國立台北大學資訊工程學系專題報告

多人連線即時互動擴增實境-AR Dota 雙城魔域

專題組員 ： 周聿晟、邱楷程、徐執恩、金峻暘

專題編號 ：PRJ-NTPUCSIE-105-004

執行時間 ： 105 年 07 月至 106 年 06 月

一、摘要

隨著智慧型手機近年來的大量普及，新型穿戴式裝置的快速研發，許多嶄新的技術也相繼被推出。擴增實境(Augmented Reality)、虛擬實境(Virtual Reality)已從數十年前的理論技術，變成現今的趨勢潮流，甚至發展出了近年來興起的混合實境(Mixed Reality)。

目前擴增實境的普及程度，已經到達在小小的一台手持型智慧裝置上安裝一個簡單的應用程式，就可以擁有虛擬 3D 物件場景融合在真實世界中的體驗了；而硬體設備的演進也促進了穿戴式虛擬影像設備的發展，更進而有了周邊體感設備的出現，使得實境效果不僅限於視覺與聽覺，更擴及了身體各個感官的刺激。

至此，「擴增實境」不再只是一個名詞，更是融入在我們生活中隨處可見的一項重要技術，而我們這次的專題開發就是致力於研發出目前市面上尚未普及的 AR 應用，並且只需要一支智慧型手機就能夠實現了。

二、介紹

一開始在構想專題的方向時，正逢當時 pokemon go 大為流行，而 AR 遊戲也漸漸在新興遊戲市場中佔有一席之地，黃俊堯教授大力推薦我們把擴增實境跟遊戲做結合。但只有擴增實境的效果並不能滿足現代人對遊戲體驗與質感的要求。參考現今火紅的手機遊戲，我們發現＜皇室戰爭＞、＜傳說對決＞等 DOTA 類型的遊戲十分受到現代玩家們的青睞，遊戲人氣居高不下，而本專題最終方向就是將擴增實境加入到 DOTA 遊戲之中，使玩家能夠在融合真實與虛擬的世界中作互動。

我們希望本專題帶給使用者了不僅是 AR 方面的遊戲體驗，也加上了 GPS 的定位功能，兩種功能的結合本身是現今遊戲市場所沒有的，再加上即時連線的特色，想必一定能夠帶來遊戲界一股嶄新旋風。

遊戲進行方式

此次專題的遊戲類型為「第一人稱」與「DOTA」的結合，玩家將化身魔法師，在遊戲中施放法術，與怪物們展開攻防。

在一局遊戲裡，玩家將會分成紅藍兩派陣營，雙方玩家各自擁有一個主城堡，此城堡為該方陣營所需守護的主塔。若讓城堡的防禦值歸零，將會導致己方失敗。

由於我們本次專題的主題是擴增實境，城堡在手機畫面中將會像是實際建築物一樣，換言之，它將會按照實際建物比例，座落在玩家周圍，使玩家在進行遊戲時，就像有一棟建築物矗立在眼前。

那麼？玩家該如何對敵方主城發

動攻擊呢？隨著遊戲的推進，將會有金幣散落在遊戲地圖的各個角落，玩家可以藉由移動到錢幣處的方式來獲取該金幣，

攢夠了足夠的金幣後，可以到已方城堡旁的商店區召喚自己想要的怪物，怪物將會依照戰鬥力的高低來制定對應的價格。成功召喚怪物後，怪物將會出現在己方主塔前，並朝著敵方主塔移動，展開攻勢。

要防禦來自敵方主塔的怪物，有兩種主要的攻擊怪物的方式，
第一種，是在己方主塔前產生新的怪物與之相抗。第二種，則是使用手上的槍枝親自對怪物造成傷害。將手機鏡頭瞄準怪物，點擊畫面將會發射子彈，玩家將需要根據當時戰況來選擇適合的攻擊策略，以獲得最終勝利。

本遊戲還結合了 GPS 定位系統，由於 Pokemon Go 的成功，我們發現除了一指打天下，隨走隨玩更是新一代的遊戲潮流。因此本遊戲拿掉了虛擬搖桿，取而代之的，便是 GPS 的定位移動，玩家若想在遊戲中朝某一方向前進，需要親自往該方向移動，藉由手機 GPS 的定位數值的變換，經過卡氏座標的換算，玩家在遊戲中的位置也隨之改變。

因為此遊戲為網路連線遊戲，因此我們也使用了資料庫的相關技術，使得玩家可以擁有個人帳號，在遊戲中的戰績，遊戲信息都可以儲存在 Server 端的資料庫中，本專題的最終成果，將是一完整的網路連線 AR 手機遊戲。

三、專題進行方式

本專題所接觸的領域包含 Unity 實作、網路 Socket 連線架構、資料庫系統、AR 擴增實境&GPS 定位，以下將各自做進一步探討

1. Unity 實作

本次開發所使用的為 Unity 遊戲引擎，選擇 Unity 的主要原因為它的市占率高，且操作容易，C#為基底的設計概念更是方便設計者可以直觀的進行關卡建置與佈署。

但選用 Unity 也帶來了一些不便，因為 Unity 有太多模組已經寫死，導致一些技術只能依照其特定的方式去操作，其中尤為明顯的是移動 GameObject 時，若使用 Unity 內建的 translate 函式，將無法使其他機器達成座標同步的效果，此現象成了 Unity 實作中的一大瓶頸。以下將探討該問題與其解決方式。

首先，Unity 有許多內建好的函式可供設計者使用，例如方才提到的 translate( )就是一例，將 GameObject 依照輸入的參數做移動，如此在「單機」遊戲中是可行的，但在網路連線遊戲中，我們需要時刻保持所有參與遊戲的玩家看到的畫面能夠同步，因此希望在某玩家 A 接受到來自使用者的 input 之後，先將其 input 儲存，並且傳送至 Server 端，再由 Server 端廣播至 B C D 玩家處，接著所有玩家再將 A 玩家的 input 設置在遊戲中，但是 translate 函式並沒有辦法先不執行。因此不能在 translate 加入 Socket 連線中的 Send 函式。

若直接傳輸參數，等各個接收端接受參數再個別執行 translate 函式，經過測試之後，發現 A B C D 的使用者看到的同一 GameObject 之位置會有落差，應是網路延遲所導致。

我們最終採用直接傳送位置座標資訊，也就是說，當接收到使用者對某 gameobject 發出的 input 時，先推算出該 gameobject 的最終 position，再將 position 的值藉由網路傳送，在每 deltatime(每個 frame 的切換時間) 都會更新一次，如此在每次更新時，就是順便做了一次全體位置的校正(因為 Server 每次廣播給各接收端的 Position 必定一樣)

上述作法雖然解決了各玩家遊

戲不同步的問題，但也因此對遊戲開發造成影響，如許多在遊戲場景中的物件，如:轉動的金幣、玩家召喚的怪物…等等，有許多 Unity 的相關技術需要建置在 translate 函式中,例如怪物的動畫(Animator)，原本在單機遊戲中，怪物的動作透過場景中 Prefab 拉入 Scripts 中即可操作,在函式中呼叫動作(state)的參數即可。但換成多人連線遊戲時，由於我們使用的是直接更改 Position,就不能直接的做使用。解決辦法必須從新的 Prefab 獲取動畫組件。

上述情況在我們的開發過程中履見不鮮，克服這些「連線」問題所帶來的挑戰，是本次專題的關鍵.

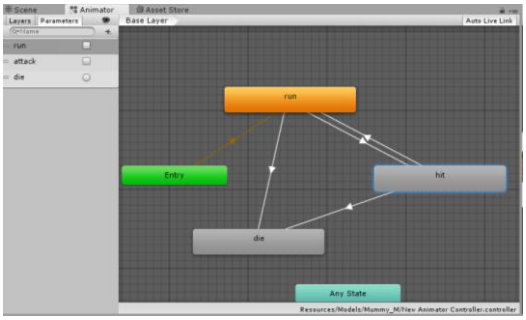
3.1.2 遊戲金幣的製作

先在場景中建一個雙層 Cylinder 然後去中央銀行官網下載國幣的圖片，轉成.png 檔後,將貼圖貼在 Cylinder 上，這樣金幣就出來了

至於要轉動的部份:
使用 Unity 內建的角度函式去讓它旋轉，並且為了遊戲好玩度，讓金幣隨機生成在場景中的某個地方

3.1.3 怪物的製作

將完整的怪物模組貼上貼圖後在場景中設定成 Prefab 才能重複在場景種使用，將每種怪物的 Prefab 都製作一個新的 AnimatorController,依照當初的遊戲設定，從遊戲商店生成怪物就開始做(跑或行走的動作)，並且朝著對方城堡前進，用 Unity 一個很好用的方法→tag 去偵測是否為敵方怪物或城堡，才去作攻擊動作。
在程式中，藍色方的怪物的 tag==BlueMonster 而紅色的為 RedMonster 藍色城堡為 Blue 而紅色城堡為 Red，這麼一來，便可以在城市中判斷，使用 OnTriggerStay() or OnTriggerEnter()判斷 tag 為敵方則作攻擊動作。至於如何做動作的切換，先在 AnimatorController 面板上拉好動作的觸發及銜接方式，中間的箭頭便是觸發條件----→----



使用 SetBool(“動作名稱”,T or F);

只要在程式中將某個 bool 值設為 True or False 便會跳至下一個動作執行(動作先設定會 Loop 不斷的重複執行)
死亡的動作也是類似概念，但是我們使用了
gameObject.GetComponent<Animator>().SetTrigger("die");
只要執行 die 便會執行死亡的動作，並且過 2 秒後 Destroy 物件，也就是怪物從場景中消失。怪物的死亡便是透過 UI Slider 的減少去判斷，若 Slider 值為 0，則觸發死亡動作(UI 血條 Slider 見 1.3)

3.1.4 遊戲商店的製作

我們照著我們起初的遊戲設計，在雙方的城堡旁都設有各自的商店(塔)，當偵測到玩家踏入(觸碰)到塔時，遊戲商店 UI 便會開啟，這個商店是整個遊戲非常重要的元素之一，想要贏得這場勝利就必須進入商店招喚不同等級的怪物去攻打敵方，想要補自己城堡的血量也是由商店裡購買，但購買這些商品的前提就是必須擁有足夠的金幣。
遊戲商店的製作便是運用 Unity 提供的 UI Panel 及 Button 等等的元素結合在一起。

3.1.5 UI 血條的製作

為了增加遊戲的好玩度及豐富度，我們加了扣血及補血的機制在城堡及怪物身上，利用 UI 的 Slider 製作血條，並且在函式中用 OnTriggerEnter()、OnTriggerStay() 函式去偵測若生成的不同方怪物接觸及持續觸碰另一方的城堡時，則減少 Slider 的值；OnTriggerExit()則是判斷若怪物死亡或離開則停止扣血，進而改變遊戲場景中的城堡血條。

2. 資料庫系統

在資料庫的實作方面，是為了帶給每位玩家一個自行管理的帳戶，擁有自己的戰績及好友互動名單，在實際操作上，在 unity 平台的架構下，我們使用了 mysql 一個被廣泛使用而且易於取得的資料庫管理系統，利用 php 作為中間的溝通媒介，讓 unity 能夠對 mysql 下指令並且操作它，而資料庫也可以經由 php 回傳 unity 所需要的資訊，在 php 的語法中，每一次的操作都會使用 mysqli 為兩端建立一個新的連線，連線建立成功後，由 unity 端對資料庫使用 mysql 語法下指令，並且將要傳的資料用 post 來將其送進變數中，而每個指令和變數都會經由一個 query 做包裝後送進資料庫，再使用一個變數去存取資料庫輸出的結果，在經由此變數作判別之後執行相對應的動作，再將結果傳回至 unity 端，而 unity 使用了 www script 去接收 php 所回傳的資料，並在遊戲中顯示或作為判斷，使用資料庫的目的不僅是建立玩家帳戶系統，也幫助遊戲控管房間的資訊，當遊戲中 server 意外中斷時，資料可以被安全的保存在資料庫中，而不會因此消失，而資料庫在整個專題裡面扮演的腳色就是能輔助整個遊戲的進行，讓遊戲的分工更加明確。

在實作上遇到的困難:

在建立資料庫時一直是對本機進行連線和資料傳遞，而如何在不同的裝置之間做溝通是在製作時遇見的一個困難點。

解決方法:

再參考許多網路上的文獻後，才發覺自己並未使用到 php mysql 遠端連線，而方法是新增一個使用者帳號並給予其特定資料表的權限，使其對特定的資料表操作，並將 server 端的 IP 修改為資料庫架設電腦所在的網域，以此方式就可以達到與 server 溝通的效果。

而因為我們在製作階段時，使用的是 wifi 網路，而 IP 並非實體 IP，所以在不同的環境下 IP 都會更動，所以在每次的測試及修改都要改變 IP 位址才能正確的連線。而在 server 端的網路架構會在網路的部分做詳細的講解。

3. 網路連線架構

網路連線遊戲中的連線通訊永遠是遊戲中最重要的架構之一，通訊中的每一則訊息決定了每一個畫面的呈現，所有的應用與遊戲體驗也是基於穩定的網路連線與封包的處理。因此，如何設計連線架構，讓使用者體驗與遊戲效能達到平衡，對我們的 AR 遊戲來說是一門非常重要的課題。

在建立 Server－Client 架構時，我們使用 C#原始的 TCP socket 與 Stream 物件，設計自己的傳輸演算法，來達成封包處理與傳送功能。

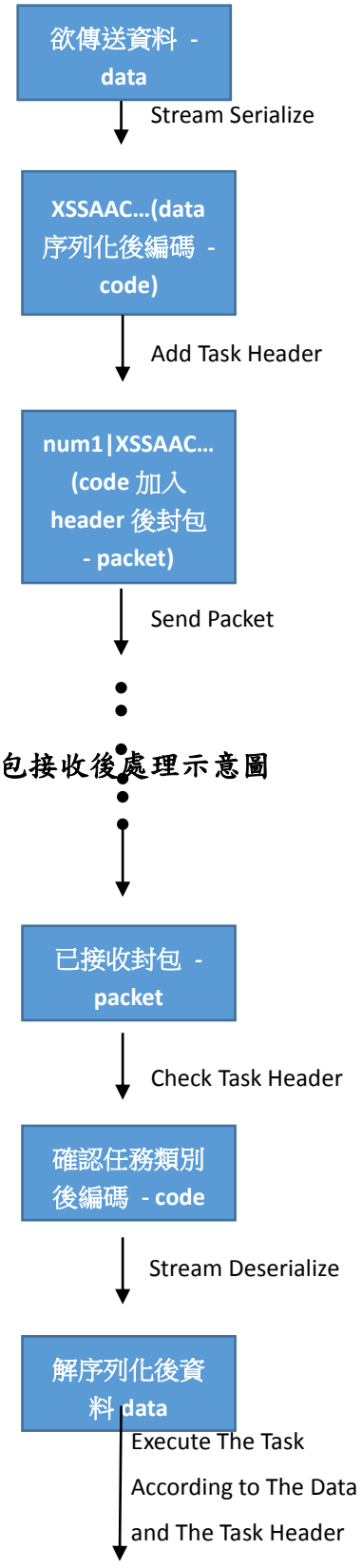
Packet processing(封包處理)：

在通訊雙方需要傳送資料時，我們使用了 C#中的資料流(Stream)物件將需要傳送的資料作序列化(Serialize)，並且將其轉換成字元編碼，從而在整串編碼前面加入此封包要做的封包任務識別標頭，再將轉換後的字元編碼寫入資料流中。

在接收到封包以後，我們先判端所收到的封包任務識別標頭，判斷出該封包隱含了什麼樣的訊息、要做什麼樣的任務。再來同樣使用 C#的資料流物件將檔頭後的一長串字元編碼

解序列化(Deserialize)成原來的資料物件，將原本雜亂的字元編碼變成有用的資訊。

↓封包傳送前處理示意圖



↓封包接收後處理示意圖

Server 通訊端 ：

只要提到網路連線架構，不外乎就是眾所周知的 Server – Client 架構。Server 通訊端必須負責與資料庫的聯繫，同時也要回應每個 client 的各種 request，抑或是主動傳送 data 給 client。

在演算法方面，server 的 main thread 將開始監聽 client 的連線請求。為了將 server 的訊息處理速度提升到最快，只要一接收到一個 client 的連線請求，就會開啟一個 thread 負責 handle 該 client 的 request，只要一個 client 連接上，就會有一個 thread 負責不斷的 check 該 client 的資料流中是否有資料，而一開始的 main thread 則在新的 port 開始繼續監聽是否有新的連線請求。除此之外，server 也會持續不斷的在確認 client 是否有資料傳送過來之前，確認該 client 的連線狀態，判斷其是否離線，以免將資料寫入已關閉的資料流中造成錯誤中斷。

↓server 通訊端監聽演算法

```
ListenForConnect()
    AcceptedNewClient()
    NewThread(ClientData)
    ListenForConnect()
```

↓server 通訊端封包交換演算法

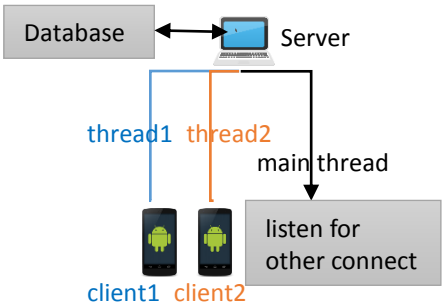
```
EachThead(ClientData)
    while (true)
        CheckForDisconnected()
        CheckForPacketAvailable()
        Process(packet)
```

而在資料結構方面，server 會動態存放著所有玩家的個人資料以及房間資訊 ： 所有玩家資訊(含 ID、帳戶、密碼、遊玩狀態、房間房號)都存放在 server 的 client list 當中，

而所有的房間資訊(房間房號、房間玩家清單、房間名稱、房間密碼)也都存放在 server 的 room list 當中，以方便 server 在處理 client request 時方便透過這些資料結構尋找出所需的資訊。

除了動態存放資料之外，在與資料庫的連接方面，server 也必須根據 client 所傳送過來的封包形式與資料，在不同的情況下判斷要如何交換資料庫的資料，以及要如何將 client 傳送過來的資料與資料庫的資料做對比。舉例來說，在玩家發送登入請求時，回傳送所輸入的帳號密碼，server 在收到封包資訊後，確認封包的 header 為登入要求，則會從玩家所傳送過來的資料切割成帳號、密碼，將帳戶資訊傳入資料庫做搜尋，並且取得資料庫的 response，再將登入請求結果回傳給玩家。

↓server 架構示意圖

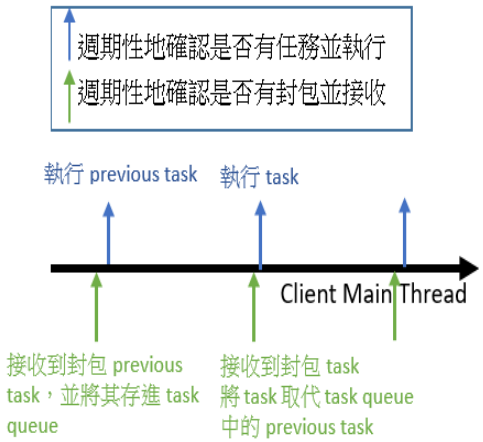


Client 通訊端 ：

每一個遊戲玩家(user)都代表一個 client 通訊端，必須處理每個 server 傳送過來的封包，根據該封包的內容決定該執行的任務。除此之外，每個 client 也會對遠端的 server 發出 request，例如物品的購買、遊戲開始、建置遊戲空間、載入場景…等。同時 client 端也動態存放著遊戲中所有房間與所有上線玩家的資訊，以讓 UI 能夠正常顯示。

而為了讓網路封包處理速度加快，我們曾經嘗試了 multi-thread 的技巧，利用兩個 thread，其中之一負責不斷確認是否有封包的傳送，另一個則負責進行遊戲畫面的更新處理。然而使用了這個做法，卻讓我們遭逢了由於手機記憶體大小較小，又開了兩個 thread 以增進封包處理速度，導致遊戲效能非常的差，畫面更新極為緩慢。最後我們在封包處理速度與效能上做了權衡，刪去了原本用來做封包處理的 thread，而改用 main thread 在固定時間週期性地確認是否有封包從 server 通訊端傳送過來，若有，則處理該封包，確認該封包的任務後將該任務存進任務緩衝區佇列(以下稱 task queue)中；同時也週期性的確認 task queue 是否為空值，若不是則執行 task queue 中第一個 task。如此一來手機的效能與遊戲體驗果真有所改善，但由於封包處理速度較開啟 multi-thread 時慢，因此在畫面更新上也有些延遲的狀況產生。為了解決此問題，我們將每個周期所接收到的 task 封包都放在 task queue 的第一個位置。若原本在 task queue 第一個位置的 task 仍尚未執行，則將其取代，並且也週期性的讓 task queue 中的第一個 task 執行。(註 ： 只對於有取代性的封包做以上處理，如玩家位置更新、玩家移動方向等；若是一般的物品購買或遊戲場景轉換則採用直接執行的方式)

↓client 端封包接收與畫面更新示意圖



4. GPS 定位

在遊戲中的移動依據，就是依靠玩家的雙腳，透過玩家本身實際移動所得的經緯度座標，再經過運算及轉換，將真實世界中的實際位置，變成我們所創造的虛擬 3D 空間中的卡式座標。

我們以台北大學的校園經緯度座標為依據，標記出四個頂點連成一個正方形，計算出其中心點後，再將四個頂點經過旋轉矩陣，讓正方形的邊長與經緯度平行，以下為示意圖。

將地圖經旋轉運算後，接著就是將其從實際的經緯度座標調整成我們創造的 3D 虛擬空間大小，並可以坐落於相對應的位置。首先計算出中心點的經緯度，將我們的實際位置經緯度減去中心點經緯度，再透過線性縮放，預設出虛擬空間的地圖大小之後，便對實際經緯度乘上相對應的比例大小，投射到 3D 空間中變為卡式座標。

5. 擴增實境效果

陀螺儀資訊取得 ：

在手機遊戲的擴增實境中，其中最核心的技術，就是根據陀螺儀的傾斜角度計算出旋轉的角度，並套用到視角上進行轉換。

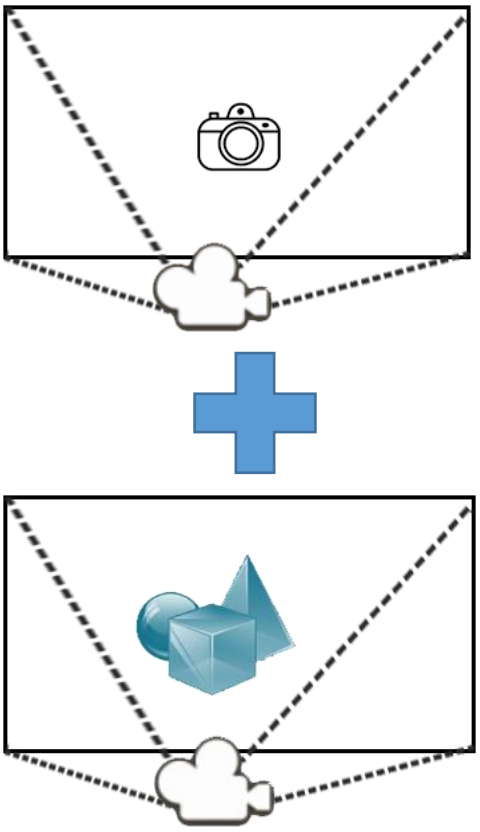
我們將手機內的陀螺儀資訊取得後，對其進行四元數(由於用三維的歐拉角會產生萬象死鎖的現象)的外積運算，並且將運算結果加入視角轉軸的相對參考，調整數值做校正，讓手機畫面中的視角可以根據手機的各個旋轉方向做出相對應的改變，旋轉至對的轉軸，達到 360 度的全方位視野變換。

↓公式轉換

視角相對旋轉量=陀螺儀傾斜角四元數 x (0, 0, 1, 0)

相機背景 ：

相機所擷取的每個 frame 畫面都會被我們當成遊戲的背景，也代表了擴增實境中的真實世界。同時，我們也會將遊戲中具有互動性的物件置入遊戲場景，並且由兩個不同的視野:其一投影相機所擷取的每個畫面，並且只針對相機畫面進行投影；其二則為投影 3D 世界中的虛擬物件，並且將二者的畫面融合，而相機畫面則作為背景置後，3D 物件則置前，如此一來就達到虛擬物件在真實世界中浮現的效果。以下為相機背景與 3D 物件融合的示意圖。



如此一來，結合前面所述的陀螺儀視角轉換，就可以讓玩家身歷其境的體驗擴增時境的虛擬與真實融為一體的感受了。

四、主要成果與評估

經過實測，整個遊戲的運行大致順利，從遊戲開始到結束十分順暢。遊戲結果也能夠傳送至資料庫作儲存，但仍存在尚待改進之處。

一．耗電量：

由於AR Dota使用多項手機服務，包含GPS、陀螺儀裝置資訊、相機鏡頭、每秒數十次的封包處理與交換，使得手機耗電量偏大。

二．硬體：

由於必須開啟多項手機服務，記憶體較小的手機可能無法負荷，導致應用程式閃退的現象；另外較舊款的手機中有些也沒有陀螺儀感測器裝置，將無法體驗擴增實境的效果。

三．同步：

虛擬場景中所有物件互動的同步化，每個玩家遊戲中的互動事件都是在用戶端自行運算所得的結果，沒有統一的事件發生傳訊，有可能會因為每台手機的運算速率不同，或網路頻寬問題，而導致不同的事件結果。

五、結語與展望

在我們的遊戲中，在技術面還有許多地方尚待改進，同時在遊戲內容也有許多可以讓遊戲體驗更豐富的部分，例如增加怪物的AI互動，3D物件的豐富度增加，都可以讓我們的遊戲更吸引人、更有自己的特色。

Augmented Reality 擴增實境是時下備受關注的一門技術，它尚有許多未知的層面等著人們去研究，本次專題只是AR的一小部分應用而已，相信未來，類似的遊戲很快的便邁入商業化，這些擴增實境所帶來的樂趣，將會在你我的生活中佔有一席之地。

六、銘謝

感謝黃俊堯教授的指導，在決定專題方向之時期，教授提到架構的重要性，組員們是第一次接觸大型Project，剛開始時就直接進行Code方面的鑽研，並沒有系統化的架構，感謝教授點出了許多將任務條理化的方法，使研發事半功倍。

七、參考文獻

陀螺儀:
http://stackoverflow.com/questions/24516794/what-sensor-can-be-used-to-detect-rotation-when-upright

GPS:
http://blog.xuite.net/trshang/myarticles/expert-view/24343951