DCLab - Final Project Duck Hunt

Team04 韓秉勳 蔡昕宇 解正平

User Manual

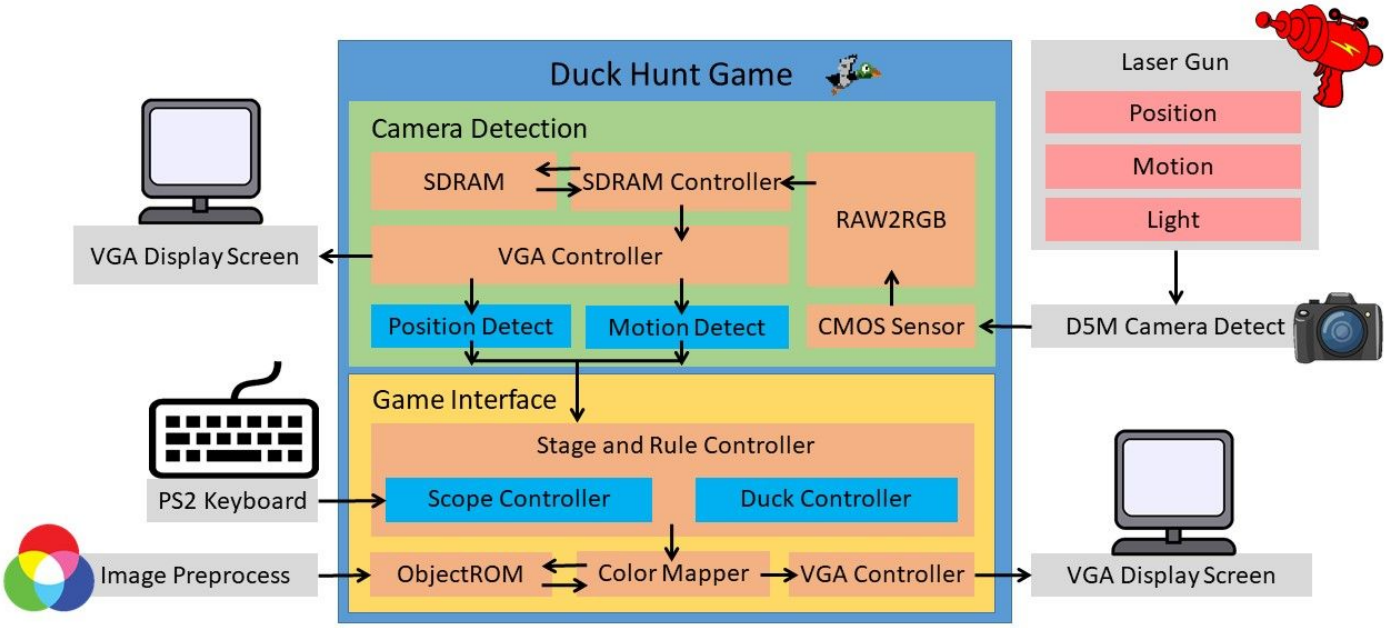
- Game rule:
  - 這是一款懷舊射擊小遊戲，玩家需利用雷射槍瞄準野鴨，在有限生命中射下越多野鴨即獲勝!
  - 畫面左下角愛心為生命值，每次遊戲玩家都有五條命，只要讓野鴨逃離畫面，命即會減損一條，當生命值歸零即結束遊戲。
  - 畫面右下角數字為射中野鴨數，可看到當下射中多少野鴨。
  - 至遊戲後期會有多隻野鴨集體飛出，須將所有野鴨都射下才能保住生命值，只有極致準度的玩家才能勝過野鴨的集體挑釁！
  - 遊戲中也統計總耗費子彈數，可看出誰才是真正的神射手！

- Game flow
  - 先將相機的曝光度調低，打開SW[1]，並且按KEY[1]，直到LEDG[0]不亮。
  - 開啟雷射槍(發光)，將準心指向start按鍵，使雷射光熄滅即開始遊戲。
  - 畫面會有隨機的野鴨由下而上飛出，玩家須將準心對準野鴨併發射(即將雷射光熄滅)，野鴨若被射中會向下掉落。
  - 野鴨出現速度會隨射中野鴨數遞增，至後期也會有多隻野鴨同時出現。
  - 遊戲中可以隨時對畫面下方Home鍵發射，遊戲會回到初始畫面並重新開始計分。
  - 當生命值歸零，畫面會跳出gameover，右下角即為此次得分。此時朝home鍵發射即回到初始畫面。

Teaching Manual

I. Introduction：以任天堂(NES)的Duck Hunt為模型設計遊戲，期望達到射擊效果。

II. Module Architecture



III. Experiment Process

A. (分工)我們將整個project分為兩個部分，一個是用相機偵測槍的準心來模擬射擊，一個是設計遊戲介面，包含畫面以及規則實作。

B. (相機)可以偵測雷射槍光源，並且不會受到其他光源干擾。

C. (相機)抓取光源每幀畫面在VGA640x480位置，以光源中心x, y當作座標。

D. (相機)偵測光源絕對位置，或是偵測光源移動相對關係(方向和距離)，用來判斷遊戲的準心需要如何移動。

E. (遊戲)將現成圖片的每個pixel的顏色存成txt檔案，以利之後呈現畫面。

F. (遊戲)根據不同遊戲過程，呈現不同畫面(初始，遊戲中，結束等)。

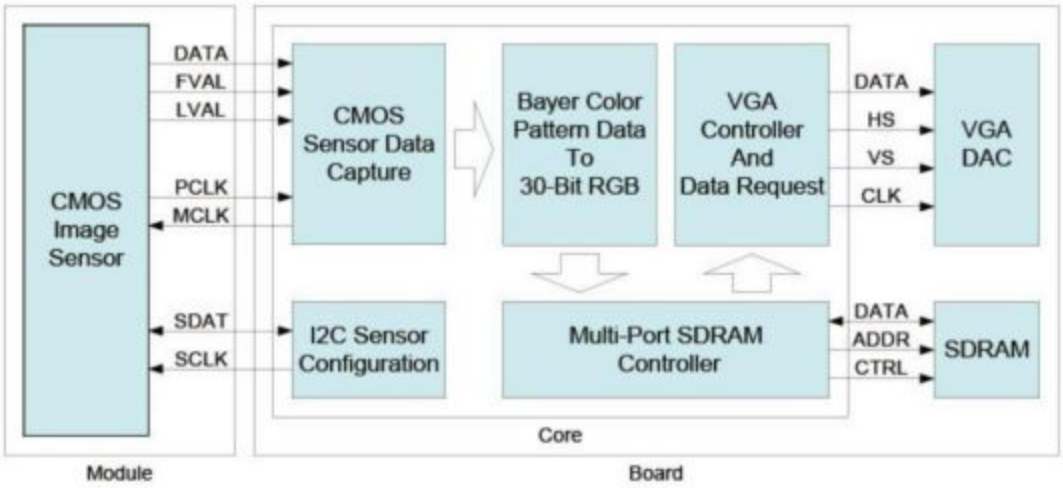
G. (遊戲)設計鴨子移動方式，以不同邊界限制和速度差異增加變化。

H. (遊戲)用鍵盤控制準心移動，完成初步鍵盤控制射擊遊戲。

I. (合併)把所有需要鍵盤控制的介面都由控制雷射槍來完成。

IV. Camera Interface

A. Camera Control



以上是Camera使用的模組，NAS便有提供相關資料可以參考。

我們使用D5M相機擷取影像，並且存入SDRAM做緩存。再從SDRAM讀出資料，以VGA控制螢幕顯示，達到即時顯示相機拍攝畫面的功能。最後再根據顯示畫面來做後續的處理。

我們將取得的畫面從RGB三色圖轉成灰階的YCbCr以此來判斷雷射槍光源，另外為了避免出現乘上小數點，我們先將係數都乘上255，最後數值再除255，取得Y值當作判斷光源依據，判斷白光的threshold設為255。

$$Y = 0.299 \times R + 0.585 \times G + 0.114 \times B$$

$$Y = (77 \times R + 149 \times G + 29 \times B) \div 255$$

$$if\ Y \geq 255\ this\ pixel\ is\ the\ gun\ light$$

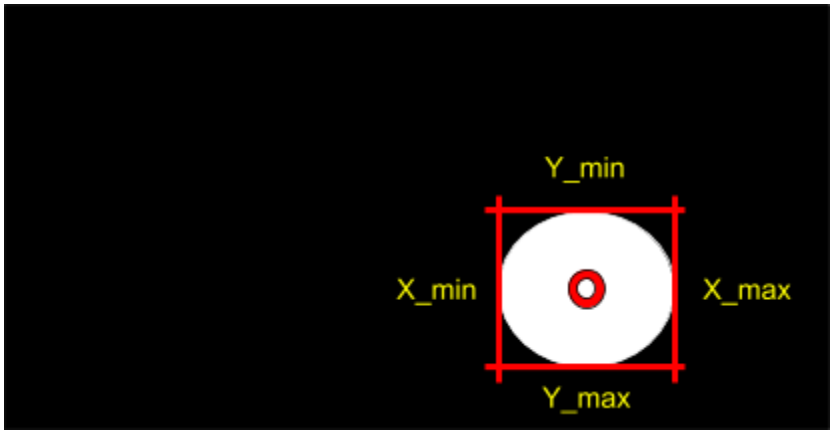
因為相機可以藉由調整clock的數值來改變曝光值，幫助我們限制相機不會被其他光源干擾，畫面只會呈現雷射槍的光。右圖將原VGA影像呈現我們抓出的pixel，會發現多為白光燈管或是較亮的衣服。

B. Position Detection

為了抓到白光位置，我們設計四個參數分別是x_max、x_min、y_max、y_min來框住白光，並將這些數值取平均得到光的位置，但這個方法很容易會因為其他光源干擾而有誤差。由於每次VGA都是移動水平垂直方向都會需要除了pixel還要傳送其他的數值，所以我們限制一個ready表示這個frame讀取完整個螢幕的畫面，取得光源的中間值，就傳出去。

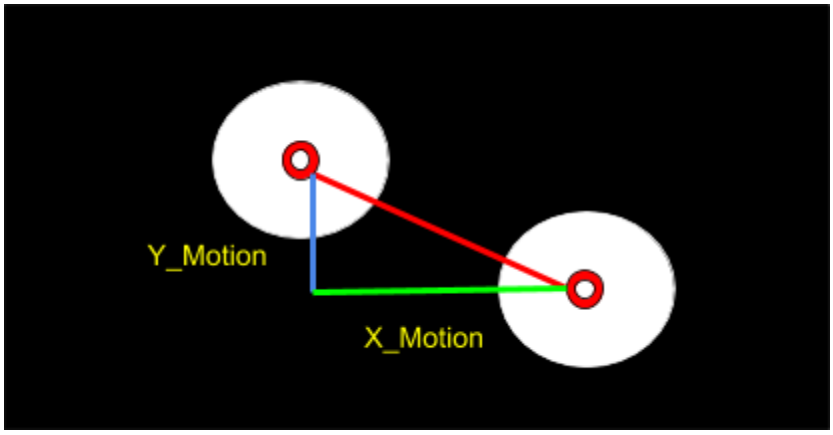
在開始後續步驟之前，我們先將中心的位置以Hexdisplay出來，確保數值是在640x480之中，以利於後續直接呈現在遊戲畫面上，但是會有個很大

的問題，就是玩家必須使用雷射槍移動到某個位子來移動準心，而不是指向某個準心就會將準心移過去，增加玩家使用的負擔。

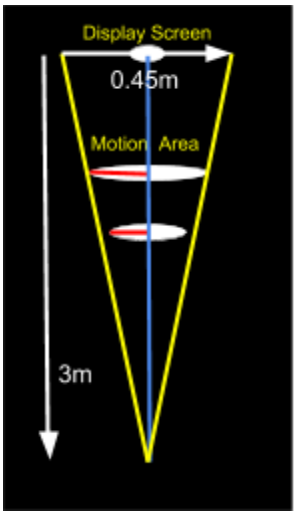


C. Motion Detection

為了解決玩家的方便性，我們多使用另一個方式來偵測雷射光源，就是觀察移動motion，也就是前後兩個frame的中心差異距離，以此距離當作一個標準，最後以一個倍率當作最後準心移動的距離。取得前後距離變化就只需要兩個register存取前後位置，然後看距離差異即可。



但是這個方式很容易因為站的位置距離相機影響很大，而且pixel會有一點小變化，再乘上倍率就被放大，因此我們做了幾個constraint和倍率逼近，希望能達到distance invariant，藉由這些方式就可以使得使用者在每個地方玩遊戲都有最舒適的射擊角度。



- Pixel變化必須大於1個pixel。
- 圓圈大小判斷放大倍率。

我們做了一些假設假設點在離camera最近的時候會是跟螢幕大小一樣大0.45m，距離越遠光點會越小，直到距離6m會使光點消失。我們劃出如右圖的比例，可以發現如果要手最舒適的比例差不多就是光

點半徑跟螢幕半徑的倍率，光點越遠倍率越大，反之光點越近倍率小的話，手要移動較多。

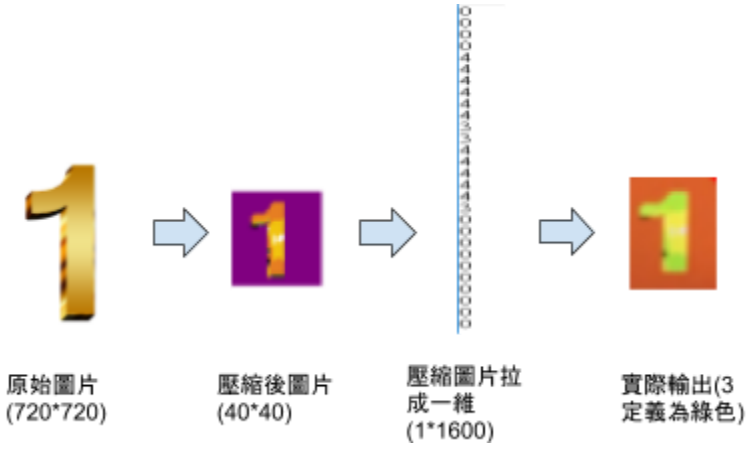
然而透過以上的方式雖然可以做到distant invariant，但是還是會有誤差，而且會有準心較不聽話的狀況，常常手移動的會跟想像的不一樣，所以還是有待加強。

V. Game Interface

A. Image Display

- 機制介紹

  因為vga顯示是1個1個pixel掃的，我們可以直接指定哪個位置要甚麼顏色顯示。故我們從網路抓下所有需要顯示的圖片，將圖片像素存進register中，再利用Color_mapper管理哪個pixel要輸出甚麼顏色。然而因網路圖片通常像素過多，為了避免記憶體爆掉，我們先利用一個python檔把圖片像素調低，並只以0-10的一維數字文檔表示所有的顏色。 之後只要設立每張圖片的flag， 當Stage and Rule Controller判斷此處應該要輸出圖片，就把此圖flag設為1，Color_mapper就根據這個flag讀取該圖片代表的0-10數字。此方法除了使圖片大小縮小，更可以自己重新定義0-10各自代表的顏色。(如下圖)



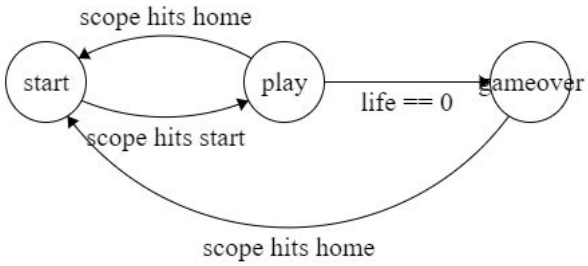
以下分別解釋各module在 image display的應用：

- Stage and Rule Controller :
  控制現在的frame應該要呈現的圖片，並同時控制該state應顯示圖的flag，再將此flag傳給Color_Mapper。
- 每個圖片的ROM :
  將用Python轉好的pixel檔存進registor(若沒降維registor容量會開極大)，等待Color_Mapper決定何時讀取Rom的檔案。
- Color_mapper :

將各個圖片的Rom與Controller丟出來的各個圖形控制flag結合，並指定每個Rom的對應像素顏色。圖片應在畫面何位置輸出也是此處決定。

- 疊圖機制
  因為可能一個畫面點上會同時出現兩張圖片，如何決定先後順序便十分重要(例如當鴨子往上飛時，他的順序就要比背景前面)故在Controller處要把圖片前後關係確認，才不會全部只看到背景圖。

B. Display State Control
State graph 如下(scope即準心的意思)



極為簡單的state，一開始進來為start頁面，畫面呈現初始頁面，當準心打到start，即開始遊戲頁面，最後當生命值歸零時即跳出gameover字樣，藉由準心打Home鍵返回start頁面，同時遊戲中也能藉由打Home鍵來回到主畫面。

然而雖然state簡單，每個state內部都有許多圖片的flag要handle，例如scope要如何在畫面中移動，野鴨被打中前與打中後的運動方向如何，分數顯示如何增減...其實都需要很多flag需handle，以下會針對較複雜的scope和duck運動做說明。

C. Scope Control
- 邊界設定
  首先基本設定準心不會超出邊框範圍，當移動範圍過大，就直接設定準心到邊界而已，不然會造成準心消失。另外我們還多設定一個校準功能，就是在遊戲前我們把準心設在中心，請玩家要將準心對好中心，我們才會開始遊戲，這個目的是為了讓用motion的時候準心跑掉。
- 移動方式
  - Position
    由camera傳過來光源的位置直接當作準星位置，因為兩者都是640x480的VGA畫面，可以直接拼過來，但要處理左右鏡像問題，而且clock必須是更新完一次clock在更新一次位置。

- Motion

  由camera傳過來兩個frame位置差異當作motion移動，可是需要多乘上一個比例的倍率，表示可以小角度移動達到準心移動，另外還要設定motion太小不要變化，不然會使準心一直震動。

  另外再移動之前，多加一個bool function判斷現在要是上下左右，因為我們不能拿前後frame的差異直接來加上去，因為會有負數判斷，造成加上負數無法處理的問題，所以我是存四個正數，要移動或減少多少量，而不是單純position + motion當作下一個目標點。

- 判斷發射

  為了達到發射，我們會將光線明暗當作射擊判斷標準。當玩家將燈熄滅再次開啟表示一次發射，這個也伴隨影響位置判斷，當你發射完畢必須位置不會沒有改變，不然會造成發射完可能準心亂飄，一次跳很遠的距離，並非連續的移動。

D. Duck Control

- 隨機移動

  為了達到像是隨機移動，並且增加各種變化，我們給鴨子XY軸上各有四種速度，X軸上還設定最小及最大邊界，當碰到邊界就會以相反的速度行進，可以做到像是反彈的效果。最後我們給鴨子四個不同的起點，每次起飛都是從不同位置，不同方向，不同邊界，不同速度，來達到隨機移動，給予遊戲很豐富的變化。

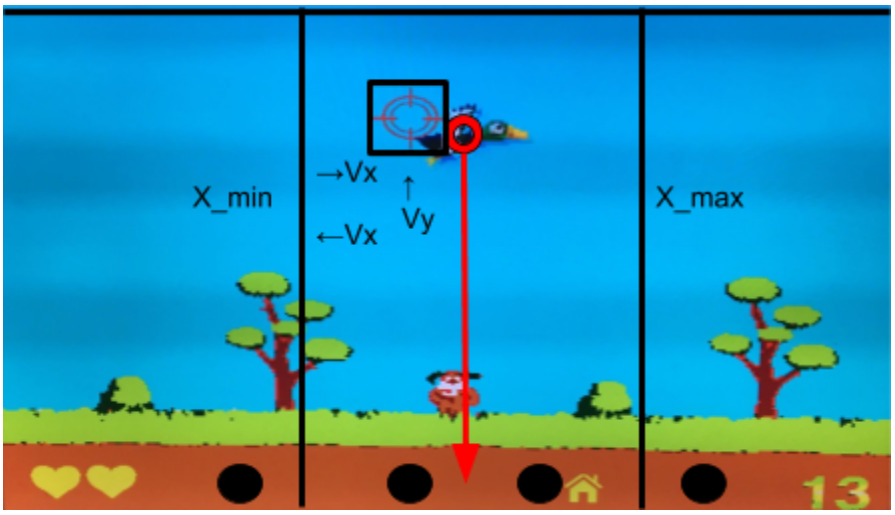
  隨機方式會根據玩家所得分數，以及counter記數來變化，這樣每死一次命，或是每擊中一次野鴨，下一次的野鴨都是不同的狀況，很少會有重複一樣的動作出現。另外當玩家分數過高，我們會加入第二隻鴨子，飛行狀況跟第一隻並不會重複，而且可以做到一槍兩隻，兩隻鴨子是獨立並不會影響對方。

- 判斷擊中

  我們將scope的方框框出來當作示意圖，當鴨子的中心出現在準心的方框內，且有發射子彈就表示擊中，野鴨便會X方向速度改為0，Y方向速度改為垂直往下以每次12pixel的速度往下。

- 判斷損命

  損命的方式只要判斷野鴨是否來到螢幕最上方，到最上方就損一條命，並將野鴨回到初始畫面最下方，在開始飛行



E. Score, Bullet and lives Control

- 分數控制
  當判斷擊中野鴨，就計分數加一，初始是00，最高是99。
- 子彈控制
  當雷射槍發光，先是熄滅再發亮一次，就記一次發射子彈。
- 生命控制
  當野鴨來到最上方，就損失一條命。

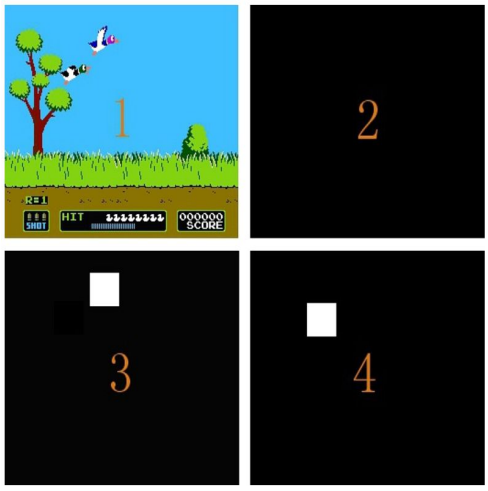
VI. Problem and Discussion:

A. 準心偵測問題:

因為我們是使用相機偵測白光，必須先讓相機偵測的亮度調小，不會受到其他光線干擾，但是用這個方式很看環境影響，很難做成真的適合玩的遊戲。我們覺得有以下幾種改良方式，有傳統也有現代的方法：

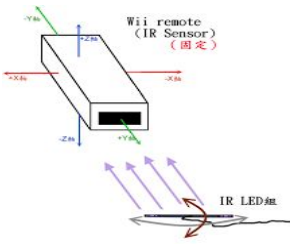
- 傳統NES的槍枝設計

傳統判斷射擊方式是用手槍當作接受端，而非普通思維用螢幕顯示端來偵測。方式是手槍有電子束接受的sensor，手槍有接線當遊戲儀器，當手槍扣下扳機，會讓遊戲畫面出現「黑->白->黑」的閃爍，白色部分就是畫面的target，這樣可以判斷當手槍有接受到010的訊號，即表示有瞄準目標並扣下扳機，方法非常的穩定，對於當時也是一大轟動。然而這個方法有個缺點，就是必須接線到遊戲儀器，而非遠端遙控，對使用者是一種負擔。



- 現代Wii紅外線遙控設計

使用紅外線是因為比較不會受到其他光線干擾，只要找到特定頻率的光即可，環境的影響比較低，而且人類肉眼看不到，如果之後這個遊戲要實作，可以改用紅外線sensor來設計，必較穩定不會準心亂飄。



B. 圖片位置調整

因為每張圖顯示位置不同，我們需要指定圖案在畫片出現的位置。例如呈現分數部分，就曾因限制條件設錯導致畫面被數字塞滿。這解決方法也十分簡單，就要一次次實驗調整擺放位置，以達到最佳效果。

C. 移動偵測誤差很大

當我們是偵測雷射光移動位移來判斷準心變化的時候，還是很容易因為小震動而出現準心亂飄，但是如果設計threshold限制必須位移多少才能算是移動，雖然可以減少小震動的放大誤差，但也造成移動很不連續，很難想要移動到某個地方，就拿正確的到那邊，而是需要手慢慢的過去，使相機比較能偵測到連續動作。

另外就是使用移動偵測，會常常出現要校準的問題，手的位子可能已經移很遠，但是準心還是在中間，造成準星移動的平面來到很不好的角度，而需要重新校準。

解決方式是希望可以透過九軸感測器之類的，判斷移動方式，就像是wii遙控器一樣，可以很準抓到手的移動方式，而且也可以達到遠端遙控的功能。

VII. Future Work：

A. 可在動畫加以精進，如加入拍翅膀動畫與鴨子落下表情。
B. 增加遊戲機制，新增第二玩家，兩人須競爭誰最快打到野鴨。
C. 增加野鴨random方式，讓軌跡能夠再更多變化。
D. 準心可以紅外線偵測，較不會受到外在環境影響。

VIII. Demo Video：https://youtu.be/NE5AVcDUJI8