Cover Page

Team28

Member List

> 108071003 李彥璋
> 108070038 簡志宇

Contents

Member Contribution

> Idea 李彥璋
> Implementation 李彥璋、簡志宇
> report 李彥璋、簡志宇

Introduction

> 發想
> 遊戲內容
> 遊玩方法

發想

Proposal 原本計畫要做西洋棋，我們在考量時間的壓力與衡量自身能力過後，便果斷放棄。

期間我們一直在思考——什麼樣的替代遊戲，適合在 final project 的舞台上呈現。而因為西洋棋遊戲本與 vga 螢幕息息相關，所以我們的思索方向就朝著螢幕邁進。

在看到螢幕課程的 demo1 範例後，我們想起了以前曾經玩過的「色彩辨識」的遊戲。經過多方考量，我們認為這樣的遊戲不但能結合許多 Logic Design 課堂上教過的裝置，並且趣味性十足。「color？color！」因而誕生。

遊戲內容

「color？color！」是一個訓練眼力的小遊戲。機器會生成三十六個矩形，其中三十五個為「相同」顏色；剩下一個則為「相近」顏色，玩家的任務便是找出那一個與眾不同的顏色。

透過這個遊戲，玩家不僅能夠獲得滿滿的成就感（原來我的眼力還是不錯的嘛！），而在這中冷無賴之際，更能享受優越的成就感，進而為這百無聊賴的生活增添小小的趣味！

遊玩方法

遊戲時間限制60秒，當玩家拉起switch，開始計時。

畫面中36個矩形，只有1個顏色不同。

玩家的起始位置（小黑點）為最左上角的矩形，透過鍵盤W、A、S、D控制位置上、左、下、右的移動。按下space（空白鍵），代表確認選擇該矩形，只有當選擇到正確的（顏色不同）那一個矩形，才能進入下一關。

當時間耗盡，畫面會以閃爍的方式提醒玩家已不能再移動。

Design

> 設計概念
>
> 架構細節及方塊圖與分工
>
> 實作完成度/難易度說明
>
> 測試完整度

設計概念

- 遊戲呈現與操作：運用 Keyboard 操作指標，並且透過 VGA 顯示在螢幕上，timer 的部分則運用 7-segment 來呈現。拉下 Switch 0 暫停同時計時停止，拉起則繼續。
- 顏色設定：上網查詢顏色 RGB 代碼，挑選出多種顏色，然後對各個顏色人工進行微調出相近的顏色。遊戲進行時則透過 random 的數字隨機送出 1 對顏色。
- 答案色塊位置設定：透過 random 的數字選出 36 格色塊中的任一個。
- 游標位置設定：開始遊戲時初始位置設定於左上角，只要持續追蹤玩家的游標位置並在 pixel 輸出時處理即可。

架構細節及方塊圖與分工

- 主要架構分支
  - Top
    - vga controller — 李
    - pixel generator — 李/簡
    - timer and 7-segment display — 簡
    - random generator — 簡
    - keyboard decoder and control design — 李/簡
    - clock divider * 3 — 簡

- vga controller

- import vga_controller ：與 hardware device 的接口，用來控制 vga 螢幕輸出，直接使用 lab 中助教提供的 code，不再贅述其內容。

```
vga_controller vga_inst(
        .pclk(clk_25MHz),
        .reset(rst),
        .hsync(hsync),
        .vsync(vsync),
        .valid(valid),
        .h_cnt(h_cnt),
        .v_cnt(v_cnt)
    );
```

- pixel generator
  - import module：根據位置判斷螢幕顯示的顏色區塊。

```
pixel_gen pixel_gen_inst(
        .clk(clk),
        .h_cnt(h_cnt),
        .v_cnt(v_cnt),
        .valid(valid),
        .random(random), //有改
        .rand_color(rand_color), //有改
        .norm_color(norm_color), //有改
        .level(level), //有改
        .x_p(x_p),
        .y_p(y_p),
        .count(count),
        .vgaRed(vgaRed),
        .vgaGreen(vgaGreen),
        .vgaBlue(vgaBlue)
    );
```

  - module 實作：

    h_center 為游標的水平座標，**v_center** 為游標的垂直座標。
    首先判斷 **valid**，vga 顯示是否為可用狀態。而若遊戲已經結束 (**count** == 0) 了則要閃爍。再來判斷 **h_center, v_center** 的值，因為游標處必須為黑色。最後才判斷色塊位置以及其之間的間隔 (這個部分我們是額外寫一個 C++ program 讓他自動產生 code)。這邊值得注意的是，**random** 代表的是目前答案色塊為第幾塊，等同於一個 index，以此來標記並且辨識該色塊。
    p.s.原本我們預計要實作 multilevel 的關卡，也就是過一段時間難度就會加深，方塊數量從原本的 2*2 慢慢加到 6*6，但礙於時間有限，我們無法 de 出 bug，造成整個遊戲性大幅下降，實在可惜！！！

```
module pixel_gen(
   input clk,
   input [9:0] h_cnt,
   input [9:0] v_cnt,
   input valid,
   input [6:0] random, //有改
   input [11:0] rand_color, //有改
   input [11:0] norm_color, //有改
   input [2:0] level, //有改
   input [2:0] x_p, //有改
   input [2:0] y_p, //有改
   input [9:0] count,
   output reg [3:0] vgaRed,
   output reg [3:0] vgaGreen,
   output reg [3:0] vgaBlue
   );

   wire [9:0] h_center;
   wire [9:0] v_center;

    assign h_center = (x_p) * (640/level) + (320/level);
    assign v_center = (y_p) * (480/level) + (240/level);
    clock_divider3 #(.n(28)) cd28(clk, clk28);

    always @(*) begin
        if(!valid || (clk28 == 1 && count == 0)) begin
            {vgaRed, vgaGreen, vgaBlue} = 0;
        end
        else if( ((h_center-8 <= h_cnt) && (h_cnt < h_center+8)) && ((v_cente
r-8 <= v_cnt) && (v_cnt < v_center+8)) ) begin
            {vgaRed, vgaGreen, vgaBlue} = 0;
        end
        else if(level == 3'd2) begin
            if( ((10<=h_cnt) && (h_cnt<310)) && ((10<=v_cnt) && (v_cnt<230))
 ) begin
                {vgaRed, vgaGreen, vgaBlue} = ((random == 0) ? rand_color : n
orm_color);
            end
            else if( ((10<=h_cnt) && (h_cnt<310)) && ((250<=v_cnt) && (v_cnt<
470)) ) begin
                {vgaRed, vgaGreen, vgaBlue} = ((random == 2) ? rand_color : n
orm_color);
            end
            else if( ((330<=h_cnt) && (h_cnt<630)) && ((10<=v_cnt) && (v_cnt<
230)) ) begin
                {vgaRed, vgaGreen, vgaBlue} = ((random == 1) ? rand_color : n
orm_color);
            end
            else if( ((330<=h_cnt) && (h_cnt<630)) && ((250<=v_cnt) && (v_cnt
<470)) ) begin
                {vgaRed, vgaGreen, vgaBlue} = ((random == 3) ? rand_color : n
orm_color);
            end
            else begin
                {vgaRed, vgaGreen, vgaBlue} = 0;
            end
```

```
        end
        else if(level == 3'd3) begin
        // 中間省略
        ........
        end
        else if(level == 3'd6) begin
            if( ((10<=h_cnt) && (h_cnt<96)) && ((10<=v_cnt) && (v_cnt<70)) )
 begin
                {vgaRed, vgaGreen, vgaBlue} = ((random == 0) ? rand_color : n
orm_color);
            end
            else if( ((10<=h_cnt) && (h_cnt<96)) && ((90<=v_cnt) && (v_cnt<15
0)) ) begin
                {vgaRed, vgaGreen, vgaBlue} = ((random == 6) ? rand_color : n
orm_color);
            end
        // 中間省略
        .......
            else if( ((540<=h_cnt) && (h_cnt<626)) && ((330<=v_cnt) && (v_cnt
<390)) ) begin
                {vgaRed, vgaGreen, vgaBlue} = ((random == 29) ? rand_color :
 norm_color);
            end
            else if( ((540<=h_cnt) && (h_cnt<626)) && ((410<=v_cnt) && (v_cnt
<470)) ) begin
                {vgaRed, vgaGreen, vgaBlue} = ((random == 35) ? rand_color :
 norm_color);
            end
            else begin
                {vgaRed, vgaGreen, vgaBlue} = 0;
            end
        end
        else begin
            {vgaRed, vgaGreen, vgaBlue} = 0;
        end
   end
endmodule
```

- timer and 7-segment display
  - import module：計時 60 秒鐘並用 7-segment 顯示數值。

```
timer timer_display(
        clk,
        rst,
        start,
        count,
        DIGIT,
        DISPLAY
    );
```

  - module 實作：用 0.1 秒的 clock 來驅動一個 counter，**count** 代表的是目前 7-segment 的counter 數值，而 7-segment 的 **value0~2** 則分別表示第 0~2 位數字,

用簡單的除法與 modulo 運算可得。至於 **start** 則用來控制是否進行倒數。

```
module timer(
    input clk,
    input rst,
    input start,
    output reg [9:0] count,
    output reg [3:0] DIGIT,
    output reg [6:0] DISPLAY
    );

    reg [1:0] d;
    //reg [9:0] count;
    reg [3:0] value0, value1, value2;

    decimal_clock_divider cd100ms(clk, clk100ms);
    clock_divider3 #(.n(12)) cd12(clk, clk12);


    always @(posedge clk100ms or posedge rst) begin
        if(rst == 1) begin
            count <= 600;
        end else if(start == 0 || count == 0) begin
            count <= count;
        end else begin
            count <= count -1;
        end
    end
    always @* begin
        value0 = count % 10;
        value1 = (count / 10) % 10;
        value2 = count / 100;
    end

    always @(posedge clk12) begin
        case (DIGIT)
            4'b1110: begin
                d = 1;
                DIGIT = 4'b1101;
            end
            4'b1101: begin
                d = 2;
                DIGIT = 4'b1011;
            end
            4'b1011: begin
                d = 3;
                DIGIT = 4'b0111;
            end
            4'b0111: begin
                d = 0;
                DIGIT = 4'b1110;
            end
            default: begin
                d = 0;
                DIGIT = 4'b1101;
            end
```

```
        endcase
    end
    always @* begin
        case (d)
            0: begin
                case (value0)
                    4'd0: DISPLAY = 7'b100_0000;
                    4'd1: DISPLAY = 7'b111_1001;
                    4'd2: DISPLAY = 7'b010_0100;
                    4'd3: DISPLAY = 7'b011_0000;
                    4'd4: DISPLAY = 7'b001_1001;
                    4'd5: DISPLAY = 7'b001_0010;
                    4'd6: DISPLAY = 7'b000_0010;
                    4'd7: DISPLAY = 7'b111_1000;
                    4'd8: DISPLAY = 7'b000_0000;
                    4'd9: DISPLAY = 7'b001_0000;
                    default: DISPLAY = 7'b111_1111;
                endcase
            end
            1: begin
                case (value1)
                    4'd0: DISPLAY = 7'b100_0000;
                    4'd1: DISPLAY = 7'b111_1001;
                    4'd2: DISPLAY = 7'b010_0100;
                    4'd3: DISPLAY = 7'b011_0000;
                    4'd4: DISPLAY = 7'b001_1001;
                    4'd5: DISPLAY = 7'b001_0010;
                    4'd6: DISPLAY = 7'b000_0010;
                    4'd7: DISPLAY = 7'b111_1000;
                    4'd8: DISPLAY = 7'b000_0000;
                    4'd9: DISPLAY = 7'b001_0000;
                    default: DISPLAY = 7'b111_1111;
                endcase
            end
            2: begin
                case (value2)
                    4'd0: DISPLAY = 7'b100_0000;
                    4'd1: DISPLAY = 7'b111_1001;
                    4'd2: DISPLAY = 7'b010_0100;
                    4'd3: DISPLAY = 7'b011_0000;
                    4'd4: DISPLAY = 7'b001_1001;
                    4'd5: DISPLAY = 7'b001_0010;
                    4'd6: DISPLAY = 7'b000_0010;
                    4'd7: DISPLAY = 7'b111_1000;
                    4'd8: DISPLAY = 7'b000_0000;
                    4'd9: DISPLAY = 7'b001_0000;
                    default: DISPLAY = 7'b111_1111;
                endcase
            end
            3: begin
                DISPLAY = 7'b100_0000;
            end
            default: DISPLAY = 7'b111_1111;
        endcase
    end
endmodule
```

- random generator
  - import module：產生類似 random 的數值 (**rrr**) 以實現各種功能。

```
LFSR rand_gen(
        clk,
        rst,
        rrr
    );
```

  - module 實作：原本是打算使用 **16-bit 的 LFSR** 來製造 pseudo random 的效果，但因為某個原因 (在**遇到的困難**詳細說明) 決定放棄，於是改直接使用 clk 的值。經過測試發現其實也能做出類似 random 的感覺。

```
module LFSR(
    input wire clk,
    input wire rst,
    output reg [6:0] random
    );

    //always @(posedge clk or posedge rst)
      //  begin if (rst == 1'b1) begin
        //    random[15:0] <= 16'b1010110011100001;
            //led <= 0;
        //end else /*if(enter == 1)*/ begin
        //    random[14:0] <= random[15:1];
        //    random[15] <= random[5] ^ (random[3] ^ (random[2] ^ random
[0]));
            //led <= random;
        //end else begin
        //    random <= random;
        //end
    //end
    always @(posedge clk) begin
        random <= random + 1;
    end
endmodule
```

- keyboard decoder and control design
  - import keyboard decoder：與 hardware device 的接口，直接使用 lab 中助教提供的 code，不再贅述其內容。

```
KeyboardDecoder key_de (
    .key_down(key_down),
    .last_change(last_change),
    .key_valid(been_ready),
    .PS2_DATA(PS2_DATA),
    .PS2_CLK(PS2_CLK),
```

```
    .rst(rst),
    .clk(clk)
  );
```

- 實作寫在 top module 裡不用 import：
  - 這個 Sequential block 處理鍵盤訊號以及其與各式功能所交織出的變化。**x_p**, **y_p** 分別記錄目前游標在水平與垂直方向第幾塊，**level** 代表現在的方塊數量開根號，**enter** 則用來記錄可否進入下一關。
    用 **been_ready** 判斷鍵盤是否處於可使用狀態，而 **key_down[last_change]** 則代表哪個按鍵被按下了。值得注意的是還必須符合以下兩個條件才能操作鍵盤：
    1. **count** != 0：表示玩家**還有時間**
    2. **start** == 1：表示玩家**不處在 pause** 的狀態
  - 接下來就判斷 **key_num** (由 **last_change** 決定) 的值以做出相應動作：
    1. **key_num** == 4’b0000 or 4’b0001 or 4’b0010 or 4’b0011：**A, W, S, D** 鍵被按下，要控制方向，處理 **x_p**, **y_p**。
    2. **key_num** == 4’b0100：SPACE 鍵被按下，玩家選定了某塊它認為是答案的色塊(可由 **level** 搭配 **x_p**, **y_p** 算出)，因此我們要將其與真正答案色塊的位置 (**random**) 比對，來確認是否正確，正確 **enter** 就設為 1 表示可進入下一關。

```
always @ (*) begin
    case (last_change)
        KEY_CODES[00] : key_num = 4'b0000;
        KEY_CODES[01] : key_num = 4'b0001;
        KEY_CODES[02] : key_num = 4'b0010;
        KEY_CODES[03] : key_num = 4'b0011;
        KEY_CODES[04] : key_num = 4'b0100;
        //KEY_CODES[05] : key_num = 4'b0101;
        default: key_num = 4'b1111;
    endcase
end

always @(posedge clk or posedge rst) begin
     if(rst == 1) begin
         level <= 3'd6;
         flag <= 1'd0;
         x_p <= 3'd0;
         y_p <= 3'd0;
         enter <= 0;
     end
     else begin
         if(been_ready && key_down[last_change] == 1'b1 && count != 0 && star
t == 1) begin
```

```
            if(key_num != 4'b1111) begin
                if(key_num == 4'b0000) begin //左
                    if(x_p > 3'd0) x_p <= x_p - 1;
                end
                else if(key_num == 4'b0001) begin //上
                    if(y_p > 3'd0) y_p <= y_p - 1;
                end
                else if(key_num == 4'b0010) begin //下
                    if(level == 3'd2) begin
                        if(y_p < 3'd1) y_p <= y_p + 1;
                    end
                    else if(level == 3'd3) begin
                        if(y_p < 3'd2) y_p <= y_p + 1;
                    end
                    else if(level == 3'd4) begin
                        if(y_p < 3'd3) y_p <= y_p + 1;
                    end
                    else if(level == 3'd5) begin
                        if(y_p < 3'd4) y_p <= y_p + 1;
                    end
                    else if(level == 3'd6) begin
                        if(y_p < 3'd5) y_p <= y_p + 1;
                    end
                end
                else if(key_num == 4'b0011) begin //右
                    if(level == 3'd2) begin
                        if(x_p < 3'd1) x_p <= x_p + 1;
                    end
                    else if(level == 3'd3) begin
                        if(x_p < 3'd2) x_p <= x_p + 1;
                    end
                    else if(level == 3'd4) begin
                        if(x_p < 3'd3) x_p <= x_p + 1;
                    end
                    else if(level == 3'd5) begin
                        if(x_p < 3'd4) x_p <= x_p + 1;
                    end
                    else if(level == 3'd6) begin
                        if(x_p < 3'd5) x_p <= x_p + 1;
                    end
                end
                else if(key_num == 4'b0100) begin //enter
                    if(flag == 0) begin
                        flag <= 1'b1;
                        enter <= 1'b1;
                        r <= rrr;
                    end else if( (y_p*(level)) + x_p == random /*flag == 1
*/) begin
                          enter <= 1'b1;
                          r <= rrr;
                    end
                    else begin
                        enter <= 0;
                        r <= r;
                    end
                end
            end
          end else begin
```

```
            enter <= 0;
            x_p <= x_p;
            y_p <= y_p;
            flag <= flag;
        end
    end
end
```

- clock divider * 3：分別為週期 0.1 秒、頻率 25MHz 和除頻數為 2 的次方之 clock divider。
  - 0.1 秒

```
module decimal_clock_divider(
    input clk,
    output clock_100ms
    );
    reg c = 1;
    reg [23:0] cnt;
    wire [23:0] cnt_next;
    //initial c = 1;
    always @(posedge clk) begin
        cnt <= cnt_next;
        if(cnt == 4999999) c <= ~c;
        else c <= c;
    end

    assign cnt_next = ((cnt == 4999999) ? 0 : cnt + 1);
    assign clock_100ms = c;
endmodule
```

  - 25MHz

```
module clock_divider(clk1, clk);
input clk;
output clk1;

reg [1:0] num;
wire [1:0] next_num;

always @(posedge clk) begin
  num <= next_num;
end

assign next_num = num + 1'b1;
assign clk1 = num[1];

endmodule
```

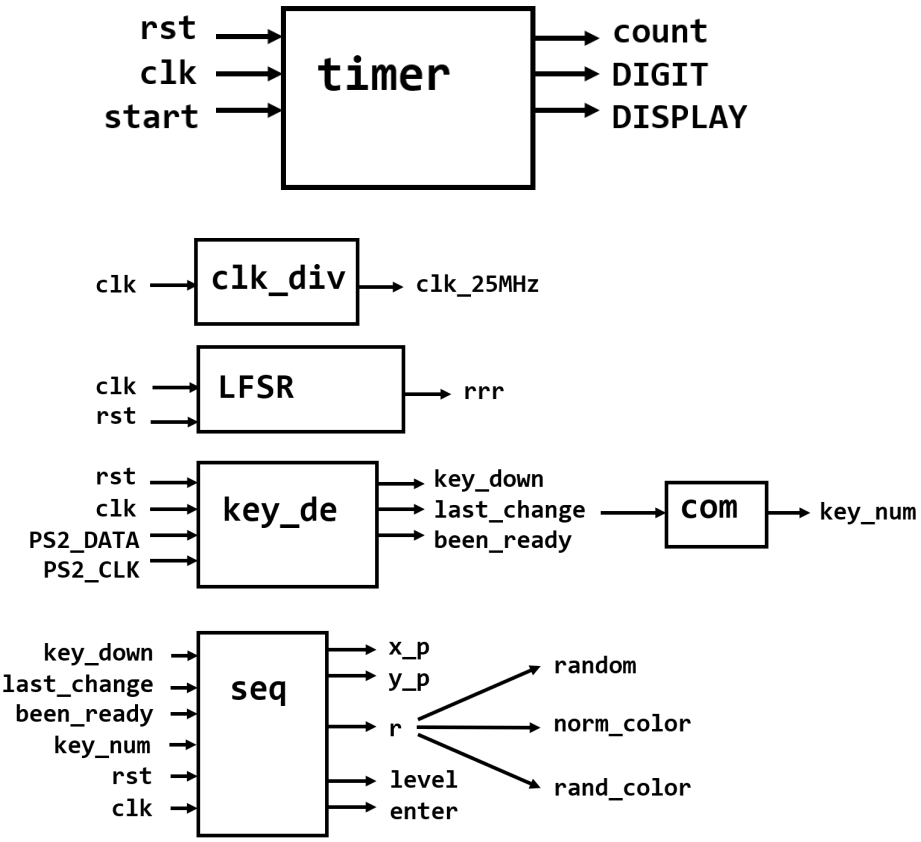
  - 2^x

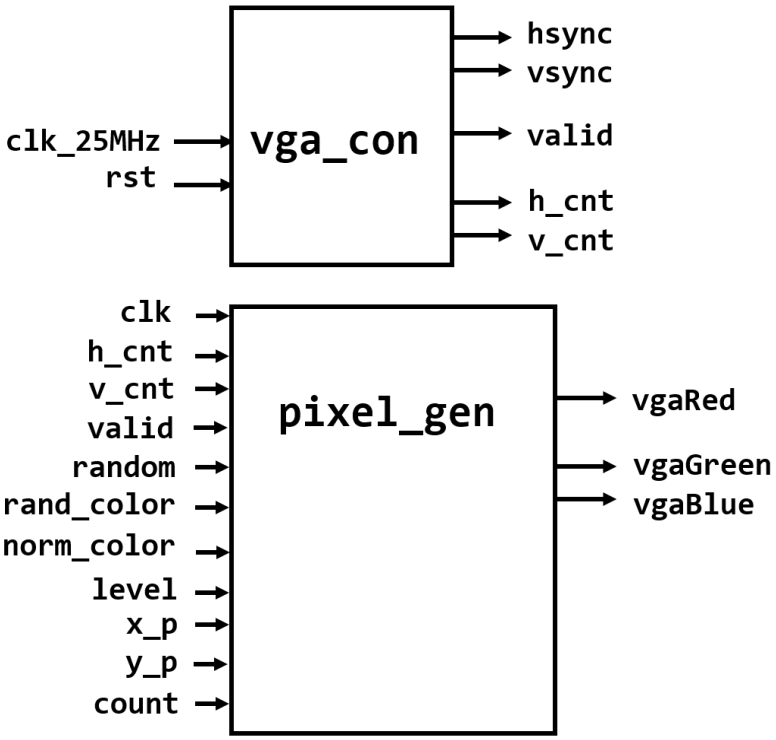
```
module clock_divider3 #(parameter n=25) (
    input clk,
    output clk_div);
    // add your design here
    reg [n-1:0] cnt, cnt_next;
    always @(posedge clk) begin
        cnt <= cnt_next;
    end

    always @* begin
        cnt_next = cnt + 1;
    end

    initial begin
        cnt = 1;
    end
    assign clk_div = cnt[n-1];
endmodule
```

- 方塊圖





實作完成度/難易度說明

- 完成度

  實作完成度大約 **70%**，畢竟原本是想做**難度不一的關卡**，結果卻未完成，除此之外我們和原本預想的目標其實差得不遠，甚至還有多出一些功能，例如：原本不想提供玩家 **pause** 的功能，畢竟這是一個考驗性質的遊戲，要有些挑戰性才行；或是畫面閃爍以示 game over，原本是想說以一張大圖寫個 **“YOU LOSE” or ”YOU WIN”** 來呈現，但我們認為這樣**頗為單調，沒有互動**的感覺，因此改為現行的表現手法，更增動態感。

- 難易度

  老實講這樣的 project 內容的確**稍嫌簡單**，但我們兩個這學期的課業 loading 都頗為繁重，並且各個科目都有意想不到的額外作業 or 變卦，讓我們猝不及防。也許是因為我們的**應變能力不足**，無法妥善安排各項事務做好時間管理；抑或是 **coding 能力低落**，效率不佳，導致完成度無法接近 100%。

  原先我們想做的**西洋棋**功能較為**繁雜**，且在設計的架構上也必須更為完整。雖然實作單一功能方面可能較為簡單，但整體下來還是有難度的。

測試完整度

測試狀況

因為我們的**主題圍繞著螢幕**，測試起來較為不方便（我們都沒有自己的螢幕），所以我們在寫final project的期間，都待在資電館寸步不離。

測試方法與Lab7-2相同，以vga線連接螢幕與FPGA板、鍵盤接上FPGA板的USB插槽、FPGA板以USB cable連到電腦，經過**synthesis**、**implementation**、**generate bit**之後，vivado就可以run program了！

而在測試的過程當中，我們也遇到了**不少問題**。除了一般的synthesis failed之外，也有全部的步驟都跑完了，在螢幕上卻無法顯示畫面的狀況，相當考驗我們debug的能力！

完整度

作為一個遊戲來說，我們的完整度**稍嫌不足**。我們以一個遊戲可能需要包含的要素來說明，像是**主畫面**（進入遊戲前的主選單）、**UI介面**（遊戲當中與玩家互動，如得到分數）、**遊戲循環**（遊戲結束自動跳回主畫面）……等，都是我們**缺乏但可以做得更好**的。

Difficulties & Feedback

> 困難與解決辦法
>
> 心得討論

困難與解決辦法

3 problems

vga 顏色顯示

當初構思這個遊戲的時候想得太完美了，在調配各種顏色的時候，赫然想起 **vga 的顏色只有 12 bits**，和一般我們常見的 24 bits 大相逕庭！我們一開始不知道該如何解決這個問題，後來想想可以直接將本來的 R、G、B 分別除以 16，做一個完美的 transformation！

Random

原本使用 LFSR 來產生 pseudo random 數沒什麼問題，但在撰寫 **level upgrade** 時處處碰壁，無法達到完美的增加難度的 multilevel 關卡，因此就想說較可能出問題的 block 簡化看看。而使用 clk 作為 random 的基數在我們實測過後其實效果不算太差，在實際跑遊戲時基本上是看不出甚麼特別的順序的，所以最後我們就沒有改回 LFSR 了。

Level upgrade問題

level 的變化可謂這個遊戲的重點之一，能讓這遊戲更有漸進式的風味。我們在實作這個部分時碰到了問題，一直因為**不明原因而 Synthesis Failed**，也因此花了將近半天在 debug 上，最後我們得出的結論是因為我們使用 **level 來決定其他變數，同時又用了其他變數搭配 level 來判斷是否進入下一關卡**，所以才會無法合成。這雖然是個常見的錯誤，但當時可能有些疲憊，無法順利發現問題所在，直到 demo 前不久才找到，礙於時間緊迫且短時間無法修復，除放棄以外別無他法。

至於解決方案我認為改變關卡變動的方式是為一個可行的方法，例如用**時間**來決定關卡，像過了 10 秒鐘後讓關卡前進的變數設為 1，並在下一次玩家按下 SPACE 時改變方塊數量；又或是使用**分數制**，當分數達到一定量時就能進入下一難度。

心得討論

做完final project並回顧一整個學期的點點滴滴，赫然發現我們已經是**收穫滿滿**。從一開始邏輯設計課程，教導偏理論的部分；到邏輯設計實驗課程，自己動手實作，也經歷了兩個學期的時光，期間我們也都樂在其中、沉浸在0101的世界。

於final project，我們結合了課堂上所學——**7-segmet**、**鍵盤**、**vga螢幕**，我們認為其豐富度是相當足夠的。但我們同樣也惋惜著，沒有機會自己去探索課堂以外的其他東西，沒有機會做出想像中的作品，沒有機會實現心中的藍圖。當天在demo現場我們也欣賞了其他組別的作品，更因此知曉了「**作品只有更豐富沒有最豐富**」。

而除了學術方面，final project更是讓我們體會到「**時間管理**」的重要性，學期接近末尾之際老師常常放一個北極熊的梗圖（越接近deadline的狀況），而我們也都只是笑笑而已，到了後來才發現真的是**寸金難買寸光陰**（即使如此我們也已經do our best）。

最後，也謝謝老師以及全體助教這一整個學期的付出與辛勞、不厭其煩地為大家解惑，使得我們的學習之路更為平坦、順遂！