Final Project Report

遊戲名稱：數位戰車遊戲 App

這是一款多人戰車對戰遊戲，由手機藍芽連線控制自己的戰車，分為兩人對戰和四人對戰的兩種模式，畫面流暢清晰，適合在讀書讀累時與是有一起連線對戰唷！

組員與分工：

姓名與學號	分工項目
102060013 陳柏儒	1. 遊戲大綱規劃，NIOS移動邏輯，遊戲繪圖，碰撞邏輯撰寫 2. SOPC 系統建立
102060020 李宇哲	1. 手機端藍芽研究 2. 手機端操控介面APP撰寫 3. 藍芽晶片模組訊號解析模組(UART Decoder)撰寫

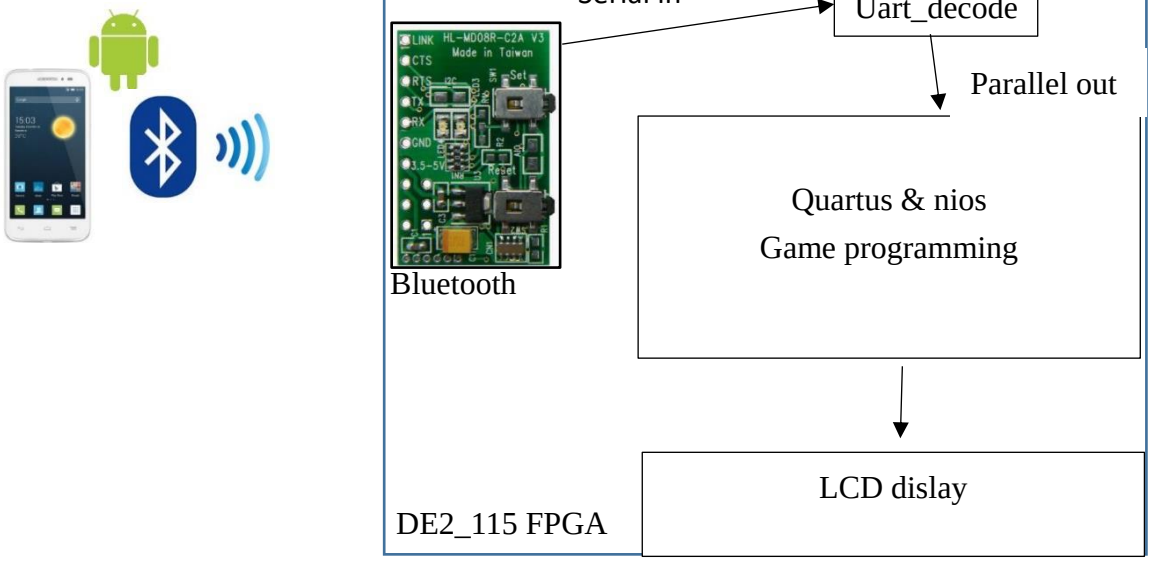
系統介紹

一、遊戲系統概述

我們的這次 project 主要的工具有(DE2-115)FPGA 實驗板子、藍芽模組及連接的杜邦線、Android 手機內的 App。我們用以上的工具互相通訊聯繫、傳遞資料，建構了這次 project 的遊戲系統。

玩家會利用 Android 系統寫好的手機 App 進行遊戲，將控制的訊息利用藍芽傳輸到藍芽模組上，藍芽模組再藉由 TX 的 pin 角將資料傳送到 FPGA 的板子，最後再藉由軟硬體相互的溝通將資料顯示在 LCD Display 上。詳細的系統架構和建構時遇到的問題會再以下討論。

二、遊戲系統架構與傳輸協定

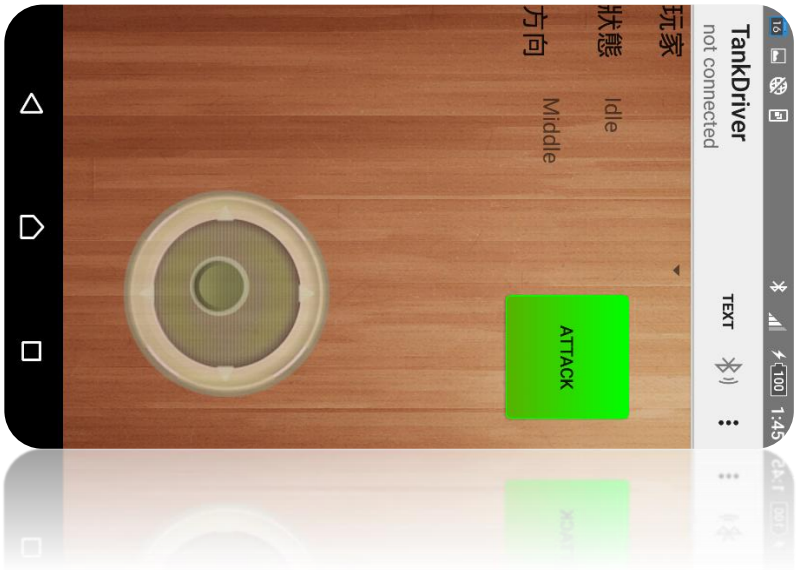


1. 手機端 Android App 架構

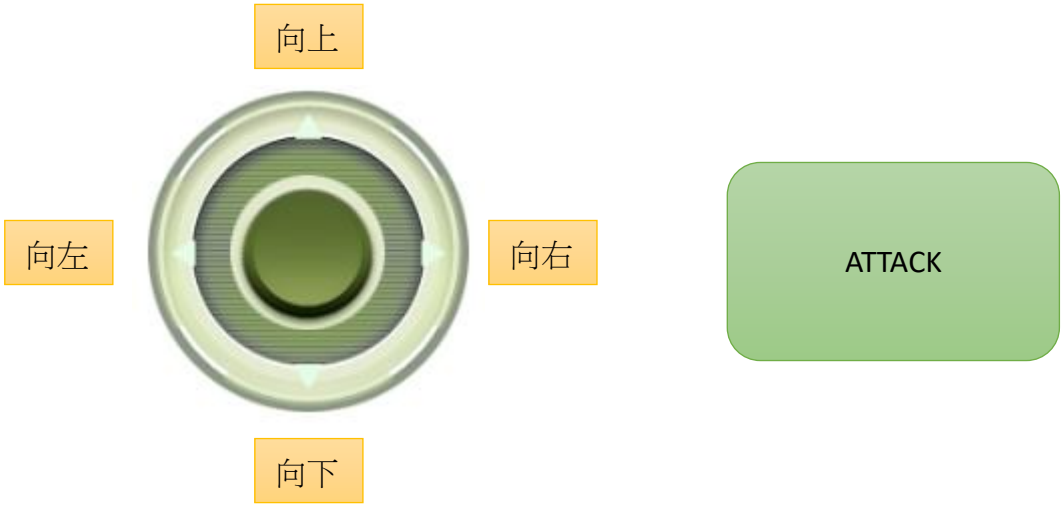
App 的名稱為 TankDriver，在 App 使用者的介面上，我們會在使用者開啟這個 App 的時候，要求使用者開啟藍芽裝置，否則執行退出程式的動作。然後在使用者進來後，可以在 App 頂端的 Action Bar 上的一個藍芽圖式的功能鍵搜尋配對的藍芽裝置，使用者可以在這裡看到自己手機裝置裡面以前配對過的裝置，或者是藍芽搜尋到附近的裝置，若使用者要連到 FPGA 的 bluetooth 的話，我們會要求使用者進行配對才能傳送資料到藍芽器上。

而在 App 的主要功能鍵包含 Shoot Button 和 Joy Sticker，前者是控制遊戲中的戰車角色是否要射擊，後者是控制戰車的方向。

手機 App 的介面如下



左邊綠色的圖案就是 Joy Sticker，它模擬一般電玩遊戲的搖桿，使用者用手指壓在螢幕上的 Joy Sticker 時可以控制遊戲中角色的不同方向，讓使用者即使在手機上還是可以感受到玩著搖桿的感覺。如下：



而右邊的 Shoot 按鈕就是在使用者按下後可以控制戰車的攻擊狀態，當按下時遊戲中的戰車就會發動攻擊。至於 App 上還會印出現在按鈕的狀態資訊，顯示這台戰車現在是否再移動或攻擊，以及一個下拉式選單選取角色。

至於 Joy Sticker 的設計，我們用了 Joy Sticker 的 class 去實作，當偵測到 Joy Sticker 不同的 State 時，依此傳不同的訊息到藍芽器上。

Joy Sticker State	解釋	執行的動作
STICK_UP	代表搖桿向上	傳出讓戰車向上的動作
STICK_RIGHT	代表搖桿向右	傳出讓戰車向右的動作
STICK_DOWN	代表搖桿向下	傳出讓戰車向下的動作
STICK_LEFT	代表搖桿向左	傳出讓戰車向左的動作
STICK_NONE	代表搖桿在中間	不傳訊息(傳 0)
ACTION_UP	手不在搖桿上	不傳訊息(傳 0)

問題與討論：

(1) 我們發現如果 Joy Sticker 如果在 STICK_NONE 和 ACTION_UP 不傳訊息的話，會造成 FPGA 的值停留在上一次的值，即使玩家沒有按任何鍵，還是會造成戰車持續坐上一次的動作，所以還是必須要在玩家放開按鈕時傳回 0 的值，這樣即使 FPGA 停留在上一次的值，還是可以讓戰車不做任何動作。

(2) 因為手機傳到藍芽的值還包含是哪個玩家傳的值，不過若沒有在下拉式選單選玩家的話，可能會造成錯誤，所以我們讓 Android App 在使用者沒有選玩家的情況不傳任何的值。

2. Android 端和藍芽器的資料傳輸

當介面完成後，我們需要將這些指令變成 data 傳送到藍芽器上面，所以我們將每一個指令用 1 個 bit 去表示。

Bits	功能
bit[0]	00:玩家 1; 01:玩家 2; 10:玩家 3; 11:玩家 4
bit[1]	
bit[2]	1:向上 0:無
bit[3]	1:向下 0:無
bit[4]	1:向左 0:無
bit[5]	1:向右 0:無
bit[6]	1:射擊 0:無
bit[7]	沒有功能

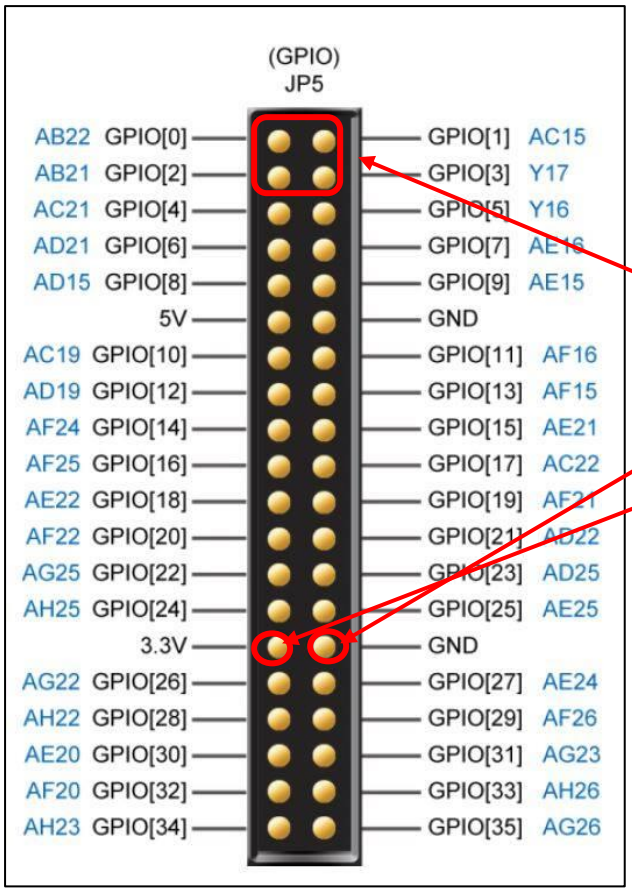
所以這樣總共是 8 bit=1 byte，正好等於藍芽在傳輸的單位資料，因為藍芽傳輸時是以 1 個 byte 的單位去傳資料。

所以接下來如何將手機上的資料傳到藍芽器，就需要 Bluetooth Stream 的 interface 與 class。首先需要建立 BlueToothAdapter，這個接口是用來搜尋附近的藍芽裝置，建立 BlueToothServiceSocket 來做資料的傳輸。若這個接口需要搜尋為配對的裝置的話，則會進而建立 BroadcastReceive，進行配對與連接。當找到目標的裝置連接後，就可以利用 BlueToothServiceSocke.connect()建立 Socket 的連線。一旦連線，傳送資料都只要對這的 Socket 操作相關的指令的可以了，除非斷線就必須在重複前面的動作。接下來就可以利用所建立的 Socket，使用 OutputStream 的 write()將每一個 byte 傳到藍芽器上面。

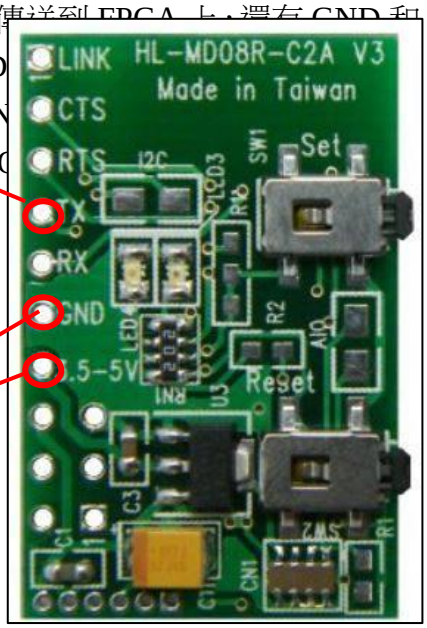
問題與討論：

當我們要傳一個 byte 出去時，我們原本是用 int 型態後去傳輸，將 int 後面的 8 位拿還使用，其他為 0，但是手機端的 outputStream 在傳資料的時候卻會在原本沒有資料的前 24 位加上奇怪的資料。為了解決這個問題，我們強制用 byte 的型態傳，避免多出一些奇怪的資料。

3. 藍芽模組和 FPGA 的連接以及 UART decode

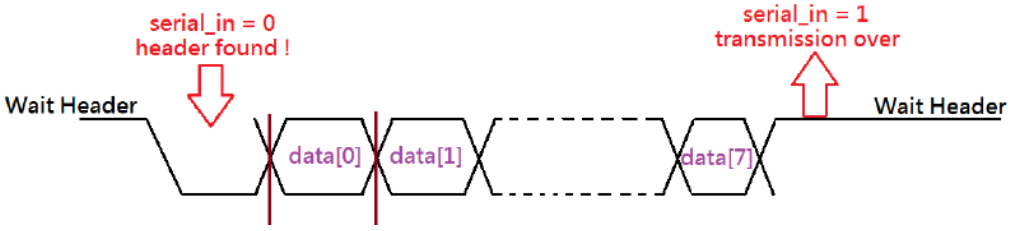


藍芽器上總共有 4 個 PIN，我們需要其中的 TX 將我們的資料傳送到 FPGA 上，還有 GND 和 VD PIN FPG



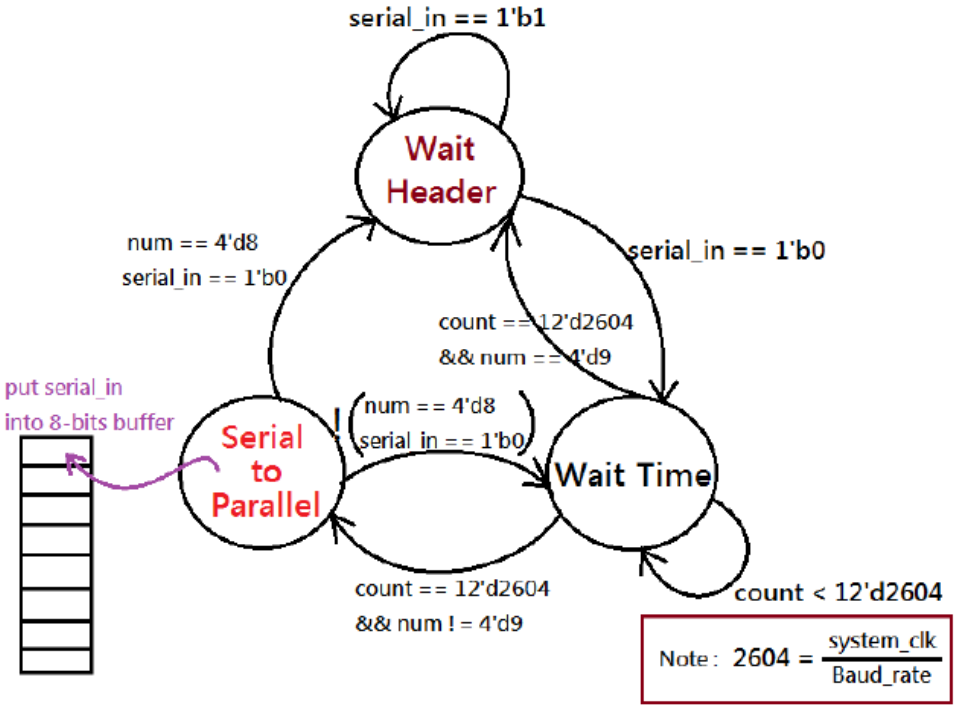
由於我們的遊戲有設計 2~4 人對戰，所以會有四個藍芽器的 TX 連接到不同的 GPIO 的 input。這時候我們就需要將 GPIO 的值做 decode 的動作。

一般藍芽器的資料都是以 serial 的形式，如下圖：



藍芽器會在一定的時間區隔(依照 baud rate)改變 output 的值，所以當有一筆資料(1 byte)來的時候，就會先傳 1’b0 告訴接收者有資料要來了，所以接下來的 8 個 bit 都是藍芽器的這一筆資料，共需要接收 8 次，所以為了應付藍芽器這樣的傳送方式，我們就用了 UART 的 decode 方法接收，進而轉成 parallel 的資料。

其實 UART decode 就是用 FSM 的方法，如以下的示意圖。



一般狀態都會停留在 Wait header 等待藍芽器是否有傳資料，當出現 1'b0 時，就會跳到 Wait time 就會開始資料的傳輸，依據 Baud rate 的間隔來回 S2P 和 Wait Time 接收 8 次 bit 的資料，最後當接收完 8 次資料後，就會馬上更新 parallel 的值，將這個值傳到 nios 內去做 decode。然後跳回 Wait Header 繼續等待。

接下來，就可以利用 4 個 UART module 去 decode 四個 GPIO input，接收這四個藍芽器的資料。然後將資料利用 SWITCH，讓 nios 去接收。

In Quartus:

```
always @*
  case ({avs_s1_read_n,avs_s1_address})
    5'b00000: avs_s1_readdata_next = {31'd0,coe_s2_ibluetooth_data[7:0]};
    5'b00001: avs_s1_readdata_next = {31'd0,coe_s2_ibluetooth_data2[7:0]};
    5'b00010: avs_s1_readdata_next = {31'd0,coe_s2_ibluetooth_data3[7:0]};
    5'b00011: avs_s1_readdata_next = {31'd0,coe_s2_ibluetooth_data4[7:0]};

    default: avs_s1_readdata_next = avs_s1_readdata;
  endcase
```

In nios:

```
void readData(int data[]){
    data[0] = IORD_32DIRECT(BLUE_TOOTH_DATA_BASE,0);
    data[1] = IORD_32DIRECT(BLUE_TOOTH_DATA_BASE,4);
    data[2] = IORD_32DIRECT(BLUE_TOOTH_DATA_BASE,8);
    data[3] = IORD_32DIRECT(BLUE_TOOTH_DATA_BASE,12);
}
```

這時候 nios 只要利用下面的 IORD 就可以讀到 FPGA decode 完的資料，只要讀取不同的 offset 就可以拿到四的藍芽器的資料。

問題與討論：

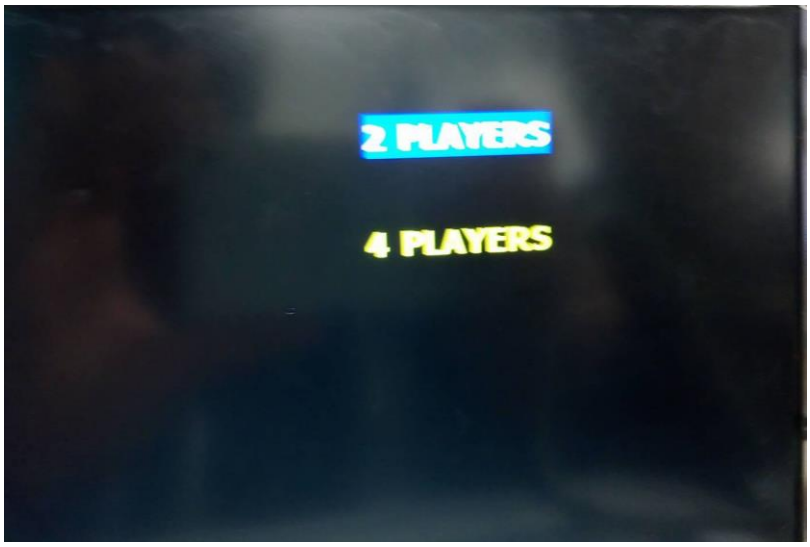
(1) 這裡我們遇到最大的問題就是藍芽器 Baud rate 的問題，因為藍芽器的 Baud rate 必須要自己用電腦上的程式手動更改，而預設值是 115000。通常這樣的 Baud rate 因為速度太快，非常容易掉資料，所以我們改成 19200 的速度，這樣速度不慢但資料又幾乎接收的到。然後 UART 的讀取區間必須要依照 Baud rate 去控制。

(2) nios IORD 的讀值是 32bit 資料，所以如果要將資料從 Quartus 上傳到 nios 就必須補 24'b0 在資料的前面。

(3)在 UART 內，我們考慮可能會發生連續兩筆資料，也就是 header 在下一筆資料的尾巴的狀況，所以我們在尾巴是 0 時，在接收一次資料，但我們覺得這樣的邏輯還是很奇怪，所以目前沒有解決，但目前看不出影響遊戲資料的傳值。

NIOS 端遊戲介紹

一、 開始畫面



有兩個按鈕，用手機按上下調整，並且按下 ATTACK 即可選擇所要的模式，並且開始遊戲。

二、 遊戲主畫面

這個遊戲主要是由以下 4 種 struct 所構成，遊戲開始時先將開出來的 struct 初始化。

(1)Tank：玩家所操控的戰車

- height → 坦克高度
- width →坦克寬度
- px→ 坦克 x 座標
- py→ 坦克 y 座標
- direction→ 坦克方向 (1:上，2:下，3:左，4:右)
- speed→ 坦克速度
- shoot_speed→ 坦克射出子彈的速度
- shoot_power → 坦克射出子彈的傷害力
- bullet_num → 坦克擁有的子彈數量
- blood → 血量
- state→ 狀態 (0:死亡，2:正常，3:被子彈打到)

(2)Obstacle：畫面上的障礙物

- px→ 障礙物 x 座標
- py→ 障礙物 y 座標
- height→ 障礙物高度
- width→ 障礙物寬度

(3)Bullet：子彈

- p_x→ 子彈 x 座標
- p_y→ 子彈 y 座標
- direction→ 子彈方向
- speed→ 子彈速度
- power→ 子彈傷害力
- isExist→ 子彈是否被射出

(4)Prop：道具

- px→ 道具 x 座標
- py→ 道具 y 座標
- isExist→ 道具是否出現
- effect→ 道具速度

1. 讀取資料

因為有 4 筆資料，因此我開一個陣列，每一次迴圈都會讀取 4 筆資料，並且將它做 Decoding，每一筆資料都有 8 個 bits，每個 bit 都有不同的功能，如下表所示。

Bits	功能
bit[0]	代表第幾個玩家，最多有 4 個玩家所以需要 2 個 bits 來控制。
bit[1]	00:玩家 1; 01:玩家 2; 10:玩家 3; 11:玩家 4
bit[2]	控制戰車往上移動
bit[3]	控制戰車往下移動
bit[4]	控制戰車往左移動
bit[5]	控制戰車往右移動
bit[6]	控制戰車射擊
bit[7]	沒有功能

每筆資料會根據前兩個 bits 決定要控制第幾台戰車，若第一筆資料前 2 個 bits 是 01，則第一筆資料的移動和射擊都是控制第二台戰車，以此類推，用這種方法的優點是可以在同一台手機自由的選擇要玩第幾個玩家，不會被綁住。

2. 接觸的判定

因為這個遊戲的各個物件(除了戰車)都是方形的，因此寫了一個兩方形是否相觸的函數。

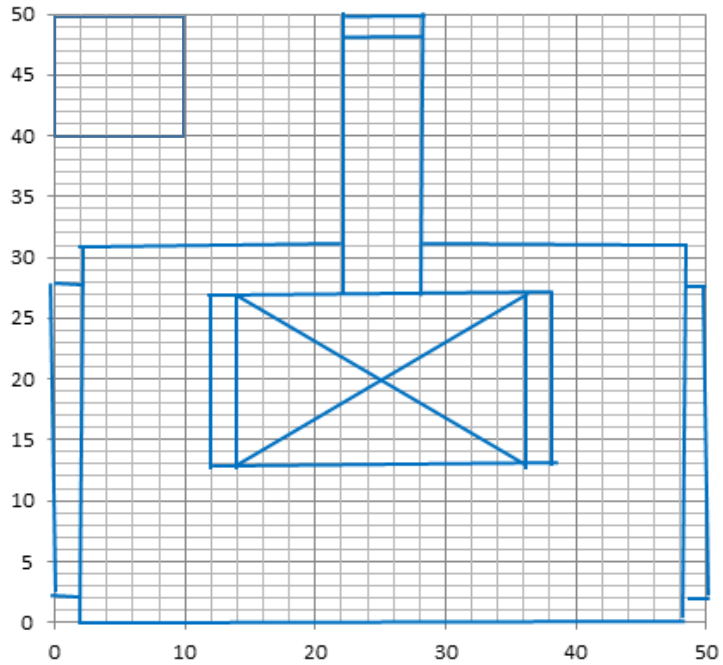
```
int isBoxTouchBox(int b1_left, int b1_right, int b1_up, int b1_down,
        int b2_left, int b2_right, int b2_up, int b2_down){
    if(b1_left<=b2_right && b1_right>=b2_left && b1_up<=b2_down &&
            b1_down>=b2_up) return 1;
    else return 0;
}
```

判斷方式為傳入兩物件的最高和最低的 x,y 座標，並判斷各邊的位置後即可得知兩物件是否接觸。

而在此遊戲中主要會發生以下四種狀況

(1) 戰車被子彈打到

以下是戰車和子彈的圖，戰車佔了 50 X 50 的 pixels，而子彈佔了 10 X 10 的 pixels。



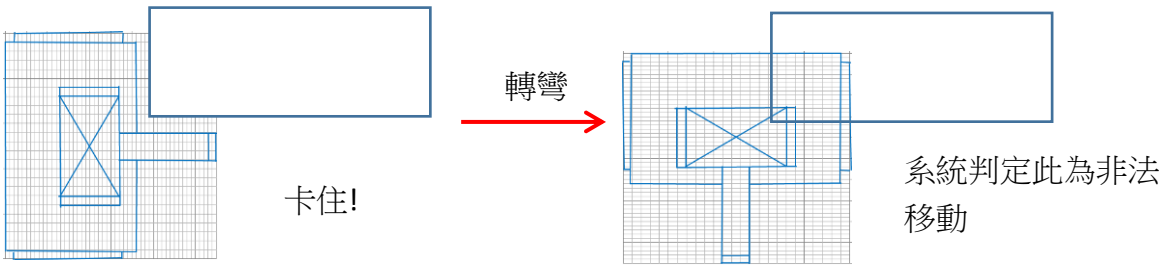
在這部分比較複雜一點，因為戰車有一根凸出去的砲管，所以我要根據戰車上下左右四個不同的方向，做不同的判定。

(2) 戰車吃到道具

作法與戰車被子彈打到一樣。

(3) 戰車撞到障礙物

這個我原本也是用跟被子彈打到一樣的做法，可是我發現這樣如果戰車撞到障礙物的邊角會發現戰車卡住不能移動的現象。



因為在此狀況下不能再前進只能改變方向，但是轉彎後戰車的屁股一定會卡到障礙物會直接判定無法進行此移動而卡住。

解決方法：

因此我把戰車當成一個方形物體作判定，這樣判定較為簡單，畫面不會有不協調的狀況，也不會發生以上的狀況了。

(4) 子彈撞到障礙物

因為子彈和障礙物都是方形的，直接判斷兩方型是否接觸的函數即可。

3. 戰車的移動

根據讀取到的資料做戰車座標的改變即可達到戰車移動的效果，重點是邊界和障礙物的判定。

4. 發射子彈

讀取到射擊的資料即可做射擊，每台戰車有發射量上限，會做邊界、障礙物和是否射擊到別台坦克的判定。

5. 道具的生成和效果

道具是每過一定週期就會在隨機位置產生出不同效果的道具，而且不會在有障礙物的地方生成，當戰車碰到道具後即會根據此道具的效過改善戰車的某樣性質，增加打敗別的玩家的機率。

遊戲中會出現 5 種道具：行走速度增加、子彈射速增加、子彈傷害力增加、子彈數量增加、補血。

6. 畫面的生成

主要使用以下 2 種函數畫圖：

(1) 畫直線

```
void vid_draw_sloped_line( unsigned short horiz_start,
    unsigned short vert_start, unsigned short horiz_end,
    unsigned short vert_end, unsigned short width,
    int color, alt_video_display* display);
```

(2) 畫方形

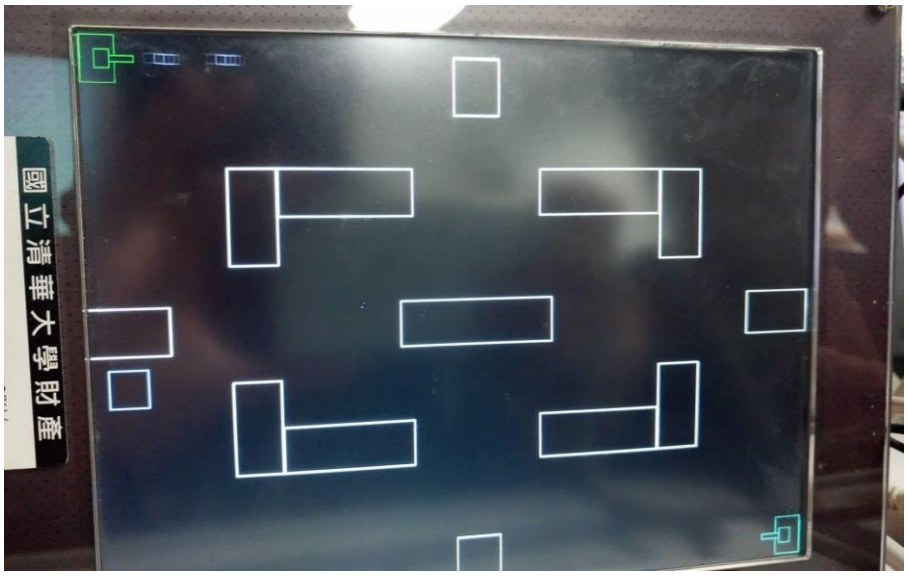
```
int vid_draw_box (int horiz_start, int vert_start, int horiz_end,
    int vert_end, int color, int fill, alt_video_display* display);
```

戰車等都是一條線一條線畫上去的，因此在這邊花了不少時間，此外在畫面的更新上有遇到一些問題，因為在遊戲中畫面隨時都會一直改變，所以要一直刷新畫面，一開始我所用的方法是每一次迴圈都會將整個畫面塗黑，再把物件重新畫上去，後來發現畫面會閃爍嚴重。

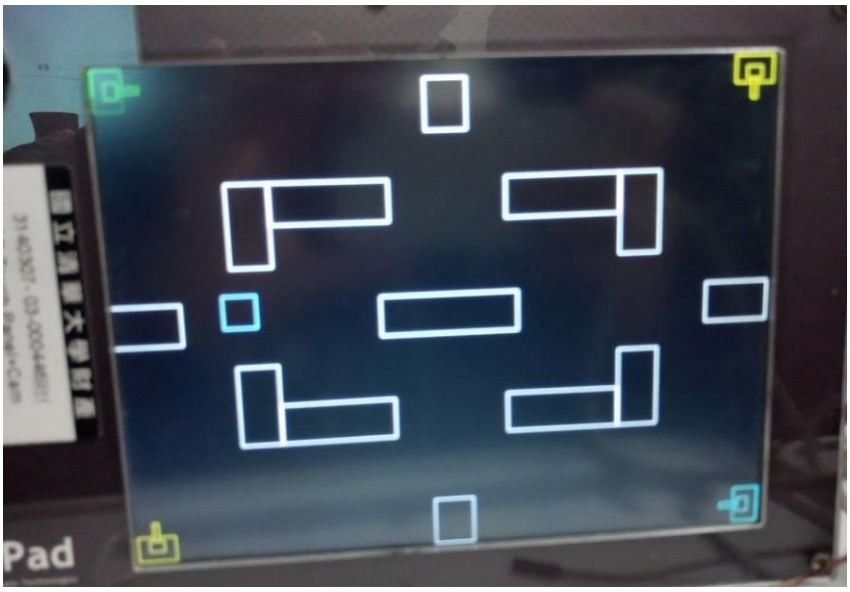
解決方法：

遊戲中會移動的就只有子彈和戰車，障礙物和道具其實不會移動，因此我另外開了 struct 記住戰車和子彈前一次的位置，並將前一次的位置塗黑在畫上新的，這樣畫面就流暢許多。

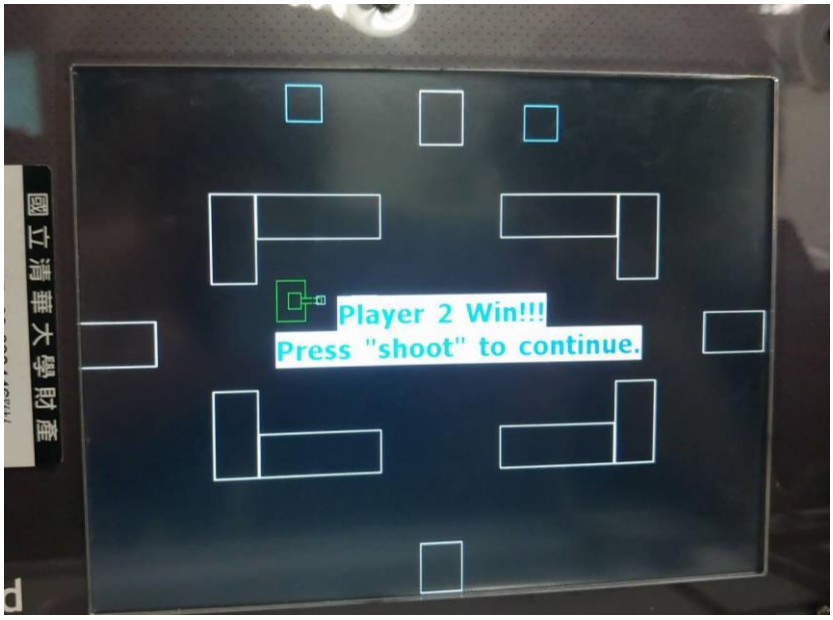
以下為一些遊戲畫面



圖一、雙人對戰模式



圖二、四人對戰模式



圖三、遊戲結束畫面

心得與結論

這次的 Final Project 可以說不但比之前的 lab 辛苦，但也比之前的 lab 有更大的收穫。說真的在期末前 5 周繳交 propasal 的時候，我們都還在猶豫自己是否真的能做出 Proposal 上提到的功能，尤其是藍芽器的運用，藍芽器傳接收值的架構真的是我們在做 Final 的一大挑戰，畢竟藍芽器算是我們資料聯繫非常重要也是非常需要花時間去了解的部分，最後能夠成功查到相關資料以及做出成功的 decoder 已經是一大慶幸之事。

除此之外，學會寫 Android App 也是一大挑戰，雖然我們之前在大二時有學過 Android App 的撰寫，不過日子久遠讓我們還是要回去複習 App coding 的技巧以及 UI 的設計，最後也慶幸完成我們的遊戲的手機控制器。因為這次的 Final，我們才得以了解更多軟硬體上的知識，以及更多相關的資料傳輸與概念。

遊戲設計部分也充滿挑戰性，雖然也遇到不少 bug，但通常都是一些小地方沒注意到，而且讓我們意識到遊戲程式撰寫一開始的架構很重要，我們這次就把很多值寫成一個變數，這樣完成後直接調整參數就可以輕易的改善遊戲運作，另外這次很多運算都是在軟體中完成，以後可以嘗試在硬體中完成，這樣效能應該可以更高，遊戲畫面會更加流暢。