一個以遊戲程式設計為基礎的物件導向程式設計實習課程
An Object-Oriented Programming Lab Based on Game Programming

陳偉凱 鄭有進
台北科技大學資工系
Woei-Kae Chen and Yu Chin Cheng
Department of Computer Science and Information Engineering
National Taipei University of Technology
Email: {wkc, yccheng}@csie.ntut.edu.tw

摘 要

在程式設計的教學上，目前的主流是物件導向程式設計。為深植物件的觀念，物件導向程式設計所配合的實習課程已漸漸成為重要的教學課程。本論文探討作者採用遊戲程式設計作為物件導向程式設計實習課程的教學經驗。我們實作了一個遊戲程式的應用程式設計框架(application framework)，透過這個框架，我們讓學生在一個學期的課程中，實作一個自行選擇的遊戲程式，藉由遊戲程式的設計導入物件設計的方法與物件的互動，從基本物件中再衍生出設計樣式(design pattern)的觀念與應用，以提升整體程式的品質與教學成效。我們發現這是一個成功的教學模式，在一個學期的實習課程中，學生普遍都有良好的表現，除創造出優良的遊戲程式作品外，並對教學成效表示肯定。

一、簡介

筆者任教之學系根據 CC 2001[5]規劃一系列程式設計的課程。我們的課程結合 CC 2001 中 Imperative-first 與 Objects-first[1]等兩種主要的教學策略，可以簡單的描述為 Imperative-first with strengthened object orientation。本論文主題之「物件導向程式設計實習」課程開設在大二上學期，修習本課程的學生已於大一完成「計算機程式設計」、「計算機程式設計實習」與「物件導向程式設計」等課程。

物件導向程式設計課程的教學目標是讓學生熟練物件導向程式設計與撰寫的技巧，希望能深植使用 C++ [4]的能力、熟練 C++的開發環境，以及訓練寫作大型程式的能力。然而，由於 C++語法非常繁複，物件導向程式設計課程往往耗費許多時間在文法的介紹，導致無法深入探討物件的設計與應用，以及物件之間的互動，因此物件導向程式設計實習課程(1 學分，3 小時)就成為極佳的輔助課程，藉由實習的過程，落實物件導向程式設計的寫作能力。

我們在長期的教學經驗中發現，寫作再多獨立的小程式，都很難讓學生深刻的體會到物件的用法與互動、無法培養系統的觀念，更遑論訓練開發軟體的能力。因此，我們在物件導向程式設計實習課程中，嘗試要求學生用一整個學期的時間，寫作一個大一點的程式，也就是說讓物件導向程式設計實習扮演初級程式設計課程之總成專題(capstone project)的角色，而遊戲程式就成為總成專題極佳的題材。

使用遊戲作為程式題目的最大優點在於提高程式設計的趣味性。由於幾乎每個人都會玩電腦遊戲，因此設計電腦遊戲對於許多學生而言具有致命的吸引力，比起傳統的作業題目(如數理的計算、商業的應用)，遊戲程式顯得更生動有趣，並且較能激發學生的自我動力。我們發現在這樣的課程中，學生往往能主動挑戰更難的題目，並廢寢忘食的完成自己的目標，不需老師不斷的叮嚀，因此教學成效自然提升了。使用遊戲作為程式題目的另一個優點則為能自然且有效的融入 CC 2001 的教學目標，一般被認為相當難在初級程式設計課程中講授的若干課題，如 PF5 事件驅動程式設計(event-driven programming)及 SE2 應用程式介面之使用(using APIs)等，在遊戲程式設計的過程中都自然出現。

一般而言，一個中小型的遊戲(例如小精靈)即已具備小型應用系統的規模。設計一個遊戲必須設計各式各樣的物件，其複雜度足以驗證課堂上講的物件導向程式設計理論，在實作過程中，學生可以體會到，每個物件怎麼設計，物件間怎麼互動、怎麼組織，因此能充分發揮物件導向程式設計的優點。

使用遊戲程式作為實習題目的最大問題或許在於：「學生做得到嗎？這些初學程式的學生連小程式都還寫得不大好，他們能寫的出遊戲程式嗎？一個學期能完成嗎？」。這些問題的關鍵在於我們必須給予學生一個良好的出發點，讓學生不必全部從頭做起。也就是說，我們必須提供學生一個容易上手的遊戲開發環境，因此我們設計了一個寫作遊戲程式專用的應用程式設計框架(application framework)，讓學生免除低階的操作(如作業系統、繪圖、音效等的控制)，直接從遊戲所需的物件寫起。

透過我們的框架，大多數的學生在實習課程中都能得到充分的收穫，也常常有非常專業的遊戲作品出現。我們發現這樣的教學模式執行的非常成功，因此本論文將我們的執行方法、教學環境，與

心得提出來與大家共享。以下各小節依序介紹我們的遊戲應用程式設計框架、遊戲程式常見的物件架構與設計樣式[3]、課程的執行與結論。

二、Game 應用程式設計框架

對程式的初學者而言，設計一個遊戲程式不是一件容易的事。遊戲程式涉及許多作業系統（繪圖及音效）介面的操作，與速度控制上的考量，而這些細節，在學生的程式設計能力還不是很扎實之前，並不合適作為課程的內容。因此，我們先將繪圖及音效的問題解決，並且寫成現成的物件供學生引用，讓學生專注在如何設計自己的遊戲程式，從中學習實作經驗，以及解決問題的方法。

自 1998 年開始，我們為物件導向程式實習，設計了一個開發遊戲程式的環境，稱為 Game [2]，作為教學的輔助工具。實作上 Game 以微軟的 VC++與 MFC(Microsoft Foundation Class)開發，為了提升遊戲的效能與精緻度，我們採用 DirectX 介面支援底層的繪圖與音效。在過去六年中，Game 歷經數度改版，由一個簡單的繪圖函式庫，漸漸轉變為一個應用程式設計框架。Game 大體上包含下列功能：

- 提供簡單易用的(點陣圖)繪圖物件
- 提供簡單易用的音效物件(wave 與 midi)
- 可選擇切換全螢幕模式或視窗模式
- 內建遊戲狀態性的控制
- 內建遊戲迴圈速度的控制

以下簡單介紹 Game(4.1 版)的架構。如圖 1 所示，Game 這個類別是整個遊戲的控制中樞，同時也是遊戲與 MFC 的 Facade，我們使用 State 設計樣式把遊戲的狀態明確地分成 3 個部份，分別是遊戲起始狀態(GameStateInit：顯示遊戲開始的畫面，並等待玩家下命令進入遊戲執行狀態)、遊戲執行狀態(GameStateRun：正在玩遊戲的狀態)及遊戲結束狀態(GameStateOver：顯示遊戲結束的畫面，並詢問玩家是否重玩)。這三個狀態各有不同的使用者介面，因此每個狀態都必須有初始化(Init 及 OnBeginState)、事件處理(Events)，以及移動(Move)與顯示(Show)遊戲物件的能力。基本上，撰寫遊戲就是撰寫程式實作每個狀態的成員函數(Init、OnBeginState、Events、Move、Show 等)。

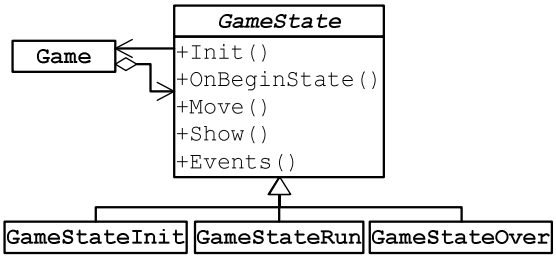

圖 1 遊戲的狀態性

圖 2 說明一個遊戲狀態內部的流程，每個狀態的成員函數都是由事件驅動(event driven)的。每當切換狀態時，Game 首先會觸發(呼叫)該狀態的 OnBeginState(進行初始化工作)，然後依固定頻率(預設為每秒 30 次)反覆觸發 Move(移動遊戲物件)與 Show(顯示遊戲物件)事件，以產生動態的效果，在這個過程中並隨時依使用者動作(滑鼠與鍵盤事件)觸發不同的 Events(事件處理程序)。當某些條件成立時(例如遊戲中的主角死亡了)，再切換至下一個狀態(通常由 Move 執行判斷工作)。

Game 每次呼叫 Show 之前會將螢幕(或遊戲視窗)全部擦掉，因此 Show 只要將全部遊戲物件依照新的座標(Move 所指定的座標)重繪即可。這樣設計的目的是為了簡化繪圖模型(詳述於後)，讓程式不用考慮移動物件時必須擦掉其現有圖形的問題，這個模型對於早期的電腦並不可行，但由於目前電腦的運算與繪圖效能大幅提升，足以應付全部重繪，所以遊戲程式的設計可以簡化很多。此外，Show 在執行時並不是立刻將圖案顯示到螢幕，而是將圖案貼到 video RAM 的緩衝區，待 Show 完成後，Game 會把該緩衝區利用硬體功能顯示到螢幕上，因此顯示時不會有物件先後出現的問題。

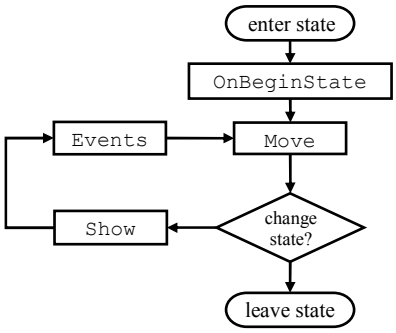

圖 2 遊戲狀態的控制

遊戲的撰寫其實就是撰寫各式各樣的遊戲物件。以飛行射擊遊戲為例(如任天堂的 1943)，遊戲程式設計者必須設計出我方飛機、敵方飛機、子彈、炸彈、船艦、補給品等物件。當遊戲進行時，Move 負責移動(所有的)遊戲物件，而 Show 則負責顯示(所有的)遊戲物件，因此每個物件本身都應有移動與顯示的能力。

自第三版開始，我們將 Game 原始程式的版權設定為 GPL。我們歡迎任何人拿這些原始程式去發展新的遊戲，也歡迎任何對於這個程式的回饋與意見。以下子小節說明我們對於繪圖與音效的抽象化，以及我們提供的範例程式。

二.一、繪圖與音效功能的抽象化

對於遊戲而言，繪圖(尤其是彩色的點陣圖)可以說是最重要的一環，一則圖形的品質與遊戲的精緻度有直接的關係，二則動態的遊戲對於繪圖與畫面重整的速度有強烈的需求。然而遊戲的繪圖也是

很繁複的工作，必須使用 double buffer 以避免畫面抖動、控制 video RAM 的配置(allocation)與釋放(release)、善用有限的 video RAM(相同的點陣圖只儲存一份在 video RAM)、處理物件移動時的重繪、解決視窗範圍變動時的重繪(OnDraw)、操作低階的 DirectX API 等，而這些議題對於物件導向程式設計的初學者而言，並不是很容易解決的。因此我們決定對繪圖作高度的抽象化，以簡化繪圖的複雜度，讓學生可以輕易地上手。

如圖 3 所示，我們把繪圖的細節全部隱藏起來，遊戲程式設計者只要操作一個簡單的繪圖類別(稱為 CMovingBitmap)即可，這個類別的物件可以使用 LoadBitmap 函數載入指定的點陣圖、使用 SetTopLeft 函數指定此圖形(的左上角)在螢幕上的座標、使用 ShowBitmap 函數將圖形貼到螢幕上，並具有查詢相關資訊的功能。

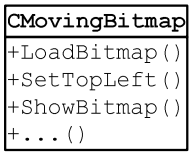

圖 3 點陣圖繪圖類別

簡單地說，設計者要在螢幕上顯示一個不斷移動的圖形，只要使用一個 CMovingBitmap 物件，在每次 Move 時(某狀態的 Move 函數)指定不同的座標，然後在 Show 時(某狀態的 Show 函數)呼叫該物件的 ShowBitmap 即可。而 Game 框架則自動在背後支援並處理前述的各種繪圖問題，因此繪圖變的非常簡單。

對於音效的支援，我們同樣做了高度的抽象化。如圖 4 所示，我們提供一個音效類別，稱為 CAudio。由於音效可能在遊戲中的任何物件用到，我們把 CAudio 設計成 Singleton 設計樣式，以便操作。使用 CAudio 時只要先利用 Load 函數載入指定的 wave 檔或 midi 檔，之後就可以使用 Play、Pause、Resume、Stop 等函數控制其撥放(可以同時撥放數個音效)。就如同繪圖一樣，CAudio 類別將低階的音效控制做了封裝，因此操作音效就變得非常簡單。

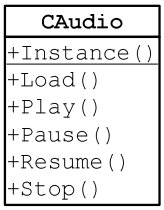

圖 4 音效類別

由於我們對於繪圖與音效都作了高度的抽象化，對於設計者而言，其使用是簡單多了，但是卻也產生了一些限制，例如設計者不能自行控制 video RAM 的配置與釋放，因此比較沒有彈性，不過，我們模型(model)仍能適用於絕大多數 2D 的遊戲，僅有少數對於繪圖或音效性能有特殊需求的遊戲才會無法使用 Game 應用程式設計框架。

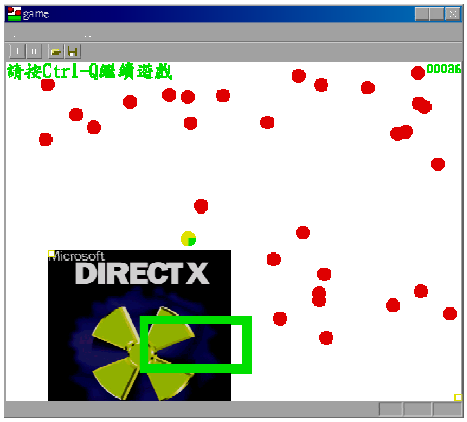

圖 5 範例程式(GameStateRun)的畫面

二.二、範例程式

Game 應用程式設計框架以原始程式的方式提供給學生使用。我們寫了一個範例程式，作為學生設計自己的遊戲程式時的參考。如圖 5 所示，我們的範例類似一個簡化的打磚塊遊戲，我們設計了「球」物件、「拍子」物件、「整數」物件(右上角)、「動畫」物件與「背景」物件。我們的範例程式在 Pentium II 等級的電腦就可以執行的很好，因此除非是設計很複雜的遊戲，目前電腦的效能都綽綽有餘。

三、遊戲程式常見的物件架構與設計樣式

學生所選擇的遊戲五花八門，每個遊戲的架構都各自不同，但是遊戲程式仍有許多共通點。以下我們以飛行射擊遊戲(例如任天堂的 1943，見圖 6)為例，說明遊戲程式常見的物件與設計樣式。

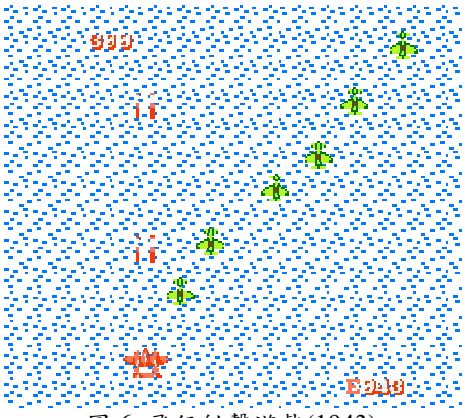

圖 6 飛行射擊遊戲(1943)

由於學生幾乎都沒有寫作過遊戲程式的經驗，其物件設計大多是採用由下而上的方式完成。第一個物件通常是我方飛機，飛機需要顯示在螢幕上，因此通常程設計成圖 7 的架構，其中 Move 依飛機的狀況(例如正在往上飛)改變其座標(x, y)，而 Show 則利用 CMovingBitmap 將圖形顯示出來。

由於 Airplane 與 CMovingBitmap 的介面並不相同，我們會建議學生使用 composition 的關係，這也是對學生解釋“is a”與“has a”差別的好時機。

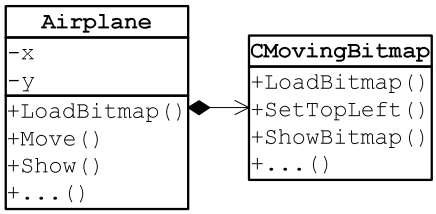

圖 7 我方飛機類別

完成初步的我方飛機之後(可以顯示與控制飛機的移動)，通常會想讓飛機的畫面更精緻，使得飛機在移動時能顯示不同的圖形、讓飛機的尾巴能噴火，因此必須引進動畫(好幾張圖輪流在同一個位置撥放)。這時候設計就會演化為如圖 8 所示的架構，其中 CAnimation 類別是我們提供的範例之一。這也與解釋物件分工原理的好時機，讓學生理解物件設計必須做不同層次的抽象化，而不是直接把動畫邏輯寫到 Airplane 裡面。

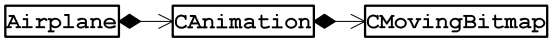

圖 8 增加動畫的我方飛機類別

接著學生會想要寫作(一顆)子彈。由於除了控制方式不同外(我方飛機的飛行由鍵盤或滑鼠控制，而子彈的飛行則由程式的邏輯控制)，子彈的程式有很多與我方飛機相似之處，因此架構就變成圖 9 所示(為簡化圖示，我們省略 CAnimation 與 ÇMovingBitmap)。子彈的生命週期與飛機並不相同(子彈一出螢幕或炸到敵機時，其生命週期就終止了)，因此我們會建議學生將飛機與子彈設計成“knows a”的關係，這個階段是解釋 composition 與 association 差別的機會，也是導入繼承架構的好時機。另外由於子彈類別會有飛機沒有的功能(如被發射)，因此飛機是直接引用 Bullet 類別，而不是 FlyingObj 類別(解釋 Concrete 類別與 Abstract 類別指標的差異性)。

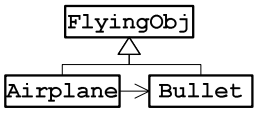

圖 9 子彈類別

其次學生會希望子彈能連發(畫面上能同時出現一群子彈)，而不是只有一顆。這時候我們會建議學生導入 Composite 設計樣式，如圖 10 所示。這樣的設計一方面讓學生理解子彈群(Bullets)類別存在的必要性(重伸物件分工與抽象化的原理)，一方面導入設計樣式與多形(polymorphism)的程式寫作。在這裡使用 Composite 設計樣式將能使子彈的樣式更有彈性(不必侷限於 Bullet 一種)，並且使子彈群能遞迴的複合(composite)組合成更大的子彈群(例如 1943 裡面有一種子彈模式會每次發射 5 顆放射狀的子彈，而這 5 顆子彈合起來即可被視為是一大群子彈中的一小群子彈)。

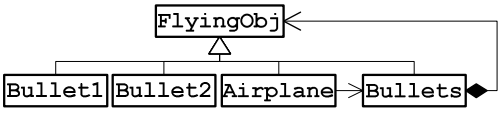

圖 10 子彈群類別

再其次學生會想要設計敵機、敵機群、敵鑑與敵鑑群，這又是 Composite 設計樣式的應用(為節省篇幅，我們就不再一一畫出類別圖)，或是設計不同的飛機狀態(當飛機吃到寶物時，飛機可以升級)，這是 State 設計樣式的應用。不同的遊戲會應用到不同的設計樣式，不過 Composite 和 State 可能是遊戲中最常見的設計樣式，其他的設計樣式如 Template Method、Factory Method、Facade、Strategy、Command、Decorator、Proxy、Observer 等也都是遊戲程式常用的。

四、課程的執行

依照我們的課程規劃，物件導向程式設計實習課程的先修課程是物件導向程式設計。也就是說當學生認識物件導向程式設計的觀念與 C++的文法後，學生就具備寫作遊戲程式的條件了，當然學生如果修過資料結構、演算法等其他課程會更好，不過這些課程並不是絕對必要的，事實上學生在設計過遊戲程式後，往往因為程式的需要而激發修習進階課程的動力。

我們把物件導向程式實習課程的執行方式，分成下列幾個要素分別說明：分組、目標的訂定、目標的確認、一對一的指導、進度的追蹤、助教的支援、期中考核、評分。

- **分組**：

  我們讓學生自行分組，每兩個學生一組合作一個遊戲程式。分組的目的一方面是減少全班的組數，以便一組一組的指導，另一方面則是讓學生形成團隊，有互相學習與合作的機會。我們發現同組的學生經常能互相學習，當同組兩人的程式能力有較大落差時，這種現象更明顯，程度普通的學生往往能從程度較好的學生身上學到很多技巧(因為不斷的看別人的程式寫法)，因此每個人都會進步，而不是一組只有一個學生在做。另外，遊戲程式有很多美工的工作，透過分工可以有效降低個人的工作量。

- **目標的訂定**：

  我們在開學第一週將歷屆學長中較出色的遊戲作品交給學生玩(含原始程式)，讓學生作為訂定自己遊戲題目的參考，學生可以自行選擇一款合適自己程度的遊戲作為一整個學期的目標。程度(程式設計能力)好的學生通常會設定較難的目標(遊戲)，希望自己做的比學長的作品更出色；程度普通的學生則通常會選擇較簡單的

題目。

- **目標的確認**：

  我們要求學生在第三週以前確定目標，為避免目標不明確，學生必須把遊戲名稱寫下來。我們鼓勵學生模仿市面上常見的遊戲(例如小精靈)，因此大部分的遊戲功能與玩法都可以定義的很明確。我們在第二週及第三週和學生個別討論其題目，並給予建議，以協助學生選擇適合自己難度的遊戲。這個確認的步驟是非常重要的，有了這個步驟，我們可以避免學生選了不好的題目(太難或太簡單)。

- **一對一的指導**：

  在目標確定後，學生立即就開始進行遊戲程式的設計。當然學生在實作過程中不斷會有問題出現，我們(任課老師)則扮演協助學生思考、設計，甚至除錯的角色。期初的時候，學生最常見的問題是一個物件怎麼設計、介面怎麼訂定，接下來是物件怎麼互動；期中的時候，學生問題會慢慢轉變成怎樣完成特定的功能(例如怎樣設計地圖、怎樣利用地圖偵測碰撞)；期末時，程度較好的學生會詢問程式(物件)怎樣組織架構會比較好(這是引導學生認識與使用設計樣式的好時機)，程度普通的學生則忙著拼湊整個程式，或尋求設計與除錯的支援。

- **進度的追蹤**：

  我們對每一組每隔一週追蹤一次進度(也就是每週追蹤全班一半的組數的進度)。我們在追蹤進度時，一方面一對一的指導學生，另一方面則紀錄學生的進度，並協助學生設定未來的預定進度。如果追蹤時發現學生沒有完成預定的進度，我們就會提醒學生要投注時間在課程上或尋求協助。我們希望學生能持續的進行程式設計，而不是在繳交成果的時候不眠不夜的趕工，因此我們把進度追蹤的結果作為學生的平時成績。

- **助教的支援**：

  在三個鐘頭的上課時間中，我們一方面要追蹤進度，一方面又要指導學生，並回答問題，經常分身乏術，因此學生也常會利用課餘時間找我們問問題。幸運的是本課程已經實施多年，部分修過本課程的學生已經成為研究生，因此我們會指定他們擔任本課程的助教，負責協助回答與解決學生的問題，有了助教的協助，學生的問題就更容易得到處理，依我們的觀察，本課程的助教們也非常受學生歡迎。

- **期中考核**：

  除了每兩週一次的進度追蹤外，我們在學期中舉行一至兩次的期中考核。期中考核的方式是由各組輪流公開報告自己的進度與成果，學生可以藉由公開報告了解他組的進度，達到互相激勵的效果。期中考核的成果也列入學期成績計算。

- **評分**：

  期末我們要求學生撰寫實習報告並繳交整個學期所完成的成果(遊戲程式)，成果的評分以題目的難易度與完成度綜合考慮。學生的學期成績由平時成績、期中考核、實習報告與成果所組成。此外，我們也讓學生互選最受歡迎的遊戲(三名)，並給予額外的鼓勵。

除了上述的執行方法以外，綜合這幾年的教學經驗，我們提出下列幾個值得討論或注意的議題，供讀者參考：

- **點陣圖的來源**：

  設計遊戲程式時總是需要大量的點陣圖，然而本課程並非美術課程，不宜花太多時間在自行設計點陣圖上。學生大多直接將現成遊戲的畫面捕捉下來，再經過圖形編輯軟體(如小畫家)裁剪與修補，然後帶進 Game 框架使用。

- **視窗程式設計的基礎**：

  大多數遊戲的使用者介面都很單純，不需視窗元件(如 dialog、button、list、tree 等)就能完成，所以撰寫遊戲程式時並不需完整的視窗程式設計能力。我們在期初介紹 Game 框架時，必須介紹 MFC 對於鍵盤與滑鼠事件的處理模式，以便導入事件驅動程式設計的觀念，但是我們並不介紹視窗元件的操作，以節省時間。通常學生在修完本課程後，很容易就能學會視窗程式設計。

- **邏輯座標 vs. 螢幕座標**：

  遊戲往往有場景大於螢幕的情形，也就是說螢幕只能看到場景的一部分。遊戲程式設計的初學者往往不知道程式必須分辨並轉換邏輯座標(場景座標)與螢幕座標，導致程式寫得一團亂，又不知道怎麼修改。我們則常常視遊戲的需要指導學生轉換邏輯座標與螢幕座標的技巧。

- **演算的問題**：

  大部分的遊戲都不需要很複雜的演算技巧，因此學生並不需要資料結構、演算法等知識就能設計遊戲程式了。如果學生選了演算很複雜的題目，我們會在確認目標的階段建議學生更換題目，以免題目超出學生現有的能力。如果是不太複雜的演算(例如在地圖上搜尋最短路徑)，則我們就直接導學生怎麼作。

- **程式設計 vs. 遊戲設計**：

  經常有學生在期初提出題目時就希望自定一款新的遊戲，經過幾次的實驗，我們發現與其讓學生自行設計遊戲，不如讓學生模仿遊戲。遊戲設計(制定遊戲規則)是一件很複雜的工作，並不容易做的好，把它放到課程中模糊了課程的焦點。另外，我們的目的是物件導向程式設計的教學，透過遊戲的模仿，學生可以證明其程式設計的能力，如果自訂定遊戲規則時，則學生很容易在寫不出正確功能的程式時，修改遊戲規則以配合其程式，變成本末倒

置。

- **早改 vs. 晚改**：

  依我們的教學經驗，當我們發現學生的基本程式架構寫的不好時，應該愈早要求學生改正愈好。有時候我們會覺得讓學生吃點苦，等到他們自己發現不對時，自己就會改了，但是由於學期很短，通常不能等那麼久，早點改可以避免問題越拖越大，才不會整個學期一事無成。

- **Game 函式庫 vs. Game 框架**：

  Game 一開始只是一個繪圖與音效的函式庫，當時我們覺得不要做太多，以免學生喪失一窺全貌的機會。但是這幾年來，我們發現物件的組織能力是需要訓練的、需要範例的，因此我們把 Game 漸漸改成一個框架，以這個框架本身當作範例，讓學生有模仿的對象，例如說學生需要用到 State 樣式時，就可以參考 Game 的實作方式。

- **用框架 vs. 不用框架**：

  偶而會有自認為程度不錯學生要求不使用 Game 框架，希望自己操作 DirectX 的 API 寫作程式(或甚至希望寫作 3D 的遊戲)。在我們的經驗中，一律要求使用 Game 框架會比較好，使用框架時，學生集中精神在遊戲物件的設計，學到物件導向程式設計的精華，而使不用框架時，學生常常模糊了焦點，在低階的 API 中不斷的掙扎，成果往往很差。此外，使用共同的出發點比較公平，也比較容易評分。

- **物件導向程式設計 vs. 遊戲程式設計**：

  由於我們的物件導向程式設計實習課程以撰寫遊戲程式作為實習的題目，因此我們很容易聯想到這門課是否乾脆正名為「遊戲程式設計」或是「遊戲設計」。我們認為使用遊戲作為題目只是我們的教學策略，這門課的本質還是物件導向程式設計，因此換了名字反而不好。

五、結論

物件導向程式設計(或許是任何課程)的教學最困難的是「如何激發學生學習的原動力？」、「如何達成深入的教學內涵？」，又同時能「滿足學生的個別需要」。我們發現我們的物件導向程式設計實習課程恰能符合這些需求：

- **動力與成就感**：

  在遊戲程式設計的過程中，每當學生完成一個新的功能時，就會得到成就感、得到進一步的動力，而且這種感覺會累積，尤其是期末看到整程式這麼大(常常有幾千行的 C++原始程式)、功能這麼多，而且還能跑時，其成就感是很強烈的。學生也往往因為自己能寫得出這樣的程式，而對自己的軟體設計能力更有信心、對於未來的軟體課程也更感興趣。

- **強化物件教學的深度**：

  多形的運用是物件導向程式設計中非常重要的一環(其實大多數的設計樣式本身就是多形的各種用法)，而遊戲裡面正好充滿了多形的物件，因此遊戲程式實是物件導向程式極佳的練習平台。寫作遊戲時所遇到的實例遠比我們在課堂上舉的片段例子更具說服力，且更能讓學生理解其用法。因此透過這樣設計的課程，學生更能深入體會件導向程式設計的精華。

- **非齊頭式的教學**：

  在我們的經驗中，程式教學很令人困擾的問題是「學生的程度不整齊(每位學生既有的程式設計能力在進入課程前並不相等)」，因此課程的內容必須取捨，以適應大多數的學生。這個問題在我們的實習課中得到很好的解決：對於程式設計已經很熟練的學生，我們推薦他們做較難的題目，在實習課中給予進階的知識，尤其是物件的組織與設計樣式的運用；對於程式設計能力尚待加強的學生，我們則讓他們做比較簡單的題目，重新熟練程式語言的文法與基本物件的設計，相當於補救教學。這種學習上的非對稱性，在一般課程(無論是實習或非實習課) 都是很難做到的。因此，不論學生程度的好壞，在修完本課程之後，總是有進步，對於本課程總是有正面的評價。

本論文探討了以遊戲程式作為物件導向程式實習教材的教學方法，我們提出了我們的教材、教學環境、執行方式與我們的經驗。由於學生在本課程中往往有令人驚艷的表現，我們覺得這種教學模式是值得推廣的成功經驗，因此提出來與大家共享。

參考文獻

[1] Stephen Cooper, Wanda Dann and Randy Pausch, “Teaching objects-first in introductory computer science,” *Proceedings of the 34th SIGCSE technical symposium on Computer science education*, pp 191 – 195, 2003.

[2] Game framework download page, http://www.ntut.edu.tw/~wkchen/works/index.htm.

[3] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, ISBN 0-201-63361-2, 1994.

[4] Stanley B. Lippman, Josee Lajoie, and Barbara E. Moo, C++ Primer (4th Edition), Addison-Wesley, ISBN 0201721481, 2005.

[5] The Joint Task Force on Computing Curricula -- IEEE Computer Society and Association for Computing Machinery, *Computing Curricula 2001 -- Computer Science,* December 15, 2001.