數電實驗 期末報告

team06

B05901016 王珽
B05901064 林承德
B05901185 戴慕潔

目錄

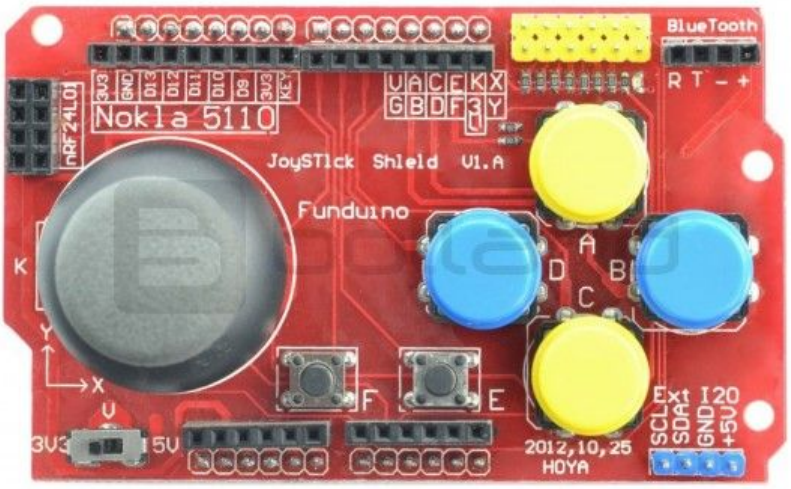
一、簡介、動機

我們這次的數電Final Project主題是坦克大戰，遊戲內容主要是由兩位玩家以操縱坦克來進行對戰的傳統射擊遊戲，會挑選這個題目的動機起先是因為我小時候很喜歡打Counter Strike這款設計遊戲，因此才和組員們討論出了這個主題的雛形。由於市售的Counter Strike構圖相當複雜，在有限的時間內恐怕無法達到目標，因此我們改以坦克對戰的形式來進行遊戲，不過雖然構圖未若市面上的遊戲複雜，但射擊遊戲的元素（子彈、移位）依舊有納入我們的設計考量，而這也達成了一開始我們所期望達成的目標。

本遊戲的操作方式為利用外接的按鈕，來讓玩家進行射擊以及移動的動作，此外，本遊戲的地圖也以沙漠底色的路線以及綠色的牆壁構成，子彈除了能射中對手坦克造成傷害外，也能對地圖上的牆壁進行破壞，此配置也讓遊戲的變化更加多元。

二、使用者教學

- 器材



Arduino joystick

VGA to VGA cable

螢幕

- 使用步驟
  - FPGA連接螢幕和Arduino把手
    FPGA的GPIO接口：

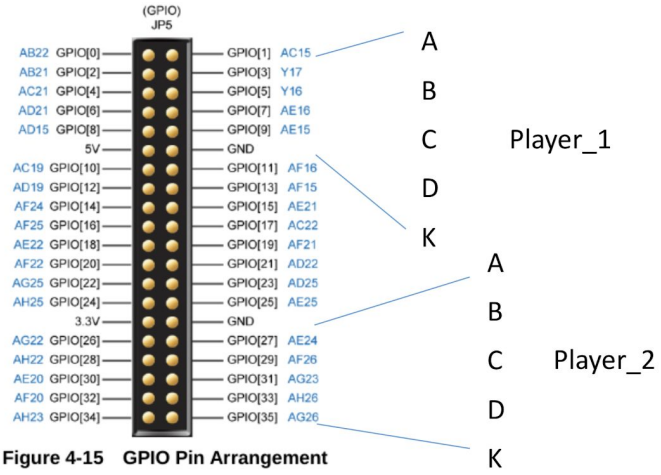
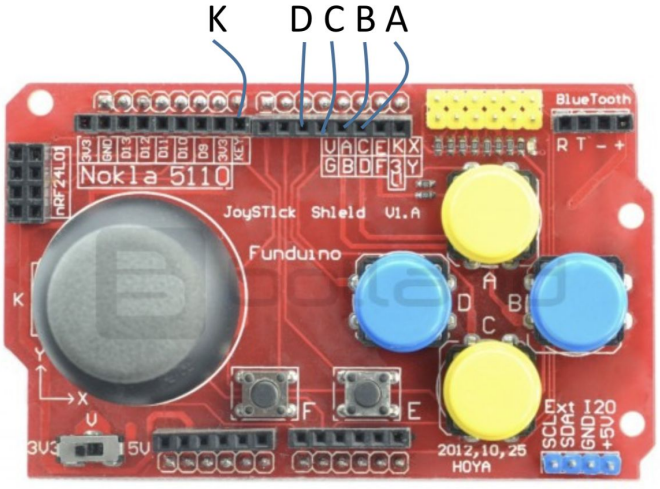


Figure 4-15 GPIO Pin Arrangement

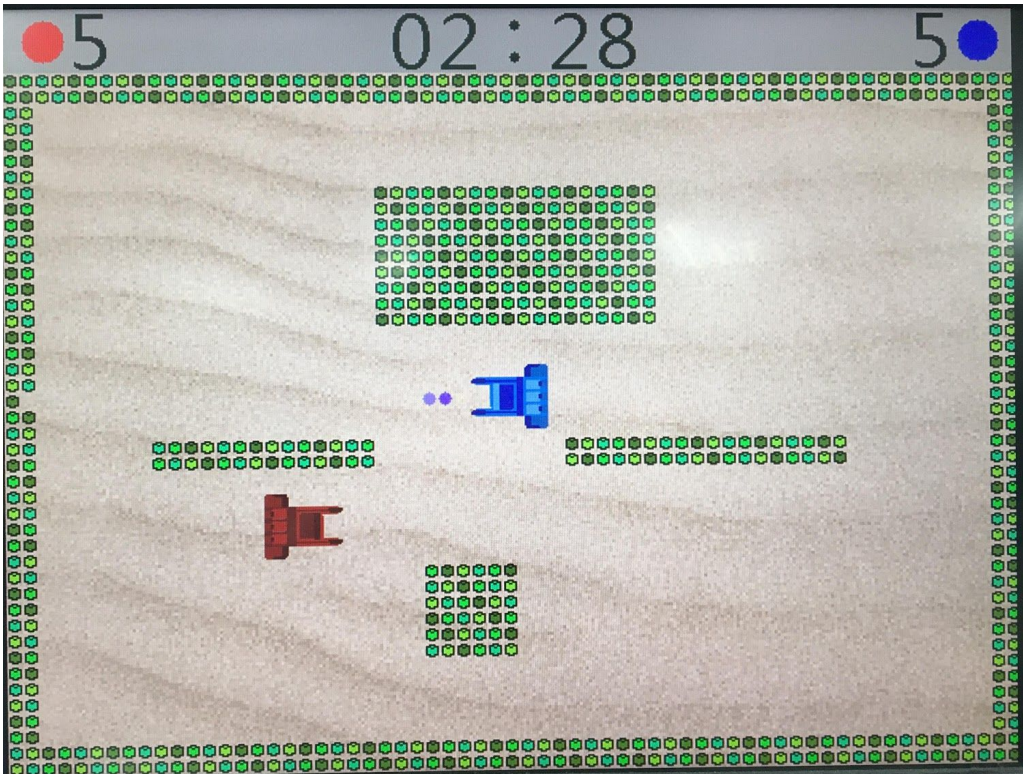
Table 4-10 Power Supply of the Expansion Header

Supplied Voltage	*Max. Current Limit*
5V	1A
3.3V	1.5A



  - 按KEY0來reset
  - 螢幕上即出現開始畫面

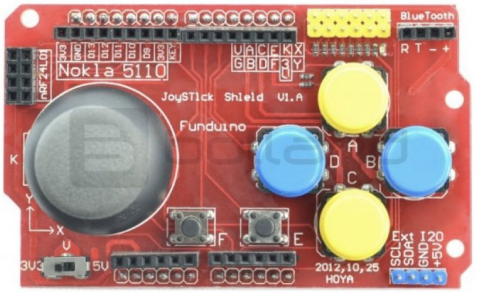
- 兩個玩家同時按下把手的**左鍵**，遊戲就開始了
  - 最上面那條兩邊的數字代表紅隊跟藍隊各剩下幾條命，被子彈打中就會少一條命
  - 最上面那條中間的數字代表遊戲開始時間過了多久
  - 每一次的地圖是隨機出現的
  - 為了避免有坦克一直躲在牆下面，地圖上每塊牆只要被子彈打三次就會消失，躲在牆後的坦克就有可能被擊中
  - 只要有坦克被打中五次，遊戲結束，另一個坦克就獲勝



- 遊戲結束畫面
  - 如果是紅隊贏，牌子就會呈現紅色，反之則呈現藍色
  - 兩玩家在同時按下**左鍵**，遊戲會再次開始

三、遊戲架構Game structure

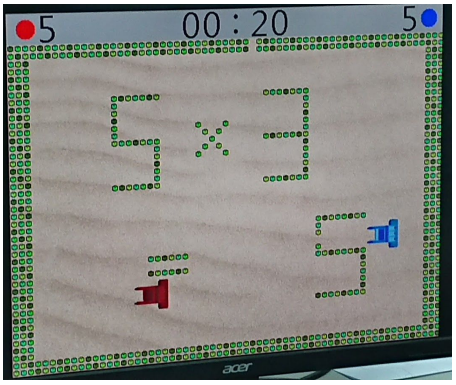
- **Joystick**



我們使用Arduino Joystick Shield當作我們的控制搖桿，右邊四顆彩色按鈕為上下左右，左邊的搖桿按下去為發射子彈。每一隻把手的5顆按鈕訊號線和VDD、GND共7條線接到FPGA上的GPIO，就可以當作正常FPGA上的按鈕使用。

按鈕的訊號線進FPGA後會接到Debounce module。上下左右鍵會使用o_debounce 輸出，讓玩家按著不放就可以持續移動；發射鍵則是用o_neg輸出，因為我們發現每次按發射只能射一發砲彈會讓遊戲更加刺激，也增加遊戲所需的技巧。

- **Display**



我們使用的VGA通訊協定是640x480的解析度，我們以每10個pixels為一單位，劃分成64x48格作為遊戲的基本單位。其中上面四行會作為狀態列，顯示剩餘生命數及經過的時間；其他64x44格則做為遊戲用的空間。

我們的坦克大小為5x5格，砲彈及牆壁皆是1x1格。任何東西進入對方的格子內都會被判斷為碰撞，執行相對應的指令。

在遊戲畫圖的過程中，每個module都會對現在這個位置給出RGB的顏色，如果該位置沒有東西則會輸出#000000。由於我們遊戲不允許任何物件的碰撞，故把所有物件的輸出OR起來，如果不是#000000則輸出OR後的結果，不然就輸出背景顏色。但牆壁有用到黑色，會被誤判成沒東西，故牆壁的黑色其實是#010101，肉眼完全無法分辨，且程式也會知道該格是有物品。

- **Image**

我們在遊戲中主要有用到開始畫面、遊戲背景及勝利畫面三張全螢幕畫面及其他坦克、數字等小圖片。為了避免全螢幕圖片把FPGA空間占滿，我們把圖片的解析度從VGA的640x320降到320x240。但這依舊不夠，因此我們要想辦法降低每個pixel使用的bit數量。

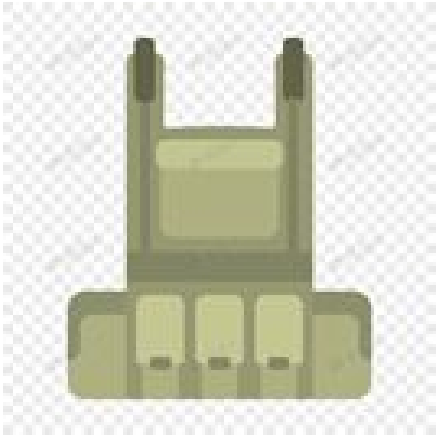
我們使用k-means Clustering演算法來降低bit數量，其原本主要用於非監督式機器學習。作法是把n筆觀察資料劃分成k組，並在每一組中取平均當作整組代表的數值，再藉由不斷迭代找出更好的分組及代表。我們在這次實作中的k取8或是16，這樣每個pixel從原本的RGB各8bits共24bits壓縮到剩下3bits或4bits，減少了約7/8的空間使用量。此演算法的好處是每個代表會依據不同圖片而改變，可以更精準的代表整張圖片。下方左圖是網路上找到的原圖，右圖也是經由k=16的k-means clustering後的結果，可以看出還有維持一定程度的畫面觀感。

在實作方面，我們用python生出每張圖片的labels.dat和values.dat，labels代表圖片中每個pixel的組別編號，values則代表每個組別的代表顏色。將labels及values讀入verilog後就可以使用，而且改變圖片也不需要大費周章的改code中的顏色。

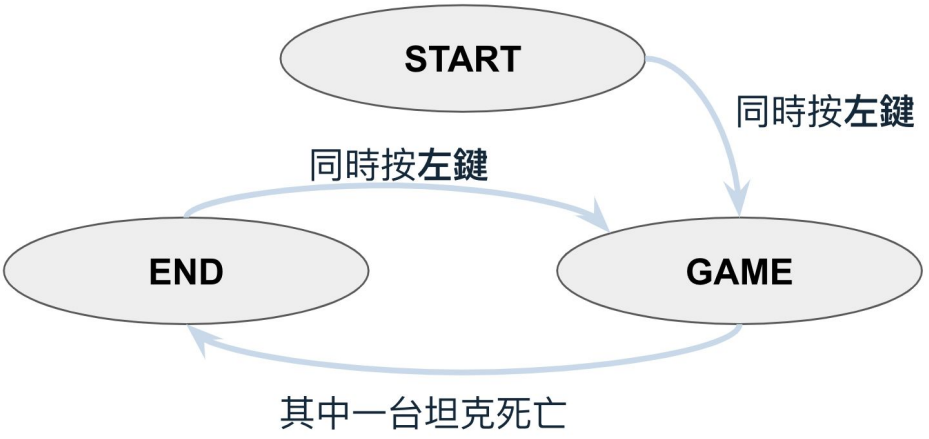
我們只有把紅色的勝利畫面燒進去FPGA，當今天藍色坦克勝利時，FPGA會依據紅色的勝利畫面改變成藍色的勝利畫面。如果values的R ≥ 16 且 G, B < 16時，繪圖時R與B會交換，達到把紅色改成藍色的效果。這樣可以減少多一張照片的空間，減少compile所花的時間。

狀態列上的數字的大小是30x40 pixels，我用photoshop開了一樣大小的1-bit點陣圖，並用18pts的文字大小寫下數字後存成bmp檔，再用python轉換成verilog可接受的格式。砲彈和計時器冒號的圓形則是直接用python和基本的幾何學所製作出來的data。

坦克的圖片我們也是上網搜尋後使用photoshop去除上面的浮水印、切成正方形後，分別上色成紅色及藍色，再用python 換成50x50 pixels大小後進行k-means轉換。下圖左邊是網路的原圖，右邊兩張則是經過處理後的最後結果。



- **Game State**



這個module的架構分成三個state，START是指遊戲開始介面，當兩個玩家同時按左鍵就會跳到GAME這個satate，當其中一台坦克死亡，遊戲結束，跳到END state，要再開始一次兩玩家必須在同時按下左鍵，會再度回到GAME state。

在GAME state時，控制了一些碰撞問題

- 子彈打到牆壁 → 子彈消失，告訴Shell這個module，告訴牆壁被打了
- 坦克撞到牆壁 → 不讓坦克繼續向前，告訴Tank這個module
- 子彈打到坦克 → 坦克少一條命
- 坦克撞到坦克 → 不讓坦克繼續向前，告訴Tank這個module

- **Tank & Shell**

Tank（坦克）：

在本遊戲中，Tank需要控制的就是方向、位置、以及速度，因此以此三種要素來進行解釋。

1. 方向：方向是由使用者經由搖桿按鈕輸入，經由state module傳遞進入Tank的module，總共有UP、DOWN、LEFT、RIGHT四種方向，並且以00、01、10、10來代表，此外，若使用者並未按按鈕給Tank方向，那麼state module會輸入給Tank一個Stand的訊號，此外，Tank也會把方向傳給VGA module，來讓坦克能在螢幕上顯示正確的方向。

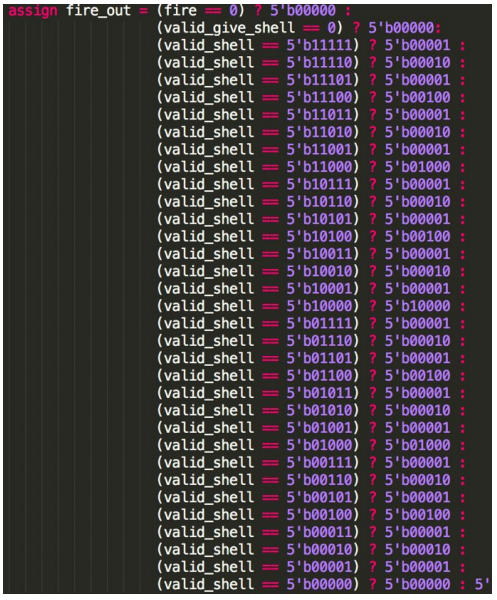
```
input [2:0] direction_in, //0_UP, 1_DOWN, 2_LEFT, 3_RIGHT, 4_STAND
```

2. 位置：利用簡單的x、y座標來記錄Tank的位置並回傳給state module和VGA module，其中x向右為正，y相下為正，而原點則設於螢幕的左上角，來讓所有x、y座標位置都是正整數。

3. **速度**：Tank的速度需要考慮到Tank的方向以及使用者按壓方向鍵的頻率，倘若使用者持續按壓某個方向鍵不動，那麼Tank的移動速度就是每四個frame會前進一格，其中，frame的接收方式是從state module接收valid訊號，若valid從0變為1時，代表一個frame的切換，若坦克維持同一方向滿四個frame的時間，坦克才會朝該方向移動一格

以上為Tank module的介紹，由於本遊戲需要兩個玩家來進行對戰，因此會以此module接上兩台不同的Tank，並分別計算其位置、方向、以及速度，來讓遊戲順利進行。

Shell（砲彈）:



Shell和Tank相似，都需要紀錄位置、方向、以及速度，方向的部分，在發射當下Tank的方向一致，並且以每400000 clock cycle就往前進一格，而位置也和Tank一樣紀錄x、y座標，不過和Tank不同的是，Shell在碰到牆壁會消失，而每個玩家也有五發子彈的配額可以發射，因此需要紀錄五個Shell在不同情況下的發射情形，而在實際的設計上，我們先以single_shell的module來記錄單一個shell的位置、方向的狀況，再以select_shell的module 來判斷在玩家按下fire進行發射時，該選擇五顆子彈中的哪一顆來進行發射，選擇的方法就是以各個Shell未發射/發射的情況來進行判定，由於每顆Shell都有未發射、已發射兩種狀況，故五顆子彈共有2^5=32種狀態，而我們則就這32種狀態來進行子彈的選擇，來確保子彈不會重複被射出。

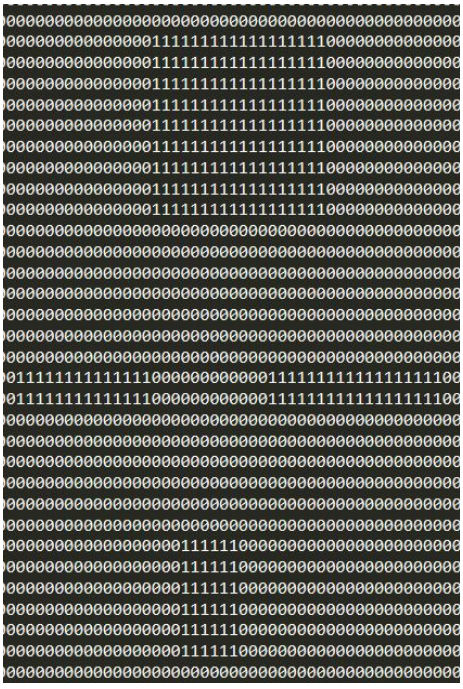
在進行完single_shell以及select_shell的設計後，我們最後再以shell這個module來將這兩個sub module來pipeline起來，來記錄每個玩家五顆子彈，共兩個玩家十顆子彈分別的狀態，來讓玩家的射擊能夠順利進行。

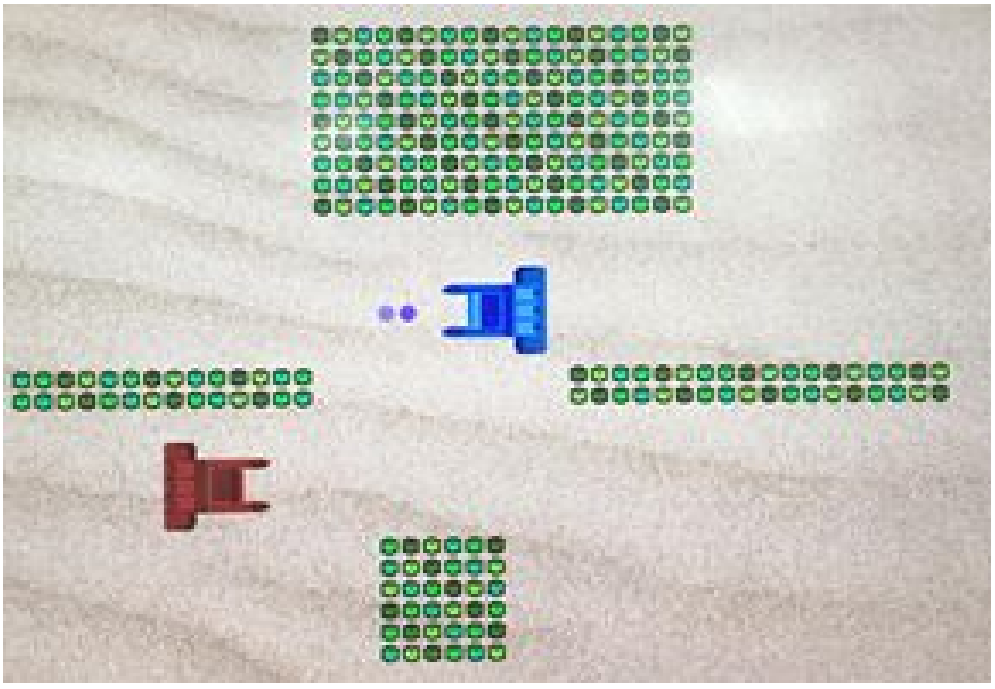
- **RandomMap**

扣掉上方的狀態列之後，我們的遊戲空間有64x44格需要設計。我們將設計的地圖打在map.dat，其檔案內有44行、每行有64個字元、0代表空地、1代表有牆壁。讀進code裡面之後RandomMap就會知道地圖應該要長怎樣，再把地圖的資訊傳給GameState。

牆壁的生命數值也是記錄在這個module。一但今天有砲彈打到牆壁，GameState會傳訊息過來，再把相對應的生命值減一。RandomMap在輸出該格是否有牆時，會把原有的地圖資訊再去 AND 該牆壁的生命值，故如果牆壁的生命值歸零，則RandomMap會跟GameState說該位置並沒有牆、也就不會觸發碰撞，畫圖時該格也不會出現牆壁。

我們總共設計了八張地圖，透過亂數產生器隨機選出一張地圖。亂數產生器是直接使用我們lab1的方法，也就是偽隨機性的算式($X_{n+1} = X_n * 16807 \% 2147483647$)。由於這種方式產生出來的隨機數字的順序永遠都一樣，我們讓隨機數字在每一個clock都更新，但只有在有坦克死掉瞬間的數值才會被採用。每場遊戲死亡時間都不盡相同，故取數字的時間點也會不一樣，獲得的數字就不會一樣，可以防止玩家猜出下一張地圖是什麼。





四、心得

本次的遊戲製作涉及了遊戲狀態、坦克及子彈位置的計算、以及如何將畫面顯示於畫面上，而我們的工作也照著這三大部分來分配，由於三者的input及output需要互相配合，因此三個人也花了許多時間來溝通、討論，而從這我們也了解到在進行電路的設計之前，必須要先溝通好input及output的protocol，才不會造成個別module寫完了之後，還要花很多時間協調才能把線接起來的狀況。