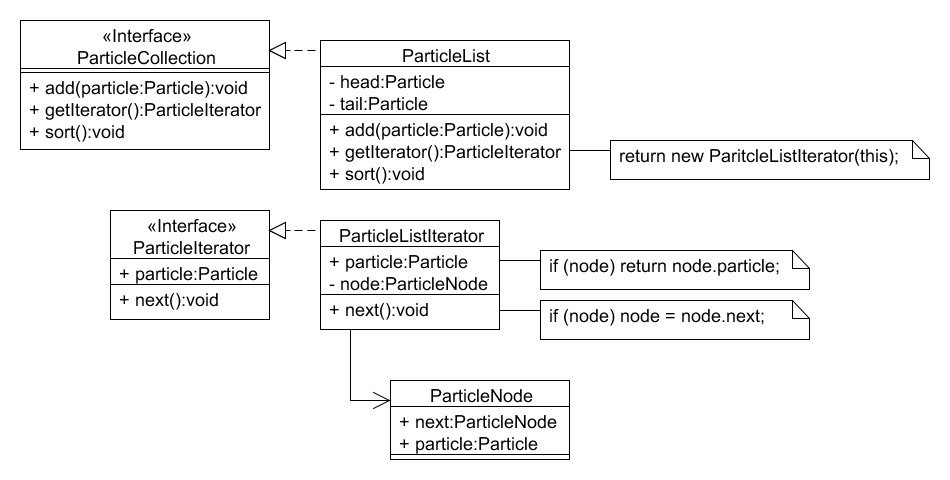
　　每一個Emitter本身都含有一個ParticleFactory物件，該物件負責生成Particle物件交給Emitter加入模擬。若Object Pool尚有Particle物件庫存，則Factory會從庫存中取得Particle物件；反之，則會生成新的Particle物件。Emitter透過ParticleFactory.createParticles()函式取得粒子物件，此為Factory Method Pattern；ParticleFactory類別為Factory，粒子物件則是Product。



## 粒子集合ParticleCollection類別

### Iterator Pattern

　　實作ParticleCollection介面的類別可以用來儲存Particle物件集合，並且使用Iterator Pattern來提供使用者Traverse此集合的途徑。ParticleList為一實作ParticleCollection介面的類別，其背後使用的資料結構是Linked-List。另外尚有一個ParticleArray類別，使用的資料結構為單純的陣列。



以下程式碼會將一個集合中的所有粒子位置重設至原點(0, 0)。

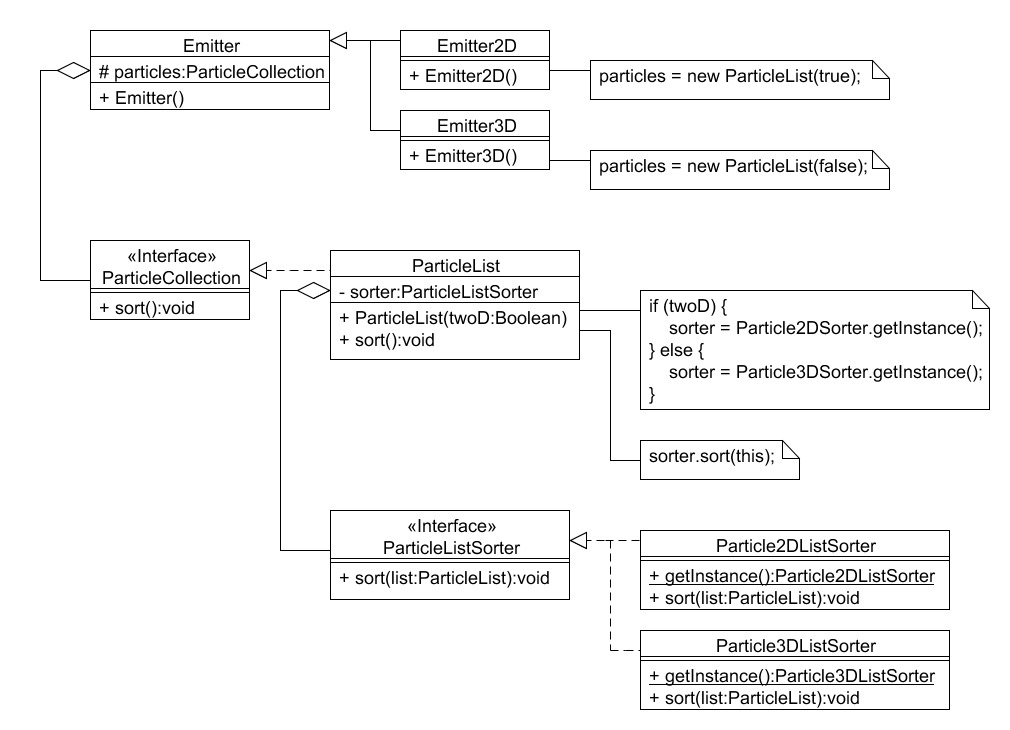
|  |
| --- |
| var iter:ParticleIterator = particles.getIterator();  var p:Particle2D;  while (p = Particle2D(iter.particle)) {  p.x = p.y = 0;  iter.next();  } |

### Strategy Pattern

　　某些Action會需要依照X軸排序完畢的ParticleCollection，以增加粒子模擬效率，例如Collide Action模擬粒子間的撞擊，就需要粒子以X軸排序。Emitter2D與Emitter3D使用不同的粒子類別(Particle2D與Particle3D)，它們使用的ParticleList物件則利用不同的ParticleListSorter物件來作排序，此為Strategy Pattern。ParticleListSorter的角色為Strategy/Behavior Object。

### Singleton Pattern

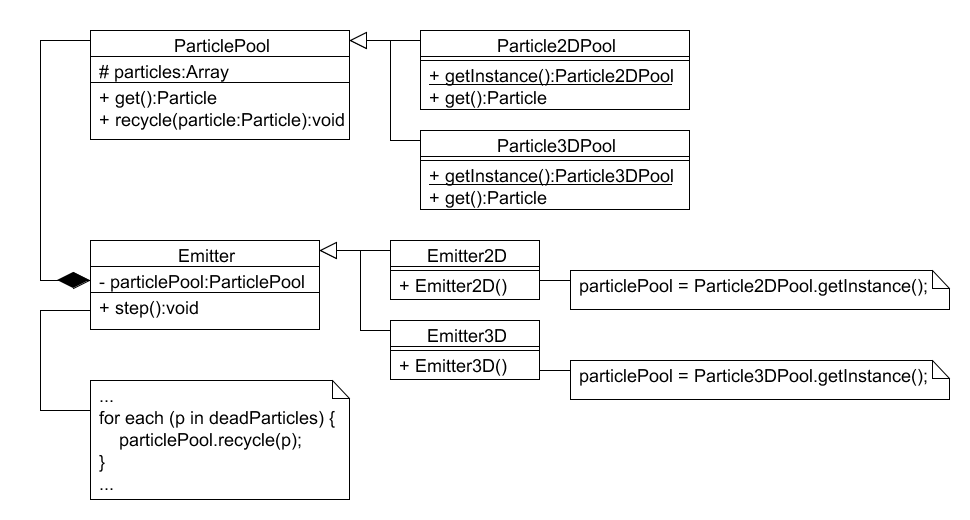
　　Particle2DSorter與Particle3DSorter作的事情都是依照粒子X軸座標排序，沒有物件之間的差異，所以各只要一個物件就夠了，此兩個類別都有getInstance() method可以呼叫，以取得唯一的物件，此為Singleton Pattern。



## 從Particle Pool中提取粒子物件

### Singleton Pattern

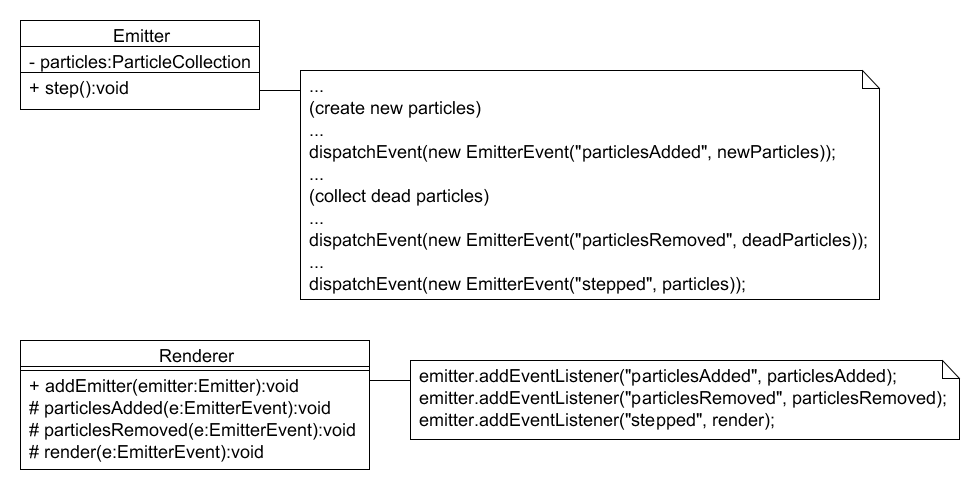
　　粒子引擎通常都會需要用到大量的粒子物件，為了省去Particle物件生成時的運算資源消耗，死去的粒子會被存放到Particle Pool以供未來重複利用。Stardust中回收Particle2D與Particle3D物件的Pool都只有單一個物件來集中管理Particle2D與Particle3D物件，此為Single Pattern。



## Renderer的視覺呈現

### Observer Pattern

　　主迴圈Emitter.step()在生成新粒子、移除死亡粒子、和粒子更新完畢時，都會通知所有監聽此Emitter的Renderer，好讓Renderer作出相對應的處理。此為Observer Pattern，Emitter的角色是Subject，而Renderer則是Observer。



### MVC Pattern

　　Stardust整體的視覺化加上Flash的互動，可以算是使用MVC Pattern。Emitter內部保有的ParticleCollection包含了所有的Particle物件，是屬於存放數值資料的Model；當Model資料更新的時候，Renderer則會將畫面更新，此為View。剩下的Controller部分，則不包含在Stardust中，Stardust當初就是為了Flash互動特效設計的，所以Controller的部分，是每一個使用Stardust的Flash應用程式中的互動：例如「滑鼠左鍵的按下觸發煙火特效」、「滑鼠軌跡上噴發出星星」等。

　　Stardust缺乏了MVC Pattern中的Controller，因為Controller是要交給Flash應用程式設計者來實作的。所以為了展示完整的MVC Pattern，以下附上一段範例程式碼，展示一個「星星噴發位置跟著滑鼠跑」的效果。

主程式

|  |
| --- |
| var emitter:StarEmitter  = new StarEmitter(new SteadyClock(1));  var renderer:Renderer  = new DisplayObjectRenderer(container);  renderer.addEmitter(emitter);  addEventListener(Event.ENTER\_FRAME, mainLoop);  function mainLoop(e:Event):void {  //更新point位置 (Controller)  emitter.point.x = mouseX;  emitter.point.y = mouseY;  //主迴圈  emitter.step();  } |

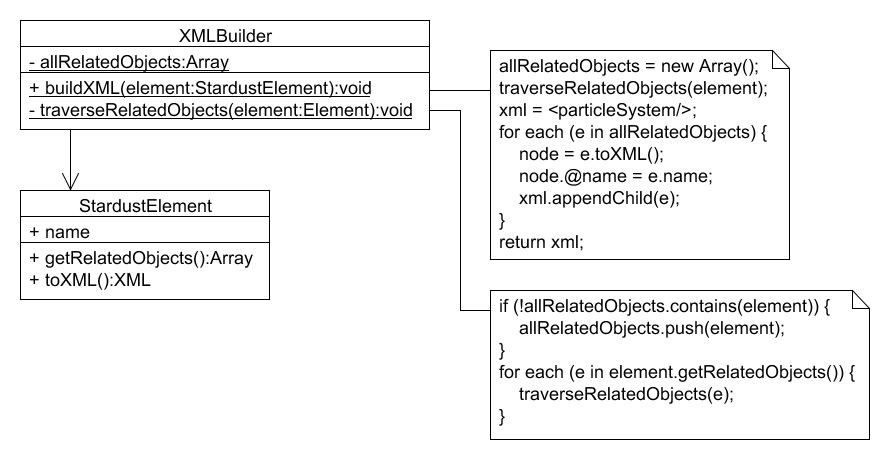
StarEmitter類別

|  |
| --- |
| //繼承Emitter2D  class StarEmitter extends Emitter2D {    //給主程式使用的Reference  public var point:SinglePoint = new SinglePoint();    public function StarEmitter() {  //initializers  //噴出星星  addInitializer(new DisplayObjectClass(Star));  //生命長度介於5010範圍內  addInitializer(new Life(  new UniformRandom(50, 10)));  //位置等於point的(x, y)座標  addInitializer(new Position(point));    //actions  //老化  addInitializer(new Age());  //生命歸零時死亡  addInitializer(new DeathLife());  }  } |

## XML序列化

### Visitor Pattern

　　要將一個粒子系統序列化為一筆XML資料 (每一個XML結點代表一個元素)，必需取得跟此系統相關的所有其他元素的參考，這就是StardustElement.getRelatedObjects()函式的目的：它會回傳包含跟這個元素相關的元素陣列，而XMLBuilder類別扮演Visitor的角色，以遞迴的方式走遍所有元素，呼叫它們的toXML()函式來取得序列化後的XML結點。



# Stardust效能檢討

　　Stardust顧及到系統維護的容易性以及未來擴充性，大量使用Design Pattern，類別數量非常多，分工非常細；然而另一方面，卻也因為類別間需要頻繁地相互溝通，導致Function Call次數頗多：例如若有1000個粒子和5個Action物件，每一次呼叫Emitter.step()就會呼叫5000次Action.update()；這對使用虛擬機器的語言而言(如ActionScript、Java、C#)，是拖垮效能的致命傷之一。

　　我(周明倫)曾於WonderFL上看到有人將所有函式功能展開寫死在一個Function Call中，而得以流暢地模擬25000個粒子；相較之下，若使用Stardust製造同樣的特效，一次只能模擬1000個粒子，效能差距非常大。我曾經試過把Stardust功能展開寫死成單一Function，果然效能就大幅上升了。不過我不認為這是一個程式正確的寫法，一個大型程式專案中如果使用這種方法撰寫程式，儘管可以得到不錯的效能，未來維護起來卻將會極度繁瑣與困難；這是一體的兩面，我覺得犧牲程式效能，換極容易維護的程式，是非常值得的。其實Stardust的效能，在實際粒子特效應用上不是什麼大問題，因為一般的Flash粒子效果用到150個粒子就算非常多、非常華麗了，離1000還有很大一段差距，執行起來是非常順暢的。

　　現在我專注的，將是如何把Stardust因為大量Function Call而造成的效能減損降低。例如在Emitter.step()中要利用Action.update()來更新所有粒子，會需要雙迴圈；原本的雙迴圈程式碼長這樣：

|  |
| --- |
| for each (var a:Action in actions) {  var iter:ParticleIterator = particles.getIterator();  var p:Particle2D;  while (p = Particle2D(iter.particle)) {  a.update(emitter, p, time);  iter.next();  }  } |

但我後來發現Iterator迴圈的速度比for…each迴圈的速度慢，於是我把程式碼改成這樣：

|  |
| --- |
| var iter:ParticleIterator = particles.getIterator();  var p:Particle2D;  while (p = Particle2D(iter.particle)) {  for each (var a:Action in actions) {  a.update(emitter, p, time);  }  iter.next();  } |

將Iterator迴圈與for…each迴圈內外對調，讓前者只走完一次，效能就改善不少了。

　　或許我還可以把一些常用的Action類別的程式碼展開寫死、包裝成單一Action類別，理論上就可以把最大模擬粒子數推至5000以上。

　　Stardust未來的改善空間還很大(當然嘛，跟展開寫死的程式效能差了至少25倍)，我會繼續尋找效能改善的可能性，然後繼續推廣Stardust。

# 課程心得

## 周明倫

　　Design Pattern，不愧是眾OOP程式設計師長久以來的智慧結晶；在上一個暑假自修完Head First Design Patterns與GoF Design Patterns之後，原本對Interface扮演的角色不甚了解的我，忽然有種豁然開朗的感覺：各種機制存在的目的都說得通了。

　　我對Design Pattern躍躍欲試，於是放棄原本的Emitter專案，重新打造一個Stardust粒子引擎。在這其間的許多大小專案，我也將Design Pattern的所學應用上去，成就感真是非常大。

　　這學期的設計模式課程，讓我首次認識了NetBeans IDE，並且令我重拾對Java視窗應用程式開發的興趣與熱情。教授的課程，讓我對原本不是很了解的Java API設計有了新的認知；例如大一下的Java課，我對於教授提供的Sample Code中寫到的這段程式碼，一直無法了解其背後意義：

|  |
| --- |
| new BufferedReader(new InputStreamReader(System.in)); |

　　學完了Decorator Pattern之後，才真正瞭解這樣子的寫法意義為何，以及當初Java API為什麼要這樣設計。

　　另外，隨著這學期課程的進展，我也漸漸認識到，自己以前撰寫程式的時候，其實已經不知不覺用到了不少Pattern；當時只是覺得這樣子寫程式似乎比較好管理與擴充，結果沒想到竟然無意間使用到了Design Pattern，心裡又驚又喜。在我了解到我這些程式的撰寫方式是Design Pattern之後，我就會開始養成習慣多留意能夠應用Pattern的地方，讓我撰寫程式的時候更能得心應手，並且會更加考慮到未來的可維護性與擴充性。

　　這學期初我修習系上的網路與多媒體實驗時，助教要我們製作一個小畫家程式，以當作開始正式做驗之前的練習。我發現幾乎所有實驗室的人，都把程式碼塞在一個JFrame類別中，近千行的程式碼看起來頗不容易維護；而我因為暑假已經有自修Design Pattern，就想辦法將Pattern應用在這個專案中。我利用了Command Pattern把線段、圓形、矩形、橡皮擦、檔案讀寫等功能分離成單一Command類別，又用Memento Pattern作出Undo/Redo的功能：其實就是對一個Stack執行Command物件的Push與Pop而已。其他同學遇到一個問題，就是由於一條短直線線段是一個Command，每一次Undo都只會將一個小線段移除掉，而不是把滑鼠畫出來的一整個筆劃都消除；這個問題雖然讓我稍為煩惱了一下，但是我馬上就想到可以用Composite Pattern將一筆畫的多個線段Command包裝成一個Composite Command，再把這個Composite給Push到Stack中，如此一來一次的Undo就可以將整個筆劃完全消除。相較於其他同學需要在近千行的程式碼中勉強湊出解決方法，我只需要再寫一個Composite Command類別就可以了，Design Pattern的威力與實用性真是不容小覷。

　　很謝謝教授這學期開了這門課，幫助我釐清了一些自修時還不是很了解的觀念。像是一語道破Flyweight Pattern本質上就是Singleton Pattern的變體，還有Visitor Pattern為什麼破壞了Information Encapsulation。這門課對我未來開發的程式專案，真是助益匪淺！

## 林昭瑋

老實說，在修習教授開的軟體設計模式前，我是一個從來沒接觸過Object-Oriented 語言的新手，只有接觸過C++，具備一般的程式語法能力而已。為了進入OO的世界，我去買了Head First系列的JAVA入門書，

我覺得OO的世界真的很有趣，它使我能夠將自己的想法變成各種Object，然後進行排列和溝通，而Design Pattern則是提供了我在不同目的下如何布好各個Object的樣板，使program之後的擴充性和可維護性大大提升，並具有結構性的規劃，使Program中不會有一堆Reuse的程式碼，讓Programmer之後在看自己的Program時，能有更深刻的了解。

從一開始的Template Method Pattern、Strategy Pattern、Visitor Pattern、Factory Pattern的各種變體、Builder Patten到最近的Adapter Pattern、Façade Pattern、Mediator Pattern等，我覺得每一個Pattern都很有味道，切合他本身的Intent，雖然有些Pattern的結構上來說是類似，甚至是相同的，但其Intent的分歧，讓每個Pattern都具有自己的特色。

譬如說Strategy Pattern和State Pattern，他們在結構上根本是一樣的，但由於Strategy Pattern的目的是將一些同樣種類的Algorithms包裝起來，使其能夠輕易的改變使用的Algorithm，但State Pattern的目的是要使Object可以根據內部State的改變而讓Behavior改變。 這兩者的差別在於前者使用者了解目前我要哪一種Behavior，像Head First的例子，我們知道RubberDuck的飛法是FlyNoWay，RedHeedDuck是FlyWithWings，並且我們能夠在Runtime時依照需求去改變他的行為，新增的FlyWithRocket即為一個例子，而後者以GumMachine為例，我們並不知道目前的State在哪裡，只要他會產生對應的Behavior就好。

像這種Pattern之間的比較，教授在授課時講了不少，在Head First的

Design Pattern 一書中每一章節的結尾附近也會出現，這種比較不僅讓我了解兩個(或多個)Pattern之間的差異，也讓我更清楚個別Pattern的Intent，我認為這種比較是這門課不可或缺，可以說是很重要的一環，也讓未來在實際Coding時，在決定使用哪種Pattern能有更明確的考量。

# 相關連結

Stardust Particle Engine首頁

<http://code.google.com/p/stardust-particle-engine>

FlashDevelop編輯器

<http://www.flashdevelop.org/community>

Adobe Systems官方網頁

<http://www.adobe.com>

CJ’s Blog (含Stardust開發現況與開發日誌)

<http://cjcat.blogspot.com>

# 參考資料

1. Flint Particle System粒子引擎

<http://flintparticles.org>

1. Box2D物裡引擎

<http://www.box2d.org>

1. ***ActionScript 3.0 Design Patterns: Object Oriented Programming Techniques*** (2007). Adobe Developer Library. William Sanders, Chandima Cumaranatunge.
2. ***Foundation ActionScript 3.0 Animation: Making Things Move!*** (2007). Friends of ED. Keith Peters.
3. ***Essential ActionScript 3.0*** (2007). Adobe Dev Library. Colin Moock.
4. ***Head First Design Patterns*** (2004). O'Reilly Media. Elisabeth Freeman, Eric Freeman, Bert Bates, Kathy Sierra.
5. ***Design Patterns: Elements of Reusable Object-Oriented Software*** (1994). Addison-Wesley Professional. Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides.