EC311 Final Project Presentation: **Whack-a-Mole**

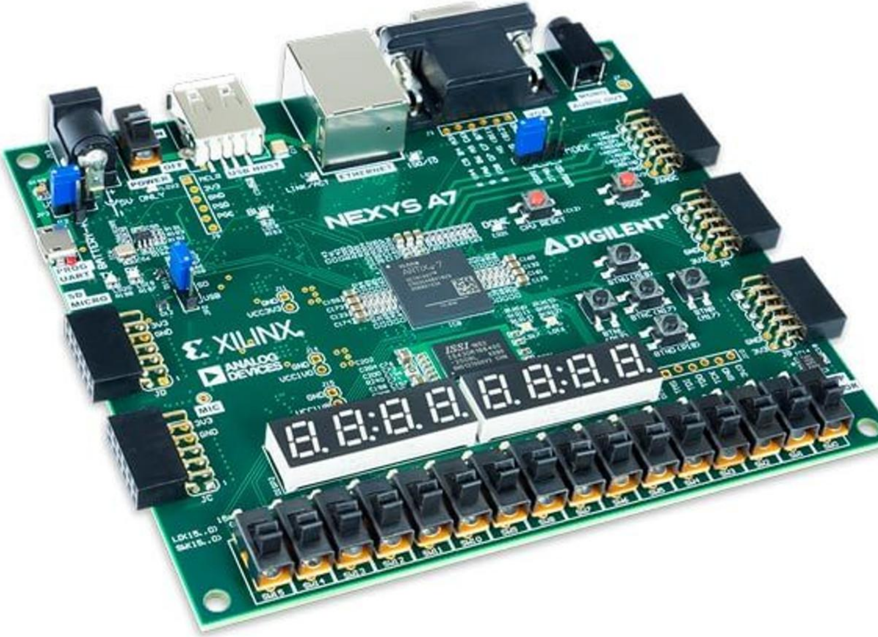
Kofi Frempong, Mingshan Guo,
Arielle Maravilla, Jacob Nissenbaum,
Mirclea Tan

Rubric Reference (Delete this Slide)

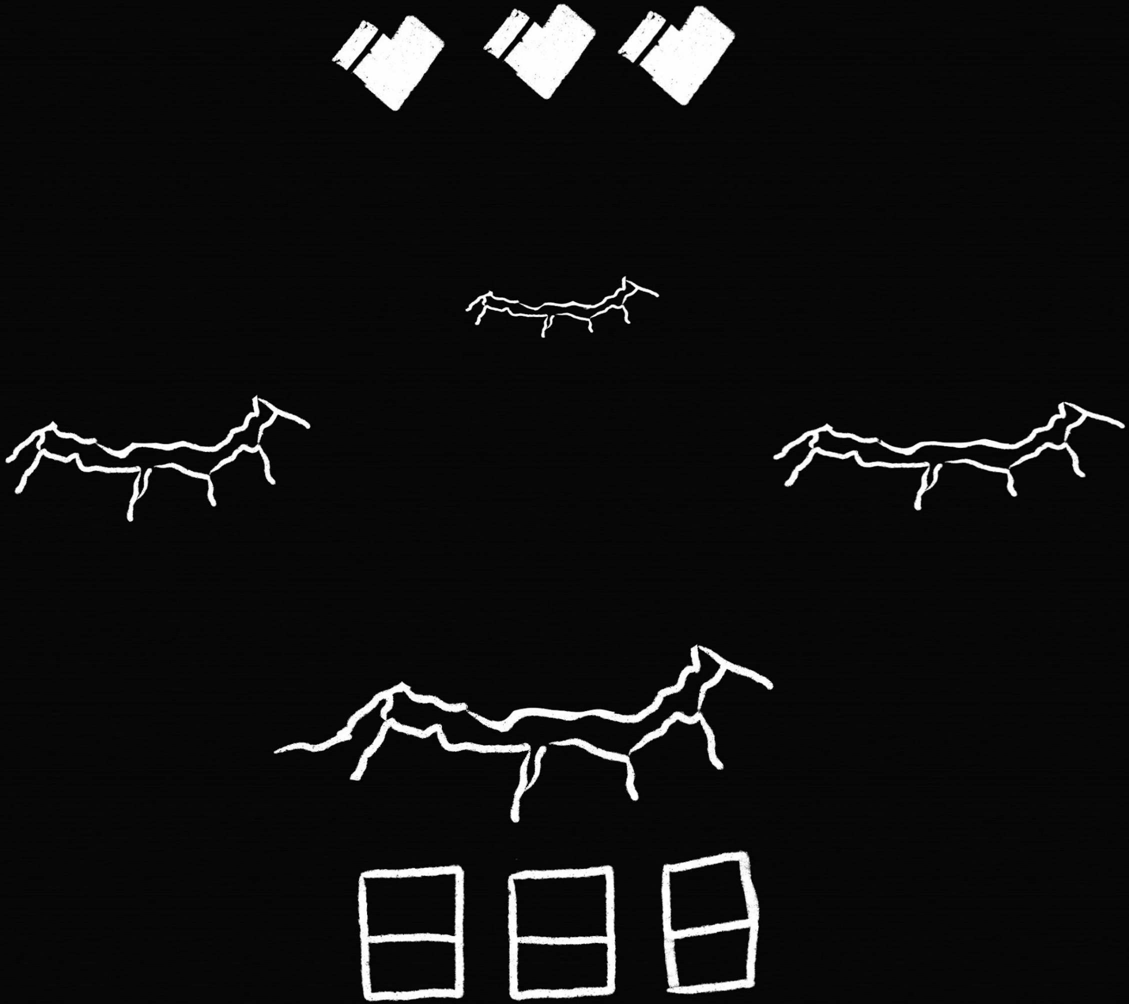
1. ✅ Project title, project members – this should be a single slide
2. ✅ Goal/Motivation –what you are doing and why. This slide should include 1 concrete example of how someone could actually use your design in real life.
3. ✅ Short Functionality – one slide max recapping what you are doing. What was your design supposed to do?
4. ✅ Short Specification – one slide max recapping the specification of the design. What were the requirements? Constraints?
5. ✅ Detailed Block Diagrams – provide the real block diagrams for your design.
6. Code Snippet – provide 1 or 2 of your best code snippets and discuss. What was unique about it? Challenging? Innovative?
7. ✅Successes – discuss how your project was successful and why.
8. Failures – discuss how your project did not work out as you planned. Provide examples of what you would do differently.

We made Whack-a-Mole on an FPGA. Someone could use this as entertainment…

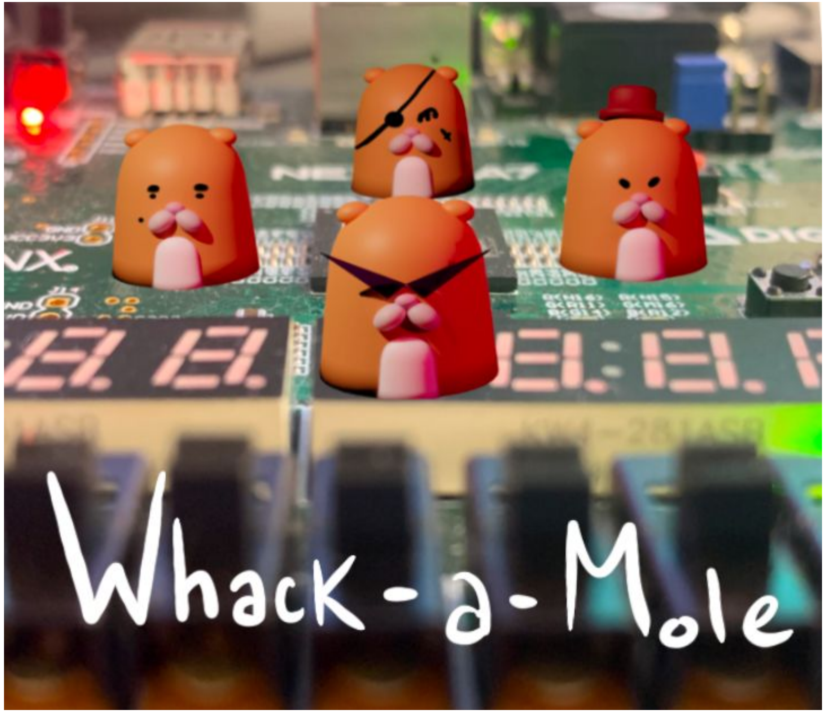
Think of it like an arcade cabinet; you press the button to start the game, the visuals of the game are displayed on the monitor. You interface with the game with the fpga (swing it)



An idea of what this could look like… :)



Also, some box-art…


Whack-a-Mole

Whack-a-Mole

Specifications:
Requirements and Constraints

- Implement a **Random Number Generator** to set time between mole appearances between 1-3 seconds with variability
- 1 second window to hit mole for points, **decreasing** with score
- 1:1 “whack” (via button/motion) per mole
- Reset Switch for resetting statistics (e.g. lives, score)

Block Diagram & Game Logic

Design Doc

Red: FSM

Blue: Logic Pseudo-code

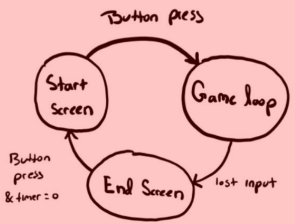
Green: Input, Output, Var.

Purple: Visuals

Yellow: Unknowns

[Whack-a-Mole]

⇒ High-level game logic



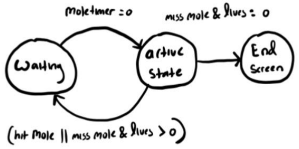
* Will need debouncer or equivalent for inputs

Pseudo-Code

```
Start Screen
    Clear screen
    Display Title
    if button:
        goto game loop

End Screen
    Clear Screen
    Display Score
    if button:
        goto Start Screen

Game loop

Waiting State [
    Turn mole off
    generate rand # called moletimer
    ↳ every clk cycle, decrease # by 1
]
active State [
    ↳ when # = 0 :
        → Mole = on
        → Start hamer timer
            • If button & hammer time != 0 :
                Score ++
                goto waiting State
            • else :
                lives --
                if lives == 0 :
                    goto Endscreen State
                else:
                    goto waiting state
]
```

* * *

// Do we what to sub life if press button & no mole? If yes, need to add this logic



Variables

on-screen
- lives
- points
- Speed (extra?: gets ↑ every 10 pts)

Probably more needed than the ones listed here
- moletimer
- Hammer timer
- mole_on (sprite)

Visuals

↳ in theory, whole game could be on board & mole and public var are led [7-seg display] * Could have a leader board?

↳ Step up would have the UI on screen & mole be glowing dot / sprite

↳ Dream would be animations & sound Effects

] Convert Input from button to accelerometer

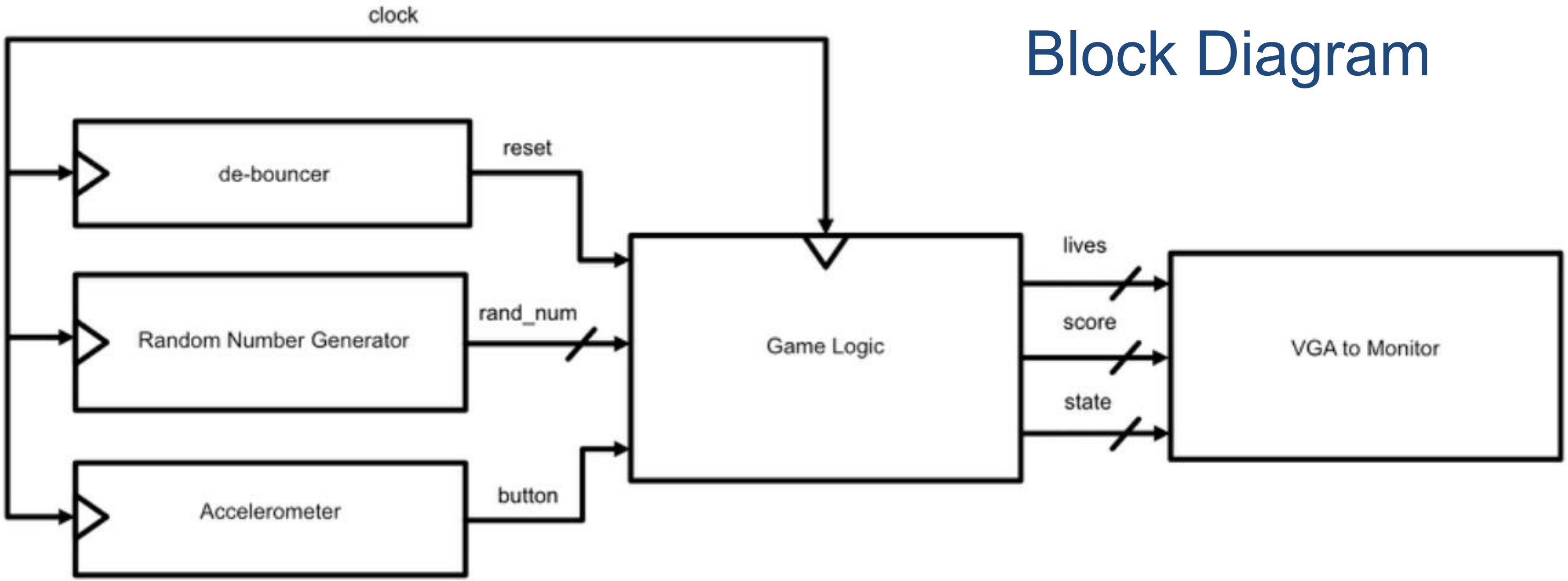
UnKnowns

⇒ How many moles?

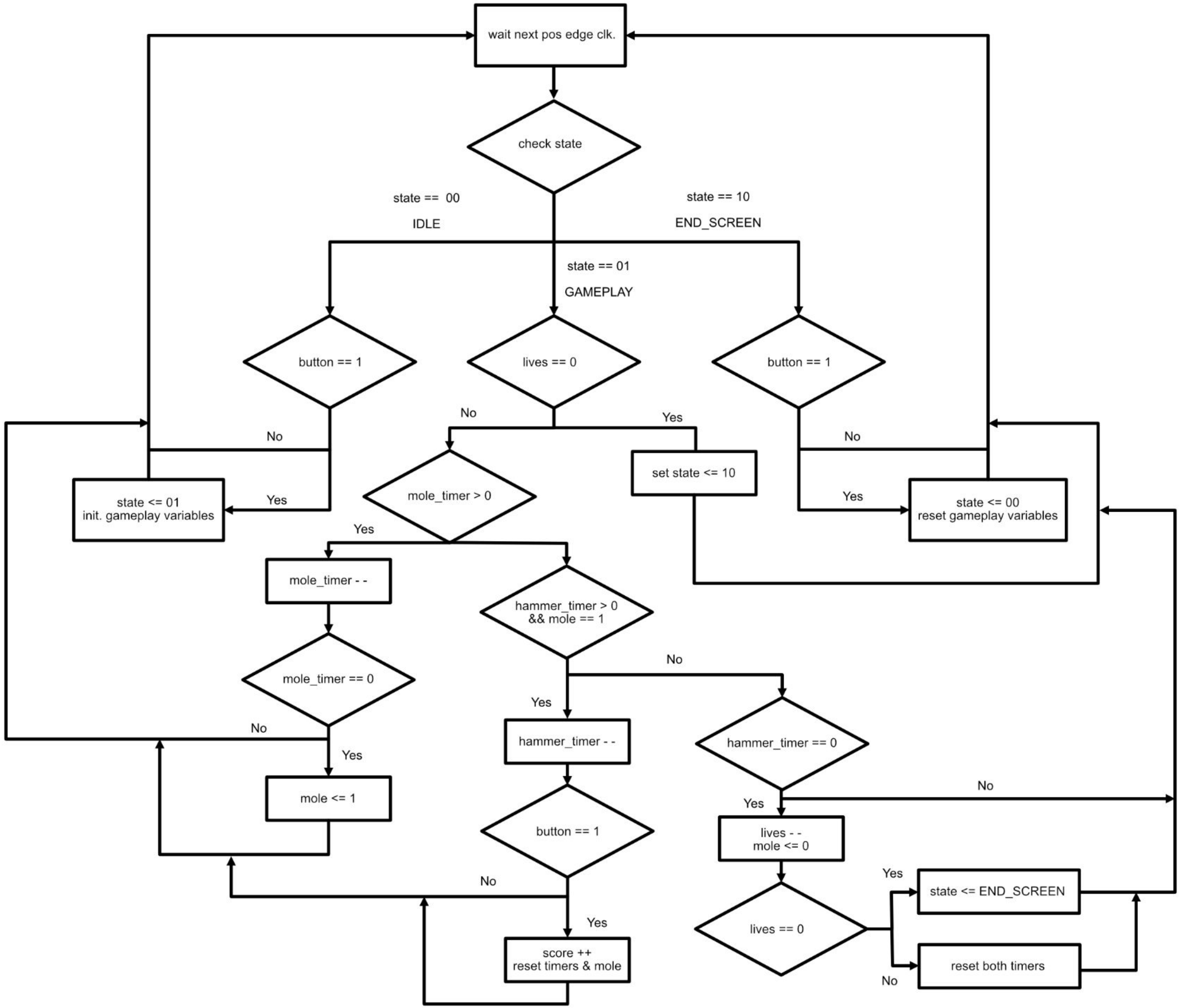
↳ one mole? multiple moles?

↳ If multiple moles, how to implement w/ button or accelerometer?

Block Diagram



Game Logic Diagram



wait next pos edge clk.

check state

state == 00

IDLE

state == 01

GAMEPLAY

state == 10

END_SCREEN

button == 1

No

Yes

state <= 01
init. gameplay variables

lives == 0

No

Yes

set state <= 10

mole_timer > 0

Yes

mole_timer - -

mole_timer == 0

No

Yes

mole <= 1

hammer_timer > 0
&& mole == 1

No

Yes

hammer_timer - -

button == 1

No

Yes

score ++
reset timers & mole

hammer_timer == 0

No

Yes

lives - -
mole <= 0

lives == 0

Yes

No

state <= END_SCREEN

reset both timers

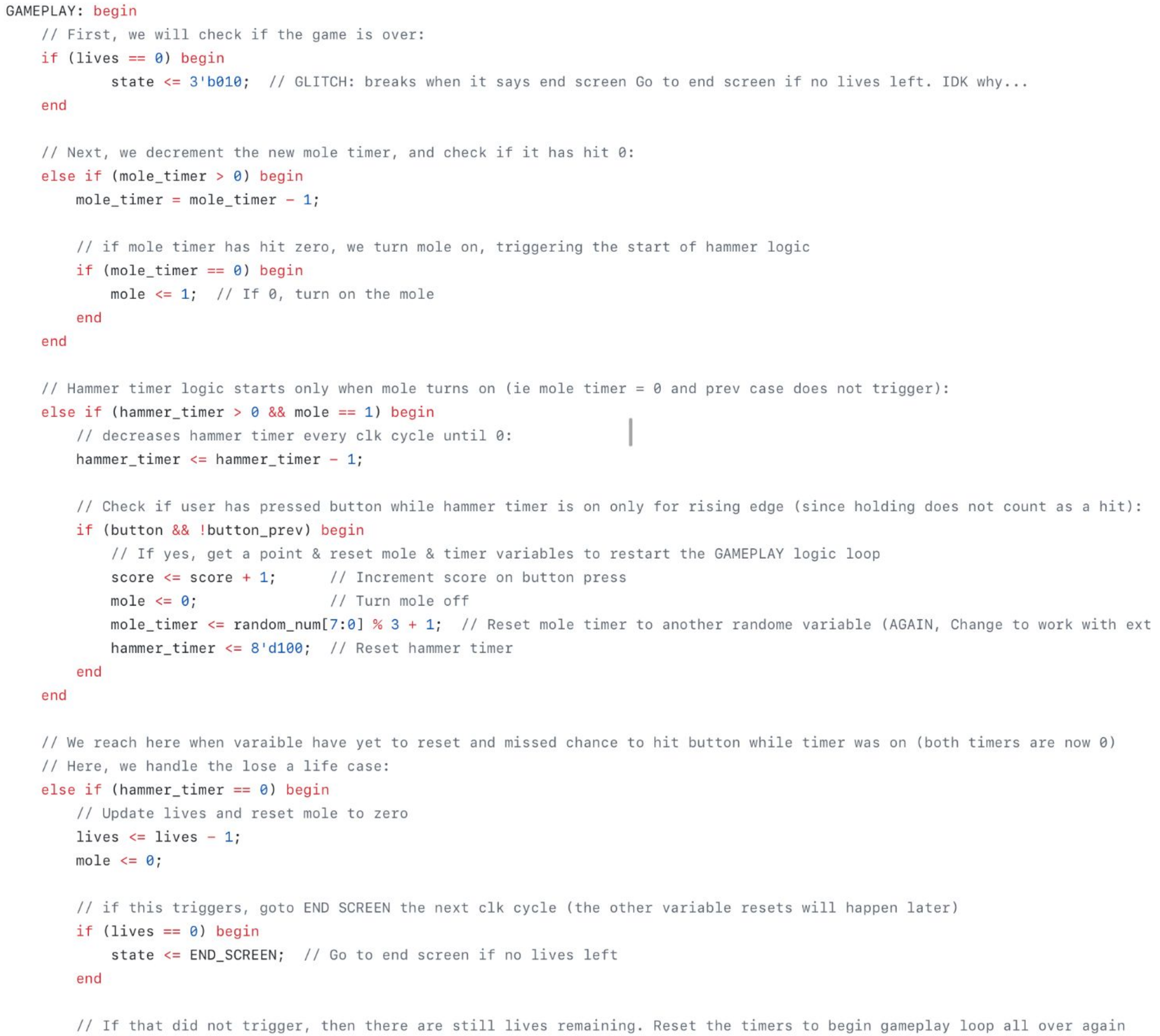
button == 1

No

Yes

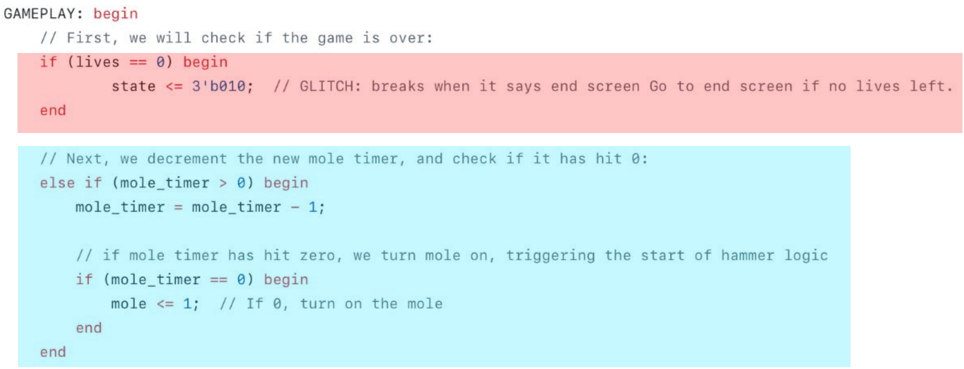
state <= 00
reset gameplay variables

Code Snippet

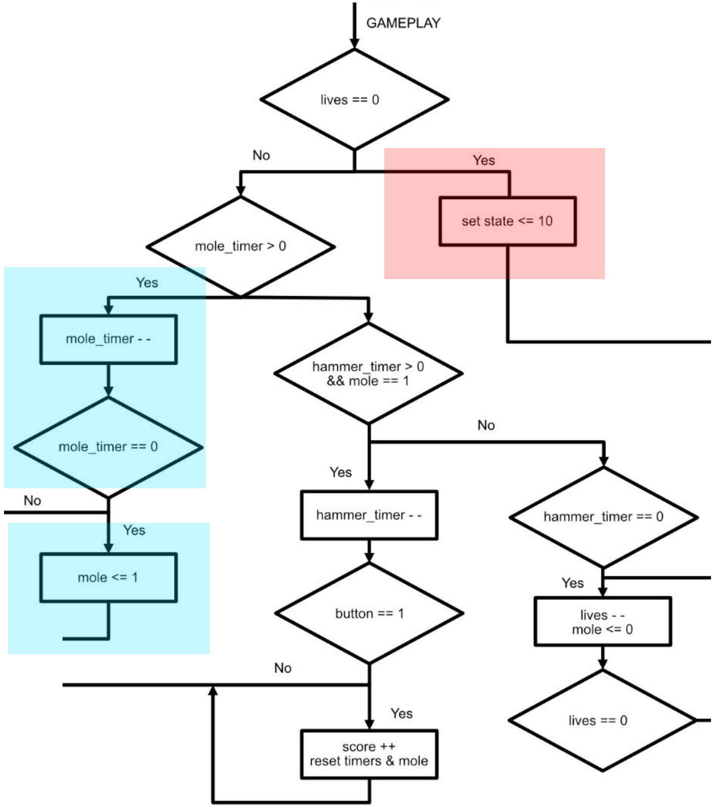


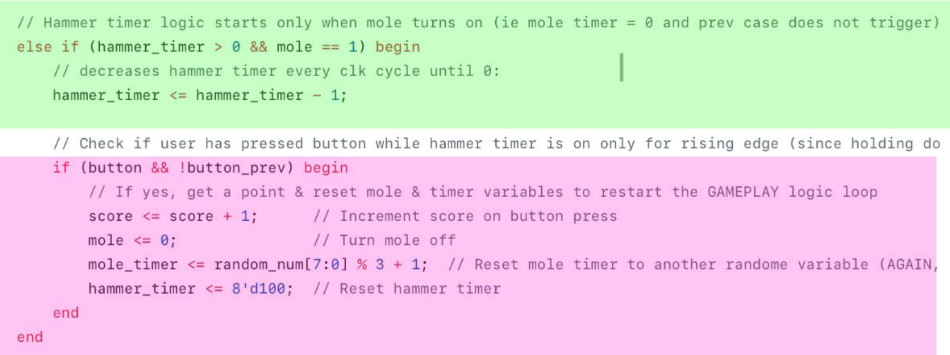
```
GAMEPLAY: begin
    // First, we will check if the game is over:
    if (lives == 0) begin
            state <= 3'b010;  // GLITCH: breaks when it says end screen Go to end screen if no lives left. IDK why...
    end

    // Next, we decrement the new mole timer, and check if it has hit 0:
    else if (mole_timer > 0) begin
        mole_timer = mole_timer - 1;

        // if mole timer has hit zero, we turn mole on, triggering the start of hammer logic
        if (mole_timer == 0) begin
            mole <= 1;  // If 0, turn on the mole
        end
    end

    // Hammer timer logic starts only when mole turns on (ie mole timer = 0 and prev case does not trigger):
    else if (hammer_timer > 0 && mole == 1) begin
        // decreases hammer timer every clk cycle until 0:
        hammer_timer <= hammer_timer - 1;

        // Check if user has pressed button while hammer timer is on only for rising edge (since holding does not count as a hit):
        if (button && !button_prev) begin
            // If yes, get a point & reset mole & timer variables to restart the GAMEPLAY logic loop
            score <= score + 1;      // Increment score on button press
            mole <= 0;               // Turn mole off
            mole_timer <= random_num[7:0] % 3 + 1;  // Reset mole timer to another randome variable (AGAIN, Change to work with ext
            hammer_timer <= 8'd100;  // Reset hammer timer
        end
    end

    // We reach here when varaible have yet to reset and missed chance to hit button while timer was on (both timers are now 0)
    // Here, we handle the lose a life case:
    else if (hammer_timer == 0) begin
        // Update lives and reset mole to zero
        lives <= lives - 1;
        mole <= 0;

        // if this triggers, goto END SCREEN the next clk cycle (the other variable resets will happen later)
        if (lives == 0) begin
            state <= END_SCREEN;  // Go to end screen if no lives left
        end

        // If that did not trigger, then there are still lives remaining. Reset the timers to begin gameplay loop all over again
```

*** Key Take-away: Timers as State Transition Conditions (Lab 2 + 3)

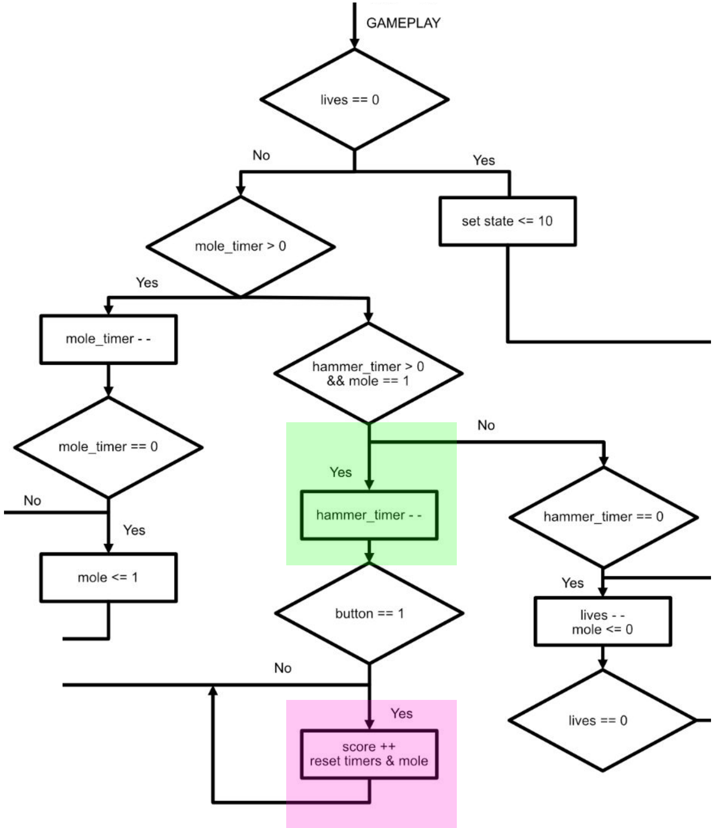


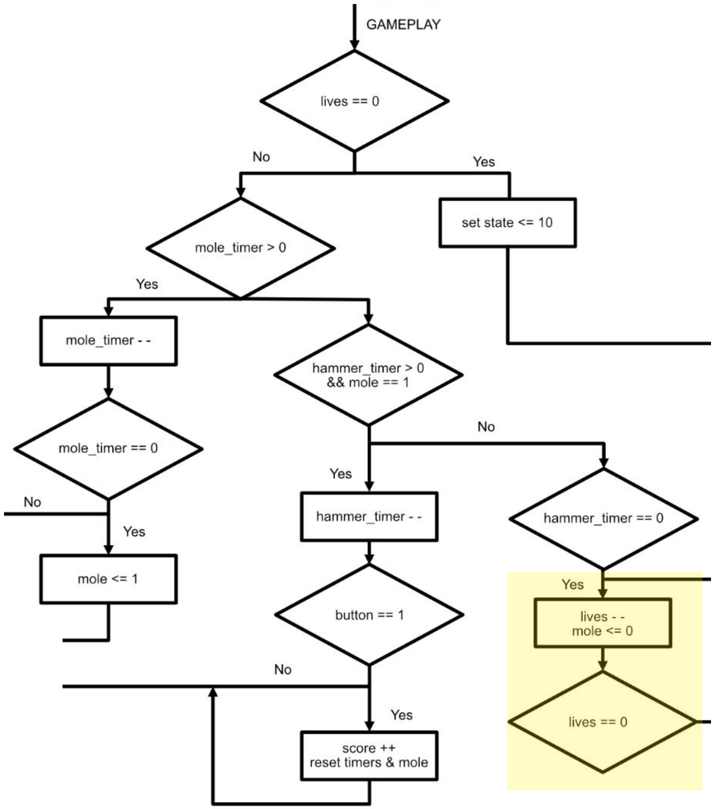
```
GAMEPLAY: begin
    // First, we will check if the game is over:
    if (lives == 0) begin
            state <= 3'b010;  // GLITCH: breaks when it says end screen Go to end screen if no lives left.
    end

    // Next, we decrement the new mole timer, and check if it has hit 0:
    else if (mole_timer > 0) begin
        mole_timer = mole_timer - 1;

        // if mole timer has hit zero, we turn mole on, triggering the start of hammer logic
        if (mole_timer == 0) begin
            mole <= 1;  // If 0, turn on the mole
        end
    end
```

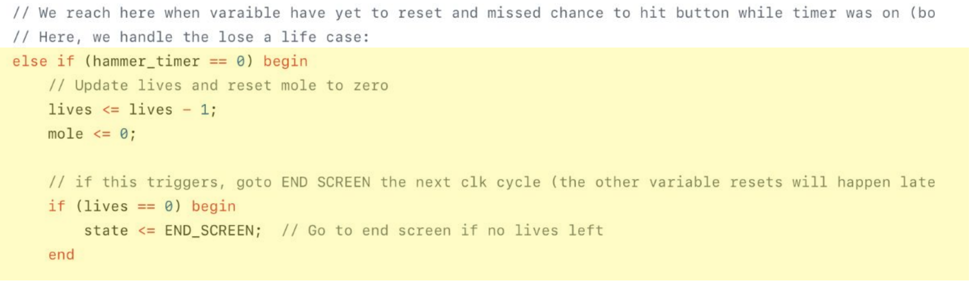




```
// Hammer timer logic starts only when mole turns on (ie mole timer = 0 and prev case does not trigger)
else if (hammer_timer > 0 && mole == 1) begin
    // decreases hammer timer every clk cycle until 0:
    hammer_timer <= hammer_timer - 1;

    // Check if user has pressed button while hammer timer is on only for rising edge (since holding do
    if (button && !button_prev) begin
        // If yes, get a point & reset mole & timer variables to restart the GAMEPLAY logic loop
        score <= score + 1;       // Increment score on button press
        mole <= 0;                // Turn mole off
        mole_timer <= random_num[7:0] % 3 + 1;  // Reset mole timer to another randome variable (AGAIN,
        hammer_timer <= 8'd100;  // Reset hammer timer
    end
end
```





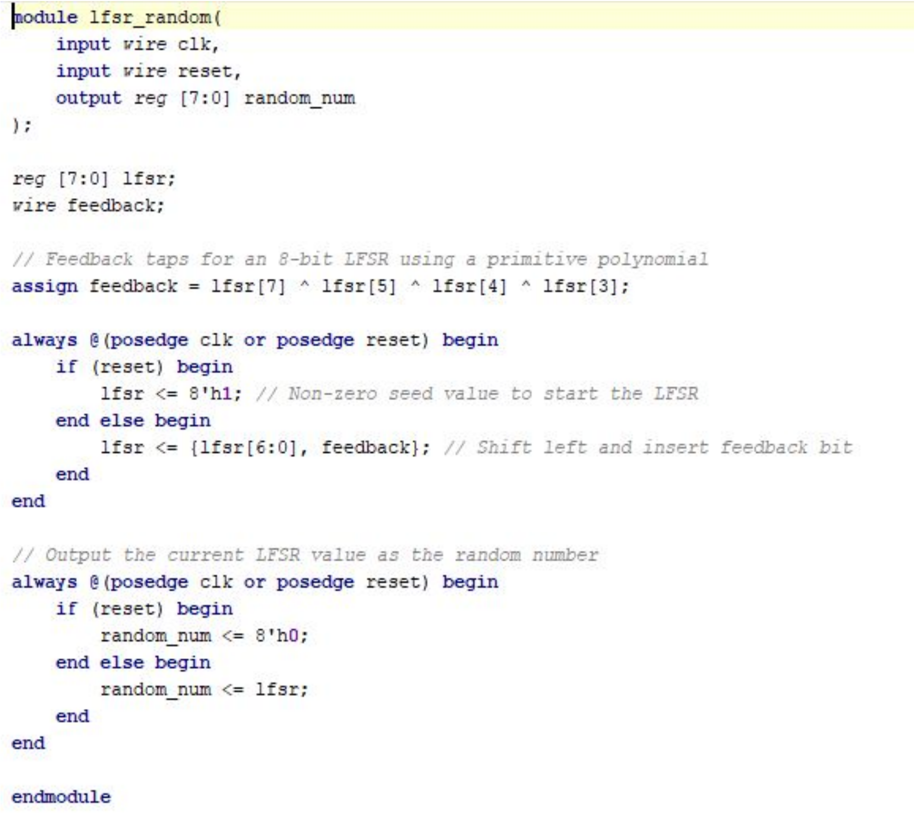


```
// We reach here when varaible have yet to reset and missed chance to hit button while timer was on (bo
// Here, we handle the lose a life case:
else if (hammer_timer == 0) begin
    // Update lives and reset mole to zero
    lives <= lives - 1;
    mole <= 0;

    // if this triggers, goto END SCREEN the next clk cycle (the other variable resets will happen late
    if (lives == 0) begin
        state <= END_SCREEN;  // Go to end screen if no lives left
    end
```

Two Notable Design Features

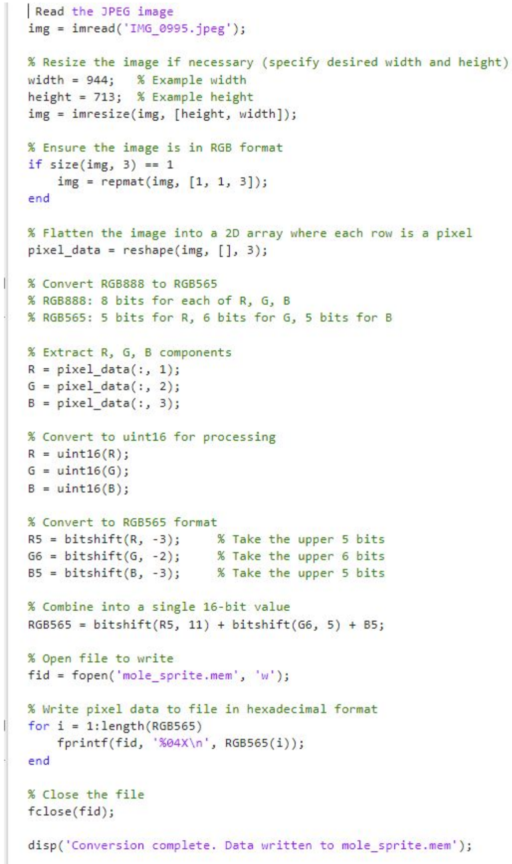
RNG Module

Responsible for random show up of mole

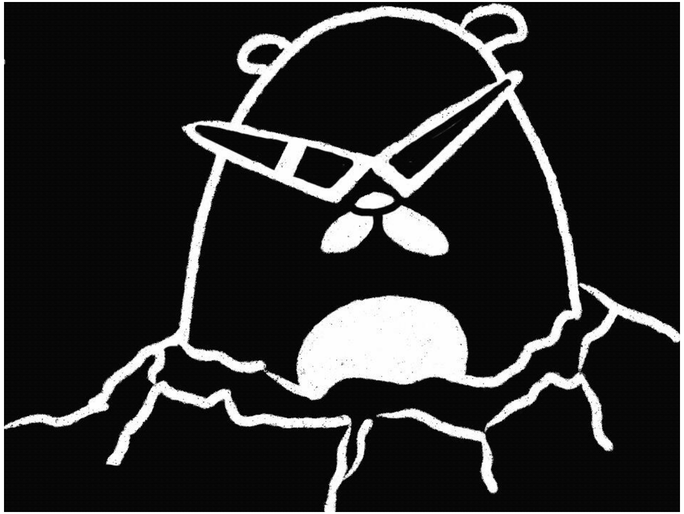


```verilog
module lfsr_random(
    input wire clk,
    input wire reset,
    output reg [7:0] random_num
);

reg [7:0] lfsr;
wire feedback;

// Feedback taps for an 8-bit LFSR using a primitive polynomial
assign feedback = lfsr[7] ^ lfsr[5] ^ lfsr[4] ^ lfsr[3];

always @(posedge clk or posedge reset) begin
    if (reset) begin
        lfsr <= 8'h1; // Non-zero seed value to start the LFSR
    end else begin
        lfsr <= {lfsr[6:0], feedback}; // Shift left and insert feedback bit
    end
end

// Output the current LFSR value as the random number
always @(posedge clk or posedge reset) begin
    if (reset) begin
        random_num <= 8'h0;
    end else begin
        random_num <= lfsr;
    end
end

endmodule
```

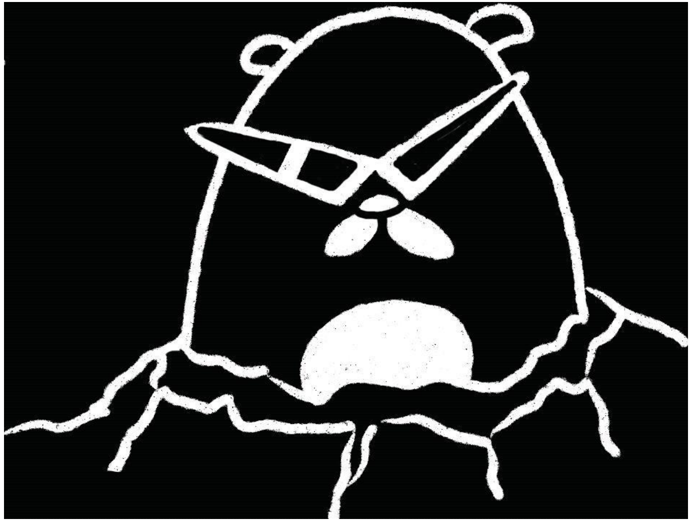
MATLAB Converter



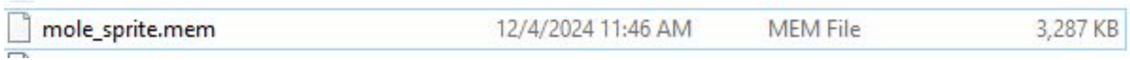
```
Read the JPEG image
img = imread('IMG_0995.jpeg');

% Resize the image if necessary (specify desired width and height)
width = 944;   % Example width
height = 713;  % Example height
img = imresize(img, [height, width]);

% Ensure the image is in RGB format
if size(img, 3) == 1
    img = repmat(img, [1, 1, 3]);
end

% Flatten the image into a 2D array where each row is a pixel
pixel_data = reshape(img, [], 3);

% Convert RGB888 to RGB565
% RGB888: 8 bits for each of R, G, B
% RGB565: 5 bits for R, 6 bits for G, 5 bits for B

% Extract R, G, B components
R = pixel_data(:, 1);
G = pixel_data(:, 2);
B = pixel_data(:, 3);

% Convert to uint16 for processing
R = uint16(R);
G = uint16(G);
B = uint16(B);

% Convert to RGB565 format
R5 = bitshift(R, -3);     % Take the upper 5 bits
G6 = bitshift(G, -2);     % Take the upper 6 bits
B5 = bitshift(B, -3);     % Take the upper 5 bits

% Combine into a single 16-bit value
RGB565 = bitshift(R5, 11) + bitshift(G6, 5) + B5;

% Open file to write
fid = fopen('mole_sprite.mem', 'w');

% Write pixel data to file in hexadecimal format
for i = 1:length(RGB565)
    fprintf(fid, '%04X\n', RGB565(i));
end

% Close the file
fclose(fid);

disp('Conversion complete. Data written to mole_sprite.mem');
```



Original



Bitmap Back to JPEG



mole_sprite.mem 12/4/2024 11:46 AM MEM File 3,287 KB

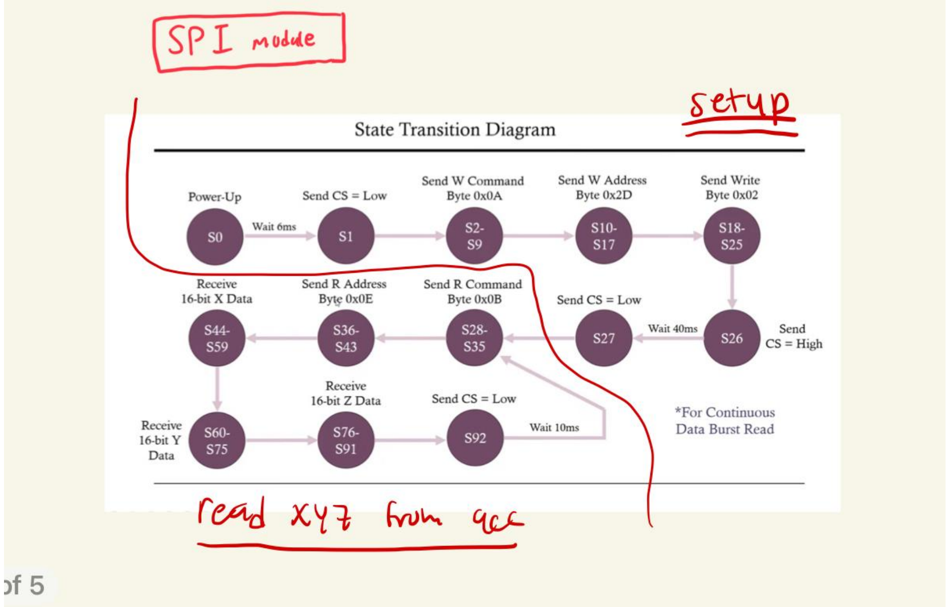
Bitmap MEM file

Preliminary 64x64 Sprites
(not yet written in Verilog)

Thus, planned 256x320
aspect ratio
(or something close)

Accelerometer

- FPGA —-> Accelerometer? SPI (serial peripheral interface)



Accelerometer going forward

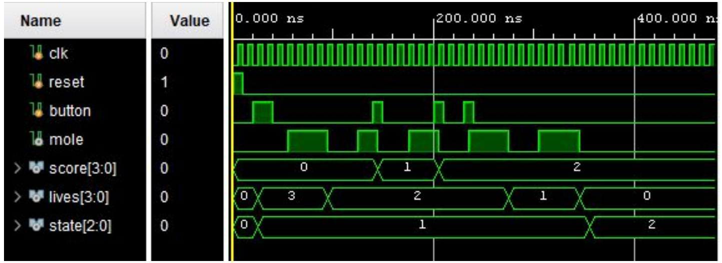
- get tilts to work (debouncer module for tilts)
- Connect tilt to button (forward tilt = front button)
- When all is done, Not show xyz values on 7segment display (used for testing)

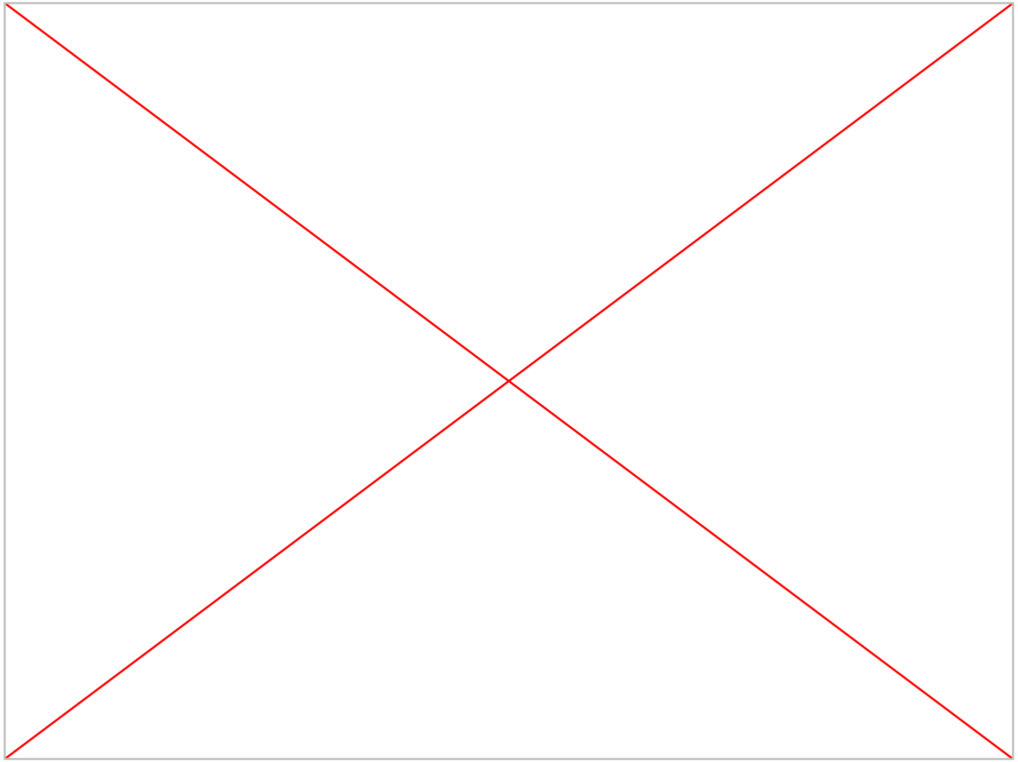
Successes?

We have a hardware only version of the game working on the FPGA by itself

✅ Playable on FPGA

✅ Working Testbench





Failures?

Next Steps…

- Expanding Core Game Logic → playing with more than one mole
- Tying the separate modules together:
  - Connecting the **Accelerometer** as an alternative input
  - Implementing **Random Number Generators**
  - Creating **VGA Modules** to load and display bitmap data

Thank you for Listening.

Questions?