Final Project

學號: 109062173 姓名: 葉昱揚

1. 設計概念

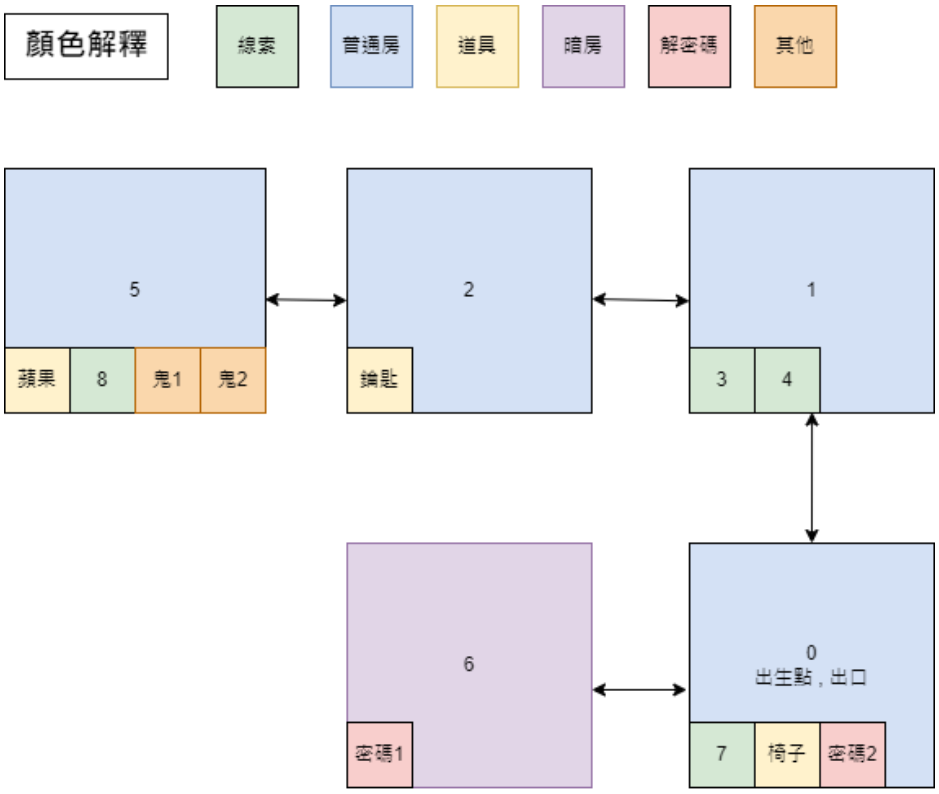

圖 1 (地圖設計)

目的：

破解大門密碼鎖，逃離密室

過程：

主要地圖有 5 張，分別為上圖的場景 1~5。

各個場景藏有道具、線索、鬼，玩家要探索地圖，尋找物品、挖掘隱藏線索、奪取鬼守護的道具，破解大門密碼最終逃離密室。

2. 架構細節及方塊圖

地圖轉換 FSM

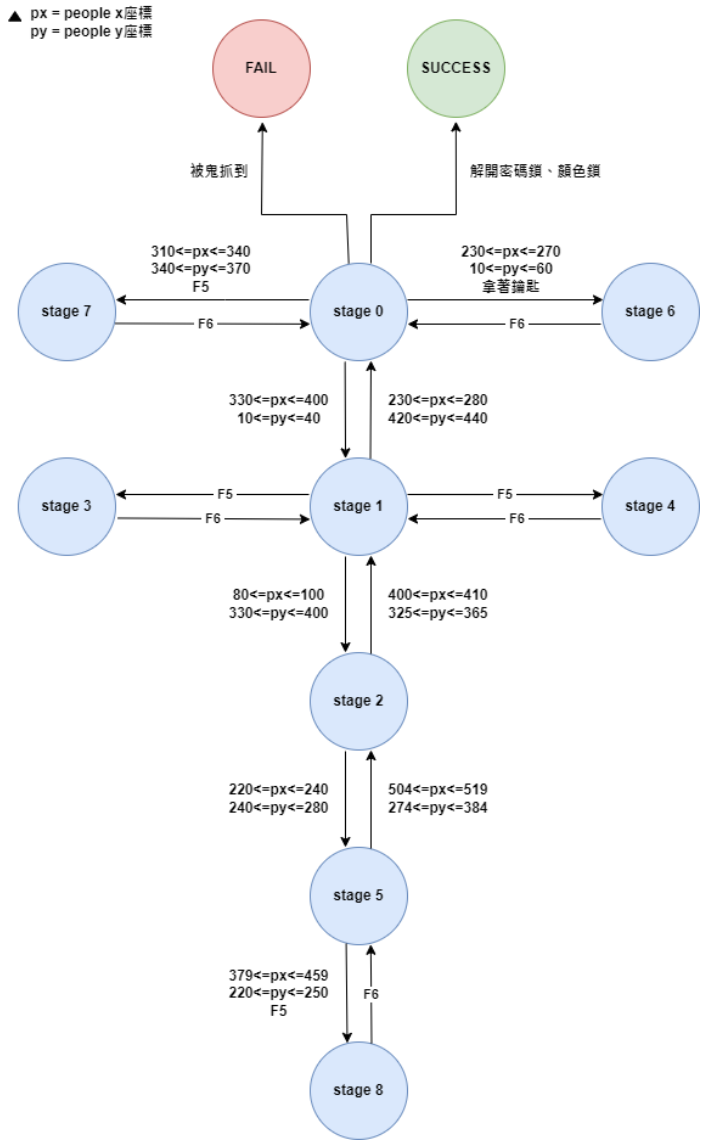

圖 2

地圖轉換依靠以下兩點轉換: (1)角色的 x、y 座標 (2)當前地圖 state

每當角色走到上圖標示的 x、y，地圖就會轉換成別張。

闖關失敗的條件為在 stage 5 的時候被鬼抓到。闖關成功的條件為解開密碼鎖與顏色鎖。

椅子地圖轉換

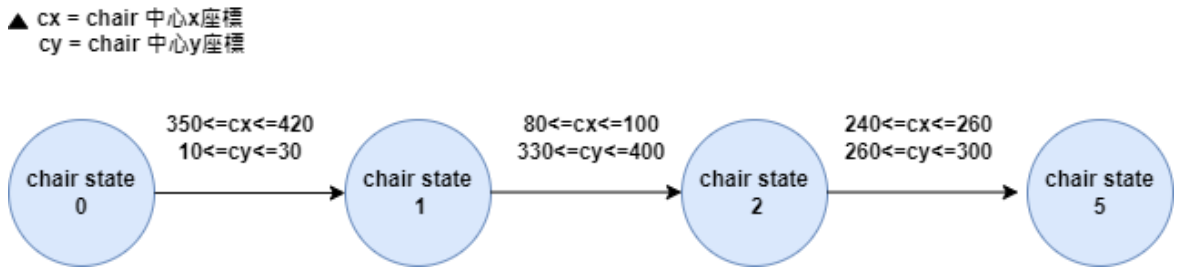

圖 3

椅子是可以推動的！

目前的設計是椅子只能夠單向推動，也就是只能往地圖前面繼續推進，無法往回推。

Block diagram

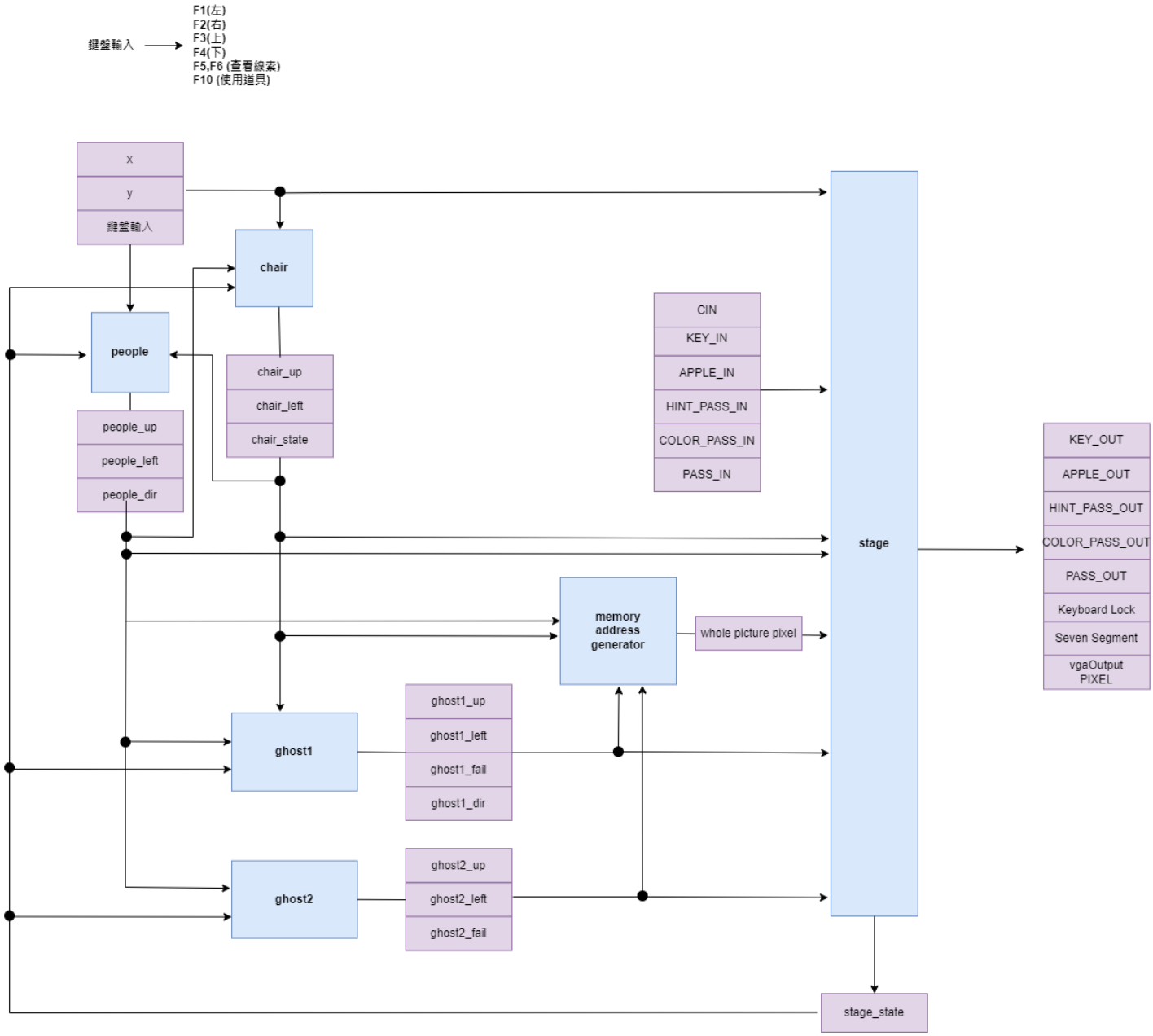

圖 4

Project 中有五個最重要的 module:

1. people -> 角色移動、碰撞
2. chair -> 椅子移動、碰撞
3. ghost1 -> 鬼 1 移動、碰撞
4. ghost2 -> 鬼 2 移動、碰撞
5. memory address generator -> 得到圖片 pixel
6. stage -> 地圖轉換、道具互動

people、chair、ghost1、ghost2 都會把 output 送到 memory address generator 內，得出存在 BRAM 裡的 pixel，其餘固定位置的圖片同理。最後把 5 個 module 的 output 接給 stage module，stage 會根據所有 input 資訊，output 出現階段的地圖，並把人物、椅子、鬼物、道具等等所有物品展示出來。

People Code Explanations

這個 module 負責操控角色移動。

```
// 往上
if(key_down[`F3]) begin
    next_people_up = people_up-2;
    next_dir = dir;
end

// 往下
if(key_down[`F4]) begin
    next_people_up = people_up+2;
    next_dir = dir;
end

// 往左
if(key_down[`F1]) begin
    next_people_left = people_left-2;
    next_people_up = people_up;
    next_dir = `LEFT_DIR;
end

// 往右
if(key_down[`F2]) begin
    next_people_left = people_left+2;
    next_people_up = people_up;
    next_dir = `RIGHT_DIR;
end
```

圖 5 角色上下左右移動

```
if(stage_state==0 && stage0_IL) begin
    // tp
    if(TP) begin
        people_left <= 320;
        people_up <= 220;
    end
    // 1 -> 0
    else if( 211<=people_left && people_left<=261 &&
             401<=people_up && people_up<=421) begin
        people_left <= 360;
        people_up <= 70;
    end

    // 6-> 0
    else if( 270<=people_left && people_left<=301 &&
             421<=people_up && people_up<=441) begin
        people_left <= 250;
        people_up <= 60;
    end

    // 7-> 0
    else if( 310<=people_left+19 && people_left+19<=340 &&
             340<=people_up+19 && people_up+19<=370) begin
        people_left <= next_people_left;
        people_up <= next_people_up;
    end

    dir <= next_dir;
    stage0_IL <= 0;
end
```

圖 6 轉換地圖時，位置初始化

如圖 5，我的操控方式是控制角色圖片的上邊界和左邊界:

people_up 為上邊界、people_left 為左邊界。
當按下 F1、F2、F3、F4 的時候，會改變對應邊界的值。
以上左圖而言，每按一次按鍵會 +-2 值。
推動效果與人物移動相同概念:

若往左移，people_left －2
若往右移，people_left ＋2
若往上移，people_up －2
若往下移，people_up ＋2

之後印圖片的時候再依據 people_left、people_up 為基點算出 address。
如此就能實現用鍵盤操控角色移動。

圖 6 是轉換地圖時，位置初始化的寫法:

以其他地圖轉換到第 0 張地圖為例，我會判斷地圖轉換前的角色位置，並寫好所有可能的判斷式，讓角色換關卡時會出現在該出現的位置。
比方說

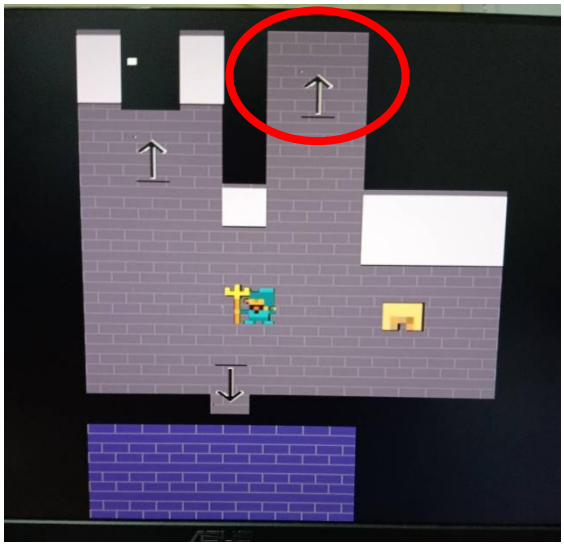

圖 7 地圖 0

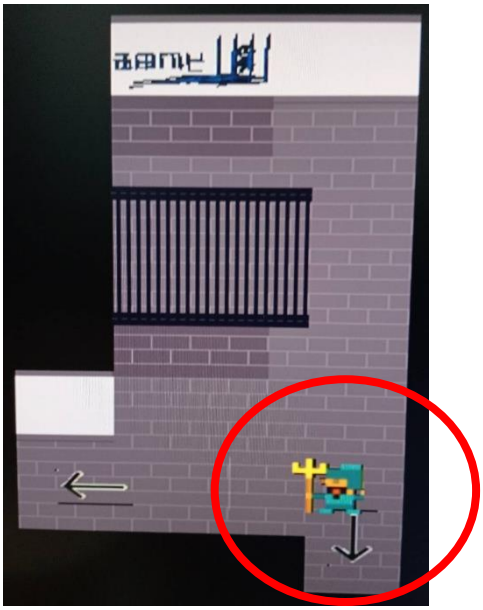

圖 8 地圖 1

當角色走到圖 7 的紅色圈圈時，他會被傳送到圖 8 的紅圈位置，而不是圖 8 的隨機一個位置。
其餘地圖的位置初始化同理。

Chair Code Explanations

這個 module 負責椅子的移動。
椅子移動需要依靠人力推動，也就是要判斷周圍是否有人在推動椅子。

```
// push UP_DIR
if(people_up+19-35<chair_up+39 && people_up+19>chair_up+39 && chair_left<people_left+19 && people_left+19<chair_left+39) next_chair_up = chair_up-5;
// push DOWN_DIR
else if(people_up+39+5 > chair_up && people_up<chair_up && chair_left<people_left+19 && people_left+19<chair_left+39) next_chair_up = chair_up+5;
else next_chair_up = chair_up;

// push LEFT_DIR
if(people_left-5 < chair_left+39 && people_left + 40 - 1>chair_left+39 && chair_up<people_up+19 && people_up+19<chair_up+39) next_chair_left = chair_left-5;
// push RIGHT_DIR
else if(people_left + 40 - 1+5 > chair_left && people_left<chair_left && chair_up<people_up+19&& people_up+19<chair_up+39) next_chair_left = chair_left+5;
else next_chair_left = chair_left;
```

圖 9 推椅子

圖 9 中，我寫了判斷式判斷椅子周圍有沒有人推動。

比方說，如果想要把椅子往上推動，需要檢查:

椅子 y 座標往下是否有人 && 人的 x 座標是否與椅子 x 座標重疊
滿足上面兩個條件代表有人在椅子正下方，這樣就達成推動的條件。

推動效果與人物移動相同概念:
若往左移，chair_left – N
若往右移，chair_left + N
若往上移，chair_up – N
若往下移，chair_up – N

```
// 0 -> 1
if(chair_IL1 && 350<=chair_left+20 && chair_left+20<=420 && 10<=chair_up+20 && chair_up+20<=30) begin
    chair_left <= 190;
    chair_up <= 350;
    chair_state <= 1;
    chair_IL1 <= 0;
end
```

圖 10　椅子轉換地圖時，位置初始化

遊戲中需要跨地圖推動椅子，因此椅子換地圖時也需要初始化位置。
避免換地圖時，椅子出現在奇怪的位置上 (地圖外、牆壁上等等)。

寫法與人的位置初始化同理，都是判斷從哪邊進入地圖，在給他對應的位置。

Ghost1 Code Explanations

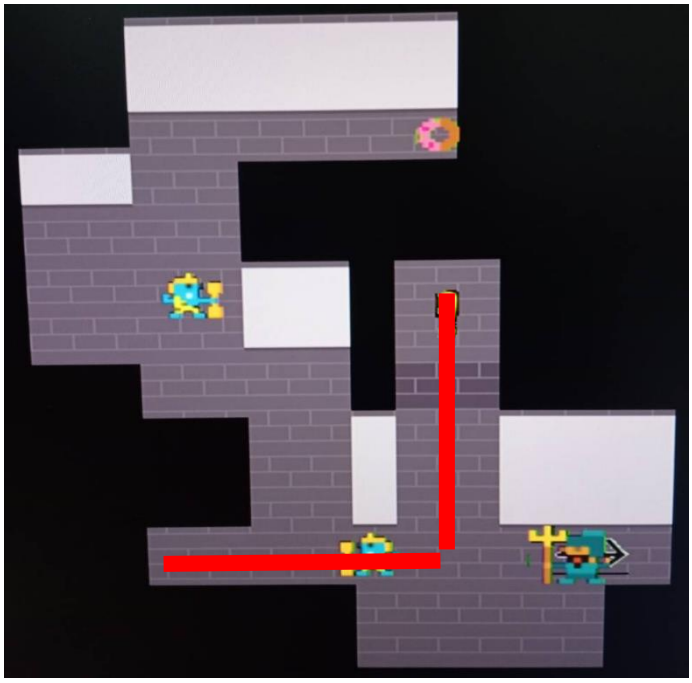

圖 11 紅線為鬼 1 的行動軌跡

圖 11 展示了鬼 1 的行動軌跡。
整個設計過程中我需要完成兩件事情 (1)鬼移動 (2)鬼轉向 (3)鬼速度

(1)鬼移動

```
always@(*) begin
    next_ghost_up = ghost_up;
    next_ghost_left = ghost_left;
    if(trigger) begin
        case(dir)
            `LEFT_DIR: next_ghost_left = ghost_left - 7;
            `RIGHT_DIR:next_ghost_left = ghost_left + 7;
            `UP_DIR:   next_ghost_up = ghost_up - 7;
            `DOWN_DIR: next_ghost_up = ghost_up + 7;
        endcase
    end
end
```

圖 12 鬼 1 移動

我給鬼 1 設計了四個轉向 LEFT、RIGHT、UP、DOWN。
如字面上的意思，當鬼 1 走向為 LEFT 時，鬼 1 的左邊界往左邊扣，其餘方向同理

另外需要設計 line4 的 trigger。因為 100MHz 的 clock frequency 太高，直接用 clock trigger 會快到肉眼看不出來，因此我設計了每 0.1 秒 trigger 一次的 design

```
always @(posedge clk) begin
    if (count == 24'd1000_0000 - 1) begin
        count <= 0;
        trigger <= 1;
    end else begin
        count <= count + 1;
        trigger <= 0;
    end
end
```

圖 13 每 0.1 秒 trigger 一次

如圖 13，100MHz 的 clk，當 counter 數到 1e7 的時候剛好經過 0.1 秒。這樣的設計能夠讓鬼 1 的移動呈現在畫面上，且不會快到肉眼看不出來。

(2) 鬼轉向

```
always@(*) begin
    // 撞牆反彈
    if(dir==`RIGHT_DIR && ghost_left>=370 ) next_dir=`UP_DIR;
    else if(dir==`UP_DIR && ghost_up<=165) next_dir=`DOWN_DIR;
    else if(dir==`DOWN_DIR && ghost_up>=330) next_dir=`LEFT_DIR;
    else if(dir==`LEFT_DIR && ghost_left<=250) next_dir=`RIGHT_DIR;
    //撞椅子反彈
    else if(chair_state==5 && dir==`RIGHT_DIR && ghost_left+30>=chair_left && ghost_left<=chair_left && chair_up<=ghost_up+15     && ghost_up+15<=chair_up+40) next_dir=`LEFT_DIR;
    else if(chair_state==5 && dir==`DOWN_DIR  && ghost_up+30 >=chair_up    && ghost_up<=chair_up     && chair_left<=ghost_left+15 && ghost_left+15<=chair_left+40) next_dir=`UP_DIR;
    else if(chair_state==5 && dir==`LEFT_DIR  && ghost_left<=chair_left+40 && ghost_left>=chair_left && chair_up<=ghost_up+15     && ghost_up+15<=chair_up+40) next_dir=`RIGHT_DIR;
    else if(chair_state==5 && dir==`UP_DIR    && ghost_up<=chair_up+40     && ghost_up>=chair_up     && chair_left<=ghost_left+15 && ghost_left+15<=chair_left+40) next_dir=`DOWN_DIR;
    else next_dir = dir;
end
```

圖 14 鬼 1 轉向

鬼 1 會撞牆反彈與撞椅子反彈。

撞牆反彈只需要判斷鬼 1 目前的 x 座標 or y 座標是否到達我設定好的邊界。

如果已經達到邊界則轉向，轉向流程為 Right -> Up -> Down -> Left -> Right，重複循環。

撞椅子反彈需要判斷鬼往前進的方向是否有椅子。

實際寫法為判斷鬼的左邊界、右邊界、上邊界、下邊界是否有椅子並撞到。

若往下時撞到椅子上方，鬼 1 反彈往上;

若往上時撞到椅子上方，鬼 1 反彈往下;

若往左時撞到椅子上方，鬼 1 反彈往右;

若往右時撞到椅子上方，鬼 1 反彈往左;

撞椅子反彈會與撞牆壁反彈配合，當鬼因為撞椅子反彈往左，又撞到左邊牆壁時，鬼 1 會再反彈往右。

亦即我們可以使用椅子把鬼 1 夾在一個神奇的夾角出不來，降低地圖困難度。

(3) 鬼速度

```
1  always@(*) begin
2      next_ghost_up = ghost_up;
3      next_ghost_left = ghost_left;
4      if(trigger) begin
5          case(dir)
6              `LEFT_DIR: next_ghost_left = ghost_left - 7;
7              `RIGHT_DIR:next_ghost_left = ghost_left + 7;
8              `UP_DIR:   next_ghost_up = ghost_up - 7;
9              `DOWN_DIR: next_ghost_up = ghost_up + 7;
10         endcase
11     end
12 end
```

圖 15 鬼速度設定

圖 15 的 line6~line9 後面的 +-7 即是速度設定，數字越大速度越快，反之。

Ghost2 Code Explanations

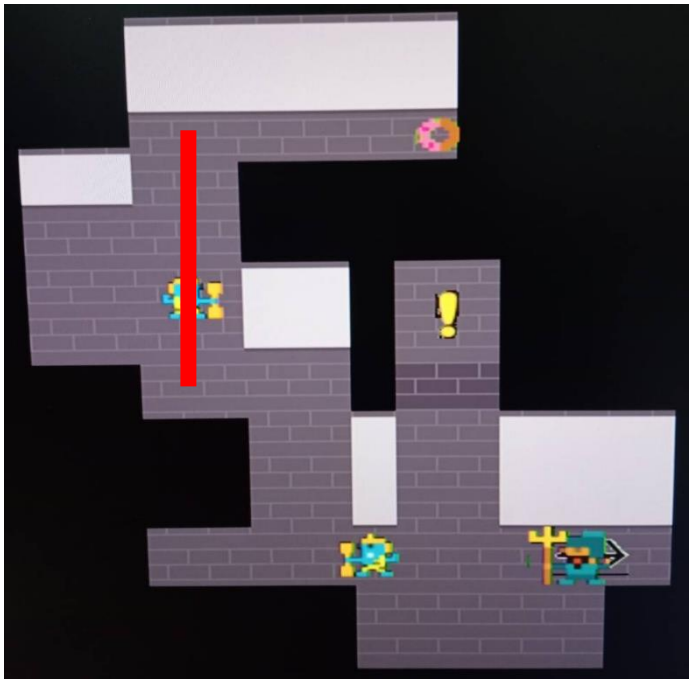

圖 16 鬼 2 移動路線

鬼 2 為鬼 1 弱化版本，只能上下移動且移速更慢。
由於**寫法與鬼 1 完全一樣**又更簡單，報告中不多贅述。

Memory Address Generator Code Explanations

這個 module 負責算出圖片的 address，並輸出 pixel

```
1  wire [9:0]ghost1_addr;
2  assign ghost1_addr = (ghost1_dir==`LEFT_DIR) ? (ghost1_left+29-(x-ghost1_left)-ghost1_left)+30*(y-ghost1_up) : (x-ghost1_left)+30*(y-ghost1_up);
3  ghost_w30h30    p6 (.clka(clk_25MHz), .wea(0), .addra( ghost1_addr ),                        .dina(garbage), .douta( ghost1_pixel   ));
4
5  wire [10:0] people_addr;
6  assign people_addr = (people_dir==`RIGHT_DIR) ? (x-people_left)+40*(y-people_up) : (people_left+39-(x-people_left)-people_left)+40*(y-people_up);
7  people_w40h40   p9 (.clka(clk_25MHz), .wea(0), .addra( people_addr ),                        .dina(garbage), .douta( people_pixel   ));
8
9  hint_w80h40     p2 (.clka(clk_25MHz), .wea(0), .addra( (x-130)+80*(y-60) ),                  .dina(garbage), .douta( hint_pixel     ));
10 carbinet_w35h35 p3 (.clka(clk_25MHz), .wea(0), .addra( ((x-330)>>1) + 35*(y-45>>1)  ),       .dina(garbage), .douta( carbinet_pixel ));
```

圖 17 Memory address gen

圖 17 展示了靜態的圖片和會移動的人、鬼 1 如何印製。

靜態圖片，line9 ~ 10:
(h_cnt - 設定好的 x 座標) + 寬度*(v_cnt – 設定好的 y 座標)

會移動的圖片，line3、line8，下面以人為例:
(h_cnt – 人的左邊界) + 人的寬度*(v_cnt – 人的上邊界)
這樣的好處就是圖片可以移動，h_cnt 和 v_cnt 都是以左、上邊界為 baseline，去計算出 address 並取出正確的 pixel 值。

Stage Code Explanations

(1) 地圖轉換

```
always@(*) begin

    else if(stage_state==0) begin
        if(331<=people_left+19 && people_left<=401 && 10<=people_up && people_up<=40) next_stage_state = 1;
        else if(KEY_OUT && 231<=people_left && people_left<=271 && 10<=people_up+19 && people_up<=61 && key_down[`F5]) next_stage_state = 6;
        else if(310<=people_left+19 && people_left+19<=340 && 340<=people_up+19 && people_up+19<=370 && key_down[`F5]) next_stage_state = 7;
        else next_stage_state = 0;
    end

    else if(stage_state==1) begin
        if(211<=people_left && people_left<=261 && 401<=people_up && people_up<=421) next_stage_state = 0;
        else if(61<=people_left && people_left<=81 && 311<=people_up && people_up <=381) next_stage_state = 2;
        else if(130<=people_left+19 && people_left+19<=210 && 81<=people_up && people_up <=121 && been_ready && key_down[`F5]) next_stage_state = 3;
        else if(130<=people_left+19 && people_left+19<=210 && 250<=people_up+19 && people_up+19 <=290 && been_ready && key_down[`F5]) next_stage_state = 4;
        else next_stage_state = 1;
    end

    else if(stage_state==2) begin
        if(381<=people_left && people_left<=391 && 306<=people_up && people_up<=346) next_stage_state = 1;
        else if(201<=people_left && people_left<=221 && 221<=people_up && people_up<=261) next_stage_state = 5;
        else next_stage_state = 2;
    end


    else if(stage_state==3) begin
        if(been_ready && key_down[`F6]) next_stage_state = 1;
        else next_stage_state = 3;
    end

    else if(stage_state==4) begin
        if(been_ready && key_down[`F6]) next_stage_state = 1;
        else next_stage_state = 4;
    end

    else if(stage_state==5) begin
        if(485<=people_left && people_left<=500 && 255<=people_up && people_up<=365) next_stage_state = 2;
        else if(370<=people_left && people_left<=440 && 220<=people_up+19 && people_up+19<=250 && been_ready && key_down[`F5]) next_stage_state = 8;
        else next_stage_state = 5;
    end
    else if(stage_state==6) begin
        if(270<=people_left && people_left<=301 && 421<=people_up && people_up<=441) next_stage_state = 0;
        else next_stage_state = 6;
    end
    else if(stage_state==7) begin
        if(been_ready && key_down[`F6]) next_stage_state = 0;
        else next_stage_state = 7;
    end
    else if(stage_state==8) begin
        if(been_ready && key_down[`F6]) next_stage_state = 5;
        else next_stage_state = 8;
    end
end
```

圖 18 state transition

可以參考最上方地圖 FSM，這邊即是把 FSM 圖寫成 code。
地圖轉換依靠以下兩點轉換: (1)角色的 x、y 座標 (2) 當前地圖 state
每當角色走到上圖標示的 x、y，state 轉換，地圖就會轉換成別張。

(2) 地圖畫法

手刻地圖

```
always@(*) begin
    if(!valid) {vgaR, vgaG, vgaB} = 12'h000;
    else begin
        case(stage_state)
            0:  begin
                {vgaR, vgaG, vgaB} = 12'h000;

                // floor1
                if(420<=x && x<=520 && 150<=y && y<=155) {vgaR, vgaG, vgaB} = `FLOOR_COLOR;
                if(220<=x && x<=250 && 10<=y && y<=15)   {vgaR, vgaG, vgaB} = `FLOOR_COLOR;
                if(290<=x && x<=320 && 10<=y && y<=15)   {vgaR, vgaG, vgaB} = `FLOOR_COLOR;
                if(320<=x && x<=350 && 145<=y && y<=150) {vgaR, vgaG, vgaB} = `FLOOR_COLOR;
                if(320<=x && x<=350 && 185<=y && y<=220) {vgaR, vgaG, vgaB} = `FLOOR_COLOR;
                if(220<=x && x<=320 && 75<=y && y<=220)  {vgaR, vgaG, vgaB} = `FLOOR_COLOR;
                if(350<=x && x<=420 && 10<=y && y<=220)  {vgaR, vgaG, vgaB} = `FLOOR_COLOR;
                if(220<=x && x<=520 && 220<=y && y<=350) {vgaR, vgaG, vgaB} = `FLOOR_COLOR;
                if(220<=x && x<=420 && 350<=y && y<=380) {vgaR, vgaG, vgaB} = `FLOOR_COLOR;

                // floor2
                if(220<=x && x<=420 && 380<=y && y<=480) {vgaR, vgaG, vgaB} = `PASSWORD_COLOR;

                if(220<=x && x<=520) if(y==10) {vgaR, vgaG, vgaB} = 12'h767;

                if(220<=x && x<=520) if(y==20) {vgaR, vgaG, vgaB} = 12'h767;

                // ........
            end
        endcase
    end
end
```

圖 19 (手刻地圖)

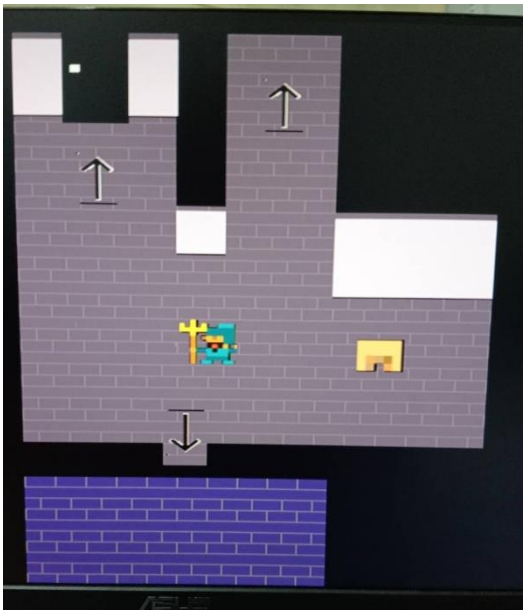

圖 20 (灰色、藍紫色地板)

由於 fpga 的資源不足，且網路上沒有我喜歡的地形圖片，我決定客製化、手刻地圖。

圖 19 是簡短的手刻地圖範例，我寫了非常多的 if 敘述判斷當前的 x (h_cnt) 、 y (v_cnt)是多少，當 x、y 是特定數字的時候就把那個區塊塗成某個顏色。圖 19 的 code 會畫出圖 20 的灰色地板和藍紫色地板。

其餘地區的地圖同理，都是判斷 x、y，切好區塊後畫成某種顏色，最終畫出全部的地圖。

疊層

由於會重疊的東西非常多。
因此我的畫法是越重要、越需要顯示的東西，code 放在越晚執行的地方。

```
// black block
if(220<=x && x<=260 && 10<=y && y<=175) {vgaR, vgaG, vgaB} = 12'h000;
if(220<=x && x<=260 && 280<=y && y<=365) {vgaR, vgaG, vgaB} = 12'h000;
if(400<=x && x<=410 && 10<=y && y<=250) {vgaR, vgaG, vgaB} = 12'h000;

// arrow 2 to 1
if(350<=x && x<=390 && 333<=y && y<=357 && arrow_2to1!=12'hFFF) {vgaR, vgaG, vgaB} = arrow_2to1;
// arrow 2 to 5
if(230<=x && x<=270 && 248<=y && y<=272 && arrow_2to5!=12'hFFF) {vgaR, vgaG, vgaB} = arrow_2to5;

// carbinet
if(330<=x && x<=400 && 45<=y && y<115) {vgaR, vgaG, vgaB} = carbinet_pixel;

// key
if(!KEY_OUT && 360<=x && x<=380 && 45<=y && y<=65 && key_pixel!=12'h000) {vgaR, vgaG, vgaB} = key_pixel;
```

圖 21 疊層寫法

圖 21 的 code 由上往下看，最上面的物體會先被畫出來，下面的物體因為比較晚被畫，會直接把下方的圖蓋過去。如圖 21 之中，key 會蓋在 carbinet 上，carbinet 會蓋在 black block 上。

去背

有很多物件是方形以外的形狀，所以需要額外處理物件以外的顏色。

圖 22 人物圖片

```
// people
if( people_left+1 < x && x < people_left + 40 - 1 &&
    people_up   < y && y < people_up+39 && people_pixel!=12'h000) begin
    {vgaR, vgaG, vgaB} = people_pixel;
end
```

圖 23 擋掉黑色色塊

圖 22 是網路上抓下來用的圖片，可以看到人物是不規則色塊組成，身體外部有很多黑色，這導致圖片顯示在螢幕上時，人物周圍會有黑色色塊，不好看，因此在 code 裡面需要把不要的顏色擋掉。

圖 23 是我擋掉黑色的寫法，people_pixel 是人物 pixel，12'h000 在 RGB 代碼代表黑色。當 people_pixel 為黑色時，代表這些是我不要的黑色色塊，此時不要把它輸出給 vga 即可。

其餘所有不規則圖片同理。

-> 下載圖片
-> 去背
-> 換上黑色背景(換其他顏色)
-> 在 code 裡面擋掉該顏色
-> 成功在 vga 顯示上達到去背效果

繪製地板花紋

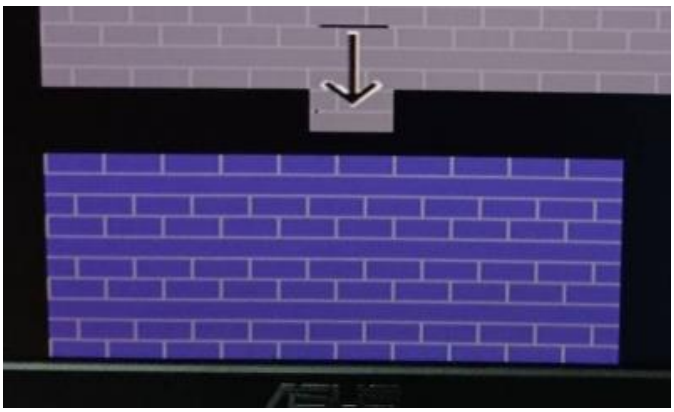

圖 24 地板花紋 Demo

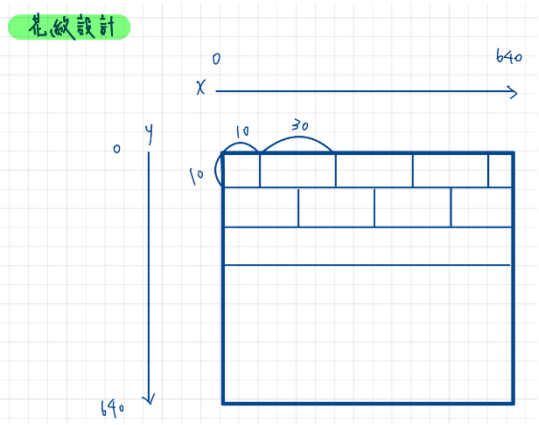

圖 25 花紋設計圖

網路上找不到滿意的花紋，因此地板的花紋也是手刻一堆 if 畫出來的，效果如上左圖。
設計圖參考上右圖，我是每三排重複畫一次一樣的花紋，這三排的座標分別為:

y~y+10. x+10,x+=20
y+10~y+20. x+20, x+=20
y+20~y+30

上面三組座標為一個循環，並搭上喜歡的顏色就能畫出照片中的好看的花紋。

```
if(220<=x && x<=520) if(y==470) {vgaR, vgaG, vgaB} = 12'h767;

if(10<=y && y<=10+10) begin
    if(x==230) {vgaR, vgaG, vgaB} = 12'h767;
    if(x==250) {vgaR, vgaG, vgaB} = 12'h767;
    if(x==270) {vgaR, vgaG, vgaB} = 12'h767;
    if(x==290) {vgaR, vgaG, vgaB} = 12'h767;
    if(x==310) {vgaR, vgaG, vgaB} = 12'h767;
    if(x==330) {vgaR, vgaG, vgaB} = 12'h767;
    if(x==350) {vgaR, vgaG, vgaB} = 12'h767;
    if(x==370) {vgaR, vgaG, vgaB} = 12'h767;
    if(x==390) {vgaR, vgaG, vgaB} = 12'h767;
    if(x==410) {vgaR, vgaG, vgaB} = 12'h767;
    if(x==430) {vgaR, vgaG, vgaB} = 12'h767;
    if(x==450) {vgaR, vgaG, vgaB} = 12'h767;
    if(x==470) {vgaR, vgaG, vgaB} = 12'h767;
    if(x==490) {vgaR, vgaG, vgaB} = 12'h767;
    if(x==510) {vgaR, vgaG, vgaB} = 12'h767;
end
```

圖 26 畫地板花紋的 verilog code 範例

最後這邊附上我自己寫的生產花紋 python 檔。花紋動輒上百行，如果一個一個手打會累死人。
(https://gist.github.com/YEH-YU-YANG/0778f38ef6c04b9efba8cfdfe9db1dcb)

(3) 手電筒發光效果

圖 27 暗房中的圓型手電筒發光

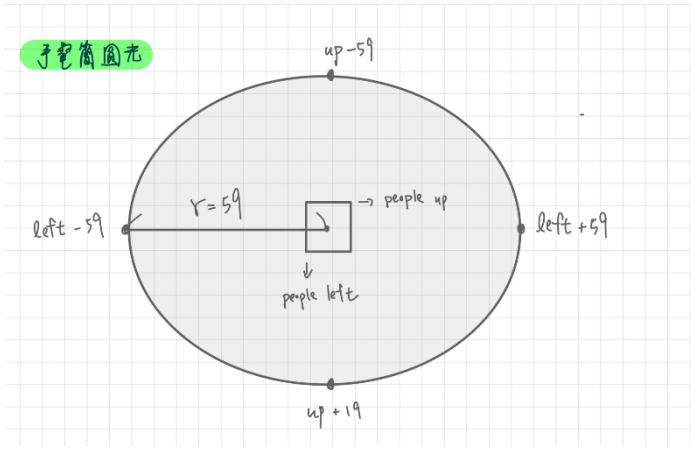

圖 28 圓型手電筒設計

這個手電筒會隨著人物移動。
要達到這樣的效果需要**判斷目前的 x、y 是否在圓形光圈裡面**:

若在圓形光圈裡面，vga output 出所有要顯示的資訊(地板、花紋、牆壁、人)
若不在圓形光圈裡面，vga 只會 output 黑色

在 fpga 這個方形的世界裡面，圓形光圈是我最自豪的設計之一。
上右圖是設計圖，我設計了以人物中心**半徑為 59** 的圓形，並選擇爆寫 if 敘述的方法畫圓。

為了畫出這個圓形我一樣弄出了 python 檔幫忙代勞，最後弄出上萬行 if。
(https://gist.github.com/YEH-YU-YANG/8aea6e1cc527844a975bd98c9661b7bc)

破萬行 if 的一小部分:

```
if ((
    (x==people_left+11 && y==people_up-40) ||
    (x==people_left+12 && y==people_up-40) ||
    (x==people_left+13 && y==people_up-40) ||
    (x==people_left+14 && y==people_up-40) ||
    (x==people_left+15 && y==people_up-40) ||
    (x==people_left+16 && y==people_up-40) ||
    (x==people_left+17 && y==people_up-40) ||
    (x==people_left+18 && y==people_up-40) ||
    (x==people_left+19 && y==people_up-40) ||
    // ....
) begin
    // 畫圖
end
```

圖 29 判斷圓形的 if 敘述

我會在這個 if 裡面用到上面第 2 點繪製地圖的技巧，把該顯示的畫面顯示出來。
如此就可以達到只在圓圈裡面顯示畫面，其餘畫面為黑的手電筒效果。

(5) 撿起發光道具、撿起鑰匙

撿起發光道具

```
1  always@(posedge clk) begin
2      if(rst) apple <= 0;
3      else if(apple) apple <= 1;
4      else apple <= next_apple;
5  end
6
7  always@(*) begin
8      if(stage_state==5) begin
9          if(360<=people_left+19 && people_left+19<=400 && 65<=people_up+19 && people_up+19<=95 && key_down[`F10]) begin
10             next_apple = 1;
11         end
12         else begin
13             next_apple = apple;
14         end
15     end
16     else begin
17         next_apple = apple;
18     end
19 end
```

圖 30 撿發光道具

我會判斷目前人是否在發光道具所在的 state -> line 8
再判斷人物是否走到發光道具旁邊並按下拾取鍵(F10) -> line 9
若有則撿起，若無則繼續判斷。

```
1  //apple
2  if(!APPLE_OUT && 380<x && x<=400 && 70<=y && y<=90 && apple_pixel!=12'h000) {vgaR, vgaG, vgaB} = apple_pixel;
```

圖 31 顯示發光道具

最後在地圖上我會判斷是否撿起了發光道具。若已經撿起則不會再把它印出來。

撿起鑰匙

```
always@(posedge clk) begin
    if(rst) begin
        key <= 0;
    end
    else begin
        if(key) key <= 1;
        else key <= next_key;
    end
end
always@(*) begin
    if(stage_state==2 && chair_state==2 && key_down[`F10] &&
        people_up < chair_up && chair_up<people_up+39 &&
        people_up+39<=chair_up+39 && chair_left<=people_left+19 && people_left+19<=chair_left+39) next_key = 1;
    else next_key = 0;
end
```

圖 32 檢鑰匙

因為鑰匙放在高處，角色需要站在椅子上才能撿到鑰匙。
所以這邊會判斷椅子跟鑰匙是否處於同一個空間，再去判斷人現在是否站在椅子上。
若條件都達成即可撿起鑰匙。

(6) 輸入密碼

```
if (!PASS_OUT && CIN && !lock && been_ready && key_down[last_change] &&
   ( (stage_state==6 && 370<=people_left && people_left<=420 && 50<=people_up && people_up<=135) || stage_state==7)) begin
    SEVEN_SEGMENT <= {SEVEN_SEGMENT[11:0],key_num};
end
```

圖 33

密碼需要站在特定位置上才能輸入，因此這邊會判斷人物是否處於正確空間的正確位置，再拉起 CIN switch 即可輸入密碼。
這邊使用 left shift 的技術，每次輸入一個數字就會讓原本的數字往左移一格，直到輸入正確的密碼或是放棄輸入為止。

```
if(PASS_OUT) begin
    SEVEN_SEGMENT[15:12] <= `P;
    SEVEN_SEGMENT[11:8] <= `A;
    SEVEN_SEGMENT[7:4] <= `S;
    SEVEN_SEGMENT[3:0] <= `S;
end
else if(!PASS_OUT && stage_state!=6 && stage_state!=7) begin
    SEVEN_SEGMENT[15:12] <= `DASH;
    SEVEN_SEGMENT[11:8] <= `DASH;
    SEVEN_SEGMENT[7:4] <= `DASH;
    SEVEN_SEGMENT[3:0] <= `DASH;
end
```

```
if(FAIL) begin
    SEVEN_SEGMENT[15:12] <= `F;
    SEVEN_SEGMENT[11:8] <= `A;
    SEVEN_SEGMENT[7:4] <= `I;
    SEVEN_SEGMENT[3:0] <= `L;
end
else if(SUCCESS) begin
    SEVEN_SEGMENT[15:12] <= `G;
    SEVEN_SEGMENT[11:8] <= `O;
    SEVEN_SEGMENT[7:4] <= `O;
    SEVEN_SEGMENT[3:0] <= `D;
end
```

圖 34 PASS、DASH　　圖 35 FAIL、SUCCESS

圖 34 是密碼鎖和顏色鎖都答對時，七段顯示器會顯示「PASS」告訴玩家可以破關了。
圖 35 是失敗和成功逃離時，七段顯示器會顯示「FAIL」 or「 SUCCESS」。

3. 實作完成度 / 難易度說明 / 分工

實作完成度

最初的 proposal 只有三個房間(關卡)、一個密碼鎖、音樂。

與最初的 proposal 比較:

缺少:

音樂、音效模組

新增:

更多房間(關卡)
黑房
手電筒
密碼鎖
鑰匙
椅子
地板花紋
手繪圖案

難易度說明

以遊戲製作的角度來說:

使用到的技術都是上課學過的，.xdc、vga、keyboard、seven segment、led、switch、button，沒有用到課外知識寫 code。

以遊玩角度來說:

假設 0 分最簡單、100 分最難，遵守規則的情況下我覺得難度落在 70 分。
因為這是個解謎遊戲，相較於動作遊戲不需要很快的手速，操作上很簡單。反而是有許多燒腦、邏輯爆炸的地方才是難的關鍵。

以下列出測試者們覺得難的地方的回饋:
欄杆 -> 要用欄杆擋住數字是個新奇的想法　(By 測試者 1)
推箱子擋鬼 -> 以為箱子只要用來拿鑰匙而已，沒想到可以擋鬼　(By 測試者 1)
顏色鎖 -> 會被圖片裡顏色的形狀誤導，而沒想到是數字而已　(By 測試者 1、3)
黑房-> 進黑房後螢幕黑掉以為是遊戲掛掉，沒想到還要找手電筒　(By 測試者 2)

測試者 1 遊玩時間 32 分鐘
測試者 2 遊玩時間 24 分鐘
測試者 3 遊玩時間 17 分鐘
本人(上帝視角)遊玩時間 13 分鐘

分工

只有我一人

4. 是否包含課程外的部分及比重

無。
全部 code 都是自己想的。

5. 測試完成度

除了邊界判定沒有完成以外，其餘所有可能性都測試過

測試人轉換地圖是否傳送到正確地圖
測試人踩上每張地圖的傳送點是否正確
測試人是否被傳送到新地圖的正確初始點

測試椅子轉換地圖是否傳送到正確地圖
測試椅子踩上每張地圖的傳送點是否正確
測試椅子是被背傳送到新地圖的正確初始點

測試鬼 1 的上下左右抓人
測試鬼 2 的上下抓人
測試偷看線索鬼 1 是否正確抓人
測試鬼 1 是否能被椅子堵住
測試鬼 1 被椅子堵住太誇張是否會生氣

測試檢取道具
測試道具撿起後是否從地圖顯示上消失
測試不靠椅子是否能撿起鑰匙

測試手電筒發光區塊
測試地板花紋
測試人站在鐵牢後面是否顯示正確
測試人站在鐵牢前面是否顯示正確

測試道具拿取順序是否影響通關
測試缺少道具是否影響通關

測試密碼鎖輸入
測試密碼鎖輸入地區邊界

6. 困難與解決方法

困難 1: Vivado synthesis or implementation failed without error

因為 message 那邊沒噴 error, 必須先打開 log 檔看錯誤訊息

如果錯誤訊息為「# # an unexpected error has occurred (exception_access_violation) # stack: no stack trace available, please use hs_err_<pid>.dmp instead.」

synthesis 階段 fail
解法 1 (無效)
在 vivado console 下指令 「set_param route.ignoreLocalClocks true」
(https://support.xilinx.com/s/question/0D52E00006hpPHJSA2/exceptionaccessviolation?language=en_US)
解法 2 (以下全部無效)
工作管理員結束 vivado.exe 再重開、電腦重開機、開新的專案重跑、檔案路徑只能有英文、電腦登入的 USER 為英文名
解法 3 (有效)
在 synthesis setting 中把 flatten_hierarchy 設為 none
(https://support.xilinx.com/s/question/0D52E00006hpM6ISAU/exceptionaccessviolation?language=en_US)
解法 4 (有效)
code 裡面能節省 register bit 數量的地方盡量節首、或是把功能弄簡單點 :(

implementation 階段 fail 解法 1 (無關)
在 vivado console 下指令 「set_param pwropt.maxFaninFanoutToNetRatio 2000」
(https://blog.csdn.net/qq_35809085/article/details/129653502)

如果錯誤訊息為「An unexpected error has occurred (EXCEPTION_ACCESS_VIOLATION)" is observed during post opt write_checkpoint.」
解法 (試過但無關)
在 vivado console 下指令 「opt_design -retarget -propconst -bufg_opt -shift_register_opt -bram_power_opt 」
(https://support.xilinx.com/s/article/71509?language=en_US)

困難 2:手繪地圖

因為網路上沒有滿意的地圖，最後我是自己設計地圖形狀。
壞處是要花很多時間設計地圖，並用 code 畫出來; 好處是可以節省記憶體

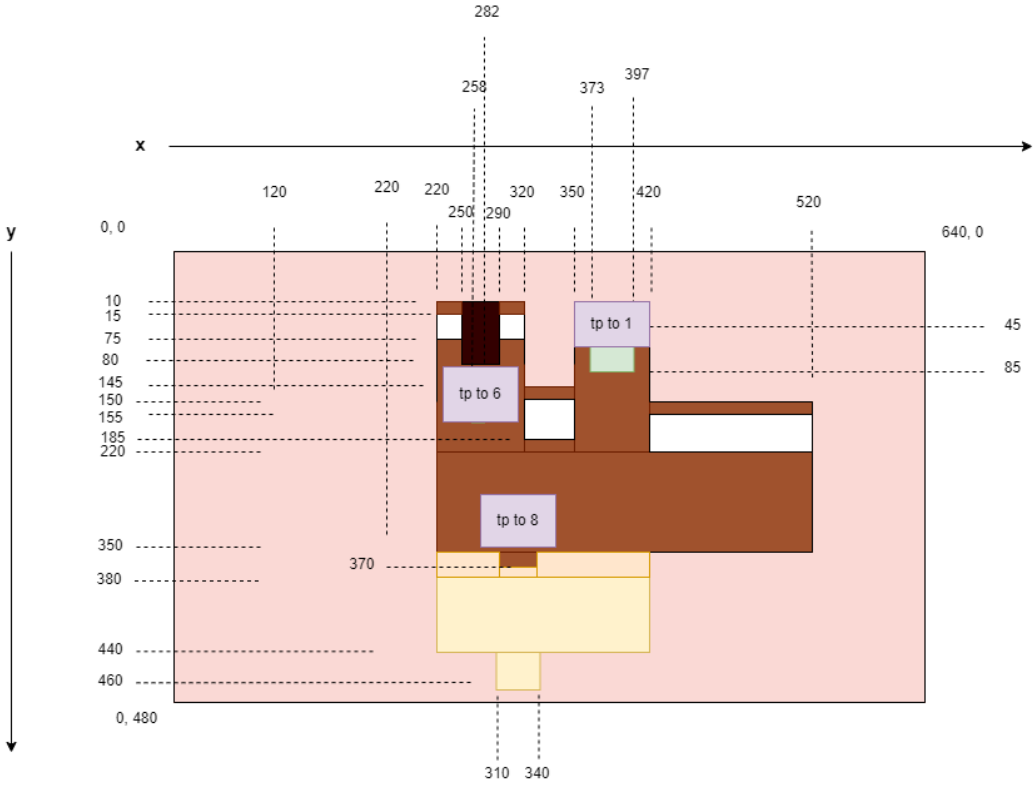

圖 36 地圖 0 設計

圖 36 是地圖 0 的設計圖，需要構思哪邊是地板、白牆、黑牆、門、道具擺放位置、傳送點等等……。

除此之外，每個物件的 x、y 軸都要標示出來，這樣寫 code 的時候才知道 h_cnt、v_cnt 時，要花出甚麼東西。

若是畫出來的結果不好看、比例不對、畫面偏移等等，就需要全部數據重新調整。最花時間的地方是設計地圖、比例調整、用 code 畫地圖。

困難 3:記憶體

如果我的地圖全部都是外部圖片匯入，只會有兩個結果:

1.BRAM 被用光
2.降低 Pixel 但圖片很糊

這兩個都不是我想看到的結果。
設計地圖階段也知道沒辦法只用一小塊圖片畫出整體圖片的技術，於是才會選擇用 code 畫地圖，而不是把圖片 Load 進 fpga。

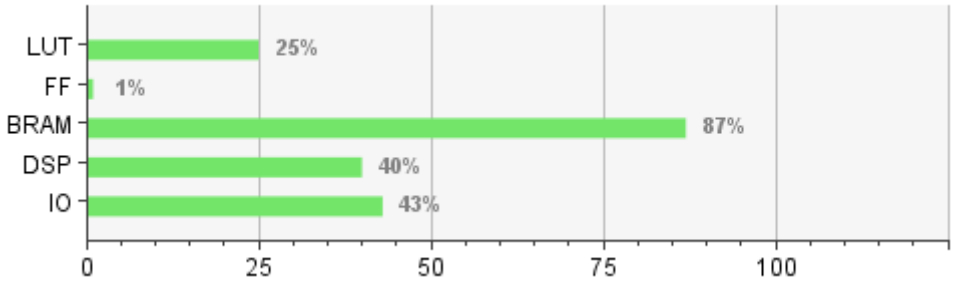

圖 36 地圖 0 設計

BRAM 最終控制在 87%上。

其餘困難:物件重疊、人物移動、地板花紋、手電筒形狀、

這些困難已在前面的報告提及，且提供解法。

7. 心得討論

起初寫 vga 的 lab 時就對 vga 很恐懼，因為我其實不太了解當時 lab 的圖片是怎麼往左往右移動的，但做完 project 後我對 vga 變得非常熟悉，逼著自己想怎麼寫出能移動的物體，要怎麼讓物體跟顯示在其他物件上面等等。最後逼著逼著就變得很會寫 vga，後期在新增圖片相關的功能已經能順手拈來。

做這份 project 的另一個印象深刻的點是設計遊戲，尤其是地圖與線索設計。

哪邊要放牆壁、哪邊要放線索、哪邊要放地板等等，且每次弄出 bit file 後只要物件擺的不好看，全部物件的數據都要重新調過。畫圖的 code 有 14000 行，每次重調都是個耗時的事情。

線索設計就是小宇宙爆發的地方，最初的 proposal 只有想到兩個破關線索:(1) 密碼鎖 (2) 推床之後做一做覺得這樣太無聊，就新增了鑰匙、椅子、欄杆、顏色鎖、黑房、手電筒、鬼等等，其中黑房、手電筒還是睡覺作夢夢到的點子。之後也很慶幸有努力地想 idea，大大的豐富了 project 的可玩性與燒腦程度。

唯一遺憾的是相較於其他同學的動作類遊戲，解謎遊戲沒有按鍵帶來的爽快感，解謎遊戲的受眾也比較小，使用到的技術難度也低了一點。但我相信在遊戲設計的層面上，我的創意、心血、美術、遊戲設計邏輯以及對這個遊戲的熱愛不會輸給同學們。

8. 附件

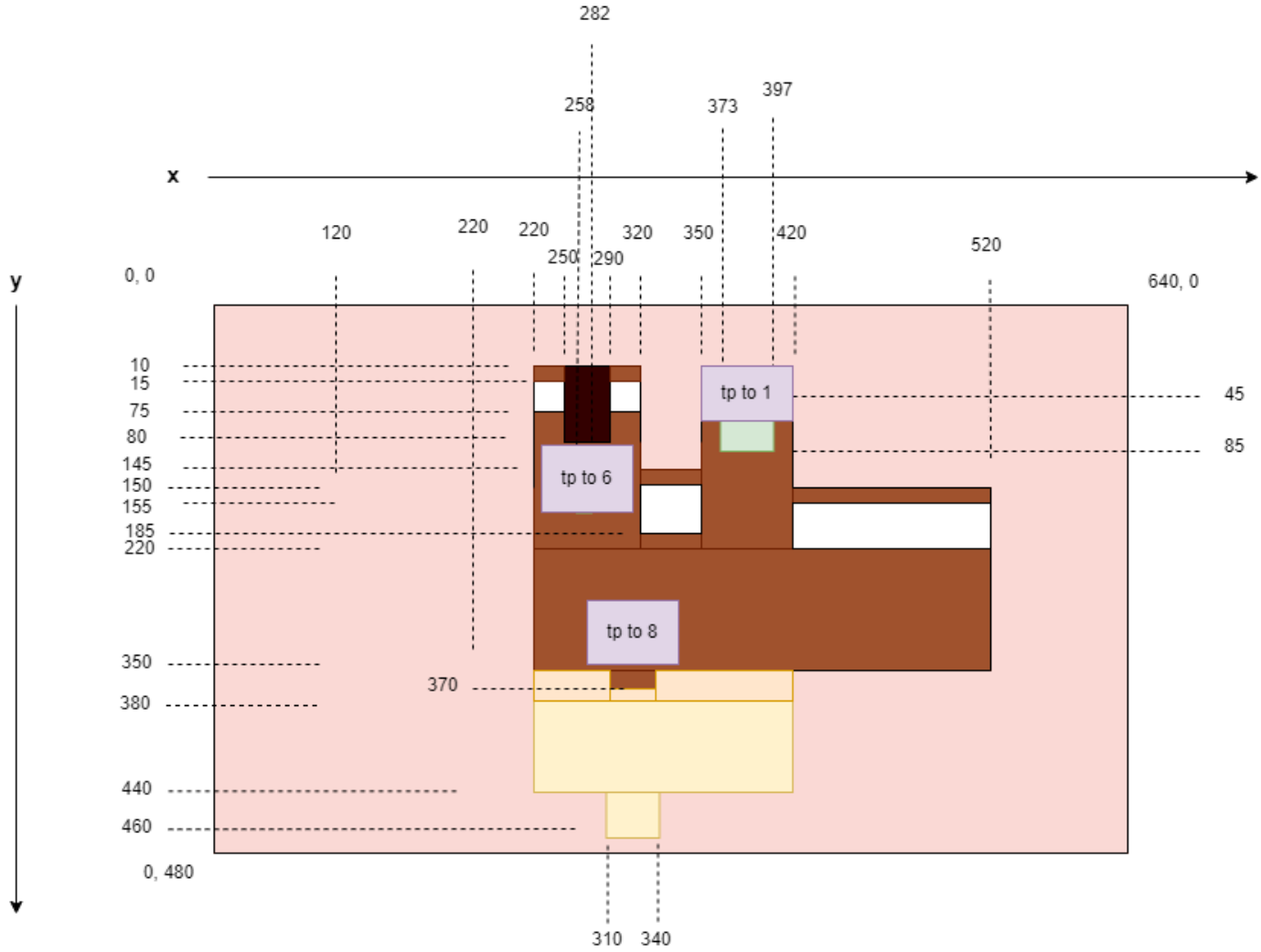

圖 37 地圖 0 設計

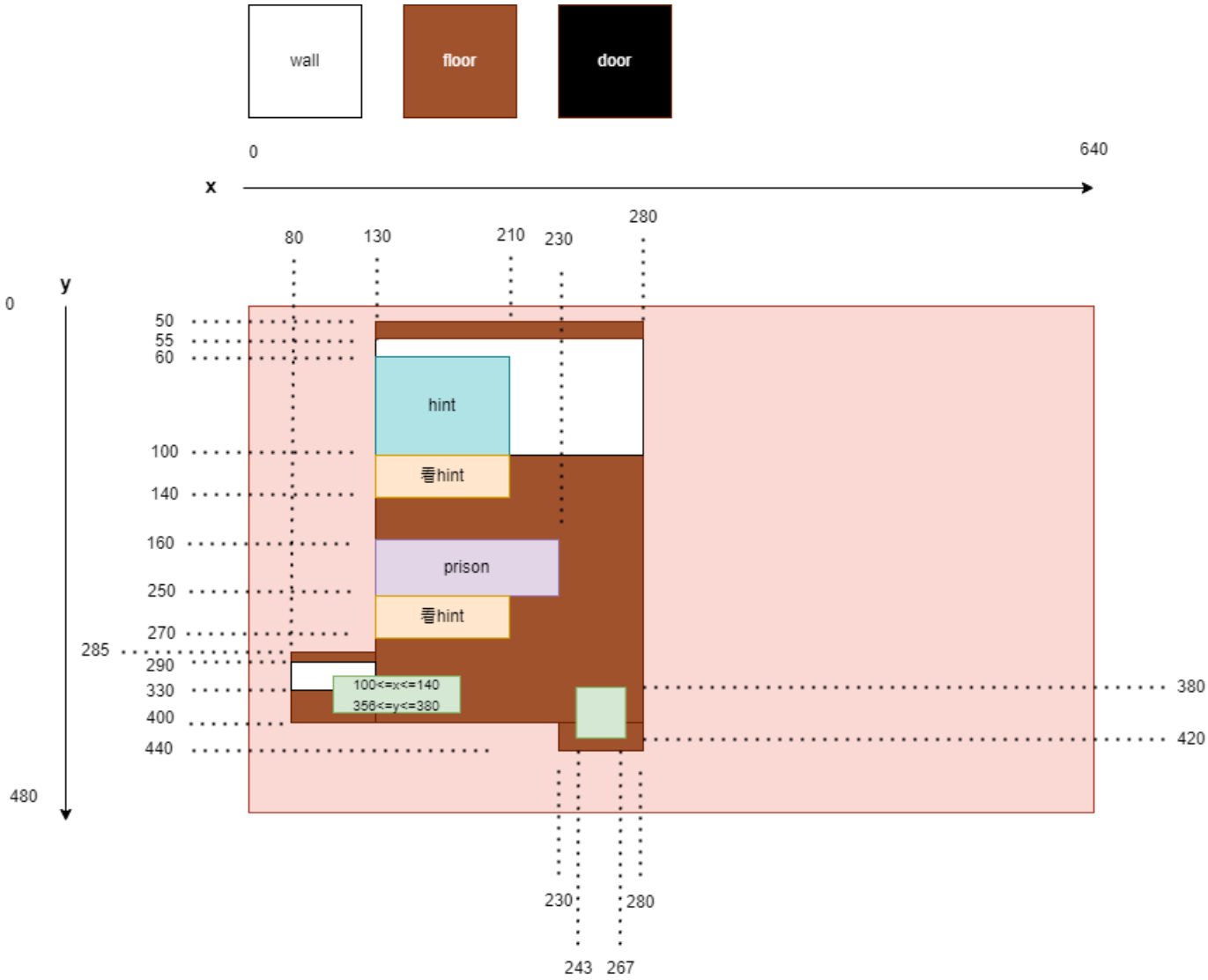
wall
floor
door
0
640
x
y
0
480
80
130
210
230
280
50
55
60
100
140
160
250
270
285
290
330
400
440
380
420
hint
看hint
prison
看hint
100<=x<=140
356<=y<=380
230
280
243
267

圖 38 地圖 1 設計

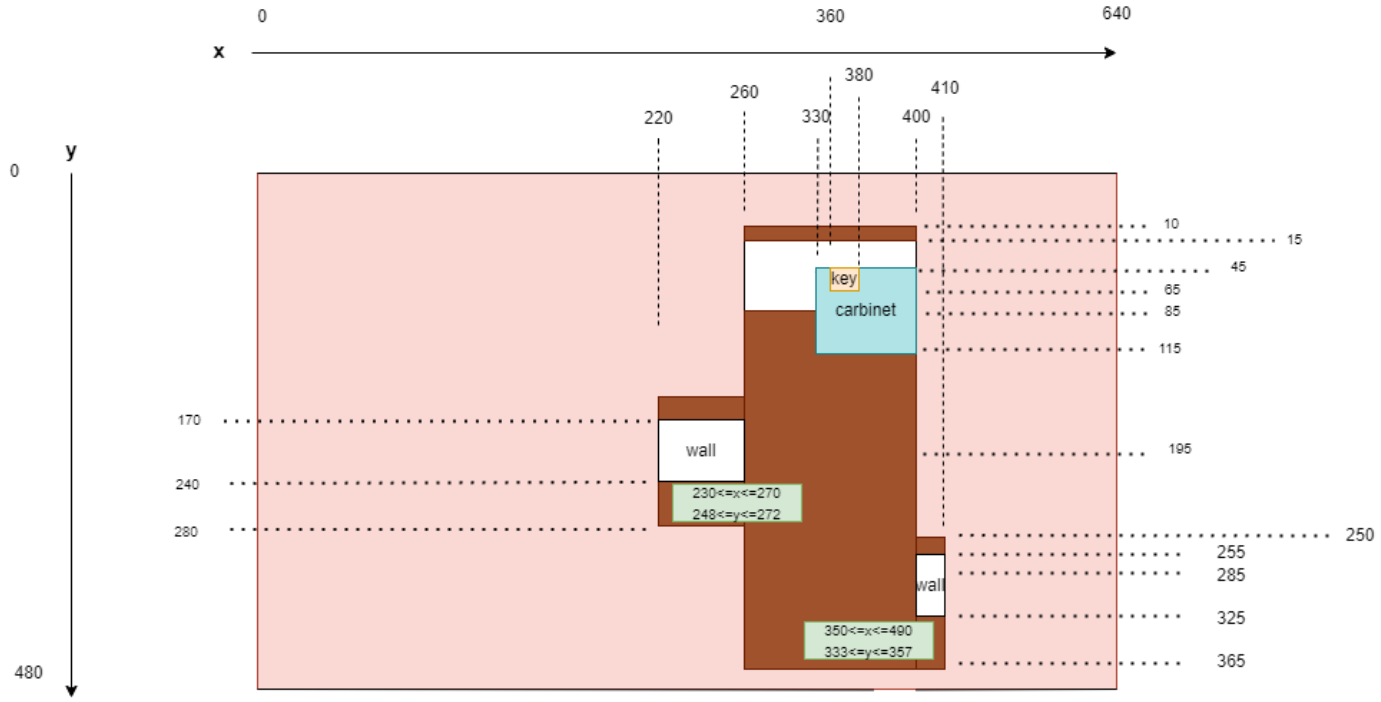
0
360
640
x
380
260
410
220
330
400
y
0
10
15
45
key
carbinet
65
85
115
170
wall
195
240
230<=x<=270
248<=y<=272
280
250
255
285
wall
325
350<=x<=490
333<=y<=357
365
480

圖 39 地圖 2 設計

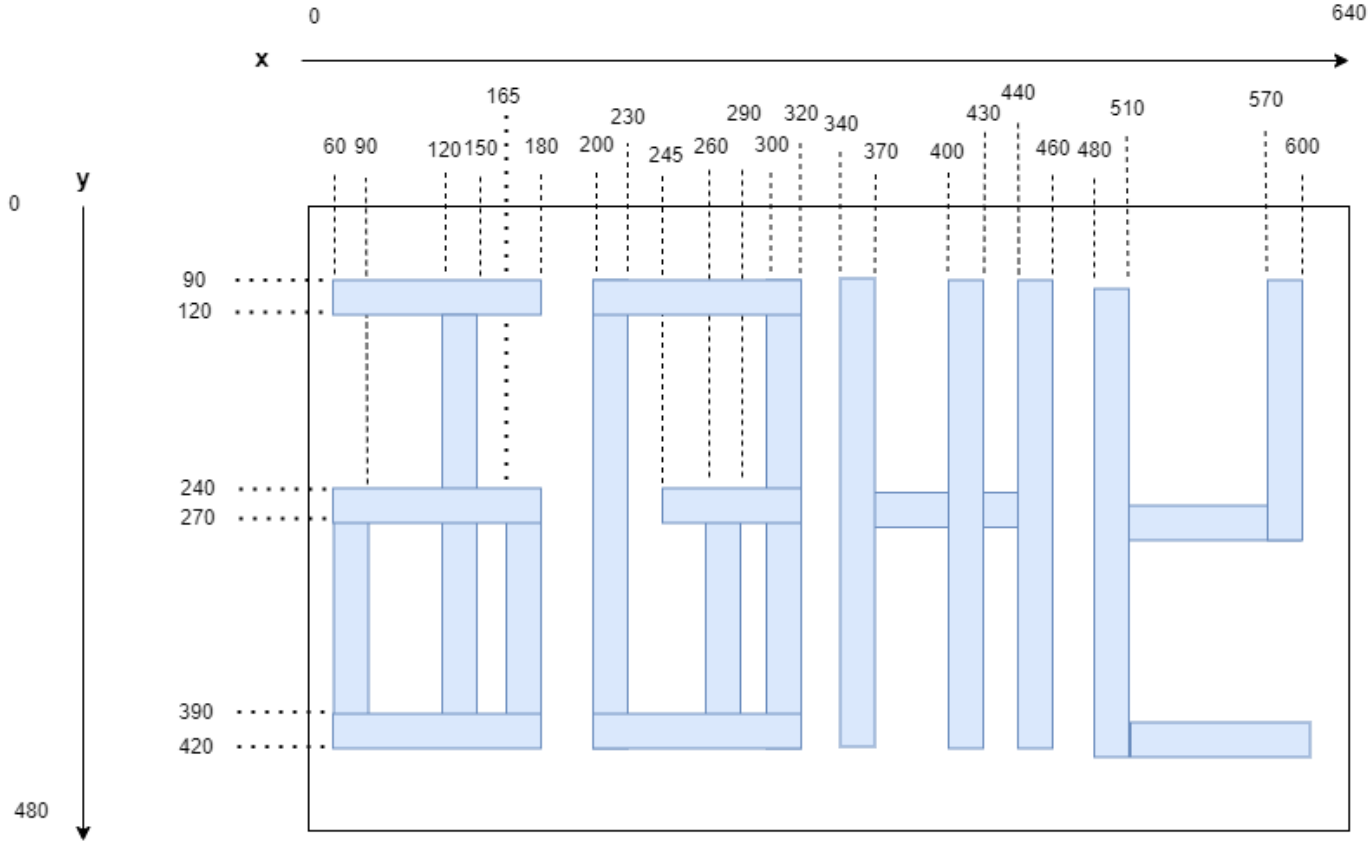
0
640
x
165
230
290
320
340
440
430
510
570
60 90
120 150
180
200
245
260
300
370
400
460 480
600
y
0
90
120
240
270
390
420
480

圖 40 地圖 3 設計

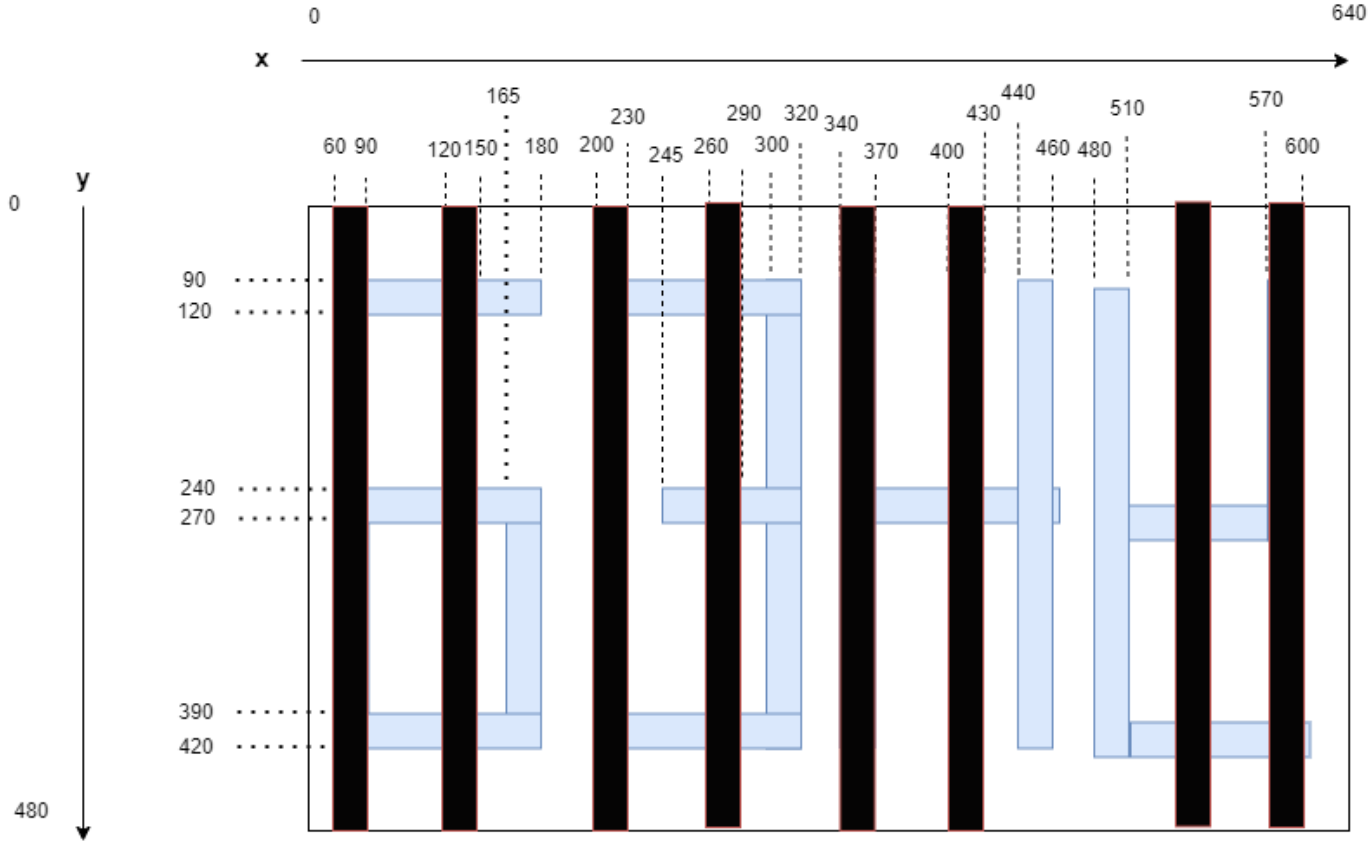
0
640
x
y
0
480
60 90
120 150
165
180
200
230
245
260
290
300
320
340
370
400
430
440
460
480
510
570
600
90
120
240
270
390
420

圖 41 地圖 4 設計

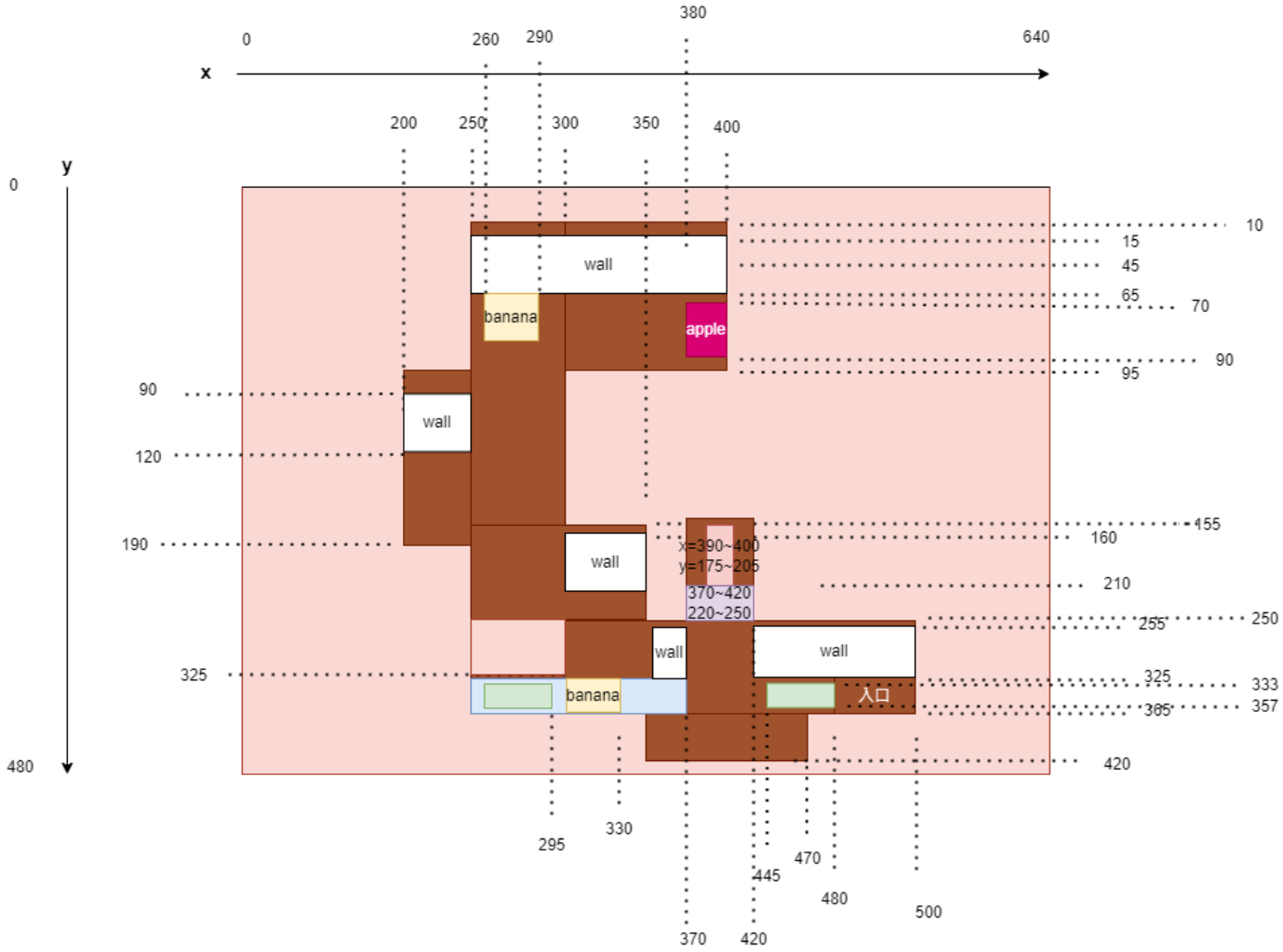
0
260
290
380
640
x
200
250
300
350
400
y
0
480
wall
banana
apple
wall
wall
x=390~400
y=175~205
370~420
220~250
wall
wall
banana
入口
10
15
45
65
70
90
95
90
120
190
155
160
210
250
255
325
325
333
357
365
420
295
330
370
420
445
470
480
500

圖 42 地圖 5 設計

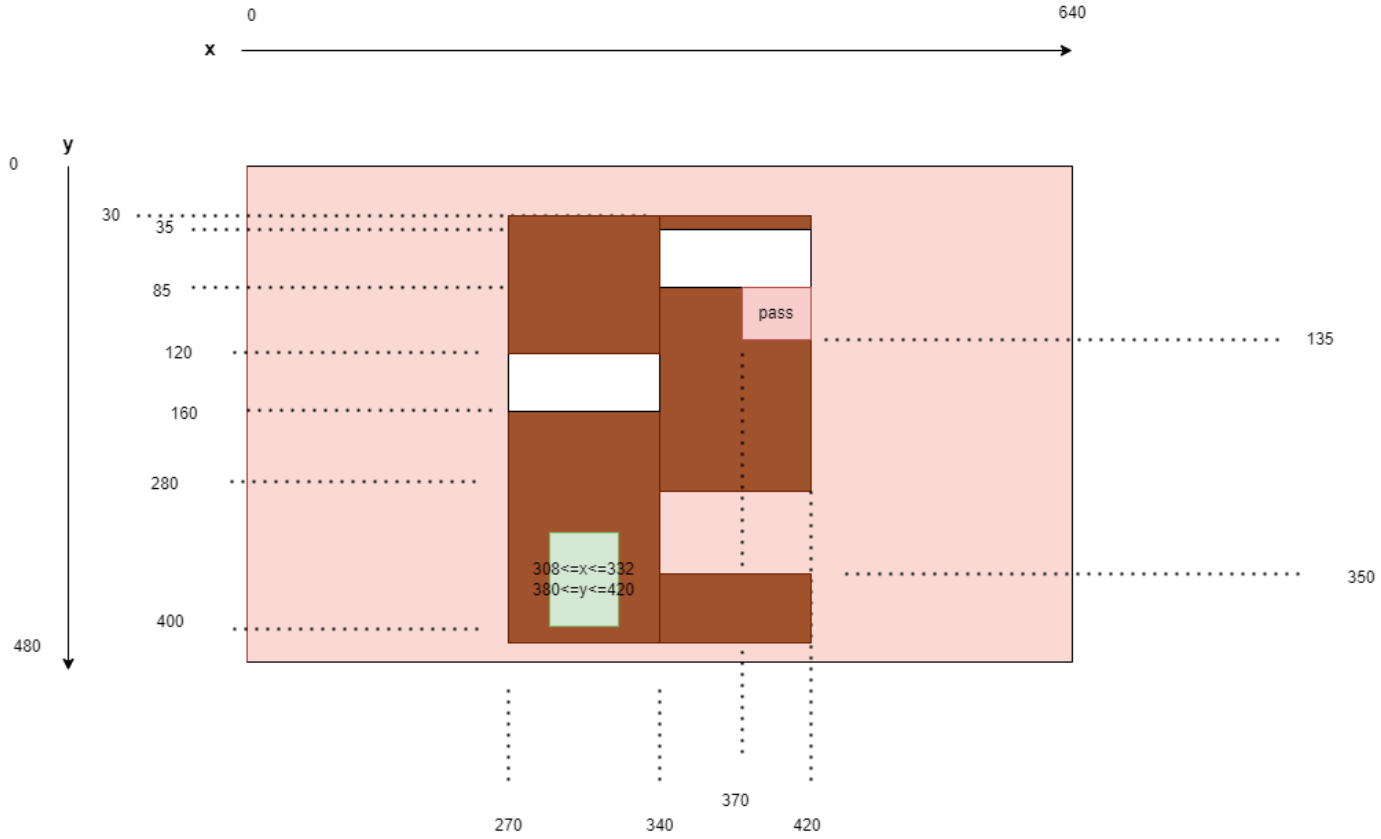
0
640
x
y
0
30
35
85
120
160
280
400
480
pass
135
350
308<=x<=332
380<=y<=420
270
340
370
420

圖 43 地圖 6 設計

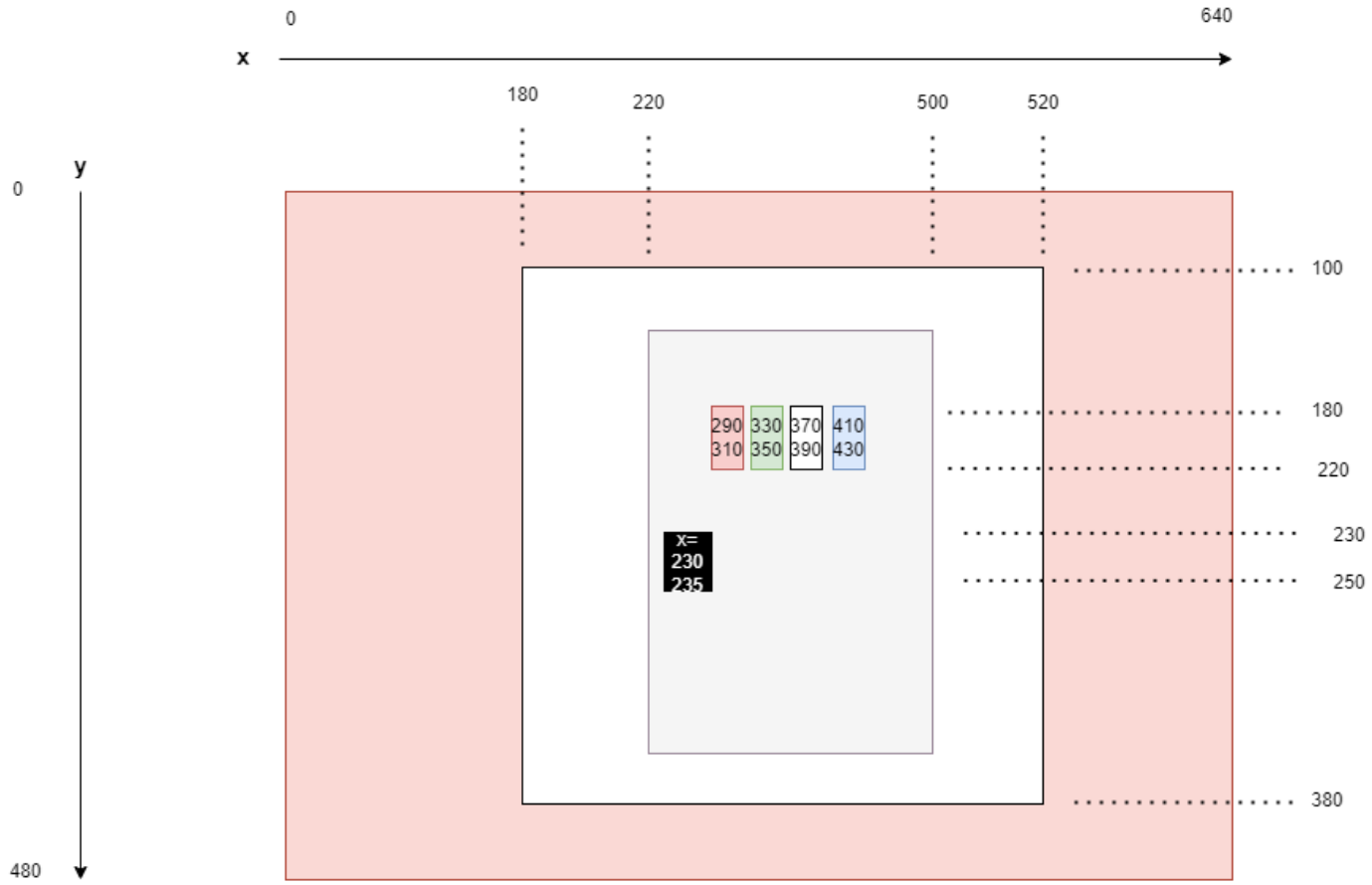
0
640
x
180
220
500
520
y
0
480
100
180
220
230
250
380
290
310
330
350
370
390
410
430
x=
230
235
370

圖 44 地圖 7 設計

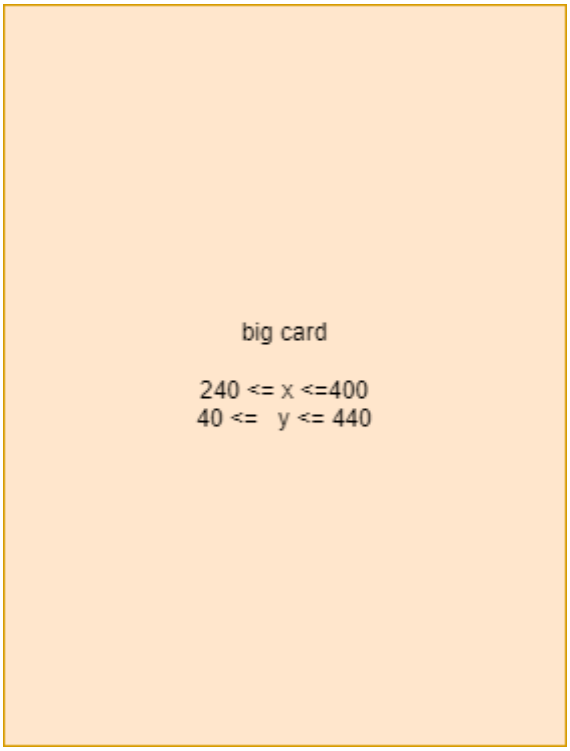

圖 45 地圖 8 設計