LOGIC DESIGN LAB

FINAL REPORT

106011206 電機大二 林俊曄

106011201 電機大二 陳昭霖

2019 年 6 月 24 日

EE2030 Logic Design Lab Final Report

T-Rex RUN

●Design Specification

Input:

clk,(接上板子的原本的震盪頻率 W5)

rst_n, (控制整個功能的開關)

pb_volume_up(讓遊戲音量變大)

pb_volume_down(讓遊戲音量變小)

pb_volume_mute(靜音/開啟聲音)

pb_highest_score(切換最高分數還有當前分數)

switch_undead(無敵模式)

switch_clk(加速模式)

Output:

[3:0]vgared, (used to control the red color, 0v(fully off) ~ 0.7v (fully on))

[3:0]vgagreen, (used to control the green color, 0v(fully off) ~ 0.7v (fully on))

[3:0]vgablue,(used to control the blue color, 0v(fully off) ~ 0.7v (fully on))

[7:0]segs, ,(接上七段顯示器的七條燈)

[3:0]scan_ctl, (接上四個七段顯示器，控制他們亮暗的頻率)

Mclk(master clock，是最快的 clk 讓其他 clk 校準,25MHz)

Lrclk(left-right clock，是要給左聲道和右聲道訊息的,25MHz/128)

Sdin(serial clk,讓一個個訊號進去的頻率，比 lrclk 的頻率快 32 倍,25Mhz/4)

Sclk(1 bit serial audio data output)

[10:0]LEDs(顯示當前音量大小，如果音量越大則亮燈越多)

Hsync,(used for video synchronization in the vertical direction)

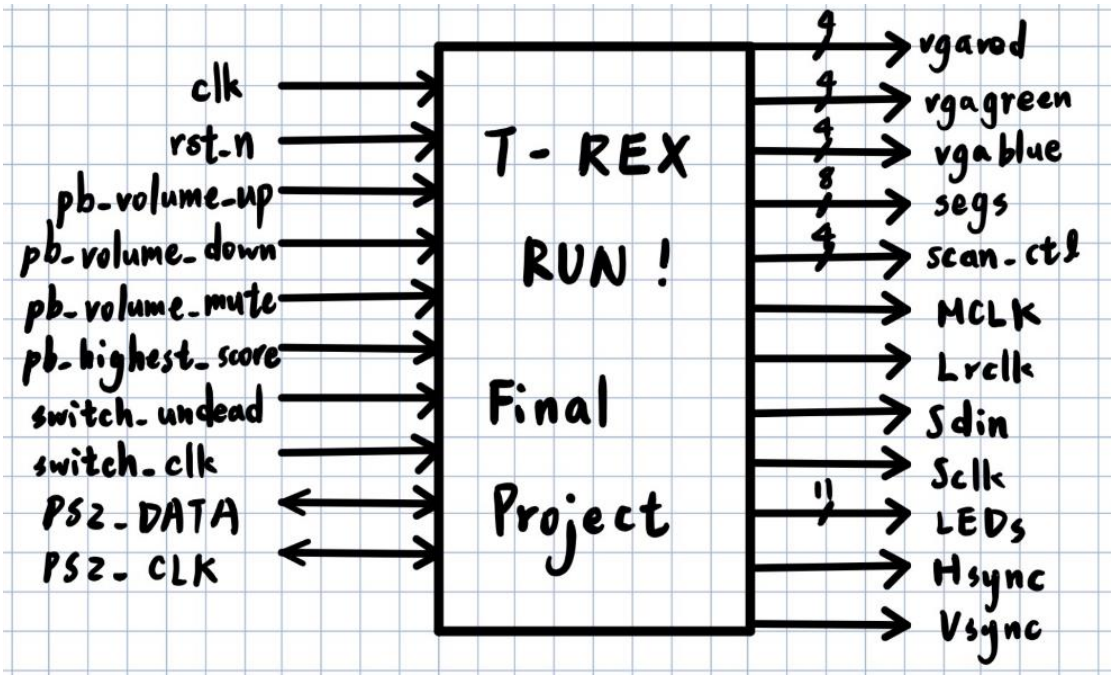
Vsync(used for video synchronization in the horizontal direction)

InOut:

PS2_DATA(讓 PS2,KEYBOARD 彼此溝通用)

PS2_CLK(讓 PS2,KEYBOARD 彼此溝通用)

(在這邊我們只標上實際有用的，其餘有些是為了 debug 方便而加上的 output 則不在此說明)



●Design Implementation

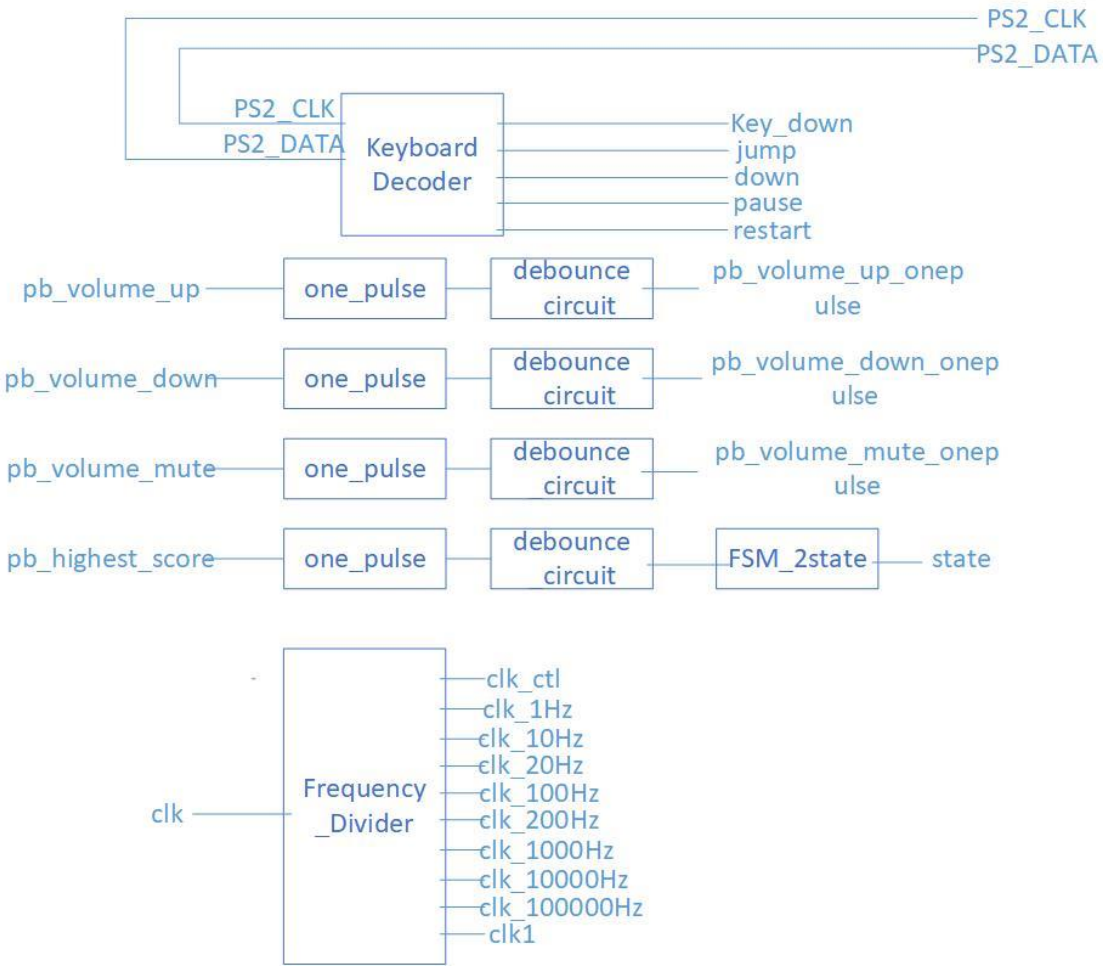
1.Outline

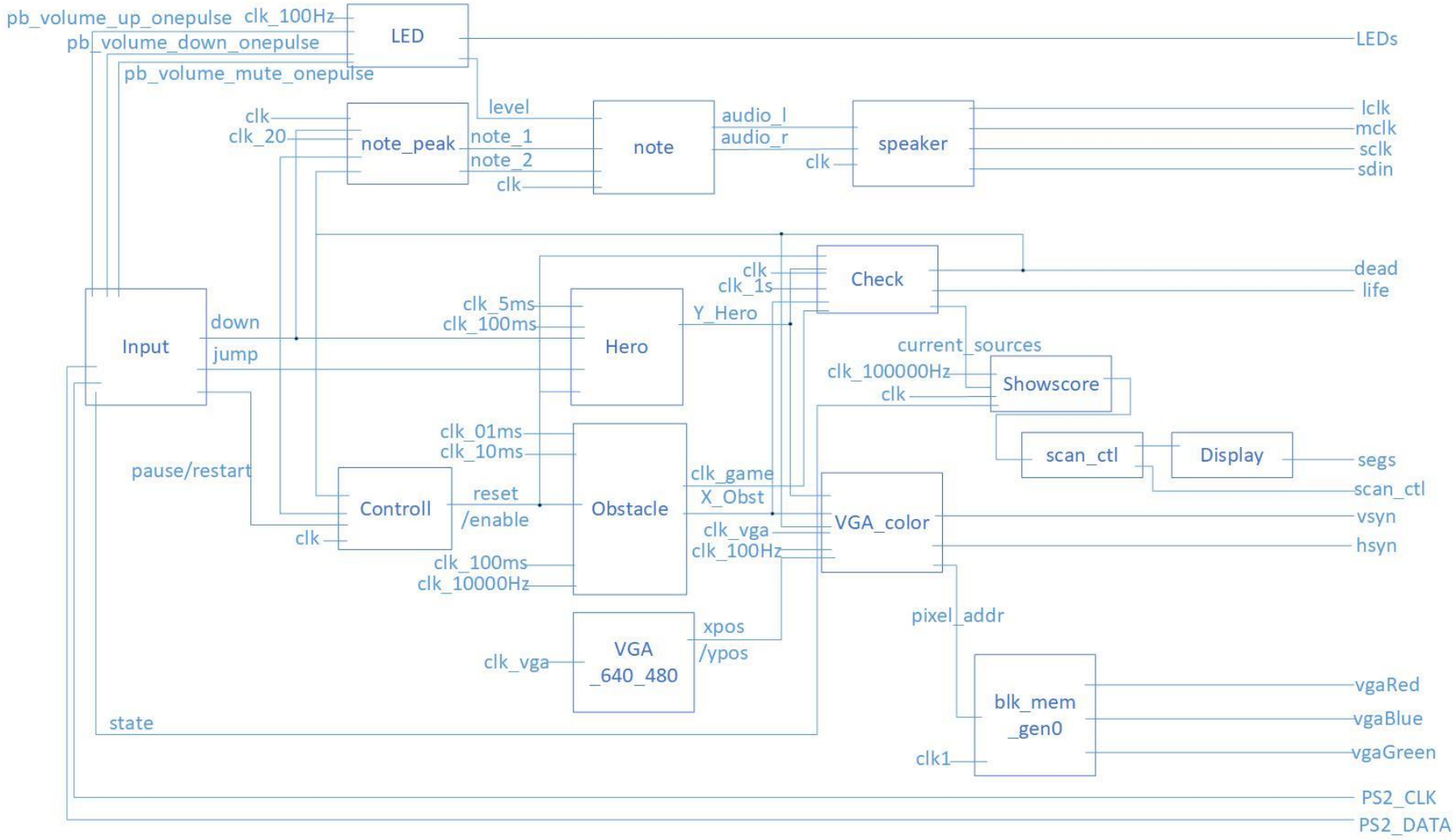
一開始想說做 google chrome 小恐龍覺得應該挺有趣的，但想不到這份有趣竟然讓我們花了很久的時間，總體來說可真的是費了很大一番功夫。

這一個 final project 可說是費勁了我們這學期所學，毫無保留，input 端中所用到按鍵的部分，需要使用到 one pulse 和 debounce 以及 FSM 的使用，再來因為需要使用鍵盤操控，因此也有接上 Keyboard decoder，並 assign 所需要的按鍵，而最麻煩的地方便是 VGA 的掃圖以及顯示，需要配合繪圖、掃圖的顯示及判斷。我們還有增加背景音樂以及音效，以及將音量大小以及靜音顯示在 LED。增加小恐龍的生命，使其有三條命，損命的時候會一閃一閃。最後還存入另外一張圖使主角從小恐龍變成青蛙馬力歐。

2.Logic Diagram

For input




pb_volume_up_onepulse
clk_100Hz
pb_volume_down_onepulse
pb_volume_mute_onepulse
LED
LEDs
clk
clk_20
note_peak
level
note_1
note_2
clk
note
audio_l
audio_r
clk
speaker
lclk
mclk
sclk
sdin
clk
clk_1s
Check
dead
life
Input
down
jump
clk_5ms
clk_100ms
Hero
Y_Hero
current_sources
clk_100000Hz
clk
Showscore
scan_ctl
Display
segs
scan_ctl
pause/restart
clk_01ms
clk_10ms
Controll
reset
/enable
clk
Obstacle
clk_game
X_Obst
clk_vga
clk_100Hz
VGA_color
vsyn
hsyn
clk_100ms
clk_10000Hz
clk_vga
VGA
_640_480
xpos
/ypos
pixel_addr
blk_mem
_gen0
clk1
vgaRed
vgaBlue
vgaGreen
state
PS2_CLK
PS2_DATA

▲VGA_640*480

這個 module 是用來控制 h_cnt、v_cnt、hsync、vsync、vaild。

hsync、vsync 主要用來給螢幕做顯示的，因為螢幕顯示有點像是七段顯示器，不過是每一個 pixel 都要掃過，利用 800*525*60(frame/sec) = 25M(pixel/sec) 速度的 clk 來掃過來達成每秒螢幕更新 60 張的效果。

h_cnt、v_cnt 主要是用來給你做你想要畫面的輸出的，我覺得有點像是 pointer 的觀念，當 h_cnt、v_cnt 指著特定的圖片上面的特定點的時候，vga_rgb 就會回去找那張圖片上面的位置並顯示該畫面。

Vaild 是讓螢幕知道該在什麼時候顯示，因為螢幕只有 640*480 的大小，換言之在外面其他地方不能亮，所以才會有下方式子產生。

```
assign {vgaRed, vgaGreen, vgaBlue} = (valid==1'b1) ? pixel:12'h0;
```

▲ VGA_Color

這個 module 主要是讓 640*480 轉變成 320*240 的解析度，還有控制所有利用掃描圖片所產生的物件。

這邊我們分為好幾種不同的物件，所處理的方式也不一樣，大致可以分為下列幾種:

1. 需要多張圖片切換來產生物件在移動的效果，e.g.小恐龍、鳥
2. 需要隨時間移動的物件，e.g.仙人掌
3. 不需要隨時間移動的物件，e.g.血量、地板

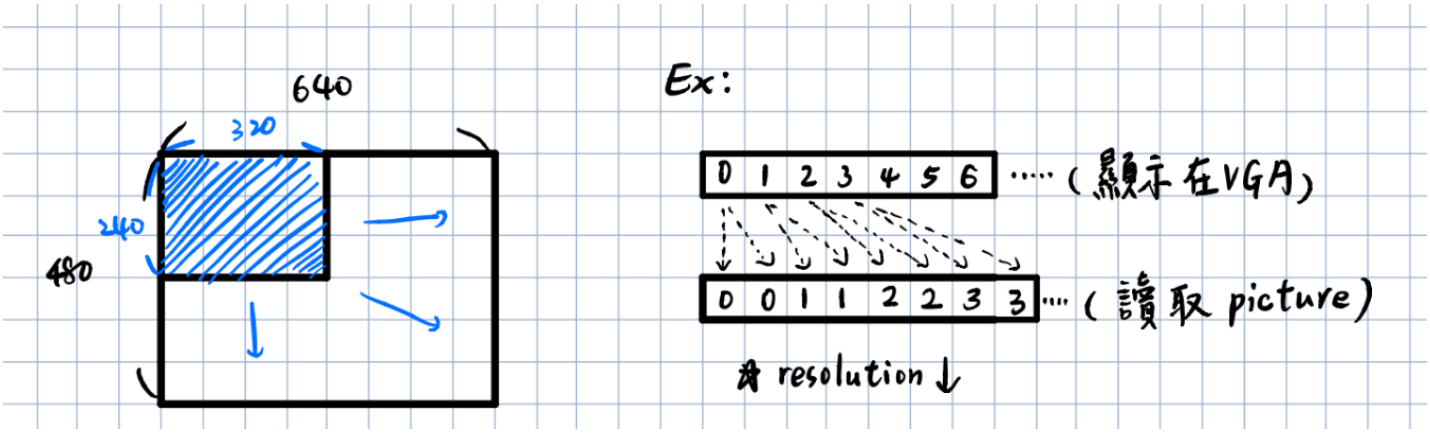
而這幾種在 module 中分為兩個步驟，第一個是**掃描**，第二個是把掃描的圖片**顯示**在特定該區域，這邊舉比較複雜的小恐龍為例:

a. 掃描:

```
//DINO WALK
always@*
begin
   case(showmode)
         1'b0 : begin
                    addr_hero = (kp_down==0)? ((xposin>>1)) % 60 + (((yposin+112-Y_hero)>>1)) % 56 * 320 + 65  + 25 * 320 : ((xposin>>1)+60) % 60 + ((yposin+112-Y_hero)>>1) % 56 * 320 + 192  + 25 * 320;
                end
         1'b1 : begin
                    addr_hero = (kp_down==0)? ((xposin>>1)) % 60 + (((yposin+112-Y_hero)>>1)) % 56 * 320 + 130 + 25 * 320 : ((xposin>>1)+60) % 60 + ((yposin+112-Y_hero)>>1) % 56 * 320 + 258  + 25 * 320;
                end
      default:   begin
                    addr_hero = (kp_down==0)? ((xposin>>1)) % 60 + (((yposin+112-Y_hero)>>1)) % 56 * 320 + 65 + 25 * 320 : ((xposin>>1)+60) % 60 + ((yposin+112-Y_hero)>>1) % 56 * 320 + 192  + 25 * 320;
                end
   endcase
end
```

這邊的 showmode 是為了讓小恐龍切換兩張照片，而顯示出在行走的畫面；而在判斷式中的(kp_down?)則是為了另外兩張蹲走的畫面。

xposin>>1、ypisin>>1，也就是向右 shift 一個 bit 可以讓解析度變差，因為在二進位中 shift 右邊一個 bit 代表著除以二，而 xposin、yposin 分別代表著橫軸與縱軸，所以我們把兩個都除以二，最後結果會讓整個面積變成只剩下四分之一，達到所要的效果。(如下圖)



之後利用到%來取餘數，是為了切割我們所想要顯示的特定圖片，透過 Xposin%60、和(((yposin+112-Y_hero)>>1)) % 56 * 320，我們可以切到一個 60*56 大小的方格，也就是我們小恐龍的大小。(要*320 則是因為 pixel_addr 只有一維，所以要利用*320 來記錄二維的數字。)

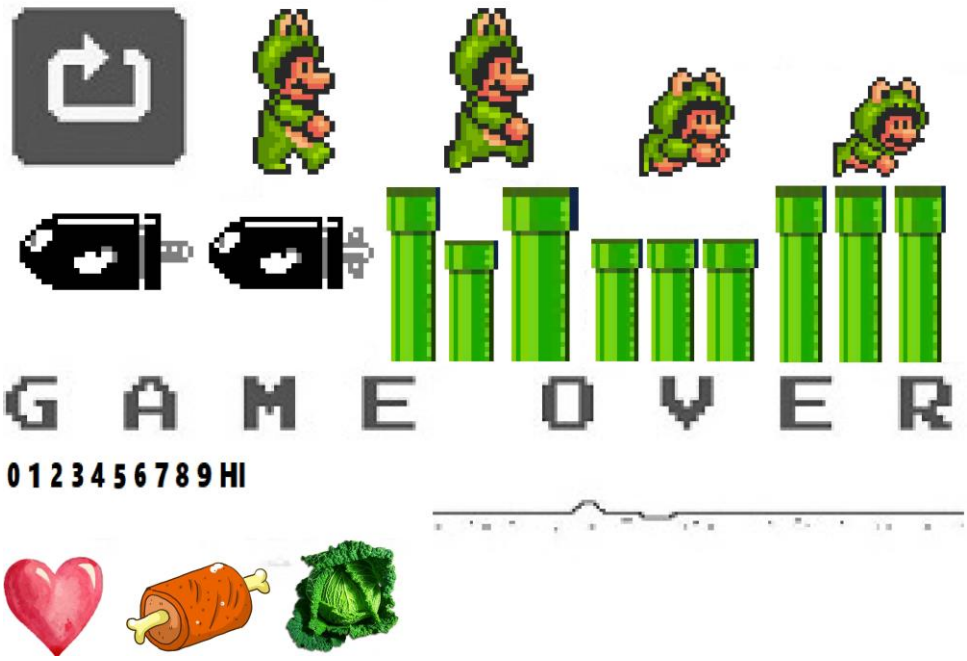
此外，我覺得最值得一提的是要-Y_hero 這個變數的原因，起初我們並沒有加上這個敘述，因為小恐龍是只對 y 軸做變化，也就是說當小恐龍跳起來的時後，理當需要一個可以表現該高度的變數來調整掃描的區域(跳起來的時候也要掃描的部分也要跳起來)，如果沒有加上這條敘述，會讓小恐龍有很多隻在畫面上，然後我們顯示的方塊就像是一個窗戶，會變成窗戶再跳，然後看到窗戶後面的很多隻小恐龍，非常詭異。

b. 顯示

顯示的部分重點是要切出跟之前在圖片上面切割時候的大小要一樣，這樣子最不容易有問題，還有一點是因為利用 if-else 的 statement，所以如果想要控制哪張圖片會顯示在哪張圖片上面的話，就要改變 if-else 的順序，越上面的會顯示在越突出的位子。

```
always@*
begin
   if((xposin > X_obst0 - wid_obst) && (xposin < X_obst0) && (yposin > Y_obst - 112) && (yposin < Y_obst))
       pixel_addr = addr_obst0;
   else if((xposin >= X_obst1 - wid_obst) && (xposin < X_obst1) && (yposin > Y_obst - 112) && (yposin < Y_obst))
       pixel_addr = addr_obst1;
   else if((xposin >= X_obst2 - wid_obst) && (xposin < X_obst2) && (yposin > Y_obst - 112) && (yposin < Y_obst))
```

我們有兩種模組，第一個是一般的小恐龍，圖是去真正小恐龍網頁遊戲裡面抓的；另外是青蛙馬力歐的模組，圖片如下，都是 320*240pixel 的圖片。



▲Hero

這個 module 主要是在控制小恐龍的跳躍，還有切換兩張照片的時機，也是這個 final，蠻重要的 module 之一。

首先，關於兩張圖片切換的部分，我們利用了 clk_100ms 來控制

```
always@(posedge clk_100ms)
begin
    if(reset)
        showmode = 1'b0;
    else
    begin
        if(enable)
        begin
            if(y_hero == y_initial)
                showmode = ~showmode;
                else showmode = 1'b0;
        end
    end
end
```

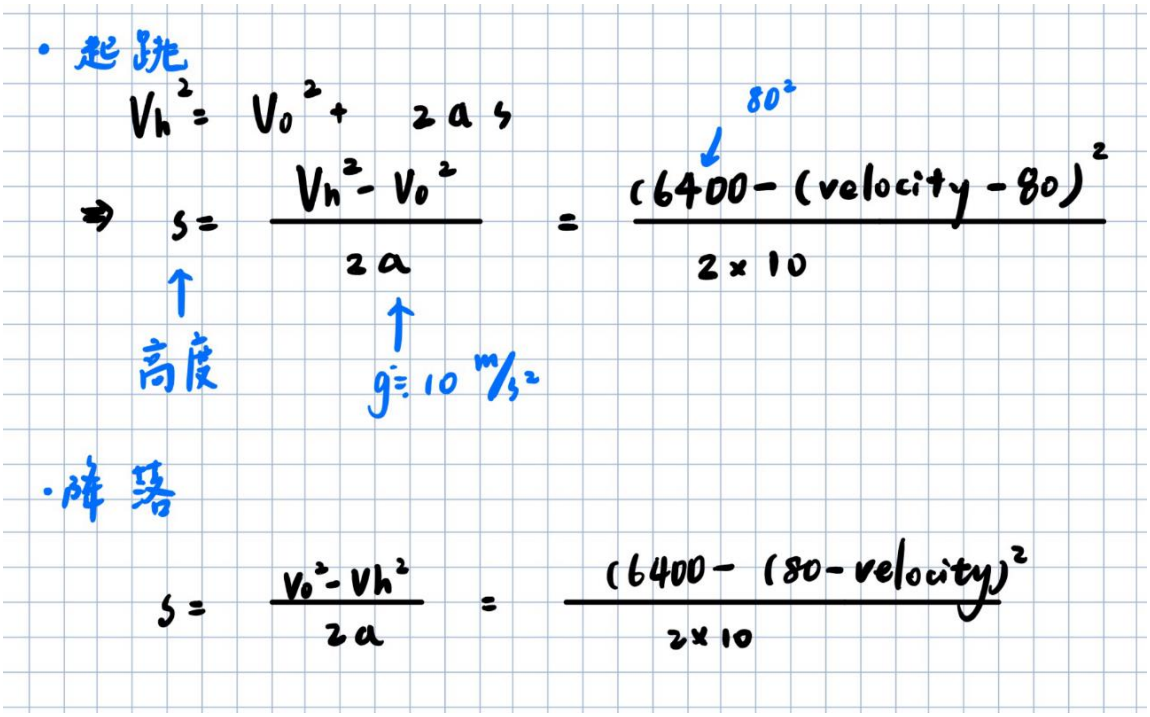
當 y_hero == y_initial 的時候，也就是小恐龍沒有跳躍的時候(如果跳躍的時候還在走路很白癡)，會一直切換兩張照片。

```
begin
    if(enable)
    begin
        if(jump&&(velocity == 0))
            begin velocity = 160; end
        else
            begin velocity = velocity; y_hero = y_hero; end
            if(velocity >= 1 && down == 0)
                begin
                    velocity = velocity - 1'b1;
                   if(velocity >= 80)
                          y_hero = y_initial - ((6400-(velocity-80)*(velocity-80))/20);
                    else
                          y_hero = y_initial - ((6400-(80-velocity)*(80-velocity))/20);
                end
           else if(velocity >=1 && down == 1)
               begin
                   velocity = velocity - 4'd2;
                  if(velocity >= 80)
                          y_hero = y_initial - ((6400-(velocity-80)*(velocity-80))/20);
                  else
                          y_hero = y_initial - ((6400-(80-velocity)*(80-velocity))/20);
                  end

    end
```

↑關於跳躍的部分，我們利用到了高中時所學的運動學公式

為了減少麻煩，我們把 velocity==80 的時候當作原點，所以當 velocity==0 的時候，也就是速度-80 的時候。當我們按下跳躍鍵的時候 velocity 直接到 160，然後會隨著 clk_5ms 的時間遞減回到原本的速度， 下面是推導式子。



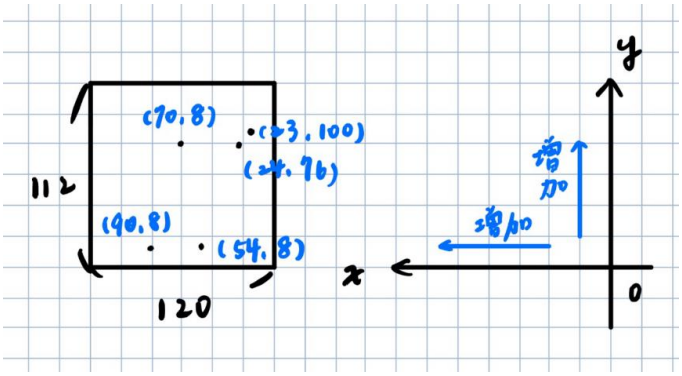
▲Check

這個 module 主要是來判斷小恐龍是否有碰到障礙物和分數計算，這邊我們同樣舉一部分作為例子:

```
if((enable)&&(!undead)&&(~kp_down)&&(~next_dead))
begin
    //obstacle 0
    if((x_hero - 23 < x_obst0)&&(x_hero- 23 + wid_obst > x_obst0)&&(y_hero- 100 > y_obst-heg_obst0))
        dead = 1'b1;
    else if((x_hero - 24 < x_obst0)&&(x_hero- 24 + wid_obst > x_obst0)&&(y_hero - 76 > y_obst-heg_obst0))
        dead = 1'b1;
    else if((x_hero - 54 < x_obst0)&&(x_hero- 54 + wid_obst > x_obst0)&&(y_hero - 8 > y_obst-heg_obst0))
        dead = 1'b1;
    else if((x_hero - 90 < x_obst0)&&(x_hero- 90 + wid_obst > x_obst0)&&(y_hero - 8 > y_obst-heg_obst0))
        dead = 1'b1;
    else if((x_hero - 70 < x_obst0)&&(x_hero- 70 + wid_obst > x_obst0)&&(y_hero - 80 > y_obst-heg_obst0))
        dead = 1'b1;
```

↑這是對第一個仙人掌的判斷式

可以看成是我們在小恐龍的身體上點很多點，當這些點碰到障礙物的時候，即會觸發 dead，下圖圖示:



這邊 x 軸是越左越大，左邊方塊代表的是小恐龍的圖片，當障礙物碰到我設定的這些點的時候，就會觸發死亡條件。

此外，我們有加上三條血，所以當我們小恐龍扣一次血的時候，會進入短暫無敵模式，小恐龍身上也會一閃一閃，在這段期間如果又碰到其他的障礙物則不會有傷害。

```
always@(posedge clk_1s or posedge reset)
begin
    if(reset)
        next_dead <= 1'b0;
    else if(dead == 1)
        next_dead <= dead;
    else
        next_dead <= 1'b0;
end
```

←利用一個 1s 的 D_flip flop 來延遲，用 next_dead 來當作判斷條件。(受傷後一秒內不會再次受傷)

```
//not pause and not invisible mode
if((enable)&&(!undead)&&(~kp_down)&&(~next_dead))
```

分數部分就是普通的 upcounter，每一秒往上加一，所以分數也可以看做是遊玩時間。

▲Obstacle

這個 module 主要控制著障礙物的控制、遊戲速率的變化、還有鳥的兩張圖片切換。

障礙物方面，我們設定了三種不同的仙人掌、還有兩種速度不同的鳥，利用 random module 來控制彼此的間距、出現頻率，也就是說我們遊戲的障礙物會隨機出現。

```
initial
begin
    x_obst0 = 610 + 250;
    x_obst1 = 610 + 350 + 250;
    x_obst2 = 610 + 350 + 350 + 250;
    x_bird_obst0 = 1260 + 350 + 350 + 250;
    x_bird_obst1 = 2780 + 350 + 350 + 250;
    diff = 5'd10; count0 = 32'd0; count1 = 32'd0;
end
```

←先設定好起始值，多次調整到比較合理的範圍，盡量避免在有仙人掌上面又有鳥的情形發生，不然非常容易掉血。

之後我們為了讓遊戲難度可以遞增，所以要讓遊戲的 clk_game 隨著時間變得越來越快，為了達到這樣的效果，我們先利用兩個 upcounter:

```
//diff can controll clk_gmae ; clk_game can controll the speed of  background
always@*
if(count0 >= diff)
    begin
        count0_next = 32'b0;
        clk_game_next = ~clk_game;
    end
else
    begin
        count0_next = count0 +1'b1;
        clk_game_next = clk_game;
    end
```

第一個是用來調整 clk_game，當 count 到 diff(我們的控制變數)的時候會 toggle，如果 diff 越小，clk_game 頻率就會越快)。

```
// diff decrease and clk_game increase while
always@*
if(count1 == 5'd30 && diff >= 32'd5 )
  begin
      diff_next = diff - 1'b1;
      count1_next = 32'd0;
  end
else
  begin
      diff_next = diff;
      count1_next = count1 + 1'b1;
  end
```

第二個 counter 則是控制 diff 變數，當 count1 每數到 30 的時候，會讓 diff-1 直到變成 5 的時候停止，也就是遊戲速度最快的時候。

最後是障礙物本身的部分:

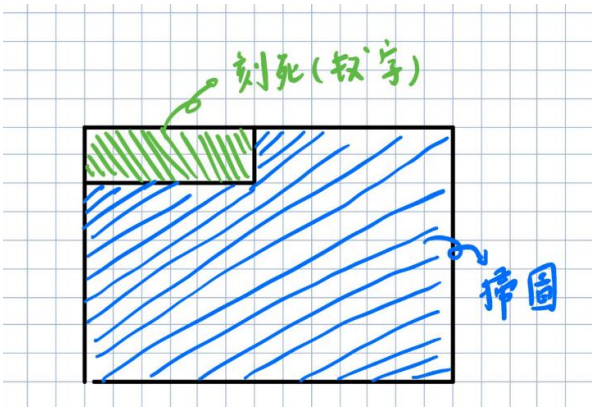
```
if(enable)
begin // reset position once it approach 0
    if(x_obst0 == 0)
        x_obst0 = x_obst2 + 300 + (240-diff*10) + random;
    else if(x_obst1 == 0)
        x_obst1 = x_obst0 + 370 + (240-diff*10) + random;
    else if( x_obst2 == 0)
        x_obst2 = x_obst1 + 420 + (240-diff*10) + random;
    else if( x_bird_obst0 == 0)
        x_bird_obst0 =  x_obst1 + 1500 + (240- diff*10) + random;
    else if( x_bird_obst1 == 0)
        x_bird_obst1 =  x_bird_obst0 + 8500 + (240- diff*10) + random;
   // push postion back
      x_obst0 = x_obst0 - 1;
      x_obst1 = x_obst1 - 1;
      x_obst2 = x_obst2 - 1;
      x_bird_obst0 =  x_bird_obst0 - 1;
      x_bird_obst1 =  x_bird_obst1 - 2;
    end
```

彼此障礙物的位置環環相扣，當一個障礙物到達終點的時候，會重新 initialize 起始位置，所以說不會有兩個仙人掌同時疊起來的狀況發生。

鳥有兩種速度，一個跟仙人掌一樣快，一個則是兩倍快。

▲ pixel_gen

這個 module 控制的是螢幕數字的顯示，原本我們想要直接用跟小恐龍方式顯示數字，但是嘗試很久之後都沒辦法成功，所以最後我決定心一橫，直接用寫死的方式刻在上面，像是刻上七段顯示器上去，這樣會比較清楚



我利用這樣的判斷式，來達到這樣的結果:

```
always@*
  begin
      begin
          if(N2 % 2 == 1'b0 && xpos > 0  && xpos <= 250 && ypos >0 && ypos <55)
              {vgaRed, vgaGreen, vgaBlue} = ~pixel_num;
          else if(N2 % 2 == 1'b1 && xpos > 0  && xpos <= 250 && ypos >0 && ypos <55)
              {vgaRed, vgaGreen, vgaBlue} = pixel_num;
          else if(N2 % 2 == 1'b0)
              {vgaRed, vgaGreen, vgaBlue} = (valid==1'b1) ? pixel0:12'h0;
          else
              {vgaRed, vgaGreen, vgaBlue} = (valid==1'b1) ? ~pixel0:12'h0;
      end
  end
```

當 xpos 在 250 之前，且 y_position 在 55 之前，vgared、vgagreen、vgablue 會是 Pixel_num，相當於在螢幕左上角切出一個方塊給數字顯示，其他都是利用掃圖的方式顯示。(會有 complement 的部分是我們有做出晚上的狀態，會讓白天變黑夜)

刻死的方式非常麻煩又暴力:

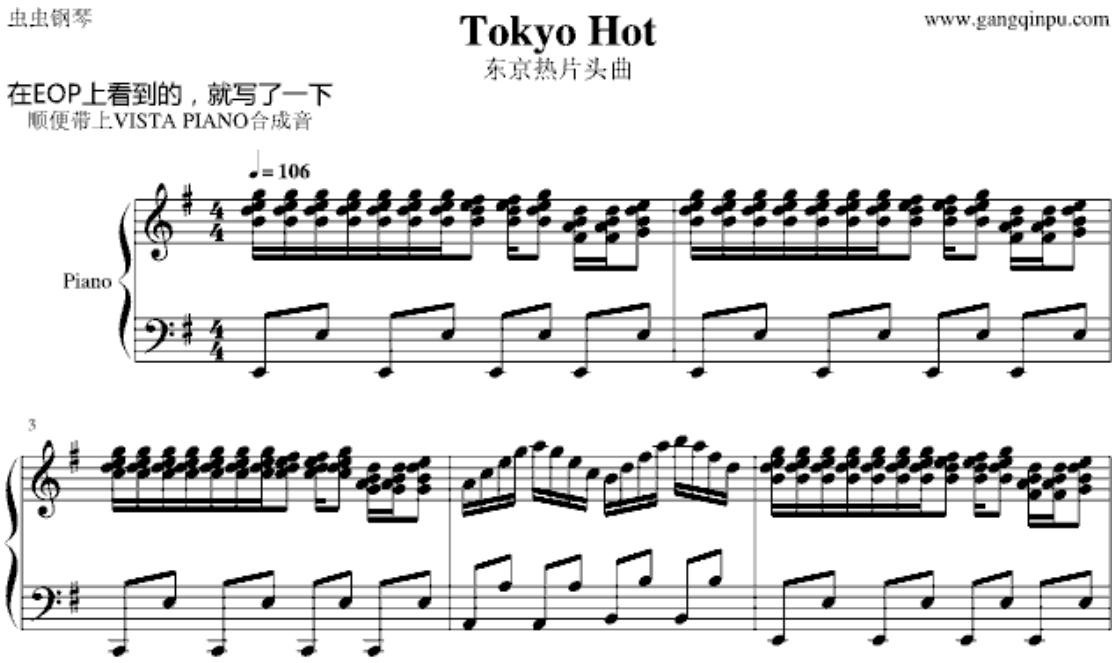
```
case (digit3)
4'd0:
begin
   if(h_cnt <= 30 && h_cnt >= 10 && v_cnt <= 8 && v_cnt >= 5)
        {vgaRed, vgaGreen, vgaBlue} = 12'hfff;
   else if(h_cnt <= 30 && h_cnt >= 27 && v_cnt <= 28 && v_cnt >= 8)
        {vgaRed, vgaGreen, vgaBlue} = 12'hfff;
   else if(h_cnt <= 30 && h_cnt >= 27 && v_cnt <= 48 && v_cnt >= 28  )
        {vgaRed, vgaGreen, vgaBlue} = 12'hfff;
   else if(h_cnt <= 30  && h_cnt >= 10 && v_cnt <= 51  && v_cnt >=  48 )
        {vgaRed, vgaGreen, vgaBlue} = 12'hfff;
   else if(h_cnt <= 13 && h_cnt >= 10 && v_cnt <= 48 && v_cnt >= 28  )
        {vgaRed, vgaGreen, vgaBlue} = 12'hfff;
   else if(h_cnt <= 13 && h_cnt >= 10 && v_cnt <= 28 && v_cnt >=  8 )
        {vgaRed, vgaGreen, vgaBlue} = 12'hfff;
   else if(h_cnt <= 30 && h_cnt >= 10 && v_cnt <= 28 && v_cnt >=  25 )
```

↑簡單來說就是把七段顯示器寫死在上面，每一段的每一個數字，都要寫一塊地 case 來判斷，總共寫了 

行，真的不是普通的麻煩。

▲note

這個模組是用來作背景音樂的，這次所挑的是某個日本有名的製片廠所創作的電子音樂，分成左右兩聲道，左耳是 bass 的部分，右耳是主要的旋律。用兩個不同的 clk(b_clk 和 c_clk)分別運用在做右兩聲道，先宣告每個音所要用的頻率，在將其用 case 存(左右各 640 個,1280 個 case)，再用 20Hz 的 clk 去跑，以顯示在以下圖的鋼琴譜為例，因為其音符的最小單位是十六分音符，為了做出點音的效果，因此 case 需要以 32 分音符為最小單位去設計音樂。




```
12'd91: b_note_div = `F;
12'd92: b_note_div = `n;
12'd93: b_note_div = `G;
12'd94: b_note_div = `G;
12'd95: b_note_div = `n;
12'd96: b_note_div = `n;
12'd97: b_note_div = `a;
12'd98: b_note_div = `n;
12'd99: b_note_div = `C;
12'd100: b_note_div = `n;
12'd101: b_note_div = `E;
12'd102: b_note_div = `n;
12'd103: b_note_div = `G;
12'd104: b_note_div = `n;
12'd105: b_note_div = `A;
12'd106: b_note_div = `n;
12'd107: b_note_div = `G;
12'd108: b_note_div = `n;
```

為了做出斷音的效果，需要用在使用 case 的時候將每個音的後面用一個靜音(n)，以下面為例，case 93~96(GGnn)代表的是 16 分音符長度的 G，以及 16 分音符的休息，所以合起來便是譜上的 16 分音符。以此類推 case 103~106(GnAn)便是 16 分音符的 G 跟 A。最後再將左右兩聲道的音量藉由按鍵操控 level 的改變，給其不同的振幅。

```
case(level)
        4'd15, 4'd14, 4'd13:
        begin
            audio_l = (b_clk == 1'b0) ? 16'hB000 : 16'h5FFF;
            audio_r = (c_clk == 1'b0) ? 16'hB000 : 16'h5FFF;
        end
        4'd12:
        begin
            audio_l = (b_clk == 1'b0) ? 16'hB800 : 16'h57FF;
            audio_r = (c_clk == 1'b0) ? 16'hB800 : 16'h57FF;
        end
        4'd11:
        begin
            audio_l = (b_clk == 1'b0) ? 16'hC000 : 16'h4FFF;
            audio_r = (c_clk == 1'b0) ? 16'hC000 : 16'h4FFF;
        end
```

▲note_peak

這個模組是來決定左右兩聲道的 counter，因為音樂有些地方有反覆，因此為了節省些 case 的使用，所以當 counter 跑到某些特定的值後，會藉由 D-flipflop 的存值，回到先前的地方再跑一次，並在音樂結束的時候，從頭再跑一次。如下圖當 counter 跑到 640 的時候會回到 0。而我也在此將音效設計進去，當跳/蹲的時候，如下圖我會將 counter 移到 case646 的位置，並用 note1_store、note2_store 存原先的位置，等待期跑了 4 個 case 後再接回來原本 counter 所存的位置再+4，而相對音的是 8 分音符的低音 E(跳是 G)，而死亡則是用一樣的方式街道 case650 但這時候不用接回來因為此時不會有聲音，加上 default 我設為 0，所以讓 counter 一直往上跑也都不會有聲音，等到 jump 後整個遊戲重新開始，便又會回到 0 重新開始。

```
else if (kp_down==1&& note_1<=640)
begin
    note1_next = 646;
    note2_next = 646;
    note1_store = note_1;
    note2_store = note_2;
end

else if (note_1==649)
begin
    note1_next = note1_store+4;
    note2_next = note2_store+4;
end

else if(note_1 == 12'd640 && note_2 == 12'd640)
begin
    note1_next = 1;
    note2_next = 1;
    A_tmp=0;
```

```
////jump
        12'd642: b_note_div = `g;
        12'd643: b_note_div = `g;
        12'd644: b_note_div = `n;
        12'd645: b_note_div = `n;

////down
        12'd646: b_note_div = `e;
        12'd647: b_note_div = `e;
        12'd648: b_note_div = `n;
        12'd649: b_note_div = `n;

////dead
        12'd650: b_note_div = `n;
        12'd651: b_note_div = `c;
        12'd652: b_note_div = `n;
        12'd653: b_note_div = `c;
```

↓以下是 I/O 接腳

Input

For button

pb_highest_score	pb_volume_up	pb_volume_down	pb_volume_mute
W19	U17	T17	T18

DIP switch and other

clk	rst_n	switch_clk	switch_mode	switch_undead
W5	R2	V16	T1	V17

Output

For SSD

segs[7]	segs[6]	segs[5]	segs[4]
W7	W6	U8	V8

segs [3]	segs [2]	segs [1]	segs [0]
U5	V5	U7	V7

scan_ctl [3]	scan_ctl [2]	scan_ctl [1]	scan_ctl [0]
W4	V4	U4	U2

For LEDs

LEDs [10]	LEDs [9]	LEDs [8]	LEDs [7]	LEDs [6]	LEDs [5]
U15	U14	V14	V13	V3	W3
LEDs [4]	LEDs [3]	LEDs [2]	LEDs [1]	LEDs [0]	dead
U3	P3	N3	P1	L1	W18

life [3]	life [2]	life [1]	life [0]
V19	U19	E19	U16

For speaker

mclk	lrclk	sclk	sdin
A14	A16	B15	B16

For vga display

vgaRed[3]	vgaRed [2]	vgaRed [1]	vgaRed [0]
N19	J19	H19	G19
vgaBlue [3]	vgaBlue [2]	vgaBlue [1]	vgaBlue [0]
J18	K18	L18	N18
vgaGreen [3]	vgaGreen [2]	vgaGreen [1]	vgaGreen [0]
D17	G17	H17	J17

hsync	vsync
P19	R19

Inout

PS2_CLK	PS2_data
C17	B17

●Discussion

這一次的期末專題可說是有很多困難需要我們克服，光是 block diagram 就讓我們無從下手，其中我們第一個遇到的問題便是如何讓小恐龍進行跳躍，原本想說就他直上直下等速率顯示，但是後來測得的效果並不是很好，看起來有怪怪的，後來才想出運運直線運動公式的第三個 $v^2 = v_0^2 + 2as$ 將小恐龍的 y 座標設為此，再來仙人掌的部分需要用到 random 的概念與 lab11_3 相同。而 clk_game 要由 2 個變數去操控，以達到視覺上速度隨遊戲時間的進行而改變的視覺效果。再來用仙人掌的座標即小恐龍的座標去判斷是否有接觸到，如果有的話便會損血或死亡。再來是接上音樂，使其有背景音樂和跳/蹲/死亡的音效。並運用 AI 和小畫家等工具，去產生所要跑的原圖，並將其設定成需要使用的 pixel 數，這樣才能掃圖上特定區塊的範圍，定將其顯示在螢幕上的特定區域，這也是我們這一次 project 的大魔王，可說是非常的困難，因為明明就照著圖上所顯示的 pixel 去用了，但時常會出現差一點的情況，但是也不知道為什麼。只好不斷的 generate bitstream 去看到底差了那一些，也是最辛苦費時的地方。然後最崩潰的還是最後顯示數字再左上角，一直掃不到正確的範圍，然後顯示又怪怪的，最後只好一氣之下直接用刻的。

●Conclusion

這一次是兩個人一起做了這個 final project 雖然耗費了不少時間，但成品做出來的時候，可說是有十足的成就感，當然我們還是有些不夠完美的地方，像是掃的圖案沒有去背，導致遊戲圖片會有重疊的地方，老師說其實只要再掃圖的時候再增加幾個條件，像是白色的地方不要掃，這樣就能達到去背的效果了。但這也算是寶貴的一次經驗，沒有實際的花時間去完成一個 project 大概學期一過，我們便會忘了一大半吧，我覺得做 final project 的難度比期末考還要更有挑戰性，畢竟這個是做越多越完整變越好，每個人都能發揮出自己的創意，但後果就是得為自己開出來的支票負責，拚了命也要把它完成，不然就跟某個市長一樣了。而兩人的分工也創造出一加一大於二的優勢，因為我們要用 AI 繪圖，但是兩個人其實都沒用過，所以我們就一個專心繪圖趕緊上手 AI 的基本操作，另一個打 code 修好以及 debug。最後不懂或有問題的地方也能互相問，這樣使整個進度快了許多。也感謝這學期老師及助教們的幫忙，沒有你們就沒有現在的我們，萬分感謝，不勝感激。