Chit

```
const std = @import("std");
const gv = @import("GlobalVariables.zig");
const print = @import("std").debug.print;
const m = @import("main.zig");

const c = @cImport({
    @cDefine("SDL_DISABLE_OLD_NAMES", {});
    @cInclude("SDL3/SDL.h");
    @cInclude("SDL3/SDL_revision.h");
    @cDefine("SDL_MAIN_HANDLED", {});
    @cInclude("SDL3/SDL_main.h");
    @cInclude("SDL3_image/SDL_image.h");
});

// ********************************************************************************
id: i32,
img_index: i32,
hex_ID: [2]i32,

// ********************************************************************************
pub const Chit = @This();

// ********************************************************************************
pub fn new(id: i32, img_index: i32, hex_ID: [2]i32) Chit {
    return .{
        .id = id,
        .img_index = img_index,
        .hex_ID = hex_ID,
    };
}

pub fn render(self: *Chit) void {
    // ***** create a surface
    const a_surf: *c.SDL_Surface = c.SDL_CreateSurface(gv.chit_square_dim, gv.chit_square_dim, c.SDL_PIXELFORMAT_RGBA8888);
    defer c.SDL_DestroySurface(a_surf);

    // ***** clip one chit and put it on the surface
    var a_rect: c.SDL_Rect = undefined;
    a_rect.x = 0;
    a_rect.y = self.img_index * gv.chit_square_dim;
    a_rect.w = gv.chit_square_dim;
    a_rect.h = gv.chit_square_dim;
    _ = c.SDL_BlitSurface(@ptrCast(m.chits_surface), &a_rect, a_surf, null); // no scaling. the target surface truncates.

    // convert the surface to a texture
    const a_texture = c.SDL_CreateTextureFromSurface(@ptrCast(m.renderer), a_surf);
    defer c.SDL_DestroyTexture(a_texture);

    // define a silly puddy rectangle and render it
    var a_rectness: c.SDL_FRect = undefined;
    const hex_ID_x: i32 = @intFromFloat(@as(f64, @floatFromInt(self.hex_ID[0])) * gv.Hex_Dim_ness[0]);
    const hex_ID_y: i32 = @intFromFloat(@as(f64, @floatFromInt(self.hex_ID[1])) * gv.Hex_Dim_ness[1]);
    const x: f32 = @as(f32, @floatFromInt(gv.Zero_Zero[0] - gv.map_loc[0] + hex_ID_x)) / gv.scaleness;
    var y: f32 = 0.0;
    if (@mod(self.hex_ID[0], 2) == 0) {
        y = @as(f32, @floatFromInt(gv.Zero_Zero[1] - gv.map_loc[1] + hex_ID_y)) / gv.scaleness;
    } else {
        y = @as(f32, @floatFromInt(gv.Zero_Zero[1] - gv.map_loc[1] + hex_ID_y + @as(i32, @intFromFloat(gv.Half_Hex_Y_ness)))) / gv.scaleness;
    }

    const w_h_ness: f32 = @as(f32, @floatFromInt(gv.chit_square_dim)) / gv.scaleness;
    a_rectness.x = x;
    a_rectness.y = y;
    a_rectness.w = w_h_ness;
    a_rectness.h = w_h_ness;
    if ((x + w_h_ness) < 0) return;
    if (x > gv.window_w) return;
    if ((y + w_h_ness) < 0) return;
    if (y > gv.window_h) return;
    _ = c.SDL_RenderTexture(@ptrCast(m.renderer), a_texture, null, &a_rectness);
}
```

joystick

```
const std = @import("std");
const print = @import("std").debug.print;
const mvzr = @import("mvzr.zig");
const m = @import("main.zig");
const gv = @import("GlobalVariables.zig");

const c = @cImport({
    @cDefine("SDL_DISABLE_OLD_NAMES", {});
    @cInclude("SDL3/SDL.h");
    @cInclude("SDL3/SDL_revision.h");
    @cDefine("SDL_MAIN_HANDLED", {});
    @cInclude("SDL3/SDL_main.h");
    @cInclude("SDL3_image/SDL_image.h");
});

// ************** Joystick states
pub var button_bits: u16 = 0;
pub var button_bits_old: u16 = 0;
pub var d_pad: u16 = 0;
pub var map_button = [_]i32{0} ** 14;
pub var map_axis = [_]i32{0} ** 6;
pub var axis_vals = [_]i32{0} ** 6;
pub var num_buttons: u32 = 0;

// ********************************************************************************************
pub fn record_events() void {
    if (gv.joystick_type == 0) return;

    // ********** clear bits & d_pad info
    button_bits_old = button_bits;
    button_bits = 0;
    d_pad = 0;

    // ********** set info bits
    for (0..num_buttons) |i| {
        if (c.SDL_GetJoystickButton(@ptrCast(m.joystick), @intCast(i))) { // if buttons are pressed
            //print("button {d}", .{i});
            const val = map_button[i];
            //print(": {d}\n", .{val});
            if (val < 0) continue;
            const bits: u16 = std.math.pow(u16, 2, @as(u16, @intCast(val)));
            button_bits |= bits;
        }
    }

    // ********** set d_pad info
    const hat = c.SDL_GetJoystickHat(@ptrCast(m.joystick), 0);
    if (hat != 0) {
        d_pad = hat;
    }

    // ********** set axis info
    for (0..6) |i| {
        if (map_axis[i] >= 0) {
            const val = c.SDL_GetJoystickAxis(@ptrCast(m.joystick), @intCast(i));
            axis_vals[@abs(map_axis[i])] = val;
        }
    }
}

// ********************************************************************************************
pub fn bind_buttons(aText: [*c]const u8) !void {
    var buffer = [_]u8{0} ** 100;
    _ = try std.fmt.bufPrintZ(&buffer, "{s}\n", .{aText});

    var regex: mvzr.Regex = mvzr.compile("RumblePad").?;
    if (regex.isMatch(&buffer)) {
        gv.joystick_type = 1;
    }

    regex = mvzr.compile("ZEROPLUS").?;
    if (regex.isMatch(&buffer)) {
        gv.joystick_type = 6;
    }

    regex = mvzr.compile("F310").?;
    if (regex.isMatch(&buffer)) {
        gv.joystick_type = 7;
    }

    regex = mvzr.compile("F710").?;
    if (regex.isMatch(&buffer)) {
        gv.joystick_type = 2;
    }

    regex = mvzr.compile("PS4").?;
    if (regex.isMatch(&buffer)) {
        gv.joystick_type = 3;
    }

    regex = mvzr.compile("SWITCH CO").?; // Sega
    if (regex.isMatch(&buffer)) {
        gv.joystick_type = 4;
    }

    regex = mvzr.compile("Xbox One").?;
    if (regex.isMatch(&buffer)) {
        gv.joystick_type = 5;
    }

    _ = try std.fmt.bufPrintZ(&buffer, "{s}\n", .{c.SDL_GetPlatform()});
```

```
regex = mvzr.compile("FreeBSD").?;
if (regex.isMatch(&buffer)) {
    gv.OS_platform = 1;
}

print("-----------> {} -- {}\n", .{ gv.joystick_type, gv.OS_platform });

// ****************************************************
// *** joystick signal to bit
// *** -1: not in use
if (gv.joystick_type == 1) {
    // ***** RumblePad 2 USB
    map_button[0] = 2; // left
    map_button[1] = 0; // down
    map_button[2] = 3; // right
    map_button[3] = 1; // up
    map_button[4] = 4; // left sholder
    map_button[5] = 5; // right sholder
    map_button[6] = 11; // left trigger
    map_button[7] = 12; // right trigger
    map_button[8] = 6; // left center
    map_button[9] = 7; // right center
    map_button[10] = 8; // left axis
    map_button[11] = 9; // right axis
    map_button[12] = -1;
    map_button[13] = -1;

    map_axis[0] = 2; // left X
    map_axis[1] = 3; // left Y
    map_axis[2] = 4; // right X
    map_axis[3] = 5; // right Y
    map_axis[4] = -1;
    map_axis[5] = -1;
}

if (gv.joystick_type == 2) {
    // ***** F710
    map_button[0] = 0; // down
    map_button[1] = 3; // right
    map_button[2] = 2; // left
    map_button[3] = 1; // up
    map_button[4] = 4; // left sholder
    map_button[5] = 5; // right sholder
    map_button[6] = 6; // center left
    map_button[7] = 7; // center right
    map_button[8] = 10; // center
    map_button[9] = 8; // left axis
    map_button[10] = 9; // right axis
    map_button[11] = -1;
    map_button[12] = -1;
    map_button[13] = -1;

    map_axis[0] = 2; // left X
    map_axis[1] = 3; // left Y
    map_axis[2] = 0; // left trigger
    map_axis[3] = 4; // right X
    map_axis[4] = 5; // right Y
    map_axis[5] = 1; // right trigger
}

if (gv.joystick_type == 3) {
    // ***** PS4
    map_button[0] = 0; // down
    map_button[1] = 3; // right
    map_button[2] = 2; // left
    map_button[3] = 1; // up
    map_button[4] = 6; // center left
    map_button[5] = 10; // center
    map_button[6] = 7; // center right
    map_button[7] = 8; // axis left
    map_button[8] = 9; // axis right
    map_button[9] = 4; // left sholder
    map_button[10] = 5; // right sholder
    map_button[11] = 13; // touch pad
    map_button[12] = -1;
    map_button[13] = -1;

    map_axis[0] = 2; // left x
    map_axis[1] = 3; // left y
    map_axis[2] = 4; // right x
    map_axis[3] = 5; // right y
    map_axis[4] = 0; // left trigger
    map_axis[5] = 1; // right trigger
}

if (gv.joystick_type == 4) {
    // ***** Sega
    map_button[0] = 1; // y (up)
    map_button[1] = 3; // B (right)
    map_button[2] = 0; // A (down)
    map_button[3] = 2; // x (left)
    map_button[4] = 4; // sholder left
    map_button[5] = 5; // sholder right
    map_button[6] = 8; // z (left axis)
    map_button[7] = 9; // C (right axis)
    map_button[8] = 7; // center right
    map_button[9] = 10; // center
    map_button[10] = -1;
    map_button[11] = -1;
    map_button[12] = 6; // center left
    map_button[13] = -1;

    map_axis[0] = 0; // d - left & right
```

```
        map_axis[1] = 1; // d - up & down
        map_axis[2] = -1; //
        map_axis[3] = -1; //
        map_axis[4] = -1; //
        map_axis[5] = -1; //

    }

    if (gv.joystick_type == 5) {
        // ***** Xbox One
        map_button[0] = 0; // down
        map_button[1] = 3; // right
        map_button[2] = 2; // left
        map_button[3] = 1; // up
        map_button[4] = 4; // sholder L
        map_button[5] = 5; // sholder R
        map_button[6] = 6; // center left
        map_button[7] = 7; // center right
        map_button[8] = 9; // left axis
        map_button[9] = 4; // right axis
        map_button[10] = 10; // heart (center)
        map_button[11] = -1; //
        map_button[12] = -1;
        map_button[13] = -1;

        map_axis[0] = 2; // left X
        map_axis[1] = 3; // left Y
        map_axis[2] = 0; // left trigger
        map_axis[3] = 4; // right X
        map_axis[4] = 5; // right Y
        map_axis[5] = 1; // right trigger
    }

    if (gv.joystick_type == 6) {
        // ***** ZeroPlus (Game:Pad 4 S)
        map_button[0] = 2; // left
        map_button[1] = 0; // down
        map_button[2] = 3; // right
        map_button[3] = 1; // up
        map_button[4] = 4; // left sholder
        map_button[5] = 5; // right sholder
        map_button[6] = -1;
        map_button[7] = -1;
        map_button[8] = 6; // center left
        map_button[9] = 7; // center right
        map_button[10] = 8; // left axis
        map_button[11] = 9; // right axis
        map_button[12] = 10; // center
        map_button[13] = -1; //

        map_axis[0] = 2; // left x
        map_axis[1] = 3; // left y
        map_axis[2] = 4; // right x
        map_axis[3] = 0; // left trigger
        map_axis[4] = 1; // right trigger
        map_axis[5] = 5; // right Y
    }

    if (gv.joystick_type == 7) {
        // ***** F310
        map_button[0] = 0; // down
        map_button[1] = 3; // right
        map_button[2] = 2; // left
        map_button[3] = 1; // up
        map_button[4] = 4; // left sholder
        map_button[5] = 5; // right sholder
        map_button[6] = 6; // center left
        map_button[7] = 7; // center right
        map_button[8] = 10; // center
        map_button[9] = 8; // left axis
        map_button[10] = 9; // right axis
        map_button[11] = -1;
        map_button[12] = -1;
        map_button[13] = -1;

        map_axis[0] = 2; // left x
        map_axis[1] = 3; // left y
        map_axis[2] = 0; // left trigger
        map_axis[3] = 4; // right x
        map_axis[4] = 5; // right y
        map_axis[5] = 1; // right trigger
    }
}
```

GlobalVariables

```
pub var OS_platform: i32 = 0; // 0: linux;   1: FreeBSD
pub var window_w: f32 = 800.0; // var because may be redefine later.
pub var window_h: f32 = 720.0; // var because may be redefine later.
pub var joystick_type: i32 = 0;

// *************** Toggles
pub var toggles: u16 = 0;
pub var toggles_old: u16 = 0;

// *************** Mapboard info
pub const Zero_Zero = [_]i32{ 293, 141 };
pub const Lower_Right = [_]i32{ 5020, 3846 };
const hex_count_row: f64 = 28.0;
const hex_count_col: f64 = 19.0;
pub const Hex_Dim_ness = [_]f64{ @as(f64, @floatFromInt((Lower_Right[0] - Zero_Zero[0]))) / hex_count_row, @as(f64, @floatFromInt((Lower_Right[1]
- Zero_Zero[1]))) / hex_count_col };
pub const Half_Hex_Y_ness: f32 = @floatCast(Hex_Dim_ness[1] / 2.0);
pub var map_loc = [_]i32{ 0, 0 };

pub var scale: i32 = 0;
pub var scaleness: f32 = 1.0;
pub var scale_old: i32 = 0;
pub var scaleness_old: f32 = 1.0;

// *************** chit
pub const chit_square_dim: i32 = 150;

// *************** Bits
pub const bit_0: u16 = 1 << 0;
pub const bit_1: u16 = 1 << 1;
pub const bit_2: u16 = 1 << 2;
pub const bit_3: u16 = 1 << 3;
pub const bit_4: u16 = 1 << 4;
pub const bit_5: u16 = 1 << 5;
pub const bit_6: u16 = 1 << 6;
pub const bit_7: u16 = 1 << 7;

pub const bit_8: u16 = 1 << 8;
pub const bit_9: u16 = 1 << 9;
pub const bit_10: u16 = 1 << 10;
pub const bit_11: u16 = 1 << 11;
pub const bit_12: u16 = 1 << 12;
pub const bit_13: u16 = 1 << 13;
pub const bit_14: u16 = 1 << 14;
pub const bit_15: u16 = 1 << 15;

// *************** Flags
// bit 0: quit game
pub var flags: u32 = 0;
```

hud_new

```
const std = @import("std");
const gv = @import("GlobalVariables.zig");
const print = @import("std").debug.print;
//const m = @import("main.zig");
const jstk = @import("joystick.zig");

const c = @cImport({
    @cDefine("SDL_DISABLE_OLD_NAMES", {});
    @cInclude("SDL3/SDL.h");
    @cInclude("SDL3/SDL_revision.h");
    @cDefine("SDL_MAIN_HANDLED", {});
    @cInclude("SDL3/SDL_main.h");
    @cInclude("SDL3_image/SDL_image.h");
});

pub fn newGame() void {
    print("New Game..............\n", .{});
    // [ newGame loop ================================================================================ ]
    main_loop: while (true) {
        var event: c.SDL_Event = undefined;
        while (c.SDL_PollEvent(&event)) {
            switch (event.type) {
                // [ Key down ============================================================== ]
                c.SDL_EVENT_KEY_DOWN => {
                    switch (event.key.scancode) {
                        c.SDL_SCANCODE_ESCAPE => {
                            break :main_loop;
                        },
                        else => {},
                    }
                },
                else => {},
            }
        }
        jstk.record_events();
        print("...{d}\n", .{jstk.button_bits});
    } // main_loop
    print("back..............\n", .{});
}
```

main

```
const std = @import("std");
const print = @import("std").debug.print;
const gv = @import("GlobalVariables.zig");
const jstk = @import("joystick.zig");
const chit = @import("Chit.zig");
const hud = @import("HUD.zig");

const c = @cImport({
    @cDefine("SDL_DISABLE_OLD_NAMES", {});
    @cInclude("SDL3/SDL.h");
    @cInclude("SDL3/SDL_revision.h");
    @cDefine("SDL_MAIN_HANDLED", {});
    @cInclude("SDL3/SDL_main.h");
    @cInclude("SDL3_image/SDL_image.h");
});

var window: ?*c.SDL_Window = undefined;
pub var renderer: ?*c.SDL_Renderer = undefined;

// *************** Surface
var mapboard_surface: ?*c.SDL_Surface = undefined;
pub var chits_surface: ?*c.SDL_Surface = undefined;
pub var frames_surface: ?*c.SDL_Surface = undefined;

// *************** Joystick
pub var joystick: ?*c.SDL_Joystick = null;

// *************** Chits
var fish: chit.Chit = undefined;

// ********************************************************************************************
pub fn main() !void {
    errdefer |err| if (err == error.SdlError) std.log.err("SDL error: {s}", .{c.SDL_GetError()});

    std.log.debug("SDL build time version: {d}.{d}.{d}", .{
        c.SDL_MAJOR_VERSION,
        c.SDL_MINOR_VERSION,
        c.SDL_MICRO_VERSION,
    });

    const version = c.SDL_GetVersion();
    std.log.debug("SDL runtime version: {d}.{d}.{d}", .{
        c.SDL_VERSIONNUM_MAJOR(version),
        c.SDL_VERSIONNUM_MINOR(version),
        c.SDL_VERSIONNUM_MICRO(version),
    });

    c.SDL_SetMainReady();

    _ = c.SDL_Init(c.SDL_INIT_VIDEO | c.SDL_INIT_AUDIO | c.SDL_INIT_GAMEPAD | c.SDL_INIT_JOYSTICK);
    defer c.SDL_Quit();

    _ = c.SDL_SetHint(c.SDL_HINT_RENDER_VSYNC, "1");

    // [ set window and renderer ========================================================== ]
    if (true) { // true: windowed
        const window_dim = c.SDL_GetCurrentDisplayMode(c.SDL_GetPrimaryDisplay());
        const window_w = @as(f32, @floatFromInt(window_dim.*.w));
        const window_h = @as(f32, @floatFromInt(window_dim.*.h));
        const percent = 0.8;
        gv.window_w = window_w * percent;
        gv.window_h = window_h * percent;
        //gv.window_w = 1280.0;
        //gv.window_h = 720.0;
        window = c.SDL_CreateWindow("Nuklear Winter '68", @intFromFloat(gv.window_w), @intFromFloat(gv.window_h), c.SDL_WINDOW_BORDERLESS);
    } else { // false: fullscreen
        const window_dim = c.SDL_GetCurrentDisplayMode(c.SDL_GetPrimaryDisplay());
        gv.window_w = @as(f32, @floatFromInt(window_dim.*.w));
        gv.window_h = @as(f32, @floatFromInt(window_dim.*.h));
        window = c.SDL_CreateWindow("Nuklear Winter '68", @intFromFloat(gv.window_w), @intFromFloat(gv.window_h), c.SDL_WINDOW_FULLSCREEN);
    }
    renderer = c.SDL_CreateRenderer(window, null);
    defer c.SDL_DestroyRenderer(renderer);
    defer c.SDL_DestroyWindow(window);

    // [ store images on surfaces ========================================================= ]
    var stream: ?*c.SDL_IOStream = undefined;

    stream = c.SDL_IOFromFile("img/Map.jpg", "r");
    //stream = c.SDL_IOFromFile("img2/fish2.jpg", "r");
    mapboard_surface = c.IMG_LoadJPG_IO(stream);

    stream = c.SDL_IOFromFile("img2/NW68-chits.png", "r");
    chits_surface = c.IMG_LoadPNG_IO(stream);

    stream = c.SDL_IOFromFile("img2/frames-1.png", "r");
    frames_surface = c.IMG_LoadPNG_IO(stream);

    // [ misc ============================================================================= ]
    fish = chit.new(12, 13, .{ 0, 1 });

    // [ game loop ======================================================================== ]
    main_loop: while (true) {
        var event: c.SDL_Event = undefined;
        while (c.SDL_PollEvent(&event)) {
            switch (event.type) {
                // [ Joystick ================================================================ ]
                c.SDL_EVENT_JOYSTICK_ADDED => {
                    if (joystick == null) {
                        joystick = c.SDL_OpenJoystick(event.jdevice.which);
                        print("open: {s}\n", .{c.SDL_GetJoystickName(joystick)});
```

```
                        try jstk.bind_buttons(c.SDL_GetJoystickName(joystick));
                        jstk.num_buttons = @as(u32, @intCast(c.SDL_GetNumJoystickButtons(joystick)));
                    }
                },
                c.SDL_EVENT_JOYSTICK_REMOVED => {
                    if ((joystick != null) and (c.SDL_GetJoystickID(joystick) == event.jdevice.which)) {
                        print("close: {s}\n", .{c.SDL_GetJoystickName(joystick)});
                        c.SDL_CloseJoystick(joystick);
                        joystick = null;
                        gv.joystick_type = 0;
                        jstk.num_buttons = 0;
                    }
                },
                // [ GUI window events ============================================================ ]
                c.SDL_EVENT_QUIT => {
                    break :main_loop;
                },
                // [ Key down ===================================================================== ]
                c.SDL_EVENT_KEY_DOWN => {
                    switch (event.key.scancode) {
                        c.SDL_SCANCODE_ESCAPE => {
                            break :main_loop;
                        },
                        else => {},
                    }
                },
                else => {},
            }
        }
        jstk.record_events();

        if ((jstk.button_bits & gv.bit_1) != 0) { // HUD mode
            draw_world();
            hud.mode();
        } else {
            draw_world();
        }

        // ***** show
        _ = c.SDL_RenderPresent(renderer);

        if (gv.flags == 1) break :main_loop;
    } // game loop
} // pub fn main()

// ************************************************************************************************
fn draw_world() void {
    // ***** clear window with color
    _ = c.SDL_SetRenderDrawColor(renderer, 255, 5, 255, c.SDL_ALPHA_OPAQUE);
    _ = c.SDL_RenderClear(renderer);

    // ***** drawing
    _ = draw_mapboard();
    _ = draw_chit1();
    _ = draw_chit2();
    _ = draw_chit3();
    fish.render();

    // ***** draw X on window
    _ = c.SDL_RenderLine(renderer, 0, 0, gv.window_w, gv.window_h);
    _ = c.SDL_RenderLine(renderer, 0, gv.window_h, gv.window_w, 0);
}

// ------------------------------------------------------------------------------------------------
fn draw_chit3() void {
    // ***** chit info
    const hex_ID = [_]i32{ 28, 0 };
    const chit_index = 7;

    // ***** create a surface
    const a_surf: *c.SDL_Surface = c.SDL_CreateSurface(gv.chit_square_dim, gv.chit_square_dim, c.SDL_PIXELFORMAT_RGBA8888);
    defer c.SDL_DestroySurface(a_surf);

    // ***** clip one chit and put it on the surface
    var a_rect: c.SDL_Rect = undefined;
    a_rect.x = 0;
    a_rect.y = chit_index * gv.chit_square_dim;
    a_rect.w = gv.chit_square_dim;
    a_rect.h = gv.chit_square_dim;
    _ = c.SDL_BlitSurface(chits_surface, &a_rect, a_surf, null); // no scaling. the target surface truncates.

    // convert the surface to a texture
    const a_texture = c.SDL_CreateTextureFromSurface(renderer, a_surf);
    defer c.SDL_DestroyTexture(a_texture);

    // define a silly puddy rectangle and render it
    var a_rectness: c.SDL_FRect = undefined;
    const hex_ID_x: i32 = @intFromFloat(@as(f64, @floatFromInt(hex_ID[0])) * gv.Hex_Dim_ness[0]);
    const hex_ID_y: i32 = @intFromFloat(@as(f64, @floatFromInt(hex_ID[1])) * gv.Hex_Dim_ness[1]);
    const x: f32 = @as(f32, @floatFromInt(gv.Zero_Zero[0] - gv.map_loc[0] + hex_ID_x)) / gv.scaleness;
    var y: f32 = 0.0;
    if (@mod(hex_ID[0], 2) == 0) {
        y = @as(f32, @floatFromInt(gv.Zero_Zero[1] - gv.map_loc[1] + hex_ID_y)) / gv.scaleness;
    } else {
        y = @as(f32, @floatFromInt(gv.Zero_Zero[1] - gv.map_loc[1] + hex_ID_y + @as(i32, @intFromFloat(gv.Half_Hex_Y_ness)))) / gv.scaleness;
    }

    const w_h_ness: f32 = @as(f32, @floatFromInt(gv.chit_square_dim)) / gv.scaleness;
    a_rectness.x = x;
    a_rectness.y = y;
    a_rectness.w = w_h_ness;
    a_rectness.h = w_h_ness;
    if ((x + w_h_ness) < 0) return;
```

```
    if (x > gv.window_w) return;
    if ((y + w_h_ness) < 0) return;
    if (y > gv.window_h) return;
    _ = c.SDL_RenderTexture(renderer, a_texture, null, &a_rectness);
}

// ------------------------------------------------------------------------------------------
fn draw_chit2() void {
    // ***** chit info
    const hex_ID = [_]i32{ 2, 0 };
    const chit_index = 4;

    // ***** create a surface
    const a_surf: *c.SDL_Surface = c.SDL_CreateSurface(gv.chit_square_dim, gv.chit_square_dim, c.SDL_PIXELFORMAT_RGBA8888);
    defer c.SDL_DestroySurface(a_surf);

    // ***** clip one chit and put it on the surface
    var a_rect: c.SDL_Rect = undefined;
    a_rect.x = 0;
    a_rect.y = chit_index * gv.chit_square_dim;
    a_rect.w = gv.chit_square_dim;
    a_rect.h = gv.chit_square_dim;
    _ = c.SDL_BlitSurface(chits_surface, &a_rect, a_surf, null); // no scaling. the target surface truncates.

    // convert the surface to a texture
    const a_texture = c.SDL_CreateTextureFromSurface(renderer, a_surf);
    defer c.SDL_DestroyTexture(a_texture);

    // define a silly puddy rectangle and render it
    var a_rectness: c.SDL_FRect = undefined;
    const hex_ID_x: i32 = @intFromFloat(@as(f64, @floatFromInt(hex_ID[0])) * gv.Hex_Dim_ness[0]);
    const hex_ID_y: i32 = @intFromFloat(@as(f64, @floatFromInt(hex_ID[1])) * gv.Hex_Dim_ness[1]);
    const x: f32 = @as(f32, @floatFromInt(gv.Zero_Zero[0] - gv.map_loc[0] + hex_ID_x)) / gv.scaleness;
    const y: f32 = @as(f32, @floatFromInt(gv.Zero_Zero[1] - gv.map_loc[1] + hex_ID_y)) / gv.scaleness;
    const w_h_ness: f32 = @as(f32, @floatFromInt(gv.chit_square_dim)) / gv.scaleness;
    a_rectness.x = x;
    a_rectness.y = y;
    a_rectness.w = w_h_ness;
    a_rectness.h = w_h_ness;
    _ = c.SDL_RenderTexture(renderer, a_texture, null, &a_rectness);
}

// ------------------------------------------------------------------------------------------
fn draw_chit1() void {
    // ***** chit info
    //const hex_loc = [_]i32{ 0, 0 };
    const chit_index = 2;

    // ***** create a surface
    const a_surf: *c.SDL_Surface = c.SDL_CreateSurface(gv.chit_square_dim, gv.chit_square_dim, c.SDL_PIXELFORMAT_RGBA8888);
    defer c.SDL_DestroySurface(a_surf);

    // ***** clip one chit and put it on the surface
    var a_rect: c.SDL_Rect = undefined;
    a_rect.x = 0;
    a_rect.y = chit_index * gv.chit_square_dim;
    a_rect.w = gv.chit_square_dim;
    a_rect.h = gv.chit_square_dim;
    _ = c.SDL_BlitSurface(chits_surface, &a_rect, a_surf, null); // no scaling. the target surface truncates.

    // convert the surface to a texture
    const a_texture = c.SDL_CreateTextureFromSurface(renderer, a_surf);
    defer c.SDL_DestroyTexture(a_texture);

    // define a silly puddy rectangle and render it
    var a_rectness: c.SDL_FRect = undefined;
    a_rectness.x = @as(f32, @floatFromInt(gv.Zero_Zero[0] - gv.map_loc[0])) / gv.scaleness;
    a_rectness.y = @as(f32, @floatFromInt(gv.Zero_Zero[1] - gv.map_loc[1])) / gv.scaleness;
    a_rectness.w = @as(f32, @floatFromInt(gv.chit_square_dim)) / gv.scaleness;
    a_rectness.h = @as(f32, @floatFromInt(gv.chit_square_dim)) / gv.scaleness;
    _ = c.SDL_RenderTexture(renderer, a_texture, null, &a_rectness);
}

// ------------------------------------------------------------------------------------------
fn draw_mapboard() void {
    var spd: i32 = 1;
    if (gv.joystick_type == 1) { // using RumblePad
        if ((jstk.button_bits & gv.bit_12) != 0) {
            spd = 200;
        }
    } else {
        spd = @intFromFloat(((@as(f32, @floatFromInt(jstk.axis_vals[1] + 32769)) / 65536.0) * 200.0) + 1.0);
    }

    // ********** D-Pad
    if (gv.joystick_type == 4) { // using Sega; converting axis to d-pad infos
        if (jstk.axis_vals[0] < -5000) jstk.d_pad = 8;
        if (jstk.axis_vals[0] > 5000) jstk.d_pad = 2;
        if (jstk.axis_vals[1] < -5000) jstk.d_pad |= 1;
        if (jstk.axis_vals[1] > 5000) jstk.d_pad |= 4;
        spd = 1;
        if ((jstk.button_bits & gv.bit_8) != 0) {
            spd = 100;
        }
    }
    if ((gv.OS_platform == 1) and (gv.joystick_type != 4)) { // FreeBSD
        if (jstk.axis_vals[2] < -10000) jstk.d_pad = 8;
        if (jstk.axis_vals[2] > 10000) jstk.d_pad = 2;
        if (jstk.axis_vals[3] < -10000) jstk.d_pad |= 1;
        if (jstk.axis_vals[3] > 10000) jstk.d_pad |= 4;
    }

    if (jstk.d_pad != 0) { // no inputs, don't bother going in; this should save time
```

```
        if ((jstk.d_pad & gv.bit_0) != 0) {
            gv.map_loc[1] += spd;
        }
        if ((jstk.d_pad & gv.bit_1) != 0) {
            gv.map_loc[0] -= spd;
        }
        if ((jstk.d_pad & gv.bit_2) != 0) {
            gv.map_loc[1] -= spd;
        }
        if ((jstk.d_pad & gv.bit_3) != 0) {
            gv.map_loc[0] += spd;
        }
    }

    // ********** Left & Right sholder bind_buttons. If continued pressing, don't change scale.
    gv.scale_old = gv.scale;
    gv.scaleness_old = gv.scaleness;
    if ((jstk.button_bits & gv.bit_4) != 0) {
        if ((jstk.button_bits_old & gv.bit_4) == 0) {
            gv.scale -= 1;
        }
    }
    if ((jstk.button_bits & gv.bit_5) != 0) {
        if ((jstk.button_bits_old & gv.bit_5) == 0) {
            gv.scale += 1;
        }
    }
    if (gv.scale_old != gv.scale) {
        if (gv.scale > 0) { // zoom in
            gv.scaleness = 1.0 / ((1.0 + @as(f32, @floatFromInt(gv.scale))) * 0.6);
        } else if (gv.scale < 0) { // zoom out
            gv.scaleness = 1.0 - (@as(f32, @floatFromInt(gv.scale)) * 0.5);
        } else {
            gv.scaleness = 1.0;
        }
    }

    // ********** clip map surface and save it on a_surf; convert a_surf to texture; render the texture
    const clip_w: f32 = gv.window_w * gv.scaleness;
    const clip_h: f32 = gv.window_h * gv.scaleness;
    const a_surf: *c.SDL_Surface = c.SDL_CreateSurface(@intFromFloat(clip_w), @intFromFloat(clip_h), c.SDL_PIXELFORMAT_RGBA8888);
    defer c.SDL_DestroySurface(a_surf);

    var a_rect: c.SDL_Rect = undefined; // clip square
    if (gv.scale_old != gv.scale) { // shift clip square
        const delta_x_half: f32 = ((gv.window_w * gv.scaleness) - (gv.window_w * gv.scaleness_old)) / 2.0;
        const delta_y_half: f32 = ((gv.window_h * gv.scaleness) - (gv.window_h * gv.scaleness_old)) / 2.0;
        gv.map_loc[0] -= @intFromFloat(delta_x_half);
        gv.map_loc[1] -= @intFromFloat(delta_y_half);
    }
    a_rect.x = gv.map_loc[0];
    a_rect.y = gv.map_loc[1];
    a_rect.w = @intFromFloat(clip_w);
    a_rect.h = @intFromFloat(clip_h);
    _ = c.SDL_BlitSurface(mapboard_surface, &a_rect, a_surf, null); // no scaling. the target surface truncates.

    const a_texture = c.SDL_CreateTextureFromSurface(renderer, a_surf);
    defer c.SDL_DestroyTexture(a_texture);

    _ = c.SDL_RenderTexture(renderer, a_texture, null, null);
}
```

HUD

```
const std = @import("std");
const gv = @import("GlobalVariables.zig");
const print = @import("std").debug.print;
const m = @import("main.zig");
const jstk = @import("joystick.zig");
const hud_0 = @import("hud_new.zig");

const c = @cImport({
    @cDefine("SDL_DISABLE_OLD_NAMES", {});
    @cInclude("SDL3/SDL.h");
    @cInclude("SDL3/SDL_revision.h");
    @cDefine("SDL_MAIN_HANDLED", {});
    @cInclude("SDL3/SDL_main.h");
    @cInclude("SDL3_image/SDL_image.h");
});

const frame_dim = [_]i32{ 500, 660, 0, 0 }; // frame & corner
const mesa_dim = [_]i32{ 480, 640, 10, 10 }; // mesa & shifts
var menu_option: i32 = 0;
var menu_option_old: i32 = -1;
var d_pad_old: u16 = 0;

pub fn mode() void {
    const clipped = c.SDL_Rect{ .x = frame_dim[2], .y = frame_dim[3], .w = frame_dim[0], .h = frame_dim[1] }; // clipped

    // ***** create a surface
    const a_surf: *c.SDL_Surface = c.SDL_CreateSurface(frame_dim[0], frame_dim[1], c.SDL_PIXELFORMAT_RGBA8888);
    defer c.SDL_DestroySurface(a_surf);

    // ***** clip the frame and put it on the surface
    _ = c.SDL_BlitSurface(@ptrCast(m.frames_surface), &clipped, a_surf, null);

    // convert the surface to a texture
    const a_texture = c.SDL_CreateTextureFromSurface(@ptrCast(m.renderer), a_surf);
    defer c.SDL_DestroyTexture(a_texture);

    // ***** "paste" the texture on the window; not using viewport
    const silly_putty = c.SDL_FRect{ .x = gv.window_w - frame_dim[0], .y = 0.0, .w = frame_dim[0], .h = frame_dim[1] };
    _ = c.SDL_RenderTexture(@ptrCast(m.renderer), a_texture, null, &silly_putty); // put texture FOR any viewports

    // ***** show
    _ = c.SDL_RenderPresent(@ptrCast(m.renderer));
    main_menu(silly_putty);
}

// ********************************************************************************************
fn main_menu(silly: c.SDL_FRect) void {
    _ = silly;
    main_loop: while (true) {
        var event: c.SDL_Event = undefined;
        while (c.SDL_PollEvent(&event)) {
            switch (event.type) {
                // [ Key down ========================================================================= ]
                c.SDL_EVENT_KEY_DOWN => {
                    switch (event.key.scancode) {
                        c.SDL_SCANCODE_ESCAPE => {
                            break :main_loop;
                        },
                        else => {},
                    }
                },
                else => {},
            }
        }

        jstk.record_events();
        if ((jstk.button_bits & gv.bit_3) != 0) break :main_loop;
    }
    // // ***** menu number
    // const num_choices: i32 = 4;

    // if (jstk.d_pad == 1) {
    //     if (jstk.d_pad != d_pad_old) {
    //         menu_option -= 1;
    //         d_pad_old = 1;
    //     }
    // }
    // if (jstk.d_pad == 4) {
    //     if (jstk.d_pad != d_pad_old) {
    //         menu_option += 1;
    //         d_pad_old = 4;
    //     }
    // }
    // if (menu_option < 0) menu_option += num_choices;
    // if (menu_option >= num_choices) menu_option = 0;
    // if (jstk.d_pad == 0) d_pad_old = 0;

    // var X: i32 = 0;
    // var Xness: f32 = 0.0;
    // var Y: i32 = 0;
    // var Yness: f32 = 0.0;
    // var W: i32 = 0;
    // var Wness: f32 = 0.0;
    // var H: i32 = 0;
    // var Hness: f32 = 0.0;
    // const scale: f32 = 2.0;

    // // ***************************************
    // X = @intFromFloat(gv.window_w - frame_dim[0]);
    // X += 11;
    // const mesa_viewport = c.SDL_Rect{ .x = X, .y = 10, .w = mesa_dim[0], .h = mesa_dim[1] };
```

```
    // _ = c.SDL_SetRenderViewport(@ptrCast(m.renderer), &mesa_viewport);
    // _ = c.SDL_RenderDebugText(@ptrCast(m.renderer), 0, 0, "-123456789-123456789-123456789-123456789-123456789-123456789");
    // _ = c.SDL_RenderDebugText(@ptrCast(m.renderer), 1, 0, "   345");

    // X = @intFromFloat(silly.x);
    // X += mesa_dim[2]; // shift left adjustment

    // // ***** 2x scale
    // _ = c.SDL_SetRenderScale(@ptrCast(m.renderer), scale, scale);
    // Xness = @floatFromInt(X);
    // Xness /= scale;
    // X = @intFromFloat(Xness);
    // Yness = @floatFromInt(mesa_dim[2]);
    // Yness /= scale;
    // Y = @intFromFloat(Yness);
    // Wness = @floatFromInt(mesa_dim[0]);
    // Wness /= scale;
    // W = @intFromFloat(Wness);
    // Hness = @floatFromInt(mesa_dim[1]);
    // Hness /= scale;
    // H = @intFromFloat(Hness);
    // const scaled_viewport = c.SDL_Rect{ .x = X, .y = Y, .w = W, .h = H };
    // _ = c.SDL_SetRenderViewport(@ptrCast(m.renderer), &scaled_viewport);
    // _ = c.SDL_RenderDebugText(@ptrCast(m.renderer), 0, (8 * 13), "         New"); // 0
    // _ = c.SDL_RenderDebugText(@ptrCast(m.renderer), 0, (8 * 14), "         Load"); // 1
    // _ = c.SDL_RenderDebugText(@ptrCast(m.renderer), 0, (8 * 15), "         Save"); // 2
    // _ = c.SDL_RenderDebugText(@ptrCast(m.renderer), 0, (8 * 16), "         Quit"); // 3
    // Yness = @floatFromInt(menu_option);
    // Yness += 13;
    // Yness *= 8;
    // _ = c.SDL_RenderDebugText(@ptrCast(m.renderer), 0, Yness, "       *");

    // // ***** back to normal
    // _ = c.SDL_SetRenderScale(@ptrCast(m.renderer), 1.0, 1.0);

    // _ = c.SDL_SetRenderViewport(@ptrCast(m.renderer), null); // null; remove viewport

    // // ***** set flags
    // if ((jstk.button_bits & gv.bit_0) != 0) { // down button pressed
    //     if (menu_option == 3) { // 3: quit option
    //         gv.flags = 1;
    //     }
    //     if ((menu_option == 0) and (menu_option_old != 0)) { // 0: new game option
    //         menu_option = 0;
    //         hud_0.newGame();
    //         menu_option_old = 0;
    //     }
    //     if ((menu_option == 1) and (menu_option_old != 1)) { // 1: load game option
    //         menu_option = 1;
    //         print("one\n", .{});
    //         menu_option_old = 1;
    //     }
    //     if ((menu_option == 2) and (menu_option_old != 2)) { // 2: save game option
    //         menu_option = 2;
    //         print("two\n", .{});
    //         menu_option_old = 2;
    //     }
    // } else {
    //     menu_option_old = -1;
    // }
}
```