第四章、系統架構與實作

在第三章中，本論文針對以前由本實驗室所研究過的 3D 角色扮演遊戲編輯器的架構進行分析，將其在製作手機上的 3D 角色扮演遊戲之不足之處一一點出。本章將會針對這些不足之處，詳細的說明如何去補足這些缺失，而讓 3D 角色扮演編輯器，也可以編輯在手機上的 3D 角色扮演遊戲。

4.1 系統架構

在第三章提到，新的編輯器必須要有提供以下三點新的功能：

1.需要事先將原先編輯器中的 3D 素材經過轉換，然後透過轉譯器建立可以在手機上的遊戲撥放器撥放的格式。

2.需要一個新的遊戲撥放器，因為原先的遊戲撥放器是在個人電腦的環境上使用。同時手機上的撥放器，必須擁有可以調整 3D 素材在手機畫面上的適當位置及大小顯示的機制。

3.新的通訊模組，原先的 DirectPlay 使用的是 ICP/IP 的連線方式，而目前在手機上還沒有提供 IP，因此，必須在本論文中，改用藍芽進行遊戲的網路連線。

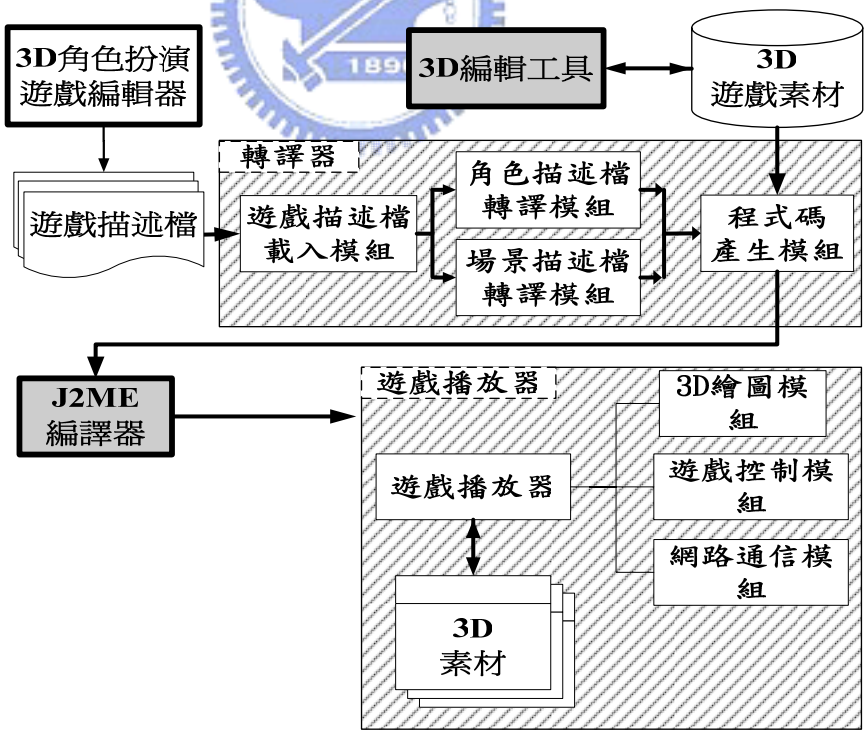

圖 12 系統架構流程圖

圖 12 為代表本論文的系統架構圖。在圖中，以斜線標示的方塊，代表是需要實作

的部分，也就是轉譯器以及遊戲撥放器部分，這兩部份將會在後面的小節中作詳盡的介紹；以粗黑方框所標示的部分，則是已經存在的程式，這些部分不需要實作，而其中以灰色作底的方塊，則表示不是由本實驗室所開發的程式，有使用來轉換 3D 素材格式的 3D 編輯工具，以及將轉譯器編輯結果轉換成 J2ME 程式後，用來編譯成可在手機上執行的編譯器。整個系統的運作流程如下所述：

1.首先，經由 3D 編輯工具，事先將遊戲中會用到的 3D 素材經過轉換，存放在遊戲編輯器的素材資源庫中。

2.接著，使用者透過 3D 遊戲編輯器編輯 3D 遊戲，編輯完後的結果儲存成遊戲描述檔。

3.由轉譯器將遊戲描述檔載入，接著進行轉譯，分別交由場景及角色描述轉譯模組進行轉譯的工作，並且根據描述檔，到素材資源庫調取 3D 素材資源檔由，最後產生 J2ME 的程式。

4.透過手機上的 JVM，執行由轉譯器產生的程式碼。

以上就是本論文系統架構以及操作流程說明，接下來的章節將會針對系統中各部份功能的實作作更詳細的說明。

4.2 3D 素材轉換

在本節中將會介紹如何將編輯器中3D素材利用其他的3D編輯工具轉換成Wavefront格式的檔案。再由這個轉換出來的檔提供3D素材的繪圖資料給轉譯器轉換出使用於遊戲之中的實際格式〔27〕。

4.2.1 Wavefront 格式簡介

Wavefront 是一種用來儲存 3D 幾何資料的檔案格式。它的特點是使用 ASCII 的編碼格式儲存，因此使用一般的文字編輯器也可以對 Wavefront 格式的檔案進行修改。目前大部分的 3D 編輯工具都支援 Wavefront 格式的檔案載入或是匯出，一般來說，當編輯者使用不同的 3D 編輯工具編輯 3D 物件時，Wavefront 格式的檔案是用來當作中介檔案使用，用來作為轉移原來的 3D 物件資料到另外一個 3D 編輯環境之下。其儲存的副檔名是 obj〔28〕。

以下介紹幾類在 Wavefront 格式中使用的關鍵字：

1. 頂點資料（Vertex Data）：

關鍵字代號	說　　明
v	代表 3D 物件中多邊形的頂點。
vn	代表 3D 物件的頂點上的法向量。
vt	代表 3D 物件貼圖時的圖片的對應位置。

2. 群組（Grouping）：

關鍵字代號	說　　明
g	代表以下所列的資料，是該 3D 物件中被 group 在一起的部份。

3. 元素（Elements）：

關鍵字代號	說　　明
f	代表由那些頂點構成的多邊形之面。

4. 顯示/繪圖屬性（Display/render attributes）：

關鍵字代號	說　　明
mtllib	3D 物件材質設定的資料檔的所在。
usemtl	指定 3D 物件材質設定方式。

以下是一個範例檔案，讓我們實際看看 Wavefront 如何來描述一個 3D 物件：

```
1    # Wavefront OBJ exported by MilkShape 3D
2    mtllib penguin.mtl
3    v 0.004520 0.000000 -0.363459
     ...#124 vertexes
4    vt 0.000000 1.000000
     ..#95 texture coordinates
5    vn 0.000000 -1.000000 0.000000
     ...# 92 normals
6    g Cylinder01
7    usemtl Material01
8    f 1/1/1 2/1/1 3/1/1
     ...
```

第一段是我所使用的 3D 編輯工具所加的註解，在 Wavefront 的檔案中，以'#'開頭的就是註解。第二行開始就是真正 3D 物件的資料。

第二段首先宣告一個給 3D 物件所使用的材質設定檔，這個檔案稍後在介紹。接下來的三、四、五段，就是整個 3D 物件的頂點、法向量、貼圖頂點的資料。

接著，第六段，宣告以下的資料組成一個叫做" Cylinder01"的 group。第七段，則是告知以下的物件所使用的材質是在" penguin.mtl"這個設定檔中的"Material01"這個材質設定。第八段，則是去定義由哪些頂點構成了一個三角形，再由這個這些三角形組合成整個 3D 物件。

4.2.2 Wavefront 的材質檔介紹

在上節中，已經對 Wavefront 的格式做了簡單的介紹。但是在上節之中很明顯的可以發現，在 obj 檔中只存在 3D 物件的架構資料，也就是說 obj 檔所表示出來的只是個沒有顏色的 3D 物件。接下來，本節將會介紹與 obj 檔相輔相成的材質設定檔（副檔名是 mtl）。

材質設定檔的基本架構大致如下所示：

newmtl *xxx*
Material color & illumination statements
texture map statements

分別說明如下：

1. newmtl
   就是材質設定的名稱，提供給 obj 檔作引用之用。

2. Material color & illumination statements
   這段是設定 3D 物件的顏色、透明度以及反射光影的程度。
   有以下四種設定可以使用：

設定代碼	說　　明
Ka	設定物件經過環境光照到後看到的顏色。
Kd	設定物件經過散射光照到後看到的顏色。
Ks	設定物件經過反射光照到後看到的顏色。
illum	設定物件在材質上所呈現的光影效果。

3. texture map statements
   這段是設定 3D 物件的貼圖效果，與上面的材質設定相關，可以有 Ka、Kd、Ks 三種不同的光源設定，以呈現不同的貼圖效果。

經過了上面對材質描述檔如何對物件的材質設定屬性之後，接下來以 4.2.1 節中所提過的材質設定檔的檔案內容為範例，以這個材質設定檔來看實際上到底是如何透過以上的屬性去設定 3D 物件的材質：

```
1   newmtl Material01
2   Ka 0.011765 0.752941 0.752941
    Kd 0.909804 0.690196 0.905882
    Ks 0.933333 0.933333 0.933333
    Ns 445.312500
3   illum 2
4   map_Kd penguin.gif
```

第一段，就是宣告材質設定的名稱。第二段，設定 3D 物件在光源的照射下所呈現的顏色，在這邊是使用 rgb 值做顏色的設定。第三段，是宣告使用已經定義好的光影效果設定，這邊的 illum 2 是指 Highlight on，就是強調這個物件。最後第四段則是宣告使用哪個圖片來進行貼圖，在這邊是使用散射光作貼圖效果的呈現。

在經過前面 4.2.1 以及 4.2.2 節的介紹後，透過 Wavefront 的 obj 與 mtl 檔提供的資料，就可以重建原來 3D 物件的樣子。更因為它本身的格式並沒有加解密的問題，因此，在本論文中採用 Wavefront 格式作為中繼格式，以便從原先 3D 遊戲編輯器中的 3D 素材，轉換至可以給手機執行的 JAVA 程式來使用。圖 13 說明的流程。

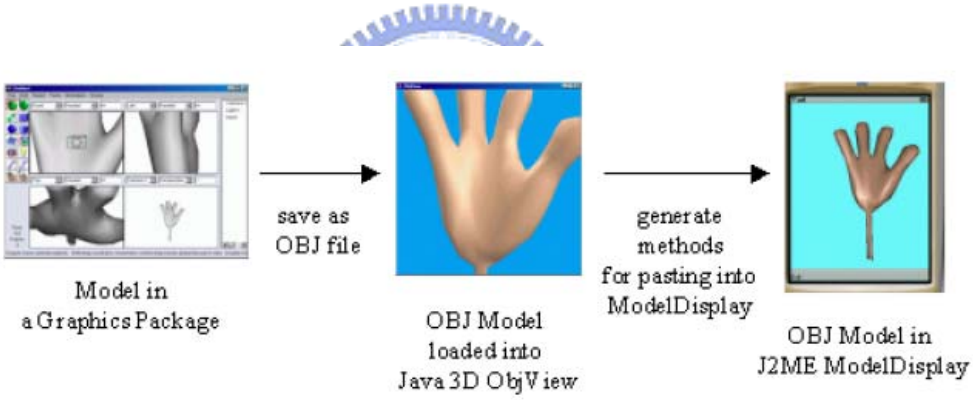

圖 13 使用 3D 編輯工具轉換 Wavefront 格式示意圖〔27〕

在本節中所述的 Wavefront 是摘錄自 Wavefront 的規格說明，只取與本論文研究相關的地方做說明。假如想要對於 Wavefront 做更深入的了解，可以參考論文後面參考文獻 28－Wavefront 規格說明。

4.3 轉譯器設計

本節將會針對如何設計將 3D 遊戲編輯器的編輯結果轉換成手機上執行的程式的轉譯器作詳細的說明。

4.3.1 遊戲描述檔介紹

在詳細說明轉譯器的實作之前，本節將會先針對編輯器中的描述檔作簡單的介紹。

3D 遊戲編輯器所提供的描述檔大致上可以分成兩類〔10〕〔11〕:

1. 場景描述檔

場景描述檔負責記錄所有遊戲場景的相關資訊。包含有：

a.場景中物件的出現的位置以及大小。

b.場景的背景設定，包含背景所用的圖片以及背景音樂。

c.與該場景有關的事件記錄。

在下頁圖 14 場景描述檔的結構示意圖中可以看到，一個場景描述檔會有一個地圖描述（Map Script），在地圖描述中，紀錄了所有在遊戲中所使用到的場景之描述紀錄。而每一筆場景紀錄又可以再細分成兩塊，就是圖四-3 中粗黑色箭頭所指的兩部份，事件描述（Event Script）以及物件描述（Model Script）。

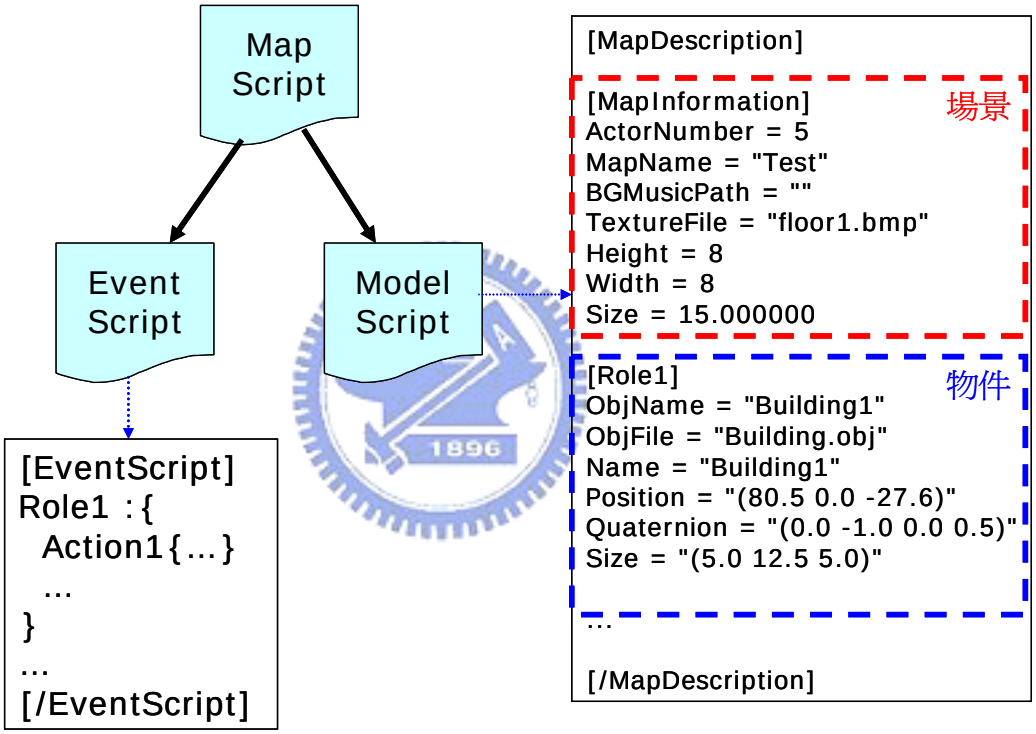

圖 14 場景描述檔結構示意圖

物件描述部分的描述格式架構可以參考圖 14 的右半部。物件描述部分本身又可以拆成兩個部份來看：

a. 場景描述

此一部分是記錄了整個場景的相關資訊，透過一些屬性的設定，定義了整個遊戲場景的構成，茲說明屬性如下：

屬 性 名 稱	說 明
ActorNumner	指在這場景之中，有多少 3D 物件存在。
MapName	場景名稱。
BGMusicPath	場景的背景音樂。
TexturePath	地板所使用的圖。
Height/Width	地板大小。
Size	地板單位。

b. 物件描述

此一部分的描述，則是詳細的記錄著在場景中每個物件的相關資料。而每個物件都會使用一些屬性來設定其在場景中的呈現方式，如下表所述：

屬性名稱	說 明
ObjName	設定物件的名稱。
ObjFile	設定物件所使用的 3D 資料來源。
Name	假如該物件是場景中的 NPC 角色的話，這個屬性就代表它的名字；否則就與物件的名稱一樣。
Position	物件在場景中出現的位置。
Quaternion	物件的面對方向。
Size	物件的大小。

至於事件描述的部份，就沒有屬性需要設定。它的格式是以場景中的角色為單位，將這個角色會有哪些事件需要處理的一一列出。當發生事件時，再到該角色的事件清單中找出對應的事件作處裡。下面是一個場景中的事件描述的範例，在這個場景中只有 Building1 這個物件擁有事件，當角色接觸到 Building1 時就會去觸發它的事件－詢問角色是否要進入房子？選擇”Yes”，就會切換場景，選擇”No”，就會顯示放棄的訊息。

```
〔EventScript〕
Actor Building1 : {
        ShowMessage(“Do you want to enter this house?”);
        ShowChoice {
                "Yes" : { Teleport : Room(62.011872 -45.096008); }
                "No" : { ShowMessage("Give up");}
        }
}
〔/EventScript〕
```

2. 角色描述檔

角色描述檔的格式與上面提到的場景物件描述是一樣的，可以參考下圖 15。

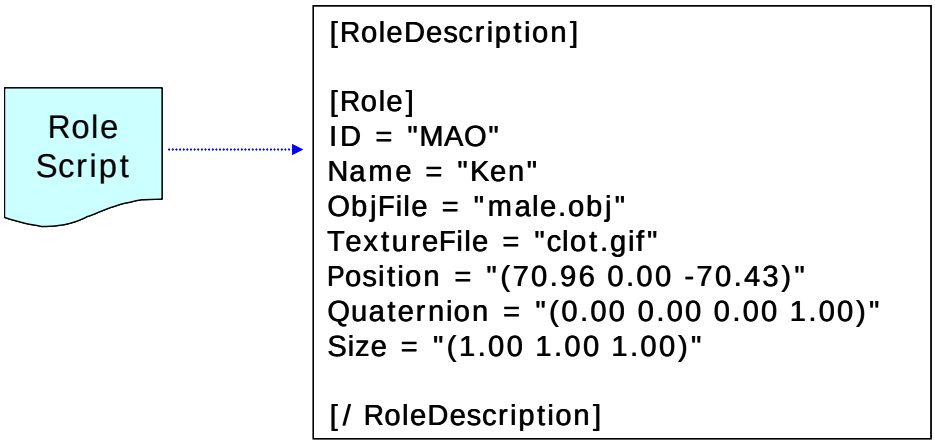

圖 15 角色描述檔格式範例

唯一的差別，在場景物件描述中的 ObjName 屬性，在角色描述檔中被改稱做 ID。角色描述檔提供使用者進行遊戲操作時的角色設定資料，被獨立出來置放在另外一個檔案之中。可以使用 3D 遊戲編輯的主角編輯功能來進行修改。

4.3.2 轉譯器架構及流程

接下來本節將會說明轉譯器的架構，以及它的運作流程。

轉譯器一共分為四個模組，分別為：描述檔載入模組、場景描述檔轉譯模組、角色描述檔轉譯模組以及程式碼產生模組。可以參考下圖 16。

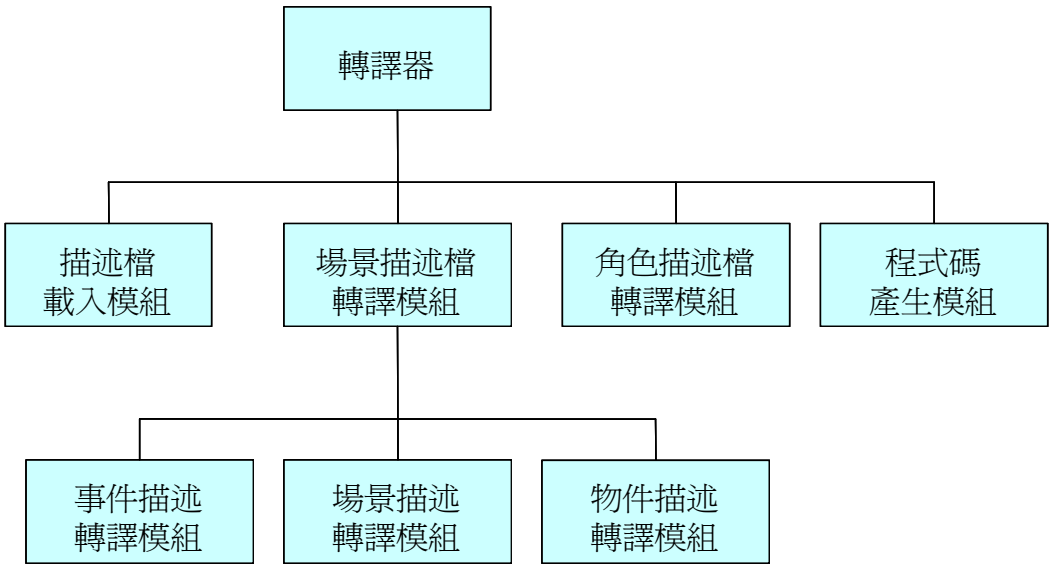

圖 16 轉譯器模組架構圖

考量到以後的遊戲描述檔的擴充性，所以在實作上就依照描述檔的類型作切割，方便以後做延伸之用。接著下面就分項說明各模組的功能：

1. 描述檔載入模組

這個模組負責將 3D 遊戲編輯器產生的遊戲描述檔載入到轉譯器中。在轉譯器中使用如圖 30 所示資料類別儲存。

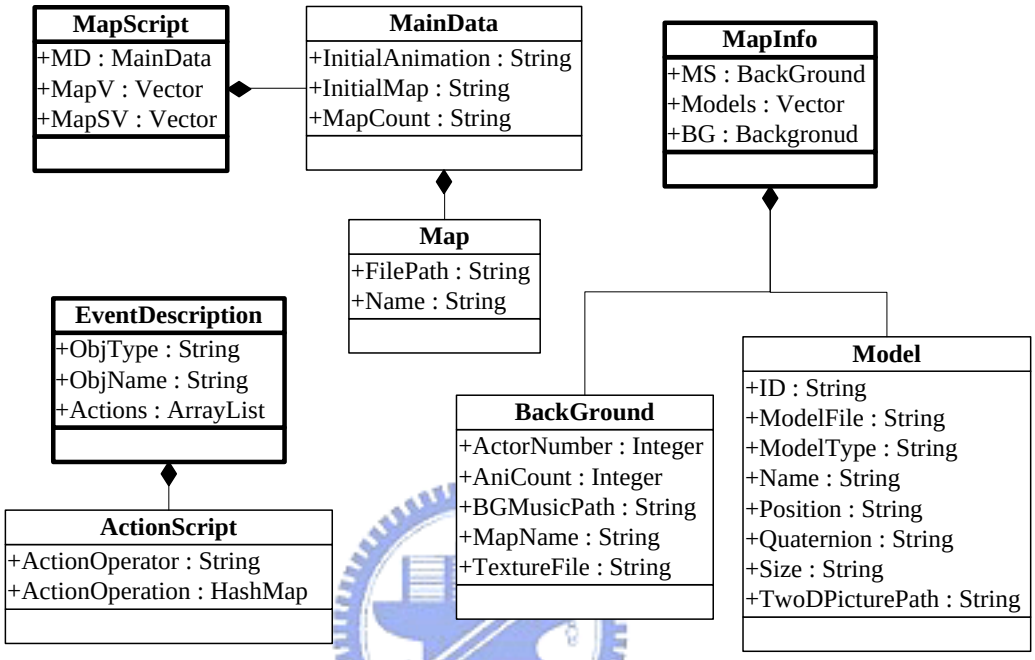

圖 17 用來儲存遊戲描述檔的資料類別

在圖 17 中，以粗黑框所表示的類別就是依照在 4.2 節中所提到的遊戲描述檔格式而定出來的資料類別。MapScript 類別對應到場景描述檔，MapInfo 類別對應到場景描述檔的場景描述部分，EventDescription 類別則是對應到場景描述檔中的事件描述部分。角色描述檔，則使用 Model 類別即可儲存其資訊。

2. 場景描述檔轉譯模組

在場景描述檔轉譯模組之下，可以在根據描述檔中不同的部份在細分為三個子模組：

a. 場景描述轉譯模組

特別針對場景描述中的場景環境部份作轉譯的工作。包含了場景中的背景、光源設定、鏡頭方向及位置設定。

b. 事件描述轉譯模組

針對場景描述檔中的事件描述作轉譯的動作。此模組的主要目的是產生該場景中的事件列表，以提供遊戲播放器，當遊戲事件發生時可以查詢，以作相對應的處理。

c. 物件描述轉譯模組

針對場景描述檔中的物件描述檔作轉譯的動作。此模組會根據描述檔中提供的資訊，到資源素材管理庫中取得物件的 3D 模組資料，然後轉換成類別，提供給遊戲播放器使用。

物件描述轉譯模組所轉譯出來的類別都是繼承自圖四-7 所示的抽象類別。

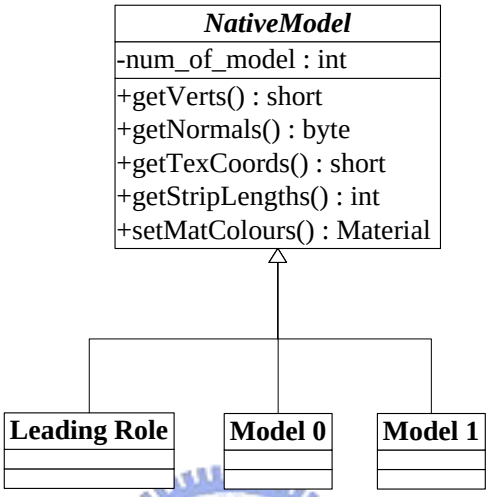

圖 18 物件描述檔所使用的抽象類別－NativeModel 類別

在圖 18 中的 NativeModel 類別就是抽象類別，所有在遊戲中的 3D 物件的資料就利用這個類別儲存。在 NativeModel 類別中定義了五種函式，讓其他的程式可以取得 3D 物件的資訊，茲說明如後

函式名稱	說　　明
getVerts	取得 3D 物件的頂點資訊。
getNormals	取得 3D 物件的法向量資訊。
getTexCoords	取得 3D 物件的貼圖頂點資訊。
getStripLengths	取得 3D 物件多邊形組成資訊。
setMatColours	取得 3D 物件材質設定資訊。

當初在設計物件的資料要如何存放在程式之中時，曾經考量過許多方式，但是因為 J2ME 在對程式記憶體上的設限，事先在程式中置放這些 3D 物件的資料是不太可行的。再加上遊戲播放器的繪圖模組是使用直接在畫面上繪圖的模式製作遊戲，所以 3D 物件的資料會跟遊戲播放器的繪圖模組有一定的相依性。因此，在物件描述轉譯模組的實作上，採用了物件導向中多型的概念。透過物件描述轉譯模組，將遊戲描述檔中的 3D 物件一個一個轉換成單一個類別，而所有的 3D 物件類別都繼承自 NativeModel 這個抽象類別，使得在後面製作遊戲播放器時，可以輕易的從這些物件類別中取得 3D 物件的繪圖資料來進行繪圖。

3. 角色描述檔轉譯模組

基本上角色描述檔的處理方式與場景描述檔轉譯模組中的物件描述處理方式是一樣的，也是將角色的 3D 資料轉換成 JAVA 的類別。而角色與物件不同的地方在於，角色的 3D 資料類別除了 3D 繪圖的資訊之外，另外還包含了角色目前在場景中的資訊，如目前角色在畫面中的位置、面向的方向、角色身上的物件。這些是在角色物件中所獨有的訊息，以便在遊戲播放器中紀錄角色接受使用者操作後所造成的變化。圖 19 是用來存放角色在遊戲中資訊的類別圖，同樣也是抽象類別，可供以後延伸設計之用。

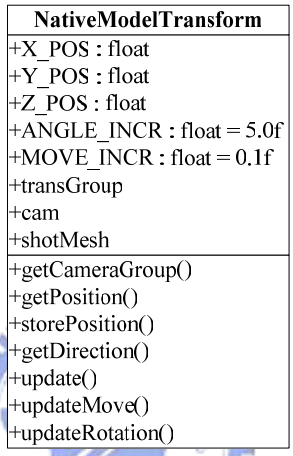

圖 19 NativeModelTransform 類別

4. 程式碼產生模組

本模組負責將描述檔中提供的遊戲資訊與下一節所提到的遊戲播放器整合在一起，然後在藉由 JAVA 的虛擬機器來執行整個遊戲。

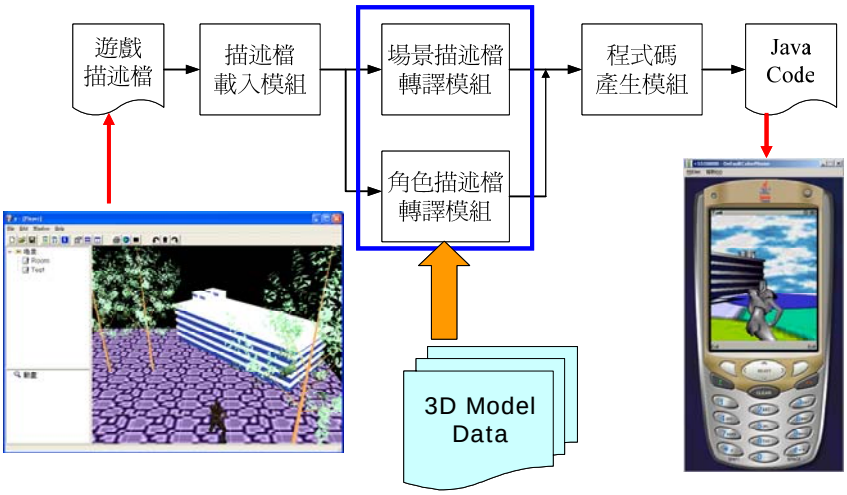

圖 20 轉義氣運作流程圖

圖 20 是轉譯器的運作流程，當使用者利用 3D 遊戲編輯器編輯完成遊戲後，經過載入模組讀入遊戲描述檔，透過轉譯模組萃取遊戲的資訊後，轉變成 JAVA 的類別，最後再交給 SUN 所提供的編譯器，編譯成 Byte Code 後，便可以載入手機，或是在模擬器上執行遊戲。

4.4 遊戲播放器之設計

本節將會針對手機上的遊戲播放器的實作細節，做詳細的說明〔25, 27, 30, 31〕。

4.4.1 遊戲播放器的架構

根據第三章中所分析的，遊戲播放器必須具備下面三種能力：

1. 在本研究中採用的是 JSR184 所提供之繪圖模式中的立即模式（Immediate Mode）。因此，遊戲播放器必須不斷更新畫面的方式進行遊戲畫面的繪製。
2. 同時，遊戲播放器也必須需要隨時偵測使用者的輸入，並反映在畫面上。
3. 當遊戲是在網路上進行時，遊戲玩家在遊戲執行時的訊息傳遞也需要透過遊戲播放器來負責傳遞、處理。

因此，本論文將遊戲播放器分成三個子模組，將上面三個功能分別交由不同的子模組處理。圖 21 說明了整個遊戲播放器的架構。接下來各小節，將會針對各個模組作詳細的說明。

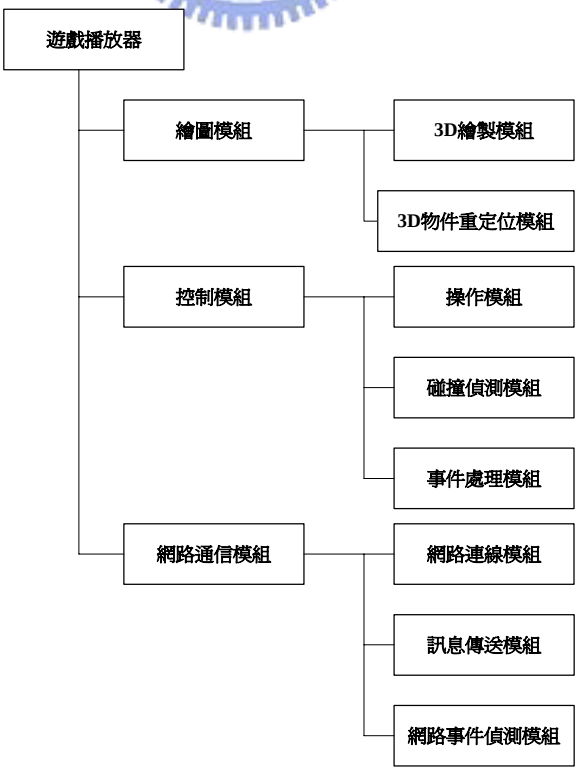

圖 21 遊戲播放器架構圖

4.4.2 繪圖模組

繪圖模組負責的部份就是遊戲畫面的繪製。同時，也負責調校整個遊戲場景中物件的大小以及位置，使得遊戲畫面可以符合手機的畫面大小。繪圖模組一共分成兩個子模組：

1. 3D 繪製模組

   此模組的功能如下所述：

   a. 負責將遊戲場景資料讀入，就是在 4.3 節所提到過由描述檔轉譯而來的資料類別。

   b. 使用 JSR184 的繪圖 API 將遊戲畫面畫出，並負責定期更新時間。

此模組的類別圖，如圖 22 所示。實際上繪製模組，就是一個使用 JSR184 所製作出來的畫圖程式。

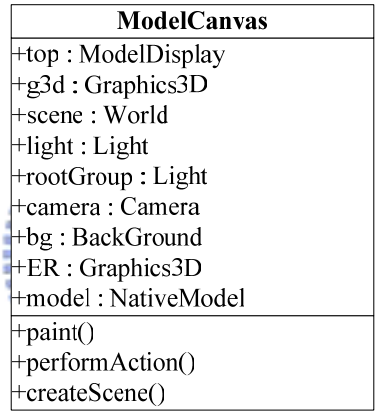

圖 22 繪圖模組類別圖

2. 3D 物件重定位模組

物件重定位的設計概念是因為使用 3D 遊戲編輯器所編輯時，所使用的 3D 素材都是以個人電腦的螢幕上看到的大小所編排的。假如將這樣大小的 3D 物件畫在手機的螢幕上，勢必變的非常巨大。所以，才會需要一個機制來重新設定畫在手機上物件的大小以及場景中物件彼此之間的相對位置。

設定物件在手機畫面中的相對位置的演算法如下所述：

a.從目前所有場景中的物件所在位置設定中，取出距離原點最遠的點。

b.以如下表示的公式重新設定物件繪製的位置：

$$(Nx,\ Nz)=\left(\frac{Ox}{Mx}*Mx\,,\ \frac{Oz}{Mz}*Mz\right)$$

Nx、Nz，代表新的 x、z 位置。

Ox、Oz，代表原本的 x、z 位置。

Mx、Mz，代表原遠點的 x、z 位置。

圖 23 為重定位演算法的示意圖。

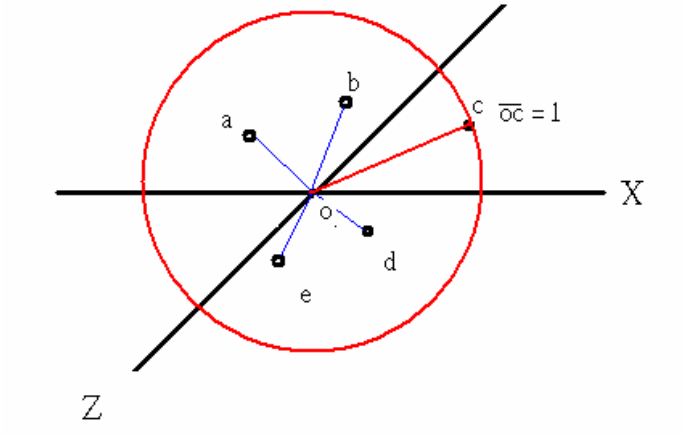

圖 23 重新計算物件位置演算法之示意圖

而重新設定物件大小的演算法不是利用上述之方式進行計算，因為一個 3D 物件的頂點數目非常的多，假如也是採用找尋最遠點，然後依照比例對物件的所有頂點計算的話，會造成程式的執行緩慢的問題。因此，在重新設定物件大小的演算法上，改用以場景地面大小的比例來縮放物件。圖 24 為此演算法的示意圖。

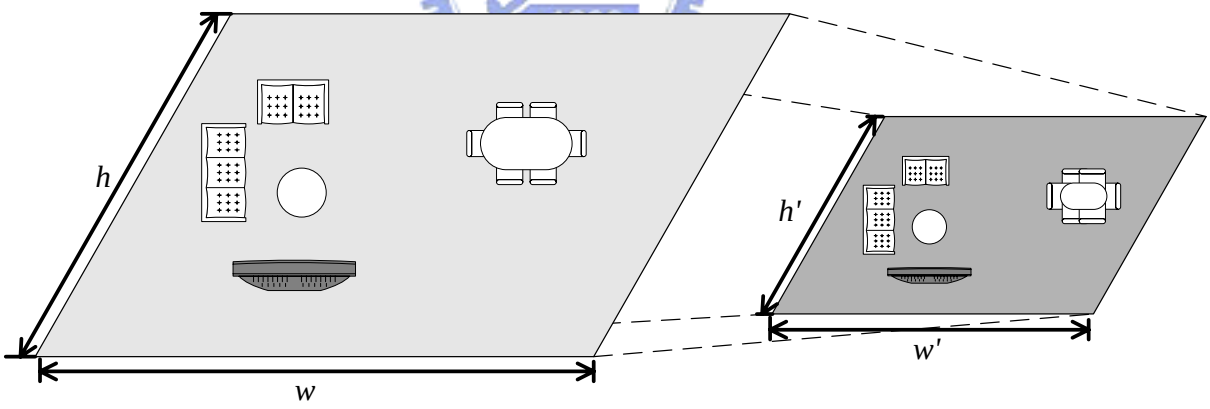

圖 24 重新設定場景物件大小演算法之示意圖

重新設定大小的演算法如下所述：

a. 從場景資訊中取得原先在 3D 遊戲編輯器中設定的地面大小。

b. 利用下面的公式來計算縮放比例：

$$R = \frac{h'*w'}{h*w}$$

R，代表計算後的縮放比例

h、w，代表在 3D 遊戲編輯器中設定的場景地面大小。

h’、w’，代表在遊戲播放器中設定的場景地面大小。

c. 將計算出來的比例，使用繪圖的 API 統一調整場景大小。

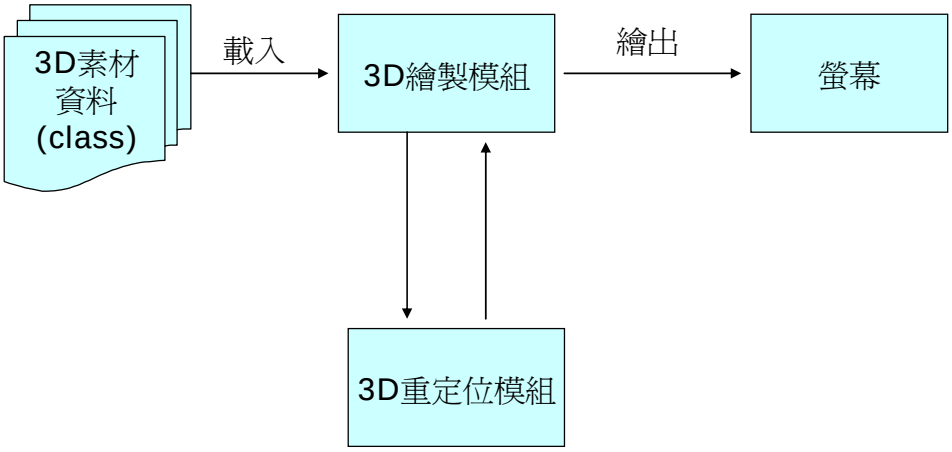

圖 25 繪圖模組運作流程圖

在介紹完繪圖模組中兩個子模組後，接著來了解繪圖模組運作的流程，可配合圖 25 來看。首先，由 3D 繪製模組將繪製遊戲畫面時需要的 3D 資料載入。接著會將這些資訊交由 3D 重定位模組重新設定各個物件的大小及位置，3D 重定位模組再將這些修改後的資訊交回給 3D 繪製模組作繪圖的動作。

4.4.3 控制模組

控制模組在整個遊戲播放器中，負責接收使用者的操作以及處裡遊戲之中發生的事件。控制模組則是再往下切割成三個子模組，分別敘述如下：

1. 操作模組

顧名思義，操作模組就是負責接受使用者的操作，並將這些操作反應在遊戲的角色身上。

本論文所製作的遊戲是以鍵盤來進行遊戲的操作，但是因為目前市面上的手機廠商對於手機鍵盤上的設計有所不同，因此在設計操作模組時，不考慮使用手機上特殊的按鈕，如發送鍵，以避免發生在某些機型上無法進行遊戲的操作。下表為系統所定義的按鈕與功能對應表。

表格 3 手機按鍵與對應之操作說明

按鍵	功　　能　　說　　明
數字鍵 2	操作角色前進。
數字鍵 8	操作角色後退。
數字鍵 4	操作角色向左轉動。
數字鍵 6	操作角色向右轉動。
數字鍵 1	當出現選單時，代表「確定」鍵。
數字鍵 3	當出現選單時，代表「確定」鍵。

2. 碰撞偵測模組

在 3D 的世界中，要確定兩個物件是否發生碰撞有許多方式可以做到，比較常見的有以下兩種方式：

a. 以物件的點之間是否產生交集來判定。

此一方式可以非常準確的判定碰撞是否發生，但是必須付出極大的運算代價，因為它是以兩個物件各自的頂點集合來作判定的運算。

b. 利用物件所佔據的範圍是否產生交集判定之。

與 a 的方式不同。並非以兩個物件的頂點集合來作判定，而是用可以包含下物件的立方體、圓柱或是圓型之範圍作為判定的基準，如下圖所示。

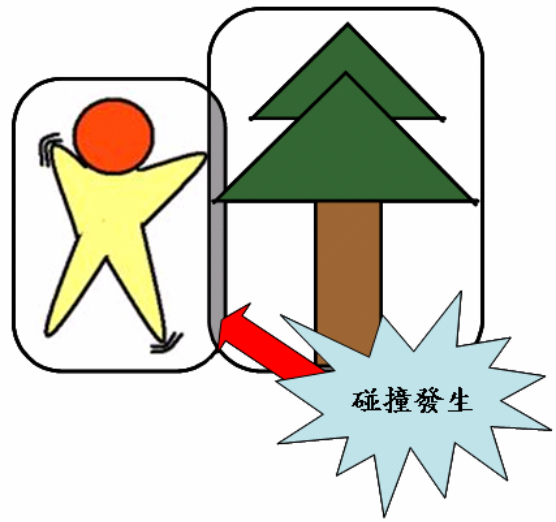

圖 26 區域式碰撞偵測示意圖

此種判定方式比使用頂點交集的方式來的迅速，也比較容易實作，缺點則是碰撞的判定比較鬆散，可能會有誤判的情況發生。

在碰撞偵測模組中，是採用 b 的判定方式來偵測碰撞。在 JSR184 中，提供類似區域式偵測的碰撞偵測函式，是使用物件的視線方向作為判定的方向，可以得知經由視線延伸出去的與碰撞到的物件之間的距離，以及碰撞到的物件為哪個物件。

如下圖 27 所示。

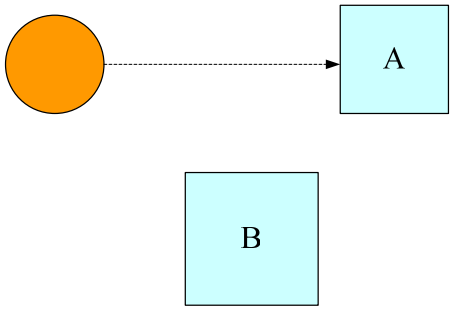

圖 27 JSR184 中提供的碰撞偵測示意圖

當碰撞偵測模組得知遊戲角色發生碰撞時，就立刻將與遊戲角色發生碰撞的物件名稱傳送給事件處裡模組，讓事件處理模組根據場景中的事件列表作相對應的處理。

3. 事件處裡模組

在事件處理模組中，儲存有經由遊戲描述檔透過轉譯器轉譯所得的事件列表。當碰撞處理模組將發生碰撞的物件名稱傳到事件處理模組後，事件處理模組依照物件的名稱找到該物件的事件，然後再依照事件的描述進行處理。

但是，在這裡的事件處理模組並不處理網路上的事件處理，因為在這邊的事件列表是從遊戲描述檔轉譯而來的。然而網路上的事件是固定，而且在遊戲描述檔中並沒有這樣的描述存在。因此，網路的事件處裡就留在網路通信模組之中再行處理。在事件處理模組所能夠處理的事件列表如下：

表格 4 在事件處理模組中能夠處理的事件類型

事件名稱	處　　裡　　說　　明
ShowMessage	在遊戲畫面上顯示訊息，作為通知玩家之用。
Speak	遊戲中玩家或是中立角色說話時所顯示的訊息。
ShowChoice	需要玩家進行選擇時，在畫面上顯示問題以及選項。

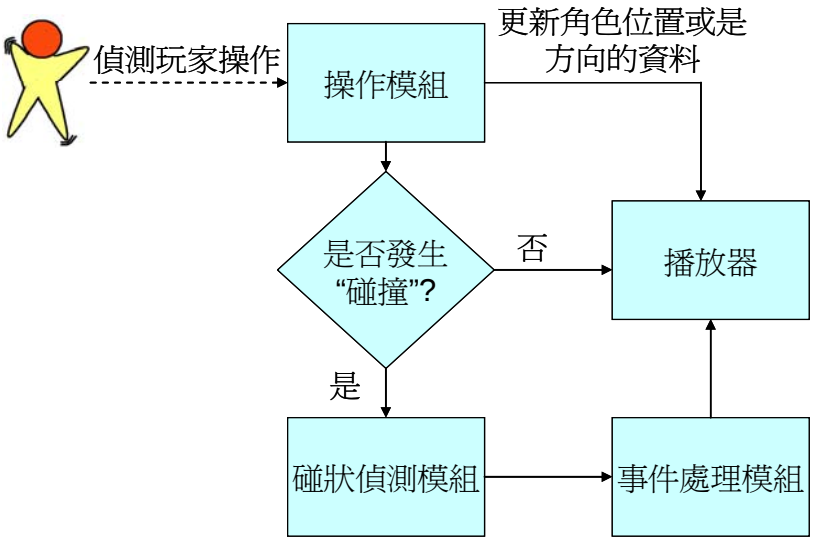

圖 28 控制模組的運作流程圖

圖 28 說明了在控制模組中三個子模組的運作流程。首先，透過操作模組偵測玩家的操作，同時將這操作的資訊交付碰撞偵測模組作碰撞的判定。假如，發生碰撞就將碰撞的資訊交給事件處理模組。事件處理模組經過搜尋找到應該要處裡的事件後，進行事件的處置。最後在交給遊戲播放器中的 3D 繪圖模組更新遊戲的畫面。

4.4.4 網路通信模組

在原本的 3D 遊戲編輯器中的遊戲播放器，是使用 DirectPlay 來進行遊戲的連線。但是，在第三章就曾經討論過此種連線方式並不適合手機，因此在本節中將會說明在新的遊戲播放器中的網路通信模組是如何做到讓兩隻手機可以做遊戲的連線。

在網路通信模組中，包含了三個子模組：

1. 網路連線模組

網路連線模組是處理比較底層的連線機能，如裝置上的設定，此模組部份會因為使用者選擇伺服器端或是客戶端而有差異。網路連線模組所負責的功能如下所述：

a. 遊戲 Server 端的建立。提供讓 Client 端可以連線的連線端口。
b. 遊戲 Client 端的建立。提供在連線範圍內的遊戲 Server 清單，以提供玩家選擇進入哪個遊戲之中。當玩家選擇完後，建立與 Server 端的連線。
c. 負責設定藍芽裝置，如初始化。

圖 29 為網路通信模組的類別圖。在圖中可看到此模組在實作上是分成兩個類別的。ModelServer 就是伺服器端的網路連線模組，ModelClient 則是客戶端的網路連線模組。

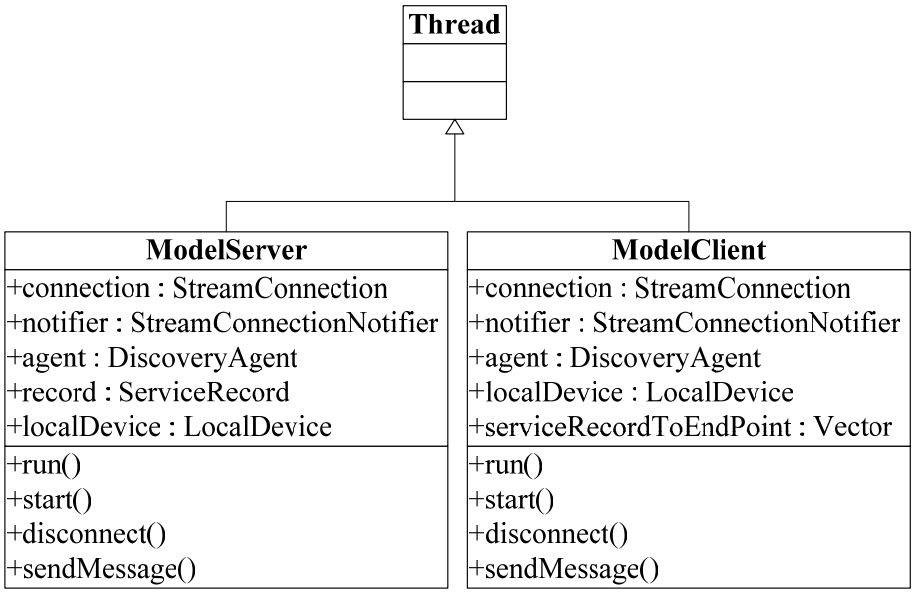

圖 29 網路連線模組類別圖

2. 訊息傳送模組

訊息傳送模組是負責傳輸遊戲的訊息以及偵測目前連線的狀態。其功能所述如下：

a. 遊戲進行時的訊息傳遞，包含玩家操作角色的位置、玩家對其他玩家要說的話。因為遊戲進行訊息會是每隔一段時間就會傳送一次，所以也可以擔負探測目前連線狀況的任務，假如發現無法傳送時，就發生網路事件，接著就是將資訊傳送給網路事件模組做處理。
b. 遊戲連線建立時的訊息，如 Server 端會收到”某玩家進入遊戲”的訊息。
c. 遊戲連線切斷時的訊息，如”某玩家離開遊戲”。

訊息的格式如下所示：

〔訊息類型〕:〔訊息內容〕

例：

〔ConMessage〕:〔Kevin〕

訊息類型如下表所列：

表格 5 訊息類型一覽表

訊息類型	說　　明
GameMessage	遊戲進行時的訊息。
ConMessage	連線訊息。
DisconMessage	斷線訊息。

3. 網路事件處理模組

在此模組中共有三種事件能夠處理：

1. 網路連線中斷

當遊戲進行訊息無法傳送時的網路事件即屬此類。當網路連線中斷時，網路事件模組會先通知網路連線模組，先依照原先連線的資訊嘗試重建連線。假若，無法重新建立，對伺服器端而言，就重新開放連線端口讓新的客戶端可以連入遊戲；對客戶端而言，則是關閉遊戲，若玩家還想繼續遊戲則必須從頭開始。圖四-19 是網路連線事件處理流程圖。

2. 玩家上線

此一事件是從伺服器端專有之事件。當伺服器端，接受一客戶端連線後，會通知繪圖模組，將客戶端的角色資料載入，然後繪製在畫面上。同時需要在伺服器端的畫面上顯示”某玩家登入遊戲”之訊息。

3. 玩家離線

此一事件是相對於玩家上線事件，只存在於客戶端中。當客戶端登入某個遊戲伺服器後，會取得目前伺服器端中另外一位玩家的角色資訊，然後通知繪圖模組更新遊戲畫面，同時顯示”登入遊戲成功”的訊息。

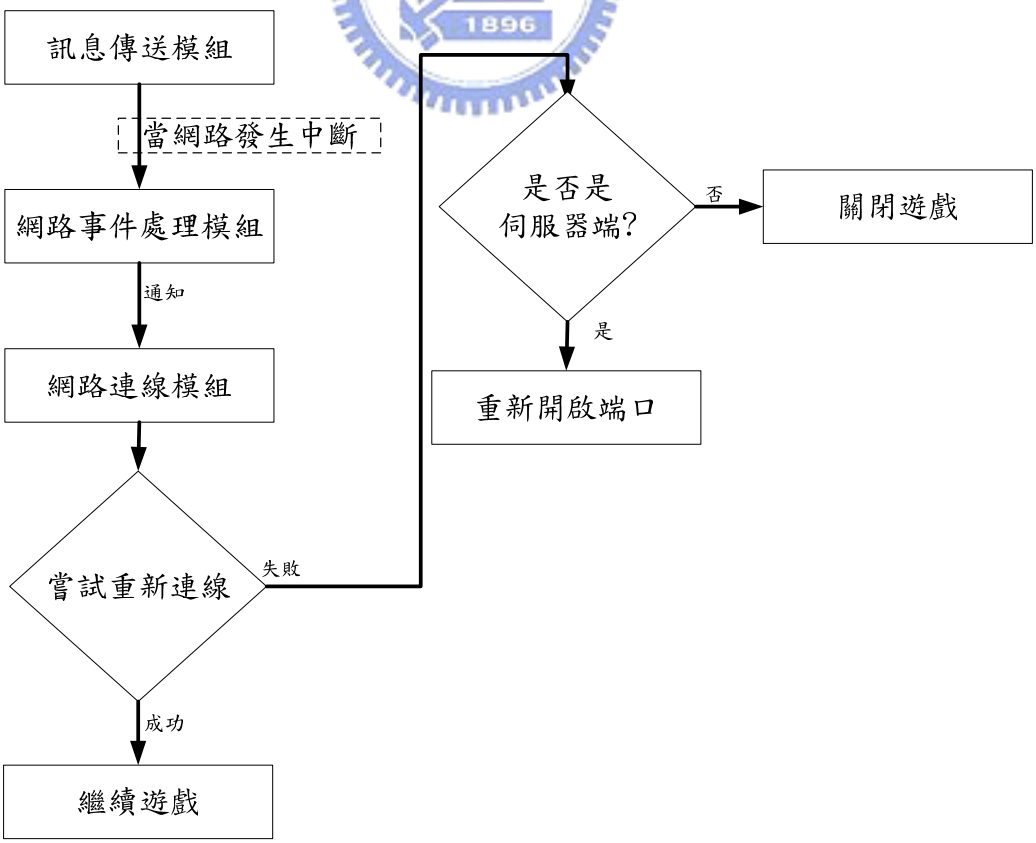

圖 30 網路中斷事件處理流程圖

至此，已經將所有遊戲播放器中的所有模組介紹完畢。接下來第五章，將會利用 3D 遊戲編輯器編輯出一個簡單的遊戲為範例，展示整個轉譯器的運作。