“Police” run

Yimin Chu
Zhongyi Wang

summary

1. Project demo
2. VGA animation
3. Keyboard
4. Game logic
5. Modelsim

Game demo

https://youtu.be/OXnSPdO99_k

Description of project

Controls: The left player uses WASD to move, while the right player uses IJKL to move.

Game Rules: Both players must avoid being caught by the pursuing police. If either player is caught within the time limit, mission fails. If both players manage to avoid being caught, mission is success. Additionally, the two players must avoid colliding with each other while evading the police.

Escape Tips:

1.Diagonal Movement: Moving diagonally is faster, so use diagonal paths to escape.

2.Police Collision: Lure the pursuing police into colliding with each other. They will be stunned for one second after a collision.

VGA object

1. Thieves Pattern

   8x8 thieves patterns are defined using a 2D array (spaceship_pattern).

   The patterns are initialized in the initial block to represent the spaceship shape, including colors for different parts.

2. Thieves Positioning and Movement

   Patterns' initial position (spaceship_x, spaceship_y) is set to the center of a 160x120 resolution.

   Both Thieves can moved independently in four directions (up, down, left, right) by pressing buttons on keyboard.

   Boundary conditions are enforced to keep the spaceship within the screen.

VGA painting (refresh rate=2604)

1.Traverse every pixel on the screen

The current pixel position (current_x, current_y) is updated using a nested loop-like structure to scan through the display. It traverses 2600 times each pixel per second which make sure the animation looks smooth.

2. Colour Detection

The color of the pixel (colour) is determined based on whether the traverse pixel position is part of the spaceship or in the background.

Object update

1.Lower the speed of animation

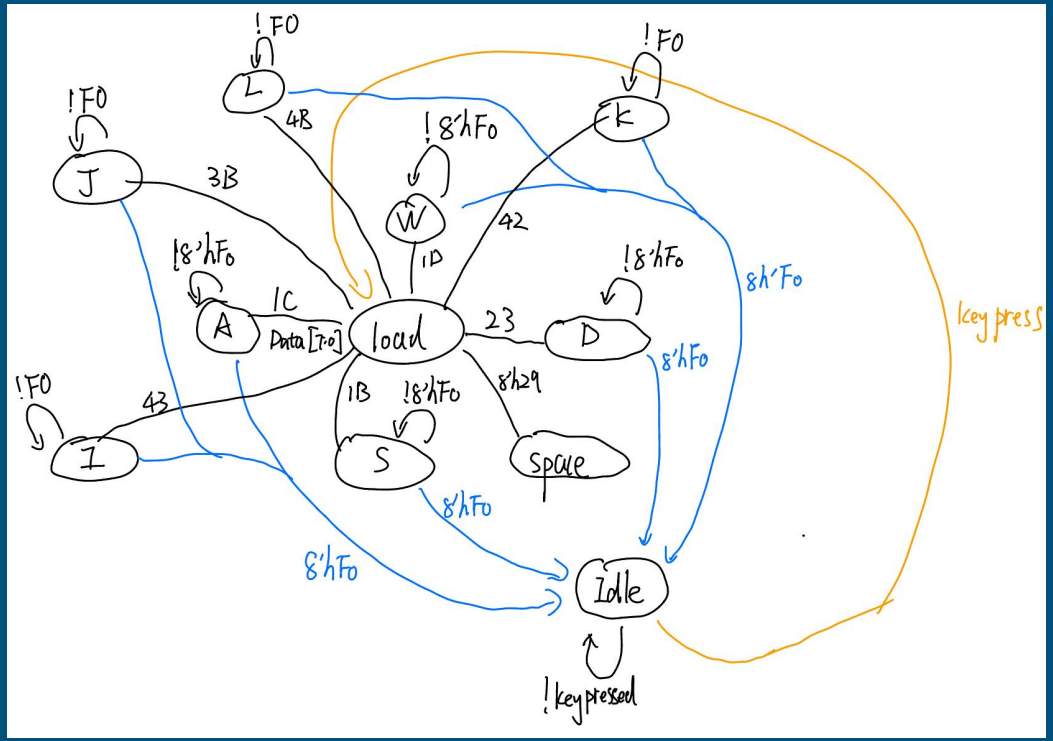
Using a 50MHz clock as the `clk` to update the position makes the ball move too fast, causing it to fly out of bounds with just a light press. To address this, we use a counter to reduce the 50MHz frequency. The counter acts as the new `clk`, ensuring the position updates are not too fast.

2.Control the speed of thieves and police

By assigning the thief a smaller counter value than the police, the thief's update frequency becomes higher than the police's, resulting in the thief moving faster than the police.

3. Same logic for drawing progress bar

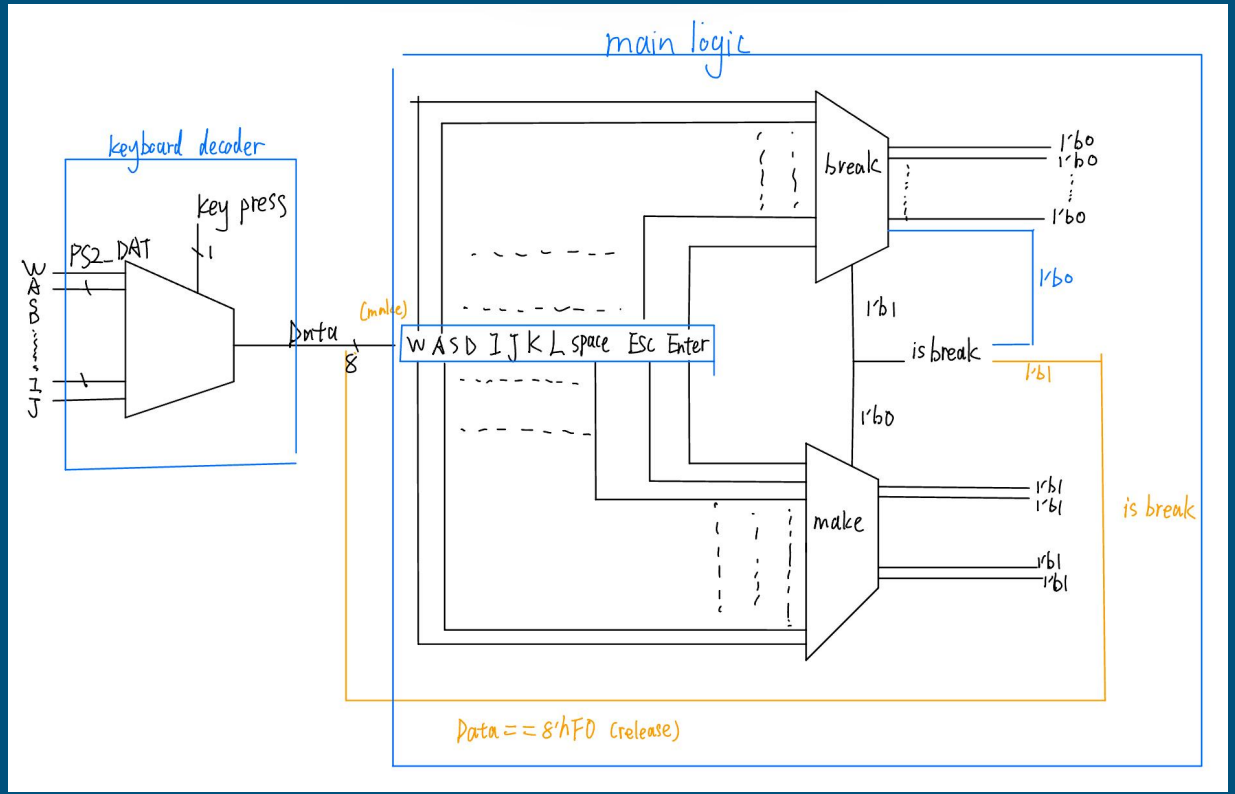
Keyboard (first version)



keyboard(first)

```
// 状态机逻辑
always @(posedge clk or posedge reset) begin
    if (reset) begin
        state <= STATE_IDLE;
        cur_key_code <= 8'h00;
        // 重置输出信号
        W         <= 1'b0;
        A         <= 1'b0;
        S         <= 1'b0;
        D         <= 1'b0;
        I         <= 1'b0;
        J         <= 1'b0;
        K         <= 1'b0;
        L         <= 1'b0;
        Fire      <= 1'b0;
        start     <= 1'b0;
        ResetGame <= 1'b0;
    end else if (keypressed) begin
        case (state)
            STATE_IDLE: begin
                if (Data == 8'hF0) begin
                    state <= STATE_BREAK;
                end else begin
                    cur_key_code <= Data;
                    state <= STATE_MAKE;
                    // 设置对应的输出信号
                    case (Data)
                        W_CODE:     W         <= 1'b1;
                        A_CODE:     A         <= 1'b1;
                        S_CODE:     S         <= 1'b1;
                        D_CODE:     D         <= 1'b1;
                        I_CODE:     I         <= 1'b1;
                        J_CODE:     J         <= 1'b1;
                        K_CODE:     K         <= 1'b1;
                        L_CODE:     L         <= 1'b1;
                        SPACE_CODE: Fire      <= 1'b1;
                        ENTER_CODE: start     <= 1'b1;
                        ESC_CODE:   ResetGame <= 1'b1;
                        default: ; // 不处理其他按键
                    endcase
                end
            end
```

Keyboard (final improvement)



Initialization

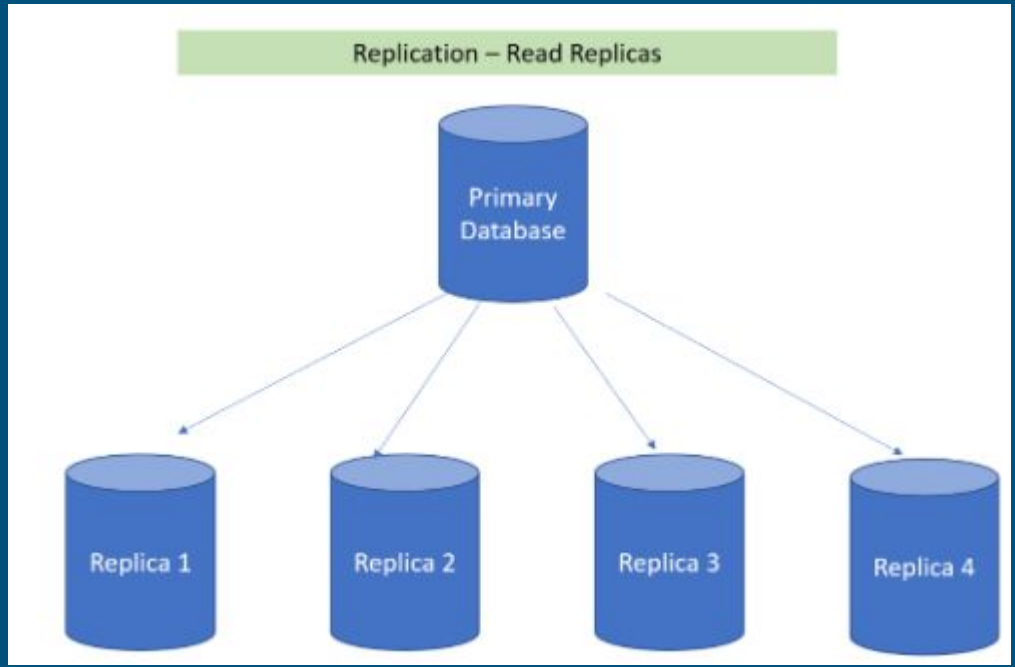
```
    initial begin
        // Original spaceship pattern (Blue spaceship)
        spaceship_pattern[0][0] = 3'b000; spaceship_pattern[0][1] = 3'b000; spaceship_pattern[0][2] = 3'b110; spaceship_pattern[0][3] = 3'b110;
        spaceship_pattern[0][4] = 3'b110; spaceship_pattern[0][5] = 3'b110; spaceship_pattern[0][6] = 3'b000; spaceship_pattern[0][7] = 3'b000;
        spaceship_pattern[1][0] = 3'b000; spaceship_pattern[1][1] = 3'b110; spaceship_pattern[1][2] = 3'b110; spaceship_pattern[1][3] = 3'b110;
        spaceship_pattern[1][4] = 3'b110; spaceship_pattern[1][5] = 3'b110; spaceship_pattern[1][6] = 3'b110; spaceship_pattern[1][7] = 3'b000;
        spaceship_pattern[2][0] = 3'b110; spaceship_pattern[2][1] = 3'b110; spaceship_pattern[2][2] = 3'b110; spaceship_pattern[2][3] = 3'b110;
        spaceship_pattern[2][4] = 3'b110; spaceship_pattern[2][5] = 3'b110; spaceship_pattern[2][6] = 3'b110; spaceship_pattern[2][7] = 3'b110;
        spaceship_pattern[3][0] = 3'b110; spaceship_pattern[3][1] = 3'b110; spaceship_pattern[3][2] = 3'b110; spaceship_pattern[3][3] = 3'b111;
        spaceship_pattern[3][4] = 3'b111; spaceship_pattern[3][5] = 3'b110; spaceship_pattern[3][6] = 3'b110; spaceship_pattern[3][7] = 3'b110;
        spaceship_pattern[4][0] = 3'b110; spaceship_pattern[4][1] = 3'b110; spaceship_pattern[4][2] = 3'b111; spaceship_pattern[4][3] = 3'b111;
        spaceship_pattern[4][4] = 3'b111; spaceship_pattern[4][5] = 3'b111; spaceship_pattern[4][6] = 3'b110; spaceship_pattern[4][7] = 3'b110;
        spaceship_pattern[5][0] = 3'b110; spaceship_pattern[5][1] = 3'b111; spaceship_pattern[5][2] = 3'b111; spaceship_pattern[5][3] = 3'b111;
        spaceship_pattern[5][4] = 3'b111; spaceship_pattern[5][5] = 3'b111; spaceship_pattern[5][6] = 3'b111; spaceship_pattern[5][7] = 3'b110;
        spaceship_pattern[6][0] = 3'b000; spaceship_pattern[6][1] = 3'b111; spaceship_pattern[6][2] = 3'b111; spaceship_pattern[6][3] = 3'b111;
        spaceship_pattern[6][4] = 3'b111; spaceship_pattern[6][5] = 3'b111; spaceship_pattern[6][6] = 3'b111; spaceship_pattern[6][7] = 3'b000;
        spaceship_pattern[7][0] = 3'b000; spaceship_pattern[7][1] = 3'b000; spaceship_pattern[7][2] = 3'b111; spaceship_pattern[7][3] = 3'b111;
        spaceship_pattern[7][4] = 3'b111; spaceship_pattern[7][5] = 3'b111; spaceship_pattern[7][6] = 3'b000; spaceship_pattern[7][7] = 3'b000;
```

List of Coordinates

```
        // Spaceship 1 initial position
        x_pos[1] = spaceship_x;
        y_pos[1] = spaceship_y;

        // Spaceship 2 initial position
        x_pos[2] = spaceship2_x;
        y_pos[2] = spaceship2_y;

        // Spaceship 3 initial position
        x_pos[3] = spaceship3_x;
        y_pos[3] = spaceship3_y;
```

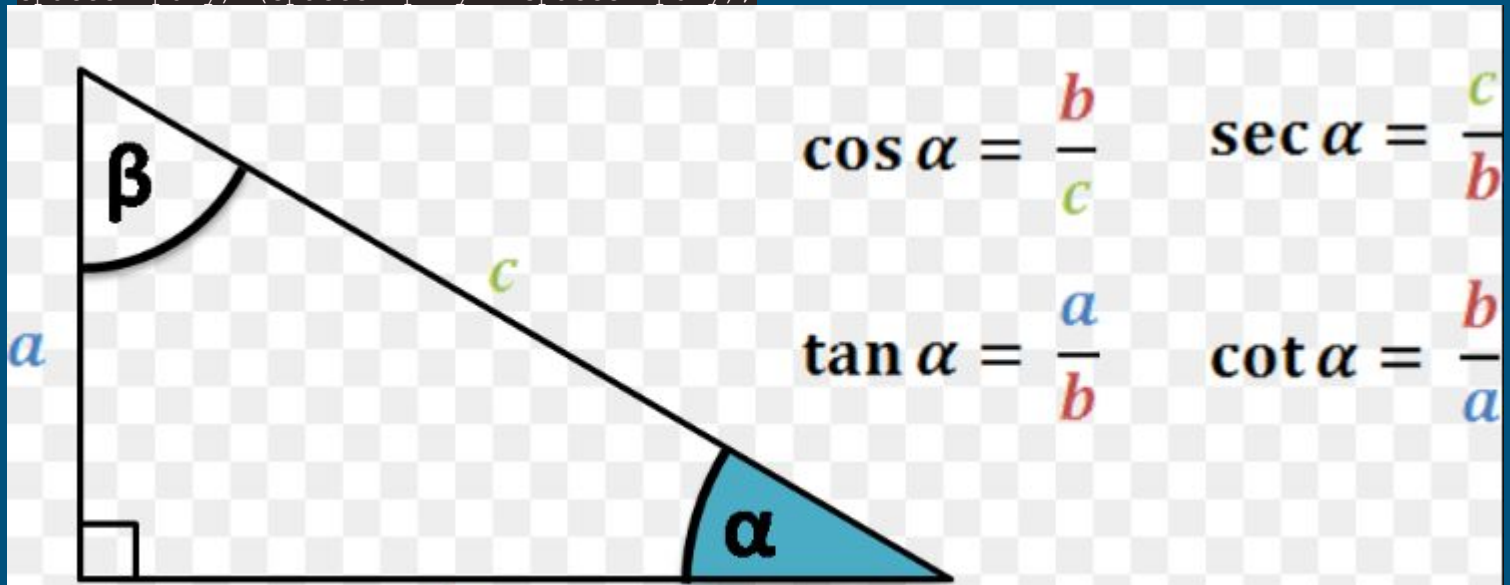


Collision Logic: Bumping State

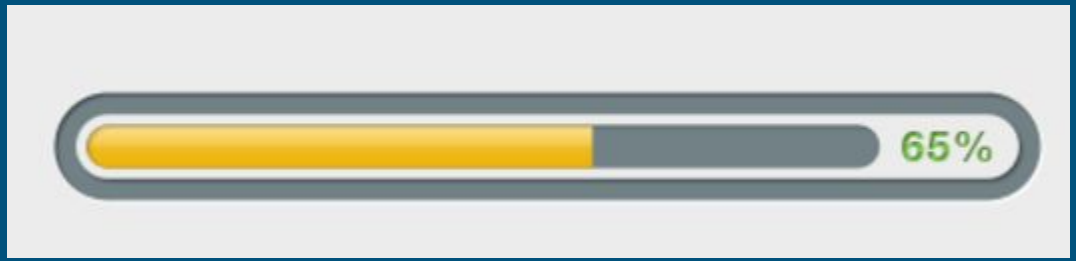
```
    // Initialize projectiles to inactive
    initial begin
        for (i = 0; i < MAX_PROJECTILES; i = i + 1) begin
            projectile_active[i] = 0;
        end
        previous_switch_state = SW[0];
        for (i = 1; i < 9; i = i + 1) begin
            stop_flag[i] = 0;
        end
    end
```

Distance Calculation and Comparison

```
        s2_1 = (spaceship2_x - spaceship_x)*(spaceship2_x - spaceship_x) + (spaceship2_y -
spaceship_y)*(spaceship2_y - spaceship_y);
        s2_5 = (spaceship2_x - spaceship5_x)*(spaceship2_x - spaceship5_x) + (spaceship2_y -
spaceship5_y)*(spaceship2_y - spaceship5_y);
```



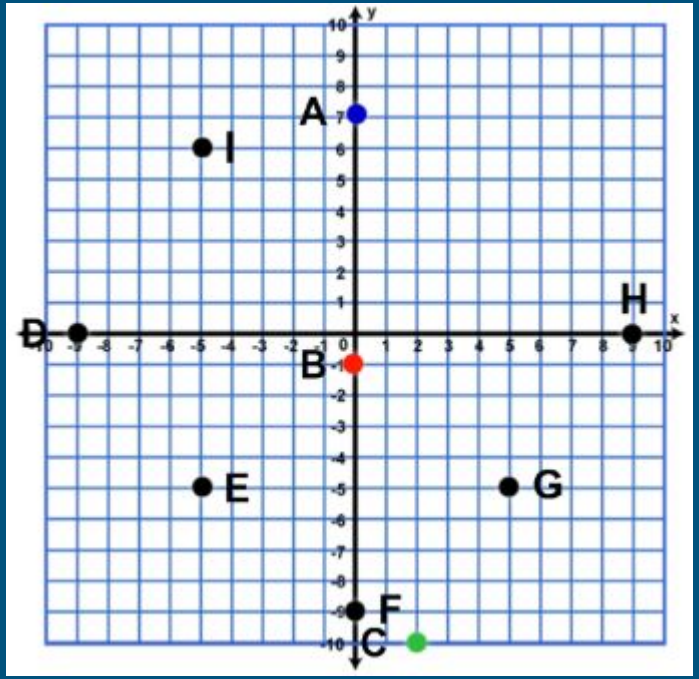
State Differentiation and Progress Bar




```
// progress bar update
if (progress_counter == 0 && stateNumber == 0) begin
    progress_bar <= progress_bar + 1;
end
// if time comes, win
if (progress_bar == finish_length && stateNumber == 0) begin
    stateNumber = 2;
end
```

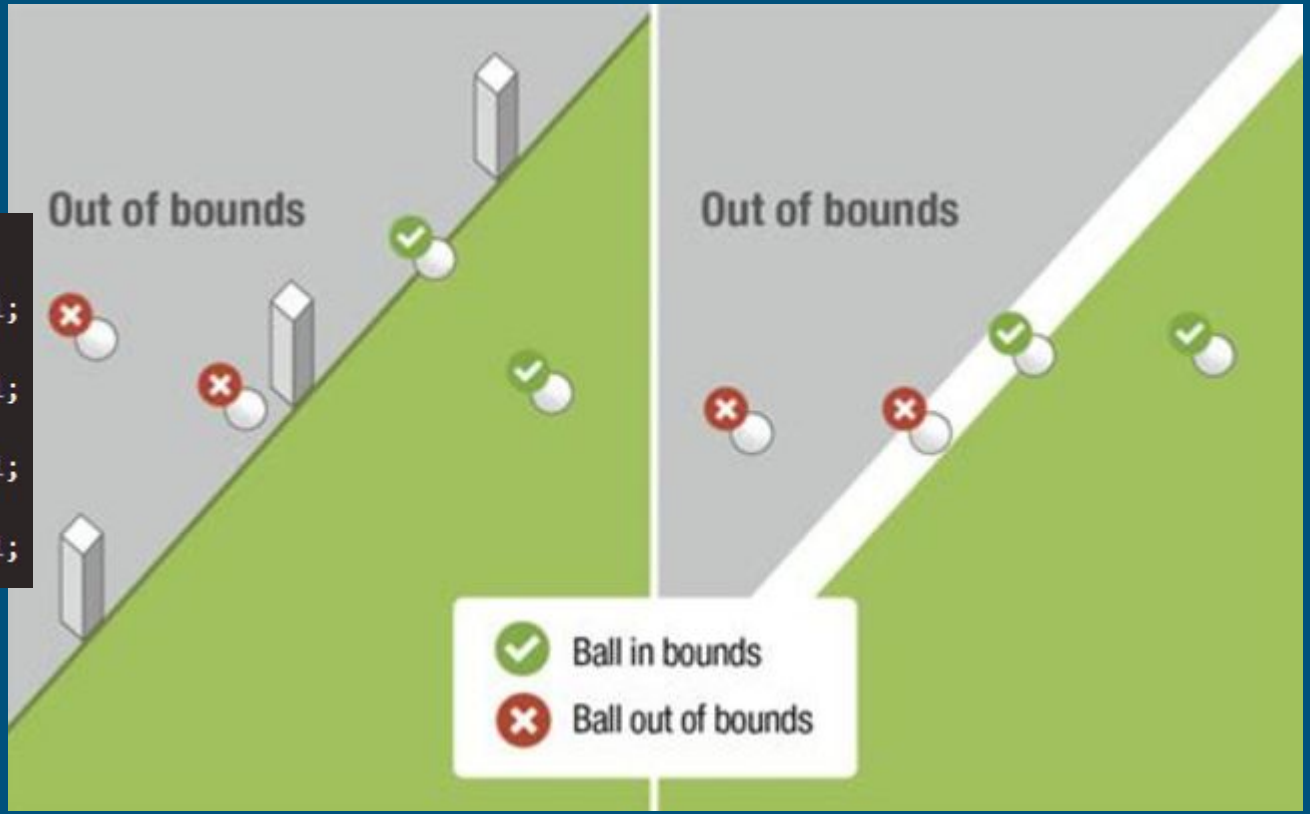
Revival Logic

```
// revive logic
if (ResetGame && stateNumber != 0) begin//reset
```



Player Treversal

```
// Spaceship movement: Player 1
if (W && spaceship_y > 0)
    spaceship_y <= spaceship_y - 1;
if (S && spaceship_y < 112)
    spaceship_y <= spaceship_y + 1;
if (A && spaceship_x > 0)
    spaceship_x <= spaceship_x - 1;
if (D && spaceship_x < 152)
    spaceship_x <= spaceship_x + 1;
```



Adverse Heuristics

```
if (stop_flag[1] == 0) begin
    // Second spaceship moves towards the firs
    if (s2_1 < s2_5) begin
        // update chaser 2 position if closer
        if (spaceship2_x < spaceship_x)
            spaceship2_x <= spaceship2_x + 1;
        else if (spaceship2_x > spaceship_x)
            spaceship2_x <= spaceship2_x - 1;
        if (spaceship2_y < spaceship_y)
            spaceship2_y <= spaceship2_y + 1;
        else if (spaceship2_y > spaceship_y)
            spaceship2_y <= spaceship2_y - 1;
    end else begin
        if (spaceship2_x < spaceship5_x)
            spaceship2_x <= spaceship2_x + 1;
        else if (spaceship2_x > spaceship5_x)
            spaceship2_x <= spaceship2_x - 1;

        if (spaceship2_y < spaceship5_y)
            spaceship2_y <= spaceship2_y + 1;
        else if (spaceship2_y > spaceship5_y)
            spaceship2_y <= spaceship2_y - 1;
    end
```

Collision Detection & Sigal update

```
for (i = 1; i <9; i = i + 1) begin
    for (j = 1; j < 9; j = j + 1) begin
        // increase threshold value
        if ((x_pos[i] >= x_pos[j] - CLOSE_THRESHOLD && x_pos[i] <= x_pos[j] + CLOSE_THRESHOLD) &&
        (y_pos[i] >= y_pos[j] - CLOSE_THRESHOLD && y_pos[i] <= y_pos[j] + CLOSE_THRESHOLD) && stateNumber == 0 && i != j) begin

            if (stop_flag[j] == 0) begin
                stop_flag[i] <= stop_flag[i] + 1;
            end


            // caught and die
            if (i == 1 || i == 5 || j == 1 || j == 5) begin
                progress_bar <= 0;

                spaceship_x <= 72;   // Initial x position of first
                spaceship_y <= 52;   // Initial y position of first
```

Death Logic

```
// caught and die
if (i == 1 || i == 5 || j == 1 || j == 5) begin
```

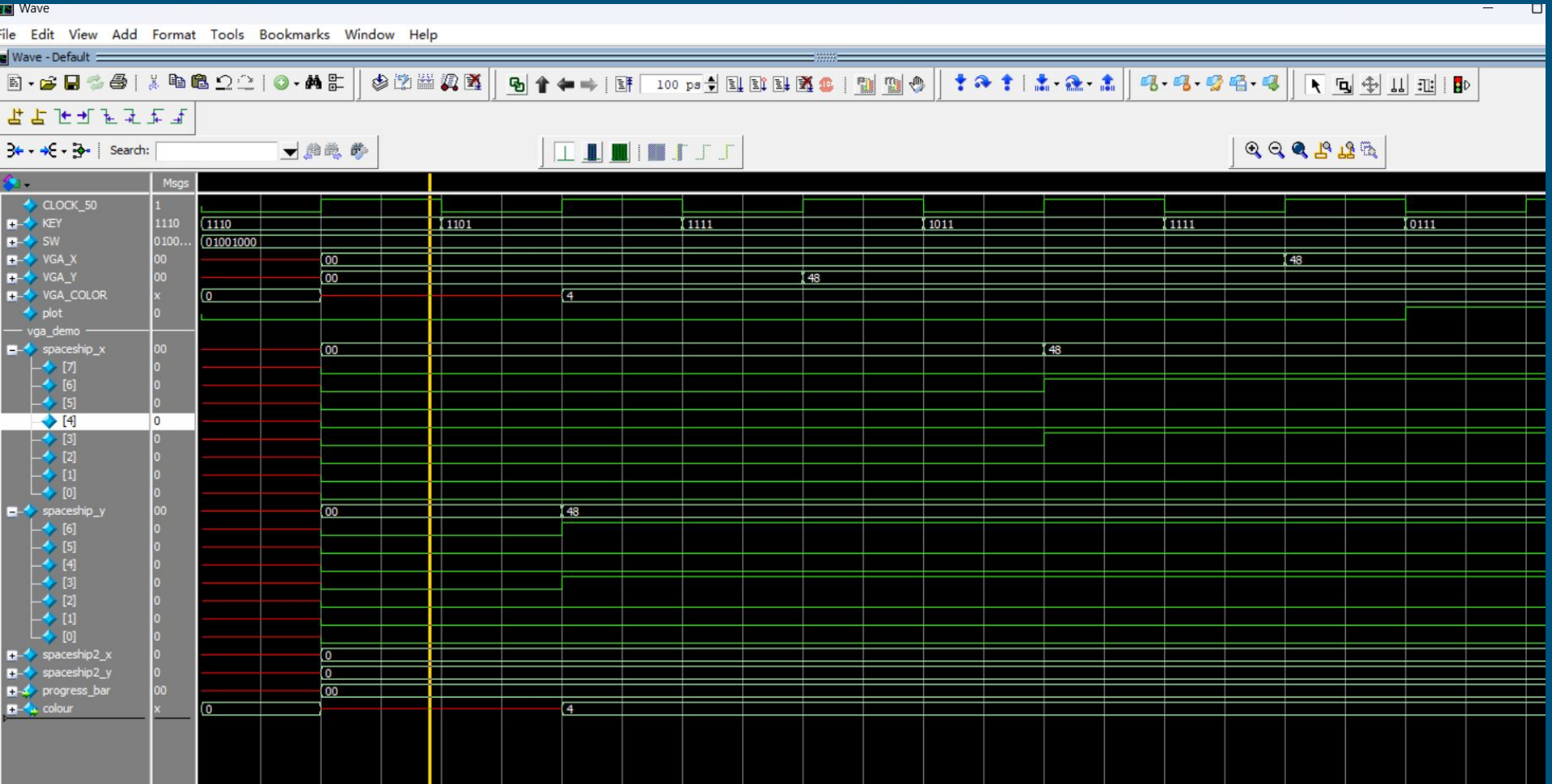
- Bar
- Position
- State

VGA Display: Drawing Logic

1. State Recognition
2. BG update
3. Object drawing Logic

```
// Player 1 ship
if (current_x >= spaceship_x && current_x < spaceship_x + 8 &&
    current_y >= spaceship_y && current_y < spaceship_y + 8) begin
    // Check the spaceship pattern to decide if this pixel should be filled
    colour <= spaceship_pattern[current_y - spaceship_y][current_x - spaceship_x];
end
```

Modelsim


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
CLOCK_50
KEY
SW
VGA_X
VGA_Y
VGA_COLOR
plot
vga_demo
spaceship_x
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
spaceship_y
[6]
[5]
[4]
[3]
[2]
[1]
[0]
spaceship2_x
spaceship2_y
progress_bar
colour
1110
1101
1111
1011
1111
0111
01001000
00
48
4
0

Future work

1.Beautify the cover page and closing pattern

2.Reduce the redundant logic cycle in keyboard (reset)

3.

Work distribution: Yimin

Yimin Chu:

1.Using array to store the objects' pattern

2.Use two loop to transverse the screen to paint the object

3.Build a inner counter to make the speed of object can be captured by eyes

4.Using the if else to control build a equivalent FSM, which make sure we can access two or more state of FSM.

Work distribution: ZHongui

Game State Control

Collision Detection Mechanism
Automated Enemy Behavior
Game State Control

Collision Detection Mechanism
Automated Enemy Behavior