報告題名：

三維空間第一人稱射擊遊戲實作

--使用 DirectX

作者：龍俊成、葉琮哲、吳昌隆、許克寬

系級：資訊系四年甲班

學號：D9146688、D9146968、D9146896、D9147161

開課老師：林國貴 老師

課程名稱：專題研究

開課系所：資訊工程學系

開課學年：94 學年度 第 1 學期

摘要

現在網路線上遊戲有愈來愈多的玩家加入，不過第一人稱射擊遊戲也還保有一定的市場，例如 id software 公司所出的 DOOM 3 和 Valve 公司所出的 Counter-Strike: Condition Zero，都還是市面上熱門的第一人稱射擊遊戲。

我們嘗試利用 Microsoft Visual C++和 DirectX 來撰寫遊戲程式，再利用 3D Studio MAX 來製作 3D 物件。在後面將介紹我們所使用的四個 DirectX 套件，包括有 DirectGraphics、DirectAudio、DirectInput 以及 DirectPlay，並且針對遊戲引擎，例如 3D 碰撞偵測，以及多人連線程式設計加以深入說明。

關鍵詞：第一人稱射擊遊戲(First Person Shooting)，DirectX

目錄

圖目錄

表目錄

Chapter 1　導論

1.1 動機

從高中開始，接觸到第一人稱射擊遊戲，那時市面上最風行的就是 Valve 公司所出的 Counter-Strike(以下簡稱 CS)。遊戲裡面注重的就是團隊合作，如何組織一個隊伍，如何帶領一個隊伍，這些能力都是玩這遊戲所必須具備的。隊員之間的默契也是相當重要的，在互相的掩護下擊敗所有的敵人拿下勝利。憑著對這遊戲的熱愛，當然就想要去瞭解其製作方法，例如如何產生 3D 物件、如何播放音效和如何建立多人連線等，對我們來說都是一個未知的領域，而如果未來要走遊戲設計這條路的話，這些背景知識是必須具備的。我們選擇了利用 DirectX 和 3DS MAX 強大的功能來建造屬於我們自己的第一人稱射擊遊戲。

1.2 目的

主要是製作出一款類似於市面上的第一人稱射擊遊戲，其詳細內容如下：

- 玩家可以使用鍵盤的 W、S、A 和 D 來控制遊戲人物的前進、後退、左橫移與右橫移，要前進的話必須一直按著 W 鍵，如放開遊戲人物也就停止前進，其他按鍵也是如此。
- 玩家點擊滑鼠左鍵可發射槍枝子彈。
- 判斷遊戲人物對於牆壁或者是其他玩家的碰撞。
- 多人連線功能。
- 利用這次的專題實驗，瞭解遊戲的製作方法，增加自己對於遊戲程式撰寫的能力。

1.3 第一人稱射擊遊戲簡介

什麼是第一人稱射擊遊戲(First Person Shooting)呢？簡單來說，它就是以主觀視點來進行的射擊遊戲。玩家可以從主觀視點觀察畫面中存在的物體，並且可對於所操縱之人物來進來射擊、行走等活動，進而消滅敵方人員。

在早期所接觸的射擊遊戲，是玩家控制一架飛機來進行射擊，視角是俯視的，稱為平面飛行射擊遊戲，其中最有名的就是日本 MOSS 公司於西元 1990 年所開發的 RAIDEN(雷電)，飛機可以前後左右的移動來對敵人進行攻擊，由於受到玩家們的喜愛，因此出了很多代，目前為止出到了第三代，並且開發許多版本，讓玩家在各種遊戲主機平台下都可以享受到這款遊戲。

到了西元 1992 年，id software 出了一款 Wolfenstein 3D(德軍總部 3D)，它可以說是 FPS 遊戲的始祖，FPS 遊戲也因此開始蓬勃發展；西元 1993 年，id software 又出了一款 FPS 遊戲——DOOM(毀滅戰士)，DOOM 最特別的是支援了連線功能，因此推出不久後，全世界的網路頻寬大部份都被 DOOM 所佔據了，許多公司的網路還因此癱瘓。由於 DOOM 大受歡迎，之後的 FPS 遊戲大都是以 DOOM 為基礎來開發，所以有“DOOM-Like”這個詞的產生。西元 2004 年 8 月，id software 發表了 DOOM 3，其遊戲畫面十分的精細，對於 3D 圖形的處理以及記憶體的需求非常龐大，不過卻沒降低玩家對這款遊戲的熱愛程度，在短短的五個月內就銷售了超過一百萬套。

我們所參照的遊戲是 CS，它跟 DOOM 不同的是 CS 注重團隊合作，而 DOOM 是屬於 Death Match，也就是獨自一個人單槍匹馬的奮戰，因此強調 Team Play 的 CS 裡隊員間可以利用內建的文字溝通，也可以自行輸入想要告訴隊友的話，互動性大大的增加。CS 之所以可以成為熱門 FPS 多人連線遊戲還有另一個原因，那就是遊戲裡面的角色扮演，它分為警察和歹徒，警察不是一般的警察，而是像維安小組的那種部隊，配備精良的武器，對抗同是擁有強大武器的歹徒，跟真實世界較為相近，模擬了時下反恐作戰，因此吸引了無數愛好者，網路上也越來越人自行組成一個團隊，來跟其他團隊對抗。電玩奧林匹克的 WCG(World Cyber Games)世界電玩大賽並把 CS 成為官方比賽項目。西元 2001 年 1 月至 10 月，台灣本島就銷售了十五萬套的 CS 遊戲，網路咖啡店的林立，也成為 CS 發展中的重要關鍵因素，走進各家網咖，看到的幾乎都是 FPS 射擊遊戲畫面，而當時坐在對面的人可能就是遊戲中正在追擊的目標。

Chapter 2　遊戲發展平台與工具

2.1　Windows 程式設計

Win32 API 之簡介

Win32 API (Application Programming Interface)，是微軟所推出的程式設計介面，類似作業系統中的 System Call；在作業系統中不允許應用程式直接使用硬體資源的，如果需要使用硬體資源的話，皆需要透過作業系統作為溝通的橋樑，而 API 就是系統所提供的函式庫，讓程式設計者所使用的。微軟則定義了 Win 32 API，作為微軟作業系統的標準。

在開發視窗程式時，需要對 Windows 程式基本的事件驅動有基本的認識，包括訊息的產生、獲得、分派、判斷、及處理，如此才能對於視窗程式的架構有明確的認知。表 2-1 是 Win32 視窗程式與一般 DOS 下 C 程式的比較。

表 2-1　Win32 視窗程式與一般 DOS 下 C 程式的比較

	Win32 視窗程式設計	**DOS 下 C 程式設計**
程式進入點	WinMain	Main
驅動架構	為建立在事件驅動的架構下，在事件發生時，系統將事件訊息放置訊息佇列中，並通知視窗程式處理	主要是靠中斷處理，提供一些中斷服務
顯示行為	需要跟系統呼叫一個視窗資源	DOS 本身已提供顯示平台

在某事件驅動後所產生的訊息(如：使用者使用鍵盤、滑鼠等訊息)，由作業系統從各週邊收到所有訊息後，會將訊息放置系統佇列中。而從 Windows 系統或其它程式所傳過來的訊息則放置程式佇列。每個視窗都會有一個函式負責處理訊息，如果視窗收到訊息，這個視窗函式必須判斷訊息，並決定處理方式。圖 2-1 是視窗程式生命週期。

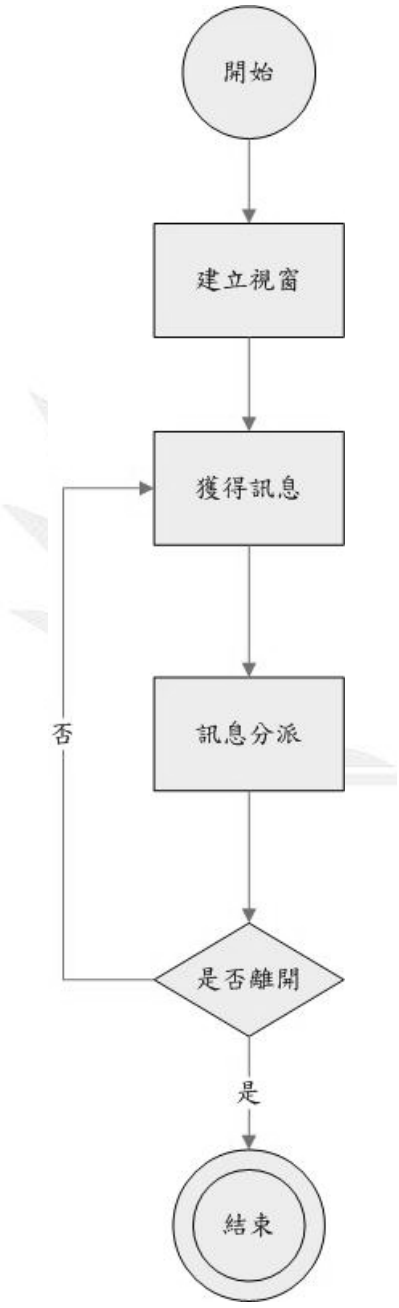


圖 2-1　視窗程式生命週期

2.2 DirectX

DirectX 為微軟公司所開發出來的一個軟體開發工具（SDK，Software Development Kit）。DirectX 的前身就是與 Windows95 差不多同時推出的 Game SDK。Game SDK 推出不久即改名為 DirectX，主要目的是在 Windows 平台下快速開發電腦遊戲與多媒體的一個軟體發展工具。

2.2.1 DirectX 的優點

在 Windows 作業系統下，所有程式必須透過應用程式介面（API，Application Programming Interface）來執行，Windows 繪圖用的 API 為圖形裝置介面（GDI，Graphic Device Interface）。由於早期的 Windows 作業系統的 GDI 處理速度很慢，繪圖的效果也有限。對於了解硬體的程式設計師，在 Windows 作業系統下撰寫和執行程式，反而無法發揮硬體的最大效能。但是 DirectX 卻克服了這個瓶頸。

DirectX 能快速處理圖形、聲音及多媒體等資訊，主要是因為 DirectX 跳過了 Windows 提供的 API，直接存取硬體，因此比 API 更能發揮硬體的效能，達到高度表現及快速的要求。另一方面，DirectX 使用驅動程式來達成對裝置的獨立性，驅動程式接受 DirectX 的指令，再轉換成硬體的指令去執行動作，對使用 DirectX 的程式設計師來說，不同的硬體並不會造成任何的差異。所以這也意味著程式設計師可以把心思都花在開發的專案上，不必考慮額外的事情大大提高了開發的效率。

2.2.2 DirectX 的缺點

DirectX 是由 COM（Component Object Model）所組成的，在 DirectX 中所有東西都是由 COM 的介面存取，也就是在寫程式時，需要先宣告物件（如頂點緩衝區物件等），使用完之後必須釋放不使用的物件，以避免佔據硬體的資源，才不會造成速度下降甚至當機的情況。由於 DirectX 必須透過 COM 介面存取，方式上較為複雜，穩定性也不如 OpenGL 好。然而最大的缺點就是只能在僅能在 Windows 作業系統上使用不能跨平台使用。

2.2.3 DirectX 的優勢

在 DirectX 之前，OpenGL 是電腦 3D 繪圖的極佳工具，使用 Silicon Graphics 公司研發且經最佳化的演算法，發展技術成熟且可跨平台應用。普遍認為 OpenGL 在電腦 3D 繪圖方面優於 DirectX，可惜由於 ARB（Architecture Review Board）維護標準的 OpenGL 使得其發展遲緩。從 1992 年出現 1.0 版以來，目前只推出到 2.0 版，相對的 DirectX 則因為微軟公司全力支持，1995 年推出後，目前已發展到 9.0c 版本，繪圖方面的效果與速度上已與 OpenGL 相差不多甚至可能已經超越 OpenGL。不過一般仍認為專業的電腦 3D 繪圖還是以 OpenGL 較適合，DirectX 則以多媒體與電腦遊戲等消費性市場為主要發展目標。除了電腦繪圖之外，DirectX 也包含了聲音，網路，輸入設備等功能，涵蓋層面均較 OpenGL 廣泛。再加上現在硬體廠商也都有與微軟合作開發 DirectX，透過硬體支援讓 DirectX 的功能更是越來越強大因此目前可以說 DirectX 是處於絕對優勢。表 2-2 是 Direct 和 OpenGL 的一些比較。

表 2-2　DirectX 和 OpenGL 比較

工具 / 項目	**DirectX**	**OpenGL**
更新速度	較快	較慢
支援功能	3D 繪圖，網路、音效、動畫、輸入	著重 3D 繪圖
開發過程	微軟強力支持，並與顯示卡廠商合作開發新技術	受限於維護標準，且較少廠商支持
現今採用的遊戲廠商	佔絕大多數	較少
可否跨平台	不可	可以

2.2.4 DirectX 的分類

DirectX 是為了電腦遊戲與多媒體等消費性娛樂市場為目標發展的，所以幾乎電腦遊戲所牽涉到的層面，DirectX 都有包含到，如電腦 3D 繪圖(2D 是由 3D 來模擬)、聲音、網路、動畫、多媒體，以及滑鼠、鍵盤、搖桿等遊戲輸入裝置，都在 DirectX 的範圍之內。隨著版本的更新所以內部成員可能都有所不同。由於我們是使用 DirectX 8.1 所以以下介紹都是針對 DirectX 8.1 解說的。DirectX 是依據功能來區分內部成員，如表 2-3 所示。

表 2-3 DirectX 8.1 的內部成員

名稱	***功能***
DirectGraphics	3D 繪圖
DirectAudio	音樂、聲音
DirectInput	鍵盤、滑鼠、搖桿
DirectPlay	網路
DirectShow	影片與多媒體
DirectAnimation	整合動畫

在我們專題使用的 CORE 中，有碰觸到的只有 DirectGraphics、DirectAudio、DirectInput、DirectPlay 等四部份，因此我們只針對這四項來做介紹。

2.2.4.1 DirectGraphics

簡單來說這部份負責將影像呈現在螢幕上，其中的轉換都是在這裡面完成。DirectX 7 版之前，電腦繪圖的部分分為 DirectDraw 的 2D 繪圖與 Direct3D 的 3D 繪圖，在 DirectX 8 之後，則把繪圖部分的 DirectDraw 與 Direct3D 合併為 DirectGraphics，事實上是已經取消 DirectDraw 的介面，若需 2D 的繪圖，也是以 3D 來模擬 2D。

2.2.4.1.1 內部元件

DirectX 8.1 版中 DirectGraphics 的元件如表 2-4 所示。

表 2-4 DirectGraphics 的元件

IDirect3D8	IDirect3DSurface8
IDirect3DDevice8	IDirect3DVolume8
IDirect3DIndexBuffer8	IDirect3DVolumeTexture8
IDirect3DVertexBuffer8	IDirect3DResource8
IDirect3DbaseTexture8	IDirect3DSwapChain8
IDirect3DTexture8	

其中最重要且常用到的就是 IDirect3D8 和 IDirect3DDevice8 在我們使用 Graphics 核心中也是主要利用這兩個。下面分別介紹它們之功能：

(1) IDirect3D8：取得圖形硬體方面的資訊，並建立環境的介面。

(2) IDirect3DDevice8：取得或設定裝置的狀態、描繪原型、開啟和關閉場景、光源與材料的設定、資料流、表面紋理的設定、建立緩衝區，以及定義投影矩陣、世界轉換矩陣、視點轉換矩陣等等，是 DirectGraphics 中最多函式的元件。

2.2.4.1.2 建立模型

對於較簡單的立體圖形，DirectX 有內建函式可以使用以方便建立模型，諸如球、圓柱、方塊、圓環等。至於其它更為複雜不規則的物體，不容易計算各個頂點空間中的相對位置，則可以在其他的 3D 軟體中先建立好，例如 3D Studio Max 等，產生副檔名為.x 的檔案。DirectX 讀取之後就可轉換為適當的頂點座標，繪製在螢幕上，此種模式稱為間接模式（Retained Mode，RM），我們便是採用這種方式來建立人物和地圖。這種方式可節省建立模型的時間，但執行效率上不如直接輸入頂點資訊的立即模式（Immediate Mode，IM）。

2.2.4.2 DirectAudio

簡單來說，這裡負責處理的就是平時在遊戲中所聽到的背景音樂或是音效。DirectAudio 是由 DirectSound 和 DirectMusic 這兩個元件所構成。DirectSound 是播放數位式聲音的主要元件。DirectMusic 則是負責處理所有的歌曲格式包含 midi、DirectMusic 的原始格式(*.SGT)和 WAVE 檔，並將它們傳給 DirectSound 來播放。

2.2.4.2.1 注意事項

在使用 DirectAudio 來播放音樂或音效時應該注意三點，不然可能會沒有聲音，或者聽起來怪怪的。

(1) 設定合作的等級：是設定 IDirectSound8 物件的合作等級，這決定與其它應用程式共用音效卡資源的方式。假如沒有設定會導致完全沒聲音。

(2) 設定正確的聲音頻率：假如要播放的音樂是以 44100Hz 錄製，但是卻設定了較低的頻率播放，那可能會讓聲音聽起來失真。

(3) 設定正確的取樣位元數：通常是以 8 或 16 bit 為主，假如設定超過音效卡支援的 bit 數，那也會播不出聲音來。

2.2.4.3 DirectInput

DirectInput 是用來處理所有輸入裝置的相關問題，如滑鼠、鍵盤以及搖桿等，在 Visual C++中也有滑鼠與鍵盤的函式，而 DirectInput 較特殊的就是對於搖桿的支援，尤其是在搖桿具有力回饋的功能時，更能加強在電腦遊戲時的娛樂效果。

2.2.4.3.1 內部元件

在 DirectInput 中包含的元件如表 2-5 所示。

表 2-5　DirectInput 的元件

名稱	功能
IDirectInput8	建立並恢復輸入裝置的狀態，並將 DirectInpu 元件初始化產生輸入設備的介面。
IDirectInputDevice8	取得或釋放輸入裝置控制權,讀取輸入裝置的資訊，建立力回饋特效等等。
IDirectInputEffect	取得特效種類,開啟或停止特效等。主要用於搖桿和某些特定滑鼠上

2.2.4.3.2　設定輸入裝置

(1) 取的設備之 GUID：每個設備都有自己的 GUID(Global Unique Identification)。譬如以鍵盤來說系統就會把它定義成 GUID_SysKeyboard。

(2) 建立 Device COM 物件：因為 DirectX 是由 COM 介面來存取。

(3) 設定資料的格式：每個設備都有自己定義的格式，譬如滑鼠滾軸、按鍵等等，在 DirectX 裡面有定義一些基本的設定可以直接套用。

(4) 設定合作等級：設定輸入裝置可否被其它應用程式所共用。

(5) 取得設備後，就可以開始讀取定義的裝置所輸入的資料了。

2.2.4.4　DirectPlay

DirectPlay 是用來處理網路連線方面的問題的解決方案。不論是使用 TCP 或者 UDP 來建立連線，還是傳輸遊戲資料、即時語音傳送都是使用 DirectPlay 的元件來完成。在 DirectX 8.0 的時候，微軟已經簡化了一些瑣碎的過程，只留下四個主要的 COM 元件分別是 IDirectPlay8Client、IDirectPlay8Server、IDirectPlay8Peer、IDirectPlay8Address。它們各自的主要功能如表 2-6 所示。

表 2-6　DirectPlay 內部元件

名稱	功能
IDirectPlay8Client	這是 Client 端的網路元件，用於建立 Client 端以及連結 Server 端。
IDirectPlay8Server	這是 Server 端的網路元件，用於建立 Server 端以及連結 Client 端。
IDirectPlay8Peer	這是對等網路(Peer To Peer)元件，可連結其它的用戶端
IDirectPlay8Address	這裡面包含了建立網路位址的元件，用於連上遠端的網路系統。

2.2.4.4.1　建立連線的方法(以 Client – Server 架構為例)

(1) 產生應用程式專屬的 GUID：只有具有相同 GUID 的程式才能互相連結，確保不會連結到不相干的程式。因此 Client 端和 Server 端要使用相同 GUID。

(2) 初始化網路元件以及位址元件：接下來就是建立網路元件和位址元件，譬如要建立 Client 端就要採用 IDirectPlay8Client 元件裡面定義的函式來建立。在 DirectX 8.1 版裡面，不論是 Client、Server 或者網路位址的建立，都已經統一使用 CoCreateInstance 這個函式，與之前各自分開函式有所不同。

(3) 取得連線設備的 GUID、遠端位址還有 Local 端使用的 Port

(4) 建立連線。

(5) 接下來連線程式之間的訊息傳遞，只要兩邊訊息格式定義一樣即可。

Chapter 3　遊戲架構與流程

遊戲是一種即時軟體，也就是對於時間有嚴格限制的軟體，遊戲中處處與時間相關，必須在有限的硬體資源下，模擬整個虛擬世界，並與時間取得平衡。

3.1　遊戲迴圈

遊戲的主迴圈是由三個同時執行的任務所組成的。第一，更新，電腦人工智慧等會改變遊戲世界的事件必須不斷的更新，例如當一名玩家發射了一發子彈，就必須不斷的更新子彈的位置。第二，輸入，遊戲必須能與玩家互動，處理玩家的輸入。第三，輸出結果，可能是畫面、聲音、或是搖桿的震動等。玩家的輸入就像是由外部手動更新程式，電腦人工智慧則像是內部自動更新程式。所以整個遊戲迴圈其實可以分成兩大部分，「更新區段」和「呈現區段」。

要確保「更新區段」和「呈現區段」能同時執行有幾種作法：

一、 將兩個區段實作在一個主迴圈內：

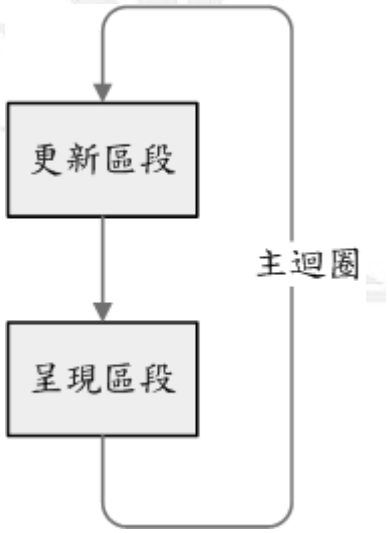


圖 3-1　單一迴圈流程圖

圖 3-1 是單一迴圈流程圖，此種做法能確保每次執行更新區段後必定執行一次呈現區段，不過此種作法卻存在著缺點，我們通常以 FPS(frame per second)，也就是每秒畫面的更新頻率來決定一個遊戲是否流暢。一般來說，遊戲在資源允許下必須多執行呈現區段，這樣才能得到最高的 FPS，過度的執行更新區段只是

浪費資源而已。但此種做法也使得遊戲和硬體效能的關連性變得太緊密，當今天換了一部效能較好的電腦，可能不只畫面變流暢而已，連帶使遊戲的速度也加快，而這並不是預期的結果。

二、 使用雙執行緒：

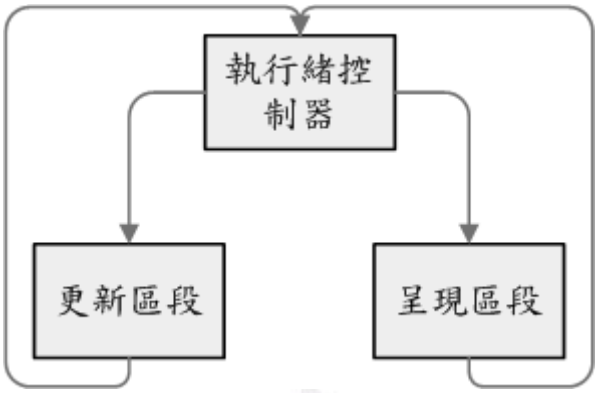


圖 3-2　雙執行緒流程圖

圖 3-2 是雙執行緒流程圖，此種做法藉由呼叫執行緒來控制區段的執行次數，能有效的做到資源分配，缺點是需要準確的時間控制及硬體的支援。

三、 單一迴圈加上計時器：

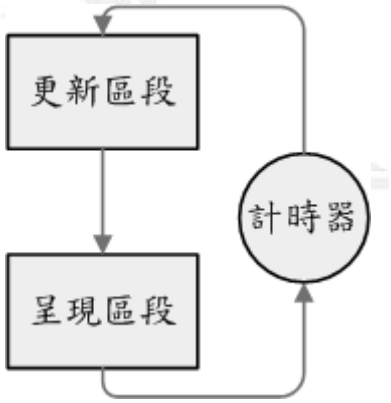


圖 3-3　單一迴圈加計時器流程圖

圖 3-3 是單一迴圈加計時器流程圖，這是我們採用的作法，在迴圈執行時候記錄時間，判斷是否需要呼叫更新器，舉例來說，如果想以每秒十次的頻率更新遊戲，只要每經過一百毫秒執行一次更新區段即可。

3.2 更新區段

更新區段是使整個虛擬世界運作的核心，包含了「玩家更新區」、「世界更新區」，世界更新區還包括非玩家角色(NPC)的更新。

玩家更新區

專門處理玩家動作的區塊，分成「玩家輸入」、「玩家限制」與「玩家更新」等。

(1) 「玩家輸入」必須能讀取玩家的輸入訊號，並轉換成對遊戲有意義的資訊，例如「同時按下鍵盤 W 鍵及滑鼠左鍵」，轉換成「玩家試圖向左邊移動並使用武器」。

(2) 「玩家限制」是進一步處理「玩家輸入」產生的資訊，並進行判斷玩家要求是否可以允許，依前例則需考慮左邊是否有障礙物阻擋，與手中武器是否有子彈等。

(3) 「玩家更新」即是考慮過限制後，將正確的結果反映到遊戲，依前例左邊如果沒有障礙物，則啟動移動動畫，並改變玩家位置。

世界更新區

世界物件可分為靜態物件與動態物件，靜態物件像是不會動的牆壁、樹木、石頭等，在遊戲中扮演了限制玩家的角色，並沒有特別的行為，故不需要特別更新。而動態物件像是會噴水的噴泉，裊裊的炊煙等，則須不斷隨著時間更新，更複雜的動態物件像 NPC，還須加入人工智慧(AI)系統，需要龐大的時間成本。在某些大型遊戲世界中，世界更新模組會事先過濾要更新的物品，以提高遊戲的效率。

整個更新區段如下：

玩家更新區

　　玩家輸入

　　玩家限制

　　玩家更新

世界更新區

　　靜態物件更新

　　　　靜態物件過濾器

　　　　更新靜態物件

　　動態物件更新

　　　　動態物件過濾器

　　　　更新動態物件

結束

3.3 呈現區段

此區是呈現整個遊戲聲光效果的地方，由於 DirectX 在繪製圖形時，須呼叫 BeginScene( )和 EndScene( )函式，所以集中在一個區塊對資源會有比較好的管控。

```
BeginScene()
    //在此處繪置圖形
EndScene()
```

Chapter 4　遊戲引擎

4.1　碰撞偵測

在虛擬的世界中，真實世界中理所當然的事都變成了複雜的事，人無法穿越牆壁，人可以踩在地板上面，這些平凡無奇的事，背後其實隱藏著大量的數學計算。

4.1.1　角色與世界間的碰撞

角色與世界的碰撞是用在人物與牆壁和地板的互動，在這裡用到了「光線與平面的交集測試」，也就是一直線與一平面之交點。圖 4-1 是光線與平面的交集測試圖。

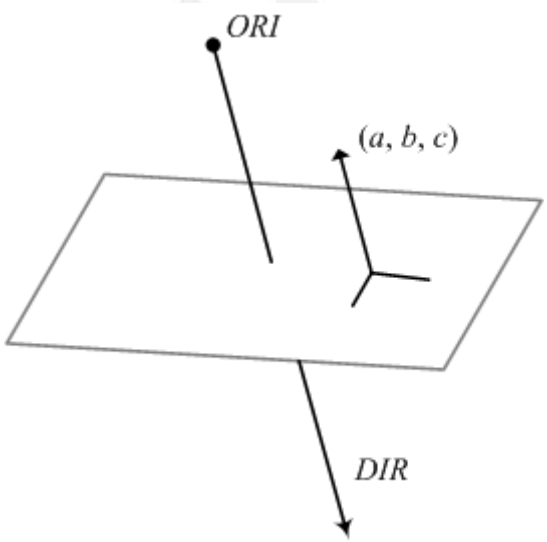


圖 4-1　光線與平面的交集測試圖

光線方程式： $X = ORI.x + DIR.x \times T$

$Y = ORI.y + DIR.y \times T$

$Z = ORI.z + DIR.z \times T$

ORI = origin (起點)

DIR = direction(方向)

平面方程式： $aX + bY + cZ + d = 0$

將直線方程式代入平面方程式，求 T，即可找出交點。

實作的程式碼：

DirectX 提供 D3DXIntersect 函式，用來做光線與網格的交點測試

D3DXIntersect

Determines if a ray intersects with a mesh.

```
HRESULT D3DXIntersect(
    LPD3DXBASEMESH pMesh,
    CONST D3DXVECTOR3* pRayPos,
    CONST D3DXVECTOR3* pRayDir,
    BOOL* pHit,
    DWORD* pFaceIndex,
    FLOAT* pU,
    FLOAT* pV,
    FLOAT* pDist,
    LPD3DXBUFFER* ppAllHits,
    DWORD* pCountOfHits
);
```

其中幾個重要的參數是 pMesh：所要測試的網格，在這裡放入地圖網格。pRayPos：光線的射出點座標。pRayDir：光線的方向向量。pHit：回傳的結果，傳回 TRUE 表示光線與網格有交點，否則回傳 FALSE。pDist：回傳的結果，傳回光線射出點與網隔間的距離。

我們只要設定一個角色可與牆壁接近的最短距離 D，當 pHit 傳回 TRUE 後，再將 pDist 與 D 做比較，就可以做出角色不會穿透牆壁的效果。

4.1.2 角色與角色間的碰撞

我們設定角色間的是不能互相穿過對方身體的，我們需要偵測物件間的碰撞，每個物件都是個不規則的多邊型，之前的光線與平面交集測試並不適合在這

裡使用，故我們使用了一種簡單的技術「球體與球體的交集測試」，我們先假定每個角色是個 3D 球體的多邊型，再套用兩點距離方程式，圖 4-2 是球體與球體的交集測試圖。

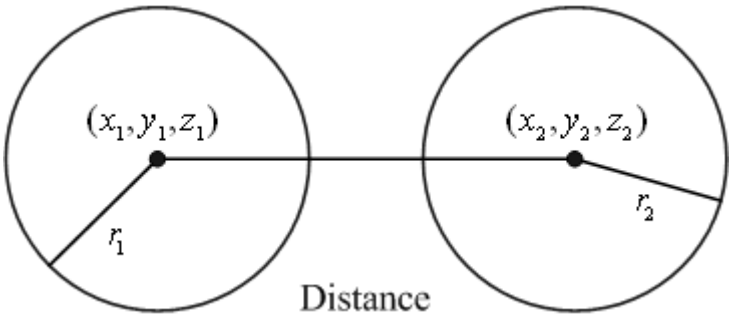


圖 4-2　球體與球體的交集測試圖

球 1 的中心 (x_1, y_1, z_1) 、半徑 r_1

球 2 的中心 (x_2, y_2, z_2) 、半徑 r_2

$(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2 = \text{Distance}^2$

如果 Distance 小於 $r_1 + r_2$ 就表示兩個球碰撞了。

這個方法的優點是所須計算成本很低，不過卻存在多缺點，像是長寬比例差距大的物件就不能精確的表示其邊界，另外球體半徑的設定其實也是個困難點，目前我們的作法是人工設定其半徑值，如果要改進可以使用程式計算離中心點最遠的頂點距離來當半徑。圖 4-3 是球體與球體交集測試模擬圖。

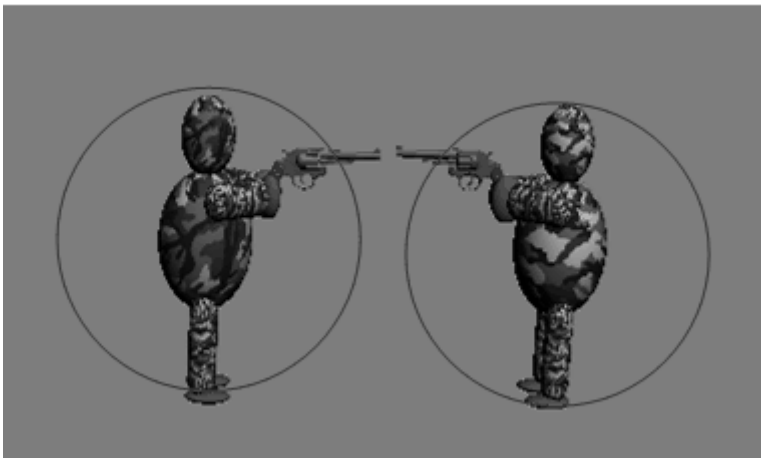


圖 4-3　球體與球體交集測試模擬圖

實作的程式碼：

```
BOOL CPlayer::CheckSphereIntersect(
float XCenter1, float YCenter1, float ZCenter1, float Radius1,
float XCenter2, float YCenter2, float ZCenter2, float Radius2)
{
    float XDiff, YDiff, ZDiff, Distance;

    XDiff = (float)fabs(XCenter1 - XCenter2);   // fabs() 取絕對值
    YDiff = (float)fabs(YCenter1 - YCenter2);
    ZDiff = (float)fabs(ZCenter1 - ZCenter2);

    Distance = (float)sqrt(XDiff * XDiff + YDiff * YDiff + ZDiff * ZDiff);

    if (Distance <= Radius1 + Radius2)
        return TRUE;
    return FALSE;
}
```

4.1.3 子彈命中判定

子彈命中判定依子彈的種類可分為兩種，第一種是「看的見的子彈」，這時可用前述的「球體與球體的碰撞測試」，只要一邊計算子彈的移動軌跡，一邊進行碰撞測試即可。第二種是「看不見的子彈」，則使用「光線與球體的交集測試」。圖 4-4 是光線與球體的交集測試圖。

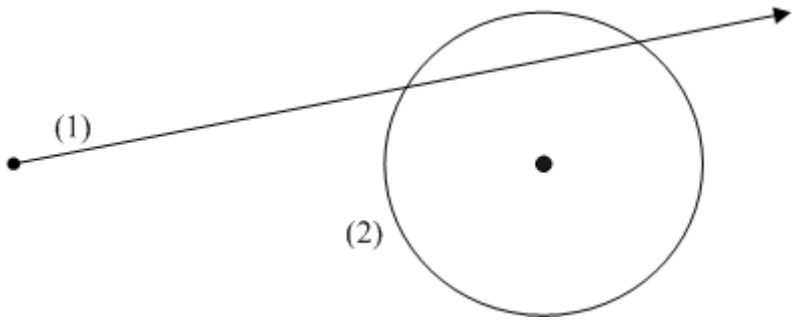


圖 4-4　光線與球體的交集測試圖

直線(1)$x = a + tu$ 與球(2) $x^2 = r^2$ 之交點

交點之位置向量同時滿足方程式(1)(2)，因此同時代入即可得到 t 的條件：

$(a+tu)^2 = r^2$

$\Rightarrow u^2t^2 + 2aut + a^2 - r^2 = 0$

$a = u^2, b = 2au, c = a^2 - r^2$　代入　$(t_1, t_2) = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

$\Rightarrow (t_1, t_2) = \dfrac{-au \pm \sqrt{a^2u^2 - u^2(a^2 - r^2)}}{u^2}$

若根號內之數 >0 ，則得到兩個實的交點，$x = a + t_1u$，$x = a + t_2u$ ，此直線為割線。

若根號內之數 $=0$ ，則得到一個實的交點，此直線為切線。

若根號內之數 <0 ，則沒有實交點，此直線不與球相交，或稱為外線。

當子彈軌跡為割線時，則代表子彈穿透過角色，即判斷為命中。

實作的程式碼：

我們可以用 DirectX 提供的 D3DXSphereBoundProb 函式來做測試。

D3DXSphereBoundProbe

Determines if a ray intersects the volume of a sphere.

```
BOOL D3DXSphereBoundProbe(
  CONST D3DXVECTOR3* pCenter,
  FLOAT              Radius,
  CONST D3DXVECTOR3* pRayPosition,
  CONST D3DXVECTOR3* pRayDirection
);
```

在 pCenter 放入圓心座標，Radius 放入圓的半徑，再將 pRayPosition 與 pRayDirection 各放入子彈的起點和方向向量，當函式回傳 TRUE 時，則表示有交點產生，FALSE 則沒有交點。

Chapter 5　武器系統與特效

在第一人稱射擊遊戲裡，最重要的莫過於武器了。武器是玩家求生的工具，也是遊戲中聲光效果的主要來源，可以說是第一人稱射擊遊戲的靈魂。

在一般的第一人稱射擊遊戲中，玩家都會擁有一把以上的武器，每把武器都會有不同外觀、子彈效果、音效、射程和破壞力等屬性，因此設計一個武器系統便是首要的工作。圖 5-1 是 CGun 類別圖。

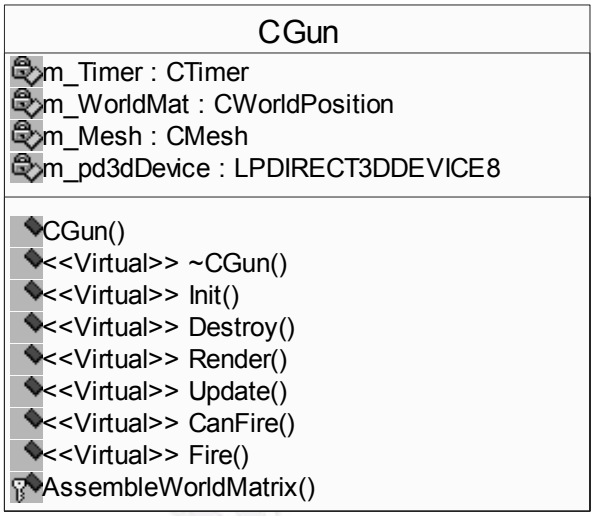


圖 5-1　CGun 類別圖

在這裡我們使用多型(polymorphism)來設計武器系統的架構，CGun 是所有武器的基礎類別，所有武器都繼承 CGun，CGun 包含其他武器類型都有的方法，Init 和 Destroy 負責武器的初始化和程式中止時所需要的記憶體釋放，Render 和 Update 負責武器的繪製和更新狀態，CanFire 和 Fire 則是判斷此時是否能發射和發射子彈的處理，這些方法都是虛擬函式，所有不同的武器類型都必須實做這些方法，好在架構中使用。AssembleWorldMatrix 在這裡是個被保護的方法，功用是武器在世界中的定位，我們將在下一節提到。

CGun 也包含所有武器的共同資料元素，包含計時器(m_Timer)、位置(m_WorldMat)、武器模型(CMesh)和裝置指標(m_pd3dDevice)。

5.1 武器定位與反彈效果

第一人稱射擊遊戲裡的武器永遠都出現在畫面裡，跟著角色的視點移動，我們需要做許多的數學運算來保持武器呈現在畫面中。以下是武器定位程式碼：

```
void CGun::AssembleWorldMatrix(cCamera &Camera, D3DXVECTOR3
vTranslation, float XRot, float YRot, float ZRot)
{
    D3DXMATRIX matTrans, matScale, matRot, matResult;
    D3DXMATRIX *matView = Camera.GetMatrix();      //取得檢視矩陣
    D3DXMATRIX matViewInverse;
    float fDet;
    //反轉檢視矩陣
    D3DXMatrixInverse(&matViewInverse, &fDet, matView);
    //預設平移量順序: X, Y, Z
    D3DXMatrixTranslation(&matTrans,
          2.5f + vTranslation.x,
        -3.0f + vTranslation.y,
        6.0f + vTranslation.z);
    //預設縮放量
    D3DXMatrixScaling(&matScale, 0.25f, 0.25f, 0.25f);
    ////預設旋轉量    順序:Y, X, Z
    D3DXMatrixRotationYawPitchRoll(&matRot,
        static_cast<float>(3.14) + YRot,
        0.0f + XRot,
        0.0f + ZRot);
    //求最後結果
```

```
    matResult = matRot * matScale * matTrans * matViewInverse;
    m_WorldMat.SetMatrix(matResult);
}
```

首先我們需要從攝影機取得檢視矩陣(matView)，然後將檢視矩陣反轉，得到檢視反轉矩陣(matViewInverse)，這樣我們就會得到攝影機所在的位置。圖 5-2 是攝影機位置圖。

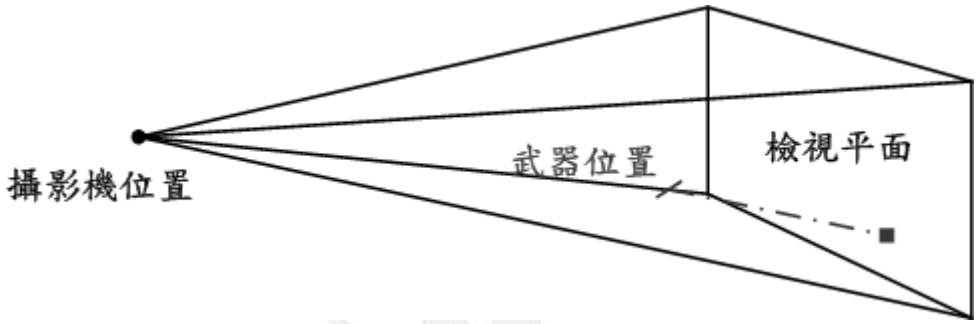


圖 5-2　攝影機位置圖

第二步，要讓武器跟攝影機保持著相對位置，所以我們要平移武器的位置，想像我們把武器架在攝影機前，這時如果以右撇子來看，武器會偏右幾個單位，左撇子則是偏左(使用-X 值)，當然，也可以這這裡加入一些旋轉來使效果更好，在這使用一些預設的值來呈現最終的結果。

最後，我們要用矩陣相乘來求出我們的結果，在作矩陣乘法運算時，順序是很重要的，先把旋轉矩陣(matRot)和縮放矩陣(matScale)相乘，這時武器還在全域位置(0, 0, 0)的地方，然後再乘以平移矩陣(matTrans)來產生相對位置，最後把武器架在攝影機上，也就是乘以檢視反轉矩陣(matViewInverse)。歸納出來的算式：

matResult = matRot × matScale × matTrans × matViewInverse

接下來要做出反彈效果只要在開槍動作的前半段，做幾單位的旋轉，讓手槍向後仰，模擬出後座力的效果，並在開槍動作的後半段，修正回原來的位置，這裡我們需要計時器的幫助。

以下是反彈效果程式碼：

```
void CPistol::Render(cCamera Camera)
{
    if(m_Timer.IsRunning() && m_Timer.GetTime() < 0.25f)
    {
        //反彈
        AssembleWorldMatrix(Camera, D3DXVECTOR3(0.0f, 0.2f, 0.0f),
            static_cast<float>(3.14 / 16), 0.0f, 0.0f);
        //省略其他特效
    }
    else if(m_Timer.IsRunning() && m_Timer.GetTime() > 0.25f)
    {
    //還原
    AssembleWorldMatrix(Camera, D3DXVECTOR3(0.0f, -0.2f, 0.0f),
        -static_cast<float>(3.14 / 16), 0.0f, 0.0f);
    }
    else{//省略}
}
```

一個開槍的動作預設是 0.5f 秒，開槍動作的前半段(也就是 time < 0.025)，將槍位置向上提高 0.2f 單位並對 X 軸旋轉 3.14 / 16 單位(後仰)，後半段則代入相對負值使位置還原。圖 5-3 是武器定位執行畫面。

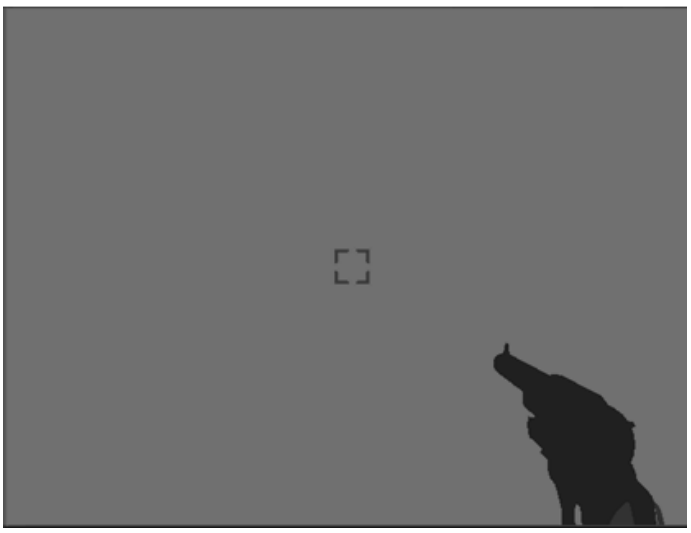


圖 5-3　武器定位執行畫面

5.2　爆破與槍口閃光特效

由於並不是每一把武器都有特效，所以我們在這裡衍生一個新的類別 CPistol，並加入了相關的屬性和方法。圖 5-4 是 CPistol 類別圖。

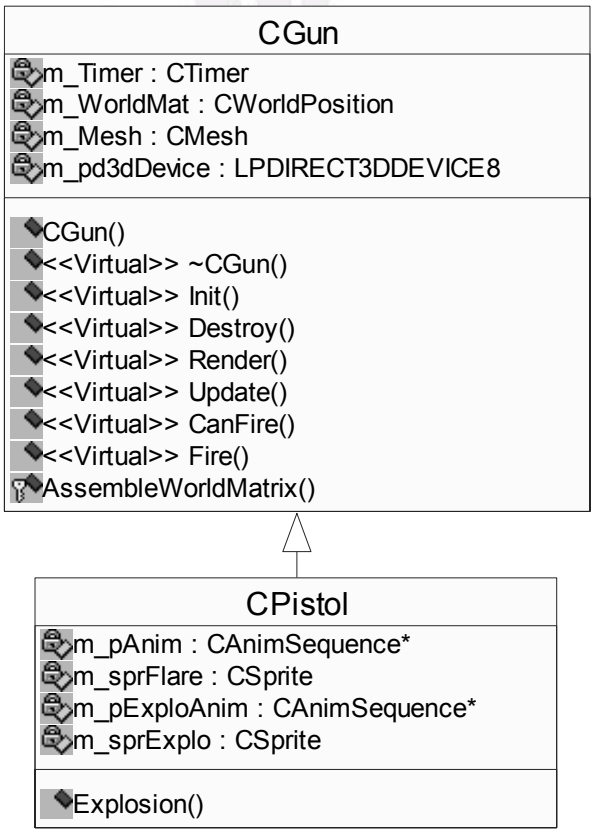


圖 5-4　CPistol 類別圖

CAnimSequence 類別可以用來建立一個動畫順序，CSprite 則是負責管理動畫如何呈現在遊戲中(留至 5.2.2 小節再詳細說明)，要繪製一個動畫，就要先載入其 Sprite。

5.2.1 槍口閃光特效

為了模擬開槍一瞬間的火花，我們需要用到動畫，在這裡使用 PistolFlare.dds(見圖 5-5)來產生一個單一頁面的動畫，動畫只延遲了 0.025f 秒，PistolFlare.dds 是 DirectX 的圖片檔案格式，然後再加到 Sprite 裡。

圖 5-5 PistolFlare.dds

以下是槍口閃光動畫初始化程式碼：

```
HRESULT CPistol::Init(LPDIRECT3DDEVICE8 pDev,char *strMeshFilename)
{
    HRESULT hr = S_OK;
    hr = CGun::Init(pDev, strMeshFilename);

    m_pAnim = new CAnimSequence(pDev);
    m_pAnim->AddFrame("PistolFlare.dds",0.025f);
    m_sprFlare.SetAnim(m_pAnim);

    return hr;
}
```

播放動畫時，我們可以讓 Sprite 隨機縮放和旋轉，使畫面不至於太單調。以下是槍口閃光動畫播放程式碼：

```
void CPistol::Fire()
{
    m_Timer.Start();
    m_sprFlare.Timer().Begin();
    m_sprFlare.SetSize(RandomNumber(1.0f, 3.0f));
    m_sprFlare.SetRotationRoll(RandomNumber(0.0f, 2.0f*(float)PI));
}
```

最後我們加入了「燈光」，DirectX 提供了三種不同的光源，分別是「點光源(Point Light)」(見圖 5-6)、「聚光燈(SpotLight)」和「指向光源(Directional Light)」，我們在這裡只用到點光源。

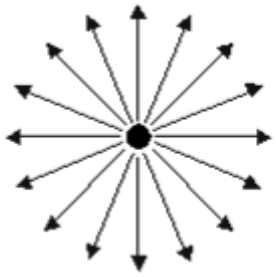


圖 5-6　點光源

點光源就像一顆燈泡，它有位置，但是沒有指定方向，向所有方向放射均勻的光線，我們將一個釋放白光的點光源置於槍口，好讓武器有閃光的效果。圖 5-7 是槍口閃光特效的執行畫面。

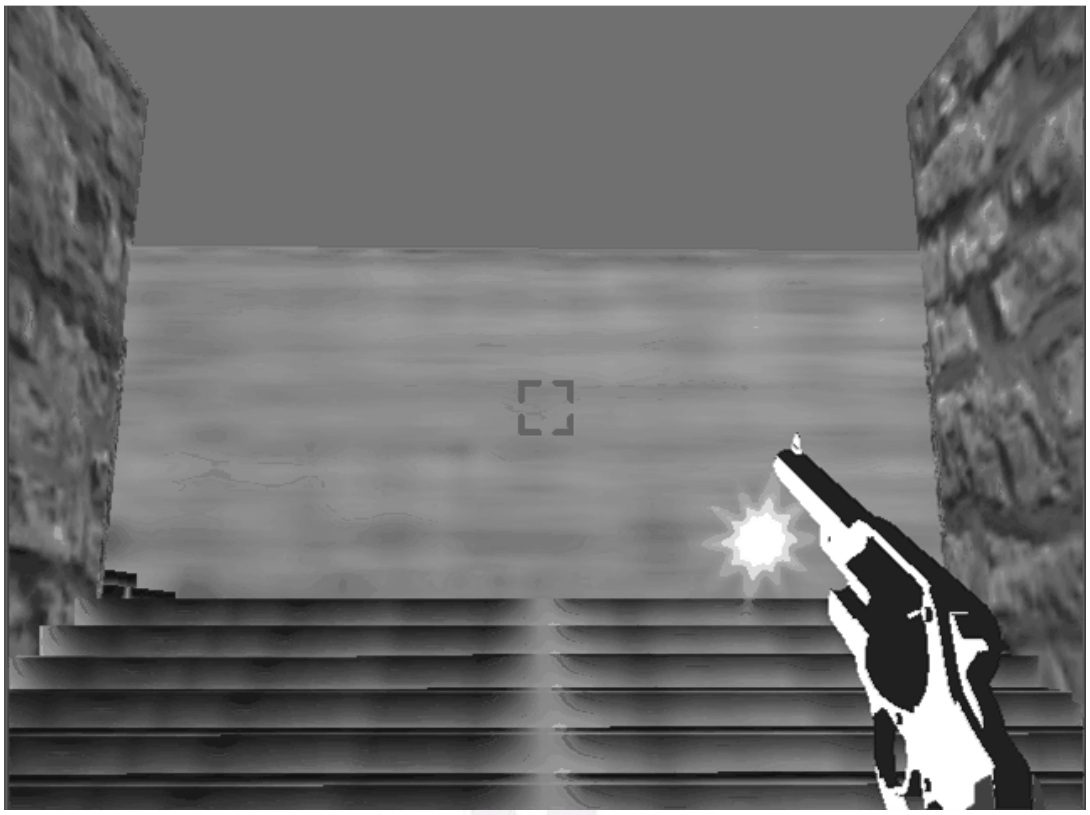


圖 5-7　槍口閃光特效執行畫面

5.2.2　爆破特效

爆破指的是子彈擊中目標後產生的效果，在真實世界裡手槍打中人是不會爆炸的，在這邊為了突顯效果，就做的稍微誇張一點。一次的爆破共包含八個頁面(見圖 5-8)。

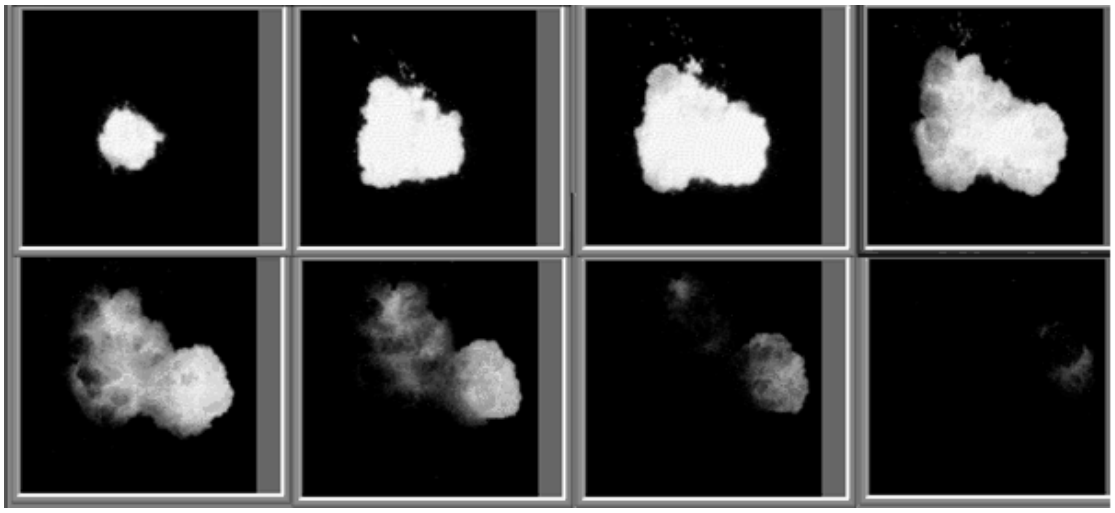


圖 5-8　爆破頁面

CSprite 運作的關鍵是在「告示板技術(Billbording)」(見圖 5-9)，可以讓二維圖形以三維的方式顯示，它最大的優點是可以節省記憶體與 CPU 時間，一棵由幾萬個三角形組成的樹模型可以由一個矩形平面取代(兩個三角形)，只要玩家不要太靠近，就不會被發現，適度的使用告示板技術可以有效的控管資源。

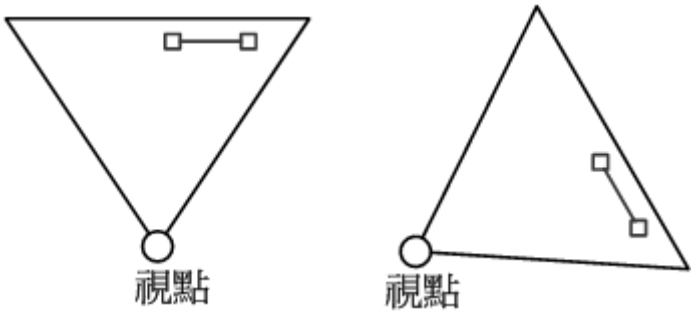


圖 5-9　告示板技術

告示板技術的原理就向在攝影機前面掛著一塊板子，當攝影機轉動時，板子永遠是面對著攝影機的方向，產生一種立體的幻象，要是從別的角度看板子，就會破壞掉這個幻象。以下是告示板技術程式碼：

```
void CSprite::Render(D3DXMATRIX view)
{
    if(!m_pAnim) return;

    // Set up a rotation matrix to orient the explo sprite towards the camera.
    D3DXMATRIX matScale, matTranslate, matTranspose, matBillboardRot,
matWorld;
    D3DXMATRIX matRot; // user-specified rotation
    D3DXMatrixTranspose(&matTranspose, &view);
    D3DXMatrixIdentity(&matBillboardRot);

    matBillboardRot._11 = matTranspose._11;
```

```
        matBillboardRot._12 = matTranspose._12;
        matBillboardRot._13 = matTranspose._13;
        matBillboardRot._21 = matTranspose._21;
        matBillboardRot._22 = matTranspose._22;
        matBillboardRot._23 = matTranspose._23;
        matBillboardRot._31 = matTranspose._31;
        matBillboardRot._32 = matTranspose._32;
        matBillboardRot._33 = matTranspose._33;

        D3DXMatrixTranslation(&matWorld, m_vPos.x, m_vPos.y, m_vPos.z);
        D3DXMatrixRotationYawPitchRoll(&matRot, m_fRotationYaw,
m_fRotationPitch, m_fRotationRoll);

        D3DXMatrixScaling(&matScale, m_fSize, m_fSize, 1);
        m_pAnim->Render(m_Timer, matScale * matRot * matBillboardRot *
matWorld);
}
```

實作的技巧是先取得檢視矩陣的轉置矩陣(matTranspose)，然後取左上方 3X3 的區塊當做旋轉矩陣，然後依照順序，先作縮放和旋轉，最後再作平移，我們就可以得到永遠面對攝影機的告示版了。圖 5-10 是爆破特效執行畫面。

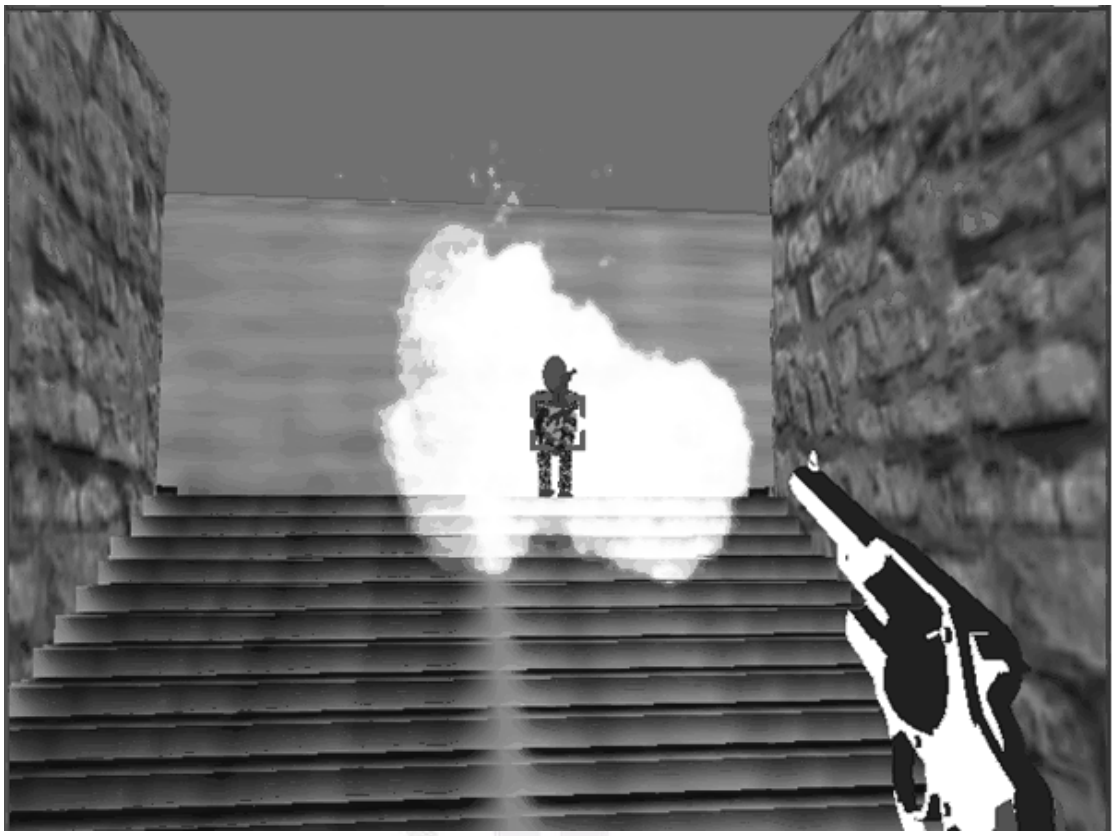


圖 5-10　爆破特效執行畫面

Chapter 6　多人連線遊戲設計

6.1　伺服端

在多人連線遊戲中，伺服端並不需要絢麗的遊戲畫面，也不需要臨場感十足的各種音效，伺服端只需要處理來自用戶端的訊息，並將更新的資料傳送給用戶端。伺服端猶如扮演了訊息處理中心的角色；所有用戶端的訊息均需要透過伺服端來處理並發送至其它用戶端。例如：用戶端加入遊戲、離開遊戲及遊戲中的各種動作等，皆需要透過伺服端來統一管理，以達到各用戶端在網路連線的過程中，可以與其它用戶端保持同步，不致失去多人網路連線遊戲的樂趣。圖 6-1 是 Server-Client 架構圖。

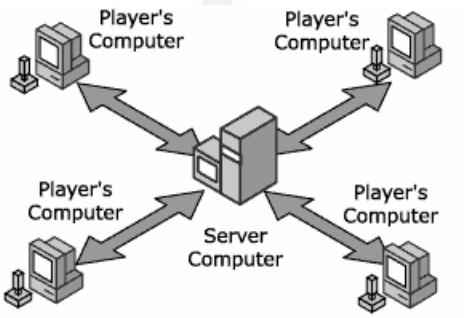


圖 6-1　Server-Client 架構；摘自 DirectX 8.1 document

在伺服器建立連線前，伺服端會列出系統上的 TCP/IP 的設備，在選取完 TCP/IP 介面後，便可以透過此介面與用戶端通訊。之後會進入到伺服器的主視窗，顯示出伺服端的 IP 與用戶端的連線狀況。

在伺服端建立網路連線後，立即進入無窮的迴圈，在迴圈內不斷地接收訊息，並將接收到的訊息從訊息佇列的頂端放入。再將接收來自各方的訊息存到佇列的同時，伺服端有一專門處理訊息的函式，會從訊息佇列的尾端中取出訊息，並且根據訊息的內容處理。圖 6-2 是伺服端簡要流程圖。

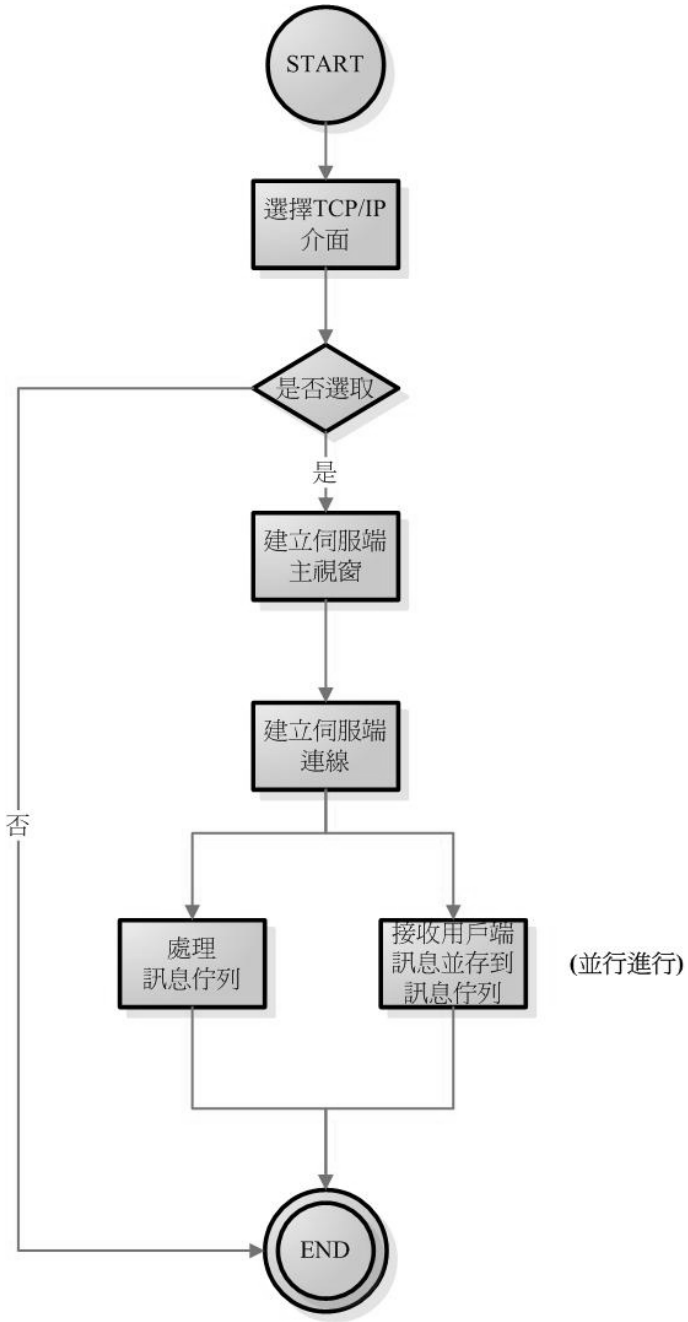


圖 6-2　伺服端簡要流程圖

6.1.1　伺服端所用之物件與函式

為了讓伺服端的處理能夠減少負擔，所以其工作大部分都是在處理由用戶端傳來的訊息，在設計上也是盡量考慮伺服端的主要工作，因此，伺服器除了建立基本的視窗以外，其它工作皆是在處理網路的訊息，以便加速其運作的速度。

本伺服端亦是使用 Game_Core 的核心去設計的，用到的核心有：Core_System 與 Core_Network，並且建立了 cApp 與 cServer 此二物件；在 Core_System 此核心中，建立物件 cApp 去繼承 cApplication 此物件，其主要的工作為建立基本的視窗，並顯示目前的連線狀況與連線人數；而在 Core_Network 核心中，物件 cServer 繼承了 cNetworkServer 物件去建立伺服端的連線，並等待用戶端的連線。表 6-1 和表 6-2 是伺服端所建立的函式。

表 6-1 cApp 類別所建立的函式

class cApp ：public cApplication	
private：	
函式名稱	**主要功能**
BOOL SelectAdapter( )	建立選取網路介面之視窗
void SetupApplicationWindow( )	設定伺服端狀態視窗
BOOL InitializeGame( )	初始化訊息佇列與遊戲角色結構
BOOL HostGame( )	建立伺服端連線資訊，並設定伺服端連線狀態
BOOL QueueMessage(void *Msg)	將收到訊息從訊息佇列的頂端放入
void ProcessQueuedMessages( )	從訊息佇列的尾端取出資料並處理
BOOL AddPlayer(sMessage *Msg)	初始化用戶端加入，並發送通知給其它用戶端
BOOL RemovePlayer(sMessage *Msg)	刪除用戶端的資訊，並發送通知其它用戶端
BOOL PlayerID(sMessage *Msg, DPNID To)	用戶端要求自己的 DirectPlayer 識別碼時使用

BOOL PlayerInfo(sMessage *Msg, DPNID To)	當網路訊息遺失時,可利用此函式再向伺服端要求一次資訊
BOOL PlayerStateChange(sMessage *Msg)	當用戶端改變狀態時,伺服端將根據其狀態通知應改變的用戶端
void UpdatePlayers( )	加快處理更新用戶端的狀態
void UpdateLatency( )	處理訊息在網路上所造成的延遲
BOOL SendNetworkMessage(void *Msg, long SendFlags, int To)	可以將訊息發送至各用戶端,或指定的用戶端
void ListPlayers( )	在視窗上列出所有已連線的使用者
public：	
cApp( )	此 cApp 物件的建構子
BOOL Init( )	Override cApplication 物件的 Init( )
BOOL Shutdown( )	Override cApplication 物件的 Shutdown( )
BOOL Frame( )	Override cApplication 物件的 Frame( )
void SetAdapter(GUID *Adapter)	設定使用介面卡的種類
BOOL CreatePlayer (DPNMSG_CREATE_PLAYER *Msg)	將收到建立連線的訊息放置訊息佇列中
BOOL DestroyPlayer (DPNMSG_DESTROY_PLAYER *Msg);	將收到取消連線的訊息放置訊息佇列中
BOOL Receive(DPNMSG_RECEIVE *Msg)	將收到的訊息放置訊息佇列中,除了建立與取消連線的訊息之外

表 6-2　cServer 類別所建立的函式

class cServer : public cNetworkServer	
Private：	
BOOL CreatePlayer (DPNMSG_CREATE_PLAYER *Msg	將收到建立連線訊息交付 cApp 物件中 CreatePlayer( )處理
BOOL DestroyPlayer (DPNMSG_DESTROY_PLAYER *Msg)	將收到取消連線訊息交付 cApp 物件中 DestroyPlayer( )處理
BOOL Receive(DPNMSG_RECEIVE *Msg)	將收到訊息交付 cApp 物件中 Receive()處理，除了建立與取消連線的訊息之外

6.1.2　伺服端建立網路連線

在伺服端執行之初，伺服端會列出所有在設備上的 TCP/IP 介面，並且建立選擇介面的基本視窗(見圖 6-3)。

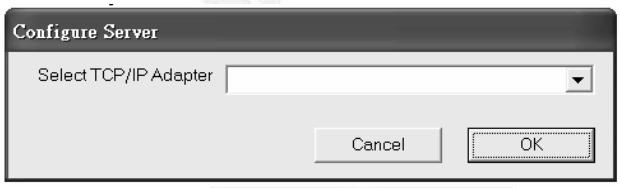


圖 6-3　伺服端介面

在選擇完 TCP/IP 的介面後，伺服端會建立主視窗(見圖 6-4)，並顯示最新的連線狀況；用戶端必須等待伺服端建立連線完成後，才可以進行連線之動作。

圖 6-4　伺服端主視窗

當伺服端連線建立後，在伺服端中之三個函式：CreatePlayer( )、DestroyPlayer( )及 Receive( )，立即不斷的接收訊息且為並行的機制。圖 6-5 是伺服端建立連線的流程。

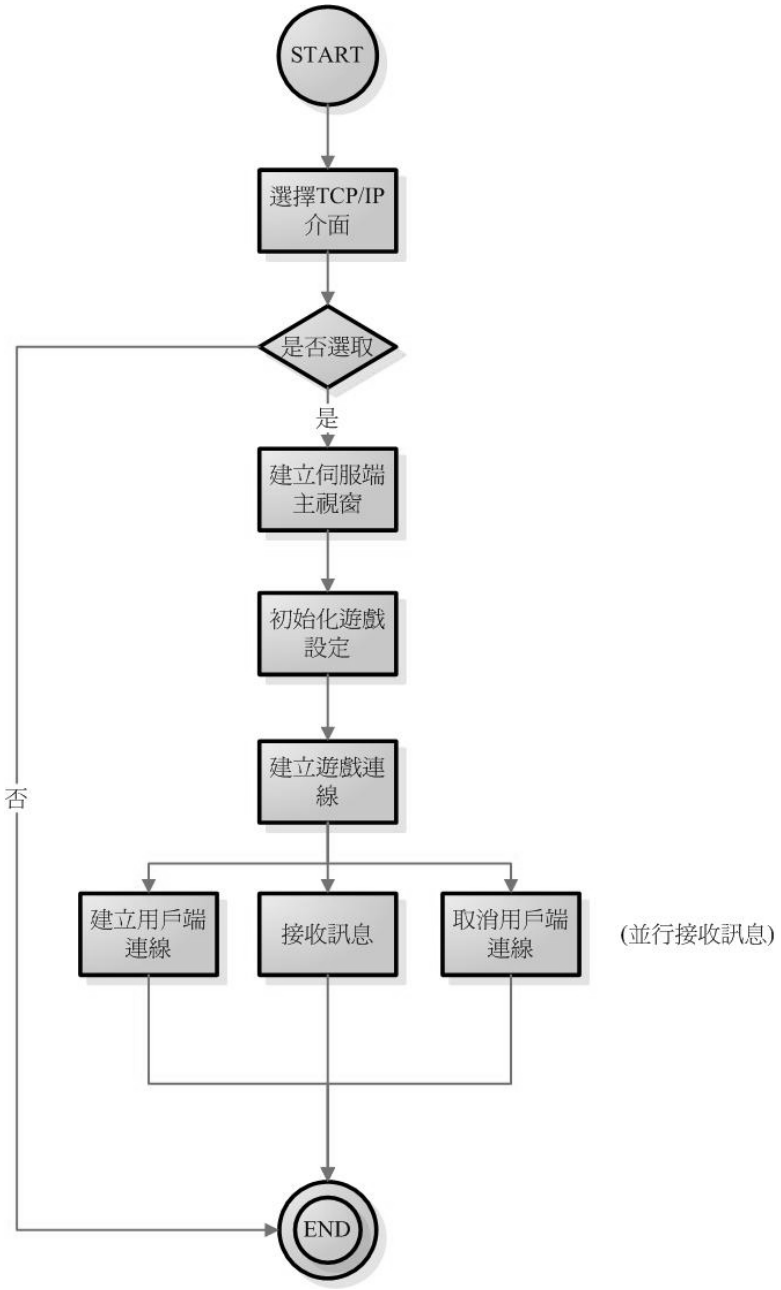


圖 6-5　伺服端建立連線流程圖

在伺服端的程式碼中，建立了物件 cApp 與 cServer，在此二物件中皆有 CreatePlayer( )、DestroyPlayer( )及 Receive( )此三個函式；因為 cServer 主要的功能即是不斷的接收訊息，但是 cServer 並不直接處理訊息，而是交付給 cApp 部分處理，如此一來，伺服端內部便可以直接的處理所有訊息。圖 6-6 此倆物件的關係圖。

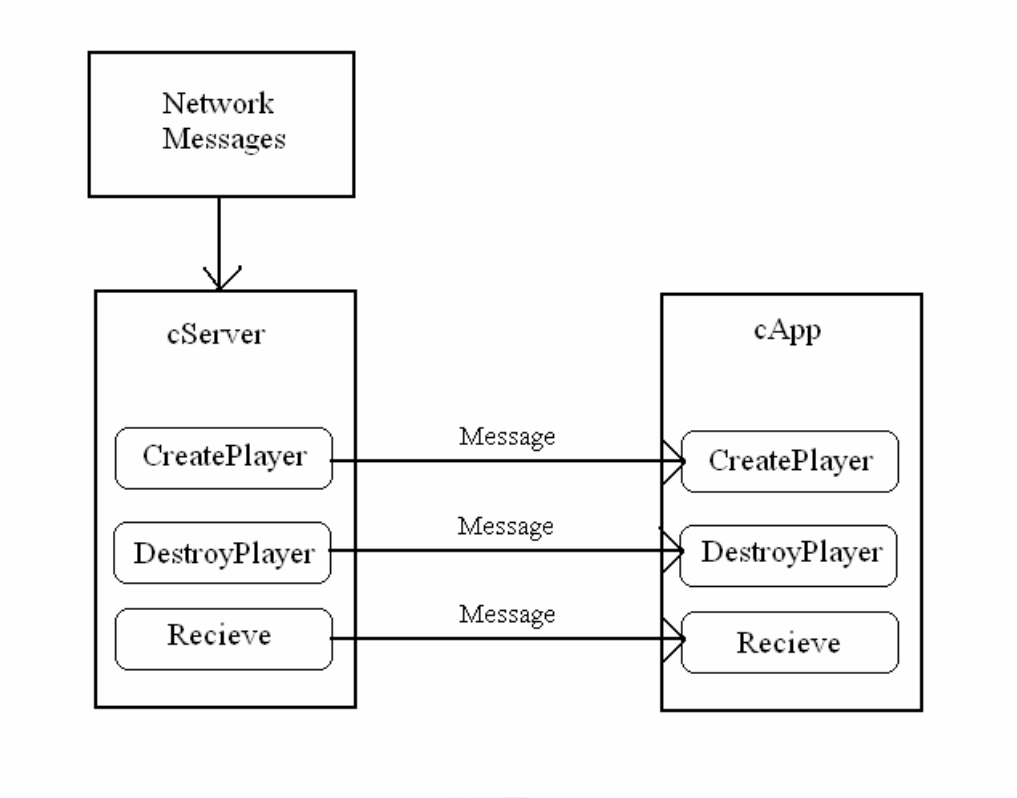


圖 6-6　cServer 與 cApp 關係圖

6.1.3　訊息佇列之處理

訊息佇列的製作，是以結構(sMessage)宣告了一個長度為 1024 的陣列，並且宣告了佇列的旗標，m_MsgHead 與 m_MsgTail，其作用為存放訊息佇列的頂端與尾端之位置；在新增訊息時，訊息將被放入佇列的頂端，而在取出訊息時，訊息將從佇列的尾端取出。訊息佇列在邏輯上可以想像為一環狀佇列，當訊息不斷被放入佇列中時，當頂端的位置超過陣列長度時，其位置將歸零，而尾端的位置亦是如此。

sMessage 結構內容如下：

```
typedef struct
{
    long Type;              //Message 的類型
    long Size;              //訊息的大小
    DPNID PlayerID;     //用戶端的識別碼 ID
}sMessageHeader;

typedef struct
{
    sMessageHeader Header;
    char Data[512];         //訊息內容
}sMessage;
```

伺服端內部處理訊息佇列的機制，非常類似作業系統中 FCFS[1]的 CPU 排班機制；當訊息從各用戶端傳至伺服端時，為了將這些雜亂的訊息用較有順序的方式整理，於是將所有收到的訊息依序放置訊息佇列中，以方便讓伺服端處理所有的訊息。圖 6-7 是伺服端內部處理訊息佇列之示意圖。

[1] FCFS (First Come First Served)：先來的 Task 先服務；為作業系統中最基本的 CPU 排程，雖然其效率最差，但是實做之方法非常簡單，也最為直觀。

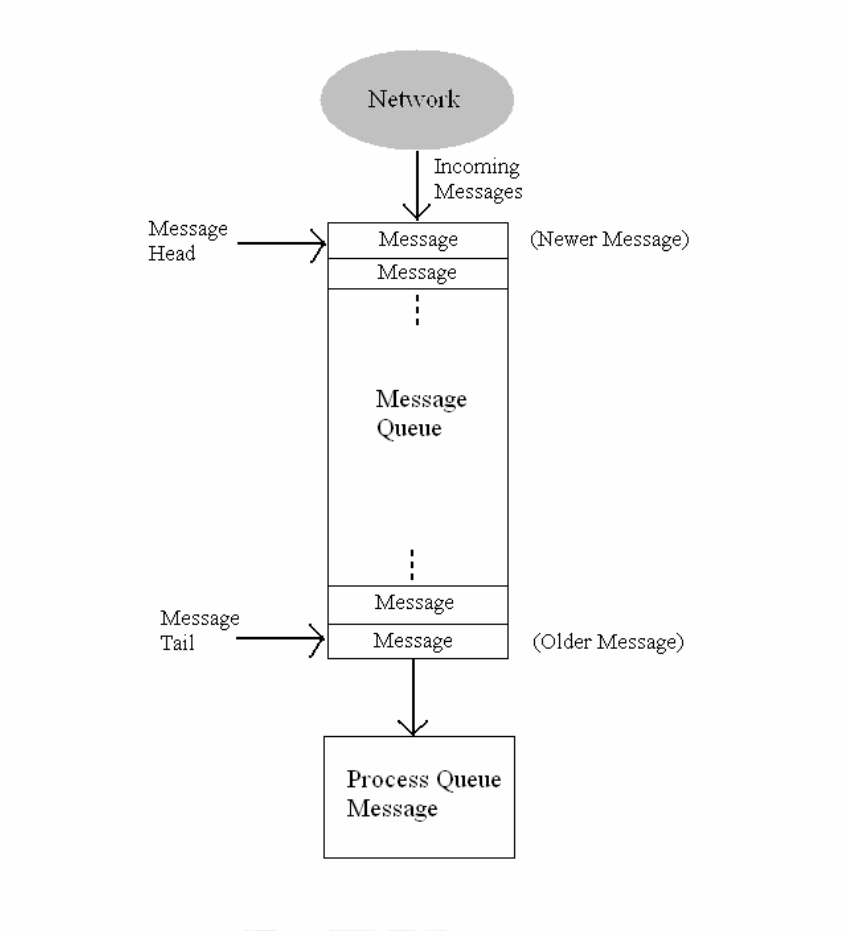


圖 6-7　伺服端內部處理訊息佇列之示意圖

在處理訊息佇列時，必須考慮 Critical Section[2]的問題；由於在伺服端內部，利用多執行緒去處理接收訊息與處理訊息，在此即產生了 Critical Section 的問題，當 Receive( )函式不斷的將接收到的訊息放入訊息佇列的頂端中，而 ProcessQueueMessage( )函式不斷的從訊息佇列的尾端取出訊息的同時，此時如果對同一筆資料作存取，必定會產生資料錯誤。

[2] Critical Section：即為臨界區間，為了避免多執行緒在存取共同的資料區間時所引發的資料衝突；為了解決此問題，在臨界區間裡存取資料時，最多允許一個執行緒進入存取資料，當其工作完成後，才能允許另一執行緒進入。

6.1.4 伺服端處理訊息佇列

在伺服端建立連線成功後，ProcessQueueMessages( )則會不斷的執行，直到伺服端關閉；在 ProcessQueueMessages( )中，其動作會去取出訊息佇列中尾端的訊息，並且根據用戶端所傳送過來的訊息種類(見表 6-3)分配至相對應的函式處理。圖 6-8 為 ProcessQueueMessages( )函式從訊息佇列中取出資料處理之示意圖。

表 6-3 訊息的種類

Type of Message	對應之函式	說明
MSG_STATE_CHANGE	PlayerStateChange( )	用戶端狀態改變
MSG_CREATE_PLAYER	AddPlayer( )	用戶端加入伺服器
MSG_DESTROY_PLAYER	RemovePlayer( )	用戶端離開伺服器
MSG_PLAYER_INFO	PlayerInfo( )	當用戶端遺失其它用戶端訊息，將再向伺服端要求一次
MSG_ASSIGNID	PlayerID( )	用戶端要求自己的 DirectPlayer 識別碼

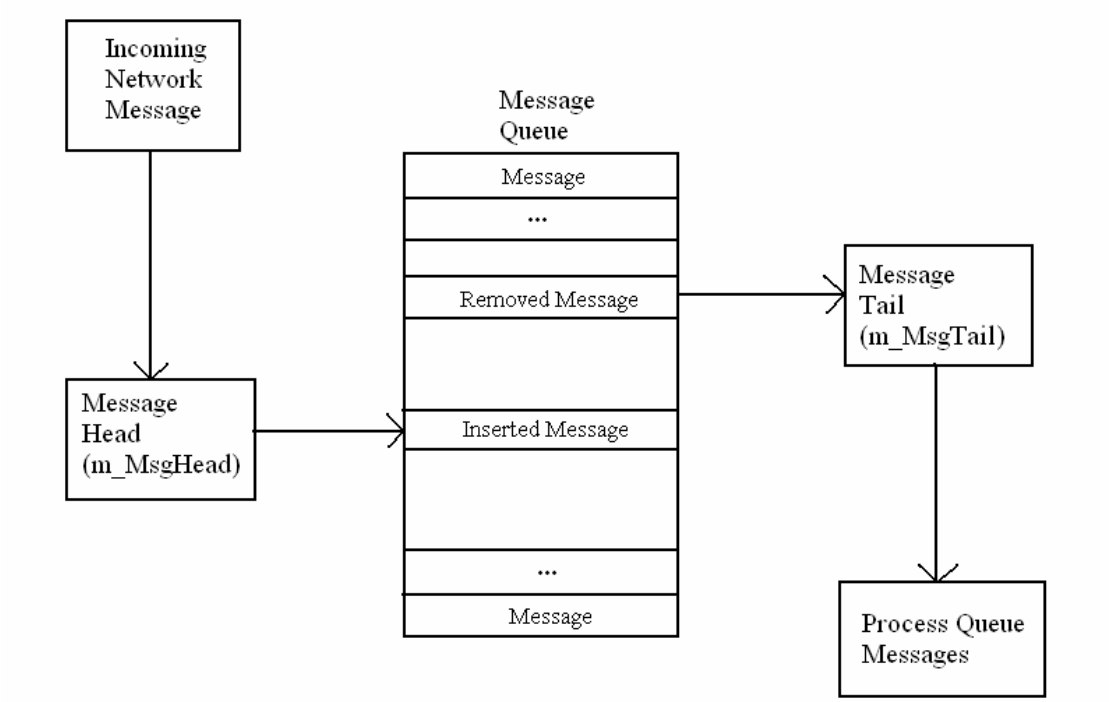


圖 6-8 ProcessQueueMessages( )函式從訊息佇列中取出資料處理之示意圖

6.2 用戶端

用戶端(Client)所扮演的角色，就是幫助玩家與 Server 端建立溝通的管道，借以把本地玩家的訊息傳送給 Server 端，讓其他(她)同時在線上的玩家用戶端可以透過 Server 端即時訊息傳遞來更新非本地玩家的狀態。

在多人連線遊戲中，Client 的功能除了要讓玩家能進行遊戲外，還必須與 Server 端建立連線必透過 Server 端來與其他(她)玩家交換訊息。因此 Client 端的組成除了遊戲主體以外，還有另外兩大部分，分別是(1) Client 端元件 (2)訊息處理。其中遊戲主體和網路部份較無相關，所以以下只對 Client 端元件和訊息處理來做說明。

6.2.1 用戶端元件

內部的成員如表 6-4 所示，主要負責與伺服端之間的連線。

表 6-4　用戶端內部函式成員

函式	**功能**
SelectAdapter( )	取得連線裝置的 GUID。
ConnectServer( )	與 Server 建立連線，並傳送本地玩家姓名
ConnectComplete( )	當 Client 和 Server 端連線成功後，Server 會傳回 “S_OK”訊息給 Client 端
Receive( ) [class m_Client]	接收 Server 發送的所有訊息，然後程式主體的 Receive( )去對應處理
Disconnect()	當 Client 端要離開遊戲時，會利用這個函式與 Server 端斷線
SendMessageToServer( )	傳送本地玩家的訊息到 Server，包含玩家位置、向 Server 要求玩家清單等等

6.2.2　訊息處理

用來處理 Cleint 端和 Server 端的交換訊息，其又分為三部份，分別是：

(1) 儲存玩家資料陣列

(2) 訊息結構

(3) 處理訊息的函式

(1) 儲存玩家資料陣列：用來儲存本地玩家以及其他連線進行遊戲中的玩家資料詳細資料內容如表 6-5。

表 6-5　儲存玩家資料陣列內包含的資料

項目 / **名稱**	**資料型態**	**記錄資料**
Connected	BOOL	記錄此陣列空間是否已經有玩家使用

dpnidPlayer	DPNID	記錄已經連線玩家的專屬 PlayerID
XMove、ZMove、YMove	float	記錄玩家的位置
RotateX、RotateY	float	記錄玩家的視點
m_Player	CPlayer	記錄玩家角色出現的整個形體

我們利用上述的資料，定義一個結構 sPlayer ,並利用 sPlayer 這個結構宣告一個 m_Player 的陣列於每個 Client 端，並利用此陣列來記錄所有連線玩家的資料。

(2) 訊息結構：就是定義要傳送的訊息的基本結構，針對不同的訊息都要去定義不同的內容，詳細內容如表 6-6 所示。

表 6-6 各訊息結構的功能一覽表

訊息結構	**功能**
sAssignPlayerIDMessage	在連線期間，向 Server 要求自己的 PlayerID
sRequestPlayerInfoMessage	較晚加入的玩家向 Server 要求玩家清單時所用的訊息
sCreatePlayerMessage	當本地玩家連上 Server 時或者有新加入的玩家時，Server 會發送這個訊息給本地玩家
sDestroyPlayerMessage	當有玩家離開遊戲時，Server 會發送這個訊息給其他繼續進行遊戲的玩家
sStateChangeMessage	當本地玩家改變狀態時，會利用此訊息發送給 Server 再傳給其他玩家。其他玩家狀態也是利用此訊息傳給本地玩家
sMessage	當接受到 Server 發送過來的訊息，會利用這個訊息結構先存起來

(3) 處理訊息的函式：對於不同的訊息，自然要有對應的函式去處理。詳細內容如表 6-7 所示。

表 6-7　訊息處理函式的功能

項目 / 函式名稱	對應訊息	功能
AssignID( )	sAssignPlayerIDMessage	會把本地玩家自己的 PlayerID 更新為 Server 回傳的 PlayerID 確保 Client 端和 Server 端相同
CreatePlayer( )	sRequestPlayerInfoMessage sCreatePlayerMessage	會把 Server 傳來的 PlayerID 和基本訊息建立到 m_Player 陣列內顯示未連線的改成連線並存入
DestroyPlayer( )	sDestroyPlayerMessage	這個函式會刪除本地玩家 m_Player 陣列內 PlyaerID 和訊息內傳送的 PlayerID 相同的玩家資料
ChangeState( )	sStateChangeMessage	用於更新除了本地玩家以外的玩家狀態改變

6.2.3　Client 端與 Server 端建立連線的流程

當玩家執行遊戲時，會先出現一個對話視窗，讓玩家輸入 Server 端的 IP，以及選擇連線裝置，和本地玩家的名字，如圖 6-9 所示。只有當輸入的資料符合格式，才可以進一步與 Server 端建立連線。按下 Cancel 鍵則直接退出遊戲。

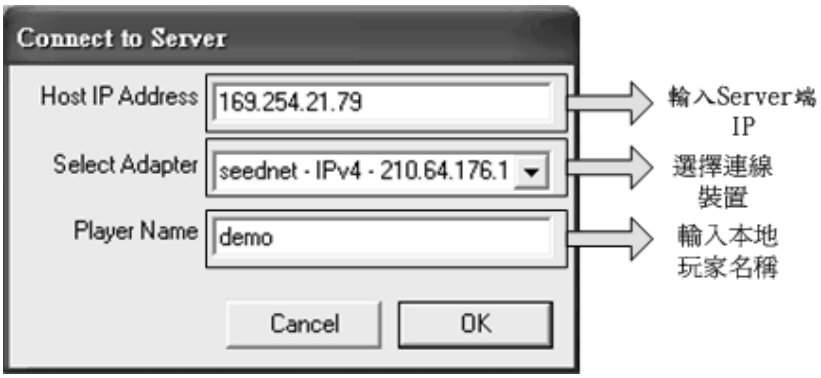


圖 6-9　Client 端執行後的輸入介面

只有當連線成功後，才會開始顯示遊戲畫面，並開始與 Server 端進行訊息交換。而與 Server 端連線的整體概要流程就如圖 6-10 所示。

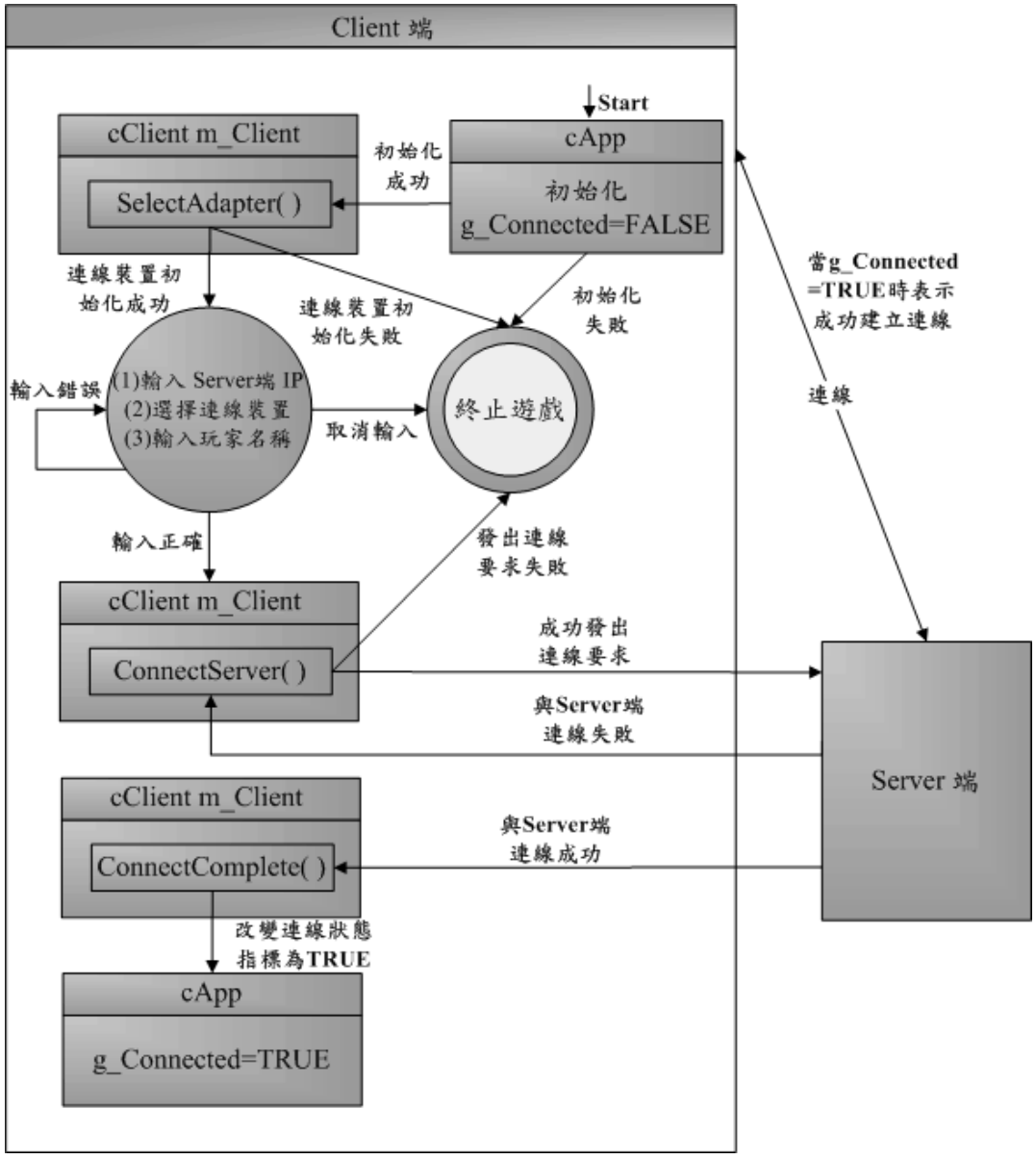


圖 6-10　Client 端與 Server 連線的概要流程圖

6.2.4 Client 端和 Server 端連線後的互動流程

當 Client 端與 Server 端連線成功後，除非是其中一方終止，不然會一直傳送交換訊息。而 Client 端訊息傳送和接收流程如圖 6-11 所示。

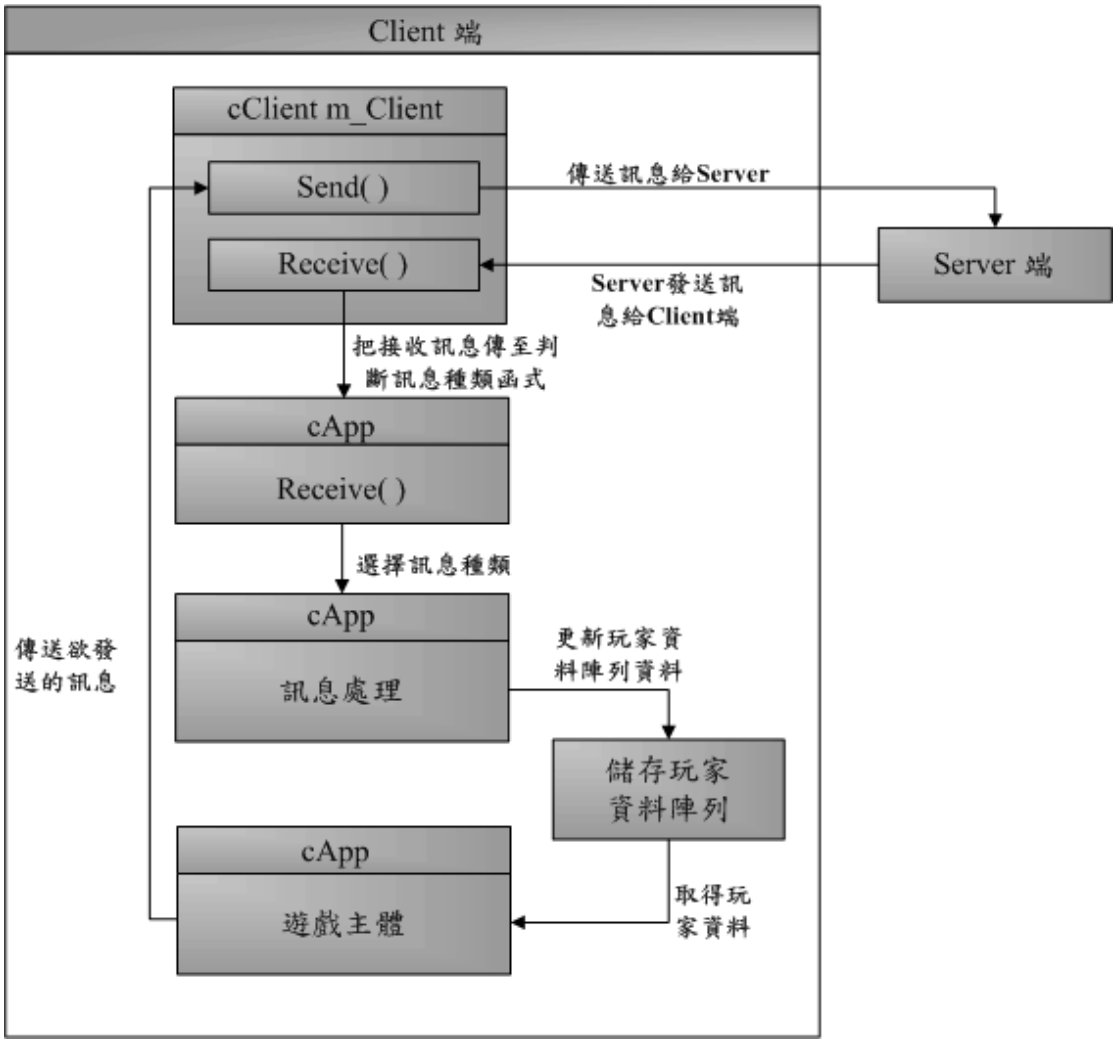


圖 6-11 Client 端和 Server 端連線後的互動流程圖

6.2.5 Client 端離開遊戲與 Server 端斷線的概要流程

當 Client 端離開遊戲時，必須先與 Server 端斷線，並清除所有存在記憶體裡面玩家的資料和遊戲資料。斷線概要流程如圖 6-12 所示。

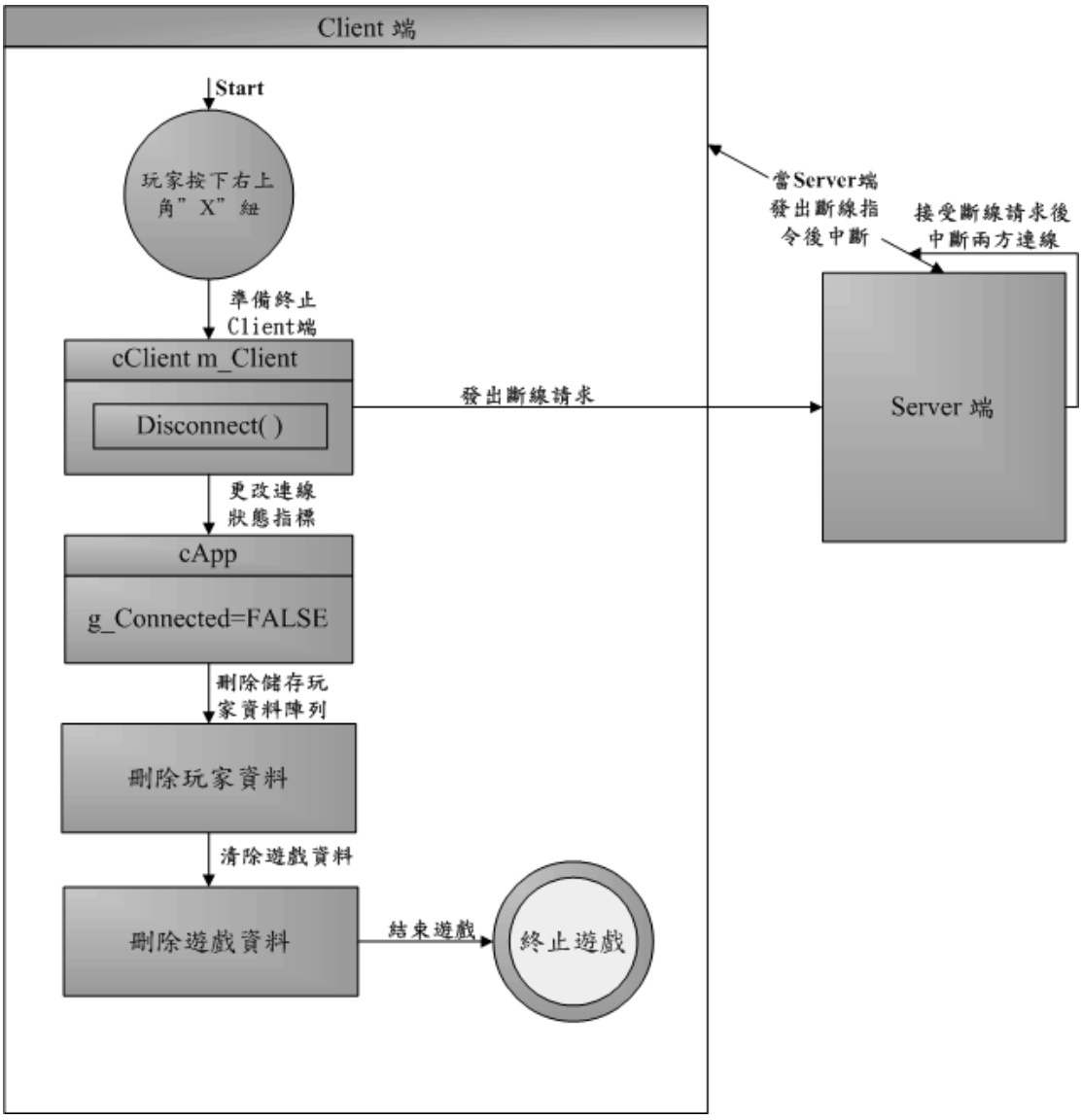


圖 6-12 Client 端離開遊戲與 Server 端斷線的概要流程圖

Chapter 7　未來發展與感想

7.1　未來發展

一個遊戲之所以可以造成熱賣，是因為其遊戲畫面、人物角色、劇情腳本以及聲光特效等都是設計的十分出色，並且受到玩家們的青睞。而我們在 DirectX 的部份就花費了相當多的時間來瞭解其工作原理以及瞭解後要如何運用在我們的遊戲實作上，另外 3DS Max 繪圖工具的操作也投入了相當的時間來學習，所以一些較為細節的地方就沒有特別去實作，或者是一些突發的構想，因沒有去實際操作過，所以不敢貿然的套用。以下我們對於了未來希望達到之功能做一些描述：

(1) 地圖編輯器

以一般遊戲開發團隊來說，程式小組和設計小組是分開的，也就是說程式和資料是獨立分開的，程式小組開發工具供美工或關卡設計小組使用，其中牽涉到「腳本語言」技術。「腳本語言」可以提供設計者最大的彈性，可以讓設計人員任意更改遊戲的內容而不需要動到程式碼。

(2) 效能優化技術

其中包含了分割大場景所使用的「四元樹」和「八元樹」技術，和以刪去阻擋物為基礎的演算法，都能有效的減少系統所需處理的多邊型數量，增進遊戲流暢度。

(3) 更為真實的地圖場景

利用 3DS Max 強大的功能來設計遊戲場景，例如燈光或者是 Particle Systems。利用 Particle Systems 可以製作出雨滴粒子、雪和雲霧等之類的大自然現象；並且利用燈光來製作出光影效果，就如同電影的拍攝，場景的打光是很重

要的，即使是在人工搭建的場景，利用打光技巧一樣可以帶來彷彿身處在真實的房屋裡。

(4) 整合 Server 和 Client

一般遊戲都是既可當 Server 也是可連去別的玩家所開的 Server，所以我們想把 Server 執行檔和 Client 執行檔整合在同一個程式中，使得玩家在執行遊戲時不用開啟兩個執行就能玩，另外當 Server 的玩家在開啟後就可以直接進入自己所開的遊戲中進行遊戲。

7.2 感想

為了完成這個“陽春”的三維空間遊戲，大家從熟悉新的程式語言、新的開發平台，到設計解決問題的資料結構和演算法，背後其實大家付出了蠻多的心血。做這次的專題讓我們學到了一些平時學校作業中不太可能接觸到的東西。我們覺得最重要有四點分別是：

(1) 專案的開發：

專題不像一般作業只要短短幾行程式碼，用到的 Library file 和 Include file 不再僅僅只限於基本內建的，甚至是會用到一些別人開發的 API。這時就要開始去學會設定每個專案所需要的資源，不然假如全部由整體去設定，那每個專案所對應的資源都不同會造成需要時常更改的麻煩。再者，假如這個專案傳到別的電腦去開發，萬一另一台沒有照原本開發電腦設定，那整個專案也可能會因為找不到對應的 Library file 或者 API 而無法執行。

(2) 團隊合作：

當多人一起開發專案的時候，這時團隊溝通就很重要。因為大家都是各自去開發自己負責的功能，所用的變數、參數都是隨自己所好來命名程式碼的區段也都是按自己的想法來放。當最後要整合的時候，往往彼此之間還要特地再找時間來詢問對方的程式如何套用，無形中浪費了不少時間。因此我們在開發時，就已經訂好變數、參數和特定函式的名稱。程式區段的流程也都有統一規格。因此我們在整合遊戲主體、網路程式、音效部分時只要按照當初訂定的規格去查詢就可以迅速整合。

(3) 從別人的程式去學習：

因為專題的東西可以說都是我們第一次接觸，因此我們有去參考別人的一些

作品，看看別人到底怎麼開發，以及到底需要注意哪些地方還有到底用哪種寫法在某種情況下可能會比較好等等。我們花了不少時間在於 Trace 別人作品的程式碼上。對於一些不是很了解的區段我們也會利用 Debug 的方式去看為什麼要這樣做。也因此讓我們對於一些原本認為很直觀的見解有了改變。因為我們不只要讓程式可以動就好，遇到一些例外情況也要能盡快掌握。譬如對於一些開發過程中可能產生的錯誤我們都要能盡快找出來，因此要加入一些不在原本程式範圍裡面的程式碼來幫助除錯。這都是除了開發專案以外要注意的。

(4) Debug 能力的學習：

由於我們的程式是 Client 端和 Server 端互動，因此一旦程式出問題將會很難判斷到底是哪邊的哪個程式區段出問題。這時就需要同時對兩邊 Debug，查看到底是哪裡出問題。再加上我們的程式有包含多執行緒，不能像一般學校作業只要按著順序 Debug 下來就好。因此也要試著去猜測哪邊出問題，才能針對某條路線去 Debug 找出問題。

藉由這次的專題製作，讓我們稍微體驗了一下真正開發一個比較像樣的程式絕對不簡單。從初期的技術資料的查詢、專題開發、成品測試、完成每個過程中都可能會出現上一個過程中想像不到問題，而造成進度中斷無法繼續製作下去。也因此我們要更加注意任何細節不能在像以前一樣想到什麼就直接做什麼；而是要謹慎的去分析各種可能出現的情況。對我們來說應該可以算是從實作中去學習技術以及各項細節。

參考資料

• 書籍

[1] Daniel/著，大師談 遊戲程式設計 核心技術與演算法。

上奇科技出版，2004 年 6 月。

[2] Jim Adams/著，張世敏/譯，2D/3D RPG 角色扮演遊戲 程式設計-使用 DirectX。

博碩文化出版，2004 年 2 月。

[3] Sinan Si Albir /著，黃莉雯/編譯，UML 學習手冊。

美商歐萊禮出版，2004 年 7 月。

[4] Mason McCuskey /著，黃聖峰/譯，DirectX 特效遊戲程式設計。

博碩文化出版，2002 年 9 月。

[5] 連承洙/著，郭淑慧 博碩文化/編譯，嗯！3DS MAX 電腦 3D 動畫我也會。

博碩文化出版，2004 年 2 月。

[6] 繆龍驥 曾俊宏/編撰，向量解析幾何。

徐式基金會出版，民 65。

[7] Oakley, C. O./撰，葉勇/譯，解析幾何。

教育部出版，民 47。

• 網頁

[1] GameDev.net - all your game development needs URL:

http://www.gamedev.net/

附錄一　類別

一、Core(核心相關類別)：

Core_Input	處理輸入作業
Core_Network	處理網路作業
Core_Sound	處理聲音作業
Core_System	處理視窗作業

二、Graphics(圖形相關類別)：

CCamera	處理視點作業
CFont	處理字型作業
CGraphics	處理圖形的主要核心
CLight	處理燈光
CWorldPosition	處理三維空間座標作業

三、FPS(第一人稱遊戲相關類別)：

CGun	武器的基本類別
CMap	處理三維空間地圖作業
CPistol	CGun 的衍生類別
CPlayer	處理玩家作業

四、Mesh(網格相關類別)：

CSkinMesh	網格的主類別
CAnimationNode	處理動畫的節點
CFrameNode	處理框架的節點
CMeshNode	處理網格的節點
CObject	處理 X 檔案的基本節點

附錄二　轉檔成.x 檔操作過程

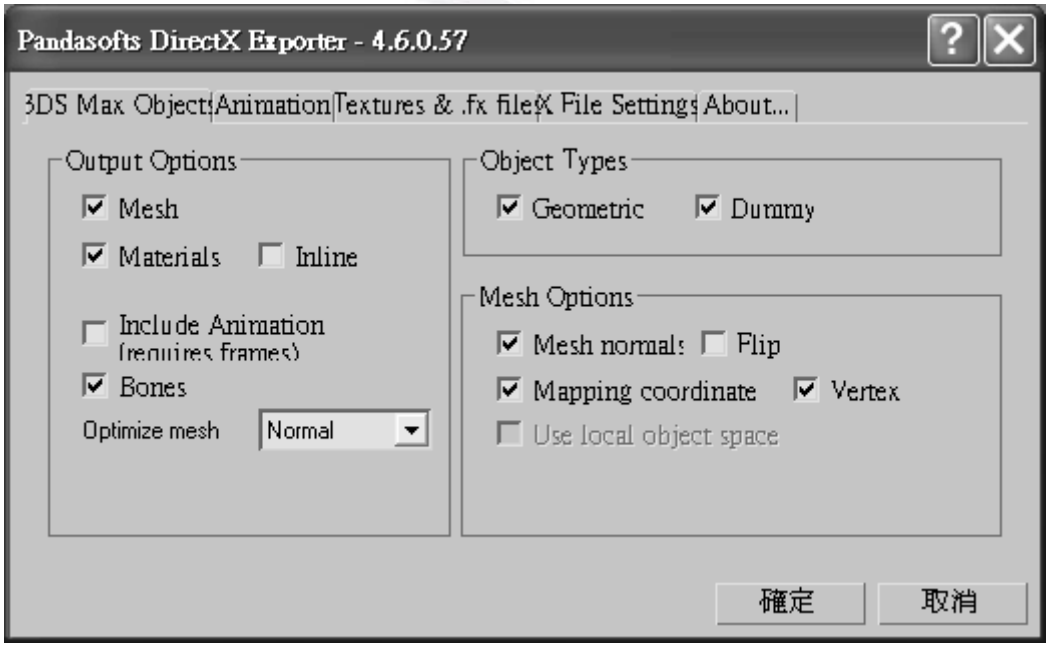
1. 從網路上下載轉換.x 檔的 plugin，放入\3dsmax6\plugins 資料夾中。

   我們是使用 Pandasoft 所製作的 plugin。

2. 利用 3DS MAX 製作好所需的 3D 物件。

3. 點選 File 中的 Export，選擇好儲存路徑，按下儲存。

4. 出現以下畫面，依照所需加以設定，設定完後按下確定。



5. 最後在程式碼中加入產生的.x 檔。

附錄三　遊戲操作手冊

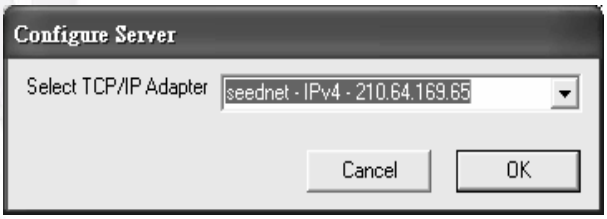
壹、建立遊戲

由於需要連線才能進行遊戲，因此要先開啟 Server 端，再以 Client 連入 Server 端進行遊戲。詳細步驟如下：

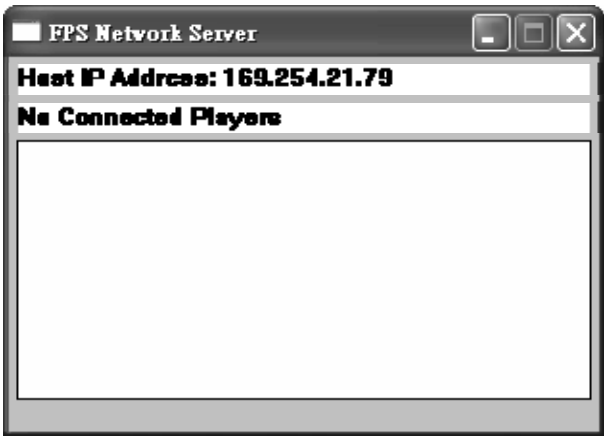
一、執行 Server 端：

(1)執行 Server.exe，此時會見到如右圖對話視窗。若不想繼續開啟 Server 端直接按下 Cancel 鈕即可。

(2)在 Select TCP/IP Adapter 這裡選擇要用來連線的網路裝置，假如沒有選擇會直接預設第一個為連線用的裝置。



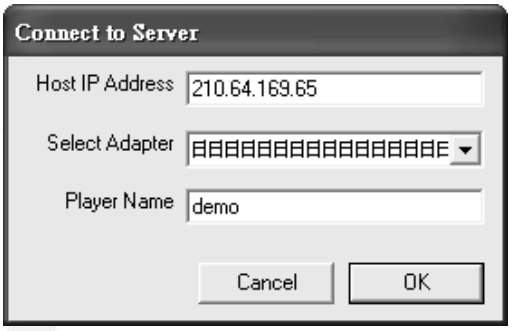
(3)按下 OK 鈕之後，就會出現如右圖所示的 Server 端介面。其中 Host IP Address 是伺服端的 IP 位址。下面的 Connected Players 會顯示連線人數。因為目前沒人連線所以會顯示 No Connected Players。最下面的空白區塊則是顯示連線玩家的名稱。



二、執行 Client 端：

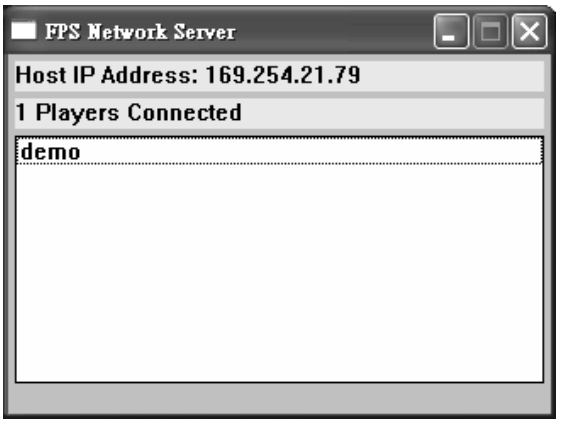
(1) 執行 Client.exe，會出現連線到 Server 端的對話視窗，如右圖所示。假如不想繼續執行 Client 直接按下 Cancel 鈕即可離開。

(2)在 Host IP Address 欄輸入 Server 端的 IP 位址。在 Select Adapter 選擇連線用的裝置，假如沒有選擇也是預設第一個連線裝置。在 Player Name 欄輸入你想要的暱稱。



(3)輸入完畢後，按下 OK 鈕就開始連線到 Server 端。當連線成功，就會出現遊戲畫面。

(4)此時在 Server 端也會顯示連線的人數和玩家暱稱。



貳、遊戲中操作按鍵

- 前進：W 鍵
- 後退：S 鍵
- 左橫移：A 鍵
- 右橫移：D 鍵
- 變形：C 鍵
- 發射槍枝子彈：滑鼠左鍵
- 旋轉視角：滑鼠左右平移