

目錄

目錄	1
摘要	4
致謝	5
第一章 緒論	6
1.1 研究動機	6
1.2 系統簡介	7
1.3 報告架構	7
第二章 相關研究	8
2.1 輸入端訊號處理開發環境介紹	8
2.1.1 Kinect	8
2.1.2 SensorKinect	10
2.1.3 OpenNI	10
2.1.4 NITE	14
2.1.5 KinectSDK(Beta2) [4]	16
2.1.6 CL NUI	18
2.2 輸入端訊號處理開發環境比較	19
2.2.1 CL NUI	19
2.2.2 Kinect SDK	19
2.2.3 OpenNI 與 NITE	20
2.2.4 結論	21
2.3 Kinect 人體骨架追蹤	22
2.4 遊戲開發環境介紹	26

2.4.1 XNA 簡介	26
2.4.2 XNA 五大函示介紹	27
2.4.3 物理引擎 JigLibX 簡介	28
2.4.4 JigLibX 物理的概念與原理.....	28
2.5 遊戲開發環境比較	29
2.6 技術部分的相關研究.....	30
2.6.1 Postprocessor	30
2.6.2 SkyBox.....	32
2.6.3 heightmap 地形.....	32
第三章 系統架構與流程規劃.....	37
3.1 系統概述	37
3.2 流程規劃	37
3.2.1 遊戲流程	37
3.2.2 Kinect 輸入端處理流程	37
第四章 系統設計	39
4.1 遊戲設計	39
4.1.1 遊戲介紹	39
4.1.2 遊戲規則	39
4.1.3 遊戲選單設計	40
4.1.4 遊戲關卡設計	43
4.2 遊戲開發	52
4.2.1 遊戲主要流程設計	52
4.2.2 遊戲框架與設計	53
4.2.3 遊戲畫面製作	55
4.3 輸入端 Kinect 訊號處理	71
4.3.1 Kinect 主要訊號處理流程設計	71
4.3.2 使用者姿勢校正範圍設計	74
4.3.3 使用者骨架追蹤範圍設計	75
4.3.4 使用者狀態設計	77
4.3.5 骨架資源設計	78

4.4 動作演算法設計	80
4.4.1 選單—推(Push)演算法流程設計	82
4.4.2 選單—滑動(Swipe) 演算法流程設計	85
4.4.3 選單—暫停(Pause) 演算法流程設計	88
4.4.4 關卡一逃離猛獸—跑(Run) 演算法流程設計	91
4.4.5 關卡一逃離猛獸—跳躍(Jump) 演算法流程設計	95
4.4.6 關卡一逃離猛獸—揮砍(Slash) 演算法流程設計	97
4.4.7 關卡一逃離猛獸—左右移動(Move) 演算法流程設計	100
4.4.8 關卡二飛行危機—飛行(Fly) 演算法流程設計	102
第五章 實作成果	106
5.1 開發環境建置	106
5.2 專案檔案解說	106
5.3 遊戲成果畫面	107
第六章 KINECT 訊號端程式使用手冊	112
6.1 與訊號端 Kinect 程式銜接步驟	112
6.2 Kinect 遊戲的動作訊號分類及實作	114
6.3 訊號使用及功能介紹	117
6.3.1 不分類訊號	117
6.3.2 關卡類	118
6.3.3 選單類	123
第七章 結論與未來展望	125
7.1 結論	125
7.2 未來展望	125
參考文獻	126

摘要

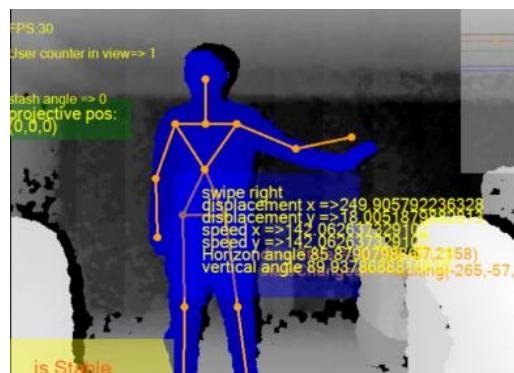
本專題透過新的體感器—Kinect，發展一個新的人機介面互動方式，期望達到不須透過任何媒介，而以身體動作控制遊戲的進行，並透過第三人稱的冒險類型的遊戲方式達到身歷其境、實際互動的效果。

輸入端部分先取得人體骨架，對體感動作定義跑、跳、左右移動、前推、手勢滑動等姿勢，並透過線性代數與圖學上的處理把人體骨架轉換至遊戲模型上與遊戲整合，控制遊戲進行；遊戲部分為了可以使擴充更彈性化，設計多管理器分別管理各項目：AudioManager、InputManager、ScreenManager、GameManager等。

本遊戲分別做出了兩關卡，進入遊戲時會先以手勢操控選擇的關卡，第一關需要透過跳躍及跑步來進行，情境模擬位在被猛獸追擊當中，須逃離至特定地點才可算過關，中途會有金幣與障礙來輔助遊戲的進行與積分累積；第二關是會有許多有洞板子向遊戲人物推過來，使用者須擺動身體來使遊戲中的人物做出一樣的動作穿過版子；由於透過身體的動作來操作，增加了趣味性與真實性，對人機的互動更友善。



圖一：實際遊戲進行畫面



圖二：輸入端處理出的畫面

致謝

此專題對我們來說是大學這幾年學習到的成果展現，因此在製作上花了許多心力下去，而在製作此的過程中，我們也學習到了很多書本以外的事物，首先要感謝指導教授張厥煒老師，對於實務專題，老師給了很多我們建議與方向，並關心我們在製作的過程中使否會偏離方向太遠，也教導了須多實務專題以外的事物，並提供實驗室的資源給我們使用，讓我們可以在好的環境下製作。

其次要感謝我們實驗室的學長姐，畢業的陳良傑學長不吝設的回來指導我們專題上所需要的技術，教導我們在製作專題這麼長的時間內該如何規劃，找尋資料學習，另外還有當時的碩士班吳貴崗學長，在撰寫論文時，還給予我們專題使用 Kinect 上的指導，以及實驗室碩士班的江文榜、吳羿柏、李宗人、謝曜駿、林家儒、陳秀紋學長姐的協助；最後要感謝我們班詹育祥、柯昆青、楊尚罡、陳至圓給予我們遊戲與動作處理上的建議，以及洪敏禎、鄭雅文、黃驥、林佳志、賴柏翰、陳昱光幫助我們遊戲完成後的測試，還有學弟妹黃翊庭、潘王翔、林鈺皓、林加好的打氣與支持。

第一章 緒論

1.1 研究動機

過去結合科技所產出的家電產品、電子娛樂商品，所設計的人機介面皆可以看到各式各樣的創意，但即便介面更加友善直覺，卻都脫離不了使用控制器操作，變成由人學習如何與「機器」溝通。

但近來的人機互動，已漸漸有了要求機器符合人的「直覺」來進行，而 2002 年的電影「關鍵報告」也帶出了這個新觀念，在當時除了給予大眾衝擊外，也帶來了新的一波研究，不過由於設備（多數使用彩色攝影機）的限制使得實現障礙過高，即使具有設備也需投入過高的成本才能實現。

而今年微軟所發表的新產品—「Kinect」體感器的誕生有戲劇性的變化，具備了攝影機所沒有的深度資訊，使辨識與影像上前處理的難度降低，且價格普及，開發的成本亦減少，加速了人機互動的發展，成為未來的新媒介，產生了更多的應用可能與新的想法。

因此我們打算使用此裝置來擴展互動應用的可能性，但回顧以往眾多的互動模式當中，遊戲才是感受性最快的呈現平台，最後決定實作以 Kinect 做為輸入端的 3D 體感遊戲，帶給使用者人機互動所帶來的趣味性及真實的感受。



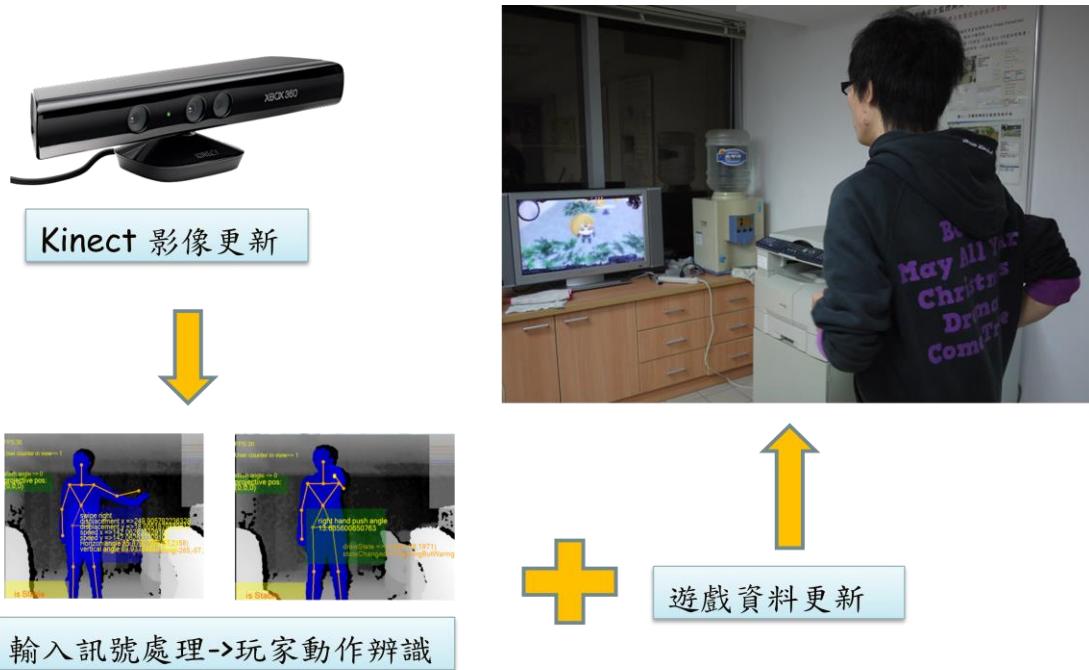
圖三：「關鍵報告」電影



圖四：微軟 Kinect 體感器

1.2 系統簡介

本遊戲分成遊戲端與輸入端，由 Kinect 體感器作為輸入端擷取影像，透過處理後再作為遊戲的輸入訊號，驅使遊戲的進行。



圖五：系統流程簡介圖

1.3 報告架構

第一章是緒論，介紹此專題的研究動機與系統的簡介。第二章是相關研究，包含了遊戲與 Kinect 輸入端在制做此專題前的一些相關資料尋找與工具研究。第三章是系統架構與流程規劃，詳細介紹專題的系統流程與架構，第四章是系統的設計，包含遊戲的規則設計、遊戲的架構設計與 Kinect 輸入端的流程設計、動作演算法設計。第五章是實作成果，介紹開發時所需要的環境、程式檔案的資料夾與類別檔的介紹，以及實作完成執行的畫面。第六章是 Kinect 的使用說明手冊，幫助想要了解或使用 Kinect 訊號端程式的使用者快速了解，銜接至自己的程式。第七章是結論與未來展望，介紹製作專題的心得，以及未來可改善的地方。

第二章 相關研究

2.1 輸入端訊號處理開發環境介紹

2.1.1 Kinect

Kinect（如圖六）是微軟在 2010 年 11 月發表的體感攝影機，他的名稱是由「運動（Kinetic）」和「溝通（Connect）」所結合而成。引用[1]，Kinect 一次可擷取三種東西，分別是彩色影像、3D 深度影像、以及聲音訊號。首先是 Kinect 機身上有 3 顆鏡頭，中間的鏡頭是一般常見的 RGB 彩色攝影機（ 640×480 30FPS），左右兩邊鏡頭則分別為紅外線發射器和紅外線 CMOS 攝影機所構成的 3D 深度感應器（ 320×240 30FPS），Kinect 主要是靠 3D 深度感應器偵測使用者的動作，而深度可以感測的距離位在 1000mm 至 4000mm。

中間視訊鏡頭則是用來辨識使用者身分（靠著人臉辨識和身體特徵）、以及辨識基本的臉部表情，此外也能應用在擴增實境遊戲、以及視訊通話時；同時 Kinect 還搭配了追焦技術，底座馬達會隨著對焦物體移動跟著轉動。

Kinect 也內建了麥克風系統（陣列式麥克風），藉由多組麥克風同時收音，比對後消除掉雜音，提供了降噪功能。

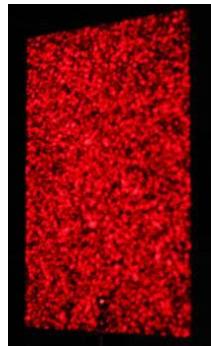


圖六：Kinect 架構圖

而最主要的部分在於 Kinect 的深度偵測技術，微軟和以色列一家名為 Prime-Sence 的公司合作，使用類似於 TOF(Time-of-Flight)的 Light Coding 技術實現，分別介紹一下這兩種技術，Time-of-Flight[2]簡單來說就是計算光線飛行的時間。

首先讓裝置發射一道光（雷射或紅外光），並在發射處接收目標物的反射光，透過之間的時間差量測距離。

至於 Light Coding[1][3]，顧名思義就是用光源照明給需要測量的空間編上碼屬於結構光技術（指的是一些具有特定模式的光，從最簡單的線、面到格狀等更複雜的形狀都有。而結構光掃描的基本原理就是將結構光投射到物體表面，再用攝影機攝取受物體表面影響的變形結構光圖像），但與傳統的結構光方法不同的是，他的光源打出去的並不是一副週期性變化的二維的圖像編碼，而是一個具有三維縱深的“體編碼”。這種光源（圖七）叫做激光散斑（laser speckle），是當激光照射到粗糙物體或穿透毛玻璃後形成的隨機衍射斑點。



圖七：激光散斑圖

這些散斑具有高度的隨機性，而且會隨著距離的不同變換圖案。也就是說空間中任意兩處的散斑圖案都是不同的，等於是將整個空間加上了標記。

只要在空間中打上這樣的結構光，整個空間就都被做了標記，再來只要針對待測空間按照一定距離取出參考平面，並紀錄每一個平面上所有的散斑，如此一來之後只要有物體在該空間中活動，我們就能透過比對散斑來定位出該物體在空間中的位置。當然，在這之前要把整個空間的散斑圖案都記錄下來，所以要先做一次光源的標定。

以往的攝影機只具備 2D 平面彩色圖，但是 Kinect 具備了 3D 的深度感測器，所以可取得距離深度圖（圖八），由於有了深度資訊，因此可以要從背景中把物體從抽離會更加容易。



圖八：左為一般影像，又為深度顯示圖

2.1.2 SensorKinect

支援 OpenNI 的 Kinect 驅動程式，他是基於 PrimeSense 官方的版本、專門針對 Kinect 修改出來的；算是 OpenNI 的一部分，要先安裝 OpenNI 後才可安裝。



圖九：裝置管理員所辨認的 Kinect 裝置

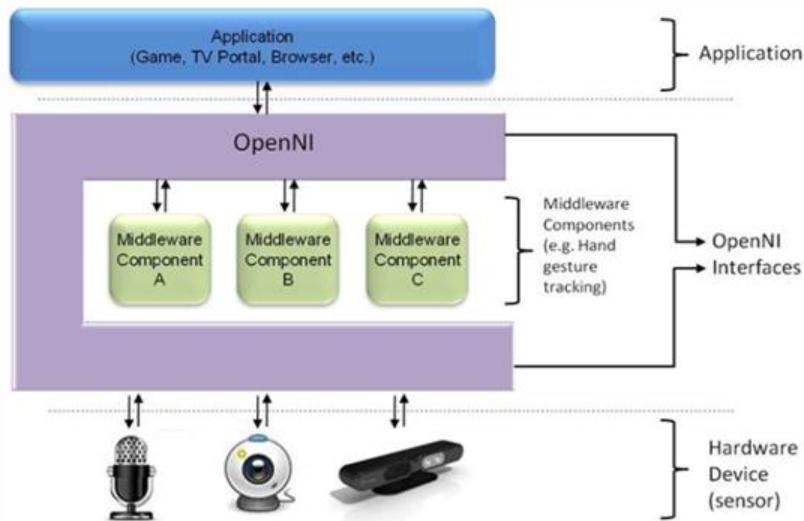
Kinect 的硬體分為三個裝置(如圖九)：Kinect Camera、Kinect Motor 和 XBox NUI Audio。其中 Kinect Camera 和 Kinect Motor 應該都可以直接找到驅動程式、並且自動完成安裝；但是目前因為沒有 XBox NUI Audio 的驅動程式，所以唯有此裝置會無法正確安裝。

2.1.3 OpenNI

OpenNI 是「Open Natural Interaction」的縮寫，大致上可以翻譯為「開放式自然操作」；而所謂的 NI 是指自然操作（Natural Interaction）的定義，包含了「語音」、「手勢」、「身體動作」等等，基本上是比較直覺、操作者身上不需

要其他特殊裝置的操作方式。

OpenNI（圖十）本身定義了撰寫自然操作程式所需要的 API，提供一個多語言（主要是 C/C++，新版新增 Java 語言）、跨平台開放原始碼架構（open source framework）；藉此提供了一個標準的介面，讓程式開發者要使用視覺、聲音相關感應器，以及對於這些資料、分析的中介軟體（middleware）時，可以更為方便。



圖十：OpenNI 的基本架構圖

上面的架構圖基本上分為三層，最上層是應用程式（Application），也就是我們程式開發者自己需撰寫的部分；最下方的一層則是硬體的部分，目前 Open -NI 支援的硬體，包含了：3D Sensor、RGB Camera、IR Camera、Audio Device 這四類。

而中間層是 OpenNI 部分，他除了負責和硬體的溝通外，也在自身內部預留了加上中介軟體（middleware）的空間，可以用來做手勢辨識、或是追蹤之類的處理。OpenNI 目前在 middleware 的部分，定義了下面四種元件：

- 全身分析（Full body analysis）

由感應器取得的資料，產生身體的相關資訊，例如關節、相對位置與角度、質心等等。

- 手部分析（Hand point analysis）

追蹤手的位置。

- 手勢偵測 (Gesture detection)

辨識預先定義好的手勢，例如揮手。

- 場景分析 (Scene Analyzer)

分析場景內的資訊，例如：分離前景和背景、地板的座標軸、辨識場景內的不同物體。

再來介紹 OpenNI 所定義的 API 部分，主要包含了兩大部分功能：節點 (Node) 與能力 (Capability)。

- 節點 (Node)

在 OpenNI 裡，他定義了所謂的「Production Node」來代表內部的基本單元，包括了硬體部分的感應器，以及 OpenNI 所提供的功能；這些 production node 分為下面三大類／層：

1. 感應器相關 (Sensor Related) Production Nodes

- 裝置 (Device) :

代表實體裝置的節點，主要是用來做這些設備的設定。

- 深度產生器 (Depth Generator) :

產生深度資訊圖 (depth-map) 的節點。

- 影像產生器 (Image Generator) :

產生彩色影像圖 (colored image-maps) 的節點。

- 紅外線影像產生器 (IR Generator) :

產生紅外線影像圖 (IR image-maps) 的節點。

- 聲音產生器 (Audio Generator) :

產生聲音串流 (audio stream) 的節點。

2. 中介軟體相關 (Middleware Related) Production Nodes

- 手勢通知產生器 (Gestures Alert Generator) :

當辨識到特定的手勢時，呼叫應用程式的 callback。

- 場景分析器（Scene Analyzer）：

分析場景，包括分離前景與背景、識別場景內的不同物體、偵測地板。

他主要的輸出會是標記過的深度資訊圖（labeled depth map）。

- 手部位置產生器（Hand Point Generator）：

支援手部偵測與追蹤，當偵測到手、或追蹤手的位置時，會產生一個通知訊息。

- 使用者產生器（User Generator）：

產生一個 3D 場景中完整、或局部的身體資訊。

3. 錄製／撥放

- 錄製器（Recorder）：用來記錄資料用的。

- 撥放器（Player）：讀取記錄下來的資料，並撥放出來。

- 編解碼器（Codec）：用來壓縮、解壓縮紀錄資料。

- 能力（Capability）

OpenNI 的「Capability」機制是用來增強中介軟體和硬體裝置的彈性的；這些不同的能力都是非必要性的，各家廠商所提供的不同的中介軟體和裝置，可以自己決定要提供那些能力。而 OpenNI 則是負責定義好一些可以使用的 capability，讓程式開發者可以快速地找到符合自己需求的中介軟體或裝置。

而目前版本的 OpenNI 所支援的 capability 則如下：

- 替換視角（Alternative View）

讓各類型的 map generator（深度、影像、紅外線）可以轉換到別的視角，就好像攝影機在別的位置一樣。這個功能可以快速地替不同的感應器產生的內容作對位。

- 裁切（Cropping）

讓各類型的 map generator（深度、影像、紅外線）輸出結果可以被裁切、

降低解析度；例如：VGA 可以裁切成 QVGA。這對效能的增進很有用。

- 畫面同步（Frame Sync）

讓兩個感應器產生結果同步化，藉此可以同步取得不同感應器的資料。

- 鏡像（Mirror）

把產生的結果鏡像（左右顛倒）。

- 姿勢偵測（Pose Detection）

讓「使用者產生器（User Generator）」可以偵測出使用者特定的姿勢。

- 骨架（Skeleton）

讓「使用者產生器（User Generator）」可以產生使用者的骨架資料。包含骨架關節的位置、並包含追蹤骨架位置和使用者校正的能力，且可追蹤多人骨架。

- 使用者位置（User Position）

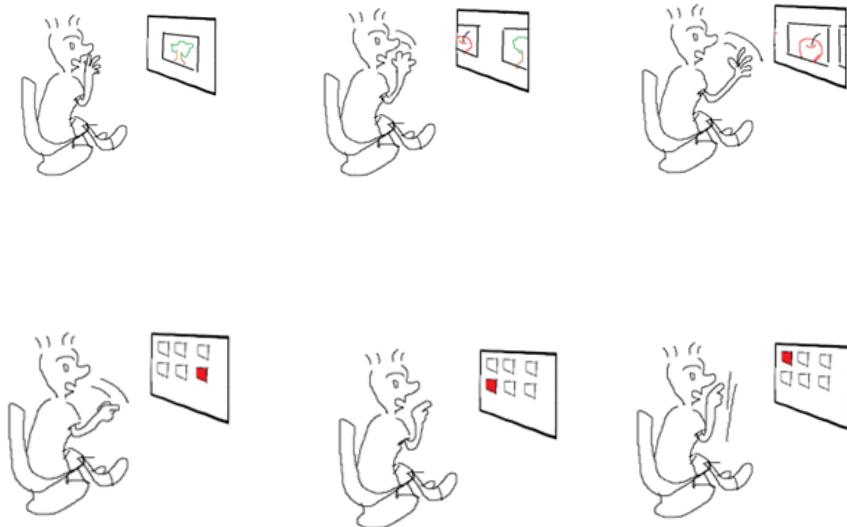
讓「深度產生器（Depth Generator）」可以針對指定的場景區域、最佳化輸出的深度影像。

- 錯誤狀態（Error State）

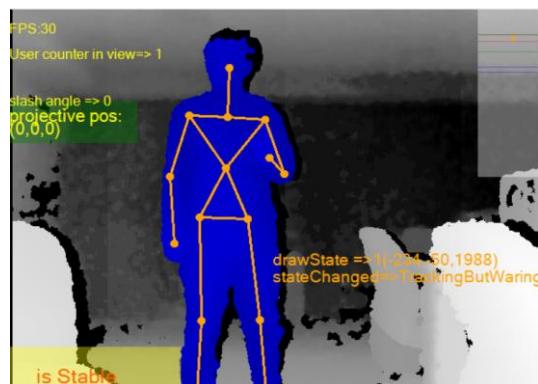
讓節點可以回報他本身的錯誤狀態。

2.1.4 NITE

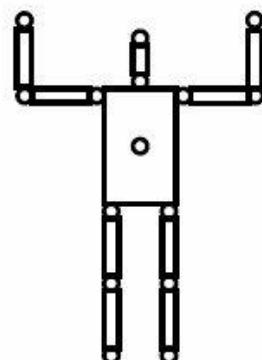
目前 PrimeSense 公司 已經提供了一套 NITE 當作最主要的 middleware、提供上面所列的功能。主要定義上述 middleware 中講述的元件，其中手勢偵測（圖十一，部分示意圖）包含了 wave、push、swipe、circle（手腕畫圓）、steady（手靜止不動）、selectSlider1D/2D，此外還多了骨架追蹤（圖十二、圖十四）的實作演算法。



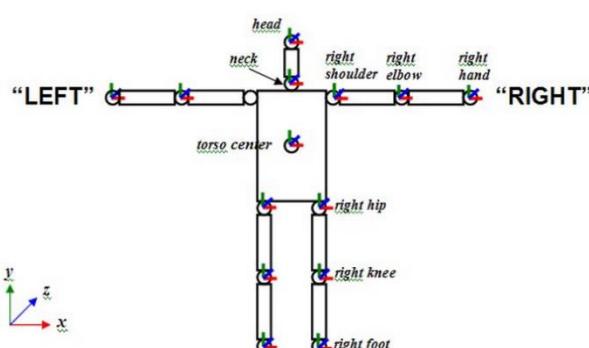
圖十一：上半部分為 swipeRight 示意圖，下半部分為 selectSlider2D 示意圖



圖十二：骨架追蹤



圖十三：Psi 姿勢



圖十四：可抓取的骨架關節點（三維座標與方向向量）－15 個

目前 NITE 最新版本 1.5 在追蹤骨架時，已不需要做校正姿勢，而在先前的版本時需要透過擺出 Psi（圖十三）才能追蹤出骨架。

2.1.5 KinectSDK(Beta2) [4]

此為微軟官方釋出的 SDK，目前已到 Beta2，而前面敘述的 openNI、Sensor-Kinect、NITE 是另一套組合，屬於半官方釋出（由製作 Kinect 技術製作的 Prismesence 公司與 OpenNI 組織所推動）。

目前 Kinect SDK Beta 僅能用在 Windows 7 的電腦上、不能在其他版本的 Windows 上執行；而微軟有同時提供 32 位元和 64 位元的版本，裡面包含了：

- Kinect 在 Windows 7 上所需的驅動程式(包含四個裝置:Microsoft Kinect Audio Array Control、Microsoft Kinect Camera、Microsoft Kinect Device、Kinect USB Audio)
- Application programming interfaces (APIs)、裝置介面 (device interface) ，以及技術文件
- 範例原始碼

另外，安裝 Kinect SDK 時會一同把驅動程式安裝進去，不過由於目前各種平台的 Kinect 驅動程式都不相同，所以如果使用微軟官方的 SDK 的話，建議先移除其他版本的驅動程式，此外，雖然微軟有提供 32 位元和 64 位元兩種版本，但是主要是針對驅動程式的部分；目前的 Kinect SDK Beta 版僅只有支援建置 x86(win32)、也就是 32 位元的程式，而新的 Beta2 則是也可以支援 x64。

而在系統需求方面，硬體上需要：

- Kinect
- 雙核心 2.66-GHz 或更快的處理器
- 支援 DirectX 9.0c、相容 Windows 7 的顯示卡
- 2GB 記憶體（建議 4GB）

軟體的部分，則需要：

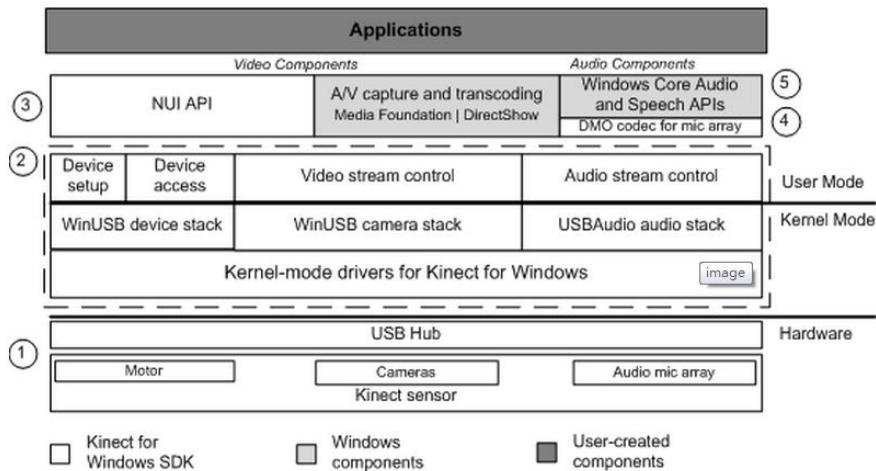
- Windows 7 (x86 or x64)
- Visual Studio 2010 (Express 版或其他版本)
- Microsoft .NET Framework 4.0

- 若需要編譯的範例程式，也還需要其他的東西，例如 [Microsoft DirectX® SDK](#)、[Microsoft Speech Platform SDK](#)、[Kinect for Windows Runtime Language Pack](#)（詳細請參考 [readme](#)）

透過 Kinect SDK，開發者可以在 Visual Studio 2010 裡使用 C++、C# 或 VisualBasic 進行開發，並且可以：

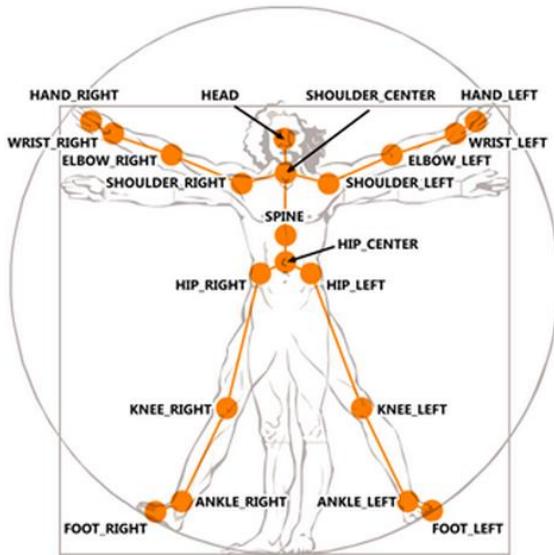
- 讀取 Kinect 感應器的原始資料（raw data streams、包括了深度感應器、彩色攝影機以及四個單元的麥克風陣列）。
- 追蹤一或兩個人的人體骨架，並根據骨架資料進行分析。
- 進階聲音處理，包含了聲音雜訊的抑制、回音的消除以及聲音來源方向的判斷，同時也可以整合 Microsoft Speech 來做語音辨識。

根據《Kinect SDK Reference》和《Programming Guide》（[PDF](#)）的內容來看的話，在 SDK 的內部（圖十五），他是把功能分為 NUI API 和 Audio API 兩個部分。前者主要就是視訊（深度、彩色影像）、骨架以及 Kinect 角度控制（Kinect 上是有馬達的）的部分，而後者，則是聲音的部分。



圖十五：Kinect SDK 架構圖

Kinect SDK 也提供了骨架追蹤與抓取骨架節點（圖十六），能抓到 20 個關節，在手腕、腳踝的部分，也都能獨立抓出來。



圖十六：Kinect SDK 所提供的骨架節點（三維座標）－20 個

Kinect SDK 並未於此實務專題中使用到，原因會在下面 2.4 小節提到，更多關於 Kinect SDK 的資訊請去官方網站搜尋。

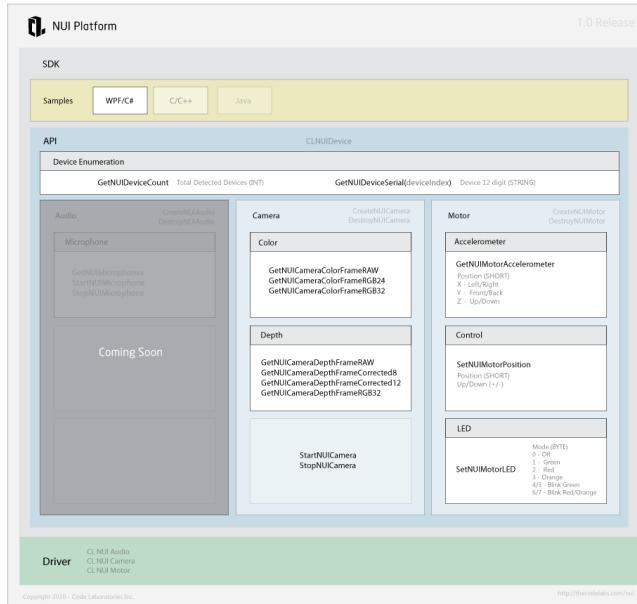
2.1.6 CL NUI

在官方 Kinect SDK 未釋出前，當時有許多的網路上網友試著突破解 Kinect 裝置，並撰寫其驅動程式與 API，而 CL NUI 即是其中之一，之後 OpenN 系列的崛起與 Kinect SDK 的發布關係，此 API 已無再更新，不過在當時是有許多人開發使用。

以下為此 API 的特色[5]：

- Support for Windows XP/Vista/Win7 both x86 and x64
- Support multiple Kinect cameras
- Device serial number retrieval
- RAW, RGB24 and RGB32 Color frame data
- RAW, Corrected8, Corrected12 and RGB32 Depth frame data
- Full 30fps camera data streaming
- Reading of built-in Accelerometer x, y, z data
- Camera motor control

- Camera LED control
- Fully signed driver
- To get you started we have included sample source code written in C# as well as C/C++ SDK lib and header files



圖十七：CL NUI 架構圖

2.2 輸入端訊號處理開發環境比較

2.2.1 CL NUI

先從 CL NUI 說起，首先 CLNUI 已無再更新，所以歡迎度與穩定度不高，再來 CL NUI 本身並未支援骨架追蹤與骨架資料擷取，亦無手勢偵測功能；支援平台部分，包含 Windows XP/Vista/Win7 both x86 and x64，但不支援 Linux 作業環境，所以不考慮在內。

因此，以下針對 Kinect SDK 與 OpenNI 系列作比較[6]。

2.2.2 Kinect SDK

- Kinect SDK 優點：
- 安裝簡易，安裝完後同時驅動與開發 API 皆裝入
- 支援音頻與基座馬達，可以辨識出聲音（目前以英文為主），可作音訊

方面的互動，且可使用基座馬達轉動 Kinect

- 骨架追蹤穩，使用的骨架演算法具有大量的樣本，穩定度高且不需擺出姿勢
- 骨架節點多，和 OpenNI 相比，可抓取節點多出了 5 個，且能抓取到手腕及腳踝關節
- 多感應器支持，可支援多台 Kinect，且撰寫步驟易，Beta2 版亦可指定哪一台 Kinect 去抓取骨架
- 穩度高與持續更新，且有教學文章與影片
- Kinect SDK 缺點：
 - 非商用，若需商用要付費
 - 只能追蹤全身（無特定追蹤部位）
 - 關節資訊只有座標，無方向性與穩定度（openNI/ NITE 具備）
 - 無內建手勢辨識與手勢偵測
 - 不支援紅外線資訊
 - 只支援 win7 (C++/C#)
 - 無使用者狀態事件，如使用者進入 Kinect 視線或離開視線
 - 不支援影片資料紀錄與播放
 - 不支援 Unity3D 遊戲引擎(OpenNI/NITE 可)

2.2.3 OpenNI 與 NITE

- OpenNI /NITE 優點：
 - 可商用化，不需付費，且有再持續更新，具有一定的穩定度。
 - 有手勢辨識與偵測框架，可使用內建的手勢偵測與辨識，若在給予框架下開發
 - 骨架追蹤資訊，且可半身或全身，且骨架資訊可取得關節三維座標、向量方向性與關節的穩定度

- 支援 win7/vista/xp、linux、Mac、Android 平台環境(跨平台)
- 語言 C++/C#/Java
- 支援紅外線數據，可取得紅外線的資訊
- 使用者的狀態事件提供多，如：使用者進入、離開、遺失.....
- OpenCV 提供部分函釋庫搭配使用
- OpenNI /NITE 缺點：
 - 不支援音頻與基座馬達
 - 骨架節點不多，只包含 15 個
 - 骨架追蹤需擺出姿勢才可以追蹤(目前最新版 NITE 演算法已改良，不須擺出姿勢)
 - 追蹤穩定度不如 KinectSDK 高
 - 支援多感應器，不過撰寫步驟稍多
 - 安裝步驟多，須先安裝 OpenNI 後，再安裝 SensorKinect 驅動，若需骨架資訊、更多的手勢偵測與辨識，再安裝 NITE。

2.2.4 結論

排除 CL NUI 後，以 Kinect SDK 來說，若是需要用在遊戲上的開發（不過或商業化需要付費），或是想要使用骨架部分來說，會是首選的推薦；但是若使想要投入再研究部分、或是想使用別的平台與語言(Linux、Andriod、Mac 平台、Java 語言)則可以使用 OpenNI 系列，此外由於提供的功能較多，所以可以對於骨架或手勢需要的資訊來說，可變動的地方較多。

當時製作專題時間，Kinect SDK 屬於最晚推出的（當時為 Beta 版），因此並未考慮上去，而使用 OpenNI 系列，而 OpenNI 系列所能做到的功能也較多符合需求。

2.3 Kinect 人體骨架追蹤

OpenNI/NITE 的骨架追蹤與辨識部分，是透過深度攝影機偵測與捕捉人體動作後，再將捕捉到的影像與內部晶片（PrimeSense PS1080）內部存有的身體模型比對，每一個符合已存身體模型的物體就會被創造成相關的骨骼模型。

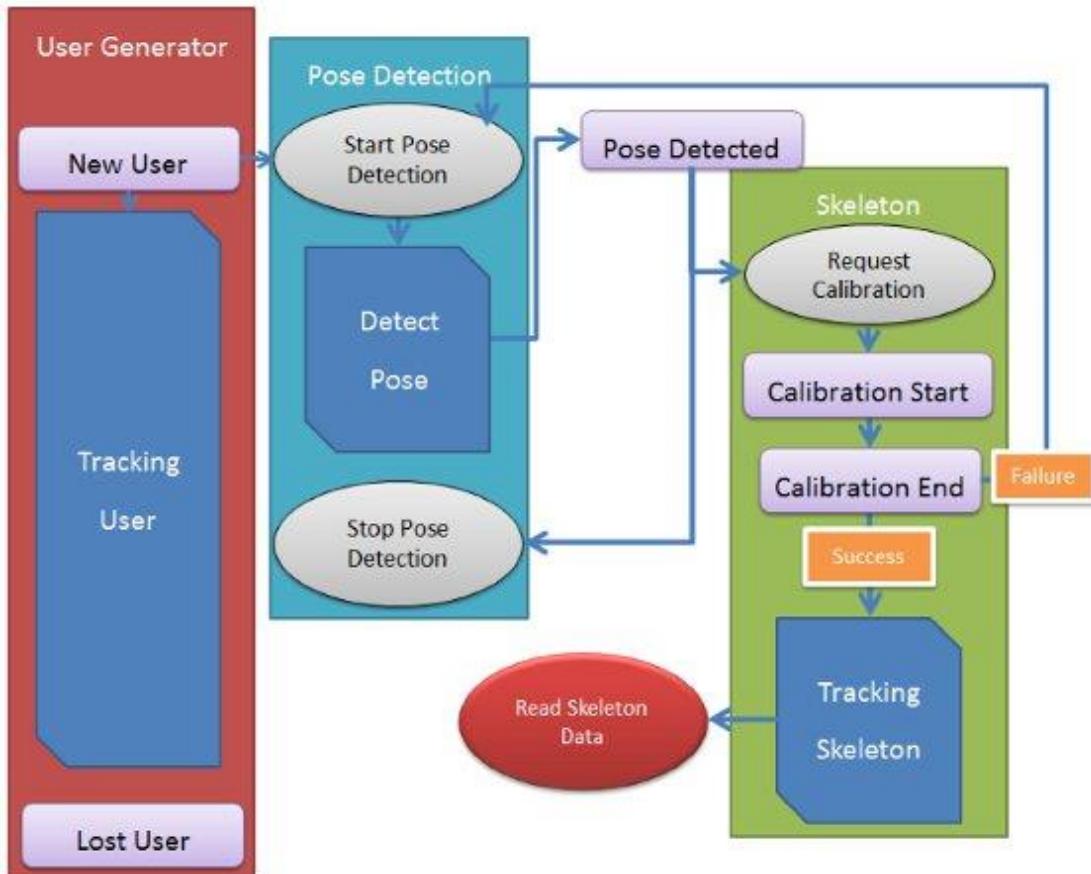
由於是使用 OpenNI/NITE，提供的功能多，分解成較多部分，所以在撰寫骨骼追蹤時也有較多的步驟須做具有幾項步驟：

首先回想 2.3.3 小節所介紹的 OpenNI 兩大功能—「節點」與「能力」，在此專題中節點部分使用了 Device、Depth Generator、User Generator，透過這些節點來生產資料類型；而能力部分則使用了 Mirror、Pose Detection、Skeleton、User Position 這些能力。

有了上述的生產節點與能力後，再去尋找 Kinect 的原始資料(Raw Data)中可能是人體的移動物體，並使用分割(Segmentation)法將人體從背景中切離，也就是使每個被追蹤的人體，在景深影像中建立分割遮罩，以分割遮罩把背景物體從景深影像中刪除[7]。

分割化的人體影像的每個像素都被傳送近一個辨別人體部位的機器學習系統中，Kinect 會對景深影像進行評估，隨後該系統會給出某個特定像素屬於身體部位的可能性，來辨別人體的不同部分。

OpenNI 偵測人體的地方是利用 Depth Generator 去產生 User Generator，再用 User Generator 去產生 Pose Detection、Skeleton、User Position 這些能力，就可以偵測到人體骨骼，主要步驟流程[4]（圖十八）如下：



圖十八：骨架追蹤流程圖

1. 以事先定義好的 OpenNI XML 檔去啟動並配置 Kinect 硬體資源。
2. 要能夠在 OpenNI 的環境裡建立人體骨架，基本上是要靠所謂的 User Generator；而由於他的資料來源是深度影像（depth map，圖十九），所以要使用的話，要先建立一個可用的 Depth Generator 出來才行。
3. 再來，偵測到骨架前，要先能偵測到人體，因此建立 User Generator，並透過 User Generator 的事件機制，註冊 New User 事件與 Lost User 事件，New User 會在畫面內偵測到新的使用者時被呼叫（圖二十）、Lost User 會在使用者離開可偵測範圍一段時間時被呼叫。而在 OpenNI1.3.2 版中則另外新增了 User Exit（使用者離開畫面）事件與 User ReEnter 事件（使用者從新進入畫面）。
4. 再來建立 Pose Detection 能力，並註冊 Pose Detection 的 Pose Detected 事件，當 User Generator 的 New User 事件觸發後，則代表有使用者進入，

此時再呼叫 Start Pose Detection 的功能，並開始追蹤有擺出 Psi 校正姿勢的使用者（此能力在 NITE1.5 版已不需要）。建立 Pose Detection 本身並不會啟動姿勢偵測。

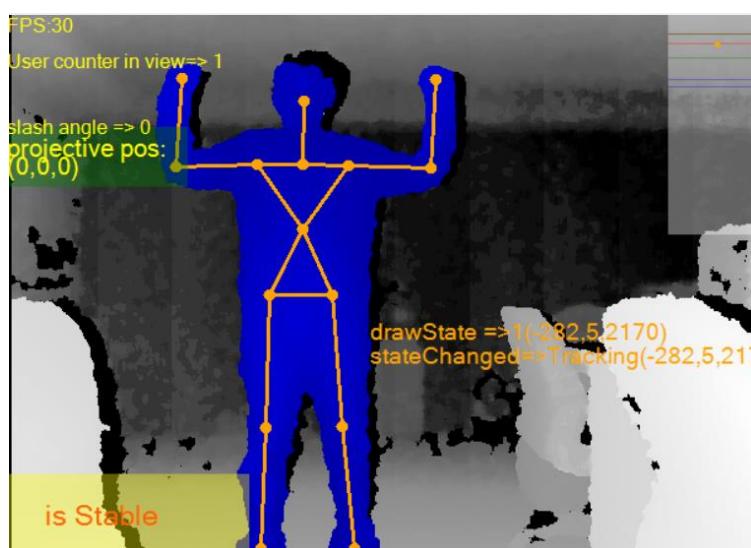
5. 當 Pose Detection 判斷出此姿勢後會就會觸發 Pose Detected 事件，並呼叫 Skeleton 能力 的「Request Calibration」函式，要求 skeleton 開始進行人體骨架的校正、分析。
6. 在 Skeleton 能力的「Request Calibration」被呼叫後，Skeleton 就會開始進行骨架的校正、分析。當開始進行骨架校正的時候，Skeleton 會去觸發「Calibration Start」事件，讓程式開發者可以知道接下來要開始進行骨架的校正了，如果有需要的話，可以註冊此事件，並在這邊做一些前置處理；而當骨架校正完後，則是會去觸發「Calibration End」事件處理。
7. 當「Calibration End」被呼叫的時候，只代表骨架的校正、辨識的階段工作結束了，並不代表骨架辨識一定成功，也有可能是會失敗的。如果成功的話，就是要進入下一個階段、呼叫 Skeleton 的「StartTracking」函式，讓系統開始去追蹤校正成功的骨架資料；而如果失敗的話，則是要再讓 pose detection 重新偵測校正姿勢，等到有偵測到校正姿勢後，再進行下一次的骨架校正。
8. 而在骨架校正成功、並開始進行追蹤骨架後，之後只要呼叫 Skeleton 用來讀取關節資料的函式（如 GetSkeletonJoint），就可以讀取到最新的關節相關資訊，並建立整個人體的骨架資料（圖二十一）。



圖十九：深度影像



圖二十：使用者進入，偵測人體



圖二十一：擺出 Psi 校正姿勢，偵測到骨架

2.4 遊戲開發環境介紹

2.4.1 XNA 簡介

XNA 是 Microsoft 於 2006 年底所提出來的 GameStudio，提供很多簡易的功能函數，讓即使是第一次設計遊戲的使用者也可以輕易上手。以往要設計遊戲，通常使用 DirectX 或者 OpenGL 的 API(Application Program Interface)為主。這兩種繪圖 API，大多使用 C/C++語言。然而，很多第一次接觸遊戲程式設計的使用者，對於 C 或 C++程式語言中的指標、資源分配等問題並不熟悉，造成遊戲設計上的困難。

XNA 所使用的程式語言為 C#，大幅地降低指標的使用，以及資源分配問題的發生。XNA 更便利於以往的 DirectX 的另一項重大原因，就是 XNA 提供了基本遊戲功能，在 XNA 的 Framework 中，會幫使用者畫好初始繪圖介面、輸入界面和音效介面，程式撰寫人員只需要藉由 XNA Framework 往外擴充，加入一些自己所想要的功能即可。

XNA 的優點

- 免費的開發工具與平台

XNA Game Studio 與 VC C# 2008Express 都可以免費取得。

- XNA Framework 會幫開發者處理許多細節方面的問題

開發者可以更專注於遊戲的內容，不必處理一堆引擎而花費大量的時間。

- 可以透過改寫 Framework 中的 Content Pipeline 達到支援其他格式的內容。

如果要開發 XNA 遊戲需要怎樣的 PC？

硬體方面：

- 具有支援 Shader 版本 1.1 版或以上的 3D 加速顯示繪圖晶片
- 至少 1Ghz 以上的 CPU

- 至少 192MB 的記憶體

軟體方面：

- XP 含以上的作業系統
- .Net Framework 2.0 或更新的版本
- DirectX 9.0c 或更新的版本
- XNA Framework27

2.4.2 XNA 五大函式介紹

XNA 另外預先設置好五大函式，構成了 XNA 的遊戲基本歷程，開發者利用這些函式，可以讓遊戲開發過程更加簡化。其函式內容如下：

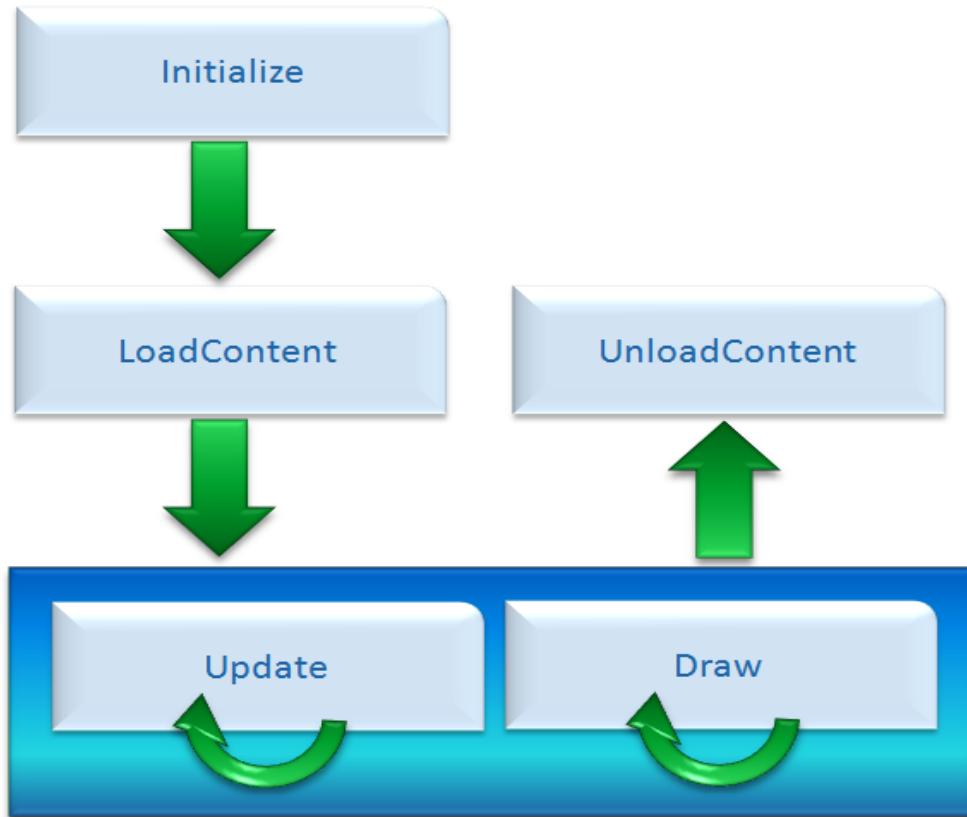
Initialize(): 用來初始化一些物件、屬性、變數等設定。

Load Content(): 負責載入遊戲中出現的資源如影像、模型、音效等。

Update(): 主處理、計算或做些程式判斷等動作。

Draw(): 將所有物件納入並顯示到螢幕上。

Unload Content(): 將所有 Load Content() 中載入的處理做卸載並釋放記憶體。



圖二十二：XNA 五大 Function

上述函數當中，只有 Initialize 式執行一次，其他函數皆有可能在遊戲執行中呼叫一次以上。

2.4.3 物理引擎 JigLibX 簡介

JigLibX 是一個使用 C# 撰寫且使用微軟的 XNA 遊戲框架所建構出來的物理引擎。它是源自於 C++ 上的物理引擎 JigLib，目前正在被移植和擴展。有一個碰撞系統和剛體的物理引擎，目前市面上 JigLibX 為好用的免費 XNA 之開放源碼物理引擎之一，圖二十三。

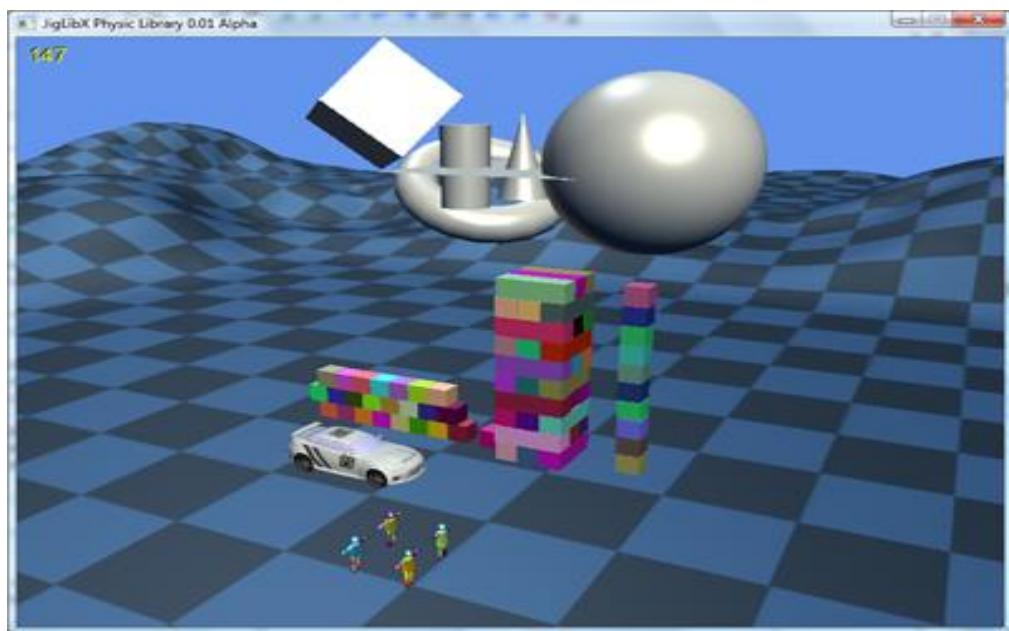
2.4.4 JigLibX 物理的概念與原理

剛體動力學模擬是一種用於在遊戲中給的“物理”行為對象的外觀。對於每個要模擬的對象，擁有一個或多個單位的定義，每一個單位都有一個或多個碰撞元

素。超過一個可移動的部分（例如，一個槓桿的支點）的東西，是一個 body 的每一塊，和 body 繩在一起使用關節。拉鋸的一個操場上，會與碰撞的皮膚為基礎，另一種與另一個身體的一部分，實際上移動的碰撞皮膚有一個 body。

碰撞元是基本形狀，如盒子，球體，膠囊體，射線/線段，以及稍微更複雜的形狀，如平面圖、高度圖和三角網格。

描述一個給定的 body（車、人物等）的大致輪廓，使用者可能想使用一個以上的形狀，並把它們放在一起。您可以通過建立一個碰撞皮膚和分配 body。注意：所有在碰撞皮膚的形狀一起移動，並與身體移動；如果想的不同部位，分別移動（門上車，在一個鏈中的鏈接），那麼要創建多個單位，其中每個人都有不同的碰撞皮膚，將它們連接在一起使用某種形式的聯合。事實上，每個單位都有一個固定的，這種模擬被稱為“剛體動力學。”



圖二十三：JIGLIBX 之模擬畫面

2.5 遊戲開發環境比較

	XNA	Untiy 3D	OGRE
Coding 難度	中上	非常低	高

開發效率	高	高	低
第三方資源	非常多	少	中
穩定度	高	高	低
可以學到的東西	很多	少	很多

表一：遊戲引擎比較表

	JigLibX	BulletX	Oops! Framework
完整性	高	中	中
效能	高	低	低
持續更新	有	無	無
穩定度	高	中	高

表二：物理引擎比較表

2.6 技術部分的相關研究

2.6.1 Postprocessor

概念：

Postprocessora 為在最終畫面上添加模糊效果，或給場景中的物體添加光澤。

實作原理：

透過計算每個像素和一些周圍像素的顏色值平均值並用這些值替換像素的原始

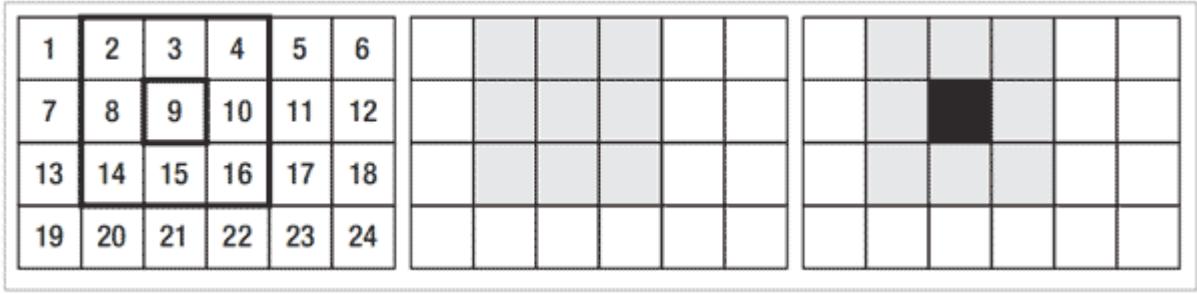
顏色實現模糊效果。如圖二十四中的格子是一張需要模糊的圖像，該如何計算像素 9 的最終顏色呢？

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24

圖二十四： 計算 2D 紹理中的像素顏色的平均值

方法是可以透過將 9 個數字相加並除以 9 得到這 9 個數字的平均數，計算顏色平均值的方法是一樣的：將 2-4 像素、8 到 10 像素、14 到 16 像素的顏色相加並除以像素的數量(此為以 9 為範例)。

如果除了像素 9 之外其他像素的顏色都是白色的，這會導致像素 9 的顏色擴展到像素 2 到 4，8 到 10 和 14 到 16，使像素 9 “擴散”了。但是，像素 9 本身的顏色變暗了，因為它也被周圍顏色的平均值替換了，如圖二十五中圖所示。要獲取一個發光效果，要混合原始圖像，在這個圖像的某處存儲了像素 9 的原始顏色。所以，最後像素 9 擁有自己的原始顏色，而像素 9 周圍的像素擁有一點像素 9 的顏色，就好像像素 9 有了一個發光效果，如圖二十五的右圖所示。



圖二十五： 創建一個發光(glow)效果

2.6.2 SkyBox

在遊戲中添加一個場景，使畫面更加生動，而不是一個單一顏色的背景。

概念：

只需簡單地繪製一個很大的立方體，在立方體內部的六個面上貼上風景或房子內部的紋理，然後把相機放在立方體的內部即可。

實作原理：

有幾種方法可以將一個天空盒放置在場景中，最簡單的方法是加入一個模型檔案，這個 3D 檔擁有自己的紋理和 effect。但是本專案想獲得更多對場景的控制權，依此自己定義一個立方體。下面舉立了一個 HLSL 範例，介紹如何使用單張 TextureCube 紹理實現天空盒的方法，而這個紋理使用了 HLSL 的 texCUBE 內置函數。

以上兩個技術相同點是：

- 立方體必須總是繪製在相機周圍，而相機應該總處於立方體中心。只有這樣才能讓玩家在移動相機時，保持相機和立方體之間的距離不變，使場景給人一種在無窮遠處的印象。
- 當繪製天空盒時應該讓 Z buffer 不可寫，這樣就無需縮放天空盒以適應場景。在繪製完天空盒後重新使 Z buffer 可寫入即可。

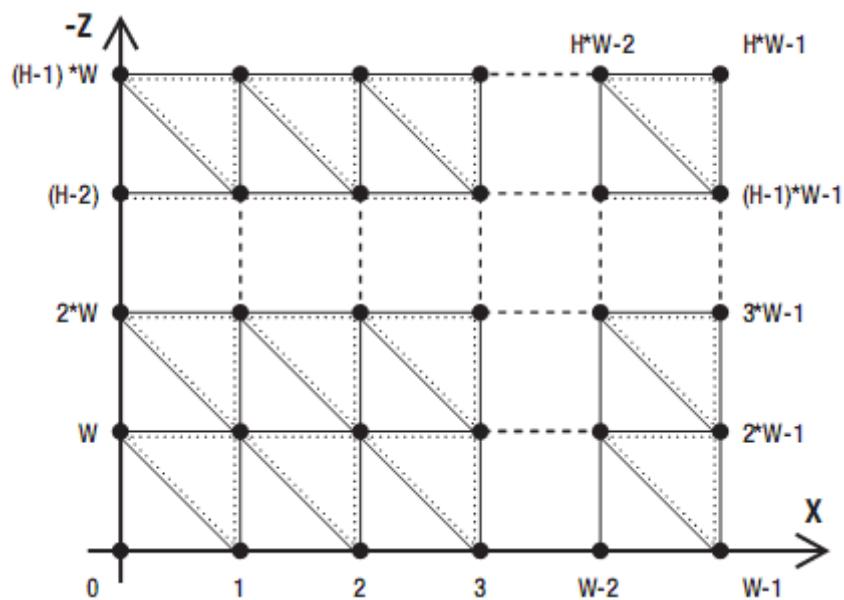
2.6.3 heightmap 地形

基於一張 2D 高度圖，創建一個 XNA 地形並以一個有效率的方法繪製它。

概念：

首先需要一張高度圖，包含所有用來定義地形的高度數據。這張高度圖有確定數量的二維數據點，我們稱之為地形的寬和高。

顯然，如果想基於這些數據創建一個地形，這個地形將從這些 width*height 頂點中繪製，如圖二十五右上角所示(從 0 開始)。



圖二十六：地形網格

要使用三角形完全覆蓋網格，需要在每個網格的四個點中繪製兩個三角形，如圖二十六所示。一行需要 $(width-1)*2$ 個三角形，整個地形需要 $(height-1)*(width-1)*2$ 個三角形。

(三角形數量) 除以(頂點數)小於 1，所以必須使用索引。所有不在邊界上的頂點會被不少於六個的三角形共享。

因為所有三角形至少共享一條邊，所以使用 TriangleStrip 而不是 TriangleList。

實作原理：

1. 定義頂點：

首先定義頂點。下面的方法首先訪問 heightData 變量，這個變量是一個包含地形所有頂點高度的 2 維數組。

再來基於 heightData 數組的大小獲取地形的高和寬。然後創建一個數組保存所有頂點。如前所述，地形需要 width*height 個頂點。

接著在兩個迴圈中創建所有頂點。裡面的一個迴圈創建一行上的頂點，當一行完成後，第一個 for 迴圈切換到下一行，直到定義完所有行的頂點。

使用 X 和 Z 坐標作為迴圈的計數器，Z 值是負的，因此地形是建立在向前(-Z)方向的。而高度信息取自 heightData 數組。

現在給與所有頂點一個默認的法線方向。因為可能要在地形上加上紋理，所以需要指定正確的紋理坐標。根據紋理，想控制它在地形上的大小。在此將除以 30，表示紋理每經過 30 個頂點重複一次。如果想增大紋理匹配更大的地形，可以除以一個更大的數值。

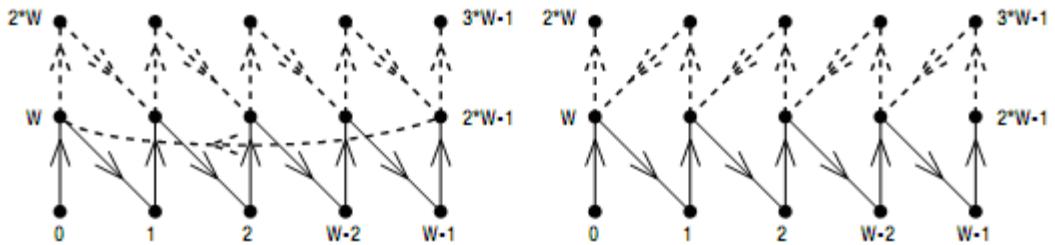
有了這些數據，就做好了創建這些新頂點並存儲到數組中的準備。

2. 定義索引：

定義了頂點後，就做好了通過定義索引數組構建三角形的準備。將以 TriangleStrip 定義三角形，基於一個索引及它前兩個索引表示一個三角形。

圖 5-16 顯示瞭如何使用 TriangleStrip 繪製三角形。數組中的第一個索引指向頂點 0 和 W。然後對行中的每個頂點，添加這個頂點和下一行對應的頂點，直到到達行的最後。這時，要定義 $2*width$ 個索引，對應($2*width-2$)個三角形，足夠覆蓋整個行。

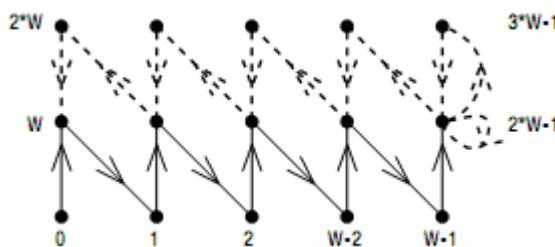
但是只定義了第一行，沒法使用這個方法繪製第二行，這是因為是基於前三個索引定義的三角形添加的每個索引的。基於這點，定義的最後一個索引指向頂點($2*W-1$)。如果再次從第二行開始，會從添加一個到頂點 W 的索引開始，如圖二十七所示。但是，這回定義一個基於頂點 W, ($2*W-1$)和(W-1)的三角形！這個三角形會橫跨第一行的整個長度，這不是想要的結果。



圖二十七：使用 TriangleStrip 定義三角形的錯誤方式

可以通過從右邊開始定義第二行解決這個問題。但是，簡單地從最後一個索引開始不是一個好主意，因為兩行的三角形的長邊有不同的方向，如教程 5-9 中的解釋，想讓三角形有相同的朝向。

圖二十八顯示瞭如何解決這個問題。在指向頂點(2^*W-1)的索引後，將立即添加一個指向相同頂點的索引！這會添加一個基於頂點(W-1)和兩個頂點(2^*W-1)的三角形，只會形成一條位於頂點(W-1)和(2^*W-1)之間的一條線，所以這個三角形不可見，叫做 ghost 三角形。接下來，添加一個指向頂點(3^*W-1)的索引，因為這個三角形基於兩個指向相同頂點(2^*W-1)的索引，所以實際上是一條線。如果從右邊開始定義第二行，正常情況下會從兩個頂點開始，記住實際上繪製了兩個看不見的三角形。



圖二十八：使用 TriangleStrip 定義三角形的正確方式

注意：可能認為無需添加第二個指向(2^*W-1)的索引，可以立即將一個索引添加到(3^*W-1)中。但是，基於兩個理由需要額外的指向頂點(2^*W-1)的索引。首先，如果沒有添加這個索引，那麼只有一個三角形被添加，會被 TriangleStrip 方式所需的繞行方向的反轉所干擾。第二，這會添加一個基於(3^*W-1), (2^*W-1)和(W-1)的三角形，如果三個頂點高度不相同那麼這個三角形會被顯示。

首先創建一個數組，存儲地形所需的所有索引。如教程二十八所示，每行需

要定義 $\text{width} \times 2$ 個三角形。在本例中，有三行頂點，但只繪製兩行三角形，所以需要 $\text{width} \times 2 \times (\text{height}-1)$ 索引。

3. 法線、頂點緩沖和索引緩衝：

要創建法線數據，通過創建一個頂點緩沖和一個索引 buffer 將這些數據發送到顯卡，然後繪製三角形。



圖二十九：實作之後的地形結果

第三章 系統架構與流程規劃

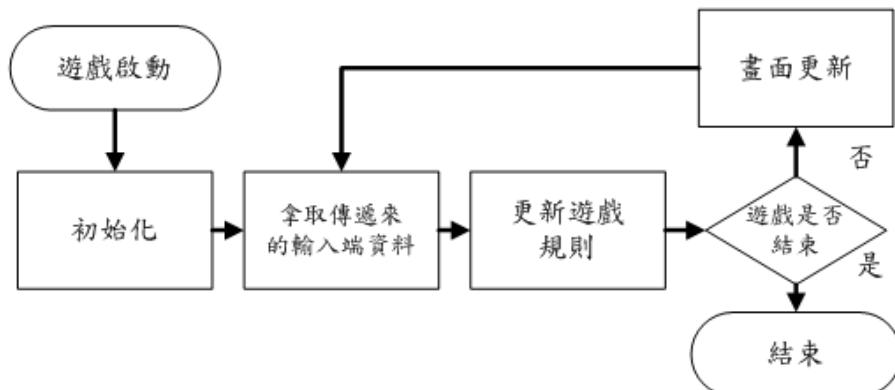
3.1 系統概述

此體感遊戲的主要分成遊戲端與輸入訊號端，其中輸入訊號的處理部分依附在遊戲中的一塊。驅動遊戲主程式，並透過遊戲來分配執行緒給輸入端做輸入影像的處理，最後以事件處理方式通知，給予遊戲對應的輸入訊號。

3.2 流程規劃

3.2.1 遊戲流程

此部分為遊戲程式的主要流程（圖三十），程式啟動時會先載入所有資源，並配置執行緒來啟動 Kinect 輸入端的程序，之後輸入端會開始進行一連流程辨識動作並以事件傳遞給遊戲保存，遊戲更新資料時便會取出，並做接續判斷和繪畫面等動作，而更細緻的流程會在下章節述說。

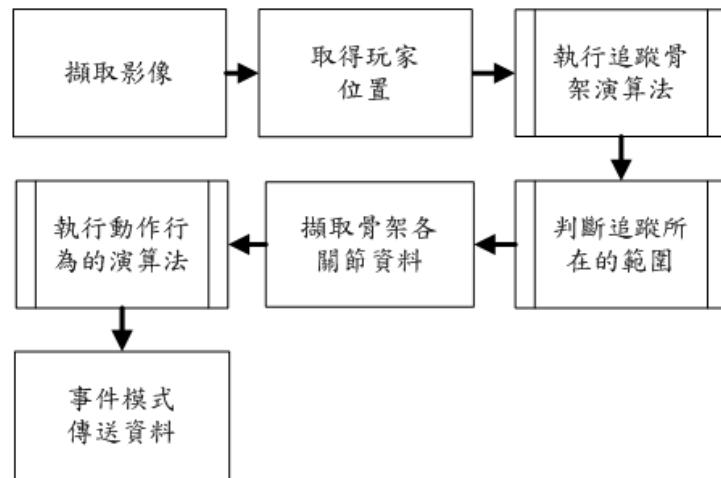


圖三十：遊戲流程圖

3.2.2 Kinect 輸入端處理流程

此為輸入端程式的主要流程，當影像輸入端（圖三十一）被啟動後，透過 Kinect 體感器取得影像，偵測出使用者後，取得使用者位置屬於合理範圍內，偵測骨架，並成功拿取骨架資料後，實作體感動作的演算法（詳細內容於第四章提），

判斷動作成功時，再以事件方式傳遞給遊戲，做為遊戲的輸入訊號，進行遊戲的操作，而更細緻的流程會在下章節述說。



圖三十一：影像端訊號處理流程圖

第四章 系統設計

4.1 遊戲設計

此小節為遊戲內容、遊戲規則與遊戲流程的設計部分，未牽扯到任何程式的設計方式，亦可視為開發需求。

4.1.1 遊戲介紹

本實務專題的遊戲模式，為直向捲軸的第三人稱冒險遊戲，採用關卡方式進行，關卡結束後會有積分排名。

各關卡的積分計算與結束方式皆不同，使用者擺脫一般遊戲的控制方式，如：鍵盤滑鼠的模式；而去使用肢體的動作去操控、來突破遊戲中的障礙，提升與遊戲的互動及趣味性。

4.1.2 遊戲規則

遊戲規則是當進行遊戲所需遵守，並束縛在這些規則下運行之意，此部分將會分別介紹遊戲的規則設計（表三）與最後完成（表四）時，要求的差異。

■ 設計：

編號	名稱	描述
Rule-01	單一進行	遊戲共 3 關，可選任一關進行遊戲
Rule-02	順序進行	遊戲共為 3 關，透過手動排序或亂序排序進行
Rule-03	積分排序	單一進行遊戲，結束後會依照各關卡的方式做排序；順序進行遊戲會把各關卡的積分累積在一起
Rule-04	時間限制	若為順序進行，則限定 3 關完成可用時間共 3 分鐘，使用者須在 3 分鐘內闖完 3 關，最後才會出現在積分排名表上，順序進行共 6 分鐘；單一進行依各關卡時間限制(4.1.4 節)
Rule-05	人物血量	遊戲整體以血量作為一般遊戲結束的依據，10 等份
Rule-06	積分計算	剩餘時間(秒) + 血條 + 各關得分數

表三：遊戲規則的設計

- 完成與設計需求落差：

編號	是否完成	落差描述	原因
Rule-01	半完成	只完成 2 關，並未皆完成	時間分配因素
Rule-02	無		時間分配因素
Rule-03	半完成	只完成單一進行積分	時間分配因素
Rule-04	無		時間分配因素
Rule-05	半完成	因只有時做出單一進行，所以各關卡不會皆需血量	時間分配因素
Rule-06	半完成	因只有時做出單一進行，所以各關卡已改成不同方式計算，且無時間限制包含	時間分配因素

表四：遊戲規則的完成與設計落差

4.1.3 遊戲選單設計

此部分會介紹主遊戲關卡外的選單部分，包含選單的層級設計操作方式、操作流程，將會分別介紹需求的設計（表五至表八）與最後完成（表九）時，要求的差異。

選單部分基本上分成兩個層級介紹與遊戲中的暫停選單，並列述了選單應有的操作方式，並如下：

- 第一層設計：

編號	名稱	描述
Menu-01	Play	進入 GameMenu：遊戲選擇部分
Menu-02	ReadHelp	進入文件說明，如何操作
Menu-03	Exit	離開遊戲

表五：選單第一層的需求設計

- 第二層設計：

編號	名稱	描述
Menu-04	certain one game	進入指定某一關的模式
Menu-05	random order game	進入隨機排序關卡模式
Menu-06	UserOrder	觀看各關卡模式遊戲的排名

表六：選單第二層的需求設計

- 暫停選單

編號	名稱	描述
Menu-07	暫停	除第一層與第二層外，各遊戲中皆可暫停，可離開遊戲或回到第一層選單

表七：暫停選單的需求設計

- 選單操作方式：

編號	名稱	描述
Menu-08	前推手勢	透過手前推的方式來選擇要進入的按鈕，請參考 4.4
Menu-09	左右滑動手勢	此用法當選擇關卡時使用，如同手機上的照片切換，左右滑動般。
Menu-10	移動手勢	以右手做為滑鼠般，移動目前座標
Menu-11	暫停手勢	左手臂向左方水平抬起約 3 秒，請參考 4.4

表八：選單操作方式的需求設計

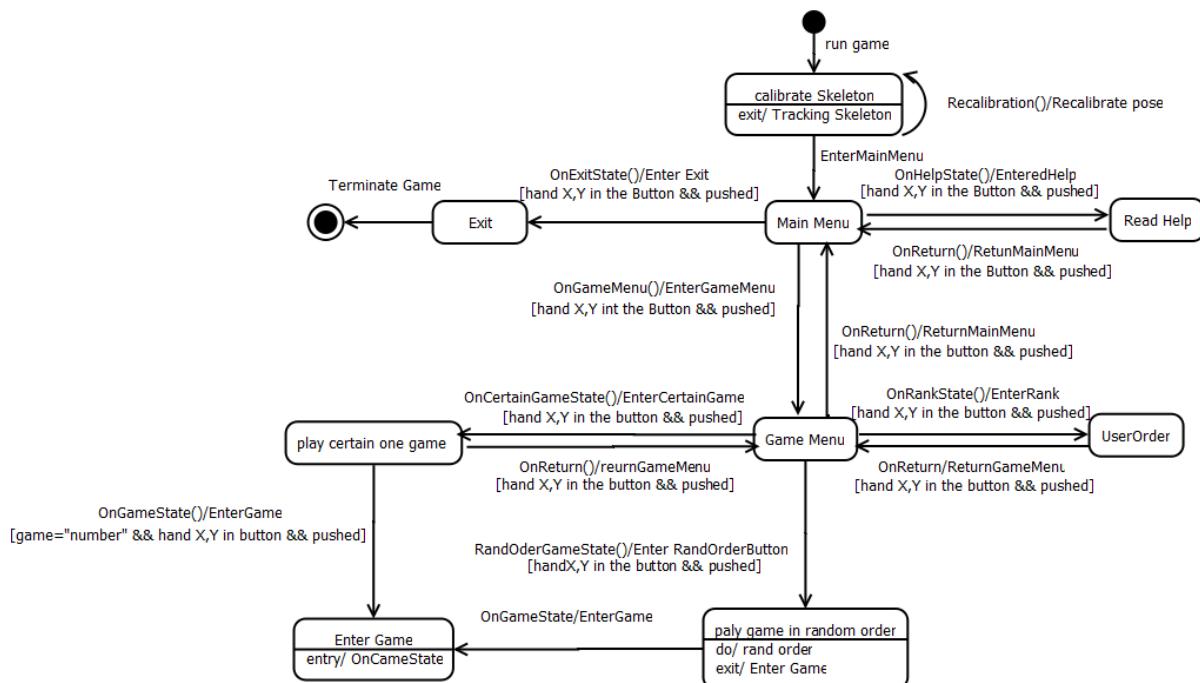
- 完成與設計需求落差：

編號	是否完成	落差描述	原因
Menu -01	完成		
Menu -02	半完成	有完成操作說明，但，是以目前選單層級而出現在畫面上，且為動畫模式	期望隨時進入遊戲時，皆會操作
Menu -03	完成		
Menu -04	完成		

Menu -05	無		時間分配因素
Menu -06	半完成	有完成積分排序功能，不過只包含單一關模式，且未放在第二層，而是第一層	第二層只剩下單一模式玩法，所以捨去多的按鈕，簡化介面
Menu -07	完成		
Menu -08	完成		
Menu -09	完成		
Menu -10	完成		
Menu -11	完成		

表九：選單全部需求設計與完成落差

而在選單進行流程方面，透過狀態圖（Sequential Diagram）表示按鈕選擇後的走向（圖三十二），其中，進入選單前會取得骨架資料與追蹤。MainMenu 為第一層選單，GameMenu 為第二層選單。



圖三十二：選單狀態圖

4.1.4 遊戲關卡設計

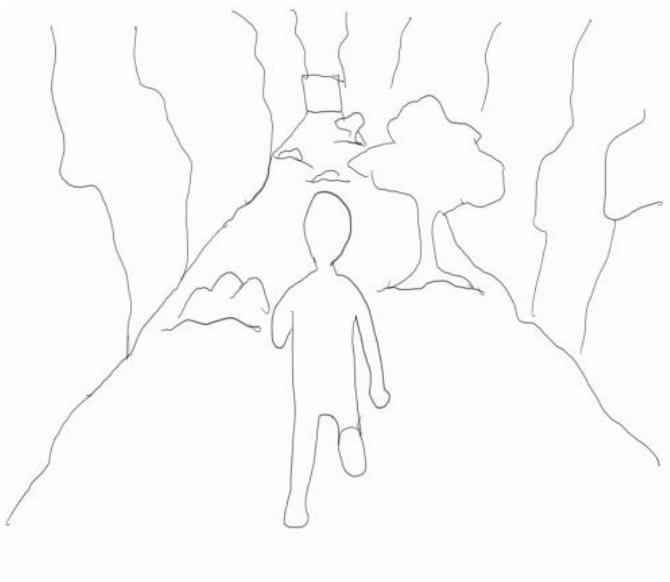
初期設計共 4 個關卡的遊戲，後改為 3 個關卡，各遊戲所使用的肢體動作均不同，並將會分別介紹需求的設計與最後完成時，要求的差異。

以下將會分別介紹關卡，以及各關卡包含的元素（障礙物、積分計算）與操作方式：

- 關卡一：逃離猛獸
- 介紹：場景設在叢林（圖三十三、圖三十四），在一開始會設置猛獸在使用者的後方，猛獸會不斷的前進，使用者也需不斷的前進來逃離猛獸，而一路上會不斷有叢林中的障礙出現，使用者須避開障礙前進到安全目的地，各類型障礙會有不同的影響效果。遊戲中除了障礙物以外，也有金幣和特殊道可以取得。



圖三十三：逃離猛獸的圖示



圖三十四：逃離猛獸的預想遊戲畫面

■ 關卡的元素需求設計：

編號	名稱	影響效果	解決手法
Run-01	石頭	血條減少 2 個等份，並暫時停止前進	使用者須閃避或跳躍
Run-02	河流	遲緩前進移動，前進速度 $\times 0.8$	使用者須跳躍
Run-03	泥沼	遲緩前進移動，前進速度 $\times 0.6$	只能加快跑步移動
Run-04	樹叢	遮蔽後方物體	使用者須揮砍
Run-05	樹木	血條減少 5 等份，並暫時停止前進	使用者須閃避
Run-06	裂縫	只會出現在接近終點時一次，掉落則遊戲結束	使用者須跳躍
Run-07	猛獸	若被猛獸追到，則遊戲結束	使用者須不斷跑
Run-08	鞋子	加快前進速度	
Run-09	星星	無視障礙物	

表十：逃離猛獸關的障礙物與道具元素

■ 關卡的積分需求設計：

編號	名稱	描述

Run-10	計算方式	剩餘時間（秒）+血量 + 金幣量
Run-11	金幣	佔 10 分，出現比例是 1/10
Run-12	銀幣	佔 5 分，出現比例是 3/10
Run-13	銅幣	佔 1 分，出現比例是 6/10
Run-14	血量	10 等份
Run-15	時間	2 分半

表十一：逃離猛獸關的積分計算與相關元素

- 操作方式：

編號	名稱	描述
Run-16	跑步	雙腿大動作跑，詳細會於 4.4 詳細介紹
Run-17	左右移動	身體左右移動，詳細會於 4.4 詳細介紹
Run-18	跳躍	詳細會於 4.4 詳細介紹
Run-19	揮砍	右手揮砍，分水平砍、垂直砍與斜砍，詳細會於 4.4 詳細介紹

表十二：逃離猛獸關的操作方式

- 完成與設計需求落差：

編號	是否完成	落差描述	原因
Run-01	完成		
Run-02	完成		
Run-03	無		時間分配因素
Run-04	無		時間分配因素
Run-05	完成		
Run-06	無		時間分配因素
Run-07	完成		
Run-08	無		時間分配因素
Run-09	無		時間分配因素

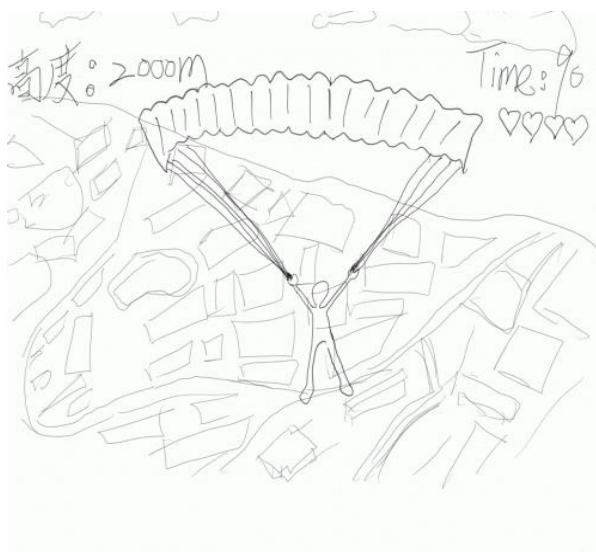
Run-10	半完成	無剩餘時間，只剩下血量 + 金幣量	遊戲規則的改變 與時間分配關戲
Run-11	完成		
Run-12	完成		
Run-13	無		時間分配因素
Run-14	完成		
Run-15	無		時間分配因素
Run-16	完成		
Run-17	完成		
Run-18	完成		
Run-19	完成		

表十三：逃離猛獸關的全部需求設計與完成落差

- 關卡二：飛傘危機
- 介紹：使用者須操控飛行傘來抵達降落地面（圖三十五、圖三十六），遊戲得畫面會不斷下降，終點地面為圓形，若未在圓形內下降會是前功盡棄，中途會有許多障礙來阻擋，並在最後下降時以風向阻擋抵達；此關操作飛行傘的方式會是額外定義。



圖三十五：飛傘危機的圖示



圖三十六：飛傘危機的預想遊戲畫面

- 關卡的元素需求設計：

編號	名稱	影響效果	解決手法
Fly-01	雲朵	遮蔽後方物體	閃避
Fly -02	風向	操控飛行傘的方向會成相反 (如：向右變向左，固定或隨 機皆可)	操控反方向
Fly -03	鳥類	血條減少 1 個等分	閃避

表十四：飛傘危機關的元素需求

- 關卡的積分需求設計：

編號	名稱	描述
Run-04	計算方式	剩餘時間 (秒) + 血量 + 金幣量
Run-05	金幣	佔 10 分，出現比例是 1/10
Run-06	銀幣	佔 5 分，出現比例是 3/10
Run-07	銅幣	佔 1 分，出現比例是 6/10
Run-08	血量	10 等份
Run-09	時間	2 分鐘

表十五：飛傘危機關的積分計算與相關元素

■ 操作方式：

編號	名稱	描述
Fly-10	預備姿勢	雙手如同拳擊防禦舉起，詳細會於 4.4 詳細介紹
Fly -11	左閃避	預備式下，左手下抬，詳細會於 4.4 詳細介紹
Fly -12	右閃避	預備式下，右手下抬，詳細會於 4.4 詳細介紹
Fly -13	下移	預備式下，雙手下抬，詳細會於 4.4 詳細介紹

表十六：飛傘危機關的操作需求

■ 完成與設計需求落差：

編號	是否完成	落差描述	原因
Fly -01	無		考慮到關卡實作難度與時間因素
Fly -02	無		考慮到關卡實作難度與時間因素
Fly -03	無		考慮到關卡實作難度與時間因素
Run-04	無		考慮到關卡實作難度與時間因素
Run-05	無		考慮到關卡實作難度與時間因素
Run-06	無		考慮到關卡實作難度與時間因素
Run-07	無		考慮到關卡實作難度與時間因素
Run-08	無		考慮到關卡實作難度與時間因素
Run-09	無		考慮到關卡實作難度與時間因素
Fly -10	完成		
Fly -11	完成		
Fly -12	完成		
Fly -13	完成		

表十七：飛傘危機關的全部需求設計與完成落差

此關卡並未實作出，因為評估遊戲實作的難度會花費大量時間。

- 關卡三：姿勢模仿
- 介紹：此關卡（圖三十七、圖三十八）是假設前方會不斷的有牆壁移過來，牆壁中都會有符合使用者骨架所能擺設的姿勢，而使用者需要擺出此姿勢來避免被壓到，另外，使用者會被迫前進。



圖三十七：姿勢模仿的圖示



圖三十八：姿勢模仿的預想遊戲畫面

- 關卡的元素需求設計：

姿勢牆：牆壁的形狀會不一，會先記錄參考版子，並透過區域性隨機的方產生，此關只會使用 7 個姿勢，並且產生過後即不會再產生，每一個姿勢牆會有 5 秒的緩衝時間前進。

編號	名稱	影響效果	解決手法
Pose-01	大面積撞到	若至少有 5 個節點碰觸到，血條減少 5 等份	穿越板子
Pose-02	邊緣撞到	只要有一個節點碰觸到，血條減少 2 等份	穿越板子

表十八：姿勢模仿關的元素需求

- 關卡的積分需求設計：

編號	名稱	描述
Run-03	計算方式	血量
Run-04	血量	共 10 等份

表十九：姿勢模仿關的積分設計

- 操作方式：

編號	名稱	描述
Pose-05	骨架動作	使現實世界人物的動作與遊戲中的動作一致

表二十：姿勢模仿關的操作方式

- 完成與設計需求落差：

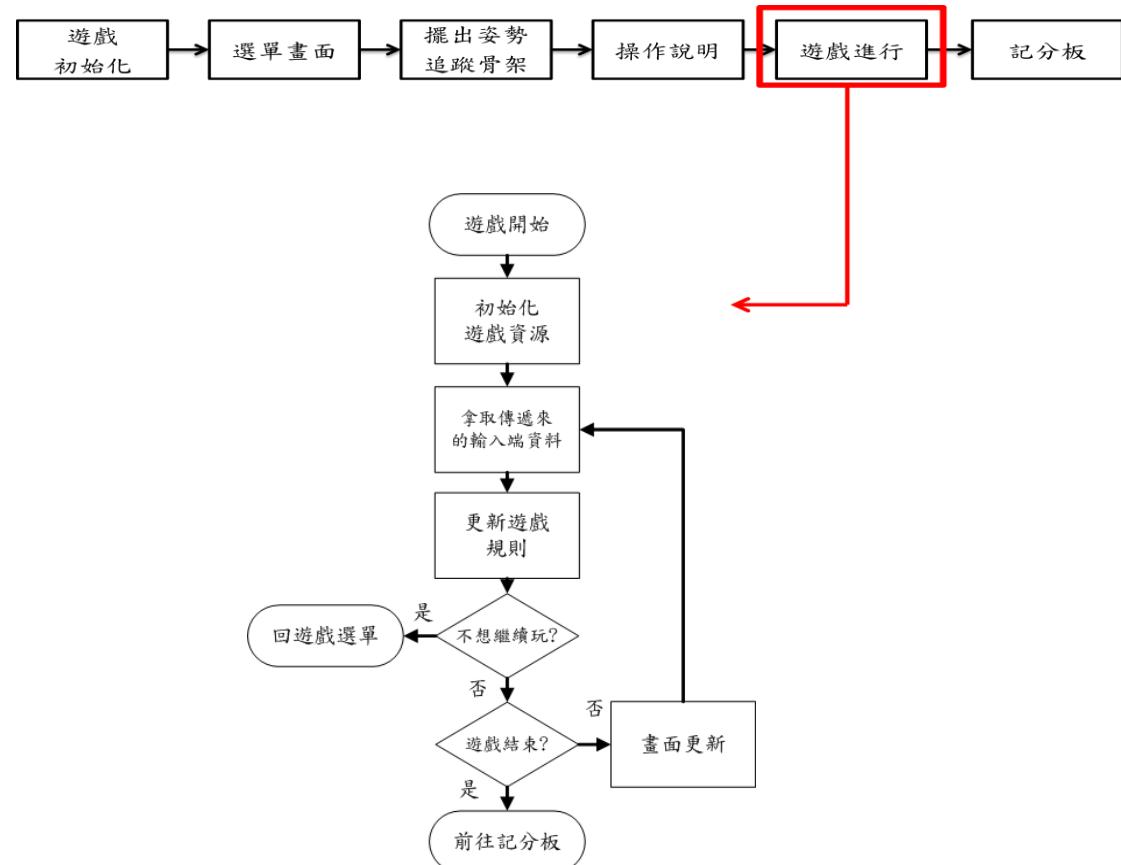
編號	是否完成	落差描述	原因
Pose -01	無		由於此關無血量，所以無必要性
Pose -02	半完成	分成通過與沒通過	由於此關無血量，所以無必要性
Pose-03	半完成	變為姿勢牆通過數	由於此關無血量，所以無必要性
Pose-04	無		遊戲內容的需求修改
Pose-05	完成		

表二十一：姿勢模仿關的全部需求設計與完成落差

4.1.5 遊戲流程設計

此部分為主要以使用者觀點的遊戲進行流程設計（圖三十九），流程步驟如下：

- (1.) 遊戲初始化：首先，開執行緒給 Kinect 輸入端處理運算，並接著註冊體感動作事件，完成使 Kinect 運作的步驟後再處理所有遊戲物件的管理器初始化。
- (2.) 選單畫面：進入選單畫面會有操作教學，透過動作去選擇關卡。
- (3.) 擺出姿勢追蹤骨架：玩家站定追蹤範圍，並擺出姿勢校正，取得骨架資源後，即可做動作，透過事件機制和遊戲做互動。
- (4.) 操作說明：選定遊戲後，就會另有進入該關卡的操作教學。
- (5.) 遊戲進行：開始遊戲的迴圈 (Loop)，更新遊戲規則，接受使用者的動作訊號，刷新畫面進行遊戲，並可透過特定動作觸發暫停選單。
- (6.) 記分板：遊戲結束後，會進入計分板觀看積分。

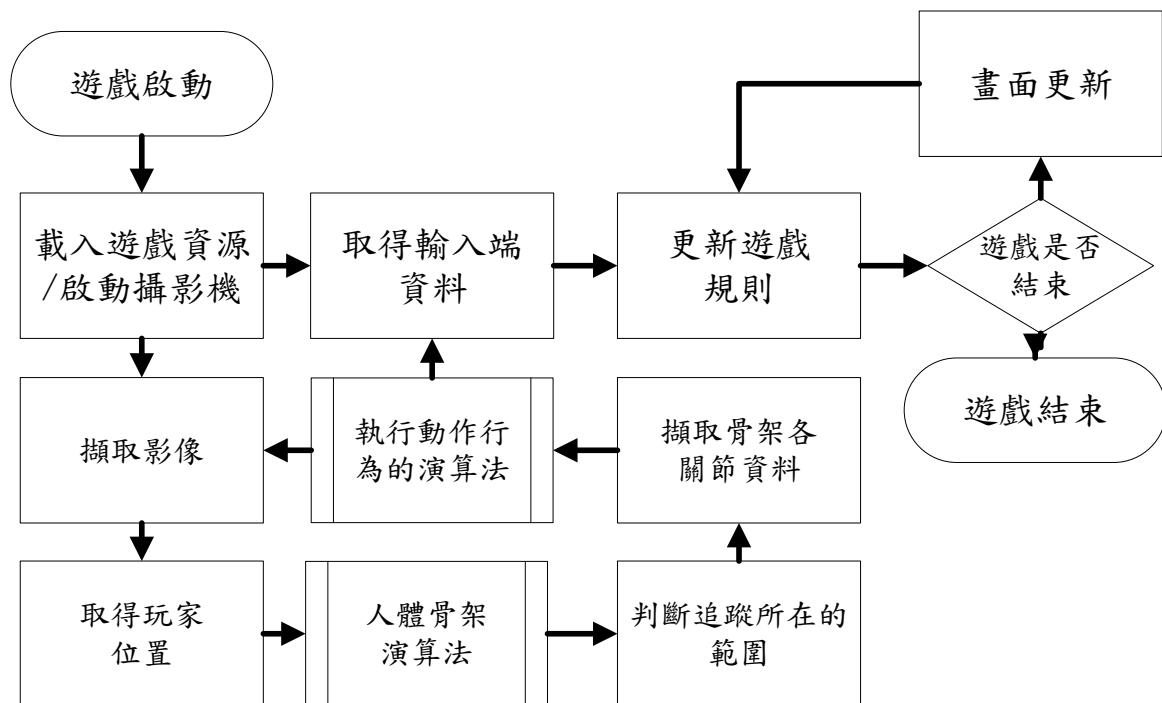


圖三十九：使用者觀點的遊戲流程圖

4.2 遊戲開發

此小節將會介紹以 XNA 作為開發平台的部分，從一開始的架構圖、圖學原理、遊戲流程與輸入端抓取 KINECT 資料，最後完成遊戲。

4.2.1 遊戲主要流程設計



圖四十：使者觀點的遊戲流程圖

圖四十為整個遊戲主程式啟動後之背景運作流程，下將會介紹主要流程的步驟：

1. 啓動遊戲主執行端，然後開啟分配給 Kinect 之執行緒，接著載入遊戲資源，且透過遊戲框架中之輸入處理取得 Kinect 所得知使用者動作。
2. 遊戲 UI 提示使用者擺出關鍵 pose 來讓系統抓取特定使用者的動作，系統找到之後會開始讓使用者操作遊戲 UI。
3. 使用者開始操作選擇 UI 中的選項，開始遊戲、計分板、操作說明，如使用者選擇計分板則進入計分畫面；操作說明則進入操作教學之畫面；離開遊戲則關閉

所有視窗回到作業系統；選擇遊戲開始則進入選擇關卡之畫面。

4.不論選擇任何關卡，皆會經過系統的取得輸入、更新規則、更新遊戲資訊至遊戲結束，接著回到計分板畫面，然後回到遊戲主選單。

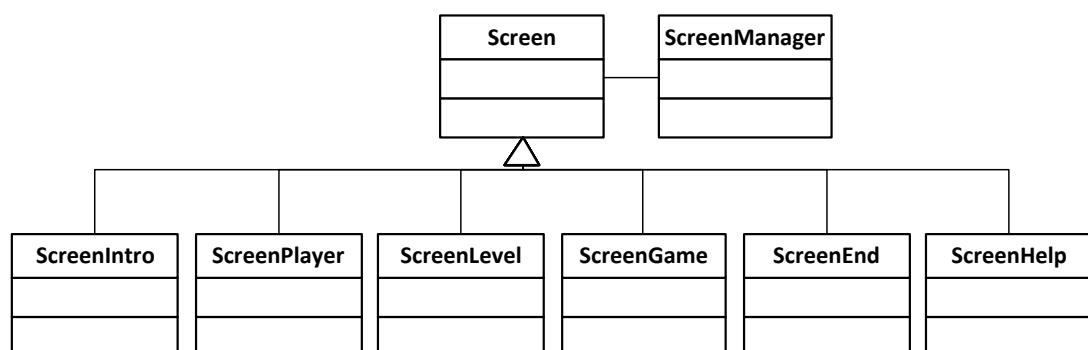
5.若遊戲執行途中發生骨架遺失、使用者超過偵測範圍等抓不到骨架之不可預期意外，則遊戲框架將會進入暫停之動作，所有更新將會停止，直到 kinect 處理部分抓到骨架，重新取得輸入後方可還原。

4.2.2 遊戲框架與設計

為了達到模組化與易偵錯之目的，本遊戲採用 OOP 中的 pattern 概念來撰寫設計整個程式的架構，以下將會一一介紹。

1. façade pattern：為撰寫一個遊戲需要相當多的系統，聲音系統、2D 畫面系統、3D 畫面系統，而每個系統中又有需多物件，如 3D 畫面系統中的人物、障礙物、樹木與金幣，如果沒有一個可以管理這些系統的元件，則整個遊戲框架設計起來將會非常複雜與困難，偵錯方面也不易進行。

因此本遊戲使用了 OOP 中 design pattern 的 façade pattern 來實現管理全部的物件。



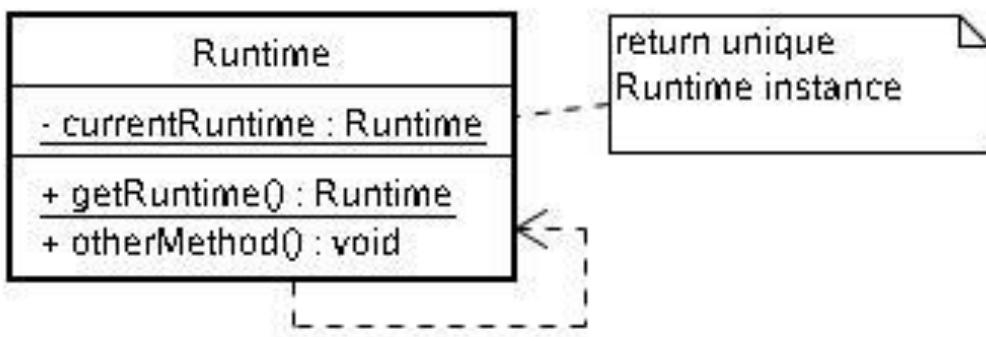
圖四十一：整個遊戲中的其中一塊架構

圖四十一中為 façade pattern 實做的其中一子系統，此系統乃實作遊戲中不同狀態之畫面，依序為選單、抓取使用者大頭貼、選擇關卡、遊戲遊玩中、遊戲結束

與使用教學。

從架構圖中可得知所有物件皆是繼承 Screen 而來，我們在撰寫一個 ScreenManager 來管理所有的 Screen 物件，這就是 façade 實做的其中一塊。

2. Singleton pattern：因為某些重要的物件只能被建構一次例如遊戲主執行緒、GameManager，如果不小心建構第二次可能會產生重大問題，因此我們必須使用 Singleton pattern 來保護我們這些重要的物件。



圖四十二：Singleton pattern 實作之 UML 圖

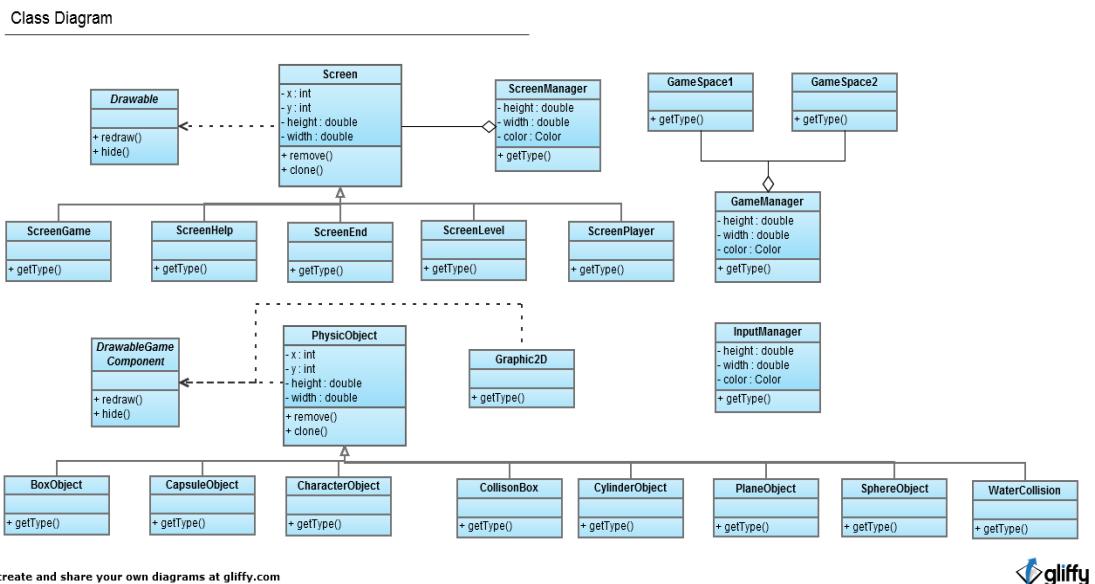
圖四十二中可以得知 Runtime 的建構式被宣告為 private，這樣可以阻止其他人使用建構方法來建立實例，pseudocode 如下：

```
public class Singleton {  
    private static Singleton instance = null;  
    private Singleton() {}  
    public static Singleton getInstance() {  
        if (instance == null) {  
            instance = new Singleton();  
        }  
        return instance;  
    }  
}
```

// .. 其它實作

}

全部架構圖：



圖四十三：遊戲架構圖（Kinect 不含訊號）

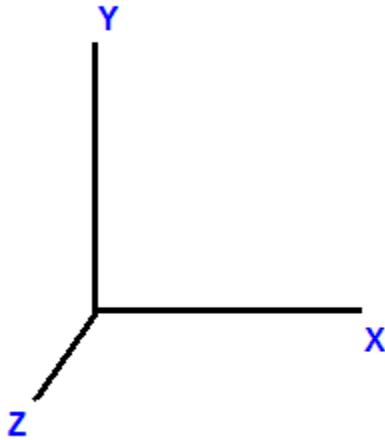
4.2.3 遊戲畫面製作

本專案遊戲畫面為 3D+2D 之設計概念，以 3D 繪圖為遊戲主體，2D 繪圖及圖層相關為介面 UI 顯示，以下將會加以討論。

1. 3D 畫面之製作：

在執行控制與顯示 3D 模型之前，首先必須了解以 XNA 為基礎的 3D 遊戲採用的座標系統，表示座標與控制座標轉換常用的資料型態，並認識 Object Space、World Space、View Space、和 Screen Space 四種空間代表的意義與用途。

以 XNA 為基礎的 3D 遊戲採用的是右手座標系統 (Right-Handed System)，其 X 軸往右遞增，Y 軸往上遞增，而 Z 軸則是往使用者的方向遞增，如圖四十四所示：



圖四十四：以 XNA 為基礎的 3D 遊戲採用的右手座標系統

控制或顯示 3D 模型，我們需要用到描述座標的 Vector2 和 Vector3 結構，以及用來協助座標計算的 Vector4 結構與 Matrix 結構。

除了描述座標和進行座標計算需要用到的結構型態以外，要顯示和控制 3D 模型，我們還需要了解 3D 模型的 Object Space (亦稱 Model Space)、World Space、View Space、和 Screen Space (亦稱 Projection Space) 四種空間表示法的意義。

簡單來說，3D 模型在 3D 的世界中有幾種不同的空間表示法，3D 模型中的座標點未經任何的轉置處理前稱為物體空間 (object space)，當 3D 模型經過旋轉，放大/縮小等轉置處理之後就會得到世界空間 (world space)，經過轉置處理過的世界空間在顯示給使用者檢視之前，還需要加入相機的位置和角度的處理，形成檢視空間 (view space)，最後的檢視空間在顯示之前必須投射到螢幕的 2 級空間，稱為螢幕空間 (screen space)，而螢幕空間顯示的內容就是使用者最終看到的結果。

要將 3D 模型的物體空間轉換成世界空間、檢視空間、和螢幕空間，我們需要借助於 Matrix 結構及其提供的屬性和方法。Matrix 結構負責定義一個 4×4

的矩陣，其常用的屬性可以參考表二十二 的說明：

表二十二：Matrix 結構常用的屬性

屬性名稱	說明
Backward	描述後退的向量。
Down	描述下移的向量。
Forward	描述前進的向量。
Identity	傳回單位矩陣。單位矩陣的定義就是任何矩陣和單位矩陣相乘之後的結果會得到原來的矩陣。
Left	描述左移的向量。
Right	描述右移的向量。
Translation	描述平移的向量。
Up	描述上移的向量。

說明：

單位矩陣的定義是矩陣的元素中，除了列編號和欄編號內容值相同的元素的內容值為 1 以外，其他的元素的內容值皆為 0。例如 2x2 的單位矩陣的內容如下：

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

3x3 的單位矩陣如下：

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

而 $n \times n$ 的單位矩陣如下：

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

Matrix 結構常用的方法請參考表二十三 的說明：

表二十三：Matrix 結構常用的方法

方法名稱	說明
Add	支援矩陣相加的方法。
CreateBillboard	依據指定的物體的位置建立球形的看板。
CreateConstrainedBillboard	依據指定的軸建立圓柱體看板。
CreateFromAxisAngle	依據指定的旋轉軸心和角度建立 Matrix 結構。
CreateFromQuaternion	依據 Quaternion 結構指定的向量和角度建立用來旋轉物體的 Matrix 結構。
CreateFromYawPitchRoll	依據 X 軸旋轉角度、Y 軸旋轉角度、和 Z 軸旋轉角度建立用來旋轉物體的 Matrix 結構。
CreateLookAt	建立以相機為原點的 View Space，並表示成 Matrix 結構。
CreateOrthographic	建立直角的 Projection Space，並表示成 Matrix 結構。
CreateOrthographicOffCenter	建立可客製化的直角 Projection Space，並表示成

	Matrix 結構。
CreatePerspective	建立透視的 Projection Space，並表示成 Matrix 結構。
CreatePerspectiveFieldOfView	依據視角 (Field of View-簡稱 FOV) 建立透視的 Projection Space，並表示成 Matrix 結構。
CreatePerspectiveOffCenter	建立可客製化的透視 Projection Space，並表示成 Matrix 結構。
CreateReflection	利用指定的平面的座標系統的反射建立 Matrix 結構。
CreateRotationX	建立能夠以 X 軸為軸心旋轉指定角度的 Matrix 結構。
CreateRotationY	建立能夠以 Y 軸為軸心旋轉指定角度的 Matrix 結構。
CreateRotationZ	建立能夠以 Z 軸為軸心旋轉指定角度的 Matrix 結構。
CreateScale	建立可以用來執行放大/縮小 3D 模型的 Matrix 結構。
CreateShadow	依據指定的光源建立可以用來表示陰影的 Matrix 結構。
CreateTranslation	建立可以用來平移 3D 模型的 Matrix 結構。
CreateWorld	建立可以用來表示 World Space 的 Matrix 結構。

Decompose	從 Matrix 結構取出放大/縮小、平移、和旋轉的成份值。
Determinant	計算 Matrix 結構的 determinant 值。以 2×2 的 $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ Matrix 結構為例，其 determinant 值為 $ad - bc$ 。
Divide	計算 Matrix 結構除以一個數值或是另外一個 Matrix 結構的結果，並表示成 Matrix 結構。
Invert	取得反矩陣，並表示成 Matrix 結構。
Lerp	對兩個 Matrix 結構對應的元素值執行線性內插。
Multiply	計算 Matrix 結構除以一個數值或是另外一個 Matrix 結構的結果，並表示成 Matrix 結構。
Negate	將 Matrix 結構每一個元素值改成負數，並將執行的結果表示成 Matrix 結構。
op_Addition	對兩個 Matrix 結構執行加法，並將執行的結果表示成 Matrix 結構。
op_Division	將 Matrix 結構除以一個數值，或是另外一個 Matrix 結構的元素，並將執行的結果表示成 Matrix 結構。
op_Equality	判斷兩個 Matrix 結構是否相等。
op_Inequality	測試 Matrix 結構和另外一個 Matrix 結構是否不相等。

op_Multiply	將 Matrix 結構乘以一個數值，或是另外一個 Matrix 的元素，並將執行的結果表示成 Matrix 結構。
op_Subtraction	對兩個 Matrix 結構執行減法，並將執行的結果表示成 Matrix 結構。
op_UnaryNegation	將 Matrix 結構的每一個元素都設定成負值。
Subtract	支援 Matrix 結構相減的方法。
Transform	依據 Quaternion 結構的內容對 Matrix 結構執行旋轉。
Transpose	對 Matrix 結構執行轉置 (Transpose) 運算，也就是列變成行，行變成列的轉換。

建立 3D 場景的時候，將 2D 的物體顯示成 3D 的效果以得到較好的效能的做法稱之為 Billboard (看板)，而 Matrix 結構提供的 CreateBillboard 方法與 CreateConstrainedBillboard 方法就是支援建立看板的方法。將 2D 的物體顯示成 3D 的效果主要的概念就是由物體圖案組成的紋理套用在矩形的元素上，由程式視需要旋轉矩形的元素，使其面向使用者的目光。請注意物體的圖案的形狀可以不是矩形的形狀，而且看板還可以允許某一部分形成透明的效果，以隱藏物件圖案的內容。

許多遊戲程式都是利用看板技巧顯示 3D 動畫，例如遊戲的主角在 3D 的迷宮內走動，所看到，而且可以取用的武器和寶物都是由 2D 的圖案套用在矩形元素形成的效果。3D 遊戲顯示樹林、灌木叢、或是天空的雲的時候，常常都是靠看板技術呈現的效果。當遊戲程式將圖案套用到看板的時候，矩形元素會先

旋轉至面對使用者的視線，然後再執行套用紋理的動作。

看板的效果適合套用在對稱的物體，特別是垂直對稱的物體，但是物體的視角不可以太高，否則當物體由上往下審視看板的時候，其效果會變成 2D 的效果，而不再具有 3D 的視覺效果。

■ 3D 模型轉置計算

Matrix 結構是遊戲程式顯示與控制 3D 模型極常用的型態，當遊戲程式需要平移、旋轉、或是放大/縮小 3D 模型時，就可以利用 Matrix 結構提供的功能建立可以將 3D 模型從 Object Space 轉換到 World Space 的 World Matrix。例如遊戲程式可以呼叫 Matrix 結構的 CreateTranslation 方法建立可以平移 3D 模型的 World Matrix，當您想將 3D 模型平移至 X 座標為 10，Y 座標為 0，Z 座標為 50 的位置，就可以利用以下的程式碼建立 World Matrix：

```
//Matrix WorldTranslation = Matrix.CreateTranslation(new Vector3(10, 0, 50));
```

做好之後，遊戲程式只要將所建立的 World Matrix 套用至 Mesh 的 Effect，就可以達到平移 3D 模型的效果。

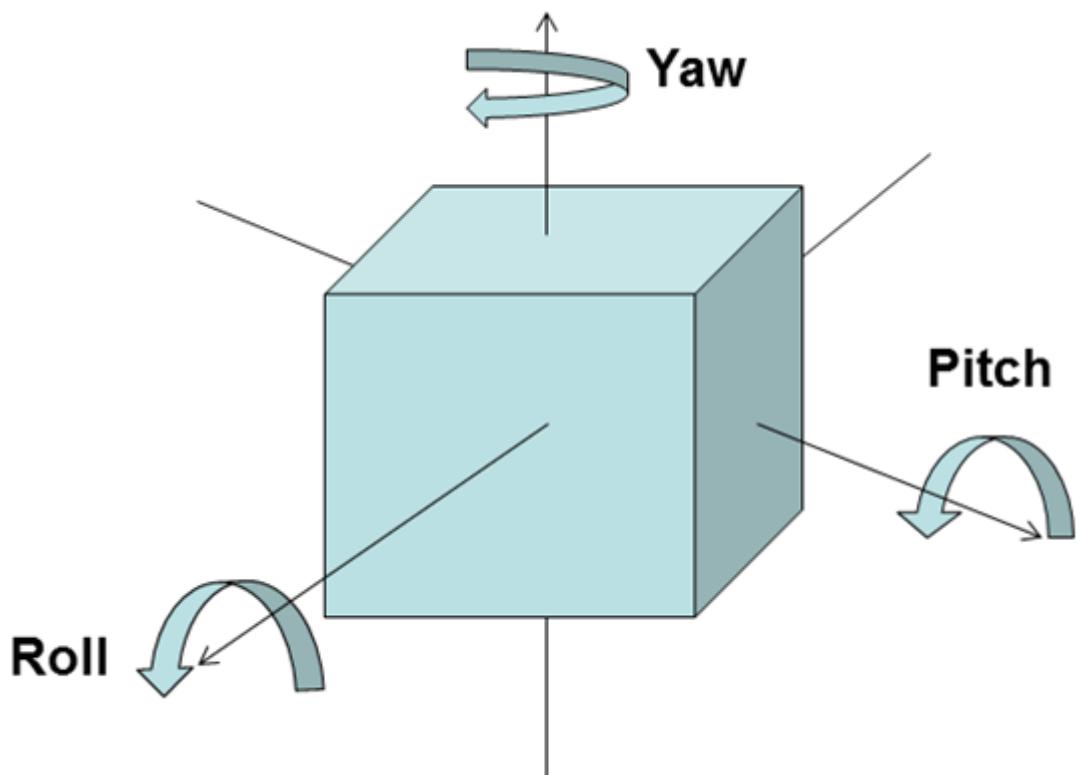
如果遊戲程式需要旋轉 3D 模型，可以呼叫 Matrix 結構的 CreateRotationX 方法並傳入旋轉角度表示要依據 X 軸為軸心旋轉指定的角度，呼叫 Matrix 結構的 CreateRotationY 方法並傳入旋轉角度表示要依據 Y 軸為軸心旋轉指定的角度，或是呼叫 Matrix 結構的 CreateRotationZ 方法並傳入旋轉角度表示要依據 Z 軸為軸心旋轉指定的角度。請注意 CreateRotationX、CreateRotationY、和 CreateRotationZ 方法需要的角度的單位是強度量，遊戲程式可以視需要先利用 MathHelper 類別的 ToRadians 方法將單位為度量的角度轉換成強度量，以便執行旋轉的動作。

以下的程式碼會建立可以將 3D 模型以 Y 軸為核心旋轉 180 度的 Matrix：

```
//Matrix worldRotation=Matrix.CreateRotationY(MathHelper.Pi);
```

遊戲程式除了可以利用 CreateRotationX、CreateRotationY、和 CreateRotationZ 方法來旋轉 3D 模型以外，也可以利用 Matrix 結構提供的 CreateFromYawPitchRoll 方法指定以 X 軸、Y 軸、和 Z 軸為軸心的旋轉角度，其中的 Yaw 代表以 Y 軸為軸心的旋轉角度，Pitch 代表以 X 軸為軸心的旋轉角度，而 Roll 代表以 Z 軸為軸心的旋轉角度，如圖四十所示：

圖四十五：Pitch 代表以 X 軸為軸心的旋轉角度，而 Roll 代表以 Z 軸為軸心的旋轉角度



圖四十五：Pitch 代表以 X 軸為軸心的旋轉角度，而 Roll 代表以 Z 軸為軸心的旋轉角度

透過上述放大/縮小、旋轉、和平移等操作，遊戲程式能夠將 3D 模型定位在指定的位置和角度。Matrix 結構進行乘法運算的時候必須按照 SRT (Scale *

Rotation * Translation) 順序來執行，也就是必須先執行放大/縮小的動作，再執行旋轉的動作，最後再執行平移的動作，因為 Matrix 結構的乘法不會有累積的效果。

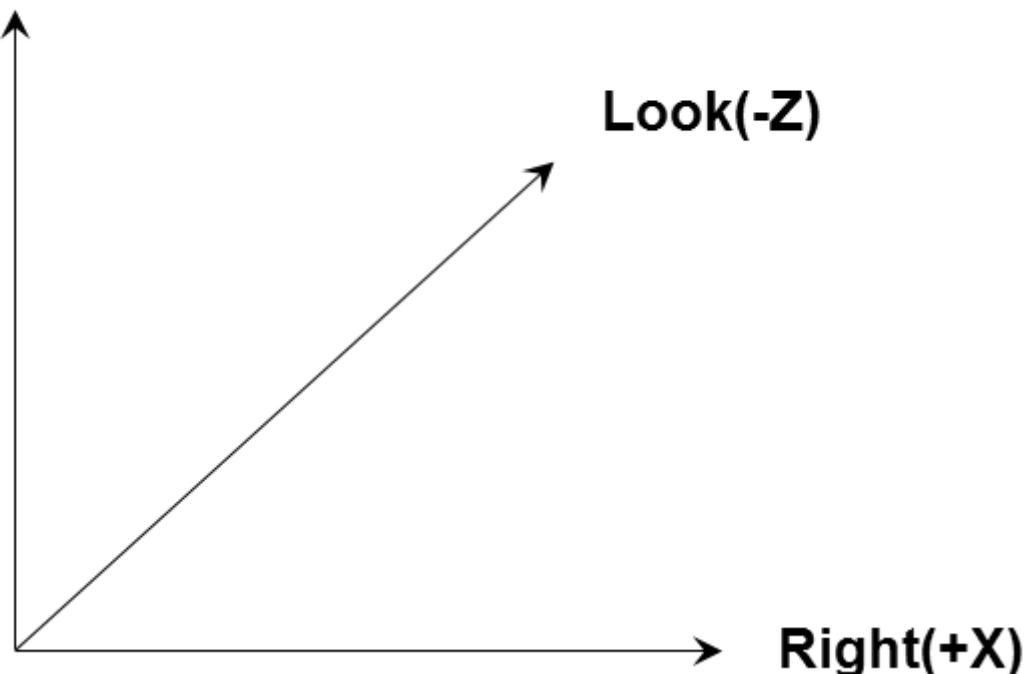
接下來就要引入相機的位置，計算出 3D 模型的檢視空間 (View Space)，也就是從使用者的角度檢視 3D 模型的位置。

遊戲程式可以呼叫 Matrix 結構提供的 CreateLookAt 方法，依據包括相機位置在內的相關資訊，建立可以將世界空間轉換成檢視空間的 Matrix 結構。例如以下的程式碼就會利用 Matrix 結構的 CreateLookAt 方法建立可以將 3D 模型世界空間轉換成檢視空間的 Matrix 結構：

```
//Matrix viewMatrix = Matrix.CreateLookAt(new Vector3(0, 3, 5), new Vector3(0, 0, 0), new Vector3(0, 1, 0));
```

CreateLookAt 方法的第一個參數代表相機的位置，第二個參數代表欲檢視的物體的位置，而第三個參數則是代表 Up 向量，也就是相機旋轉依據的軸心，傳入 (0,1,0) 表示相機可以以 Y 軸為軸心旋轉 (即圖四十六 中的 UP 軸)，傳入 (0,0,-1) 表示相機可以以 Z 軸為軸心旋轉 (即圖四十六 中的 LOOK 軸)，而傳入 (1,0,0) 表示相機可以以 X 軸為軸心旋轉 (即圖四十六 中的 RIGHT 軸)，如圖四十六 所示：

Up(+Y)



圖四十六：相機座標軸

計算好 3D 模型的檢視空間之後，遊戲程式還需要將檢視空間轉換成螢幕空間，也就是將 3D 模型的檢視空間投影到 2D 的平面空間的位置，完成呈現 3D 模型的動作，因此螢幕空間 (Screen Space) 又稱為投影空間 (Projection Space)。要完成從檢視空間至投影空間的轉換，遊戲程式可以利用 Matrix 結構提供的 CreatePerspectiveFieldOfView 方法，建立能夠執行轉換工作的 Matrix 結構 (即 Projection Matrix)。

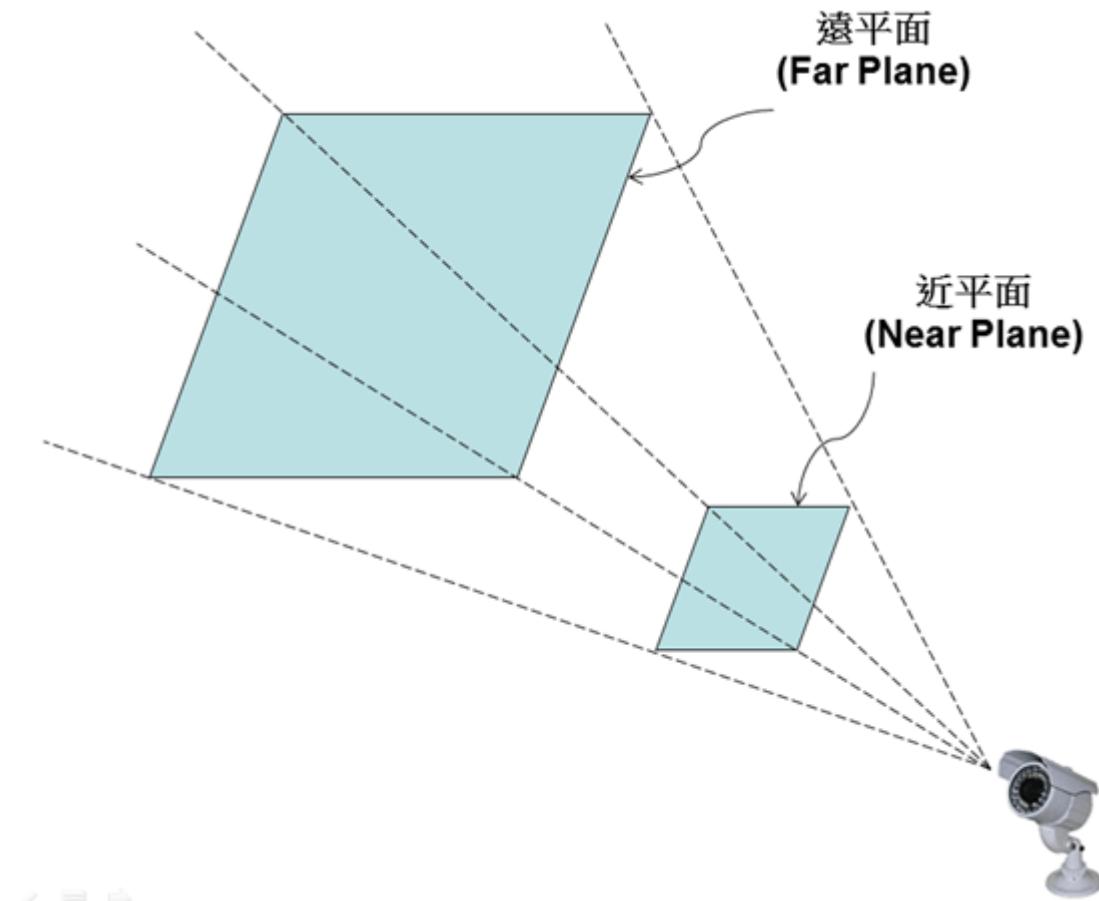
例如以下的程式碼就會呼叫 Matrix 結構的 CreatePerspectiveFieldOfView 方法，建立可以將 3D 模型的檢視空間投射到螢幕空稱的 Matrix 結構：

```
//float aspect = (float)Window.ClientBounds.Width / (float)Window.ClientBounds.Height;  
  
//Matrix projectionMatrix = Matrix.CreatePerspectiveFieldOfView(
```

```
MathHelper.PiOver4, aspect, 1, 100);
```

Matrix 結構的 CreatePerspectiveFieldOfView 方法需要的第一個參數是單位為弧度量的視角，其角度必須介於 $0\text{~to~}180$ 度之間，第二個參數是高寬比例，您可以將遊戲視窗的寬度除以遊戲視窗的高度當做高寬比例，或是直接使用 GraphicsDeviceManager 的 GraphicsDevice 屬性的 Viewport 屬性的 AspectRatio 屬性的內容值當做高寬比例，第三個參數是與近平面 (Near Plane) 的距離，請注意距離比與近平面還近的內容都不會呈現在遊戲視窗中，第四個參數是與遠平面 (Far Plane) 的距離，請注意距離比與遠平面還遠的內容都不會呈現在遊戲視窗中。

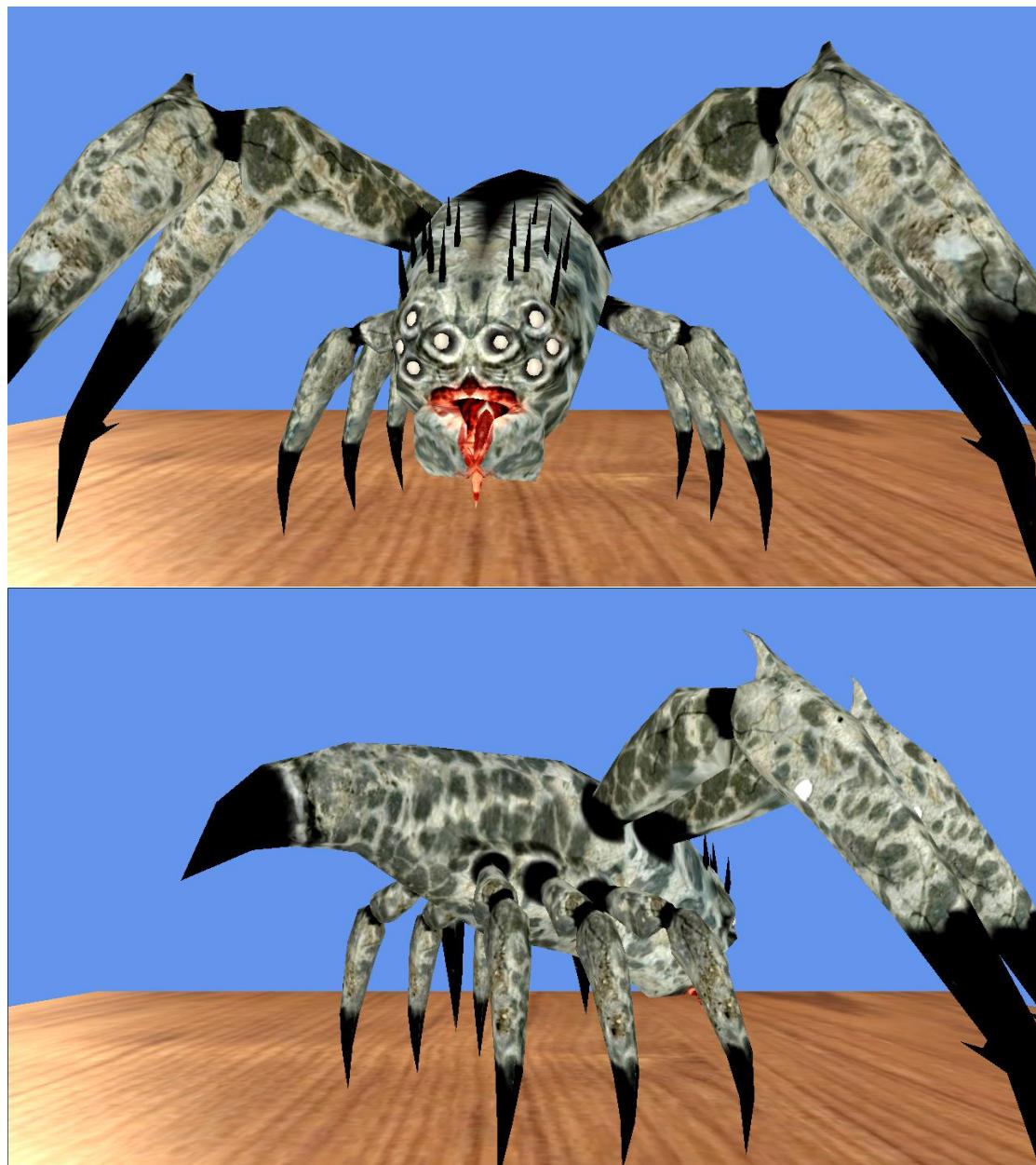
有關近平面與遠平面在 3 度空間的意義可以參考圖四十七：



圖四十七：近平面與遠平面在 3 度空間的意義

- 實作結果：

允許使用者控制 3D 模型旋轉角度和大小的遊戲程式的執行畫面。



2. 2D 製作 UI

XNA Framework 支援的 SpriteBatch 類別可以協助遊戲程式輸出 2 維的圖片和文字，並提供批次輸出的功能，避免頻繁地輸出遊戲的內容造成閃爍的現象。

雖然以 XNA 為基礎的 3D 遊戲輸出 3 維的圖形內容時不需要依靠 SpriteBatch 類別提供的功能，但是當 3D 遊戲需要提供文字的功能選單供使用

者選擇的時候，還是需要使用到 `SpriteBatch` 類別提供的文字輸出功能。表二十四所示為 `SpriteBatch` 類別常用的屬性

表二十四：`SpriteBatch` 類別常用的屬性

屬性名稱	說明
<code>GraphicsDevice</code>	取得遊戲程式所使用的 <code>GraphicsDevice</code> 類別的物件的屬性。

表二十五所示為 `SpriteBatch` 類別常用的方法：

方法名稱	說明
<code>Begin</code>	宣告批次繪圖的動作開始。
<code>Draw</code>	繪製 2 維圖案。
<code>DrawString</code>	繪製文字。
<code>End</code>	宣告批次繪圖的動作結束，將從呼叫 <code>Begin</code> 方法之後繪製的所有內容輸出到遊戲視窗，並將 <code>GraphicsDevice</code> 的狀態還原到呼叫 <code>Begin</code> 方法之前的狀態。

表二十五：`SpriteBatch` 類別常用的方法

`SpriteBatch` 類別的功能看似簡單，但是負責繪製 2 維圖案的 `Draw` 方法有高達 7 個不同的多載 (Overload) 版本，負責繪製文字的 `DrawString` 方法也有多達 6 個不同的多載版本，讓遊戲程式可以經由傳入不同數量的參數，控制方法執行的結果，如果再加計列舉型態的參數的內容值選擇，就可以組合出各種繁複的變化，滿足遊戲程式就變更遊戲程式內容的需求。

SpriteBatch 類別與圖形特效支援

SpriteBatch 類別支援繪製 2 維圖案的 Draw 方法一共有以下 7 個不同的多載版本：

SpriteBatch.Draw(texture, destinationRectangle, color)

SpriteBatch.Draw(texture, destinationRectangle, sourceRectangle, color)

SpriteBatch.Draw(texture, destinationRectangle, sourceRectangle, color, rotation, origin, effects, layerDepth)

SpriteBatch.Draw(texture, position, color)

SpriteBatch.Draw(texture, position, sourceRectangle, color)

SpriteBatch.Draw(texture, position, sourceRectangle, color, rotation, origin, scale, effects, layerDepth)

SpriteBatch.Draw(texture, position, sourceRectangle, color, rotation, origin, scale, effects, layerDepth)

參數請參考表二十六的詳細說明：

表二十六：SpriteBatch 類別的 Draw 方法需要使用的參數

參數名稱	說明
texture	型態為 Texture2D 類別的參數，負責管理欲繪製的 2 維圖案。
destinationRectangle	負責描述欲繪製在遊戲視窗上的物體的矩形。
color	負責控制欲繪製的圖形的色調的參數，傳入 Color.White 表示不改變欲顯示的物體的色調。
sourceRectangle	指定欲用來取用來源物件的矩形。
scale	圖案放大/縮小的倍數。
rotation	旋轉角度。

origin	旋轉圖形時所依據的圓心。
effects	設定成 <code>SpriteEffects.FlipHorizontally</code> 表示要水平翻轉圖案，設定成 <code>SpriteEffects.FlipVertically</code> 表示要垂直翻轉圖案。
layerDepth	圖層深度。0 代表前景層，1 代表背景層，可以搭配呼叫 <code>SpriteBatch</code> 類別的 <code>Begin</code> 方法傳入的 <code>SpriteSortMode</code> 參數控制是否要對欲繪製的內容依 <code>layerDepth</code> 的內容值排序。
position	指定欲繪製的圖案的左上角點座標。

光是靠 `SpriteBatch` 類別的 `Draw` 方法的眾多參數，就能夠創造繁複的遊戲效果，例如透過 `color` 參數設定圖案的色調和透明度，透過 `rotation` 參數指定旋轉圖案的角度，利用 `origin` 參數設定圖案旋轉的圓心座標，利用 `effects` 參數水平翻轉或垂直翻轉圖案，利用 `scale` 參數指定放大/縮小圖案的倍數，或是利用 `layerDepth` 參數設定圖案要顯示在那一個圖層。

我們只要在主程式之 `Draw` 方法中呼叫 `SpriteBatch` 的 `Draw` 方法，傳入適當的參數，就可以顯示出各種特殊的效果，例如以下的 `Draw` 方法便會旋轉、放大/縮小、水平/垂直翻轉、改變圖案色調、以及設定圖案的透明度。

4.3 輸入端 Kinect 訊號處理

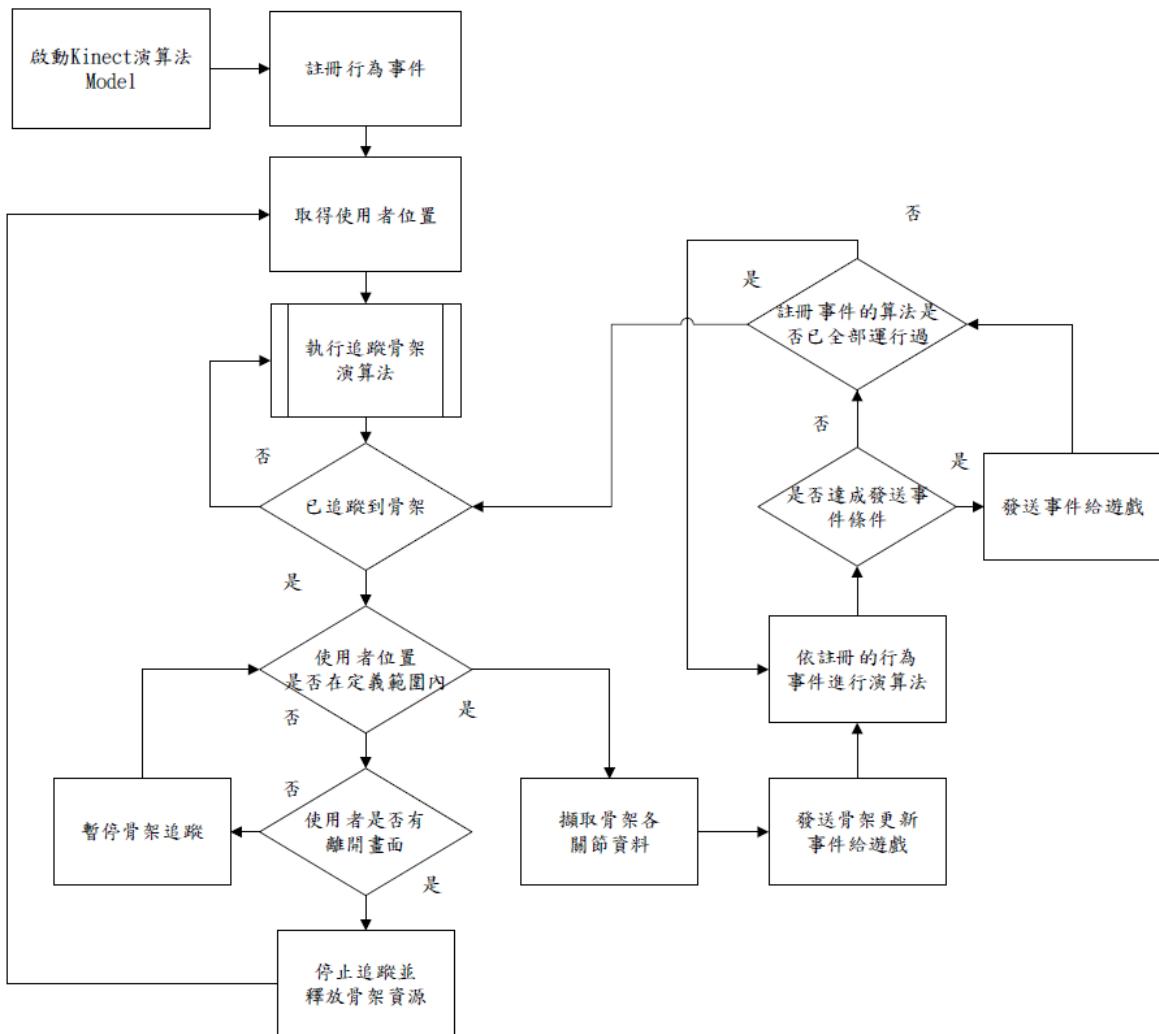
此小節將會介紹 Kinect 部分，從前處理的辨識骨架節點與追蹤，到動作的處理與發送。

4.3.1 Kinect 主要訊號處理流程設計

圖四十七是 Kinect 程式的流程處理，以下將會介紹主要流程的步驟，另外前述所提的「動作」在此部份是對應「動作行為」：

- (1.) 啟動 Kinect 演算法 Model：由遊戲端主程式，透過配與執行緒（Thread）啟動 Kiecnt 程式。
- (2.) 註冊事件：遊戲端主程式對 Kinect 程式註冊動作的事件，此事件的目的為，遊戲需要的訊號或資訊，當註冊後，Kinect 會運行註冊過的需求去演算動作行為訊號是否觸發。
- (3.) 取得使用者位置：Kinect 程式啟動後，先尋找場景中的使用者（人），判斷出人形後，取得使用者的軸心 3 維座標作標，以便之後的活動距離判斷。
- (4.) 執行骨架追蹤演算法：執行 OpenNI 內建的骨架追蹤演算法（參考 2.3 節），另外，因應遊戲需求，因此另把可追蹤的骨架人數鎖定成一人，先擺出姿勢校正到的使用者，則可取得追蹤骨架的資源。
- (5.) 已追蹤到骨架？：判斷使否以追蹤到骨架，否則回第(4)項重新判斷。
- (6.) 使用者位置是否在定義的範圍內：若以追蹤到使用者，則接著去透過之前的軸心座標，去尋找使用者是否在合理的活動範圍，此目的為了避免判斷的不穩定，並有合理的活動空間而實施；若使用者不在範圍內，則進到第(6)點去判斷是否仍在 Kinect 畫面；若有在範圍內，則跳至第(7)點
- (7.) 使用者是否有離開畫面：若使用者仍在 Kinect 畫面內，但離開了規定的範圍，則進入第(8)項，要求使用者回到合理範圍內後，並回到上一步的判斷；但若使用者也不在範圍內，進入第(9)項，並等待新的使用者做偵測。

- (8.) 暫停骨架追蹤：保留目前使用者的骨架資訊並暫停骨架追蹤。
- (9.) 停止追蹤並釋放骨架資源：強制停止目前使用者的追蹤，並把目前的資料全部砍掉釋放。
- (10.) 擷取骨架各關節資料：正在追蹤的骨架各關節資料，把此資料保存下來，留給後續的去演算動作。
- (11.) 發送骨架更新事件給遊戲：透過事件把每次取得的骨架 3 維座標資訊給予遊戲端，遊戲端可利用此座標去做遊戲人物與實際人物的骨架座標的對應。
- (12.) 依註冊的行為事件進行演算法：即為把註冊過的動作行為事件，各執行演算法運算是否有此動作。
- (13.) 是否達成觸發條件：若動作行為演算法運算成功，及代表使用者有此動作發生，即可進入第(14)項；若未運算成功，則代表無此動作發生，或姿勢未做到位，並繼續別的動作行為事件演算法。
- (14.) 發送事件給遊戲：運算出的動作行為，及代表有把動作做到，則把遊戲需要的訊號透過事件發給遊戲。
- (15.) 註冊事件的演算法是否已全部執行過：若未全部執行完所有的動作行為演算法，則回到第(12)項繼續；若以全部執行完，則離開，回到第(5)項繼續。



圖四十七：Kinect 程式流程圖

- KinectModel.cs 程式碼：Kinect 執行緒工作得主函式

```
public void ModelDoWork()
```

至於要如何用執行緒起動，請直接參考第六章節，會有詳細的步驟。

- KinectDevice.cs 程式碼：執行追蹤骨架的演算法

```
#region Event handler 區塊段落，此段落為事件註冊的處理函式
```

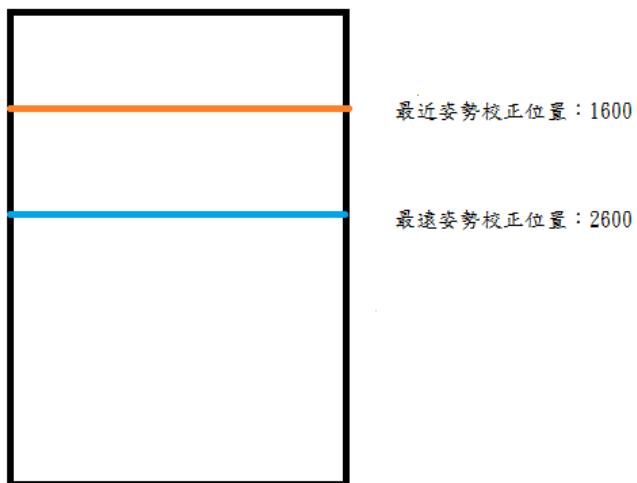
4.3.2 使用者姿勢校正範圍設計

預設的使用者姿勢偵測與校正，是沒有範圍上的限制，只要能夠辨識出使用者上半身的 Psi 動作即可，但為了另遊戲在進行時，進行姿勢校正可順利，所以因此設定此範圍。

以下介紹姿勢校正的活動範圍名稱與定義，如圖四十八，包含：

- 最近姿勢校正位置(userMinPoseDetectionPosition)：表示使用者若要取得骨架，而進行偵測姿勢時要佔定的最近位置，要至少超過 1600 公厘(1.6 公尺)，擺出 Psi 才會有反應。
- 最遠姿勢校正位置(userMaxPoseDetectionPosition)：表示使用者若要取得骨架，而進行偵測姿勢時要佔定的最遠位置，要至少小於 2600 公厘(2.6 公尺)，擺出 Psi 才會有反應。

因此使用者須介在 1.6 至 2.6 公尺才可做姿勢的校正。



圖四十八：使用者偵測姿勢校正的範圍

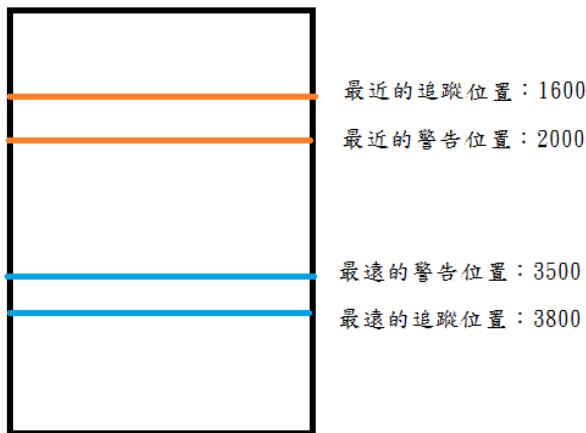
- KinectModel.cs 程式碼：校正範圍變數

```
//使用者校正的範圍  
private int userMaxPoseDetectionPosition = 2600; //進入校正的最大範圍距離  
private int userMinPoseDetectionPosition = 1600; //進入校正的最小範圍距離
```

4.3.3 使用者骨架追蹤範圍設計

此部分為圖四十七的使用者位置是否在追蹤範圍內的設計，此目的是為了限制使用者的活動空間，確保使用者的最佳動作活動空間所加的限制，以下為活動範圍的名稱與描述，如圖四十九，包含：

- 最近的追蹤位置(usrMinTrackingPosition)：表示使用者若在骨架追蹤模式下，小於 1600 公厘(1.6 公尺)的數值就會暫停追蹤骨架，其目的是因為此時骨架無法取得全身資訊，可能會影響遊戲進行而加的限制。
- 最近的警告位置(usrMinWarningPosition)：表示使用者若在骨架追蹤模式下，在小於 2000 公厘(2.0 公尺)的數值時仍會繼續追蹤，但使用者狀態會顯示警告。
- 最遠的追蹤位置(usrMaxTrackingPosition)：表示使用者若在追蹤模式下，大於 3800 公厘(3.8 公尺) 的數值就會被暫停追蹤骨架，其目的是因為此時骨架會處於不穩狀態，可能會影響遊戲進行而加的限制。
- 最遠的警告位置(usrMaxWarningPosition)：表示使用者若在追蹤模式下，在大於 3500 公厘(3.5 公尺)時仍會繼續追蹤，但使用者狀態會顯示警告。



圖四十九：追蹤的邊界圖

- KinectModel.cs 程式碼：追蹤的可移動範圍變數

```
//為使用者骨架追蹤的可移動範圍區塊,為預設值  
  
private int userMaxTrackingPosition = 3800; //最遠可追蹤的距離  
  
private int userMaxWaringPosition = 3500; //最遠進入警告的距離
```

```
private int userMinWaringPosition = 2000; //最近進入警告的距離  
private int userMinTrackingPosition = 1600; //最近可追蹤的距離,也是進入校正的最小範圍距離
```

4.3.4 使用者狀態設計

使用者狀態的設計 UserStateEnum，用來告知使用者目前的狀態以及 Kinect 程式內部的判斷，如下表二十七：

使用者狀態名稱	描述
LookingForPose	尋找擺出校正姿勢的人
Calibrating	姿勢校正中
Tracking	追蹤使用者(表校正成功)
TrackingButWaring	仍骨架追蹤，但站的位置太近或太遠
ReTracking	重新骨架追蹤
StopTracking	停止骨架追蹤
NewUser	新的使用者進入畫面
LostUser	遺失使用者(離開 Kinect 畫面過久)
UserReEnter	離開畫面(但 Kinect 仍在尋找)的使用者重新進入畫面
UserExit	使用者離開畫面(剛從 Kinect 畫面離開，在 LostUser 之前)
JointUserExit	有骨架的使用者離開畫面
JointUserMissed	有骨架的使用者不見(例外狀況發生，可能有過多人擋住 Kinect 畫面或畫面不完全，導致 Kinect 運算錯誤)
JointUserLost	有骨架的使用者遺失(離開 Kinect 畫面過久)

表二十七：使用者狀態

- UserStateEnum.cs 程式碼：設計列舉部分

同上

- KinectModel.cs 程式碼：取得使用者目前 UserState 的方法

```
public Dictionary<int, UserStateEnum> UserCurrentState
```

Int 是使用者的編號，UserStateEnum 是目前這位使用者的狀態。

- KinectModel.cs 程式碼：記錄所有使用者的 UserState 資料結構

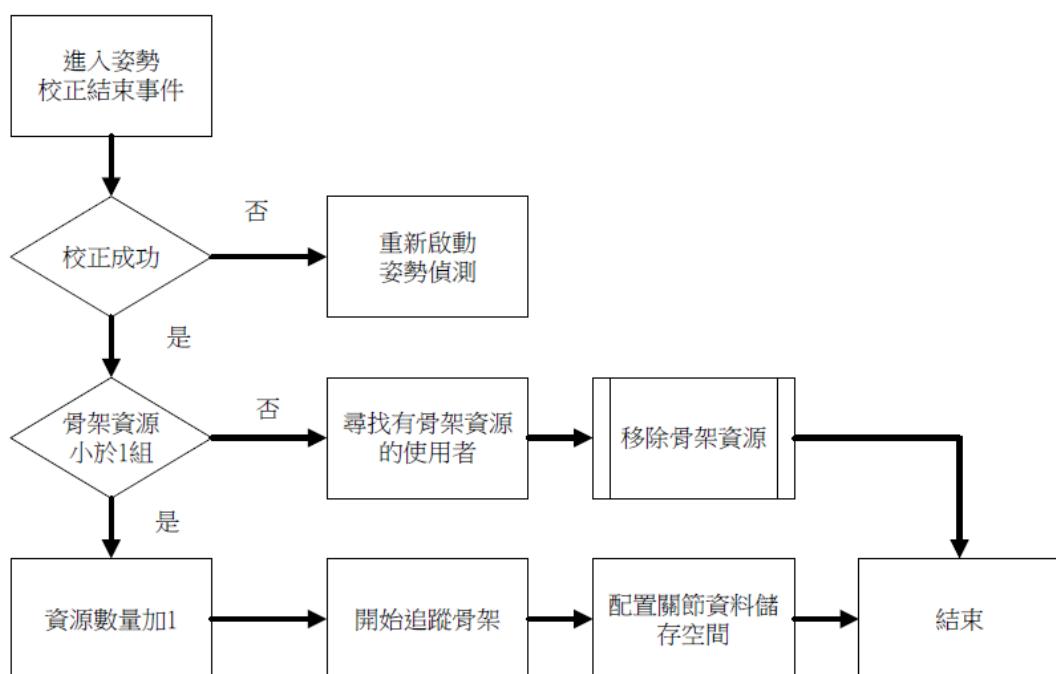
```
//記錄每一個使用者目前狀態的資料結構
```

```
Dictionary<int, UserStateEnum> usercurrentState;
```

4.3.5 骨架資源設計

搭配 OpenNI 系列所使用的骨架資源，預設是可以追蹤到 2 組以上，但由於遊戲的需求設計，以及為了使環境單純化的原由，所以在骨架資源的追蹤上，限制為 1 組資源，即代表最早進到畫面，擺出偵測姿勢，並校正完成的使用者才可取得骨架資源。

下圖五十為姿勢校正結束的事件，也就是圖十八的 CalibrationEnd 區塊，在此事件觸發後，會需要做校正成功的判斷以及開始追蹤骨架、配置骨架儲存關節空間的部分，在這裡透過 Counter 的方式去做資源的限制，當資源小於 1，則把資源數加 1，並開始追蹤；當資源已滿時，則找出目前有骨架的使用者，並移除此使用者的關節資料與追蹤。

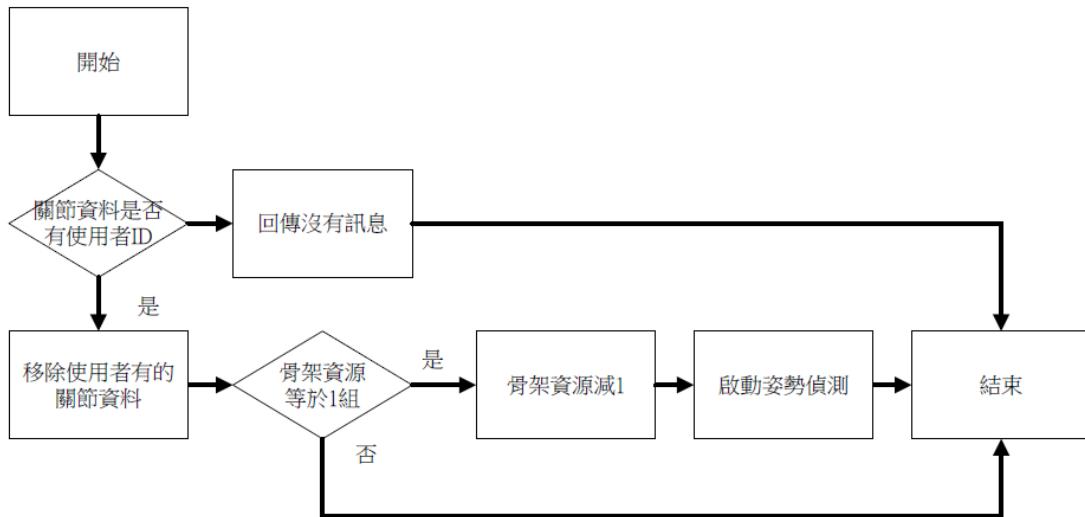


圖五十：骨架資源的設定

而其後，為了可以令使用者隨時切換，並直接進入遊戲的關係，所以多加了搶奪骨架的模式。也就是當有使用者已取得骨架資源時，其他使用者進入畫面，並擺出偵測姿勢並校正完後，即會把已存在的骨架資源釋放掉，並轉為自己使用，

不過前提是站在偵測姿勢的校正範圍內才可。

下圖五十一，則是移除骨架資源的程序，尋找出使用者後，再找出其 ID 編號，並移除關節資源，釋放骨架資源的 Counter 數。



圖五十一：骨架資源移除程序

- KinectDevice.cs 程式碼：骨架資源設定的事件處理函式

```
private void skeletonCapbility_CalibrationEnd(object sender, CalibrationEventArgs e)
```

- KinectDevice.cs 程式碼：移除骨架資源函式

```
public bool RemoveJointAndResource(int userId)  
//移除骨架和釋放骨架的佔用資源，呼叫此功能的情況是：使用者離開畫面或使用者已經完全  
遺失或系統找到的骨架有問題
```

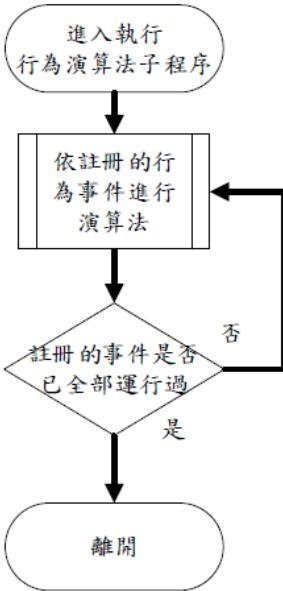
4.4 動作演算法設計

此小節是介紹遊戲所需要的動作設計，透過 Kinect 拿取骨架資訊，並以骨架節點為資料來源設計出一系列的動作。

遊戲進行的動作演算法總共包含了 7 項，包含：選單所使用的推(Push)、滑動(Swipe)、暫停(Pause)與關卡一的跑(Run)、跳躍(Jump)、左右移動(Move)與揮砍(Slash)，以上的動作除暫停(Pause)與飛行(Fly)外，都需事先累積 frame 的張數(累積的張數是透過人工實驗而借定出的數值，如下表)，取出 frame 中需要的關節點，並計算前後不同張 frame 之間關節點需要的 MetaData(速度或角度等)做為觸發動做的判斷；而暫停(Pause)因為不需要計算速度的關係，所以不需使用 buffer 累積。下面將會介紹這些動作演算法的流程設計。

動作	累積的張數
Swipe	3
Run	2
Jump	6
Move	6
Slash	6
Push	3

而為了啟動動作演算法的判斷，須要先進行註冊動做的判斷，在判斷到有哪些動作註冊後，才會進行演算法的運算，並於達到動作的觸發滿足條件後，以事件的方式通知，下圖五十二為註冊演算法動作的流程圖：



圖五十二：註冊演算法動作的流程圖

- KinectModel.cs 程式碼：執行演算法子程序，依註冊事件執行

```
//依據要計算的演算法來收集關節的資料

private void JointUpdate(Dictionary<SkeletonJoint, SkeletonJointPosition> userJointPos)
```

判斷是否有註冊事件，依照有註冊的事件，進行收集更新骨架節點的工作，由 ModelWork 函式呼叫。

此章節為演算法的設計，至於要如何使用這些動作與事件(包含事件的描述)，請至 6.3 節了解。

4.4.1 選單一推(Push)演算法流程設計

Push 演算法是用來在選單畫面時，確認選項的動作，也就是按下按鈕的意思，圖五十三即為推的示意圖（以左手為例）。

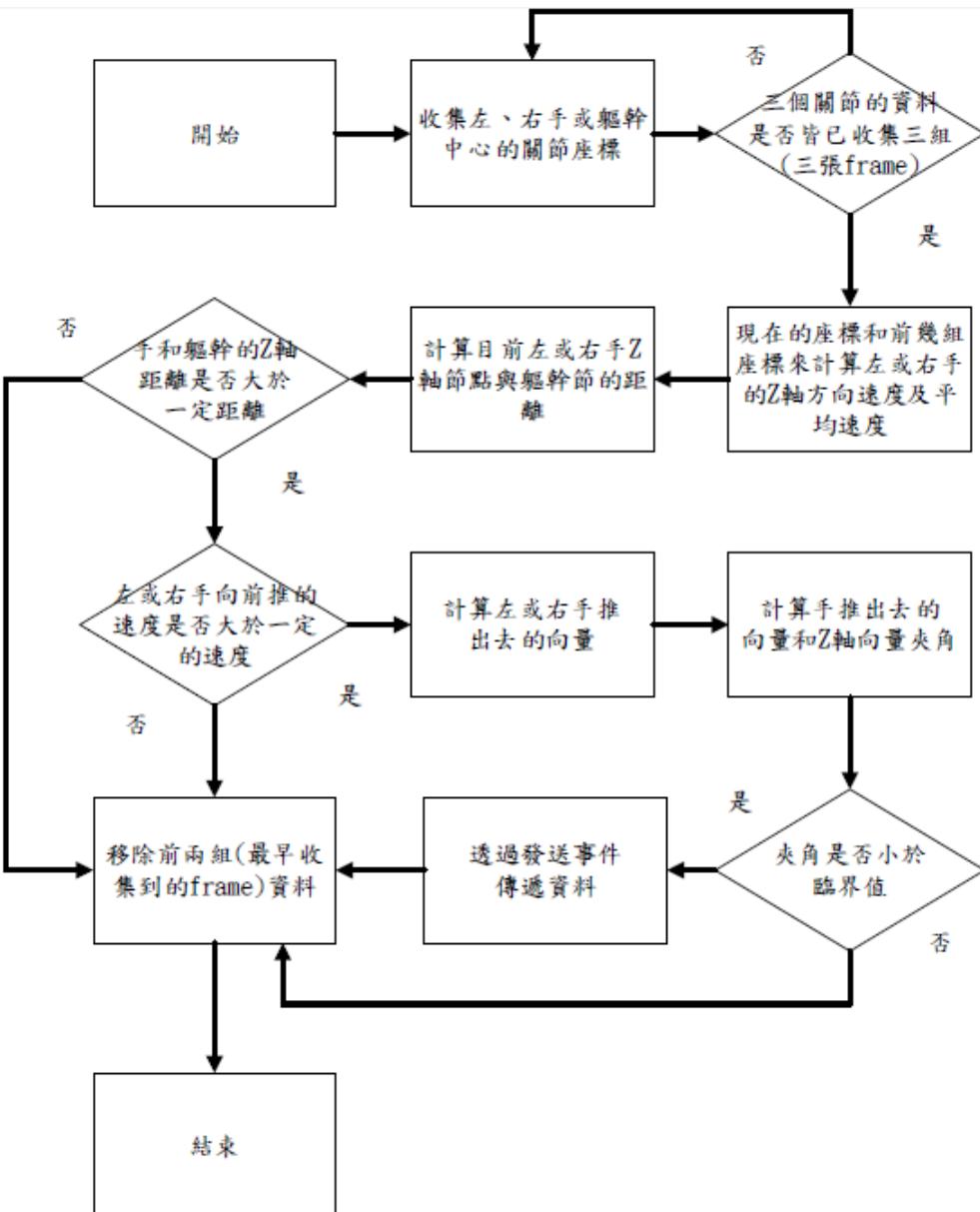


圖五十三：Push 的示意圖

- 演算法概述：

先判斷身體與目前取得的 frame 中左(或右)手的 z 軸距離差，再運算 z 軸 Vz 速度，若條件滿足，再以目前累積的 buffer 中，最早的 frame 的左手節點為軸心，往 z 軸延伸出，若速度與角度接有滿足實驗出的經驗值，即判斷有向前推得動作。

- 流程圖：



圖五十四：推(Push)演算法的流程圖

- KinectModel.cs 程式碼：資料結構

```
private Dictionary<SkeletonJoint, LinkedList<SkeletonJointPosition>> jointNodeBufferForPushAndDragAndSwipe;
```

使用字典搭配 Linklist 來累積 frame 的張數，透過 SkeletonJoint 來找出需要的關節點作為 Key 對應到游 LinkList 所組成的儲存資訊空間，後面介紹的動作設計，資料結構皆是使用相同的概念。

- KinectModel.cs 程式碼：累積的張數臨界值變數

```
private int pushAndSwipeTimeSpan = 3;
```

由於 Push 和 Swipe 要用的關節點和累積的張數都一致，因此使用相同的變數。

- KinectModel.cs 程式碼：收集更新資料的函式

```
private void UpdatePushAndSwipeData(SkeletonJoint j, SkeletonJointPosition pos);
```

更新資料的函式，Push 和 Swipe 也共用

- 收集的關節點名稱

SkeletonJoint
LeftHand
RightHand
Torso

- KinectModel.cs 程式碼：演算法的函式

```
private void CalculatePushVelocityTrigger(SkeletonJoint j, Dictionary<SkeletonJoint,  
LinkedList<SkeletonJointPosition>> listPos);
```

- KinectModel.cs 程式碼：相關臨界值變數

```
private int pushSpeedThreshold = -60; //向前推的瞬間速度臨界值  
  
private int pushAngleThreshold = 40; //向前推的角度容忍值  
  
private int pushDistanceWithTorsoThreshold = 300; //向前推時手伸出的與骨架中心的距離臨  
界值
```

4.4.2 選單—滑動(Swipe) 演算法流程設計

Swipe 演算法是用來作為切換關卡時所設計的動作，而 Swipe 的資料結構和收集資料的演算法一致(參考 4.3.6)，所以此不敘述。

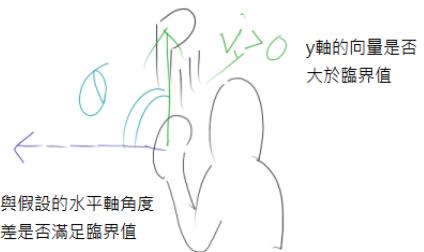
- 演算法概述：

取得一定的左手關節點樣本 frame，計算 Vx 和 Vy 的速度，當 Vx 和 Vy 的速度滿足後，再依照向量判斷方向，最後並確認揮動的角度是否滿足臨界值，作為 Swipe 的觸發，下圖左圖五十五為水平方向，向左的 Swipe，右圖五十六則為垂直 Swipe (皆以左手為例)。



水平向量小於一定的臨界值，垂直向量未超過一定的臨界值。

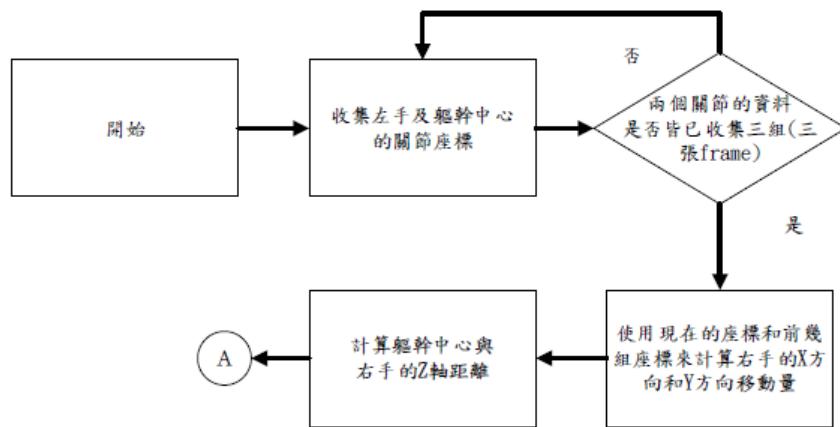
圖五十五：水平 swipe 示意圖



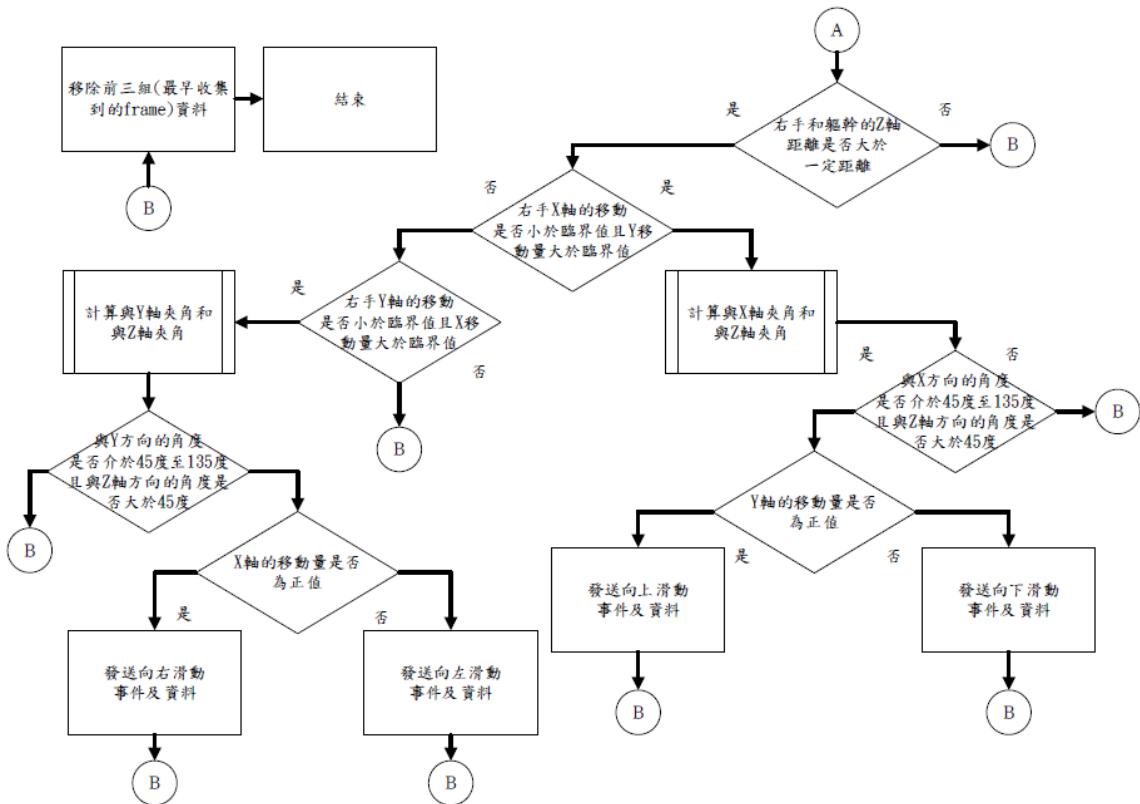
圖五十六：垂直 swipe 示意圖

- 流程圖：

此流程共分為上下兩張，如下圖五十七、五十八：



圖五十七：Swipe 演算法流程圖一上



圖五十八：Swipe 演算法流程圖一下

■ KinectModel.cs 程式碼：演算法的函式

```

private void CalculateSwipeTrigger(Dictionary<SkeletonJoint, LinkedList<SkeletonJointPosition>>
listPos);

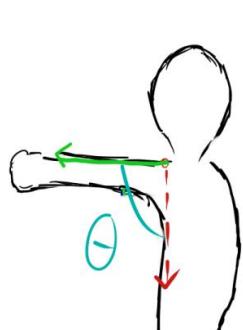
```

- KinectModel.cs 程式碼：相關臨界值的變數

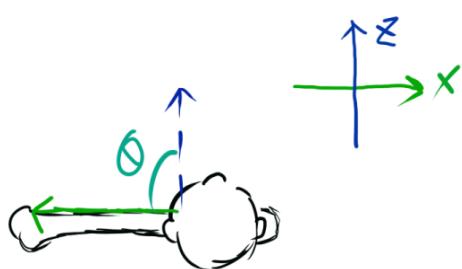
```
private const int SWIPE_UP_DOWN_ANGLE_THRESHOLD = 45;  
  
private const int SWIPE_RIGHT_LEFT_ANGLE_THRESHOLD = 45;  
  
private const int SWIPE_Z_ORI_THRESHOLD = 45;  
  
private const int SWIPE_VIRTUAL_POINT_DISTANCE = 300;  
  
private const int SWIPE_UP_DOWN_DISPLACEMENT = 200;  
  
private const int SWIPE_RIGHT_LEFT_DISPLACEMENT = 220;  
  
private const int SWIPE_TORSO_DISTANCE_THRESHOLD = 100;
```

4.4.3 選單—暫停(Pause) 演算法流程設計

Pause 演算法適用在關卡一、關卡二，幫助遊戲進入暫停選單。



圖五十九：Pause 手臂水平抬起姿勢圖

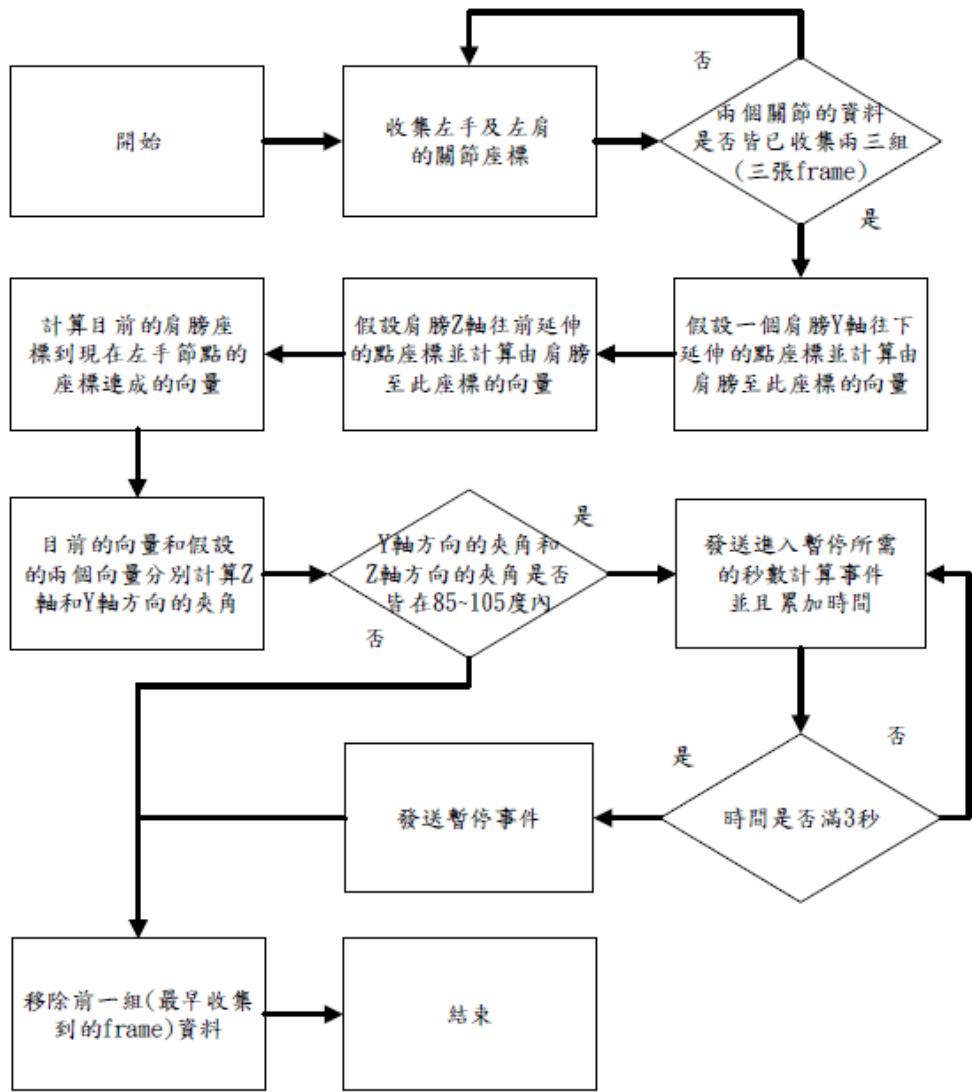


圖六十：俯瞰圖視野

- 演算法概述：

針對目前的 frame，先拿取左肩的關節點，並往 z 軸前方和 y 軸下方各延伸出一條向量，也就是 leftShoulderToHorizonPointVector(簡稱 lhv)和 leftShoulderTo - VerticalPointVector(簡稱 lvv)，再算出把左肩到左手關節點的 xy 軸向量 Vxy，並與 lhv 做點積，算出水平的角度(圖五十九)、lvv 算出 z 軸垂直的角度(圖六十)，確保兩個夾角都介在一個範圍之間（同時左手與左肘要保持一直線），並維持直到 Counter 到達 90，即算完成。

流程圖：



圖六十一：Pause 演算法流程圖

- KinectModel.cs 程式碼：資料結構

```
private Dictionary<SkeletonJoint, SkeletonJointPosition> jointNodeBufferForPauseAngel;
```

- KinectModel.cs 程式碼：判斷是否為 Pause 的變數

```
private int pauseTimeSpan = 90;
```

pauseTimeSpan 變數的目的要由 Pause 這個動作的定義說起，Pause 的判斷方式是定義成左手臂水平抬起(圖五十九)，並且要維持將近一段時間才會進入暫停，因

此這個變數是用來判斷手的抬起姿勢是否有累積到 3 秒(影像 1 秒 30 張 frame 來算，所以 3 秒 90 frame)。

- 收集的關節點

SkeletonJoint
LeftHand
LeftShoulder
LeftElbow

- KinectModel.cs 程式碼：收集更新資料的函式

```
private void UpdatePauseData(SkeletonJoint j, SkeletonJointPosition pos);
```

- KinectModel.cs 程式碼：演算法的函式

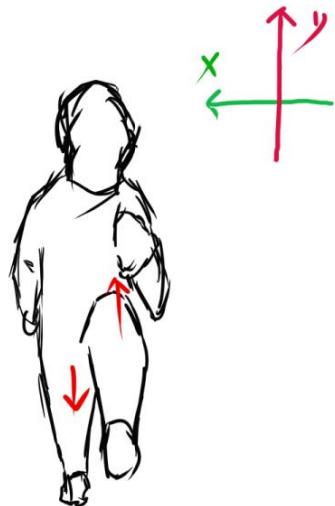
```
private void CalculatePauseAngelTrigger(Dictionary<SkeletonJoint, SkeletonJointPosition> jpos);
```

- KinectModel.cs 程式碼：相關臨界值變數

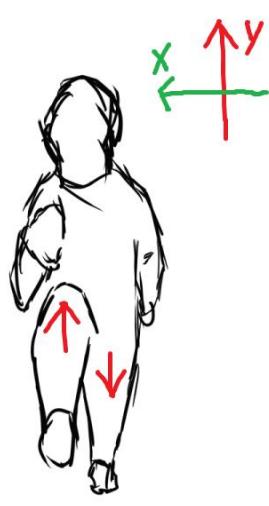
```
private const int PAUSE_VERTICAL_ANGLE_UPPER_THRESHOLD = 100;  
private const int PAUSE_VERTICAL_ANGLE_LOWER_THRESHOLD = 80;  
private const int PAUSE_HORIZON_ANGLE_UPPER_THRESHOLD = 100;  
private const int PAUSE_HORIZON_ANGLE_LOWER_THRESHOLD = 80;
```

4.4.4 關卡一逃離猛獸—跑(Run) 演算法流程設計

由於逃離猛獸是需要玩家主動跑才會向前的遊戲，因此需要有跑步的動作。



圖六十二：右腿下，左腿上



圖六十三：左腿下，右腿上

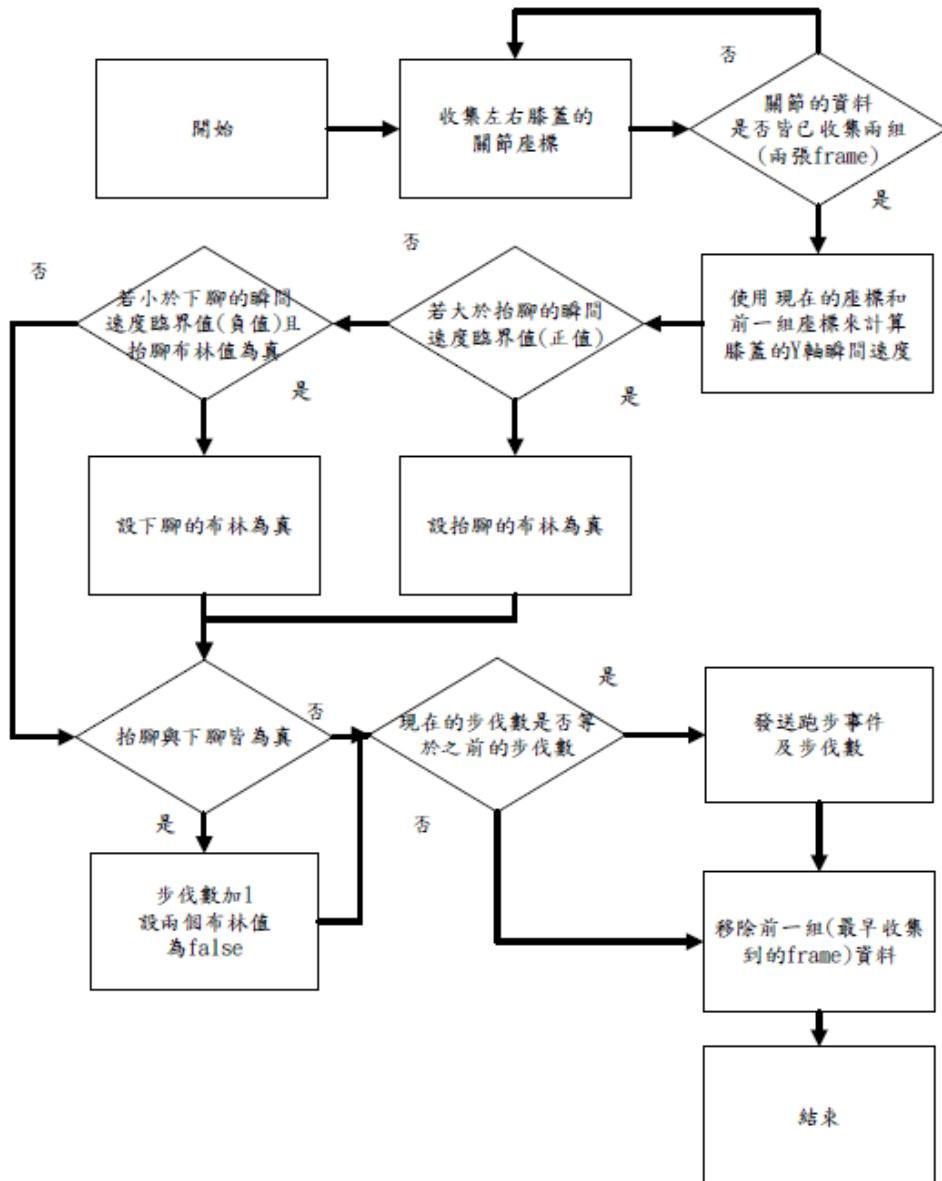
- 演算法概述：

跑步的明顯動作是雙腳會抬腿，也就是有抬膝的動作，而當大腿抬膝時，必定是一腳往下放，另一腳上抬，因此我們透過膝蓋的前後 2 張 frame 來計算瞬間速度是為向上，還是向下，而速度是否有滿足實驗所測出臨界值，當臨界值滿足時並，判斷有抬腿（或下腳）的動作。

透過 Boolean 的方式來得知雙腿是否皆有抬腿下腳(一上一下)的條件發生，但是，還有一個條件需要確認，舉例來說，當左腳下放時，依照常理，右腿是上抬的，也就是說，當我左腳速度與方向滿足下放時(圖六十三)，還得確認右腳是否是上抬（速度與方向），並透過雙腳的 Boolean 來作為開關判斷。

當上述的條件滿足後，腳步步伐的次數就會隨著增加，而判斷跑步的真正方式，就是透過步伐是否有更新來判斷，若步伐未更新，且滿足兩張 frame 時，就便跑玩家已無跑步。

- 流程圖：



圖六十四：跑(Run)演算法流程圖

- KinectModel.cs 程式碼：資料結構

```
private Dictionary<SkeletonJoint, LinkedList<SkeletonJointPosition>> jointNodeBufferForRun;
```

- KinectModel.cs 程式碼：累積 frame 張數的臨界值變數

```
Private int runTimeSpan = 2;
```

- KinectModel.cs 程式碼：判斷沒有跑步的所使用的 frame 變數

```
Private int runStableTimeSpan = 2;
```

當停止跑步時，抬膝的動作會停止，而 runStableTimeSpan 變數的目的是用來確定雙膝是否都各有停止。

- 收集的關節點：

SkeletonJoint
LeftKnee
RightKnee

- KinectModel.cs 程式碼：收集更新資料的函式

```
private void UpdateRunData(SkeletonJoint j, SkeletonJointPosition pos);
```

- KinectModel.cs 程式碼：演算法的函式

```
private void CalculateRunTimesTrigger(SkeletonJoint j, LinkedList<SkeletonJointPosition> listPos);
```

- KinectModel.cs 程式碼：相關臨界值變數

```
private int runLegUpVelocity = 10; //膝蓋向上抬的瞬間速度臨界值
```

```
private int runLegDownVelocity = -8; //膝蓋向下放的瞬間速度臨界值
```

- KinectModel.cs 程式碼：前一次的步伐變數

```
int preTotalSwingTimes
```

- KinectModel.cs 程式碼：現在的步伐變數

```
int curTotalSwingTimes
```

- KinectModel.cs 程式碼：記錄抬腳下腳的布林變數

```
private bool[] isLeftSwing; //跑步時,用來記錄左腳向上抬及向下放是否超臨界值,都有則為 true
```

```
private bool[] isRightSwing; //跑步時,用來記錄右腳向上抬及向下放是否超臨界值,都有則為 true
```

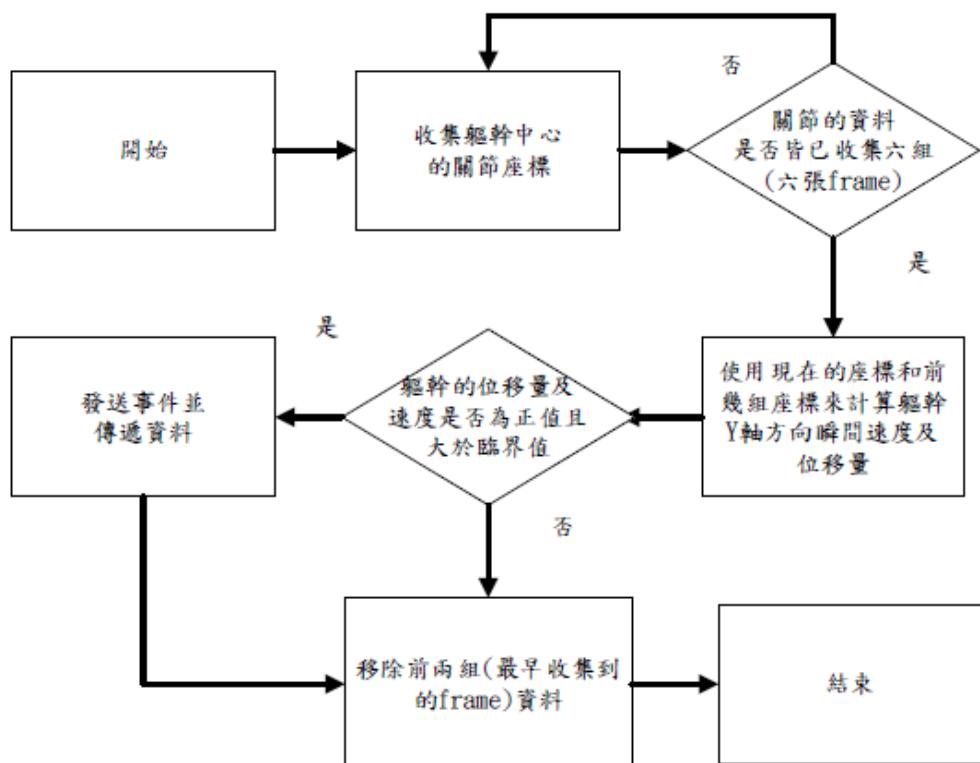
4.4.5 關卡一逃離猛獸—跳躍(Jump) 演算法流程設計

Jump 演算法是用來躲避一些石頭、河流等障礙物的動作，

- 演算法概述：

取得最後兩張 frame 的軀幹關節點，計算瞬間速度，若滿足臨界值條件，即為跳躍，而會用到 6 張 frame 的原因在於為了取得跳躍的位移量之緣故，並做為遊戲跳躍高度的數據。

- 流程圖：



圖六十五：跳躍(Jump)演算法流程圖

- KinectModel.cs 程式碼：資料結構

```
private Dictionary<SkeletonJoint, LinkedList<SkeletonJointPosition>> jointNodeBufferForSpeed;
```

```
//計算揮砍及跳躍用的資料結構 Buffer
```

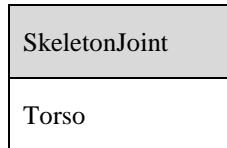
此資料結構由跳躍和揮砍共用，而主要原因在於累積 frame 數量一致的原故。

- KinectModel.cs 程式碼：累積 frame 張數的臨界值變數

```
private int jumpAndSlashTimeSpan = 6;
```

由實驗所取得的 frame 張數，發現跳躍和揮砍一致，所以共用此變數。

- 收集的關節點：



- KinectModel.cs 程式碼：收集更新資料的函式

```
private void UpdateSpeedData(SkeletonJoint j, SkeletonJointPosition pos)
```

由於揮砍和跳躍所使用 frame 張數的臨界值變數一致，所以除了收集的關節點的部分用判斷是分開外，都相同，因此更新資料的函式才共用。

- KinectModel.cs 程式碼：演算法的函式

```
private void CalculateJumpTrigger(LinkedList<SkeletonJointPosition> listPos);
```

- KinectModel.cs 程式碼：相關臨界值變數

```
//向上跳躍的部分,為預設值
```

```
private int jumpVelocityThreshold = 40; //也就是瞬間速度的臨界值
```

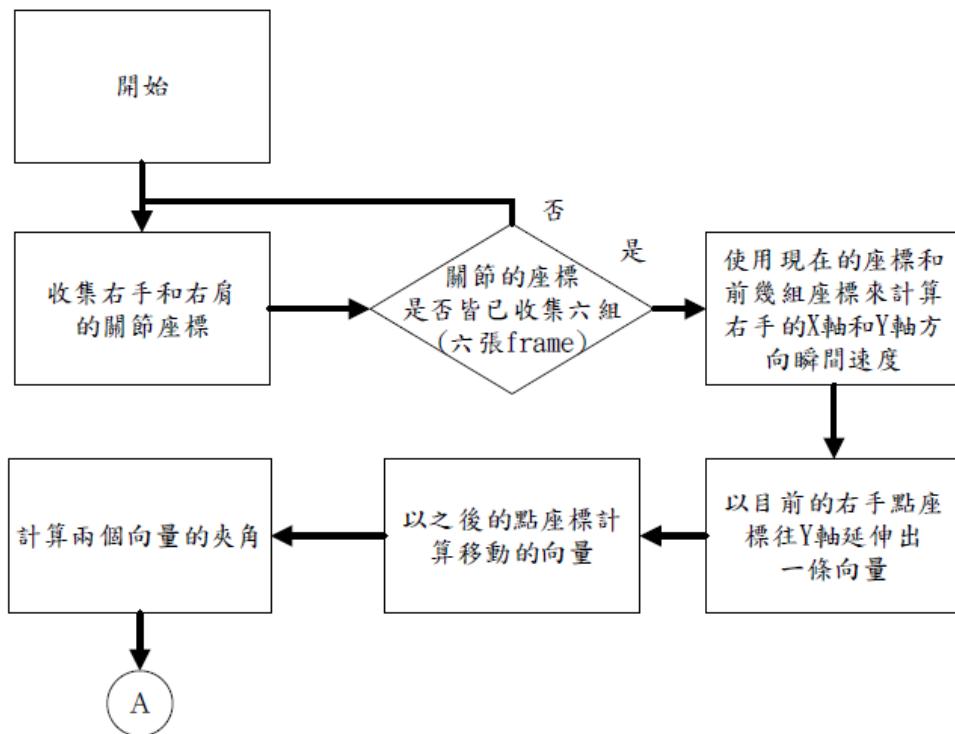
4.4.6 關卡一逃離猛獸—揮砍(Slash) 演算法流程設計

Slash 演算法用來砍倒遊戲中的樹叢障礙物，幫助前進。

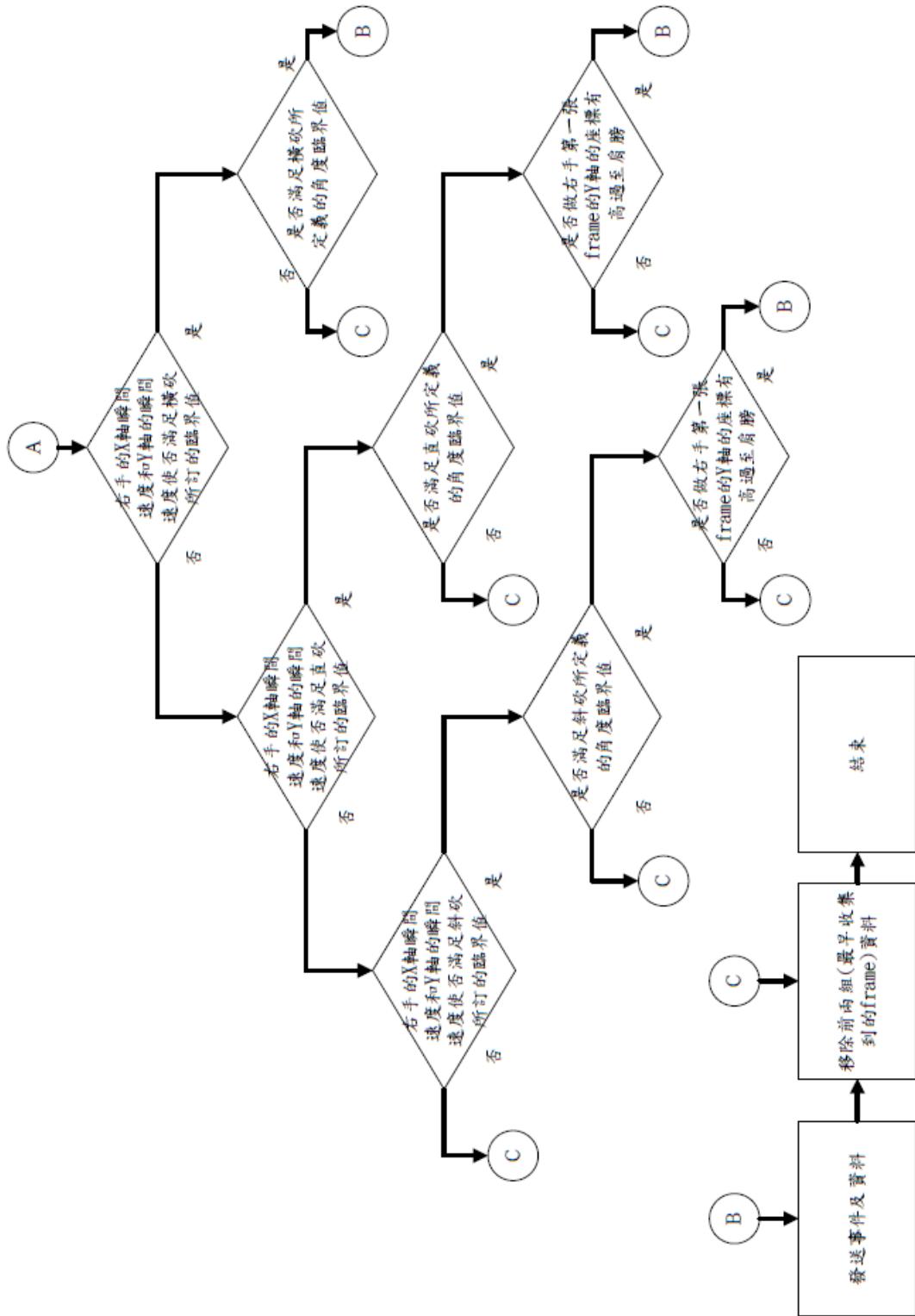
- 演算法概述：

揮砍分成水平、垂直與斜砍這三種砍法，而水平砍又依速度的方向分成左橫砍、右橫砍，斜砍也分成向左斜砍與向右斜砍；透過計算瞬間速度與角度(角度是目前揮砍的移動向量與假想的向量軸心做點積取得)，來判斷是水平砍、垂直砍還是斜砍。

- 流程圖：



圖六十六：揮砍(Slash)演算法流程圖一上



圖六十七：揮砍(Slash)演算法流程圖一下

資料結構、累積 frame 張數的臨界值變數、收集更新資料的函式皆和跳躍(Jump)一樣，所以此不再述說。

- 收集的關節點：

SkeletonJoint
RightShoulder
RightHand

- KinectModel.cs 程式碼：演算法的函式

```
private void CalculateSlashVelocityTrigger(Dictionary<SkeletonJoint,  
LinkedList<SkeletonJointPosition>> listPos);
```

- KinectModel.cs 程式碼：相關臨界值變數

```
// 挥砍部分  
  
private int slashRightLeftVelocity = 130; //橫砍的 X 軸瞬間速度  
  
private int slashDownVelocity = -110; //縱砍的 Y 軸瞬間速度  
  
private int slashMaxHorizonAngle = 110; //橫砍的水平角度(與假想的 Y 軸垂直向量做計算得知)  
  
private int slashMinHorizonAngle = 80; //橫砍的水平角度(與假想的 Y 軸垂直向量做計算得知)  
  
private int slashVerticalAngle = 30; //縱砍的水平角度(與假想的 Y 軸垂直向量做計算得知)  
  
private int slashRakeMaxAngleThreshold = 75; //斜砍的最大角度(與假想的 Y 軸垂直向量做計算  
得知)  
  
private int slashRakeMinAngleThreshold = 35; //斜砍的最小角度(與假想的 Y 軸垂直向量做計算  
得知)  
  
private int slashRakeRightLeftVelocity = 65; //斜砍的 X 軸瞬間速度  
  
private int slashRakeDownVelocity = -65; //斜砍的 Y 軸瞬間速度
```

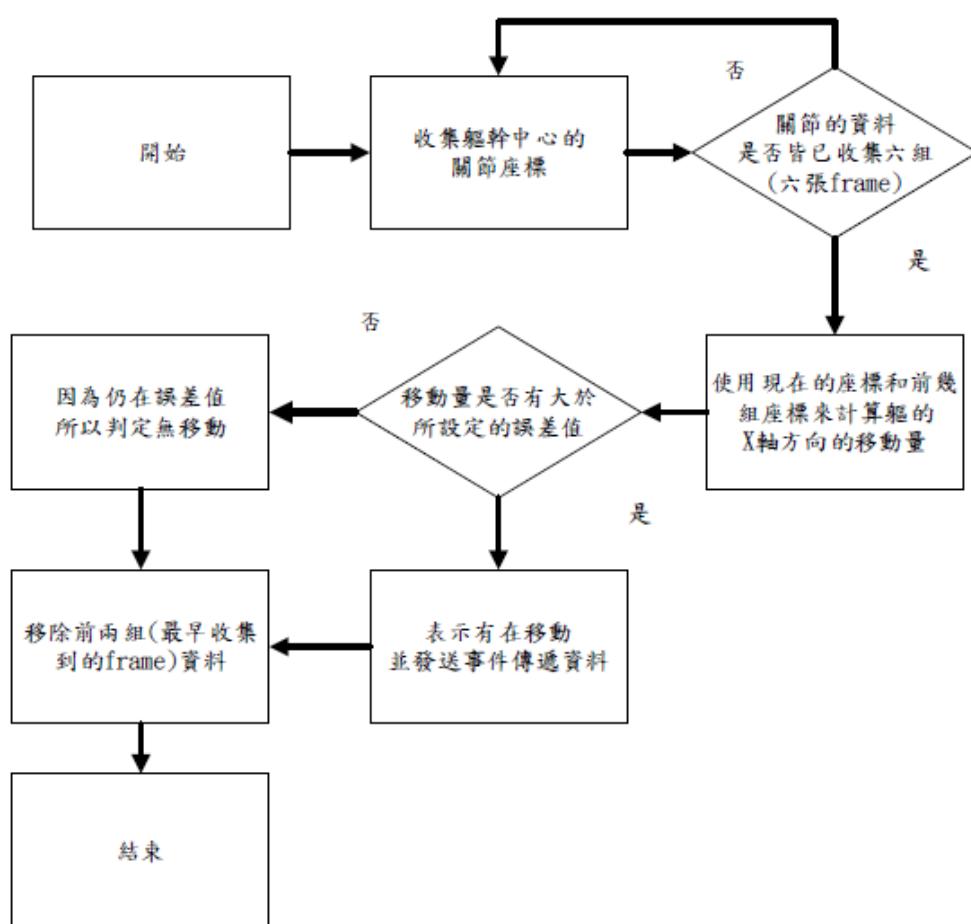
4.4.7 關卡一逃離猛獸—左右移動(Move) 演算法流程設計

Move 是用來在關卡一中左右移動，避開障礙物用。

- 演算法概述：

透過藉由前一組座標與現在的軀幹關節 X 軸座標，計算移動的速度向量，並判斷是否有超過臨界值，若超過，再藉由向量的正負判斷向左還是向右移動。

- 流程圖：



圖六十八：左右移動(Move)演算法流程圖

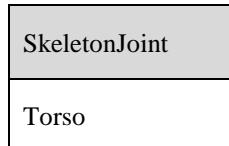
- KinectModel.cs 程式碼：資料結構

```
private Dictionary<SkeletonJoint, LinkedList<SkeletonJointPosition>> jointNodeForMove; //計算  
左右移動用的資料結構 Buffer
```

- KinectModel.cs 程式碼：累積 frame 張數的臨界值變數

```
private int moveRightAndLeftTimeSpan = 6;
```

- 收集的關節點：



- KinectModel.cs 程式碼：收集更新資料的函式

```
private void UpdateMoveData(SkeletonJoint j, SkeletonJointPosition pos);
```

- KinectModel.cs 程式碼：演算法的函式

```
private void CalculateMoveVelocityTrigger(LinkedList<SkeletonJointPosition> listPos);
```

- KinectModel.cs 程式碼：相關臨界值變數

```
private int deviationBorder = 10; //也就是平均速度的臨界值  
private int deviceXMaxBorder = kinect 鏡頭的 x 軸寬 - 40; /向右移動的到底時邊界  
private int deviceXMinBorder = 40; //向左移動到底時的邊界
```

deviceXMaxBorder 與 deviceXMinBorder 是確保玩家在左右移動超過螢幕時，可以把遊戲中的角色鎖在邊界，使遊戲的畫面邊界與 Kinect 鏡頭的邊界固定住。

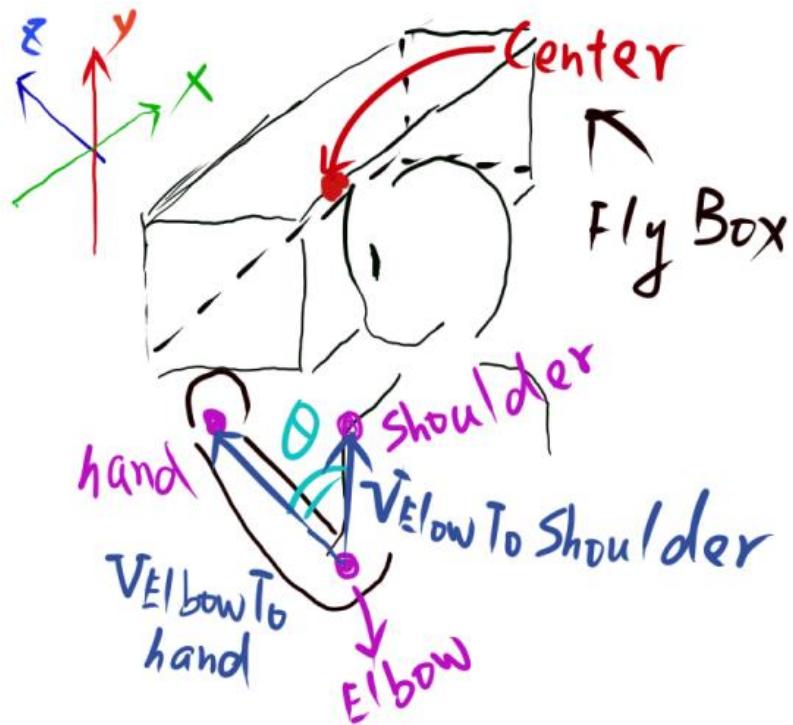
4.4.8 關卡二飛行危機—飛行(Fly) 演算法流程設計

飛行 (Fly)原本是用來作為關卡二的操作動作，此關卡是以操縱滑翔翼來進行，因此飛行方式的定義以 參考滑翔翼為主。

- 演算法概述：

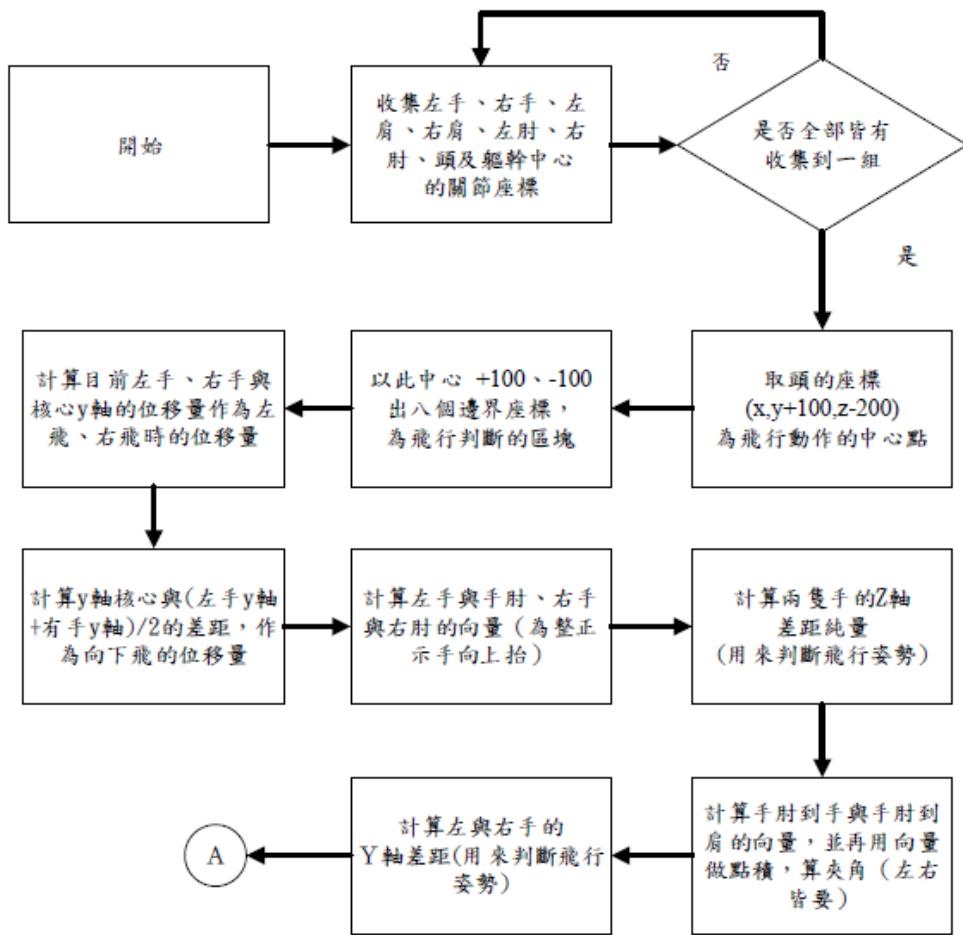
要如何判斷是飛行還是沒有飛行，是以手肘彎曲的角度(130 度內)以及手肘至手的 y 軸向量是否為正值來判斷，而判斷是否為直飛，則以設定一個飛行觸發區塊 FlyBox 來判斷，只要左手或右手是上抬得，且皆在此區塊，則是前直飛，若有任一手不在，如左手不在此區塊（此時左手仍是上抬，上抬以手肘的角度， $V_{ElbowToHand}$ 和 $V_{ElbowToShoulder}$ 彎曲為判斷）則為向左飛，反之亦然；比較例外的是，兩隻手皆在此飛型區塊（方型區塊 y 軸邊界的下方）的下方，同時兩手仍是向上抬，判斷微滑翔翼要向下飛行。

FlyBox 的核心 Center 是頭關節點 Head = (x,y+100,z-100)所構成，如下圖六十九。

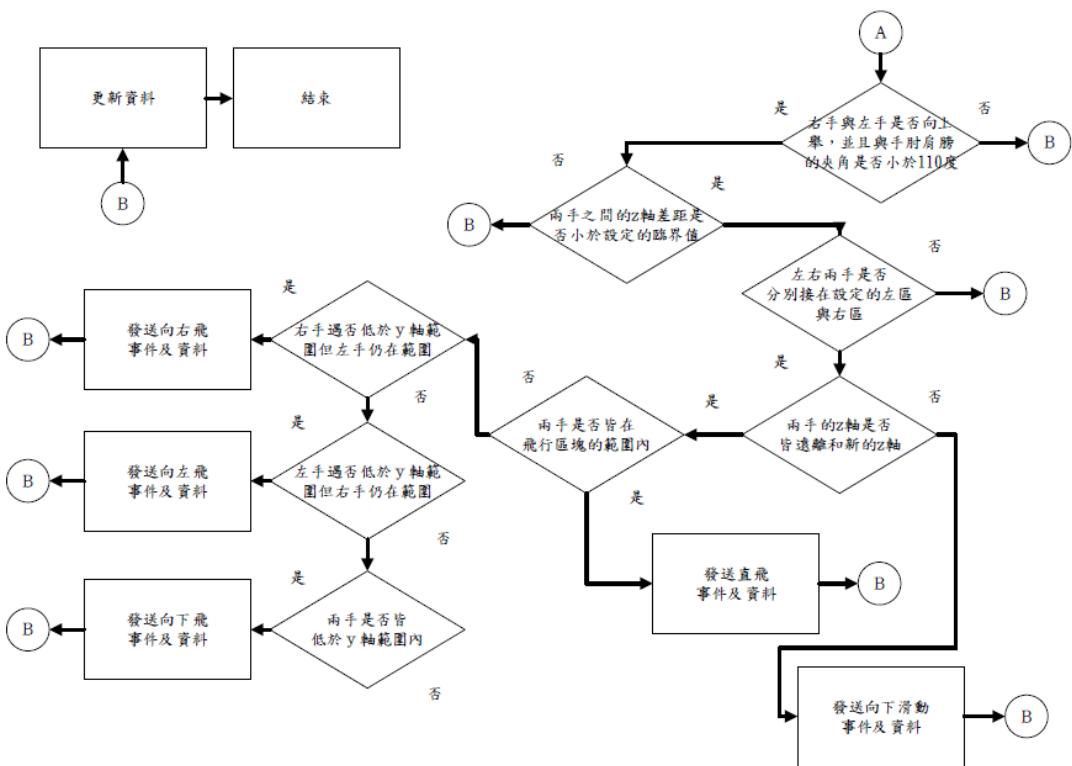


圖六十九：飛行的定義示意圖

- 流程圖：



圖七十：飛行(Fly)演算法流程圖一上



圖七十一：飛行(Fly)演算法流程圖一下

■ KinectModel.cs 程式碼：資料結構

```
private Dictionary<SkeletonJoint, SkeletonJointPosition> jointNodeBufferForFly; //計算飛行  
用的資料結構 Buffer
```

■ 收集的關節點：

SkeletonJoint
head
Torso
RightHand
LeftHand
RightElbow
LeftElbow
RightShoulder

LeftShoulder

- KinectModel.cs 程式碼：收集更新資料的函式

```
private void UpdateFlyData(SkeletonJoint j, SkeletonJointPosition pos)
```

- KinectModel.cs 程式碼：演算法的函式

```
private void CalculateFlyTrriger(Dictionary<SkeletonJoint, SkeletonJointPosition> jpos)
```

- KinectModel.cs 程式碼：相關臨界值變數

```
// 飛行部分  
  
private const int HANDS_Y_ORI_DIFFERENCE = 100;  
  
private const int HANDS_Z_ORI_DIFFERENCE = 300;  
  
private const int HANDS_ANGLE = 130;//手肘下放時彎曲的角度臨界值  
  
private const int RANGE_BOX_UP_THRESHOLD = 100;  
  
private const int RANGE_BOX_THRESHOLD = 100;
```

第五章 實作成果

5.1 開發環境建置

硬體裝置：Kinect + AUX To USB 轉接線

Kinect 驅動程式：SensorKinect-Win-OpenSource32-5.0.3.4

開發工具：Visual Studio 2010、XNA4.0、OpenNI-Win32-1.3.2.3-Dev.msi、NITE-Win32-1.4.1.2-Dev.msi

作業系統：Windows 7

顯示卡支援：GT130

5.2 專案檔案解說

1. XNA 遊戲專案(Game_Kinect_Project 文件夾)：

- ◆ Game_Kinect_Project.sln：整個專案的主要專案檔，此專案會包含遊戲主框架、DisplaySimulation 專案及其他需參考到的專案。
- ◆ Game1.cs：專案執行的起始點，此檔案會開起遊戲主要執行以及 kinect 部分的 Thread。
- ◆ Screen 資料夾：處理遊戲中畫面切換的檔案都在此，其中 ScreenManager 為管理所有 Screen 的中樞。
- ◆ GameSpace 資料夾：裡面為關卡一及關卡二的所有規則定義等等，其關卡實作也在此。
- ◆ AudioManager.cs：遊戲中所有音樂皆為此管理。
- ◆ BloomComponent.cs：遊戲中霧化效果的特效處理，其中 code 皆為參考而來。
- ◆ Camera.cs：遊戲中場景的攝影機，控制此項便可在 3d 場景中做視野的切換。
- ◆ Graphic2D.cs：遊戲中所有 2D_UI 介面的處理。
- ◆ InputManager.cs：遊戲中所有控制皆為此管理，包含從 kinect 傳遞而來的訊號。

2. MrozKinect 專案：負責把 Kinect 截取的骨架資訊與其坐標系統，轉換至遊戲 3D 人物模型的骨架上，使得可以操控遊戲中的 Model

3. HeightMap 專案：實作 2.6.3 章節 heightmap 地形演算法的程式

4. SkinnedModel 專案：實作執行 3D 模型的程式

5. Kinect 專案(DisplaySimulation 文件夾)：

- ◆ DisplaySimulation.sln：在開發時，Kinect 程式與遊戲是分開開發的，因此，為了能夠使 Kinect 獨立開發並測試，才有此專案檔。
- ◆ KinectDisplay.cs：KinectDisplay 此專案的 View，為了呈現 Kinect 動作設計的結果。
- ◆ KinectModel 文件夾：
 - KinectActionModel.cs：包含了所有的動作設計演算法與啟用演算法的事件定義，此類別需依賴 KinectDevice.cs 才能啟用。
 - KinectDevice 文件夾—KinectDevice.cs：此類別封裝了啟動 Kinect 與骨架抓取等所有方法。
 - BehaviorStruct 文件夾—專門定義當動作事件發生時，要傳入的資料結構，如：SwipeBehaviorData、JumpBehaviorData....等等。
 - StateEnum 文件夾—自定義了各個動作演算法中可能需要、判斷用的指令，因為指令眾多，因此透過 Enum 來方便呈現與閱讀，如：SwipeBehaviorActionEnum、UserStateEnum、SlashBehaviorEnum....等等。

5.3 遊戲成果畫面

- ◆ 進入遊戲，須先取得使用者的架資訊，下圖七十二為擺出 Psi 動作



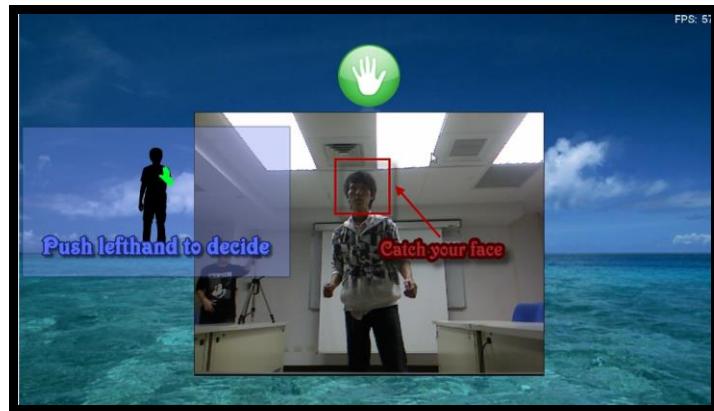
圖七十二：遊戲指示擺出 Psi 的圖片

- ◆ 遊戲選單，從左至右分別為開始、計分板、說明、離開。



圖七十三：選單遊戲圖，其中左圖是動作教學

- ◆ 撷取人物頭像之畫面，作為記分板記錄之用



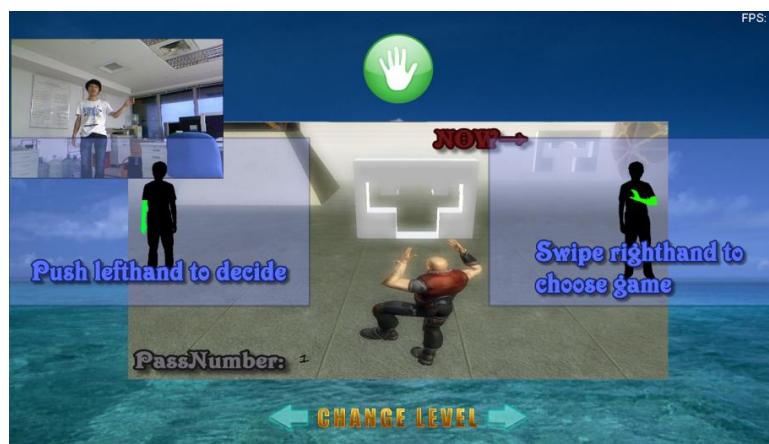
圖七十四：臉部截取圖

- ◆ 選擇遊戲的畫面一逃離猛獸關



圖七十五：逃離猛獸的關卡畫面，左上角是實際畫面

- ◆ 選擇遊戲的畫面一姿勢模仿



圖七十六：姿勢模仿的關卡畫面

- ◆ 逃離猛獸遊戲畫面，開始的怪獸圖，如果追到玩家會 gameover



圖七十七：逃離猛遊戲畫面，開始畫面

- ◆ 玩家須不斷的左右閃躲、跳躍、揮砍來前進，中途會有障礙。



圖七十八：遊戲進行中畫面，右上方是血量

- ◆ 逃離猛獸，失敗畫面



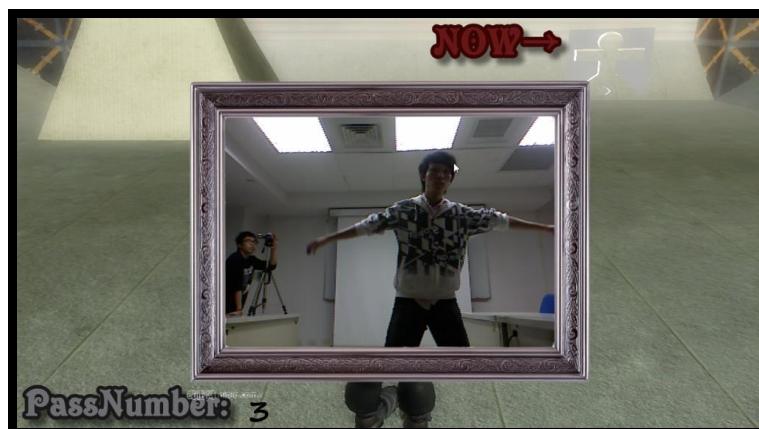
圖七十九：失敗畫面

- ◆ 姿勢模仿的遊戲畫面，遊戲中的人物姿勢與真實世界中的玩家姿勢一樣，玩家必須做出與板子相同動作才可通過。



圖八十：姿勢模仿遊戲畫面，左下角是通過數，右上角是目前的板子

- ◆ 通過版子後會擷取玩家影像



圖八十一：通過板子時，會截取玩家現實畫面

- ◆ 未通過版子的失敗圖



圖八十二：失敗效果畫面

- ◆ 通過版子的效果成功圖



圖八十三：通過畫面

- ◆ 遊戲的計分畫面



圖八十四：兩關各自的排行榜

- ◆ 擺動姿勢的畫面



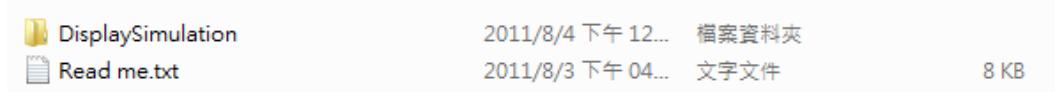
圖八十五：玩家現實中畫面

第六章 Kinect 訊號端程式使用手冊

6.1 與訊號端 Kinect 程式銜接步驟

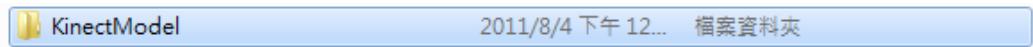
請依照下列順序和 Kinect 程式銜接。

1. 打開 KinectSignal_v1.7.53 的資料夾，方案名為 SimulaeEnviroment(圖八十六)

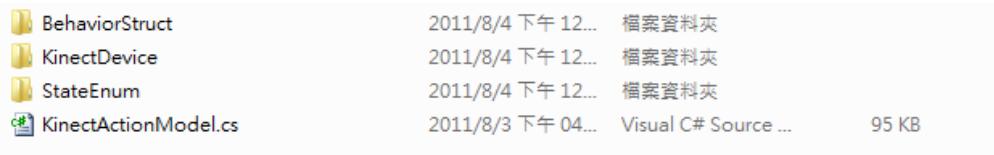


圖八十六：方案資料夾

2. 裡面有 KinectModel 的檔案夾(圖八十七)，裡面包含了所有 Kinect 程式的.cs 檔(圖八十八)

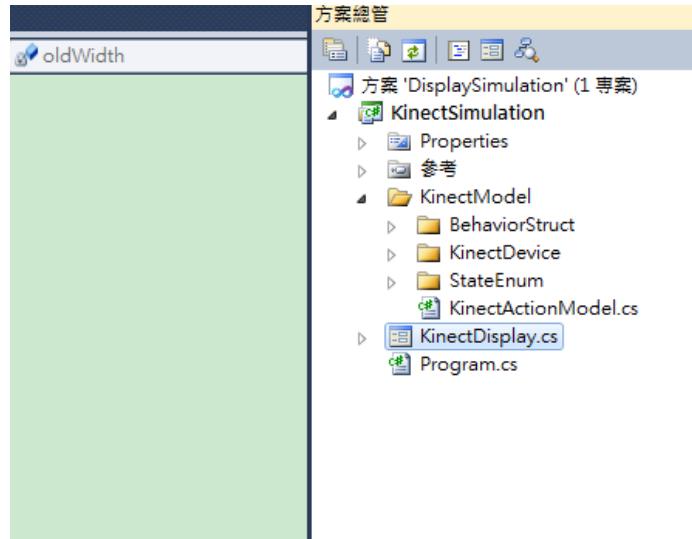


圖八十七：KinectModel 資料夾



圖八十八：KinectModel 底下的資料夾及.cs 檔

3. 把 KinectModel 的檔案夾加入至你的方案中(圖八十九)



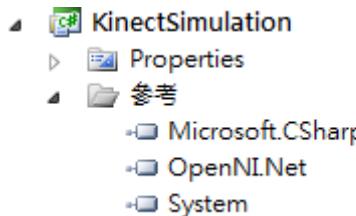
圖八十九：(範例檔)方案的畫面

4. 加入三個 Xml 檔(圖九十), xml 檔應該是從方案資料夾往下兩層放置(否則會和 Kinect 部分使用到的路徑衝突, 但也可以調)

licenses.xml	2011/4/25 下午 0...	XML Document	1 KB
modules.xml	2011/4/25 下午 0...	XML Document	2 KB
SamplesConfig.xml	2011/4/18 上午 1...	XML Document	1 KB

圖九十：此三個 xml 檔須加入

5. 在程式專案加入 OpenNI 參考(圖九十一)



圖九十一：OpenNI.Net

6. 在程式中加入 Using Namespace(圖九十二)

```
//////////  
//銜接Kinect訊號要加入的命名空間  
using OpenNI;  
using KinectModel;  
using KinectModel.BehaviorStruct;  
using KinectModel.StateEnum;  
//////////
```

圖九十二：會用到的 Namespace

6.2 Kinect 遊戲的動作訊號分類及實作

1. 訊號分類：

✓ 不分類：

- 骨架更新(SkeletonUpdate)
- 使用者狀態(UserState)

✓ 關卡類：

- ◆ 暫停(Pause)
- 關卡一(逃離猛獸)
 - ◆ 挥砍(Slash)
 - ◆ 跑步(Run)
 - ◆ 跳躍(Jump)
 - ◆ 左右移動(MveLeftAndRight)
- 關卡二(飛傘危機)
 - ◆ 飛行(Fly)

✓ 選單類：

- 推(Push)
- 滑動(Swipe)

2. 訊號的實作順序

所有的實作方式與範例，可以參考 KinectDisplay.cs 類別，此類別的原先目的是用來開發時的測試，可已把此類別當作一個想用使用 Kinect 的獨立程式。

I. 把 KinectActionModel 類別宣告物件、初始化，並加入製程式碼中(圖九十三)，此時已開始對底層 KinectDevice 做資料撈取的事件註冊等動作。

```
//銜接Kinect訊號要用到的類別，裡面包含了許多的訊號事件註冊  
KinectActionModel KinectAction;
```

圖九十三：含有許多訊號事件的類別，可以透過註冊使用

II. 註冊想要的事件(此舉例滑動 Swipe 訊號，圖九十四、九十五)



圖九十四：事件的註解

```
KinectAction.OnSwipe += new SwipeBehaviorEventHandler(KinectAction_OnSwipe);  
KinectAction.OnNotSwipe += new NoBehaviorMessageEventHandler(KinectAction_OnNotSwipe);
```

圖九十五：事件的註冊

III. 使用 Thread 方式跑 KinectActionModel.ModelDoWork(圖九十六、九十七)

```
private Thread KinectWorkerThread;
```

圖九十六：宣告執行緒(要先 Using System.Threading)

```
this.KinectWorkerThread = new Thread(new ThreadStart(KinectAction.ModelDoWork));  
this.KinectWorkerThread.Start();
```

圖九十七：執行緒的啟動法

IV. 對註冊的事件做函式處理(圖九十八)

```
private void KinectAction_OnSwipe(SwipeBehaviorData swipe)  
{  
    swipeData = swipe;  
    isSwipe = true;  
}  
private void KinectAction_OnNotSwipe(string message)  
{  
    swipeMessage = message;  
    isSwipe = false;  
}
```

圖九十八：事件處理函式

V. 若要記錄資料，可以使用提供的資料結構型態，每一種事件皆有不同的回傳資料型態(表二十八)

事件名稱	事件傳遞的參數型態
OnHandPush	PushBehaviorData
OnHandPushStable	String
OnFootRun	int
OnFootRunStable	
OnHandSlash	SlashBehaviorData
OnHandSlashStable	string
OnJump	JumpBehaviorData
OnJumpStable	
OnPause	double,double
OnWaitingPause	int
OnNotPause	
OnMoveLeftOrRight	MoveBehaviorData
OnMoveLeftOrRightStable	
OnFly	FlyBehaviorData

OnNotFly	string: 參考 Fly BehaviorMessageEnum
OnBoxChanged	FlyRangeBoxData
OnSwipe	SwipeBehaviorData
OnNotSwipe	string :參考 SwipeBehaviorMessageEnum
OnUserStateChanged	int, UserStateEnum
OnJointsUpdate	Dictionary<SkeletonJoint,SkeletonJointPosition>
OnDepth	DepthMetaData

表二十八：資料結構型態

6.3 訊號使用及功能介紹

6.3.1 不分類訊號

1. 骨架更新：若有註冊此事件，則當使用者校正取得骨架後，事件會不斷的把每次更新的骨架資訊發送出去。

- ◆ 事件名稱：

```
xxx.OnJointsUpdate+=new SkeletonJointUpdateEventHandler(xxx_OnJointsUpdate);
```

- ◆ 處理事件的函式引數：骨架資訊

```
Private void xxx_OnJointsUpdate(Dictionary<SkeletonJoint, SkeletonJoint Position>pos)
{
    Do something...
}
```

2. 使用者狀態：表現目前使用者的狀態，唯有在改變狀態時才發生，可以用來在畫面上顯示狀態，狀態的類型請參考 UserStateEnum 列舉

- ◆ 事件名稱：

```
xxx.OnUserStateChanged+=new UserStateChangedEventHandler(xxx_OnUserStateChanged);
```

- ◆ 處理事件的函式引數：

```
Private void xxx_OnUserStateChanged (int user,UserStateEnum state)
{
    Do something....
}
```

- ◆ UserStateEnum 包含：

名稱	描述
LookingForPose	尋找擺出校正姿勢的人
Calibrating	校正中
Tracking	追蹤使用者(表校正成功)
TrackingButWaring	追蹤 但站的位置太近或太遠
ReTracking	重新追蹤
StopTracking	停止追蹤(使用者不見)

NewUser	新的人進入畫面(尚未追蹤)
LostUser	遺失使用者(包含無追蹤和有追蹤的人)
UserReEnter	離開畫面的使用者重新進入畫面
UserExit	使用者離開畫面(包含無追蹤和有追蹤的人)
JointUserExit	有骨架的使用者離開畫面
JointUserMissed	有骨架的使用者遺失(因為不知名而發生)
JointUserLost	有骨架的使用者不見(在畫面中找不到)

表二十九：UserStateEnum 解說

6.3.2 關卡類

1. 暫停(每關卡皆須要)：

把左手臂向左方水平抬起，則會記錄累積時間，只要累積時間(3秒)，暫停訊號才會發生，不動作時也會有不動作的事件發生，此外，累積時間時，會發生等待暫停的訊號，告知使用者目前的時間條。

- ◆ 暫停事件名稱：

```
xxx.OnPause+=new PauseBehaviorEventHandler(xxx_OnPause);
```

- ◆ 暫停處理事件的函式引數：成水平擺放時的角度(請參考四、訊號演算法)

```
Private void xxx_OnPause (double verticalAngle, double horizonAngle)
{
    Do something....
}
```

- ◆ 沒進入暫停的事件名稱：

```
xxx.OnNotPause+=new NoBehaviorEventHandler(xxx_OnNotPause);
```

- ◆ 沒進入暫停處理事件的函式引數：無引數

```
Private void xxx_OnNotPause ()
{
    Do something....
}
```

- ◆ 等待暫停事件名稱：

```
xxx.OnWaitingPause+=new TimesChangedEventHandler(xxx_OnWaitingPause);
```

- ◆ 處理事件的函式引數：時間數(最大值為 90 個 frame = 3 秒)

```
Private void xxx_OnWaitingPause (int waitingTimes)
{
    Do something....
}
```

6.3.2.1 逃離猛獸關

1. 跑步：

此訊號是使用者在跑步時才會有的發生，以每秒為單位記錄一秒內的踏步數，沒在跑步時也會有不動作的事件發生，可以選擇是否兩個事件皆註冊。

- ◆ 跑步事件名稱：

```
xxx.OnFootRun+=new TimesChangedEventHandler(xxx_OnFootRun);
```

- ◆ 處理事件的函式引數：踏步數

```
Private void xxx_OnFootRun (int runTimes)
{
    Do something....
}
```

- ◆ 沒跑步事件名稱：

```
xxx.OnFootRunStable+=new NoBeahviorEventHandler
(xxx_OnFootRunStable);
```

- ◆ 處理事件的函式引數：無引數

```
Private void xxx_OnFootRunStable ()
{
    Do something....
}
```

2. 右手揮砍：

右手水平衡揮、垂直下砍、斜砍等動作若有達到速度會觸發各自的動作事件

- ◆ 揮砍事件名稱：

```
xxx.OnHandSlash+=new SlashBeahviorEventHandler(xxx_OnHandSlash);
```

- ◆ 處理事件的函式引數：揮砍的資料結構

```
Private void xxx_ OnHandSlash (SlashBeahviorData slashData)
{
    Do something....
}
```

- ◆ 沒揮砍事件名稱：

```
xxx.OnHandSlashStable+=
new NoBeahviorMessgaeEventHandler(xxx_ OnHandSlashStable);
```

- ◆ 處理事件的函式引數：無引數

```
Private void xxx_ OnHandSlashStable ()
{
    Do something....
}
```

- ◆ SlashBehaviorEnum 列舉：

名稱	描述
LeftHorizonSlash	向左水平揮動
RightHorizonSlash	向右水平揮砍
VerticalSlash	由上往下(要過肩膀)揮砍
LeftRakedSlash	由上往下(要過肩膀)往左方斜揮砍
RightRakedSlash	由上往下(要過肩膀)往右方斜揮砍

表三十：SlashBehaviorEnum 列舉解說

3. 跳躍：

向上跳事件就會發生，沒動作時也會有沒動作的事件。

- ◆ 跳躍事件名稱：

```
xxx.OnJump+=new JumpBehaviorEventHandler(xxx_ OnJump);
```

- ◆ 處理事件的函式引數：跳躍的資料結構

```
Private void xxx_ OnJump (JumpBehaviorData jumpData)
{
    Do something....
}
```

- ◆ 沒跳躍事件名稱：

```
xxx.OnJumpStable+=new NoBehaviorEventHandler(xxx_OnJumpStable);
```

- ◆ 處理事件的函式引數：無引數

```
Private void xxx_OnJumpStable ()  
{  
    Do something....  
}
```

4. 左右移動：

人物左右移動時會發生的事件，左移會有左移的訊號、又移則有右移的訊號，區分用字串。

- ◆ 移動事件名稱：

```
xxx.OnMoveLeftOrRight+=new MoveLeftAndRightBehaviorEventHandler(xxx_  
OnMoveLeftOrRight);
```

- ◆ 處理事件的函式引數：移動的資料結構

```
Private void xxx_OnMoveLeftOrRight (MoveBehavior moveData)  
{  
    Do something....  
}
```

- ◆ 沒移動的事件名稱：

```
xxx.OnMoveLeftOrRightStable+=new NoBehaviorEventHandler(xxx_OnMoveLeftOrRightStable);
```

- ◆ 處理事件的函式引數：無引數

```
Private void xxx_OnMoveLeftOrRightStable ()  
{  
    Do something....  
}
```

6.2.4.2 關卡二—飛行危機關

1. 飛行

飛行訊號發生的條件，要先把雙手微舉高至額頭高度，且兩手的手肘約呈九十度才會發生，若手舉過高或是兩手手肘角度過大，則會有不動作的訊號發生，此外也有判斷飛行訊號的範圍也會有事件發生，裡面記錄了現在可飛行範圍的中心點

及邊界座標。

- ◆ 飛行事件名稱：

```
xxx.OnFly+=new FlyBehaviorEventHandler(xxx_OnFly);
```

- ◆ 處理事件的函式引數：飛行的資料結構

```
Private void xxx_OnFly ( FlyBehaviorData flyData)
{
    Do something....
}
```

- ◆ 不是飛行時的事件名稱：

```
xxx.OnNotFly+=new FlyBehaviorEventHandler(xxx_OnNotFly);
```

- ◆ 處理事件的函式引數：字串，訊息請參考 FlyBehaviorStateMessage 列舉

```
Private void xxx_OnNotFly ( string message)
{
    Do something....
}
```

- ◆ FlyBehaviorStateMessageEnum 包含：

名稱	描述
RightHand_Z_Point_Over_Shoulder	左手 Z 軸太靠近肩膀
LeftHand_Z_Point_Over_Shoulder	右手 Z 軸太靠近肩膀
RightHand_UpLift_Over_High	右手舉過高
BothHands_UpLift_Over_High	左手舉過高
RightHand_Position_Not_In_RightZone	左手 x 軸不在身體左邊
LeftHand_Position_Not_In_LeftZone	右手 x 軸不在身體右邊
RightHand_Not_UpLift	右手沒有舉起來
LeftHand_Not_UpLift	左手沒有舉起來
RightElbow_And_RightShoulder_Not_Vetical	左肘和肩膀的角度太斜
LeftElbow_And_LeftShoulder_Not_Vetical	右肘和肩膀的角度太斜
RightHand_And_LeftHand_Z_Axies_Difference_Over	左右手的 Z 軸相差太遠
Correct	正確
Init	初始化
User_Z_Position_Over_TwoDotFour_Meter	使用者在太前面(小於 2.4 公尺)

表三十一：FlyBehaviorStateMessageEnum 解說

- ◆ 飛行訊號的範圍改變事件名稱：

```
xxx.OnBoxChanged+=new FlyRangedBoxChanged(xxx_OnBoxChanged);
```

- ◆ 處理事件的函式引數：飛行訊號的範圍資料結構

```
Private void xxx_OnBoxChanged (FlyRangeBoxData rangeBox)
{
    Do something....
}
```

6.3.3 選單類

1. 推(Push)

只要左或右手向前作推的動作即會發生，無動作時也會有事件發生，主要用來做為按鈕選擇確定時使用。

- ◆ 推的事件名稱：

```
xxx.OnHandPush+=new PushBehaviorEventHandler(xxx_OnHandPush);
```

- ◆ 處理事件的函式引數：推的資料結構

```
Private void xxx_OnHandPush (PushBehaviorData pushData)
{
    Do something....
}
```

- ◆ 沒有推的事件名稱：

```
xxx.OnHandPushStable+=new NoBehaviorMessageEventHandler
(xxx_OnHandPushStable);
```

- ◆ 處理事件的函式引數：字串、LeftHand 或 RightHand

```
Private void xxx_OnHandPush (string whichHand)
{
    Do something....
}
```

2. 滑動(Swipe)：

推是向前，而滑動則分成上下左右，只要朝某一方向揮動，且平均速度夠則會發生此事件，若沒有做出動作時也會有沒動作的事件，此功能可以用在做一些畫面選擇上的特效，如向左滑，則畫面會有向左捲動之效果等。

- ◆ 滑動的事件名稱：

```
xxx.OnSwipe+=new SwipeBehaviorEventHandler(xxx_OnSwipe);
```

- ◆ 處理事件的函式引數：滑動的資料結構

```
Private void xxx_OnSwipe (SwipeBehaviorData swipeData)
{
    Do something....
}
```

- ◆ 沒有滑動的事件名稱：

```
xxx.OnNotSwipe+=new NoBehaviorMessageEventHandler(xxx_OnNotSwipe);
```

- ◆ 處理事件的函式引數：字串，訊息參考 SwipeBehaviorState - Enum 列舉

```
Private void xxx_OnSwipe (string message)
{
    Do something....
}
```

- ◆ SwipeBehaviorState Enum 包含：

名稱	描述
Init	初始化
Correct	正確
Swipe_Up_AngleBetweenVerticalOrientation_Over	往上揮時揮的太前面
Swipe_Up_AngleBetweenHorizonOrientation_Over	往上揮時揮太斜
Swipe_Down_AngleBetweenVerticalOrientation_Over	往下揮時揮的太前面
Swipe_Down_AngleBetweenHorizonOrientation_Over	往下揮時揮太斜
Swipe_Right_AngleBetweenVerticalOrientation_Over	往右揮時揮的太前面
Swipe_Right_AngleBetweenHorizonOrientation_Over	往右揮時揮太斜
Swipe_Left_AngleBetweenVerticalOrientation_Over	往左揮時揮的太前面
Swipe_Left_AngleBetweenHorizonOrientation_Over	往左揮時揮太斜
Swipe_Velocity_NotEnough	揮的速度不夠
Swipe_Length_NotEnough	手伸的不夠前面，就揮動

表三十二：SwipeBehaviorState Enum 解說

第七章 結論與未來展望

7.1 結論

本專題結合了 Kinect 硬體設備、openNI API 與 XNA 遊戲框架，透過 Kinect 深度攝影機截取人體姿勢與關節骨架資料，並且以 openNI 函式庫來處理輸入端取的畫面，處理完畢之後把使用者表達之動作形成訊號變成參數傳入遊戲輸入端，使玩家能夠利用身體動作操縱遊戲，進行關卡，達成人機互動之最終目的。

由於是第一次使用不同的輸入端來供給遊戲，所以在輸入端的處理設計上，沒有更加彈性，如動作的演算法部分，不是採用學習訓練，而是透過觀察得來經驗值，去設定判斷觸發的邊界值。

在遊戲設計方面本專題利用了許多圖學技巧來撰寫構成整個場景，從一開始 2D 平面圖形架構起，到最後的 3 維立體遊戲主畫面，這當中利用了很多圖層、數學理論與畫面後製的原理；而程式碼方面之撰寫也利用 OO 之設計的概念來完成本專題遊戲部分之實作；

7.2 未來展望

1. 遊戲的強化

本遊戲由於是純資訊背景的關係，所以美工方面非常微弱，希望在畫面方面未來能夠加強，另外能夠把遊戲的架構在昇華，使之能夠變得更加彈性方便抽換內容。

2. 非遊戲類的互動設計

Kinect 對於互動的應用與影像的處理無疑是一大幫助，希望能夠把互動的程線拓展到遊戲之外，如：家電的操作、機器人的眼睛、互動廣告、互動藝術、互動教學等等方面。

參考文獻

[1] T客邦，身體就是控制器，微軟 Kinect 是怎麼做到的？

<http://www.techbang.com.tw/posts/2936-get-to-know-how-it-works-kinect>

[2] [創新趨勢] 身體就是控制器：初探 Kinect 背後的 3D 測量技術

<http://blog.uns.org.tw/node/417>

[3] 微软 Project Natal 三维测量原理

http://blog.sina.com.cn/s/blog_461db08c0100is8o.html

[4] Heresy's Space OpenNI / Kinect 相關文章目錄

http://kheresy.wordpress.com/index_of_openni_and_kinect/

[5] Windows Kinect Driver/SDK - CL NUI Platform Release v1.0.0.1210

<http://codelaboratories.com/forums/viewthread/442/>

[6] Microsoft Kinect SDK vs PrimeSense OpenNI

<http://www.cnblogs.com/TravelingLight/archive/2011/06/20/2085149.html>

[7] How you Become the Controller

<http://bbs.a9vg.com/thread-1662418-1-1.html>