逢　甲　大　學

資訊工程學系專題報告
英雄式角色扮演2D遊戲設計與實作

學　　生：王克強(四丙)
指導教授：林國貴

中華民國九十二年十二月

目　　錄

目錄 I
圖表目錄 III
摘要 IV
第一章　前言 1
1.1　動機與目的 1
1.2　遊戲簡介 1
第二章　遊戲系統架構 2
2.1　角色扮演遊戲初步分析 2
2.2　角色扮演遊戲基本架構 3
2.3　FSM理論的運用 4
第三章　遊戲開發類別庫說明 5
3.1　GAF簡介 5
3.2　GAF使用說明 5
3.3　GAF類別與DirectX元件的比較 7
3.4　GAF函數說明 8
3.4.1　GAFApp類別 8
3.4.2　GAFDDraw類別 10
3.4.3　GAFDDrawSurface類別 12
3.4.4　GAFDInput類別 14
第四章　遊戲實作說明 16
4.1　建立標題視窗的基礎 16
4.2　遊戲狀態的替換 19
4.3　場景及地圖 20
4.4　角色實作 23
4.5　劇情、腳本及道具 26
4.6　戰鬥實作 28
第五章　心得感想 30
5.1　遭遇的困難及解決辦法 30
5.2　系統評估與展望 31
5.3　結論 31
參考資料 33
附錄
A　遊戲基本操作 34
B　地圖編輯器 34
C　場景編輯器 35

D 角色編輯器 36
E 道具編輯器 37

圖 表 目 錄

圖2.1 傳統的被動更新方式.....3
圖2.2 訊息主控更新方式.....3
圖2.3 一般遊戲狀態轉換圖.....4
圖3.1 加入Include files.....6
圖3.2 加入Library files.....6
圖3.3 設定引用的lib檔.....7
圖3.4 GAF類別與DX元件之對應.....7
圖4.1 簡易標題畫面.....18
圖4.2 戰鬥流程圖.....29
圖B.1 地圖編輯器.....34
圖C.1 場景編輯器.....35
圖D.1 角色編輯器.....36
圖E.1 道具編輯器.....37

摘　　要

以往在台灣，電腦遊戲被認為不過就是小孩子不專心課業、怠惰時在玩的不入流玩意兒，但隨著韓國遊戲產業的興起發展，以及曾政承等國際性電玩高手的出現，使得政府開始重視電玩業的潛力，並將之納入「數位內容」的範疇，自此，電腦遊戲也總算得到應有的尊重。

電腦遊戲的製作群總共分為三種人：企劃、美術及程式人員，說起來，電腦遊戲跟電影事實上是很相像的東西，一個遊戲的企劃就像是電影的編劇及製作人，而美術則包辦了演員、場景、道具、特效，程式人員則就像是導演以及動作指導，一個好玩的遊戲就像是一部經典名片一樣，是許多人努力的心血，也是藝術的結晶，它能帶給人們由衷的歡愉，也能表現出許許多多鮮明生動的形象。於是曾有人說過，電腦遊戲是繼繪畫、雕刻、建築、音樂、詩歌(文學)、舞蹈、戲劇、電影(影視藝術)之後人類歷史上的第九藝術。

正如我上面所說，在遊戲製作的過程中，程式人員扮演的角色就像是導演以及動作指導，所以本專題的重點即放在如何以GAF類別庫實作一個角色扮演遊戲的整體架構，至於企劃及美術自然無力旁及，尤其是美術需要專業的素養，所以本遊戲中的圖片皆為從市面上販售之遊戲設計書籍所附光碟中取得。

第一章　前言

1.1動機與目的

自從我在國小五年級收到第一台電腦(生日禮物)之後，電腦遊戲就一直是我的最愛，十多年來，電腦遊戲已經變成了我生活中的一部份，從SLG(SimuLation Game，模擬類遊戲)玩到RPG(Role Playing Game，角色扮演遊戲)，從單機玩到網路，角色扮演遊戲始終都是我晚欣賞的遊戲類型。

我第一次希望從玩家的身份跳到設計者的立場是在民國八十八年七月中的事情，在八十八年七月初，台灣電腦遊戲界出現了第一個改變，那就是萬王之王---台灣第一個萬人線上角色扮演遊戲---的發行，那時剛好我高中畢業，正考完聯考，很快地我一頭栽進了MMOPRPG(Massively Multiplayer Online Role Playing Game，大量多人線上角色扮演遊戲)迷人的世界當中，從此我便不能再滿足於單純地做個玩家，我希望能夠創建一個世界，一個充滿了魔法、劍與愛的奇幻夢境，這便是我專題的動機。

一開始時，我本來是希望能做個區網連線的角色扮演遊戲，就像頂頂有名的《暗黑破壞神》那樣的遊戲，但由於班上似乎沒有其他人想要做遊戲的，所以很明顯這並非我一個人可以做得出來的題目。後來我打算做個3D的單機角色扮演遊戲，可是當我接觸到3D人物模型架構時我便放棄了，因為要叫我一個人製作一整個遊戲需要用到的所有3D模型檔是不可能的事情，何況我並沒有任何美術的底子。於是我又將題目改成2D的單機角色扮演遊戲。

在研究的過程中，我找到了GAF類別庫，這是一套專為2D單機角色扮演遊戲設計的類別庫，它所採用的基礎是Visual C++及DirectX，它將許多Windows API及DirectX SDK的函數封裝了起來，讓我們可以專心於遊戲內容的製作，讓遊戲設計的新手可以儘早熟悉遊戲設計的流程及架構，而不須去煩惱Windows程式設計及DirectX SDK那繁雜又囉嗦的程式碼。

由於我是一人小組，又是Windows程式設計的新手，GAF類別庫自然成為我不二的選擇。

1.2遊戲簡介

我所demo的遊戲是用GAF類別庫所開發，主要有一個開頭標題跟兩個場景檔，出場的NPC(Non Player Control，非玩家控制角色)共有4個，分別是村姑、年輕鐵匠、長老及惡霸。

正如我的專題題目所示---英雄式角色扮演遊戲---可以很明顯地看得出來，玩家是操縱著一位英雄，經由跟村姑、年輕鐵匠、長老的對話，接到消滅惡霸的任務，然後經由跟惡霸戰鬥並打倒他，任務就達成，遊戲也就結束。

整個遊戲都是2D的，遊戲場景是一塊大型的bmp圖檔，並沒有採用一般最常見的拼接地圖及45度視角，因為GAF並不支援。戰鬥則是採取回合制戰鬥，也就是戰鬥時會由場景轉入戰鬥畫面，然後出現戰鬥選單讓玩家選擇動作。與其相對的是即時制戰鬥，也就是戰鬥時不會進入戰鬥畫面，並且不具有戰鬥選單，而是直接在一般場景中用滑鼠點擊敵人便會產生攻擊的動作。

道具方面，遊戲中並沒有提供道具的圖像，而只有以文字來表達。至於魔法以及特效，因為這也都需要圖片的配合，所以並沒有製作，因此遊戲中的戰鬥選單只有「攻擊」、「防禦」、「道具」及「撤退」四個選項。

第二章　遊戲系統架構

2.1角色扮演遊戲初步分析

在角色扮演遊戲中，玩者通常會是指揮一群英雄進行一系列的冒險。遊戲過程集中在逐漸提升他的英雄們的能力與力量。

角色扮演遊戲和冒險遊戲一樣，具有廣大的世界與逐步揭露的故事情節。玩者會要求能從所攜帶的武器到各身體部位所穿戴的盔甲種類精細管理他的隊伍。戰鬥是重要的成分—它通常是英雄們獲取力量，經驗與購買新護甲所需金錢的機制。

幻想型RPGs通常也具有複雜的魔法系統，以及可以用來組成玩者隊伍的眾多不同種族。

經典的角色扮演遊戲除了強調角色成長這項遊戲要素外，尚具有七項不能缺少的關鍵要素：

1. 探索內在：簡單地說就是角色的內心世界應該要引人入勝，讓玩家可以藉由扮演角色，獲得某種程度的內在探索。
2. 史詩故事：簡單地說就是能激勵玩家繼續玩下去的劇情。
3. 戰鬥：從早期的RPGs被稱為砍殺遊戲，就可以了解到角色扮演遊戲裡戰鬥所佔的份量。
4. 支線任務：早期的RPGs大多只具有主線任務，也就是玩家別無選擇性地必須跟著單一的劇情玩下去，但是隨著遊戲的發展，目前的RPGs已全部都具有支線的任務，這可以讓玩家在朝著遙遠的結局前進時，不至於因沒有任何選擇性的劇情而

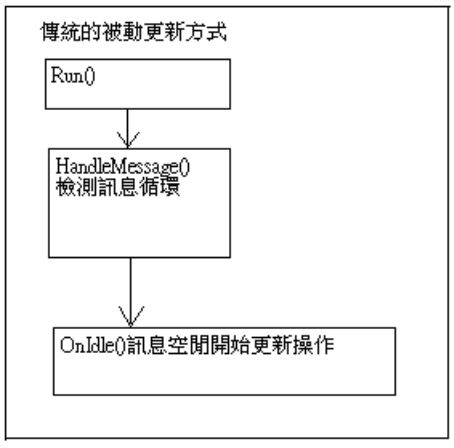
陷入麻痺。

5. 收集寶物：在RPGs中，最讓人興奮的一點就是藉著更換較好、較高等的裝備來提升主角的能力，尤其是在線上遊戲裡，高等的裝備更是炙手可熱的寶貝，甚至有許多玩家願意以現金來交換虛擬的寶物。
6. 資源管理：玩家必須精打細算地管理遊戲中有限的資源，像是魔力值、藥水、金錢等等，像是只能放一次就會花盡所有魔力的大魔法，自然不會是在打低等級的怪物時施放。
7. 問題解決：也就是遊戲中的各式挑戰，一個經典的遊戲絕對是讓玩家隨時必須全神貫注地應付各種突如其來的挑戰，不論是戰鬥方面、解謎方面，也唯有如此才能讓玩家感到新鮮刺激而不會厭煩。

2.2角色扮演遊戲基本架構

角色扮演遊戲基本上必須具備以下的各項要素：劇本、規則、界面、場景、角色、道具、事件、對話、音樂和音效。將以上幾點以程式實作出來，就成了一個RPGs，而將所有要素串連起來的，即是遊戲的主控循環，也就是程式中檢測訊息的系統。

在傳統的Windows訊息系統中，更新方式是被動的，大致上是如圖2.1般的流程：



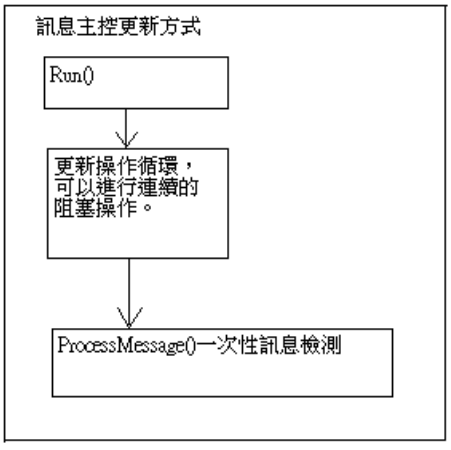


圖2.1 傳統的被動更新方式　　圖2.2 訊息主控更新方式

在遊戲中，我們則是使用主動的訊息檢測方式，如圖2.2。而我們之所以必須使用圖2.2的訊息檢測方式，主要是因為圖2.1的方式會造成主程式更新操作的連續性被打斷，一些阻塞同步操作也因此無法完成。

比如說，當玩家通過鍵盤操縱人物行走，如果鍵盤訊息不斷發送

(像是一直按著某個鍵，就會發送一連串的鍵盤訊息)，這時程式根本就沒有時間進入空閒處理，那就會影響畫面的更新，當然就容易發生程式的錯誤。

而圖2.2的方式則是將Run()改寫，將圖2.1中OnIdle()所做的畫面更新以及其他空閒處理搬到Run()中做掉，最後再統一檢測一次訊息。由於拋棄了空閒處理機制，所以只需要使用ProcessMessage()來處理消息即可。

2.3 FSM理論的運用

一個RPG遊戲中實際上也有著不同的情況，需要不同的程式去處理。比如標題畫面的顯示，在場景之中主角的控制，戰鬥的時候使用各種魔法等等。試想一下，如果把這一切全部放到一個函數中去完成，想想這是多麼臃腫、可怕的一個函數！因此，必須把對應這些不同情況的處理程式碼分開來寫，放在不同的函數甚至單獨做一個類別來處理。那麼，關鍵問題是哪一個情況需要用哪一個對應的函數來處理，以及如何來管理這麼多的情況和對應函數。

解決這個問題可以採用計算理論值中最簡單的FSM來處理。FSM(Finite Status Machine，有限狀態機)。簡單點說，FSM就是一台抽象的機器，它可以處在有限制的狀態(即前面講的各種情況)。這台機器可以讀取各種輸入，它可以根據輸入和當前狀態按照某一個法則(對應關係)進入到一個新的狀態。實際上，RPG遊戲也可以視為一台有限狀態機，遊戲可以選取的狀態有「場景狀態」、「標題畫面狀態」、「戰鬥狀態」、「片頭動畫狀態」等等。對應每一個狀態，編寫一個(或幾個)函數和類別來完成這個狀態的操作，並且在接收到了足夠的資訊之後就跳轉到其他狀態。這樣，整個遊戲就可以有效地組織運作起來了。圖2.3是一個實際的遊戲常見狀態轉換圖。

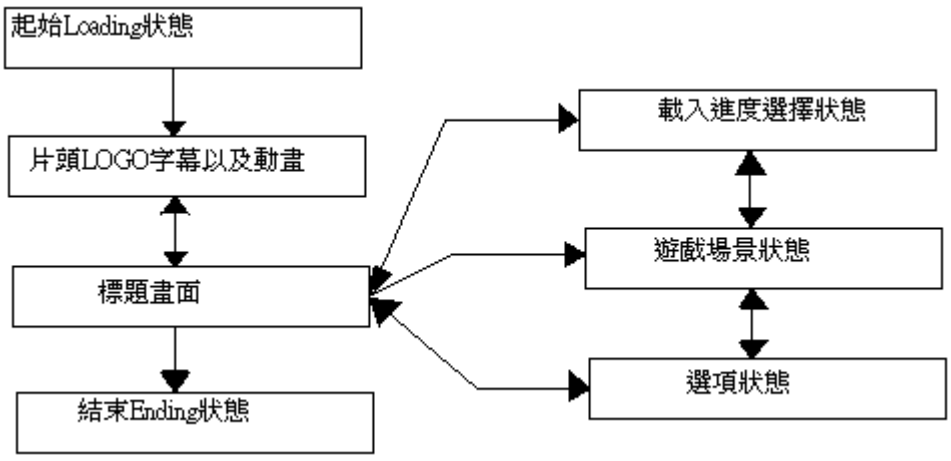


圖2.3 一般遊戲狀態轉換圖

第三章　遊戲開發類別庫說明

3.1 GAF簡介

在DOS時代，寫遊戲可不是那麼簡單的事，它需要用到很多非常低階的工具，比如組合語言。在Windows時代，遊戲設計師似乎有了一線曙光，因為通過WIN32 SDK開發平台，可以做出設計師想做的遊戲。但是問題也隨之而來，數量龐大的API函數使程式碼變得冗長；而一些重複性極大的工作又總是在消耗遊戲設計者的寶貴時間。

對於初學者而言，要想學好VC或其他語言開發工具，往往需要經過C++語言的基本功訓練，到Windows程式設計的學習，然後才能開始著手一項工程。而如此漫長的學習過程，往往導致初學者望而卻步。

不過GAF類別庫在極大程度上解決了程式初學者難學難用的苦惱。GAF就是這樣一套方便好用的類別庫，與其他類別庫相比，它有一個極大的特色：它是為遊戲應用而生的，所以它的設計者將其命名為GAF，意思是「Game Application Framework—遊戲應用程式框架」；它集合了許多便於使用的模組，只要認真學習其製作思想和方法，就可以通過簡單的程式碼製作出一部RPG遊戲；易用性是GAF的一大特色，使我們可以盡快地瞭解RPG遊戲設計的秘密，可以更輕鬆地掌握遊戲設計的奧妙。

3.2 GAF使用說明

在開始引用GAF來設計遊戲之前，我們必須先做一些起始設定。先打開VC++6.0，然後在功能表的地方選擇【Tools】再選擇【Options...】，之後會彈出一個對話方塊，選擇Directories頁籤，然後在Directories:中加入Include files，方法是：點選空白行，然後會出來一個文字方塊，可以由自己填入路徑，也可以按文字方塊末端的按鈕來瀏覽路徑，在加入了新的路徑之後，必須將其放在其他路徑之上，可使用右上角的上下鍵來移動路徑順序。如圖3.1。

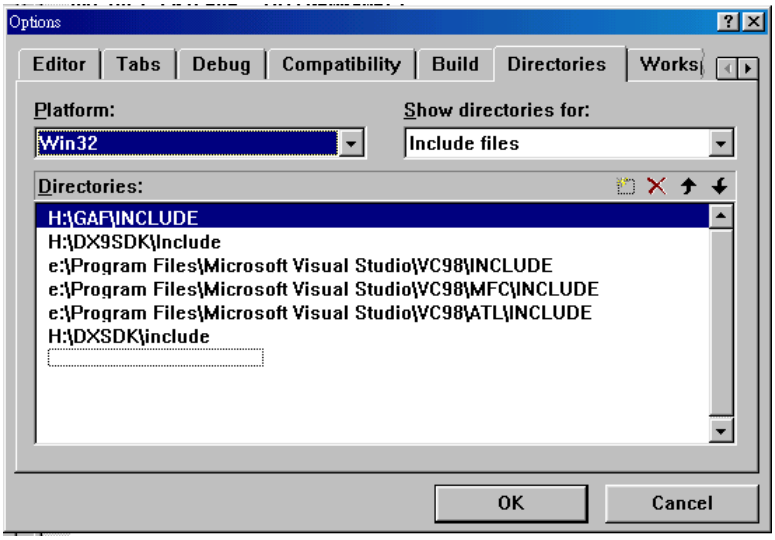


圖3.1　加入Include files

新增完Include files之後，再來是要新增Library files，先點選Show directories for:下拉式清單，選擇Library files，然後就如同新增Include files一樣，在下面的文字方塊中輸入路徑，再將其移到所有路徑之上。如圖3.2。

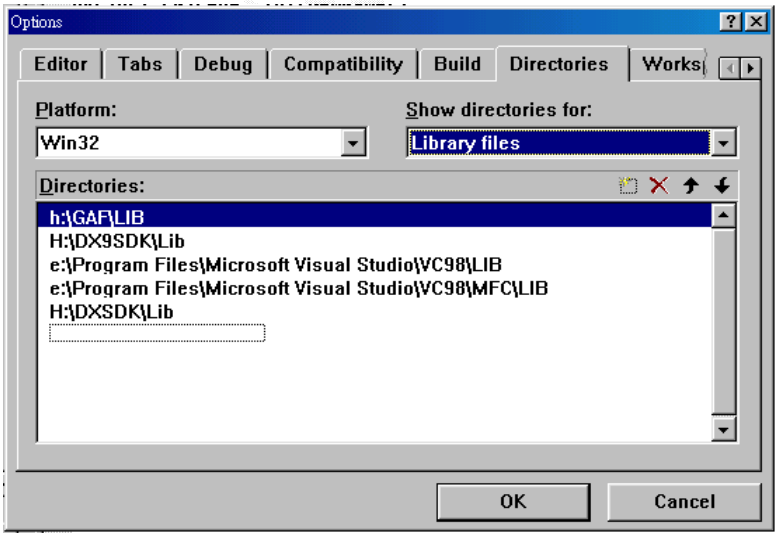


圖3.2　加入Library files

最後的一項工作則是執行功能表的【Project】→【Settings...】，會彈出如圖3.3的對話方塊，然後在Link頁籤中的Object/library modules:欄位裡加入gafd.lib，再將其他的.lib都消掉。

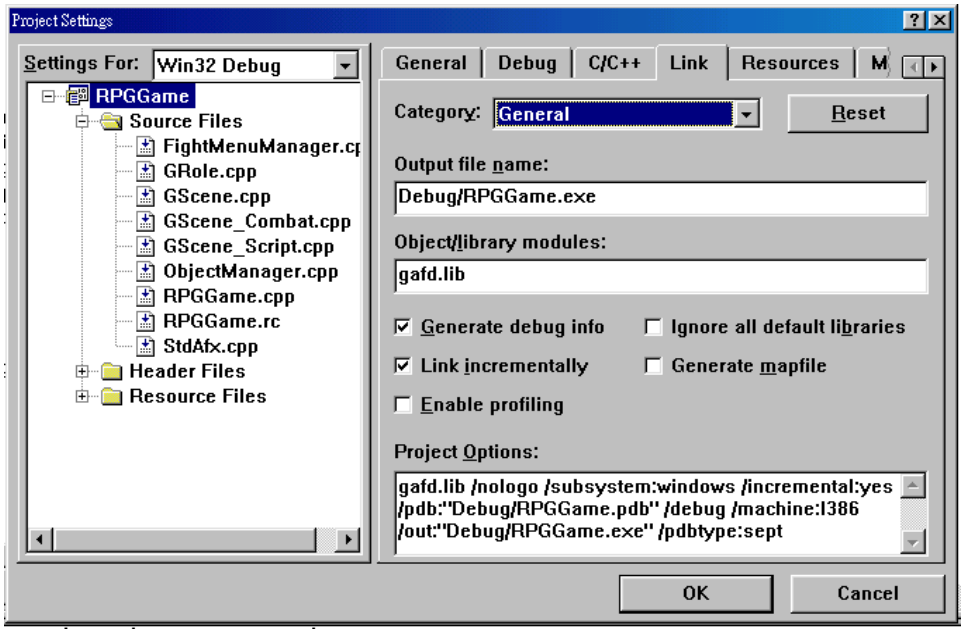


圖3.3　設定引用的lib檔

接下來我們就可以使用GAF類別庫了。

3.3 GAF類別與DirectX元件的比較

GAF的封裝方式事實上跟MFC很像，只是MFC是封裝了Windows API，而GAF是封裝了DirectX(當然也有封裝WinAPI)。

相對應於DirectX的三大元件：DirectDraw(8.0版以上則跟Direct3D合併為DirectX Graphics)、DirectInput、DirectX Audio(包含DirectSound及DirectMusic)，GAF類別庫分別以GAFDDraw、GAFDDrawSurface、GAFDInput、GAFDMusic、GAFDSound、GAFDSoundBuffer等六大類別來封裝它們。它們彼此間的關係如圖3.4。

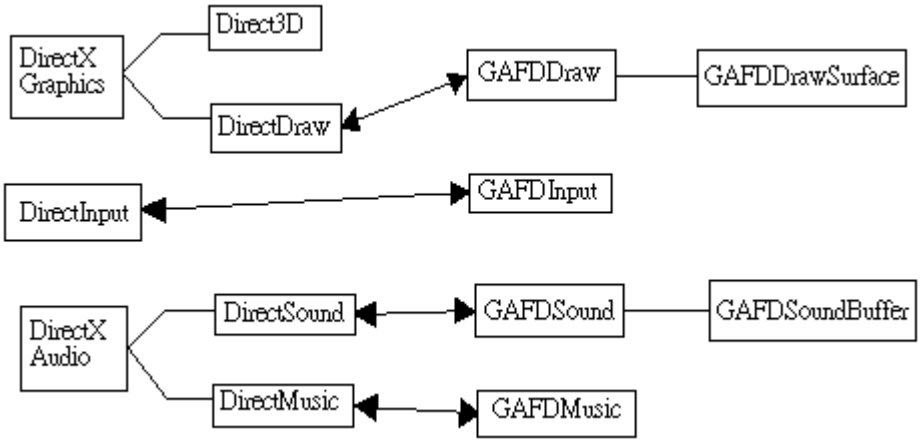


圖3.4　GAF類別與DX元件之對應

至於DirectX的其他元件，像是Direct3D、DirectPlay，GAF並沒

有支援，就如同我之前的章節中所說的，GAF是專門為2D單機RPG所設計的類別庫。

3.4 GAF函數說明

在下一章的遊戲實作說明之前，我必須先簡單地介紹一下幾個GAF的主要類別及其函數之功能。

3.4.1 GAFApp類別

這個類別主要是跟視窗建立、訊息機制有關。

1. 函數：virtual BOOL InitInstance();
   功能:初始化視窗實體,可以由使用者重新載入初始化自己的資料。
2. 函數：virtual int ExitInstance();
   功能：結束視窗實體，可以由使用者重新載入釋放自己的資料。
3. 函數：virtual int Run();
   功能:遊戲主循環迴圈,可以由使用者重新載入做自己的遊戲循環。
4. 函數：virtual void OnIdle();
   功能：訊息空閒時間處理，可以在這裡放置重繪和更新遊戲資料操作。
5. 函數：BOOL HandleMessage();
   功能：阻塞訊息循環，如果沒有訊息則呼叫OnIdle()。返回FALSE則表示可以退出程式。
6. 函數：BOOL ProcessMessage();
   功能：異步消息處理，由程式設計者自己控制消息處理，不呼叫OnIdle()，返回FALSE則表示可以退出程式。
7. 函數：void BindWndMenu(UINT uID);
   功能：給視窗新增功能表。參數uID是功能表的資源ID編號。
8. 函數：void BindWndClassName(LPCTSTR szWndClassName=GAFDEFWNDCLASSNAME);
   void BindWndClassName(UINT uID);
   功能：修改視窗類別。這個函數有兩個形式，第一個形式可以通過指定字串來修改視窗類別名稱，第二個形式則通過資源ID編號。
9. 函數:void BindWndTitle(LPCTSTR szWndTitle=GAFDEFWNDTITLE);
   void BindWndTitle(UINT uID);
   功能：修改視窗標題。這個函數有兩個形式，第一個形式通過字串來修改視窗標題，第二個形式則通過資源ID編號。
10. 函數：void BindWndWidth(int w);
    功能：修改視窗寬度。參數w是你想要的視窗寬度。

11. 函數：void BindWndHeight(int h);
    功能：修改視窗高度。參數h是你想要的視窗高度。
12. 函數：void BindWndIcon(UINT uID);
    功能：修改視窗小圖示。參數uID是小圖示的資源ID編號。
13. 函數：void BindWndStyle(DWORD dwExStyle=WSEXAPPWINDOW, DWORD dwStyle=WSPOPUP|WSVISIBLE|WSCAPTION|WSSYSMENU);
    功能：修改視窗風格。第一個參數dwExStyle指定視窗延伸風格，第二個參數dwStyle指定視窗基本風格。通常情況下可以選擇預設值，即不填寫任何參數。
14. 函數：void ScreenToClient(LONG *px, LONG *py);
    void ScreenToClient(LPPOINT lpPoint);
    功能：將螢幕座標轉換為客戶區(即除了標題，視窗邊框以及功能表工具欄區的視窗範圍)座標。有兩種形式，第一個可以指定要轉換的座標x和y座標值，第二種可以指定一個POINT指標。
15. 函數：void ClientToScreen(LONG *px, LONG *py);
    void ClientToScreen(LPPOINT lpPoint);
    功能：將客戶區座標換成螢幕座標。有兩種形式，第一種可以指定要轉換的座標的x和y座標值，第二種可以指定一個POINT指標。
16. 函數：virtual int OnActivateApp(BOOL bActive, DWORD dwThreadID);
    virtual int OnCreate(LPCREATESTRUCT lpCreateStruct);
    virtual int OnSetFocus();
    virtual int OnChar(UINT nChar, UINT nRepCnt, UINT nFlags);
    virtual int OnClose(HWND hWnd);
    virtual int OnCommand(UINT nNotifyCode, UINT uID, HWND hWndCtrl);
    virtual int OnDestroy();
    virtual int OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
    virtual int OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags);
    virtual int OnKillFocus();
    virtual int OnLButtonDblClk(UINT nFlags, LONG XPos, LONG YPos);
    virtual int OnLButtonDown(UINT nFlags, LONG XPos, LONG YPos);
    virtual int OnLButtonUp(UINT nFlags, LONG XPos, LONG YPos);
    virtual int OnMouseMove(UINT nFlags, LONG XPos, LONG YPos);

```
virtual int OnRButtonDblClk(UINT nFlags, LONG XPos, LONG
YPos);
virtual int OnRButtonDown(UINT nFlags, LONG XPos, LONG
YPos);
virtual int OnRButtonUp(UINT nFlags, LONG XPos, LONG YPos);
virtual int OnPaint(HDC hDC);
virtual int OnSetCursor(HWND hWnd, UINT nHitTest, UINT
message);
virtual int OnShowWindow(BOOL bShow, UINT nStatus);
virtual int OnTimer(UINT nIDEvent, TIMERPROC *tmPrc);
```

功能：由於訊息處理函數比較多，所以下面做統一的介紹。這裡所有的參數均不需要填寫，由Windows訊息機制來完成。

OnActivateApp函數處理程式被啟動或由活動狀態轉為非活動狀態的訊息。bactive表示視窗是否被啟動，dwThreadID是執行緒ID編號。

OnCreate函數處理視窗建立訊息。

OnSetFocus函數處理視窗獲得焦點訊息。

OnChar函數處理鍵盤訊息、nChar是虛擬鍵盤值。

OnClose函數處理視窗關閉訊息、hWnd是視窗控制碼。

OnCommand函數處理功能表、控制項訊息。

OnDestroy函數處理視窗銷毀訊息。

OnKeyDown函數處理鍵盤按鍵按下的訊息。

OnKeyUp函數處理鍵盤按鍵放開的訊息。

OnKillFocus函數處理視窗失去焦點的訊息。

OnLButtonDblClk函數處理滑鼠左鍵雙擊訊息，XPos和YPos分別代表滑鼠所處的座標的x和y座標。相應的函數有：

OnLButtonDown, OnLButtonUp, OnRButtonDblClk, OnRButtonDown, OnRButtonUp。它們分別處滑鼠左鍵按下、放開，滑鼠右鍵雙擊、按下、放開的訊息。

OnMouseMove函數處理滑鼠移動訊息。XPos和YPos分別代表滑鼠所處位置。

OnPaint函數處理視窗重繪訊息。

OnSetCursor函數處理螢幕上的滑鼠設置訊息。

OnShowWindow函數處理視窗顯示訊息。bShow代表視窗是否被顯示。

OnTimer函數處理計時器訊息。

3.4.2 GAFDDraw類別

這個類別如同之前所說的，是對應於DirectX的DirectDraw，主要是用來設定圖片，不論是背景或是圖片表層，都可以在此設定顯示模式，另一個重點則是顯示放記憶體的使用與釋放。

1. 函數：HRESULT CreateFullScreen(HWND hWnd, DWORD Width, DWORD Height);
   功能：設置寬高為指定值的全螢幕16bit模式，並且自動建立主表面和後臺表面的翻轉鏈。hWnd指定視窗的控制碼，Width指定表面寬度，Height指定表面高度。
2. 函數：HRESULT CreateWindowMode(HWND hWnd, DWORD Width, DWORD Height, BYTE bBackSurfacePos=GAFMEMVIDTHENSYS);
   功能：設置寬高為指定值的視窗模式，如果視窗客戶區小於指定大小，會自動調整視窗大小 自動建立主表面，並自動加上防止視窗之間遮擋出錯的裁剪器，自動建立後臺表面，可以指定後臺表面放的位置。hWnd指定視窗的控制碼，Width和Height指定表面的寬度和高度。bBackSurfacePos指定後臺緩衝的位置，如為GAFMEMVIDEOONLY，表示圖層表面只放在顯示卡記憶體；為GAFMEMSYSTEMONLY，表示圖層表面只放在系統內存；如果是GAFMEMVIDETHENSYS，也即預設選擇，則圖層表面擺放由DDRAW自動決定。
3. 函數：HRESULT Present(BOOL Vsync=TRUE, BOOL displayFPS=FALSE);
   功能：用來顯示表面，如果是全螢幕模式則自動翻轉主後表面，如果是視窗模式會自動Blt，預設值為TRUE。VSync指定是否等待顯示器垂直回掃週期。displayFPS指定是否顯示每秒幀速率，預設為FALSE。
4. 函數：HRESULT Restore();
   功能：發生主表面資料遺失的時候，自動恢復主表面。這個操作會由GAFDDraw自動呼叫自動完成。
5. 函數：BOOL is555();
   功能：檢查顯示卡的16位元顏色模式是否是555格式，如果顯示卡支持555格式的16位色，則返回TRUE。
6. 函數：BOOL isFullScreen();
   功能：檢查程式是否為全螢幕模式。如果是全螢幕模式，則返回TRUE。
7. 函數：LPDIRECTDRAW7 GetDD();
   功能：獲取原始的LPDIRECTDRAW7介面。
8. 函數：LPCGAFDDRAWSURFACE GetPrimary();

功能：獲取由GAFDDraw生成的主表面(前臺緩衝)。

9. 函數：LPCGAFDDRAWSURFACE GetBack();
   功能：獲取由GAFDDraw生成的後臺緩衝。

3.4.3 GAFDDrawSurface類別

這個類別跟上一個類別有很密切的關係，它的主要用途是設定圖層的載入、屬性設定等。

1. 函數：HRESULT CreateSurface(LPCGAFDDRAW lpGAFDD, int Width, int Height, BYTE MemoryType=GAFMEMVIDTHENSYS);
   功能：建立一個空的指定大小的圖層表面，可以指定表面放的位置。lpGAFDD指定GAFDDraw類別指標。Width和Height分別指定表面的寬度和高度。MemoryType指定表面放置的位置，如為GAFMEMVIDEOONLY，表示圖層表面只放在顯示卡記憶體；為GAFMEMSYSTEMONLY，表示圖層表面只放在系統記憶體；如果是GAFMEMVIDTHENSYS，也即預設選擇，則圖層表面擺放由DDRAW自動決定。
2. 函數：HRESULT CreateByJPEG(LPCGAFDDRAW lpGAFDD, LPCTSTR szFileName, BYTE MemoryType=GAFMEMVIDTHENSYS);
   功能：根據一個JPEG檔來自動建立表面。lpGAFDD指定GAFDDraw類別指標。SzFileName指定需要載入的圖檔檔名。MemoryType指定表面所在位置。用法同上。
3. 函數：HRESULT CreateByBitmap(LPCGAFDDRAW lpGAFDD, LPCTSTR szFileName, BYTE MemoryType-GAFMEMVIDTHENSYS);
   功能：根據一個BMP檔來自動建立表面。參數用法同上。
4. 函數：HRESULT Restore();
   功能：圖層資料遺失的時候自動呼叫Restore恢復圖層，但是目前還沒有自動恢復圖層內圖像的功能。
5. 函數：HRESULT BltFastTo(CGAFDDrawSurface *lpDDS, LONG lDestX, LONG lDestY, BOOL bTrans, LPRECT srcRect=NULL, BOOL bAutoClip=TRUE);
   功能：向lpDDS表面複製圖像。lpDDS是目標表面的指標。lDestX和lDestY分別指定目標位置的x和y座標。bTrans決定是否做透明色檢查。srcRect指定要複製的矩形區域指標，如為NULL則為複製整個圖層。bAutoClip指定是需要自動裁剪(比如要複製的區域超出目標圖層的部分會被自動裁掉)。
6. 函數：HRESULT BltTo(CGAFDDrawSurface *lpDDS, LONG lDestX, LONG lDestY, BOOL bTrans, LPRECT srcRect = NULL, BOOL

bAutoClip = TRUE);
功能：功能和參數用法同上。

7. 函數：HRESULT StretchBltTo(CGAFDDrawSurface *lpDDS, BOOL bTrans, LPRECT destRect = NULL, LPRECT srcRect = NULL, BOOL bAutoClip = TRUE);
功能：功能和參數用法大致同上，但可以指定目標圖層上接受圖像的區域大小。destRect指定複製到目標表面的矩形區域指標，如果為NULL則為複製到整個圖層。
8. 函數：HRESULT BltFast(CGTAFDDrawSurface *lpDDS, LONG lDestX, LONG lDestY, BOOL bTrans, LPRECT srcRect=NULL, BOOL bAutoClip=TRUE);
HRESULTBlt(CGTAFDDrawSurface *lpDDS, LONG lDestX, LONG lDestY, BOOL bTrans, LPRECT srcRect=NULL, BOOL bAutoClip=TRUE);
HRESULT StretchBlt(CGTAFDDrawSurface *lpDDS, BOOL bTrans, LPRECT destRect=NULL, LPRECT srcRect=NULL , BOOL bAutoClip=TRUE);
功能：功能與前面介紹的3個函數一樣，不過這裡是將lpDDS表面的圖像複製過來。
9. 函數：void SetClipRect(const LPRECT lprt=NULL);
功能：設置表面可以接受圖片位元傳送的區域。lprt指定裁剪區矩形，如果為NULL則指定整個表面。
10. 函數：BOOL ClipRect(LPRECT lprt);
功能：根據表面自身的裁剪區域來裁剪矩形區域。lprt指定該矩形區域。
11. 函數：BOOL ClipRectOffset(LPRECT lprt, LONG *plDestX, LONG *plDestY);
功能：根據表面自身的裁剪區域來裁剪位移(lDestX, lDestY)以後的lprt。
12. 函數：void GetSurfaceRect(LPRECT lprt);
功能:獲取整個表面的區域。lprt為呼叫後獲得的矩形區域的指標。
13. 函數：DWORD GetWidth();
DWORD GetHeight();
功能：獲取表面的寬度和高度。返回值分別為寬度和高度值。
14. 函數：LPDIRECTDRAWSURFACE7 GetSurface();
功能：獲取原始的LPDIRECTDRAWSURFACE7介面。
15. 函數：HRESULT SetColorKey(DWORD dwColorKey);

HRESULT SetColorKey();
功能：設置顏色鍵。第一個函數指定dwColorKey為顏色鍵。第二個函數則設置圖片左上角的像素為顏色鍵。

16. 函數：HRESULT Lock(LPRECT lprt=NULL);
HRESULT Unlock(LPRECT lprt=NULL);
功能：第一個函數Lock用於鎖定表面。lprt指定要鎖定的區域，如果為NULL則鎖定整個表面。第二個函數Unlock則是解鎖表面。
17. 函數：DWORD GetPitch();
功能：獲取鎖定以後的每行字節數。
18. 函數：BYTE* GetSurfacePointer();
功能：獲取鎖定以後的記憶體指標。
19. 函數：HRESULT Fill(DWORD FillColor, LPRECT lprt=NULL, BOOL bAutoClip=FALSE);
功能：用指定顏色填充矩形。FillColor指定要填充的顏色鍵。lprt指定要填充的矩形區域，如果為NULL則填充整個表面。bAutoClip指定是否要進行自動裁剪，預設為FALSE。
20. 函數：HRESULT SetFont(const LPCTSTR szFontName, const int nHeight, const int nWidth=0, const int nAttributes=FWNORMAL);
功能：設置字體。SzFontName指定字體名。nHeight指定字體高度。nWidth指定字體寬度，預設為0。nAttributes指定字體風格，預設為FWNORMAL。
21. 函數：HRESULT TextXY(int x, int y, DWORD col, LPCTSTR pString);
功能：在表面上顯示文字。x,y指定了文字在表面上的位置的座標。Col指定文字的顏色。pString指定文字內容。
22. 函數：HDC GetDC();
功能：獲取表面的DC。返回值為DC的控制碼。
23. 函數：HRESULT ReleaseDC();
功能：釋放表面DC。

3.4.4 GAFDInput類別

這個類別主要是對DirectX裡的DirectInput元件進行封裝，並且額外對立即方式的輸入增加了「按鍵放開」的支援。不過此類別並沒有提供對搖桿的支援。

1. 函數：HRESULT InitDInput(const HINSTANCE hInstance, const HWND hWnd);

功能：用來初始化DInput，HInstance和HWnd是主程式的視窗實體和視窗控制碼。

2. 函數：HRESULT Update(BOOL updateMouse=TRUE, BOOL updateKeyboard=TRUE);
   功能：更新DInput狀態，獲取新的資料，可以指定是否更新滑鼠或者鍵盤。
3. 函數：intGetKeyState(int nKeyIndex);
   int GetMouseKeyState(int nMouseKeyIndex);
   功能：讀取鍵盤按鍵狀態和滑鼠按鍵狀態。KeyIndex為按鈕的索引號，比如DIK_SPACE，DIK_ESCAPE，DIK_E等等。滑鼠按鈕索引為DIK_MOUSELEFT，DIK_MOUSERIGHT等。
   返回值：
   GAFDIKEY_NONE //當前按鍵未按下
   GAFDIKEY_RELEASE //按鍵恰好放開
   GAFDIKEY_PRESS //按鍵按下
4. 函數：void GetMousePos(LONG *px, LONG *py);
   void GetSysMousePos(LONG *px, LONG *py);
   功能：獲取DInput滑鼠位置和獲取系統預設滑鼠位置。DInput的滑鼠和系統的滑鼠是不同步的，因此使用者可以自己決定採用哪一個。讀取系統滑鼠速度比較慢；而DInput的滑鼠位置由於和系統滑鼠不同步，最好自己畫滑鼠遊標。另外，滑鼠位置是相對於螢幕而言的，如果要得到相對於視窗的資料，請自行利用ScreenToClient來操作。
5. 函數：void SetMouseAcceleration(BOOL ac);
   BOOL IsMouseAcceleartion();
   功能：設置和讀取GAFDInput滑鼠自動加速開關。當覺得DInput的滑鼠速度慢時，那就打開這個選項。
6. 函數:HRESULT SetMouseExculsive(BOOL ExclusiveMouseAccess);
   功能：設置是否採用滑鼠獨佔瀏覽。
7. 函數：void SetMouseLimit(LONG x1, LONG y1, LONG x2, LONG y2);
   void SetMouseLimit(const LPRECT lprt);
   功能:設置滑鼠運動範圍,可以使用左上角(x1,y1)和右下角(x2,y2)座標來指定範圍，也可以使用RECT來指定一個矩形區域。如果不指定，那麼GAFDInput將自動指定運動範圍為螢幕範圍。
8. 函數：BOOL hasKeyBoard();
   BOOL hasMouse();
   功能：查詢是否擁有滑鼠和鍵盤設備的不常用功能。

第四章　遊戲實作說明

4.1建立標題視窗的基礎

使用GAF來建立一個空視窗很簡單，只要在標頭檔中宣告一個類別繼承CGAFApp，如下列程式碼：

```
class CRPGGame : public CGAFApp
{
public:
   CRPGGame();
   virtual ~CRPGGame();
public:
   BOOL   InitInstance();      //程式初始化部分
   int    Run();            //主動式遊戲主迴圈
   int    ExitInstance();      //結束程式
protected:
   BOOL   InitDXMember();      //初始化DirectX成員
};
```

然後在相對的cpp檔中，於InitInstance()及InitDXMember()裡加入下面的程式碼：

```
BOOL CRPGGame::InitInstance()
{
   BindWndIcon(IDI_GAFGAME);          //設置視窗圖示
   BindWndWidth(640);BindWndHeight(480); //設置寬度高度
   //設置視窗類別名稱
   BindWndClassName(IDS_STR_WNDCLASS);
   BindWndTitle(IDS_STR_WNDTITLE);       //設置視窗標題
   BindWndStyle(WS_EX_APPWINDOW,WS_VISIBLE|WS_POPUP|WS_C
  APTION|WS_SYSMENU); //設置視窗風格
       TESTBOOL_MSGRET(CGAFApp::InitInstance(),"GAF程式建
立視窗出錯，程式終止!");       //繼承父類別函數建立視窗
   TESTBOOL_MSGRET(InitDXMember(),"GAF程式初始化DX成員出
錯，程式終止!");     //初始化DirectX成員
   return TRUE;
}
BOOL CRPGGame::InitDXMember()
{
```

```
    try
    {
        TESTHR_MSGTRW(GAFDDraw.CreateFullScreen(m_hWnd,640
    ,480),"GAFDDraw建立全螢幕發生錯誤!");
        TESTHR_MSGTRW(GAFDInput.InitDInput(m_hInstance,m_h
    Wnd),"GAFDInput初始化發生錯誤!");
    GAFDInput.SetMouseLimit(0,0,640,480);
    GAFDInput.SetMouseAcceleration(TRUE);
    }
    catch(...)
        return FALSE;
    return TRUE;
}
```

便可以輕易地建立一個640*480的全螢幕視窗。

視窗建立好之後，第一步就是先建立標題。在建立標題之前，必須先說一下UI(User Interface，使用者介面)元素。

UI是用來聯繫遊戲核心程式和玩家的橋樑，也就是提供給玩家操作遊戲功能的一系列元素。

UI的運用已經滲透到各種各樣的軟體之中。比如傳統的DOS，它的UI是一個C:\\>提示字元，使用者需要不斷輸入各種複雜生硬的指令來操作電腦，這是最原始的UI。後來出現了以GUI(Graphics User Interface，圖形化使用者介面)為核心的PowerMac和Microsoft Windows，這種直覺式、清晰、方便的介面直到今天仍然經久不衰。

Windows系統中的常用UI元素有：按鈕、功能表、複選視窗、輸入框、文字方塊、捲軸等等。在全螢幕的DirectX畫面下運行遊戲，這些標準的UI元素根本沒有辦法繪製。因此，遊戲中所使用的UI元素就必須要自己動手畫。

一個遊戲中第一個需要UI元素的地方就是標題畫面，最基本的標題畫面中必須有「進入遊戲」及「離開遊戲」兩個按鈕，下面即為一個簡單的標題畫面：

圖4.1　簡易標題畫面

從圖中可以看出來，左上角有個「New game」的按鈕，而右上角的則是「Exit game」按鈕。

為了處理標題畫面，我必須在剛才建立的CRPGGame類別裡加入一個成員函數ProcessMainTitle()。在這個函數裡我必須建立一個offscreen surface，將標題圖片load進去，並且定義兩塊矩形區域，對應到圖片中兩個按鈕的位置，之後只要加入下列程式碼就可以很容易地完成了我的第一個UI元素及標題畫面。

```
GAFDInput.Update();      //更新輸入資料
        //按下滑鼠左鍵
if(GAFDInput.GetMouseKeyState(DIK_MOUSELEFT)==GAFDIKEY_RELEASE)
{
    LONG lX,lY;
    GAFDInput.GetSysMousePos(&lX,&lY);//讀取滑鼠位置
    POINT Pt; Pt.x=lX; Pt.y=lY;
    AdjustMousePosition(&Pt);
    if(PtInRect(&NewGameRect,Pt))      //點擊NewGame進入場景狀態
    {
        GAFGameStatus=GAF_GAMESTATUS_INSCENE;
        return S_OK;
    }
```

```
        if(PtInRect(&ExitGameRect,Pt))     //點擊ExitGame退出
        {
            GAFGameStatus=GAF_GAMESTATUS_ ENDING;
            return S_OK;
        }
    }
```

4.2遊戲狀態的替換

在上面一節的最後程式碼的部份，可以看到以下兩行程式碼：

```
GAFGameStatus=GAF_GAMESTATUS_INSCENE;
GAFGameStatus=GAF_GAMESTATUS_ ENDING;
```

這個就是這節的重點了，也就是遊戲狀態的替換。在之前的2.3節中我已經稍微提過，遊戲中遊戲狀態的替換大多是以FSM來實作，因為這樣可以將不同遊戲狀態的程式碼分開處理，降低程式的複雜度。

在實作的部份，跟上面兩行程式碼相呼應的程式就在我們的遊戲主迴圈函數Run()裡面，如下：

```
while(GAFGameStatus!=GAF_GAMESTATUS_ENDING)
{
    //檢測到退出訊息
    if(ProcessMessage()==FALSE)
        GAFGameStatus=GAF_GAMESTATUS_ENDING;
    switch(GAFGameStatus)
    {
        case GAF_GAMESTATUS_MAINTITLE:     //標題畫面狀態
            if(FAILED(ProcessMainTitle()))
                GAFGameStatus=GAF_GAMESTATUS_ENDING;
            break;
        case GAF_GAMESTATUS_INSCENE:       //遊戲場景狀態
            if(FAILED(ProcessScene()))
                GAFGameStatus=GAF_GAMESTATUS_ENDING;
            break;
        case GAF_GAMESTATUS_ENDING:        //直接關閉窗口
            PostQuitMessage(0);
            GAFGameStatus=GAF_GAMESTATUS_ENDING;
            break;
        default:
            ALARMMSG("GAF程式檢測到未知的遊戲狀態，程式終止
```

```
");
                PostQuitMessage(0);
                break;
        }
    }
```

從上面可以看出，在GAF_GAMESTATUS_MAINTITLE狀態時，會執行ProcessMainTitle()，如果執行失敗，狀態就會變成GAF_GAMESTATUS_ENDING並結束遊戲；如果執行成功，則會依ProcessMainTitle()中的程式碼所示：使用者如果點選New game則狀態為GAF_GAMESTATUS_INSCENE，如果是點選了Exit game則狀態就變成GAF_GAMESTATUS_ENDING並回到Run()結束遊戲。

以上就是遊戲狀態替換的FSM實作，至於GAFGameStatus列舉結構如下：

```
enum GAMESTATUS{
        GAF_GAMESTATUS_NONE = 0,
        GAF_GAMESTATUS_START,
        GAF_GAMESTATUS_ENDING,
        GAF_GAMESTATUS_LOGO,
        GAF_GAMESTATUS_MAINTITLE,
        GAF_GAMESTATUS_INSCENE,
        GAF_GAMESTATUS_CREDIT
};
```

這是定義在GAF類別庫的gaf.h標頭檔當中，這裡看起來似乎有很多的狀態，但我實作時只取了主要的3個狀態。

4.3場景及地圖

一個RPG大致上來說就是以場景及地圖來組成。地圖便是指讓主角四處走動的背景圖片，而場景則是畫面中去掉地圖及主角的剩餘物品(包括NPC)皆可算在其內。因此我們可以知道一塊區域的地圖檔及場景檔是相對應的。

當然我們可以把每塊區域的地圖及場景設定都各自寫出自己的程式碼，但是這麼一來，一旦一個遊戲的區域數量龐大時(一般的市面上賣的RPG大概都有50個以上的區域)，程式碼的處理也就變得繁瑣無比，因此場景編輯器及地圖編輯器就變得不可或缺了。

幸運的是，GAF類別庫有附上它專屬的場景編輯器及地圖編輯器，底下就是它所預設的場景檔及地圖檔的格式：

- 場景檔格式：

檔案名：*.SCN
存放位置：和對應的MAP檔一起放在遊戲主程式的/Scene資料夾裏面
文件內容：

DWORD	'GSCN'	標識
DWORD	0x0106	版本號(當前為v1.6)
CHAR[16]	szName	場景名稱
CHAR[32]	szMapName	場景對應MAP檔案名
DWORD	Width	由對應的MAP檔設定
DWORD	Height	由對應的MAP檔設定
DWORD	HCellsWidth/16	水平格子數(方格大小16*16)
DWORD	VCellsHeight/16	垂直格子數
DWORD	ObjCount	道具個數(如果沒有道具則必須設定為0，下面的結構陣列可以省略，下同)

OBJStruct[ObjCount]結構陣列（道具個數至多16）

	WORD	X，Y格子座標（0..HCells-1，0..VCells-1）
	WORD	道具編號
	WORD	是否空箱子？ 1表示有東西，0表示空了

DWORD	ExitCount	出口個數（至多16）

ExitStruct[ExitCount]

	WORD X1，Y2，X2，Y2	出口觸發區域（像素座標）
	CHAR[32]	對應的場景檔
	WORD TX，TY	切換場景以後在目標場景出現的像素座標

DWORD	RoleCount	角色個數（至多64）

RoleStruct[RoleCount]

	WORD X，Y	站立的像素座標
	WORD Dir	方向
	CHAR[32]	對應角色檔(*.ROL)

- 地圖檔格式：

檔案名：*.MAP
存放位置：和對應的圖片檔一起放在遊戲主程式的/Scene資料夾

裏面
文件內容：

```
DWORD       'GMAP'      標識
DWORD       0x0101      版本號(當前為v1.1)
CHAR[32]    GraphFName  圖片檔案名
DWORD       Width       寬度
DWORD       Height      高度

WORD[]      MapInfo     地圖資訊，大小為
(Width/16)*(Height/16)
其中  BIT0      有無障礙，1表示有障礙
      BIT1～7   保留
      BIT8～15 高8bit表示層次資訊，共256層，由小到大沿螢
幕方向向外

DWORD       Layer0Count    ;第0層矩形的數目
Struct[Layer0Count]        ;矩形陣列，下同
{
    WORD    Left           ;像素單位
    WORD    Top
    WORD    Width
    WORD    Height
}

DWORD       Layer1Count    ;第1層矩形的數目
Struct[Layer1Count]
{
    WORD    Left
    WORD    Top
    WORD    Width
    WORD    Height
}
```

共256層，如果這個層次上沒有矩形則必須放一個DWORD 0表示沒有矩形

只要使用這兩種編輯器，設置相對應的文件檔，然後在程式碼中依照這個格式將檔案讀入，便可以輕鬆地完成多個遊戲區域。

4.4角色實作

角色方面，一般RPG可以概分為三類，就是主角、NPC和敵人(大魔王)。這三類角色的基本格式都差不多，都有名字、角色狀態、站立方向、體力、魔力、攻擊力、防守力等等的屬性，因此我們自然可以做一個角色編輯器來新增每個角色的資料。而GAF類別庫在這方面也幫我們準備好了，底下就是GAF的角色檔格式：

檔案名：*.ROL

存放位置：和對應的圖片檔一起放在遊戲主程式的/Role文件夾裏面

文件內容：

DWORD	'GROL'	標識
DWORD	0x0103	版本號（當前為v1.3）
CHAR[16]	szName	角色名稱
SHORT	Status	角色狀態
SHORT	Flag	角色類型
SHORT	Direction	站立方向（從正下方開始順時針0-7）
CHAR[32]	szScript	角色腳本文件（如果沒有腳本文件則設定為全0，下同）
CHAR[32]	szGraph	角色圖片檔
DWORD	GraphWidth	圖片檔寬度（由圖片檔設定）
DWORD	GraphHeight	圖片檔高度（由圖片檔設定）
DWORD	CellWidth	圖片分割單元寬度
DWORD	CellHeight	圖片分割單元高度
POINT	KeyPos	角色腳底座標（用來定位角色）
RECT	InsideRECT	用於檢測碰撞的內部矩形框
SHORT	MoveFrames	運動幀數（第0幀是站立靜止）
SHORT	MoveDelay	運動延時（毫秒）
SHORT	MoveSwapFrame	運動交換幀速率（一般為2或者3）
SHORT	MoveSpeed	運動速度（像素/步）
SHORT	MoveGrpahLine[8]	紀錄對應方向i的運動幀在圖像中的行數（行數從0開始計算）
SHORT	FightFrames	戰鬥幀數

```
SHORT       FightGrpahLine      戰鬥分解幀在圖像中的行數

LONG        HP;                 //體力
LONG        HP_MAX;             //體力上限
LONG        MP;                 //魔法
LONG        MP_MAX;             //魔法上限
LONG        AP;                 //攻擊力
LONG        DP;                 //防禦力
LONG        DEX;                //靈敏
```

有了這個編輯器後，就是要寫一個跟它相應的類別，這個類別首先當然是從ROL檔中把各項資料讀出來，接下來就是要根據角色類型來分開處理，我的角色類型結構如下：

```
enum ROLEFLAG{                      //角色類型標識
        ROLEFLAG_UNKNOWN=0,         //未知類型
        ROLEFLAG_LEADER,            //主控主角
        ROLEFLAG_NPC,               //普通對話NPC
        ROLEFLAG_BOSS,              //BOSS
};
```

大致上主角、NPC跟魔王都可以共用一些基本的函數，像是SetMapPos(int mx, int my)負責將角色圖片放置到螢幕上、RenderRole()負責將繪出角色圖片、SetDirection(int newdir)負責決定要取用角色圖片分鏡中，哪個方向的圖片、CheckMouseHit(LONG lX,LONG lY)負責確認滑鼠是否點擊到某塊區域。市面上有些遊戲會設定NPC及魔王會依AI自行四處走來走去，如果要這樣做的話，或許可以再增加些Walk()或AI()的函數，但因為我並沒有如此設定，所以便省略掉了。

接下來我們必須在角色類別中為主角多設置幾個它專用的函數，因為主角跟其他角色不同，必須由玩家來操控，所以我設置了以下兩個函數：

```
void CGRole::CheckInput()
{
    assert(Flag==ROLEFLAG_LEADER);
    assert(m_pScene);

    Status = ROLESTAT_STAND;
    LONG SX=XMPos,SY=YMPos,lX,lY;
```

```
        m_pScene->MapToScreen(SX,SY); //變換到sprite相對於屏
幕的座標

        //按下滑鼠右鍵走動
        if(GAFDInput.GetMouseKeyState(DIK_MOUSERIGHT)==GAFDIK
EY_PRESS)
        {
             GAFDInput.GetSysMousePos(&lX,&lY);
             AdjustMousePosition(&lX,&lY);
             SetDirection(DecideMoveDirection(SX,SY,lX,lY));
        LONG dx=SX-lX;
        LONG dy=SY-lY;
        if(dx*dx+dy*dy>16*16) Status = ROLESTAT_WALKING;
        }
    }

    void CGRole::AIThinking()
    {
        if(Flag==ROLEFLAG_LEADER) CheckInput(); //主角控制

        DWORD timer = GetTickCount();
        switch(Status)
        {
             case ROLESTAT_NONE:
             case ROLESTAT_STAND:
                   Status=ROLESTAT_STAND;
                   ticktimer=timer;
                   TempFrame = 0;
                   CurrentFrame = 0;
                   break;
             case ROLESTAT_WALKING:
                   if((LONG)(timer-ticktimer)<MoveDelay)
                        break;
                   ticktimer=timer;
                   //選擇一個合適的方向移動一步
                   RoleWalkOneStep();
                   break;
```

```
            default:
                    break;
        }
    }
```

從以上程式碼中可以看出來角色類型另外一個重要的結構，那就是角色狀態：

```
enum ROLESTATUS{ //角色狀態
    ROLESTAT_NONE = 0,
    ROLESTAT_STAND,
    ROLESTAT_WALKING,
};
```

這個列舉型態可以讓我們知道角色目前處於什麼狀態中，是站立的或者正在走動，狀態之間的轉移是以滑鼠按鍵來做為觸發，而不同的狀態會有不同的處理程式碼來負責，這個也是FSM的一種運用。

4.5劇情、腳本及道具

腳本文件檔可說是一個RPG的靈魂所在，我們會在腳本裡處理掉所有的對話，以及點選某物件(人物或物品)之後的反應，這也就形成了劇情以及道具的使用。

要讓人理解到底腳本文件檔長成什麼樣子，最快的方式就是直接看它的內容到底是什麼，底下是遊戲中一名鐵匠的腳本文件檔：

```
Talk("主角","你好，請問你是打鐵師父嗎？");
Talk("年輕鐵匠","是的・・請問你是・・？");
Talk("主角","我在河的下游遇到一位女孩，她跟我說明了河霸之事，我希望能夠幫你們除掉他，而我聽說你能給我一些幫助。");
Talk("年輕鐵匠","・・・你想一個人去對付他？");
Talk("主角","沒錯！");
Talk("年輕鐵匠","唉・・之前村子裏的年輕人也有聚集起來反抗過，但都被那惡霸一個人給打敗了，他的功夫非同小可，我雖感謝你的好意，但還是得勸你作罷。");
Talk("主角","別說喪氣話了！都還沒試過呢，總是要試試看才知道，希望你能給我一點幫助。");
Talk("年輕鐵匠","・・・既然你都這麼說了，好吧！這裏是我祖傳留下來的寶劍，反正我不會武功留著也沒用，就給了你吧！");
Talk("年輕鐵匠","但是要對付惡霸光有寶劍是不夠的，你可以從河的對岸往上走，那裡可以通到河的上游，在上游的盡頭處，我弟弟就在那邊，他那有祖傳的寶甲，你可以跟他說是我叫你去拿的。
```

```
希望你能幫我們打倒惡霸！");
Talk("主角","放心吧！包在我身上，我走了！");
Talk("年輕鐵匠","先等等，別那麼急，我還沒跟你說道具要怎麼使用呢。武器跟防具要用之前要先裝備，你用滑鼠在自己身上按一下左鍵便可以開啟道具欄，然後在裝備上面按兩下，就可以裝備了。");
Talk("年輕鐵匠","你可以看看畫面左下角的AP和DP，這是你的攻擊力和防禦力，穿上裝備後數值應該會增加。惡霸就在河的上游，去吧！祝福你！");
AddItemOnce("祖傳寶劍");
Return();
```

這是非常淺顯易懂的一個檔案，其內容大多都是以Talk()指令為主，Talk裡面會用一個","隔開兩個被「""」括起來的句子，前面的括號裡的是說話者的名字，而後面的括號裡則是他所說的話，由對話也就形成了劇情。

再看到檔案倒數第二行的AddItemOnce("祖傳寶劍");從字義上便可以看出是增加括號裡的道具一個，也就是在這行指令之後，主角的道具欄裡會出多一把「祖傳寶劍」。最後的Return()則是代表此腳本檔到此結束。以下是我所定義的所有腳本檔指令：

```
DEFINE_COMMAND("Talk",FuncTalk);              //說話
DEFINE_COMMAND("AddEAP",FuncAddEAP); //增加攻擊力
DEFINE_COMMAND("AddEDP",FuncAddEDP); //增加防禦力
DEFINE_COMMAND("AddHP",FuncAddHP);        //增加HP
DEFINE_COMMAND("AddMP",FuncAddMP);        //增加MP
DEFINE_COMMAND("AddItemOnce",FuncAddItemOnce); //增加一個道具
DEFINE_COMMAND("Combat",FuncCombat);      //進入戰鬥
DEFINE_COMMAND("GameEnd",FuncGameEnd);    //遊戲結束
```

至於DEFINE_COMMAND的定義如下：

```
#define DEFINE_COMMAND(command,fuction) if (m_Command==command) return fuction();
```

從上面可以看出，當程式從腳本檔中讀到Talk時，就會呼叫FuncTalk來處理Talk後面「( )」裡的東西，其他的依此類推，但是很明顯地有一個指令並沒有在這裡面，那就是Return()，這是因為只要讀到Return就會從腳本檔跳回，只要一個break;指令就夠了，所以並沒有特地為這個指令寫一個函數。

道具的使用方面，也可以從上面看出來，是直接用腳本檔去定義

的，舉上面出現過的「祖傳寶劍」為例，它的腳本檔內容如下：

```
AddEAP(30);
Return();
```

也就是使用了這個武器的話，就會呼叫它相對應的腳本檔，然後增加攻擊力30點，之後結束這個腳本檔。很容易明白的功能。

4.6戰鬥實作

在我的Demo當中，戰鬥是由跟魔王對話而引起的，也就是在魔王的腳本文件檔當中加入了Combat();的指令。當戰鬥開始時，畫面會從一般場景轉入戰鬥畫面，然後會出現一個戰鬥選單，經由選擇指令來攻擊或防守。

由於是戰鬥開始時會轉換畫面，所以在處理FuncCombat()時，會呼叫RunCombat()來繪製戰鬥場景，在戰鬥分出勝負後，就會從RunCombat()回到FuncCombat()，如果輸了，就在FuncCombat()中輸出Gameover之類的字樣，然後將遊戲狀態改成GAF_GAMESTATUS_MAINTITLE回到標題畫面重新來過；如果贏了，那當然就結束FuncCombat()回到腳本檔，繼續接下去的劇情。

戰鬥實作的重點部分就在RunCombat()之中，這個函數的結構流程如下：

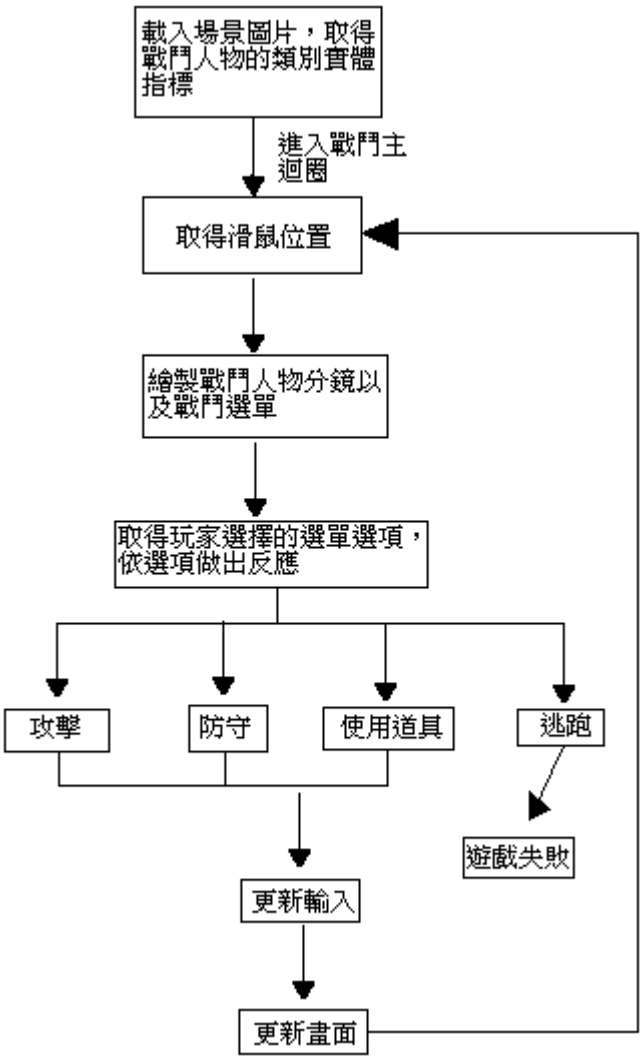


圖4.2 戰鬥流程圖

其中取得滑鼠位置、更新輸入、更新畫面的方法，就只要使用3.4節所介紹過的函數就可以輕鬆做到，如下列範例：

- 取得滑鼠位置：

```
POINT MousePos;
GAFDInput.GetSysMousePos(&MousePos.x, &MousePos.y);
AdjustMousePosition(&MousePos);
```

- 更新輸入：

```
GAFDInput.Update();
```

- 更新畫面：

```
GAFDDraw.Present(FALSE,TRUE);
```

戰鬥選單的做法也很簡單，就跟一般的對話框一樣顯示文字，然後讀取滑鼠的點選範圍，如果標題畫面的New game按鈕一樣，這也是一個UI元素。之後依點選位置將玩家的選擇存在一個變數中，回傳給RunCombat()然後用一個switch函數來處理就好了。

第五章　心得感想

5.1遭遇的困難及解決辦法

在專題實作的過程中，主要遭遇的困難有兩個，一個是在遊戲執行時，顯示出來的繁體字會變成亂碼；另一個則是必須配合GAF所給定的檔案格式來寫演算法。

關於字碼方面，GAF類別庫的確有附上一個可以改變字體的SetFont函數(詳見3.4.3節的第20個函數)，我試著使用過這個函數，可是除了改變字體大小外，並沒有得到任何的效用。這終究是字碼的問題，而不是字體的問題，所以我只好以簡體字來輸出。

但是將字幕以簡體字輸出也不是一件簡單的事，因為在筆記本(編輯腳本文件檔用)及Visual C++ 6.0的編輯視窗中，並不支援簡體字的顯示，所以在word中輸入繁體，再利用工具轉成簡體後，並沒有辦法直接把簡體字貼到這兩個編輯區裡去，因為貼過去時會因無法支援所以造成某些字碼的流失，在遊戲執行時也就無法正常地顯示文字。

因此我用了相當繁瑣的方式，我先開啟IE，連到某個留言板，然後將IE的編碼改成簡體，再將我從word裡得到的簡體字貼上去，這樣一來簡體字就可以正常顯示，然後送出留言，之後再將IE的編碼改成繁體，然後以繁體字碼去閱讀留言板上的簡體字碼，就會得到「以繁體字碼顯示的簡體字」，這些字雖然看起來是亂碼，但是並不會造成字碼的流失，所以將這些亂碼貼到腳本文件檔及程式碼中，就可以在遊戲執行時得到正常的簡體字幕。

不過以上始終只是個治標不治本的方法，而且經過嘗試發現，並不是在每台電腦上都可以正常的顯示出簡體字幕，所以我目前正在學習怎樣不用GAF類別庫嘗試輸出繁體中文。

至於另一個困難可以由我之前的報告內容看出，相信細心一點的人都會發現，我使用了各式GAF所附的編輯器，並且延用它們的格式，就像4.3節中的場景檔格式及地圖檔格式，以及4.4節的角色檔格式。

原因當然是為了節省寫編輯器所需的時間，在一般遊戲設計的過程中，編輯器的設計往往就花掉了總時數的3分之1，因為好的編輯器才能確保遊戲引擎完成之後，遊戲擴充及修改的便利性，也提供了劇本編撰的獨立性，讓企劃可以在不接觸太多程式碼的狀態下，得以直接操控遊戲的製作。

但也因為我使用了它所提供的格式，因此演算法的撰寫便會受限於此，舉一個最明顯的例子就是地圖檔中的WORD[] MapInfo，只要看到這個陣列，便可以了解在實作遊戲時，處理摭擋關係的演算法便只

有畫家演算法一個選擇了。

這並不是說畫家演算法很難，這意思是說，我必須先了解它所提供的格式背後真正的意義是什麼，然後才能選擇到底有什麼演算法可以用，像是WORD[] MapInfo，這是一個儲存層次資料的陣列，它將整張地圖以16*16的小方格畫分開來，然後將每個方格給予一個層次資訊，之後再按照畫家演算法，將背景及人物依層次高低，由低畫到高，這樣高層次的人物或背景如果有跟低層次的重疊到，那麼就會顯示出較晚畫者，也就是層次高者。

5.2系統評估與展望

GAF類別庫就遊戲設計來說，的確帶來很大的方便，但是從另一面來說，也帶來很大的限制，但是不論什麼工具大致上都是如此，越是方便的，限制往往也越大。

在我專題的實作中，我的確成功的做出一個最基本的遊戲引擎，但可惜的是，這個引擎的製作頂多就是讓人了解了遊戲設計的基本技巧與概念，而無法具有更大的改善空間，換句話說，就是沒有可能商品化。因為說到底，GAF類別庫只是一個讓人練習遊戲設計用的小型類別庫，並不像DirectX一般具有完整的函數庫及延伸性。

但是改善的空間也並非完全沒有，像是道具方面及特效方面，應該也是可以做出道具箱、會舞動的火焰、魔法攻擊、下雨天、戰鬥後獲得隨機道具之類的功能。劇情方面，應該有辦法做出分支的劇情，不像現在的Demo，只有一直線式的劇情，顯得太過單調。可是由於時間及人力問題，這些加強的功能都是「有機會再做」的事了。

5.3結論

我原本是跟另一個人同組做專題，但由於溝通協調不良，到最後變成各做各的，如同我前面1.1節裡說過，我們一開始本來是想做3D遊戲，但後來由於技術方面太過困難，而我又希望能做出某種「具有遊戲型態」的demo，所以便改成做2D的遊戲。

跟我同一組的同學便是在此時跟我產生分歧，因為與其做出遊戲，他更希望能在3D的圖型領域研究，所以我們便拆組，我做2D遊戲，他研究3D圖學，但是拆組時大約已是八月中到九月初的時候了，所以事實上我的這個專題做得很匆忙。

在找到GAF類別庫之前，我也試過用DirectX做出一個45度的拼接地圖，附上一些花草樹木，然後主角可以在地圖上走來走去，那時所使用的render演算法也是畫家演算法，不過跟目前這個demo的畫家演算法有所不同，那時是把每塊小地圖及主角、花草樹木等等的東西，

每個都當成一個獨立的層次，然後將所有層次依y座標的位置做氣泡排序，不過由於沒有遊戲設計的經驗，所以在一開始的結構規畫時就出了點問題，因此碰撞處理始終搞不好，於此同時剛好發現了GAF類別庫，於是便改成用GAF類別庫來實作，但是卻也只剩下一些時間了。

本來上學期時是很想將專題做得大一點，但很明顯地我高估了自己的能力，對於溝通協調的工作又沒做好，所以最後的demo才會做得如此基本，不過這也讓我學到了很多，像是在時間的估算上以及跟組員的溝通協調方面，都讓我得到了很寶貴的經驗，當然了，demo小雖小，在遊戲設計方面我也學到了很多。

參 考 資 料

1. 第二遊戲人生工作室，遊戲編程大師，數位人資訊股份有限公司，2003年6月20日
2. Bob Bates著、葉明璋、黃啟禎譯，創造遊戲的藝術與實務，第三波資訊股份有限公司，2002年10月
3. Neal Hallford with Jana Hallford著、黃啟禎譯，劍與線路的邂逅RPG設計師養成指南，2003年5月
4. 榮欽科技，新Visual C++遊戲設計，第三波資訊股份有限公司，2003年2月
5. 坂本千尋著、李于青譯，RPG角色扮演遊戲程式設計完整實例與經驗分享，博碩文化股份有限公司，2002年6月
6. Jim Adams著、張世敏譯，2D/3D RPG角色扮演遊戲程式設計—使用DirectX，博碩文化股份有限公司，2003年2月
7. Ivor Horton著、蔡明志譯，C++教學範本，碁峰資訊股份有限公司，2000年8月

附　錄　A

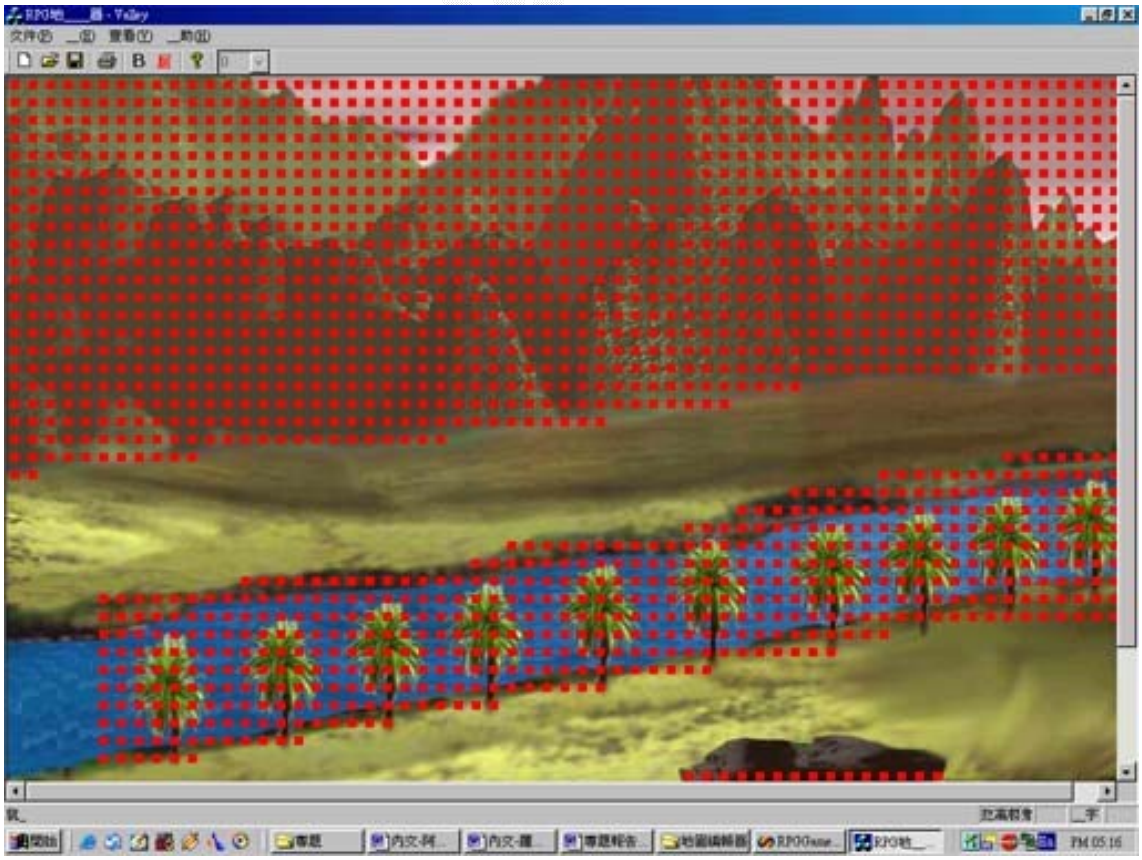
遊戲基本操作

滑鼠左鍵：點選/確定/使用道具
滑鼠右鍵：移動/取消
ESC鍵：退出遊戲回到標題畫面

滑鼠左鍵點到NPC身上會引起對話，在對話框出現時，點一下對話就會繼續；點到自己身上會開啟道具欄，在道具名稱上點兩下會使用道具。
滑鼠右鍵一直按住，人物便會往滑鼠所在方向移動；在道具欄開啟時點右鍵就會關閉道具欄。
在非對話及開啟道具欄狀態下，按ESC鍵會跳回標題畫面。

附　錄　B

地圖編輯器



圖B.1　地圖編輯器

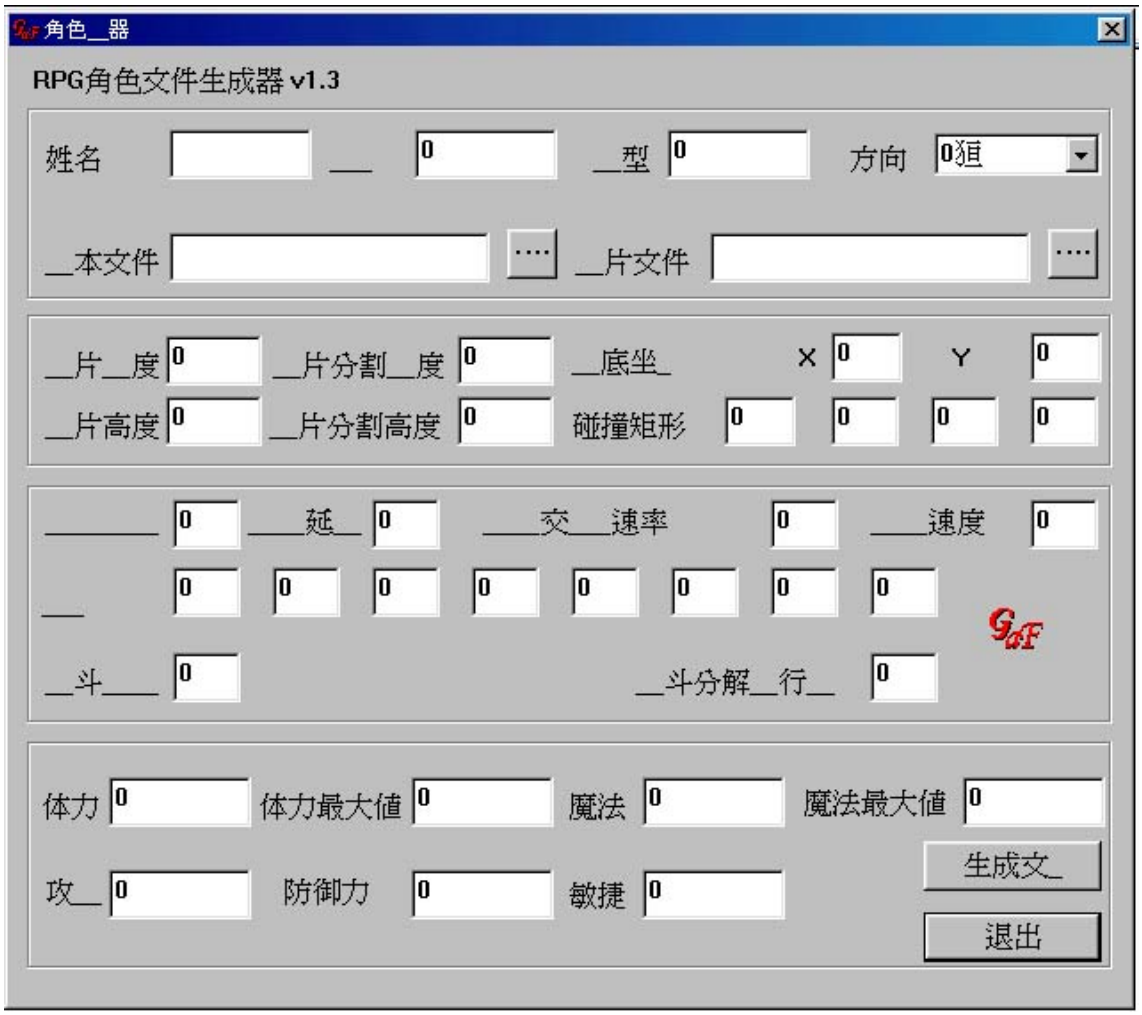
附　錄　C

場景編輯器

圖C.1　場景編輯器

附　錄　D

角色編輯器



圖D.1　角色編輯器

附　錄　E

道具編輯器

圖E.1　道具編輯器