Milestone 2 report

- Keyboard control

PS2_Controller to handle communication with the keyboard. Key scan codes are received via the received_data signal, with received_data_en indicating when new data is available.

We use a break code mechanism to differentiate between press and release events, specifically break code (8'hF0) signals that make the next scan code correspond to a key release.

We assigned several variables to manage the key state,w_pressed, a_pressed, s_pressed, d_pressed, and space_pressed. Below is how they operate:
On detecting a key press, the corresponding signal is set to 1.
On detecting a key release, the corresponding signal is set to 0.

Issue:
The first and second movement in x direction can work as expected, while it arrives at the terminal of the second contour, the ball starts to debounce. Same issue happened in y direction.

Tried:
We try to change testbench.v and use `$display("Received Data: %h", received_data);`

- Second player

Implementation:
The second ship (spaceship2) moves towards the first ship (spaceship) using a basic pursuit algorithm.If spaceship2's X or Y coordinate is less than the first ship's, it increments its position.
If greater, it decrements its position.

Progress:
Fully implemented. The second ship adjusts its position in both X and Y directions during every movement cycle (movement_counter).

- Changing background

Implementation:
A "DIE" pattern is displayed on the screen, indicating the game-over condition.
Progress: Basic death logic is complete and integrates with VGA rendering.

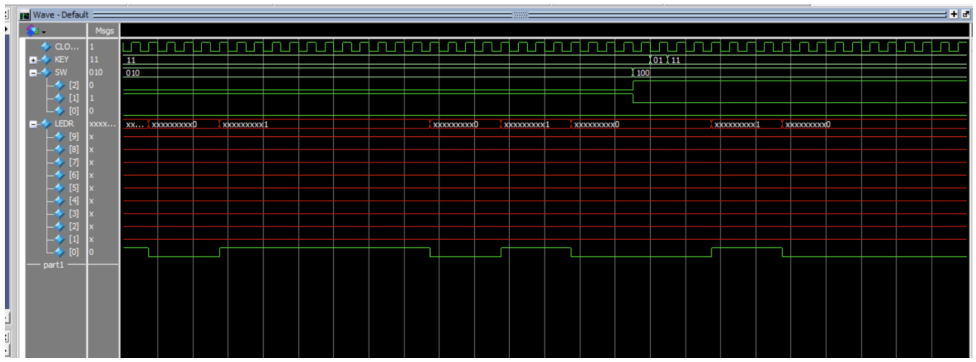
- Collision detection

Implementation:
Collision detection is performed using CLOSE_THRESHOLD.If the distance between the two ships in both X and Y coordinates is within this threshold, a collision is registered. Active projectiles are checked against spaceship2's position to determine hits, which deactivate the projectile upon collision.

Progress:
Fully functional for both ship-to-ship collisions and projectile impacts.

Death Logic:
Implementation: When spaceship and spaceship2 collide:The game transitions to a "death state" (stateNumber = 1).

Modelsim



```
module vga_demo(
    input CLOCK_50,
    // Removed the switch input as it's no longer needed for firing
    // input [0:0] SW,           // Switch 0 for firing (Removed)
    inout PS2_CLK,
    inout PS2_DAT,
    output [7:0] VGA_R, VGA_G, VGA_B,
    output VGA_HS, VGA_VS, VGA_BLANK_N, VGA_SYNC_N, VGA_CLK,
     output [2:0]LEDR
```

```
);


    // Define an 8x8 spaceship pattern with colors
    reg [2:0] spaceship_pattern [0:7][0:7];

    // Initialize the spaceship pattern with a gun at the tip
    initial begin
        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;
    end

    // Spaceship position registers, initialized to center of 160x120
resolution
    reg [7:0] spaceship_x = 80;   // Initial x position
    reg [6:0] spaceship_y = 60;   // Initial y position

    // Projectile properties
    parameter MAX_PROJECTILES = 10;
    reg [7:0] projectile_x [0:MAX_PROJECTILES-1];
    reg [6:0] projectile_y [0:MAX_PROJECTILES-1];
    reg projectile_active [0:MAX_PROJECTILES-1];

    // VGA position signals
    reg [7:0] current_x;
    reg [6:0] current_y;
    reg [2:0] colour;

    // Available slot for projectile
    reg found_slot;

     // Movement flags based on keyboard input
```

```
    reg break_code_received;
    reg [7:0] last_received_data;
    reg w_pressed;
    reg a_pressed;
    reg s_pressed;
    reg d_pressed;
    reg space_pressed;

    // Debounce logic and movement counter
    reg [19:0] counter;
    integer i;

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

    // PS/2 Keyboard Interface using PS2_Controller
    wire [7:0] received_data;
    wire received_data_en;
    wire command_was_sent;
    wire error_communication_timed_out;

    PS2_Controller #(.INITIALIZE_MOUSE(0)) PS2 (
        .CLOCK_50(CLOCK_50),
        .reset(1'b0),
        .the_command(8'h00),
        .send_command(1'b0),
        .PS2_CLK(PS2_CLK),
        .PS2_DAT(PS2_DAT),
        .command_was_sent(command_was_sent),
        .error_communication_timed_out(error_communication_timed_out),
        .received_data(received_data),
        .received_data_en(received_data_en)
    );

     assign LEDR[0] = w_pressed;// for debug
```

```
    // Shift Register to Capture Key Presses
    // We'll use a 16-bit shift register to store the last 4 key presses
(4 bits each)
    //reg [15:0] key_shift_reg;

    // Process scan codes to set movement flags and update shift register
    always @(posedge CLOCK_50) begin
    if (received_data_en) begin
        // Add debugging output
        $display("Received Data: %h", received_data);
        if (received_data == 8'hF0) begin
            // Break code received, next code will be key that was
released
            break_code_received <= 1;
        end else begin
            if (break_code_received) begin
                // Key release
                break_code_received <= 0;
                case (received_data)
                    8'h1D: w_pressed <= 0; // W key released
                    8'h1C: a_pressed <= 0; // A key released
                    8'h1B: s_pressed <= 0; // S key released
                    8'h23: d_pressed <= 0; // D key released
                    8'h29: space_pressed <= 0; // Spacebar released
                    default: ;
                endcase
            end else begin
                // Key press
                case (received_data)
                    8'h1D: w_pressed <= 1; // W key pressed
                    8'h1C: a_pressed <= 1; // A key pressed
                    8'h1B: s_pressed <= 1; // S key pressed
                    8'h23: d_pressed <= 1; // D key pressed
                    8'h29: space_pressed <= 1; // Spacebar pressed
                    default: ;
                endcase
            end
        end
    end
end
```

```
    // Update spaceship and handle firing based on key_shift_reg and
Spacebar
    always @(posedge CLOCK_50) begin
    counter <= counter + 1;
    if (counter == 0) begin
        // Update spaceship position based on movement flags
        if (w_pressed && spaceship_y > 0)
            spaceship_y <= spaceship_y - 1;
        if (s_pressed && spaceship_y < 112) // 120 - 8 (spaceship height)
            spaceship_y <= spaceship_y + 1;
        if (a_pressed && spaceship_x > 0)
            spaceship_x <= spaceship_x - 1;
        if (d_pressed && spaceship_x < 152) // 160 - 8 (spaceship width)
            spaceship_x <= spaceship_x + 1;

        // Handle projectile firing based on Spacebar press
        if (space_pressed) begin
            found_slot = 0;
            for (i = 0; i < MAX_PROJECTILES && !found_slot; i = i + 1)
begin
                if (!projectile_active[i]) begin
                    projectile_active[i] <= 1;
                    projectile_x[i] <= spaceship_x + 3;
                    projectile_y[i] <= spaceship_y - 1;
                    found_slot = 1;
                end
            end
        end

        // Move each active projectile
        for (i = 0; i < MAX_PROJECTILES; i = i + 1) begin
            if (projectile_active[i]) begin
                if (projectile_y[i] > 0)
                    projectile_y[i] <= projectile_y[i] - 1;
                else
                    projectile_active[i] <= 0;
            end
        end
    end
```

```
end
    // VGA Adapter instantiation with color logic for spaceship,
projectiles, and background
    always @(posedge CLOCK_50) begin
        // Update current_x and current_y to scan through the display
        if (current_x == 159) begin
            current_x <= 0;
            if (current_y == 119)
                current_y <= 0;
            else
                current_y <= current_y + 1;
        end else begin
            current_x <= current_x + 1;
        end

        // Set color based on whether the current pixel is within the
spaceship or projectile area
        colour <= 3'b000; // Default background color

        // Check if current pixel is within any active projectile
        for (i = 0; i < MAX_PROJECTILES; i = i + 1) begin
            if (projectile_active[i] && current_x == projectile_x[i] &&
current_y == projectile_y[i]) begin
                colour <= 3'b111; // White color for projectile
            end
        end

        // Check if current pixel is within the spaceship area
        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
    end

    vga_adapter VGA (
        .resetn(1'b1),             // No reset on VGA adapter
        .clock(CLOCK_50),
```

```
        .colour(colour),
        .x(current_x),
        .y(current_y),
        .plot(1'b1),                // Constant plot signal
        .VGA_R(VGA_R),
        .VGA_G(VGA_G),
        .VGA_B(VGA_B),
        .VGA_HS(VGA_HS),
        .VGA_VS(VGA_VS),
        .VGA_BLANK_N(VGA_BLANK_N),
        .VGA_SYNC_N(VGA_SYNC_N),
        .VGA_CLK(VGA_CLK)
    );

    // VGA adapter configuration parameters
    defparam VGA.RESOLUTION = "160x120";
    defparam VGA.MONOCHROME = "FALSE";
    defparam VGA.BITS_PER_COLOUR_CHANNEL = 1;
    defparam VGA.BACKGROUND_IMAGE = "NONE"; // Black background

endmodule

/*// PS/2 Keyboard Interface Module
module ps2_keyboard(
    input wire clk,           // System clock
    input wire reset,
    input wire ps2_clk,
    input wire ps2_dat,
    output reg [7:0] received_data,
    output reg received_data_en
);

    reg [7:0] ps2_clk_sync;
    reg [7:0] ps2_dat_sync;

    // Synchronize PS2 signals to system clock
    always @(posedge clk) begin
        ps2_clk_sync <= {ps2_clk_sync[6:0], ps2_clk};
        ps2_dat_sync <= {ps2_dat_sync[6:0], ps2_dat};
    end
```

```
    wire ps2_clk_falling_edge = (ps2_clk_sync[7:1] == 7'b1111111) &&
(ps2_clk_sync[0] == 0);

    reg [3:0] bit_count;
    reg [10:0] shift_reg;

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            bit_count <= 0;
            shift_reg <= 0;
            received_data_en <= 0;
        end else begin
            if (received_data_en)
                received_data_en <= 0; // Reset the flag

            if (ps2_clk_falling_edge) begin
                if (bit_count < 11) begin
                    shift_reg[bit_count] <= ps2_dat_sync[7]; // Shift in
data bit
                    bit_count <= bit_count + 1;
                end else begin
                    bit_count <= 0;
                    // Extract data bits (bits 1 to 8)
                    received_data <= {shift_reg[8], shift_reg[7],
shift_reg[6], shift_reg[5],
                                      shift_reg[4], shift_reg[3],
shift_reg[2], shift_reg[1]};
                    received_data_en <= 1;
                end
            end
        end
    end

endmodule*/
```

```
module vga_demo(
    input CLOCK_50,
    input [3:0] KEY,
    input [0:0] SW,          // Switch 0 for firing
    output [7:0] VGA_R, VGA_G, VGA_B,
    output VGA_HS, VGA_VS, VGA_BLANK_N, VGA_SYNC_N, VGA_CLK
);

    // Define an 8x8 spaceship pattern with colors
    reg [2:0] spaceship_pattern [0:7][0:7];
    reg [2:0] spaceship2_pattern [0:7][0:7]; // Define a new pattern for spaceship2
    reg [2:0] skull_pattern [0:7][0:7]; // Define a new pattern for a skull face
    reg [7:0] die_pattern [0:27]; // 28 columns of 8 bits each for "DIE"

    // Initialize the spaceship pattern with a gun at the tip
    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;

        // New spaceship pattern for spaceship2 (Red spaceship)
        spaceship2_pattern[0][0] = 3'b000; spaceship2_pattern[0][1] = 3'b000;
spaceship2_pattern[0][2] = 3'b100; spaceship2_pattern[0][3] = 3'b100;
        spaceship2_pattern[0][4] = 3'b100; spaceship2_pattern[0][5] = 3'b100;
spaceship2_pattern[0][6] = 3'b000; spaceship2_pattern[0][7] = 3'b000;
        spaceship2_pattern[1][0] = 3'b000; spaceship2_pattern[1][1] = 3'b100;
spaceship2_pattern[1][2] = 3'b100; spaceship2_pattern[1][3] = 3'b100;
        spaceship2_pattern[1][4] = 3'b100; spaceship2_pattern[1][5] = 3'b100;
spaceship2_pattern[1][6] = 3'b100; spaceship2_pattern[1][7] = 3'b000;
        spaceship2_pattern[2][0] = 3'b100; spaceship2_pattern[2][1] = 3'b100;
spaceship2_pattern[2][2] = 3'b100; spaceship2_pattern[2][3] = 3'b100;
        spaceship2_pattern[2][4] = 3'b100; spaceship2_pattern[2][5] = 3'b100;
spaceship2_pattern[2][6] = 3'b100; spaceship2_pattern[2][7] = 3'b100;
        spaceship2_pattern[3][0] = 3'b100; spaceship2_pattern[3][1] = 3'b100;
spaceship2_pattern[3][2] = 3'b100; spaceship2_pattern[3][3] = 3'b111;
        spaceship2_pattern[3][4] = 3'b111; spaceship2_pattern[3][5] = 3'b100;
spaceship2_pattern[3][6] = 3'b100; spaceship2_pattern[3][7] = 3'b100;
        spaceship2_pattern[4][0] = 3'b100; spaceship2_pattern[4][1] = 3'b100;
spaceship2_pattern[4][2] = 3'b111; spaceship2_pattern[4][3] = 3'b111;
        spaceship2_pattern[4][4] = 3'b111; spaceship2_pattern[4][5] = 3'b111;
spaceship2_pattern[4][6] = 3'b100; spaceship2_pattern[4][7] = 3'b100;
```

```
        spaceship2_pattern[5][0] = 3'b100; spaceship2_pattern[5][1] = 3'b111;
spaceship2_pattern[5][2] = 3'b111; spaceship2_pattern[5][3] = 3'b111;
        spaceship2_pattern[5][4] = 3'b111; spaceship2_pattern[5][5] = 3'b111;
spaceship2_pattern[5][6] = 3'b111; spaceship2_pattern[5][7] = 3'b100;
        spaceship2_pattern[6][0] = 3'b000; spaceship2_pattern[6][1] = 3'b111;
spaceship2_pattern[6][2] = 3'b111; spaceship2_pattern[6][3] = 3'b111;
        spaceship2_pattern[6][4] = 3'b111; spaceship2_pattern[6][5] = 3'b111;
spaceship2_pattern[6][6] = 3'b111; spaceship2_pattern[6][7] = 3'b000;
        spaceship2_pattern[7][0] = 3'b000; spaceship2_pattern[7][1] = 3'b000;
spaceship2_pattern[7][2] = 3'b111; spaceship2_pattern[7][3] = 3'b111;
        spaceship2_pattern[7][4] = 3'b111; spaceship2_pattern[7][5] = 3'b111;
spaceship2_pattern[7][6] = 3'b000; spaceship2_pattern[7][7] = 3'b000;



        // D
        die_pattern[0]  = 8'b11111100;
        die_pattern[1]  = 8'b11000110;
        die_pattern[2]  = 8'b11000011;
        die_pattern[3]  = 8'b11000011;
        die_pattern[4]  = 8'b11000011;
        die_pattern[5]  = 8'b11000011;
        die_pattern[6]  = 8'b11000110;
        die_pattern[7]  = 8'b11111100;
        die_pattern[8]  = 8'b00000000; // Gap between letters

        // I
        die_pattern[9]  = 8'b11111111;
        die_pattern[10] = 8'b00011000;
        die_pattern[11] = 8'b00011000;
        die_pattern[12] = 8'b00011000;
        die_pattern[13] = 8'b00011000;
        die_pattern[14] = 8'b00011000;
        die_pattern[15] = 8'b00011000;
        die_pattern[16] = 8'b11111111;
        die_pattern[17] = 8'b00000000; // Gap between letters
```

```
    // E
    die_pattern[18] = 8'b11111111;
    die_pattern[19] = 8'b11000000;
    die_pattern[20] = 8'b11000000;
    die_pattern[21] = 8'b11111100;
    die_pattern[22] = 8'b11111100;
    die_pattern[23] = 8'b11000000;
    die_pattern[24] = 8'b11000000;
    die_pattern[25] = 8'b11111111;
    die_pattern[26] = 8'b00000000; // Gap after letter
    die_pattern[27] = 8'b00000000;

end

// Spaceship position registers, initialized to center of 160x120 resolution
reg [7:0] spaceship_x = 80;   // Initial x position of first spaceship
reg [6:0] spaceship_y = 60;   // Initial y position of first spaceship

// Second spaceship position
reg [7:0] spaceship2_x = 0;  // Initial x position of second spaceship
reg [6:0] spaceship2_y = 0;  // Initial y position of second spaceship

reg [7:0] skull_x = 40;   // Initial x position of skull
reg [6:0] skull_y = 30;   // Initial y position of skull

// Projectile properties
parameter MAX_PROJECTILES = 10;
reg [7:0] projectile_x [0:MAX_PROJECTILES-1];
reg [6:0] projectile_y [0:MAX_PROJECTILES-1];
reg projectile_active [0:MAX_PROJECTILES-1];

// Track previous switch state for detecting changes
reg previous_switch_state;

// VGA position signals
reg [7:0] current_x;
reg [6:0] current_y;
```

```
    reg [2:0] colour;

    // Debounce logic and movement counter
    reg [19:0] counter;
    reg [21:0] movement_counter;
    integer i;
    reg found_slot;
    parameter CLOSE_THRESHOLD = 5;
    reg stateNumber = 0;

    // 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];
    end

    always @(posedge CLOCK_50) begin
        counter <= counter + 1;
        movement_counter <= movement_counter + 1;
        if (counter == 0) begin
            // Spaceship movement
            if (!KEY[0] && spaceship_y > 0)          // Move up
                spaceship_y <= spaceship_y - 1;
            if (!KEY[1] && spaceship_y < 112)        // Move down (112 to keep within
bounds)
                spaceship_y <= spaceship_y + 1;
            if (!KEY[2] && spaceship_x > 0)          // Move left
                spaceship_x <= spaceship_x - 1;
            if (!KEY[3] && spaceship_x < 152)        // Move right (152 to keep within
bounds)
                spaceship_x <= spaceship_x + 1;

            if (movement_counter == 0) begin
            // Second spaceship moves towards the first spaceship
            if (spaceship2_x < spaceship_x)          // Move right to catch the first spaceship
```

```
            spaceship2_x <= spaceship2_x + 1;
        else if (spaceship2_x > spaceship_x)      // Move left to catch the first spaceship
            spaceship2_x <= spaceship2_x - 1;

        if (spaceship2_y < spaceship_y)           // Move down to catch the first spaceship
            spaceship2_y <= spaceship2_y + 1;
        else if (spaceship2_y > spaceship_y)      // Move up to catch the first spaceship
            spaceship2_y <= spaceship2_y - 1;
        end

        if ((spaceship_x >= spaceship2_x - CLOSE_THRESHOLD && spaceship_x <=
spaceship2_x + CLOSE_THRESHOLD) &&
            (spaceship_y >= spaceship2_y - CLOSE_THRESHOLD && spaceship_y <=
spaceship2_y + CLOSE_THRESHOLD))

            stateNumber = 1;

        // Detect switch state change to trigger a new projectile
        if (SW[0] != previous_switch_state) begin
            found_slot = 0;

            // Find an inactive projectile slot
            for (i = 0; i < MAX_PROJECTILES && !found_slot; i = i + 1) begin
                if (!projectile_active[i]) begin
                    projectile_active[i] <= 1;
                    projectile_x[i] <= spaceship_x + 3; // Position at the spaceship tip
                    projectile_y[i] <= spaceship_y - 1;
                    found_slot = 1; // Set flag to stop further activation in this cycle
                end
            end
            previous_switch_state <= SW[0]; // Update the previous switch state
        end

        // Move each active projectile
        for (i = 0; i < MAX_PROJECTILES; i = i + 1) begin
            if (projectile_active[i]) begin
```

```
            if ((projectile_x[i] >= spaceship2_x - CLOSE_THRESHOLD &&
projectile_x[i] <= spaceship2_x + CLOSE_THRESHOLD) &&
                (projectile_y[i] >= spaceship2_y - CLOSE_THRESHOLD &&
projectile_y[i] <= spaceship2_y + CLOSE_THRESHOLD))
                    projectile_active[i] <= 0; // Deactivate if within close range


            if (projectile_y[i] > 0 && projectile_y[i] < 120 && projectile_x[i] > 0 &&
projectile_x[i] < 160 && projectile_x[i] ) begin
                // Update projectile x position towards spaceship2
                if (projectile_x[i] < spaceship2_x)
                    projectile_x[i] <= projectile_x[i] + 1;
                else if (projectile_x[i] > spaceship2_x)
                    projectile_x[i] <= projectile_x[i] - 1;

                // Update projectile y position towards spaceship2
                if (projectile_y[i] < spaceship2_y)
                    projectile_y[i] <= projectile_y[i] + 1;
                else if (projectile_y[i] > spaceship2_y)
                    projectile_y[i] <= projectile_y[i] - 1; // Removed the erroneous
semicolon here
            end
            else
                projectile_active[i] <= 0; // Deactivate if it goes off-screen
        end
    end
end

// VGA Adapter instantiation with color logic for spaceship, projectiles, and
background
always @(posedge CLOCK_50) begin
    // Update current_x and current_y to scan through the display
    if (current_x == 159) begin
        current_x <= 0;
        if (current_y == 119)
            current_y <= 0;
```

```
            else
                current_y <= current_y + 1;
        end else begin
            current_x <= current_x + 1;
        end

        // Set color based on whether the current pixel is within the spaceship or projectile
area
        colour <= 3'b000; // Default background color

        if (stateNumber == 0)begin
        // Check if current pixel is within any active projectile
        for (i = 0; i < MAX_PROJECTILES; i = i + 1) begin
            if (projectile_active[i] && current_x == projectile_x[i] && current_y ==
projectile_y[i]) begin
                colour <= 3'b111; // White color for projectile
            end
        end

        // Check if current pixel is within the first spaceship area
        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

        // Check if current pixel is within the second spaceship area
        if (current_x >= spaceship2_x && current_x < spaceship2_x + 8 &&
            current_y >= spaceship2_y && current_y < spaceship2_y + 8) begin
            // Check the spaceship2 pattern to decide if this pixel should be filled
            colour <= spaceship2_pattern[current_y - spaceship2_y][current_x -
spaceship2_x];
        end
        end

if (stateNumber == 1) begin
    // Set a default background color (e.g., red)
```

```
    colour <= 3'b000; // Red background

    // Check if the current pixel is within the "DIE" pattern area
    if (current_x >= skull_x && current_x < skull_x + 28 &&
        current_y >= skull_y && current_y < skull_y + 8) begin

        // Check if the corresponding bit in die_pattern is set to 1
        if (die_pattern[current_x - skull_x][7 - current_y - skull_y] == 1'b1) begin
            colour <= 3'b100; // Set to white if the bit is 1
        end
    end
end




    end

    vga_adapter VGA (
        .resetn(1'b1),            // No reset on VGA adapter
        .clock(CLOCK_50),
        .colour(colour),
        .x(current_x),
        .y(current_y),
        .plot(1'b1),              // Constant plot signal
        .VGA_R(VGA_R),
        .VGA_G(VGA_G),
        .VGA_B(VGA_B),
        .VGA_HS(VGA_HS),
        .VGA_VS(VGA_VS),
        .VGA_BLANK_N(VGA_BLANK_N),
        .VGA_SYNC_N(VGA_SYNC_N),
        .VGA_CLK(VGA_CLK)
    );

    // VGA adapter configuration parameters
    defparam VGA.RESOLUTION = "160x120";
```

```
    defparam VGA.MONOCHROME = "FALSE";
    defparam VGA.BITS_PER_COLOUR_CHANNEL = 1;
    defparam VGA.BACKGROUND_IMAGE = "NONE"; // Black background
endmodule
```