Target Game Project

FPGA + Raspberry Pi target shooting game with CV detection

Table of Contents

FPGA Modules (Verilog)

Raspberry Pi Programs (C++)

Shell Script

FPGA Modules (Verilog)

1. rgb_matrix_top.v

```
// RGB Matrix Controller Top Module
// Q[pS■F27p^[

module rgb_matrix_top (
input wire clk, // 12MHz system clock
input wire btn, // {^■iZbgpj
input wire [4:0] pattern_sel, // p^[Ii5bit = 32p^[j

// RGB Matrix Interface
output wire r1, g1, b1, // Upper half RGB
output wire r2, g2, b2, // Lower half RGB
output wire [3:0] addr, // Row address A,B,C,D
output wire lat, // Latch
output wire oe_n, // Output Enable (active low)
output wire clk_out // Matrix clock

// Debug LED
// output wire led0 // I{[hLEDimFpj
);

// Parameters
parameter MATRIX_WIDTH = 32*9;

// Reset logic
wire rst_n = ~btn;

// Internal signals
wire clk_enable;
wire [2:0] state;
wire [8:0] col_count;
wire [3:0] row_count;
wire upper_r_wire, upper_g_wire, upper_b_wire;
wire lower_r_wire, lower_g_wire, lower_b_wire;

// Clock divider module
clock_divider clk_div_inst (
.clk(clk),

.rst_n(rst_n),

.clk_enable(clk_enable)

);


// State machine module

state_controller state_ctrl_inst (

.clk(clk),

.rst_n(rst_n),

.clk_enable(clk_enable),

.matrix_width(MATRIX_WIDTH),

.state(state),

.col_count(col_count),

.row_count(row_count),

.r1(r1), .g1(g1), .b1(b1),

.r2(r2), .g2(g2), .b2(b2),

.addr(addr),

.lat(lat),

.oe_n(oe_n),

.clk_out(clk_out),

// .led0(led0),

.upper_r_in(upper_r_wire),

.upper_g_in(upper_g_wire),

.upper_b_in(upper_b_wire),

.lower_r_in(lower_r_wire),

.lower_g_in(lower_g_wire),

.lower_b_in(lower_b_wire)

);


// Pattern generator module

pattern_generator pattern_gen_inst (

.pattern_sel(pattern_sel),

.col_count(col_count),

.row_count(row_count),

.upper_r(upper_r_wire),

.upper_g(upper_g_wire),

.upper_b(upper_b_wire),

.lower_r(lower_r_wire),

.lower_g(lower_g_wire),
.lower_b(lower_b_wire)

);


endmodule
```

2. state_controller.v

```
//===========================================
// State Controller Module
//===========================================
module state_controller (
input wire clk,
input wire rst_n,
input wire clk_enable,
input wire [8:0] matrix_width,

output reg [2:0] state,
output reg [8:0] col_count,
output reg [3:0] row_count,

output reg r1, g1, b1,
output reg r2, g2, b2,
output reg [3:0] addr,
output reg lat,
output reg oe_n,
output reg clk_out,
// output reg led0,

input wire upper_r_in, upper_g_in, upper_b_in,
input wire lower_r_in, lower_g_in, lower_b_in
);

// State definitions
localparam [2:0]
IDLE = 3'b000,
SETUP_DATA = 3'b001,
CLOCK_PULSE = 3'b010,
LATCH_DATA = 3'b011,
DISPLAY = 3'b100,
NEXT_ROW = 3'b101;

reg [7:0] display_timer;

always @(posedge clk or negedge rst_n) begin
if (!rst_n) begin
state <= IDLE;
```

```
col_count <= 9'd0;

row_count <= 4'd0;

display_timer <= 8'd0;


r1 <= 1'b0; g1 <= 1'b0; b1 <= 1'b0;

r2 <= 1'b0; g2 <= 1'b0; b2 <= 1'b0;

addr <= 4'b0000;

lat <= 1'b0;

oe_n <= 1'b1;

clk_out <= 1'b0;

// led0 <= 1'b0;


end else if (clk_enable) begin


// led0 <= 1'b1;


case (state)

IDLE: begin

oe_n <= 1'b1;

lat <= 1'b0;

clk_out <= 1'b0;

addr <= row_count;

col_count <= 9'd0;

display_timer <= 8'd0;

state <= SETUP_DATA;

end


SETUP_DATA: begin

oe_n <= 1'b1;

clk_out <= 1'b0;


r1 <= upper_r_in;

g1 <= upper_g_in;

b1 <= upper_b_in;

r2 <= lower_r_in;

g2 <= lower_g_in;

b2 <= lower_b_in;


state <= CLOCK_PULSE;

end


CLOCK_PULSE: begin
```

```
clk_out <= 1'b1;

if (col_count < (matrix_width - 1)) begin

col_count <= col_count + 1;

state <= SETUP_DATA;

end else begin

state <= LATCH_DATA;

end

end

LATCH_DATA: begin

clk_out <= 1'b0;

lat <= 1'b1;

state <= DISPLAY;

end

DISPLAY: begin

lat <= 1'b0;

oe_n <= 1'b0;

if (display_timer < 8'd63) begin

display_timer <= display_timer + 1;

end else begin

state <= NEXT_ROW;

end

end

NEXT_ROW: begin

oe_n <= 1'b1;

if (row_count < 15) begin

row_count <= row_count + 1;

end else begin

row_count <= 4'd0;

end

state <= IDLE;

end

default: begin

state <= IDLE;

end
```

```
endcase

end

end


endmodule
```

3. pattern_generator.v

```
//==========================================
// Pattern Generator Module
//==========================================
module pattern_generator (
input wire [4:0] pattern_sel,
input wire [8:0] col_count,
input wire [3:0] row_count,

output reg upper_r, upper_g, upper_b,
output reg lower_r, lower_g, lower_b
);

// pl■■■u■■i`F[■j
// panel_num: 0, 1, 2, 3, 4, 5, 6, 7, 8
// ■u: , ^■, , ^■, E, ^, ^■E, ^■, E
//
// 3~3zu:
// 2() 3(^■) 4(E)
// 1(^■) 5(^) 6(^■E)
// 0() 7(^■) 8(E)
//
// z_}bsO:
// 10_: 2(), 4(E), 0(), 8(E) = 4
// 3_: 3(), 1(), 6(E), 7() = 4
// 5_: 5(^)

// z_p■■
function [3:0] get_panel_score;
input [3:0] panel;
begin
case (panel)
4'd0, 4'd2, 4'd4, 4'd8: get_panel_score = 4'd10; // 4F10_
4'd1, 4'd3, 4'd6, 4'd7: get_panel_score = 4'd3; // 4■F3_
4'd5: get_panel_score = 4'd5; // F5_
default: get_panel_score = 4'd0;
endcase
end
endfunction
```

```
// pl■vZi0-8j
wire [3:0] panel_num = col_count[8:5];

// epl■■■ui0-31j
wire [4:0] local_col = col_count[4:0];

// plEo
wire is_left_edge = (local_col == 5'd0);
wire is_right_edge = (local_col == 5'd31);
wire is_panel_border = is_left_edge || is_right_edge;

// ■Eo
wire is_top_edge = (row_count == 4'd0);
wire is_bottom_edge = (row_count == 4'd15);

// \pF16~16sNZ■i■j■Ei■j
wire is_left_half = (local_col < 5'd16);

// always■O■oreg■
reg pixel_show;
reg [3:0] digit;
reg show_text;
reg [7:0] current_char;
reg [2:0] char_x, char_y;
reg [3:0] score;
reg [3:0] tens_digit;
reg [3:0] ones_digit;
reg [3:0] score_lower;
reg [3:0] tens_digit_lower;
reg [3:0] ones_digit_lower;

reg [4:0] text_start;
reg [4:0] char_pos;
reg [3:0] text_row_start;

// 5~7tHg`iAt@xbgpj
function get_char_pixel;
input [7:0] char_code; // R[hi'S', 'T', 'A', 'R', 'T'j
input [2:0] x; // 0-4XW
input [2:0] y; // 0-6YW
reg [34:0] bitmap; // 5~7 = 35bit
begin
case (char_code)
"S": bitmap = 35'b01110_10001_10000_01110_00001_10001_01110; // S
"T": bitmap = 35'b11111_00100_00100_00100_00100_00100_00100; // T
"A": bitmap = 35'b01110_10001_10001_11111_10001_10001_10001; // A
"R": bitmap = 35'b11110_10001_10001_11110_10100_10010_10001; // R
default: bitmap = 35'b00000_00000_00000_00000_00000_00000_00000;
endcase

// rbg}bvYsNZ■i■■AEj
get_char_pixel = bitmap[34 - (y * 5 + x)];
end
endfunction

// 7ZOgtHg■i16x16sNZj
function get_7seg_pixel;
input [3:0] digit;
input [4:0] x;
input [3:0] y;
reg seg_a, seg_b, seg_c, seg_d, seg_e, seg_f, seg_g;
begin
case (digit)
4'd0: {seg_a, seg_b, seg_c, seg_d, seg_e, seg_f, seg_g} = 7'b1111110;
4'd1: {seg_a, seg_b, seg_c, seg_d, seg_e, seg_f, seg_g} = 7'b0110000;
4'd2: {seg_a, seg_b, seg_c, seg_d, seg_e, seg_f, seg_g} = 7'b1101101;
4'd3: {seg_a, seg_b, seg_c, seg_d, seg_e, seg_f, seg_g} = 7'b1111001;
4'd4: {seg_a, seg_b, seg_c, seg_d, seg_e, seg_f, seg_g} = 7'b0110011;
4'd5: {seg_a, seg_b, seg_c, seg_d, seg_e, seg_f, seg_g} = 7'b1011011;
4'd6: {seg_a, seg_b, seg_c, seg_d, seg_e, seg_f, seg_g} = 7'b1011111;
4'd7: {seg_a, seg_b, seg_c, seg_d, seg_e, seg_f, seg_g} = 7'b1110000;
4'd8: {seg_a, seg_b, seg_c, seg_d, seg_e, seg_f, seg_g} = 7'b1111111;
4'd9: {seg_a, seg_b, seg_c, seg_d, seg_e, seg_f, seg_g} = 7'b1111011;
default: {seg_a, seg_b, seg_c, seg_d, seg_e, seg_f, seg_g} = 7'b0000000;
endcase

if (seg_a && y >= 4'd1 && y <= 4'd2 && x >= 5'd3 && x <= 5'd12)
get_7seg_pixel = 1'b1;
else if (seg_b && y >= 4'd3 && y <= 4'd6 && x >= 5'd12 && x <= 5'd13)
get_7seg_pixel = 1'b1;
```

```
else if (seg_c && y >= 4'd9 && y <= 4'd12 && x >= 5'd12 && x <= 5'd13)
get_7seg_pixel = 1'b1;
else if (seg_d && y >= 4'd13 && y <= 4'd14 && x >= 5'd3 && x <= 5'd12)
get_7seg_pixel = 1'b1;
else if (seg_e && y >= 4'd9 && y <= 4'd12 && x >= 5'd2 && x <= 5'd3)
get_7seg_pixel = 1'b1;
else if (seg_f && y >= 4'd3 && y <= 4'd6 && x >= 5'd2 && x <= 5'd3)
get_7seg_pixel = 1'b1;
else if (seg_g && y >= 4'd7 && y <= 4'd8 && x >= 5'd3 && x <= 5'd12)
get_7seg_pixel = 1'b1;
else
get_7seg_pixel = 1'b0;
end
endfunction

// ■\p■■
function [3:0] get_score_digit;
input [4:0] pattern;
input is_upper; // 1=\■, 0=■
reg [5:0] score;
begin
case (pattern)
5'd13: score = 6'd0;
5'd14: score = 6'd3;
5'd15: score = 6'd5;
5'd16: score = 6'd6;
5'd17: score = 6'd8;
5'd18: score = 6'd9;
5'd19: score = 6'd10;
5'd20: score = 6'd11;
5'd21: score = 6'd13;
5'd22: score = 6'd15;
5'd23: score = 6'd16;
5'd24: score = 6'd18;
5'd25: score = 6'd20;
5'd26: score = 6'd23;
5'd27: score = 6'd25;
5'd28: score = 6'd30;
```

```
default: score = 6'd0;
endcase

if (is_upper)
get_score_digit = score / 10;
else
get_score_digit = score % 10;
end
endfunction

// p^[■
always @(*) begin
case (pattern_sel)
// ============ { ============
5'd0: begin // STANDBY: "START"\iplj
if (panel_num == 4'd5) begin // ^■plicol 160-191j
// START\i5~6sNZ = 30sNZAzuj
// ■Jn■ui: (32-30)/2 = 1j
text_start = 5'd1;
char_pos = local_col - text_start;

// c■Jn■ui: (16-7)/2 = 4.5 4j
text_row_start = 4'd4;

show_text = 1'b0;

// \■■`FbNirow 4-10, col 1-30j
if (row_count >= text_row_start && row_count < (text_row_start + 4'd7) &&
local_col >= text_start && local_col < (text_start + 5'd30)) begin

char_y = row_count - text_row_start;

// e■■u
if (char_pos < 5'd5) begin
current_char = "S";
char_x = char_pos[2:0];
show_text = get_char_pixel(current_char, char_x, char_y);
end else if (char_pos >= 5'd6 && char_pos < 5'd11) begin
current_char = "T";
char_x = (char_pos - 5'd6);
show_text = get_char_pixel(current_char, char_x, char_y);
```

```
end else if (char_pos >= 5'd12 && char_pos < 5'd17) begin
current_char = "A";
char_x = (char_pos - 5'd12);
show_text = get_char_pixel(current_char, char_x, char_y);
end else if (char_pos >= 5'd18 && char_pos < 5'd23) begin
current_char = "R";
char_x = (char_pos - 5'd18);
show_text = get_char_pixel(current_char, char_x, char_y);
end else if (char_pos >= 5'd24 && char_pos < 5'd29) begin
current_char = "T";
char_x = (char_pos - 5'd24);
show_text = get_char_pixel(current_char, char_x, char_y);
end
end

if (show_text) begin
upper_r = 1'b0; upper_g = 1'b1; upper_b = 1'b0; //
lower_r = 1'b0; lower_g = 1'b1; lower_b = 1'b0;
end else begin
upper_r = 1'b0; upper_g = 1'b0; upper_b = 1'b0; //
lower_r = 1'b0; lower_g = 1'b0; lower_b = 1'b0;
end
end else begin
upper_r = 1'b0; upper_g = 1'b0; upper_b = 1'b0;
lower_r = 1'b0; lower_g = 1'b0; lower_b = 1'b0;
end
end

5'd1: begin // EXPLANATION: ■g+epl■z_\
// ■g■
if (is_panel_border || is_top_edge) begin
upper_r = 1'b0; upper_g = 1'b0; upper_b = 1'b1;
end else begin
// gFepl■\
score = get_panel_score(panel_num);
tens_digit = score / 10;
ones_digit = score % 10;

if (is_left_half) begin
```

```
pixel_show = get_7seg_pixel(tens_digit, local_col, row_count);
end else begin
pixel_show = get_7seg_pixel(ones_digit, local_col - 5'd16, row_count);
end

// z_■F
case (score)
4'd10: begin // 4FF
if (pixel_show) begin
upper_r = 1'b1; upper_g = 1'b1; upper_b = 1'b0;
end else begin
upper_r = 1'b0; upper_g = 1'b0; upper_b = 1'b0;
end
end
4'd3: begin // 4■FVA
if (pixel_show) begin
upper_r = 1'b0; upper_g = 1'b1; upper_b = 1'b1;
end else begin
upper_r = 1'b0; upper_g = 1'b0; upper_b = 1'b0;
end
end
4'd5: begin // F}[^
if (pixel_show) begin
upper_r = 1'b1; upper_g = 1'b0; upper_b = 1'b1;
end else begin
upper_r = 1'b0; upper_g = 1'b0; upper_b = 1'b0;
end
end
default: begin
upper_r = 1'b0; upper_g = 1'b0; upper_b = 1'b0;
end
endcase
end

// l
if (is_panel_border || is_bottom_edge) begin
lower_r = 1'b0; lower_g = 1'b0; lower_b = 1'b1;
end else begin
score_lower = get_panel_score(panel_num);
```

```
tens_digit_lower = score_lower / 10;

ones_digit_lower = score_lower % 10;


if (is_left_half) begin

pixel_show = get_7seg_pixel(tens_digit_lower, local_col, row_count);

end else begin

pixel_show = get_7seg_pixel(ones_digit_lower, local_col - 5'd16, row_count);

end


case (score_lower)

4'd10: begin

if (pixel_show) begin

lower_r = 1'b1; lower_g = 1'b1; lower_b = 1'b0;

end else begin

lower_r = 1'b0; lower_g = 1'b0; lower_b = 1'b0;

end

end

4'd3: begin

if (pixel_show) begin

lower_r = 1'b0; lower_g = 1'b1; lower_b = 1'b1;

end else begin

lower_r = 1'b0; lower_g = 1'b0; lower_b = 1'b0;

end

end

4'd5: begin

if (pixel_show) begin

lower_r = 1'b1; lower_g = 1'b0; lower_b = 1'b1;

end else begin

lower_r = 1'b0; lower_g = 1'b0; lower_b = 1'b0;

end

end

default: begin

lower_r = 1'b0; lower_g = 1'b0; lower_b = 1'b0;

end

endcase

end

end


5'd2: begin // GAME_PLAYING
```

```
if (is_panel_border || is_top_edge) begin

upper_r = 1'b0; upper_g = 1'b0; upper_b = 1'b1;

end else begin

upper_r = 1'b0; upper_g = 1'b0; upper_b = 1'b0;

end


if (is_panel_border || is_bottom_edge) begin

lower_r = 1'b0; lower_g = 1'b0; lower_b = 1'b1;

end else begin

lower_r = 1'b0; lower_g = 1'b0; lower_b = 1'b0;

end

end


// ============ qbgoi3-11j ============

5'd3, 5'd4, 5'd5, 5'd6, 5'd7, 5'd8, 5'd9, 5'd10, 5'd11: begin

if (panel_num == (pattern_sel - 5'd3)) begin

upper_r = 1'b1; upper_g = 1'b0; upper_b = 1'b0;

lower_r = 1'b1; lower_g = 1'b0; lower_b = 1'b0;

end else begin

if (is_panel_border || is_top_edge) begin

upper_r = 1'b0; upper_g = 1'b0; upper_b = 1'b1;

end else begin

upper_r = 1'b0; upper_g = 1'b0; upper_b = 1'b0;

end


if (is_panel_border || is_bottom_edge) begin

lower_r = 1'b0; lower_g = 1'b0; lower_b = 1'b1;

end else begin

lower_r = 1'b0; lower_g = 1'b0; lower_b = 1'b0;

end

end

end


5'd12: begin // MISS

upper_r = 1'b0; upper_g = 1'b0; upper_b = 1'b0;

lower_r = 1'b1; lower_g = 1'b0; lower_b = 1'b0;

end


// ============ ■\i13-28j ============

5'd13, 5'd14, 5'd15, 5'd16, 5'd17, 5'd18, 5'd19, 5'd20,
5'd21, 5'd22, 5'd23, 5'd24, 5'd25, 5'd26, 5'd27, 5'd28: begin

if (panel_num == 4'd5) begin

if (is_left_half) begin

digit = get_score_digit(pattern_sel, 1'b1);

pixel_show = get_7seg_pixel(digit, local_col, row_count);

end else begin

digit = get_score_digit(pattern_sel, 1'b0);

pixel_show = get_7seg_pixel(digit, local_col - 5'd16, row_count);

end


if (pixel_show) begin

upper_r = 1'b1; upper_g = 1'b1; upper_b = 1'b0;

lower_r = 1'b1; lower_g = 1'b1; lower_b = 1'b0;

end else begin

upper_r = 1'b0; upper_g = 1'b0; upper_b = 1'b0;

lower_r = 1'b0; lower_g = 1'b0; lower_b = 1'b0;

end

end else begin

if (is_panel_border || is_top_edge) begin

upper_r = 1'b0; upper_g = 1'b0; upper_b = 1'b1;

end else begin

upper_r = 1'b0; upper_g = 1'b0; upper_b = 1'b0;

end


if (is_panel_border || is_bottom_edge) begin

lower_r = 1'b0; lower_g = 1'b0; lower_b = 1'b1;

end else begin

lower_r = 1'b0; lower_g = 1'b0; lower_b = 1'b0;

end

end

end


default: begin

upper_r = 1'b0; upper_g = 1'b0; upper_b = 1'b0;

lower_r = 1'b0; lower_g = 1'b0; lower_b = 1'b0;

end

endcase

end


endmodule
```

4. clock_divider.v

```
//===========================================
// Clock Divider Module
//===========================================
module clock_divider (
input wire clk,
input wire rst_n,
output wire clk_enable
);

reg [1:0] clk_div;

assign clk_enable = (clk_div == 2'b11); // 12MHz/4 = 3MHz

always @(posedge clk or negedge rst_n) begin
if (!rst_n) begin
clk_div <= 2'b00;
end else begin
clk_div <= clk_div + 1;
end
end

endmodule
```

5. LED_3232.xdc

```
## 12 MHz Clock Signal
set_property -dict { PACKAGE_PIN L17 IOSTANDARD LVCMOS33 } [get_ports clk]; #IO_L12P_T1_MRCC_14 Sch=gclk
create_clock -add -name sys_clk_pin -period 83.33 -waveform {0 41.66} [get_ports clk];
## GPIO Pins
## Pins 15 and 16 should remain commented if using them as analog inputs

## Buttons
set_property -dict { PACKAGE_PIN A18 IOSTANDARD LVCMOS33 } [get_ports { btn }]; #IO_L19N_T3_VREF_16 Sch=btn[0]
#set_property -dict { PACKAGE_PIN B18 IOSTANDARD LVCMOS33 } [get_ports { btn[1] }]; #IO_L19P_T3_16 Sch=btn[1]

set_property -dict { PACKAGE_PIN M3 IOSTANDARD LVCMOS33 } [get_ports { pattern_sel[0] }]; #IO_L8N_T1_AD14N_35 Sch=pio[01]
set_property -dict { PACKAGE_PIN L3 IOSTANDARD LVCMOS33 } [get_ports { pattern_sel[1] }]; #IO_L8P_T1_AD14P_35 Sch=pio[02]
set_property -dict { PACKAGE_PIN A16 IOSTANDARD LVCMOS33 } [get_ports { pattern_sel[2] }]; #IO_L12P_T1_MRCC_16 Sch=pio[03]
set_property -dict { PACKAGE_PIN K3 IOSTANDARD LVCMOS33 } [get_ports { pattern_sel[3] }]; #IO_L7N_T1_AD6N_35 Sch=pio[04]
set_property -dict { PACKAGE_PIN C15 IOSTANDARD LVCMOS33 } [get_ports { pattern_sel[4] }]; #IO_L11P_T1_SRCC_16 Sch=pio[05]
#set_property -dict { PACKAGE_PIN H1 IOSTANDARD LVCMOS33 } [get_ports { pio6 }]; #IO_L3P_T0_DQS_AD5P_35 Sch=pio[06]
#set_property -dict { PACKAGE_PIN A15 IOSTANDARD LVCMOS33 } [get_ports { pio7 }]; #IO_L6N_T0_VREF_16 Sch=pio[07]
#set_property -dict { PACKAGE_PIN B15 IOSTANDARD LVCMOS33 } [get_ports { pio8 }]; #IO_L11N_T1_SRCC_16 Sch=pio[08]
#set_property -dict { PACKAGE_PIN A14 IOSTANDARD LVCMOS33 } [get_ports { pio9 }]; #IO_L6P_T0_16 Sch=pio[09]
#set_property -dict { PACKAGE_PIN J3 IOSTANDARD LVCMOS33 } [get_ports { pio10 }]; #IO_L7P_T1_AD6P_35 Sch=pio[10]
#set_property -dict { PACKAGE_PIN J1 IOSTANDARD LVCMOS33 } [get_ports { pio11 }]; #IO_L3N_T0_DQS_AD5N_35 Sch=pio[11]
#set_property -dict { PACKAGE_PIN K2 IOSTANDARD LVCMOS33 } [get_ports { pio12 }]; #IO_L5P_T0_AD13P_35 Sch=pio[12]
#set_property -dict { PACKAGE_PIN L1 IOSTANDARD LVCMOS33 } [get_ports { pio13 }]; #IO_L6N_T0_VREF_35 Sch=pio[13]
#set_property -dict { PACKAGE_PIN L2 IOSTANDARD LVCMOS33 } [get_ports { pio14 }]; #IO_L5N_T0_AD13N_35 Sch=pio[14]
#set_property -dict { PACKAGE_PIN M1 IOSTANDARD LVCMOS33 } [get_ports { pio17 }]; #IO_L9N_T1_DQS_AD7N_35 Sch=pio[17]
#set_property -dict { PACKAGE_PIN N3 IOSTANDARD LVCMOS33 } [get_ports { pio18 }]; #IO_L12P_T1_MRCC_35 Sch=pio[18]
#set_property -dict { PACKAGE_PIN P3 IOSTANDARD LVCMOS33 } [get_ports { pio19 }]; #IO_L12N_T1_MRCC_35 Sch=pio[19]
#set_property -dict { PACKAGE_PIN M2 IOSTANDARD LVCMOS33 } [get_ports { pio20 }]; #IO_L9P_T1_DQS_AD7P_35 Sch=pio[20]
#set_property -dict { PACKAGE_PIN N1 IOSTANDARD LVCMOS33 } [get_ports { pio21 }]; #IO_L10N_T1_AD15N_35 Sch=pio[21]
#set_property -dict { PACKAGE_PIN N2 IOSTANDARD LVCMOS33 } [get_ports { pio22 }]; #IO_L10P_T1_AD15P_35 Sch=pio[22]
#set_property -dict { PACKAGE_PIN P1 IOSTANDARD LVCMOS33 } [get_ports { led0 }]; #IO_L19N_T3_VREF_35 Sch=pio[23]
set_property -dict { PACKAGE_PIN R3 IOSTANDARD LVCMOS33 } [get_ports { r1 }]; #IO_L2P_T0_34 Sch=pio[26]
set_property -dict { PACKAGE_PIN T3 IOSTANDARD LVCMOS33 } [get_ports { g1 }]; #IO_L2N_T0_34 Sch=pio[27]
set_property -dict { PACKAGE_PIN R2 IOSTANDARD LVCMOS33 } [get_ports { b1 }]; #IO_L1P_T0_34 Sch=pio[28]
#set_property -dict { PACKAGE_PIN T1 IOSTANDARD LVCMOS33 } [get_ports { pio29 }]; #IO_L3P_T0_DQS_34 Sch=pio[29]
```

```
set_property -dict { PACKAGE_PIN T2 IOSTANDARD LVCMOS33 } [get_ports { r2 }]; #IO_L1N_T0_34 Sch=pio[30]

set_property -dict { PACKAGE_PIN U1 IOSTANDARD LVCMOS33 } [get_ports { g2 }]; #IO_L3N_T0_DQS_34 Sch=pio[31]

set_property -dict { PACKAGE_PIN W2 IOSTANDARD LVCMOS33 } [get_ports { b2 }]; #IO_L5N_T0_34 Sch=pio[32]

#set_property -dict { PACKAGE_PIN V2 IOSTANDARD LVCMOS33 } [get_ports { pio33 }]; #IO_L5P_T0_34 Sch=pio[33]

set_property -dict { PACKAGE_PIN W3 IOSTANDARD LVCMOS33 } [get_ports { addr[0] }]; #IO_L6N_T0_VREF_34 Sch=pio[34]

set_property -dict { PACKAGE_PIN V3 IOSTANDARD LVCMOS33 } [get_ports { addr[1] }]; #IO_L6P_T0_34 Sch=pio[35]

set_property -dict { PACKAGE_PIN W5 IOSTANDARD LVCMOS33 } [get_ports { addr[2] }]; #IO_L12P_T1_MRCC_34 Sch=pio[36]

set_property -dict { PACKAGE_PIN V4 IOSTANDARD LVCMOS33 } [get_ports { addr[3] }]; #IO_L11N_T1_SRCC_34 Sch=pio[37]

#set_property -dict { PACKAGE_PIN U4 IOSTANDARD LVCMOS33 } [get_ports { pio38 }]; #IO_L11P_T1_SRCC_34 Sch=pio[38]

set_property -dict { PACKAGE_PIN V5 IOSTANDARD LVCMOS33 } [get_ports clk_out]; #IO_L16N_T2_34 Sch=pio[39]

set_property -dict { PACKAGE_PIN W4 IOSTANDARD LVCMOS33 } [get_ports lat]; #IO_L12N_T1_MRCC_34 Sch=pio[40]

set_property -dict { PACKAGE_PIN U5 IOSTANDARD LVCMOS33 } [get_ports oe_n]; #IO_L16P_T2_34 Sch=pio[41]

#set_property -dict { PACKAGE_PIN U2 IOSTANDARD LVCMOS33 } [get_ports { pio42 }]; #IO_L9N_T1_DQS_34 Sch=pio[42]

#set_property -dict { PACKAGE_PIN W6 IOSTANDARD LVCMOS33 } [get_ports { pio43 }]; #IO_L13N_T2_MRCC_34 Sch=pio[43]

#set_property -dict { PACKAGE_PIN U3 IOSTANDARD LVCMOS33 } [get_ports { pio44 }]; #IO_L9P_T1_DQS_34 Sch=pio[44]

#set_property -dict { PACKAGE_PIN U7 IOSTANDARD LVCMOS33 } [get_ports { pio45 }]; #IO_L19P_T3_34 Sch=pio[45]

#set_property -dict { PACKAGE_PIN W7 IOSTANDARD LVCMOS33 } [get_ports { pio46 }]; #IO_L13P_T2_MRCC_34 Sch=pio[46]

#set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS33 } [get_ports { pio47 }]; #IO_L14P_T2_SRCC_34 Sch=pio[47]

#set_property -dict { PACKAGE_PIN V8 IOSTANDARD LVCMOS33 } [get_ports { pio48 }]; #IO_L14N_T2_SRCC_34 Sch=pio[48]
```

Raspberry Pi Programs (C++)

6. target_game.cpp

```
/*
* ■■■■■■■■■■■■■■lgpio■■■■
* END■■■■■■■■■■■■■■■■■■■■■■■■■
*/

#include <iostream>
#include <thread>
#include <chrono>
#include <vector>
#include <map>
#include <fstream>
#include <cstdio>
#include <deque>
#include <numeric>
#include <iomanip>

#include <opencv2/opencv.hpp>
#include <lgpio.h>

const int GPIO_PINS[5] = {17, 27, 22, 23, 24};
const int NEXT_BUTTON_PIN = 25;
const int END_BUTTON_PIN = 20;

enum DisplayPattern {
STANDBY = 0b00000,
EXPLANATION = 0b00001,
GAME_PLAYING = 0b00010,
HIT_AREA_0 = 0b00101,
HIT_AREA_1 = 0b00110,
HIT_AREA_2 = 0b00111,
HIT_AREA_3 = 0b00100,
HIT_AREA_4 = 0b01000,
HIT_AREA_5 = 0b01001,
HIT_AREA_6 = 0b00011,
HIT_AREA_7 = 0b01010,
HIT_AREA_8 = 0b01011,

MISS = 0b01100,

RESULT_00 = 0b01101, RESULT_03 = 0b01110, RESULT_05 = 0b01111, RESULT_06 = 0b10000,

RESULT_08 = 0b10001, RESULT_09 = 0b10010, RESULT_10 = 0b10011, RESULT_11 = 0b10100,

RESULT_13 = 0b10101, RESULT_15 = 0b10110, RESULT_16 = 0b10111, RESULT_18 = 0b11000,

RESULT_20 = 0b11001, RESULT_23 = 0b11010, RESULT_25 = 0b11011, RESULT_30 = 0b11100

};

enum GameState {

STATE_STANDBY, STATE_EXPLANATION, STATE_PLAYING, STATE_HIT_EFFECT, STATE_RESULT

};

struct GameConfig {

static const int MAX_BALLS = 3;

static constexpr double HIT_EFFECT_DURATION = 2.0;

static constexpr double RESULT_DISPLAY_DURATION = 5.0;

static const std::map<int, int> AREA_SCORES;

};

const std::map<int, int> GameConfig::AREA_SCORES = {

{0, 10}, {1, 3}, {2, 10}, {3, 3}, {4, 5}, {5, 3}, {6, 10}, {7, 3}, {8, 10}

};

class BallDetector {

private:

cv::VideoCapture cap;

cv::Mat frame, hsv, mask;

const int GRID_SIZE = 3;

const int MISS_PANEL = 9;

const int MIN_BALL_AREA = 50;

const int MAX_BALL_AREA = 10000;

const cv::Scalar YELLOW_LOWER = cv::Scalar(15, 80, 30);

const cv::Scalar YELLOW_UPPER = cv::Scalar(35, 255, 255);

struct TrajectoryPoint {

double timestamp;

cv::Point position;

double area;

};

std::vector<TrajectoryPoint> trajectory;
```

```
std::chrono::steady_clock::time_point tracking_start_time;
bool is_tracking = false;
int frames_without_detection = 0;
const int MAX_FRAMES_WITHOUT_DETECTION = 30;
std::vector<cv::Point2f> src_points, dst_points;
cv::Mat perspective_matrix;
bool calibrated = false;
int detected_result = -1;
bool detection_complete = false;

public:
BallDetector() {
cap.open(0);
if (!cap.isOpened()) {
std::cerr << "Error: Cannot open camera" << std::endl;
return;
}
cap.set(cv::CAP_PROP_FRAME_WIDTH, 640);
cap.set(cv::CAP_PROP_FRAME_HEIGHT, 480);
cap.set(cv::CAP_PROP_FPS, 30);
setupPerspectiveTransform();
}

void setupPerspectiveTransform() {
src_points = {cv::Point2f(50, 50), cv::Point2f(590, 80), cv::Point2f(30, 430), cv::Point2f(610, 460)};
dst_points = {cv::Point2f(0, 0), cv::Point2f(600, 0), cv::Point2f(0, 600), cv::Point2f(600, 600)};
if (loadCalibration()) {
perspective_matrix = cv::getPerspectiveTransform(src_points, dst_points);
calibrated = true;
std::cout << "[Camera] Calibration loaded" << std::endl;
} else {
perspective_matrix = cv::getPerspectiveTransform(src_points, dst_points);
calibrated = false;
}
}

bool loadCalibration() {
std::ifstream file("/tmp/calibration.txt");
if (!file.is_open()) return false;
```

```
src_points.clear();

for (int i = 0; i < 4; i++) {

float x, y;

if (!(file >> x >> y)) return false;

src_points.push_back(cv::Point2f(x, y));

}

file.close();

return true;

}


cv::Point detectBall(const cv::Mat& input_frame, double& area_out) {

cv::Mat working_frame;

if (calibrated) {

cv::warpPerspective(input_frame, working_frame, perspective_matrix, cv::Size(600, 600));

} else {

working_frame = input_frame.clone();

}


cv::cvtColor(working_frame, hsv, cv::COLOR_BGR2HSV);

cv::Mat blue_mask;

cv::inRange(hsv, cv::Scalar(100, 100, 100), cv::Scalar(140, 255, 255), blue_mask);

cv::bitwise_not(blue_mask, blue_mask);

cv::inRange(hsv, YELLOW_LOWER, YELLOW_UPPER, mask);

cv::bitwise_and(mask, blue_mask, mask);


cv::Mat kernel = cv::getStructuringElement(cv::MORPH_ELLIPSE, cv::Size(3, 3));

cv::morphologyEx(mask, mask, cv::MORPH_OPEN, kernel, cv::Point(-1,-1), 1);

cv::morphologyEx(mask, mask, cv::MORPH_CLOSE, kernel, cv::Point(-1,-1), 1);


std::vector<std::vector<cv::Point>> contours;

cv::findContours(mask, contours, cv::RETR_EXTERNAL, cv::CHAIN_APPROX_SIMPLE);


cv::Point ball_center(-1, -1);

double max_area = 0;


for (const auto& contour : contours) {

double area = cv::contourArea(contour);

if (area >= MIN_BALL_AREA && area <= MAX_BALL_AREA && area > max_area) {

double perimeter = cv::arcLength(contour, true);

double circularity = 4 * CV_PI * area / (perimeter * perimeter);

if (circularity > 0.5) {
```

```
cv::Moments m = cv::moments(contour);

if (m.m00 > 0) {

ball_center.x = static_cast<int>(m.m10 / m.m00);

ball_center.y = static_cast<int>(m.m01 / m.m00);

max_area = area;

}

}

}

}

area_out = max_area;

return ball_center;

}


cv::Point analyzeTrajectoryAndFindImpact() {

if (trajectory.size() < 5) return cv::Point(-1, -1);

int turning_point_idx = -1;

for (size_t i = 1; i < trajectory.size() - 1; i++) {

int dx_before = trajectory[i].position.x - trajectory[i-1].position.x;

int dx_after = trajectory[i+1].position.x - trajectory[i].position.x;

if (dx_before < 0 && dx_after > 0) {

turning_point_idx = i;

break;

}

}

if (turning_point_idx == -1) {

int min_x = trajectory[0].position.x;

turning_point_idx = 0;

for (size_t i = 1; i < trajectory.size(); i++) {

if (trajectory[i].position.x < min_x) {

min_x = trajectory[i].position.x;

turning_point_idx = i;

}

}

}

return trajectory[turning_point_idx].position;

}


int getPanelNumber(const cv::Point& position, const cv::Size& frame_size) {

if (position.x < 0 || position.y < 0) return -1;
```

```
int margin_x = frame_size.width * 0.30;
int margin_y = frame_size.height * 0.25;
if (position.x < margin_x || position.x >= frame_size.width - margin_x ||
position.y < margin_y || position.y >= frame_size.height - margin_y) {
return MISS_PANEL;
}
int adjusted_x = position.x - margin_x;
int adjusted_y = position.y - margin_y;
int target_width = frame_size.width - 2 * margin_x;
int target_height = frame_size.height - 2 * margin_y;
int panel_x = adjusted_x * GRID_SIZE / target_width;
int panel_y = adjusted_y * GRID_SIZE / target_height;
if (panel_x >= GRID_SIZE || panel_y >= GRID_SIZE || panel_x < 0 || panel_y < 0) {
return MISS_PANEL;
}
int panel_num = panel_y * GRID_SIZE + panel_x;
std::cout << "[Camera] ■■■■: (" << position.x << ", " << position.y << ")" << std::endl;
std::cout << "[Camera] ■■■■: (x=" << panel_x << ", y=" << panel_y << ")" << std::endl;
std::cout << "[Camera] ■■■■■: " << panel_num << std::endl;
std::cout << "[Camera] 3x3■■:" << std::endl;
for (int y = 0; y < 3; y++) {
std::cout << " ";
for (int x = 0; x < 3; x++) {
int area = y * 3 + x;
if (area == panel_num) {
std::cout << "[" << area << "] ";
} else {
std::cout << " " << area << " ";
}
}
std::cout << std::endl;
}
return panel_num;
}

void startDetection() {
trajectory.clear();
is_tracking = false;
```

```
frames_without_detection = 0;

detected_result = -1;

detection_complete = false;

std::cout << "[Camera] Ball detection started" << std::endl;

}


bool processFrame() {

if (!cap.read(frame)) return false;

double area = 0;

cv::Point ball_pos = detectBall(frame, area);

if (ball_pos.x >= 0 && ball_pos.y >= 0) {

if (!is_tracking) {

is_tracking = true;

trajectory.clear();

tracking_start_time = std::chrono::steady_clock::now();

}

auto now = std::chrono::steady_clock::now();

double elapsed = std::chrono::duration<double>(now - tracking_start_time).count();

TrajectoryPoint point;

point.timestamp = elapsed;

point.position = ball_pos;

point.area = area;

trajectory.push_back(point);

frames_without_detection = 0;

} else if (is_tracking) {

frames_without_detection++;

if (frames_without_detection >= MAX_FRAMES_WITHOUT_DETECTION) {

std::cout << "[Camera] Tracking ended, analyzing..." << std::endl;

cv::Point impact_point = analyzeTrajectoryAndFindImpact();

if (impact_point.x >= 0 && impact_point.y >= 0) {

cv::Size frame_size = calibrated ? cv::Size(600, 600) : frame.size();

detected_result = getPanelNumber(impact_point, frame_size);

detection_complete = true;

std::cout << "[Camera] Detection complete: Panel " << detected_result << std::endl;

return false;

}

is_tracking = false;

trajectory.clear();
frames_without_detection = 0;
}
}
return true;
}

int getDetectedResult() const { return detected_result; }
bool isDetectionComplete() const { return detection_complete; }
void cleanup() { cap.release(); }
};

class TargetGameController {
private:
bool use_gpio;
GameState current_state;
int ball_count;
std::vector<int> scores;
int total_score;
BallDetector ball_detector;
int gpio_handle = -1;

void initGPIO() {
if (!use_gpio) return;
gpio_handle = lgGpiochipOpen(0);
if (gpio_handle < 0) {
std::cerr << "[GPIO] Failed to open gpiochip0: " << gpio_handle << std::endl;
return;
}
for (int pin : GPIO_PINS) {
lgGpioClaimOutput(gpio_handle, 0, pin, 0);
}
lgGpioClaimInput(gpio_handle, 0, NEXT_BUTTON_PIN);
std::cout << "[GPIO] NEXT button configured on BCM " << NEXT_BUTTON_PIN << " (external pull-up)" << std::endl;
lgGpioClaimInput(gpio_handle, 0, END_BUTTON_PIN);
std::cout << "[GPIO] END button configured on BCM " << END_BUTTON_PIN << " (external pull-up)" << std::endl;
std::cout << "[GPIO] Initialized successfully (lgpio)" << std::endl;
std::cout << "[GPIO] Output pins: ";
for (int pin : GPIO_PINS) std::cout << pin << " ";
std::cout << std::endl;
```

```
}

std::string getPatternName(DisplayPattern pattern) {
switch (pattern) {
case STANDBY: return "STANDBY";
case EXPLANATION: return "EXPLANATION";
case GAME_PLAYING: return "GAME_PLAYING";
case MISS: return "MISS";
default: return "HIT/RESULT";
}
}

std::string getStateName(GameState state) {
switch (state) {
case STATE_STANDBY: return "■■■■";
case STATE_EXPLANATION: return "■■■■";
case STATE_PLAYING: return "■■■■";
case STATE_HIT_EFFECT: return "■■■■■";
case STATE_RESULT: return "■■■■";
default: return "■■";
}
}

public:
TargetGameController(bool enable_gpio = false)
: use_gpio(enable_gpio), current_state(STATE_STANDBY), ball_count(0), total_score(0) {
if (use_gpio) initGPIO();
}

void sendDisplayPattern(DisplayPattern pattern) {
std::cout << "[DISPLAY] ■■■■■■: " << getPatternName(pattern) << " (0b";
for (int i = 4; i >= 0; i--) std::cout << ((pattern >> i) & 1);
std::cout << " = " << static_cast<int>(pattern) << ")";
if (use_gpio && gpio_handle >= 0) {
std::cout << " -> GPIO [";
for (int i = 0; i < 5; i++) {
int bit_value = (pattern >> i) & 1;
lgGpioWrite(gpio_handle, GPIO_PINS[i], bit_value);
std::cout << "pin" << GPIO_PINS[i] << "=" << bit_value;
if (i < 4) std::cout << ", ";
}

std::cout << "]";

}

std::cout << std::endl;

}


bool isNextButtonPressed() {

if (use_gpio && gpio_handle >= 0) {

int value = lgGpioRead(gpio_handle, NEXT_BUTTON_PIN);

return (value == 0);

}

return false;

}


bool isEndButtonPressed() {

if (use_gpio && gpio_handle >= 0) {

int value = lgGpioRead(gpio_handle, END_BUTTON_PIN);

return (value == 0);

}

return false;

}


bool checkEndButton() {

static int call_count = 0;

call_count++;

if (call_count <= 5 || call_count % 20 == 0) {

std::cout << "[DEBUG] checkEndButton() ■■■■ (" << call_count << "■■)" << std::endl;

}

if (use_gpio && gpio_handle >= 0) {

int button_value = lgGpioRead(gpio_handle, END_BUTTON_PIN);

if (call_count <= 5) {

std::cout << "[DEBUG] END button value: " << button_value << std::endl;

}

if (button_value == 0) {

std::cout << "\n[DEBUG] END■■■■■!" << std::endl;

std::this_thread::sleep_for(std::chrono::milliseconds(50));

while (lgGpioRead(gpio_handle, END_BUTTON_PIN) == 0) {

std::this_thread::sleep_for(std::chrono::milliseconds(10));

}
```

```
std::cout << "[INPUT] END■■■■■■■■■■ -> ■■■■■■■■■" << std::endl;

return true;

}

}

return false;

}


char waitForInput() {

if (use_gpio && gpio_handle >= 0) {

std::cout << ">>> ■■■■■■ (NEXT or END)..." << std::endl;

while (true) {

if (isNextButtonPressed()) {

std::this_thread::sleep_for(std::chrono::milliseconds(50));

while (isNextButtonPressed()) {

std::this_thread::sleep_for(std::chrono::milliseconds(10));

}

std::cout << "[INPUT] NEXT■■■■■■■■■■" << std::endl;

return 'n';

}

if (isEndButtonPressed()) {

std::this_thread::sleep_for(std::chrono::milliseconds(50));

while (isEndButtonPressed()) {

std::this_thread::sleep_for(std::chrono::milliseconds(10));

}

std::cout << "[INPUT] END■■■■■■■■■■" << std::endl;

return 'e';

}

std::this_thread::sleep_for(std::chrono::milliseconds(50));

}

} else {

std::cout << ">>> ■■■■■■ (n:NEXT, e:END): ";

char input;

std::cin >> input;

return input;

}

}


int getHitPosition() {

std::cout << "[INPUT] ■■■■■■■■■..." << std::endl;
```

```
std::cout << "[DEBUG] ========== ■■■■■■■■■■ ==========" << std::endl;
ball_detector.startDetection();

int loop_count = 0;
while (ball_detector.processFrame()) {
loop_count++;
if (loop_count <= 10 || loop_count % 100 == 0) {
std::cout << "[DEBUG] ■■■■■■■■ (" << loop_count << "■■)" << std::endl;
}
if (checkEndButton()) {
std::cout << "[DEBUG] END■■■■■■■■" << std::endl;
return -2;
}
std::this_thread::sleep_for(std::chrono::milliseconds(10));
}

std::cout << "[DEBUG] ========== ■■■■■■■ ==========" << std::endl;
int result = ball_detector.getDetectedResult();
if (result >= 0 && result <= 9) {
std::cout << "[INPUT] ■■■■: " << result << std::endl;
return result;
} else {
std::cout << "[INPUT] ■■■■: ■■■■■" << std::endl;
return 9;
}
}

DisplayPattern getHitPattern(int area) {
switch (area) {
case 0: return static_cast<DisplayPattern>(0b00101);
case 1: return static_cast<DisplayPattern>(0b00110);
case 2: return static_cast<DisplayPattern>(0b00111);
case 3: return static_cast<DisplayPattern>(0b00100);
case 4: return static_cast<DisplayPattern>(0b01000);
case 5: return static_cast<DisplayPattern>(0b01001);
case 6: return static_cast<DisplayPattern>(0b00011);
case 7: return static_cast<DisplayPattern>(0b01010);
case 8: return static_cast<DisplayPattern>(0b01011);
default: return MISS;
```

```
}

}

DisplayPattern getResultPattern(int score) {

static const std::map<int, DisplayPattern> result_map = {

{0, RESULT_00}, {3, RESULT_03}, {5, RESULT_05}, {6, RESULT_06},

{8, RESULT_08}, {9, RESULT_09}, {10, RESULT_10}, {11, RESULT_11},

{13, RESULT_13}, {15, RESULT_15}, {16, RESULT_16}, {18, RESULT_18},

{20, RESULT_20}, {23, RESULT_23}, {25, RESULT_25}, {30, RESULT_30}

};

auto it = result_map.find(score);

return (it != result_map.end()) ? it->second : RESULT_00;

}

void resetGame() {

ball_count = 0;

scores.clear();

total_score = 0;

current_state = STATE_STANDBY;

std::cout << "\n" << std::string(50, '=') << std::endl;

std::cout << "■■■■■■■ -> ■■■■" << std::endl;

std::cout << std::string(50, '=') << "\n" << std::endl;

sendDisplayPattern(STANDBY);

}

void handleStandbyState(char input) {

if (input == 'n') {

std::cout << "\n[STATE] ■■■■ -> ■■■■" << std::endl;

current_state = STATE_EXPLANATION;

sendDisplayPattern(EXPLANATION);

}

}

void handleExplanationState(char input) {

if (input == 'n') {

std::cout << "\n[STATE] ■■■■ -> ■■■■■1■■■■■" << std::endl;

current_state = STATE_PLAYING;

sendDisplayPattern(GAME_PLAYING);

} else if (input == 'e') {

resetGame();
```

```
}

}

void handlePlayingState() {

std::cout << "\n[DEBUG] ========== handlePlayingState() ■■ ==========" << std::endl;

int hit_result = getHitPosition();

if (hit_result == -2) {

std::cout << "[DEBUG] END■■■■■■■■■■■" << std::endl;

resetGame();

return;

}

ball_count++;

std::cout << "\n[PLAY] " << ball_count << "■■■■■■" << std::endl;

if (hit_result == 9) {

std::cout << " ■■: ■■■ (0■)" << std::endl;

sendDisplayPattern(MISS);

scores.push_back(0);

} else {

int score = GameConfig::AREA_SCORES.at(hit_result);

std::cout << " ■■: ■■■" << hit_result << "■■■ (" << score << "■)" << std::endl;

DisplayPattern hit_pattern = getHitPattern(hit_result);

sendDisplayPattern(hit_pattern);

scores.push_back(score);

}

current_state = STATE_HIT_EFFECT;

std::cout << "[DEBUG] ========== ■■■■■■■ ==========" << std::endl;

auto start_time = std::chrono::steady_clock::now();

auto effect_duration = std::chrono::milliseconds(static_cast<int>(GameConfig::HIT_EFFECT_DURATION * 1000));

int check_count = 0;

while (std::chrono::steady_clock::now() - start_time < effect_duration) {

check_count++;

if (check_count <= 10 || check_count % 10 == 0) {

auto elapsed = std::chrono::duration_cast<std::chrono::milliseconds>(

std::chrono::steady_clock::now() - start_time).count();

std::cout << "[DEBUG] ■■■■■■ " << check_count << "■■ (■■: " << elapsed << "ms)" << std::endl;
}

if (checkEndButton()) {

std::cout << "[DEBUG] END■■■■■■■■■■■■■■■■" << std::endl;

resetGame();

return;

}

std::this_thread::sleep_for(std::chrono::milliseconds(50));

}


std::cout << "[DEBUG] ========== ■■■■■■■ ==========" << std::endl;


if (current_state != STATE_HIT_EFFECT) {

return;

}


total_score = 0;

for (int s : scores) total_score += s;

std::cout << " ■■■■■: " << total_score << "■ (" << ball_count << "/3■)" << std::endl;


if (ball_count >= GameConfig::MAX_BALLS) {

std::cout << "\n[STATE] ■■■■■ -> ■■■■" << std::endl;

current_state = STATE_RESULT;

} else {

std::cout << "\n[STATE] ■■■■■ -> ■■■■■" << (ball_count + 1) << "■■■■■" << std::endl;

current_state = STATE_PLAYING;

sendDisplayPattern(GAME_PLAYING);

}

}


void showResult() {

std::cout << "\n[RESULT] ■■■■" << std::endl;

std::cout << "■■■■: ";

for (size_t i = 0; i < scores.size(); i++) {

std::cout << scores[i];

if (i < scores.size() - 1) std::cout << ", ";

}

std::cout << std::endl;

std::cout << "■■■■: " << total_score << "■" << std::endl;

DisplayPattern result_pattern = getResultPattern(total_score);

sendDisplayPattern(result_pattern);
```

```
auto start_time = std::chrono::steady_clock::now();
auto result_duration = std::chrono::milliseconds(static_cast<int>(GameConfig::RESULT_DISPLAY_DURATION * 1000));

while (std::chrono::steady_clock::now() - start_time < result_duration) {
if (checkEndButton()) {
resetGame();
return;
}
std::this_thread::sleep_for(std::chrono::milliseconds(50));
}
resetGame();
}

void runGame() {
std::cout << "\n" << std::string(50, '=') << std::endl;
std::cout << "■■■■■■■■" << std::endl;
std::cout << "■■: n=NEXT■■■, e=END■■■" << std::endl;
std::cout << std::string(50, '=') << "\n" << std::endl;
sendDisplayPattern(STANDBY);

while (true) {
std::cout << "\n■■■■■: " << getStateName(current_state) << std::endl;
if (current_state == STATE_STANDBY) {
char input = waitForInput();
handleStandbyState(input);
} else if (current_state == STATE_EXPLANATION) {
char input = waitForInput();
handleExplanationState(input);
} else if (current_state == STATE_PLAYING) {
handlePlayingState();
} else if (current_state == STATE_RESULT) {
showResult();
}
}
}

void cleanup() {
ball_detector.cleanup();
if (gpio_handle >= 0) {
lgGpiochipClose(gpio_handle);
std::cout << "[GPIO] Closed" << std::endl;
}
std::cout << "\n■■■■■■■" << std::endl;
}
};

int main() {
try {
TargetGameController controller(true); // true: GPIO■■
controller.runGame();
} catch (const std::exception& e) {
std::cerr << "■■■: " << e.what() << std::endl;
return -1;
}
return 0;
}
```

7. ball_detector_4.cpp

```
#include <opencv2/opencv.hpp>
#include <iostream>
#include <deque>
#include <vector>
#include <numeric>
#include <fstream>
#include <iomanip>
#include <chrono>

class BallDetector {
private:
cv::VideoCapture cap;
cv::Mat frame, hsv, mask, corrected_frame;

// ■■■■■■■
const int GRID_SIZE = 3;
const int TOTAL_PANELS = 9;
const int MISS_PANEL = 9;
const int MIN_BALL_AREA = 50;
const int MAX_BALL_AREA = 10000;

// HSV■■■
const cv::Scalar YELLOW_LOWER = cv::Scalar(15, 80, 30);
const cv::Scalar YELLOW_UPPER = cv::Scalar(35, 255, 255);

// ■■■■■
struct TrajectoryPoint {
double timestamp;
cv::Point position;
double area;
};

std::vector<TrajectoryPoint> trajectory;
std::chrono::steady_clock::time_point tracking_start_time;
bool is_tracking = false;
int frames_without_detection = 0;
const int MAX_FRAMES_WITHOUT_DETECTION = 30; // 1■■■■■■■■■■■

// ■■■■■■■■■■
```

```
std::vector<cv::Point2f> src_points;
std::vector<cv::Point2f> dst_points;
cv::Mat perspective_matrix;
bool calibrated = false;
std::vector<cv::Point2f> clicked_points;
cv::Mat calibration_frame;

public:
BallDetector() {
cap.open(0);
if (!cap.isOpened()) {
std::cerr << "Error: Cannot open camera" << std::endl;
return;
}

cap.set(cv::CAP_PROP_FRAME_WIDTH, 640);
cap.set(cv::CAP_PROP_FRAME_HEIGHT, 480);
cap.set(cv::CAP_PROP_FPS, 30);
cap.set(cv::CAP_PROP_BRIGHTNESS, 0.5);
cap.set(cv::CAP_PROP_CONTRAST, 1.2);
cap.set(cv::CAP_PROP_SATURATION, 1.3);

setupPerspectiveTransform();

std::cout << "Ball Detector initialized (Trajectory Analysis Mode)" << std::endl;
std::cout << "========================================" << std::endl;
std::cout << "Controls:" << std::endl;
std::cout << " 'c' - Start calibration" << std::endl;
std::cout << " 'r' - Reset tracking" << std::endl;
std::cout << " 'q' - Quit" << std::endl;
std::cout << "========================================" << std::endl;
}

void setupPerspectiveTransform() {
src_points = {
cv::Point2f(50, 50),
cv::Point2f(590, 80),
cv::Point2f(30, 430),
cv::Point2f(610, 460)
};
```

```
dst_points = {
cv::Point2f(0, 0),
cv::Point2f(600, 0),
cv::Point2f(0, 600),
cv::Point2f(600, 600)
};

if (loadCalibration()) {
perspective_matrix = cv::getPerspectiveTransform(src_points, dst_points);
calibrated = true;
std::cout << "Loaded saved calibration data" << std::endl;
} else {
perspective_matrix = cv::getPerspectiveTransform(src_points, dst_points);
calibrated = false;
std::cout << "No saved calibration found. Press 'c' to calibrate." << std::endl;
}
}

static void onMouse(int event, int x, int y, int flags, void* userdata) {
BallDetector* detector = static_cast<BallDetector*>(userdata);

if (event == cv::EVENT_LBUTTONDOWN) {
detector->clicked_points.push_back(cv::Point2f(x, y));
std::cout << "Point " << detector->clicked_points.size()
<< ": (" << x << ", " << y << ")" << std::endl;

cv::circle(detector->calibration_frame, cv::Point(x, y), 8, cv::Scalar(0, 255, 0), -1);
cv::putText(detector->calibration_frame, std::to_string(detector->clicked_points.size()),
cv::Point(x + 10, y - 10), cv::FONT_HERSHEY_SIMPLEX, 1, cv::Scalar(0, 255, 0), 2);
cv::imshow("Calibration", detector->calibration_frame);

if (detector->clicked_points.size() == 4) {
std::cout << "Calibration complete! Press any key to continue..." << std::endl;
}
}
}

bool loadCalibration() {
std::ifstream file("/tmp/calibration.txt");
if (!file.is_open()) return false;
```

```
src_points.clear();
for (int i = 0; i < 4; i++) {
float x, y;
if (!(file >> x >> y)) return false;
src_points.push_back(cv::Point2f(x, y));
}

file.close();
return true;
}

void saveCalibration() {
std::ofstream file("/tmp/calibration.txt");
if (!file.is_open()) {
std::cerr << "Failed to save calibration" << std::endl;
return;
}

for (const auto& point : src_points) {
file << point.x << " " << point.y << std::endl;
}

file.close();
std::cout << "Calibration saved to /tmp/calibration.txt" << std::endl;
}

void calibratePerspective() {
std::cout << "\n========== CALIBRATION MODE ==========" << std::endl;
std::cout << "Click on the 4 corners of the LED matrix in this order:" << std::endl;
std::cout << "1. Top-Left corner" << std::endl;
std::cout << "2. Top-Right corner" << std::endl;
std::cout << "3. Bottom-Left corner" << std::endl;
std::cout << "4. Bottom-Right corner" << std::endl;
std::cout << "======================================\n" << std::endl;

clicked_points.clear();

if (!cap.read(calibration_frame)) {
std::cerr << "Failed to read frame for calibration" << std::endl;
return;
}
```

```
cv::namedWindow("Calibration");

cv::setMouseCallback("Calibration", onMouse, this);

cv::imshow("Calibration", calibration_frame);


while (clicked_points.size() < 4) {

char key = cv::waitKey(100);

if (key == 'q') {

std::cout << "Calibration cancelled" << std::endl;

cv::destroyWindow("Calibration");

return;

}

}


cv::waitKey(0);

cv::destroyWindow("Calibration");


src_points = clicked_points;

perspective_matrix = cv::getPerspectiveTransform(src_points, dst_points);

calibrated = true;

saveCalibration();


std::cout << "\nCalibration completed and saved!" << std::endl;

}


cv::Point detectBall(const cv::Mat& input_frame, double& area_out) {

cv::Mat working_frame;


if (calibrated) {

cv::warpPerspective(input_frame, working_frame, perspective_matrix, cv::Size(600, 600));

} else {

working_frame = input_frame.clone();

}


cv::cvtColor(working_frame, hsv, cv::COLOR_BGR2HSV);


cv::Mat blue_mask;

cv::inRange(hsv, cv::Scalar(100, 100, 100), cv::Scalar(140, 255, 255), blue_mask);

cv::bitwise_not(blue_mask, blue_mask);


cv::inRange(hsv, YELLOW_LOWER, YELLOW_UPPER, mask);

cv::bitwise_and(mask, blue_mask, mask);
```

```
cv::Mat kernel = cv::getStructuringElement(cv::MORPH_ELLIPSE, cv::Size(3, 3));

cv::morphologyEx(mask, mask, cv::MORPH_OPEN, kernel, cv::Point(-1,-1), 1);

cv::morphologyEx(mask, mask, cv::MORPH_CLOSE, kernel, cv::Point(-1,-1), 1);


std::vector<std::vector<cv::Point>> contours;

cv::findContours(mask, contours, cv::RETR_EXTERNAL, cv::CHAIN_APPROX_SIMPLE);


cv::Point ball_center(-1, -1);

double max_area = 0;


for (const auto& contour : contours) {

double area = cv::contourArea(contour);


if (area >= MIN_BALL_AREA && area <= MAX_BALL_AREA && area > max_area) {

double perimeter = cv::arcLength(contour, true);

double circularity = 4 * CV_PI * area / (perimeter * perimeter);


if (circularity > 0.5) {

cv::Moments m = cv::moments(contour);

if (m.m00 > 0) {

ball_center.x = static_cast<int>(m.m10 / m.m00);

ball_center.y = static_cast<int>(m.m01 / m.m00);

max_area = area;


cv::circle(working_frame, ball_center, 15, cv::Scalar(0, 0, 255), 3);

cv::putText(working_frame, "Area: " + std::to_string((int)area),

cv::Point(ball_center.x + 20, ball_center.y),

cv::FONT_HERSHEY_SIMPLEX, 0.5, cv::Scalar(255, 255, 255), 1);

}

}

}

}


area_out = max_area;


cv::imshow("Ball Detection", working_frame);

cv::imshow("Mask", mask);


return ball_center;

}


cv::Point analyzeTrajectoryAndFindImpact() {
```

```
if (trajectory.size() < 5) {

std::cout << "Not enough data points: " << trajectory.size() << std::endl;

return cv::Point(-1, -1);

}


std::cout << "\n========== TRAJECTORY ANALYSIS ==========" << std::endl;

std::cout << "Total tracking points: " << trajectory.size() << std::endl;


// X■■■■■■■■■■■■■■→■■■■■■■■■■

int turning_point_idx = -1;


for (size_t i = 1; i < trajectory.size() - 1; i++) {

int dx_before = trajectory[i].position.x - trajectory[i-1].position.x;

int dx_after = trajectory[i+1].position.x - trajectory[i].position.x;


// ■■■■■■■■■■■■■■■■■■■■■ = ■■■

if (dx_before < 0 && dx_after > 0) {

turning_point_idx = i;

std::cout << "X-coordinate turning point detected at frame: " << i << std::endl;

std::cout << " dx_before: " << dx_before << " (decreasing)" << std::endl;

std::cout << " dx_after: " << dx_after << " (increasing)" << std::endl;

break;

}

}


// ■■■■■■■■■■■■■■■■■■■■■■■■

if (turning_point_idx == -1) {

std::cout << "No clear turning point found. Using minimum X value as fallback." << std::endl;

int min_x = trajectory[0].position.x;

turning_point_idx = 0;


for (size_t i = 1; i < trajectory.size(); i++) {

if (trajectory[i].position.x < min_x) {

min_x = trajectory[i].position.x;

turning_point_idx = i;

}

}

}


cv::Point impact_point = trajectory[turning_point_idx].position;


std::cout << "\nImpact point at frame: " << turning_point_idx << std::endl;
```

```
std::cout << " Position: (" << impact_point.x << ", " << impact_point.y << ")" << std::endl;

std::cout << " Timestamp: " << std::fixed << std::setprecision(3)

<< trajectory[turning_point_idx].timestamp << " sec" << std::endl;


// ■■■■■■■■■■■■■■■■■

if (turning_point_idx > 0) {

std::cout << " Previous frame: (" << trajectory[turning_point_idx-1].position.x

<< ", " << trajectory[turning_point_idx-1].position.y << ")" << std::endl;

}

if (turning_point_idx < (int)trajectory.size() - 1) {

std::cout << " Next frame: (" << trajectory[turning_point_idx+1].position.x

<< ", " << trajectory[turning_point_idx+1].position.y << ")" << std::endl;

}


std::cout << "========================================\n" << std::endl;


return impact_point;

}


int getPanelNumber(const cv::Point& position, const cv::Size& frame_size) {

if (position.x < 0 || position.y < 0) return -1;


int margin_x = frame_size.width * 0.30;

int margin_y = frame_size.height * 0.25;


if (position.x < margin_x || position.x >= frame_size.width - margin_x ||

position.y < margin_y || position.y >= frame_size.height - margin_y) {

return MISS_PANEL;

}


int adjusted_x = position.x - margin_x;

int adjusted_y = position.y - margin_y;

int target_width = frame_size.width - 2 * margin_x;

int target_height = frame_size.height - 2 * margin_y;


int panel_x = adjusted_x * GRID_SIZE / target_width;

int panel_y = adjusted_y * GRID_SIZE / target_height;


if (panel_x >= GRID_SIZE || panel_y >= GRID_SIZE || panel_x < 0 || panel_y < 0) {

return MISS_PANEL;

}
```

```
return panel_y * GRID_SIZE + panel_x;

}


void run() {

std::cout << "Starting ball detection..." << std::endl;

std::cout << "Tracking will start automatically when ball is detected" << std::endl;


while (true) {

if (!cap.read(frame)) {

std::cerr << "Failed to read frame" << std::endl;

break;

}


double area = 0;

cv::Point ball_pos = detectBall(frame, area);


if (ball_pos.x >= 0 && ball_pos.y >= 0) {

// ■■■■■■■


if (!is_tracking) {

// ■■■■

is_tracking = true;

trajectory.clear();

tracking_start_time = std::chrono::steady_clock::now();

std::cout << "\n>>> Tracking started <<<" << std::endl;

}


// ■■■■■

auto now = std::chrono::steady_clock::now();

double elapsed = std::chrono::duration<double>(now - tracking_start_time).count();


TrajectoryPoint point;

point.timestamp = elapsed;

point.position = ball_pos;

point.area = area;

trajectory.push_back(point);


frames_without_detection = 0;


} else if (is_tracking) {

// ■■■■■■■■■

frames_without_detection++;
if (frames_without_detection >= MAX_FRAMES_WITHOUT_DETECTION) {
// ■■■■■■■■
std::cout << "\n>>> Tracking ended (no detection for "
<< MAX_FRAMES_WITHOUT_DETECTION << " frames) <<<" << std::endl;

cv::Point impact_point = analyzeTrajectoryAndFindImpact();

if (impact_point.x >= 0 && impact_point.y >= 0) {
cv::Size frame_size = calibrated ? cv::Size(600, 600) : frame.size();
int panel_num = getPanelNumber(impact_point, frame_size);

std::cout << "===========================================" << std::endl;
if (panel_num == MISS_PANEL) {
std::cout << "BALL MISSED THE TARGET (PANEL: 9)" << std::endl;
} else {
std::cout << "BALL HIT PANEL: " << panel_num << std::endl;
std::cout << "Grid Position: (" << (panel_num % GRID_SIZE)
<< ", " << (panel_num / GRID_SIZE) << ")" << std::endl;
}
std::cout << "Impact Position: (" << impact_point.x
<< ", " << impact_point.y << ")" << std::endl;
std::cout << "===========================================" << std::endl;

// ■■■■■■■■■■
std::ofstream result_file("/tmp/ball_result.txt");
if (result_file.is_open()) {
result_file << panel_num << std::endl;
result_file.close();
}
}

// ■■■■
is_tracking = false;
trajectory.clear();
frames_without_detection = 0;
}
}

// ■■■■■■
char key = cv::waitKey(30) & 0xFF;
```

```
if (key == 'q') break;

else if (key == 'c') calibratePerspective();

else if (key == 'r') {

trajectory.clear();

is_tracking = false;

frames_without_detection = 0;

std::cout << "Tracking reset" << std::endl;

}

}


cleanup();

}


void cleanup() {

cap.release();

cv::destroyAllWindows();

std::cout << "Ball detector stopped" << std::endl;

}

};


int main() {

try {

BallDetector detector;

detector.run();

} catch (const std::exception& e) {

std::cerr << "Error: " << e.what() << std::endl;

return -1;

}


return 0;

}
```

Shell Script

8. start_game.sh

```
#!/bin/bash

# ■■■■■■■■■■■■■

# ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

echo "========================================"

echo " ■■■■■■■■■■■■■"

echo "========================================"

echo ""

# ■■■■■■■■■■■■■

SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"

cd "$SCRIPT_DIR"

# ■■■■■■■■■■■■

echo "[1/4] ■■■■■■■..."

if [ ! -f "./ball_detector_4" ]; then

echo "■■■: ball_detector_4 ■■■■■■■■"

echo "■■■■■■■■■■: $SCRIPT_DIR"

exit 1

fi

if [ ! -f "./target_game" ]; then

echo "■■■: target_game ■■■■■■■■"

echo "■■■■■■■■■■: $SCRIPT_DIR"

exit 1

fi

echo "✓ ■■■■■■■■■■■■■■"

echo ""

# ■■■■■■■■■■■■■■■■

echo "[2/4] ■■■■■■■■■■■■..."

if [ -f "/tmp/calibration.txt" ]; then

echo "✓ ■■■■■■■■■■■■■■■■■■■■"
echo ""

echo "■■■■■■■■■■■■■■■■■■■■"

echo " [Y] ■■ - ■■■■■■"

echo " [N] ■■■ - ■■■■■■■■■■■■"

read -p "■■ (Y/N): " USE_EXISTING

if [[ "$USE_EXISTING" =~ ^[Yy]$ ]]; then

echo "■■■■■■■■■■■■■■■■■■"

SKIP_CALIBRATION=true

else

echo "■■■■■■■■■■■■■■■■■"

SKIP_CALIBRATION=false

fi

else

echo "! ■■■■■■■■■■■■■■■■■■■■■"

echo "■■■■■■■■■■■■■■■■"

SKIP_CALIBRATION=false

fi

echo ""

# ■■■■■■■■■■■

if [ "$SKIP_CALIBRATION" = false ]; then

echo "[3/4] ■■■■■■■■■■■..."

echo "========================================"

echo "■■■■:"

echo " 1. ■■■■■■■■■■■■"

echo " 2. 'c'■■■■■■■■■■■■■■■■■"

echo " 3. LED■■■4■■■■■■■■■:"

echo " ① ■■"

echo " ② ■■"

echo " ③ ■■"

echo " ④ ■■"

echo " 4. Enter■■■■■■■■"

echo " 5. 'q'■■■■■■■■■■■■■■"

echo "========================================"

echo ""

read -p "■■■■■■■Enter■■■■■■■■■■..."

# ball_detector_4■■■
```

```
./ball_detector_4

# ■■■■■■■■■■■■■■■■■
if [ ! -f "/tmp/calibration.txt" ]; then
echo ""
echo "■■■: ■■■■■■■■■■■■■■■■■■"
echo "■■■■■"
exit 1
fi

echo ""
echo "✓ ■■■■■■■■■■■"
echo ""
else
echo "[3/4] ■■■■■■■■■■■■■■"
echo ""
fi

# ■■■■■■■■
echo "[4/4] ■■■■■■■■■■■..."
echo "========================================"
echo "■■■■:"
echo " - NEXT■■■ (BCM 25): ■■■■■■■"
echo " - END■■■ (BCM 26): ■■■■■■■"
echo " - Ctrl+C: ■■■■■■■"
echo "========================================"
echo ""
read -p "■■■■■■■Enter■■■■■■■■■■..."

# ■■■■■■■■
echo ""
echo "■ GPIO■■■■■■■■■■■■■■■"
sudo ./target_game

echo ""
echo "========================================"
echo " ■■■■■"
echo "========================================"
```