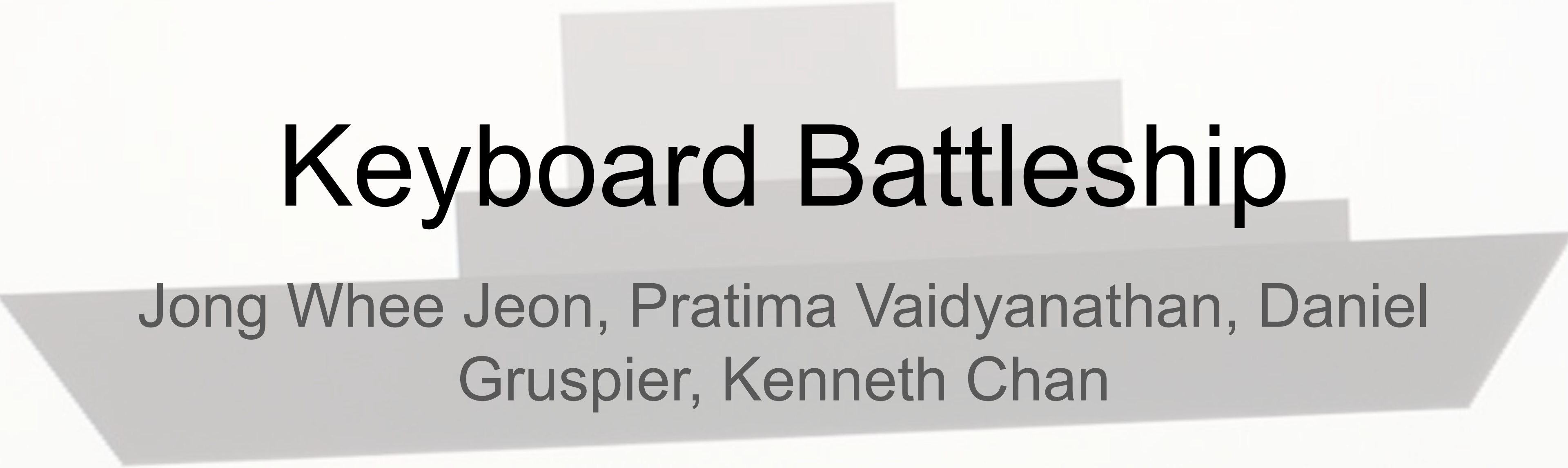
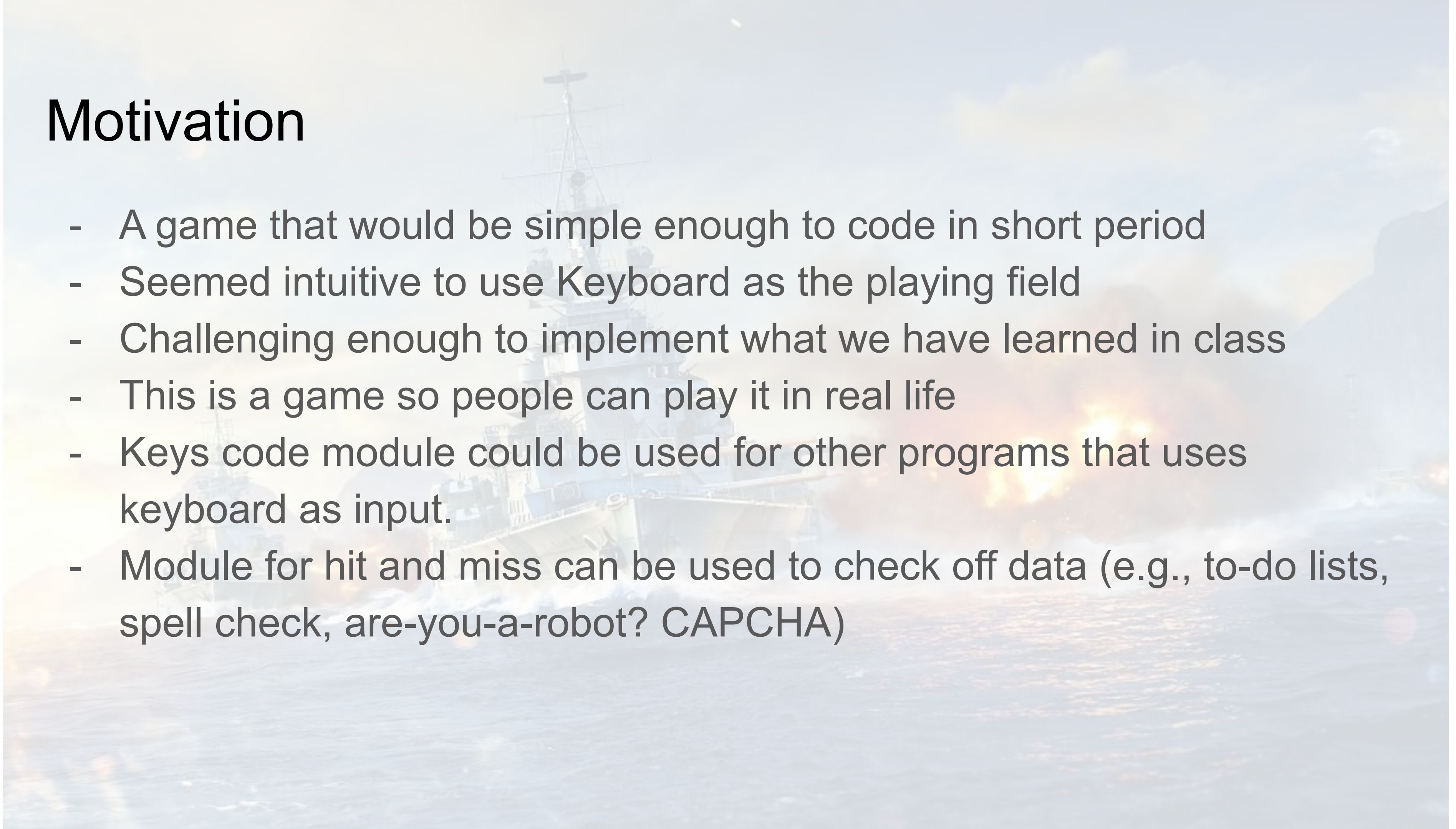


Keyboard Battleship

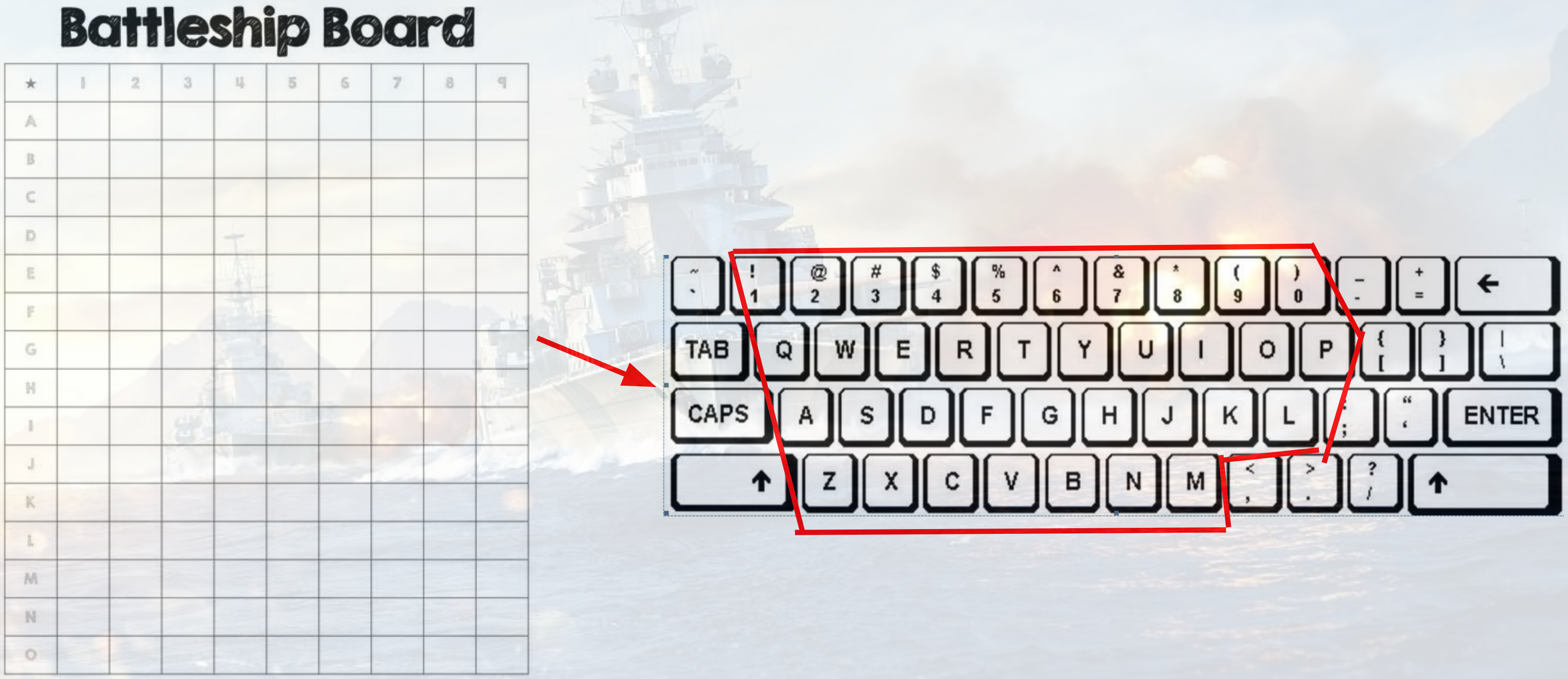
Jong Whee Jeon, Pratima Vaidyanathan, Daniel Gruspier, Kenneth Chan

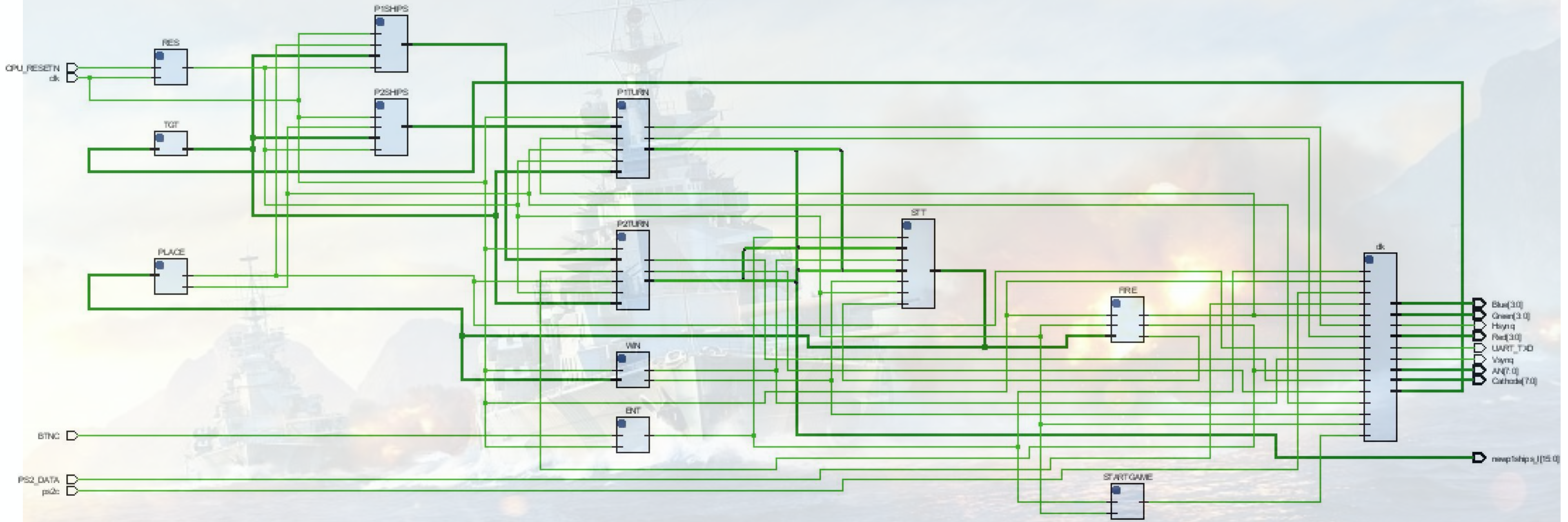


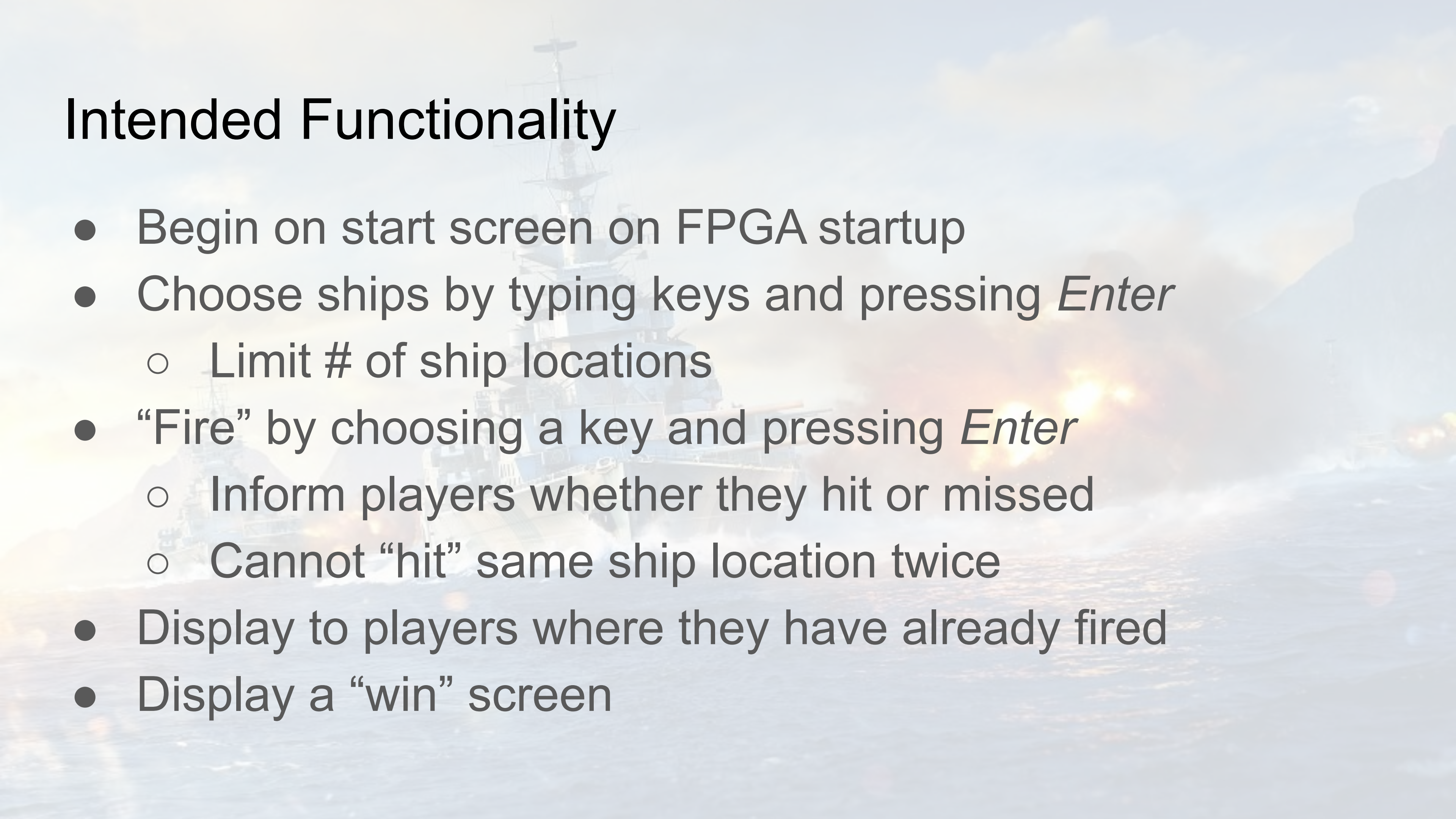
Motivation

- A game that would be simple enough to code in short period
- Seemed intuitive to use Keyboard as the playing field
- Challenging enough to implement what we have learned in class
- This is a game so people can play it in real life
- Keys code module could be used for other programs that uses keyboard as input.
- Module for hit and miss can be used to check off data (e.g., to-do lists, spell check, are-you-a-robot? CAPCHA)

Intended Functionality




CPU_RESETN
clk
RES
TGT
PLACE
P1SHIPS
P2SHIPS
P1TURN
P2TURN
WIN
ENT
STT
FIRE
START GAME
clk
BTNC
PS2_DATA
ps2c
Blue[3:0]
Green[3:0]
Hsync
Red[3:0]
UART_TXD
Vsync
AN[7:0]
Cathode[7:0]
newp1ships_l[15:0]

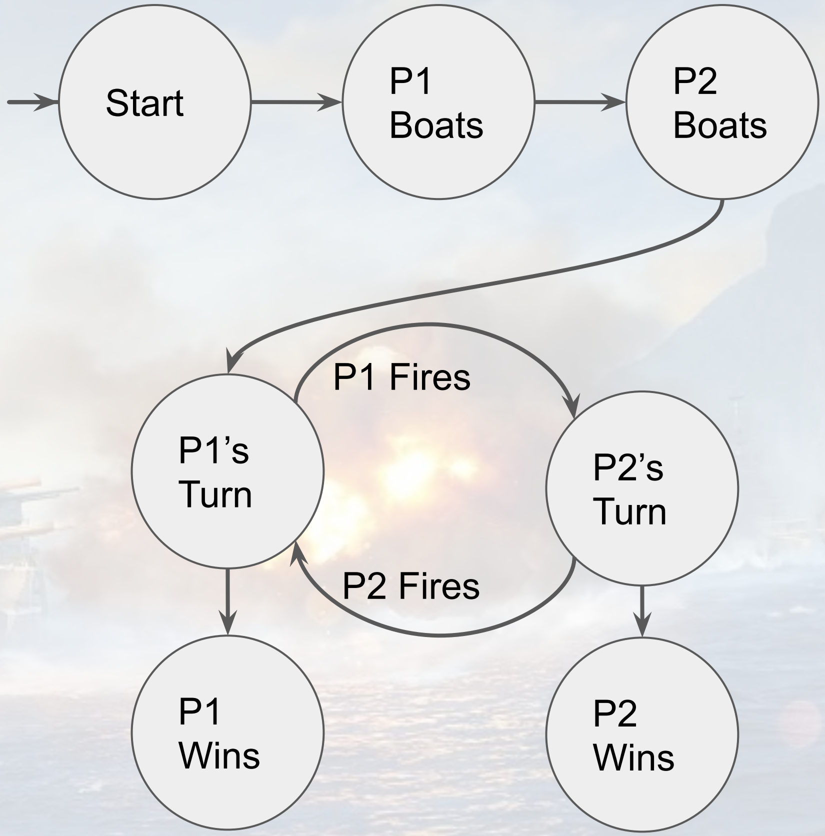


Intended Functionality

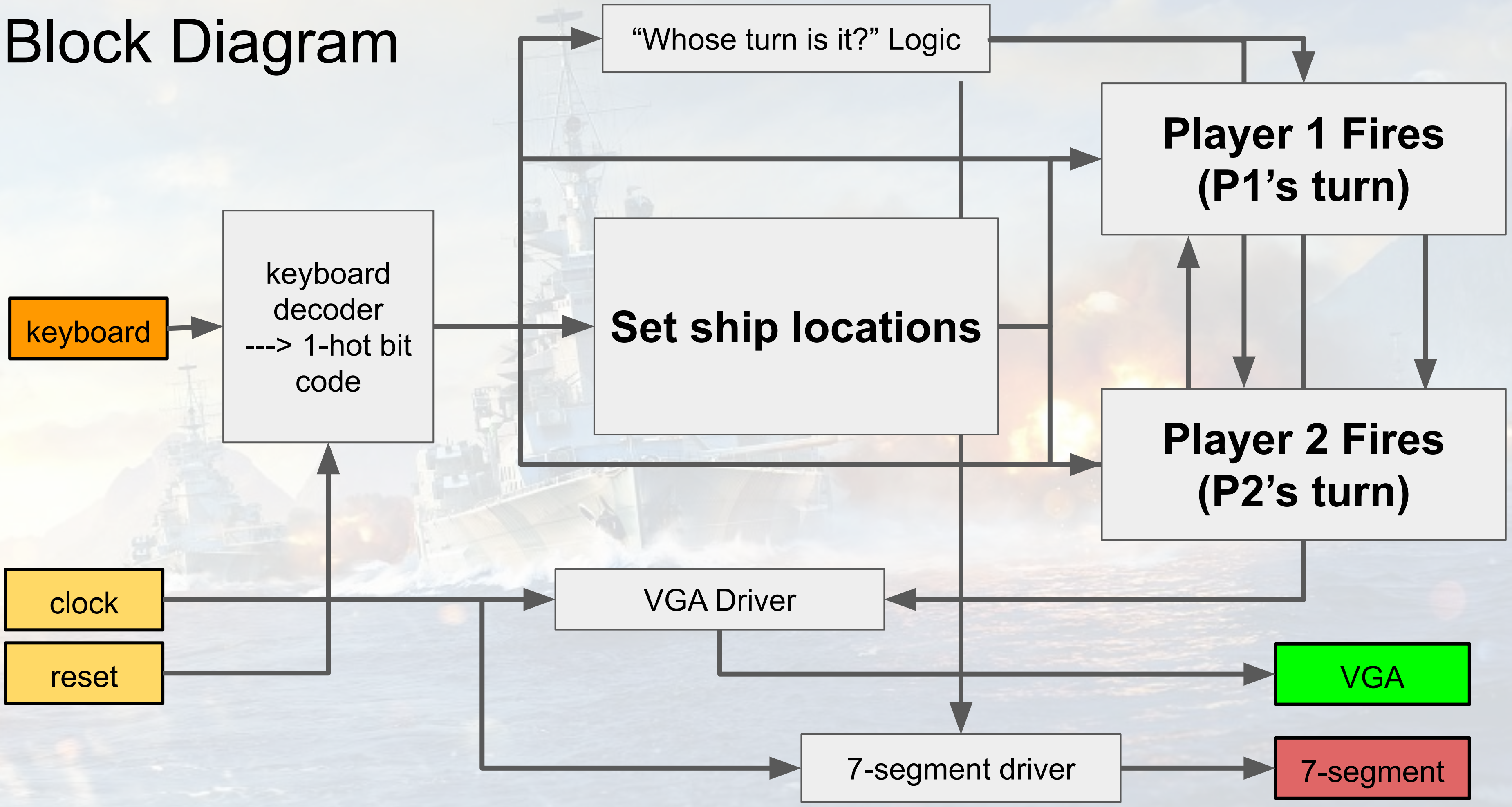
- Begin on start screen on FPGA startup
- Choose ships by typing keys and pressing *Enter*
  - Limit # of ship locations
- “Fire” by choosing a key and pressing *Enter*
  - Inform players whether they hit or missed
  - Cannot “hit” same ship location twice
- Display to players where they have already fired
- Display a “win” screen

Specifications

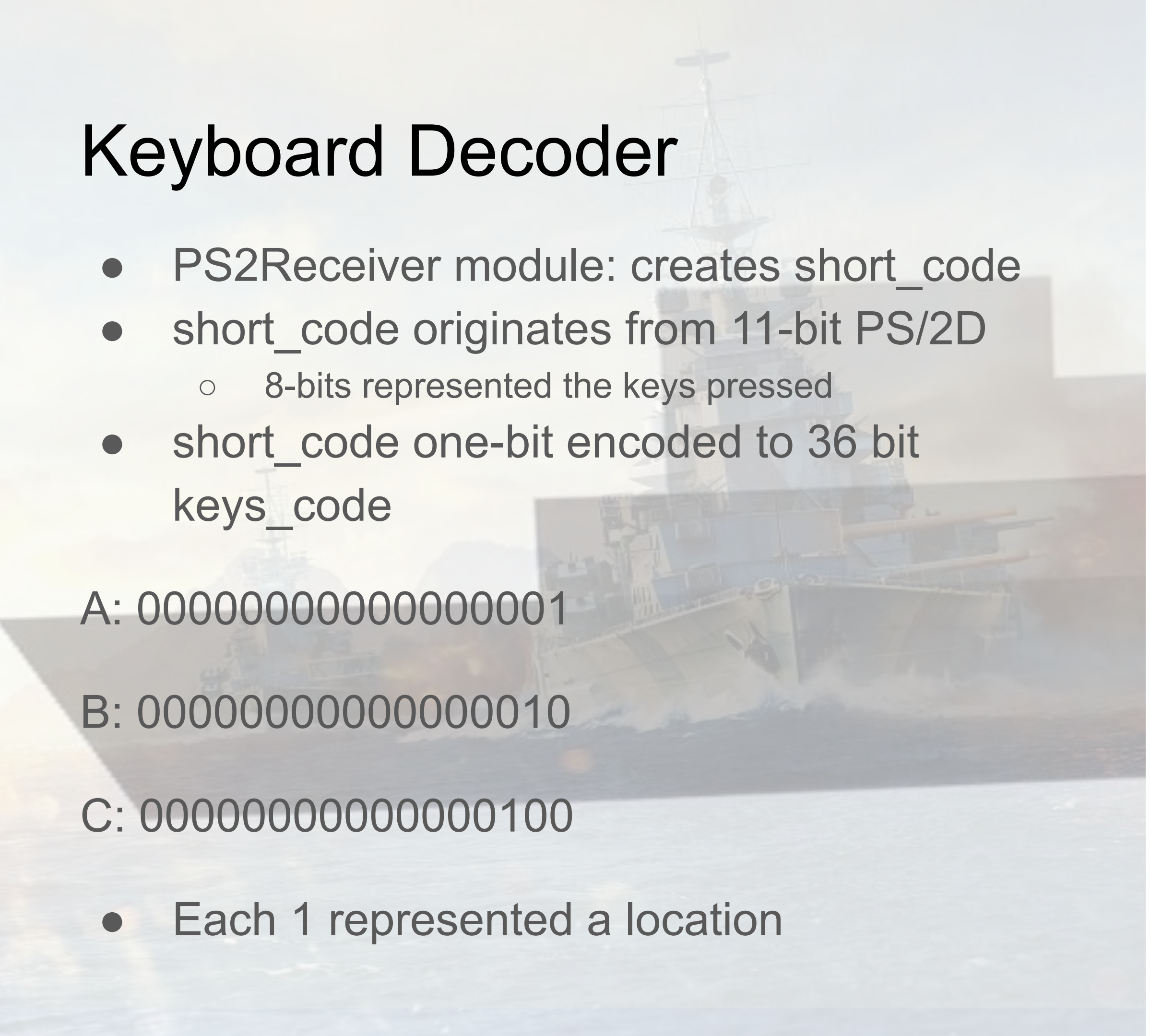
- Bitmap to represent locations
- Let players set their own ships
- Taking turns between 2 players
    - Displaying whose turn it is
- Showing hits and misses
- Displaying the winner



Block Diagram



"Whose turn is it?" Logic
Player 1 Fires (P1's turn)
keyboard
keyboard decoder ---> 1-hot bit code
Set ship locations
Player 2 Fires (P2's turn)
clock
VGA Driver
reset
VGA
7-segment driver
7-segment



Keyboard Decoder

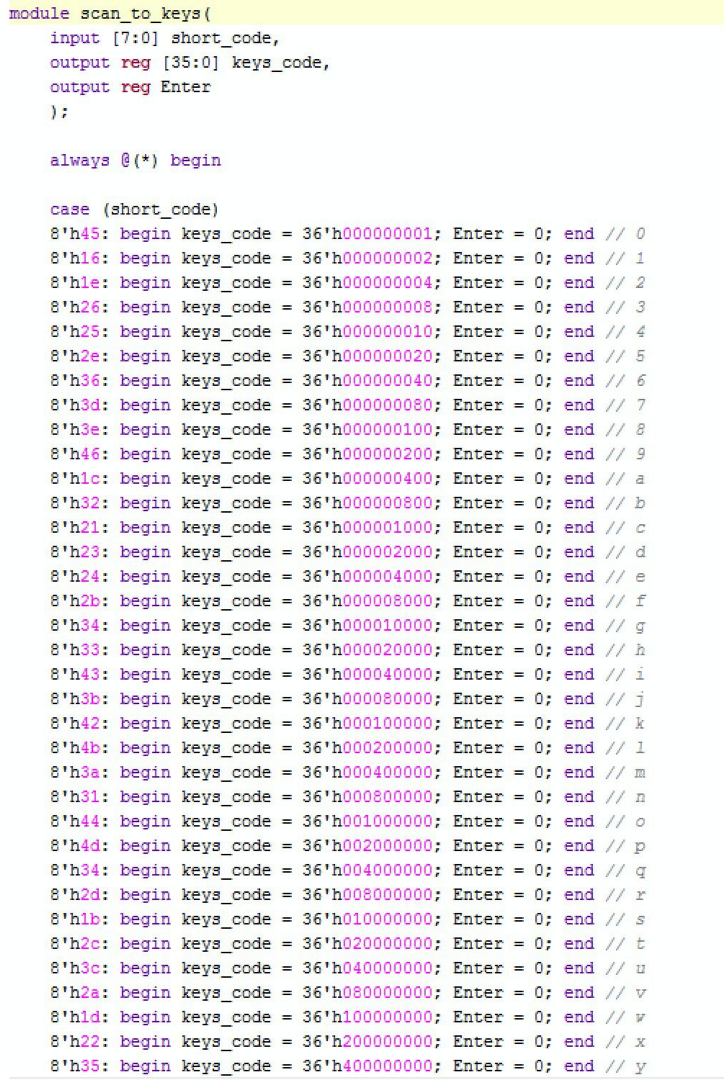
- PS2Receiver module: creates short_code
- short_code originates from 11-bit PS/2D
  - 8-bits represented the keys pressed
- short_code one-bit encoded to 36 bit keys_code

A: 000000000000000001

B: 000000000000000010

C: 000000000000000100

- Each 1 represented a location



```
module scan_to_keys(
    input [7:0] short_code,
    output reg [35:0] keys_code,
    output reg Enter
    );

    always @(*) begin

    case (short_code)
    8'h45: begin keys_code = 36'h000000001; Enter = 0; end // 0
    8'h16: begin keys_code = 36'h000000002; Enter = 0; end // 1
    8'h1e: begin keys_code = 36'h000000004; Enter = 0; end // 2
    8'h26: begin keys_code = 36'h000000008; Enter = 0; end // 3
    8'h25: begin keys_code = 36'h000000010; Enter = 0; end // 4
    8'h2e: begin keys_code = 36'h000000020; Enter = 0; end // 5
    8'h36: begin keys_code = 36'h000000040; Enter = 0; end // 6
    8'h3d: begin keys_code = 36'h000000080; Enter = 0; end // 7
    8'h3e: begin keys_code = 36'h000000100; Enter = 0; end // 8
    8'h46: begin keys_code = 36'h000000200; Enter = 0; end // 9
    8'h1c: begin keys_code = 36'h000000400; Enter = 0; end // a
    8'h32: begin keys_code = 36'h000000800; Enter = 0; end // b
    8'h21: begin keys_code = 36'h000001000; Enter = 0; end // c
    8'h23: begin keys_code = 36'h000002000; Enter = 0; end // d
    8'h24: begin keys_code = 36'h000004000; Enter = 0; end // e
    8'h2b: begin keys_code = 36'h000008000; Enter = 0; end // f
    8'h34: begin keys_code = 36'h000010000; Enter = 0; end // g
    8'h33: begin keys_code = 36'h000020000; Enter = 0; end // h
    8'h43: begin keys_code = 36'h000040000; Enter = 0; end // i
    8'h3b: begin keys_code = 36'h000080000; Enter = 0; end // j
    8'h42: begin keys_code = 36'h000100000; Enter = 0; end // k
    8'h4b: begin keys_code = 36'h000200000; Enter = 0; end // l
    8'h3a: begin keys_code = 36'h000400000; Enter = 0; end // m
    8'h31: begin keys_code = 36'h000800000; Enter = 0; end // n
    8'h44: begin keys_code = 36'h001000000; Enter = 0; end // o
    8'h4d: begin keys_code = 36'h002000000; Enter = 0; end // p
    8'h34: begin keys_code = 36'h004000000; Enter = 0; end // q
    8'h2d: begin keys_code = 36'h008000000; Enter = 0; end // r
    8'h1b: begin keys_code = 36'h010000000; Enter = 0; end // s
    8'h2c: begin keys_code = 36'h020000000; Enter = 0; end // t
    8'h3c: begin keys_code = 36'h040000000; Enter = 0; end // u
    8'h2a: begin keys_code = 36'h080000000; Enter = 0; end // v
    8'h1d: begin keys_code = 36'h100000000; Enter = 0; end // w
    8'h22: begin keys_code = 36'h200000000; Enter = 0; end // x
    8'h35: begin keys_code = 36'h400000000; Enter = 0; end // y
```

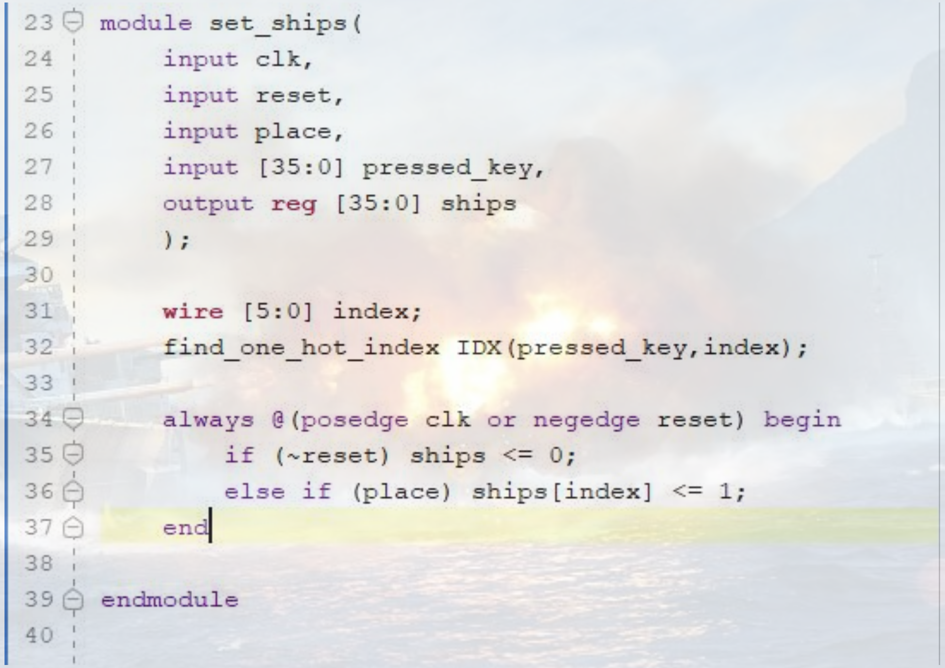
Keys_code:

Changes keys into one-hot

A => 000001

B=> 000010

AB => 000011

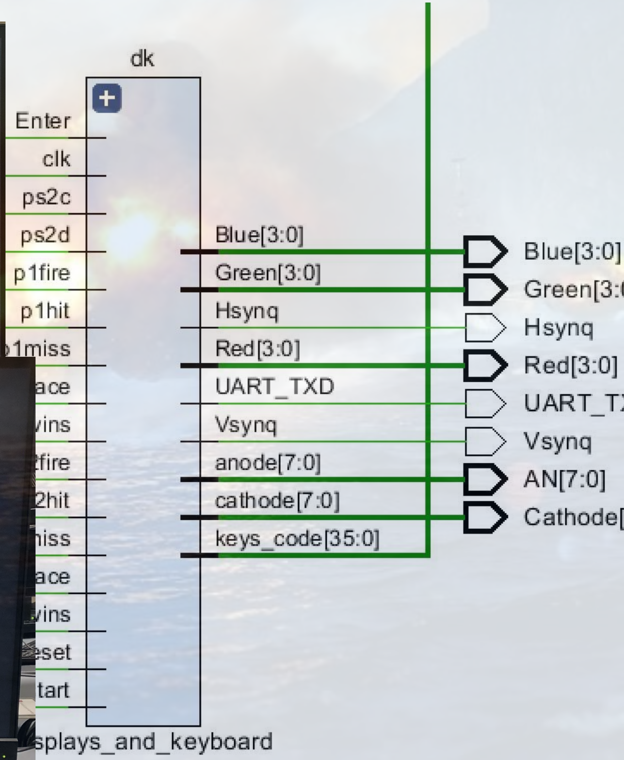


```
module set_ships(
    input clk,
    input reset,
    input place,
    input [35:0] pressed_key,
    output reg [35:0] ships
    );

    wire [5:0] index;
    find_one_hot_index IDX(pressed_key,index);

    always @(posedge clk or negedge reset) begin
        if (~reset) ships <= 0;
        else if (place) ships[index] <= 1;
    end

endmodule
```

Display module inputs/outputs

- Several inputs determine the seven segment display/VGA outputs

M
P1!
P2!
H
PRESS 2 FOR FPGA ON FPGA SWITCH
PRESS 2 FOR FPGA ON FPGA SWITCH



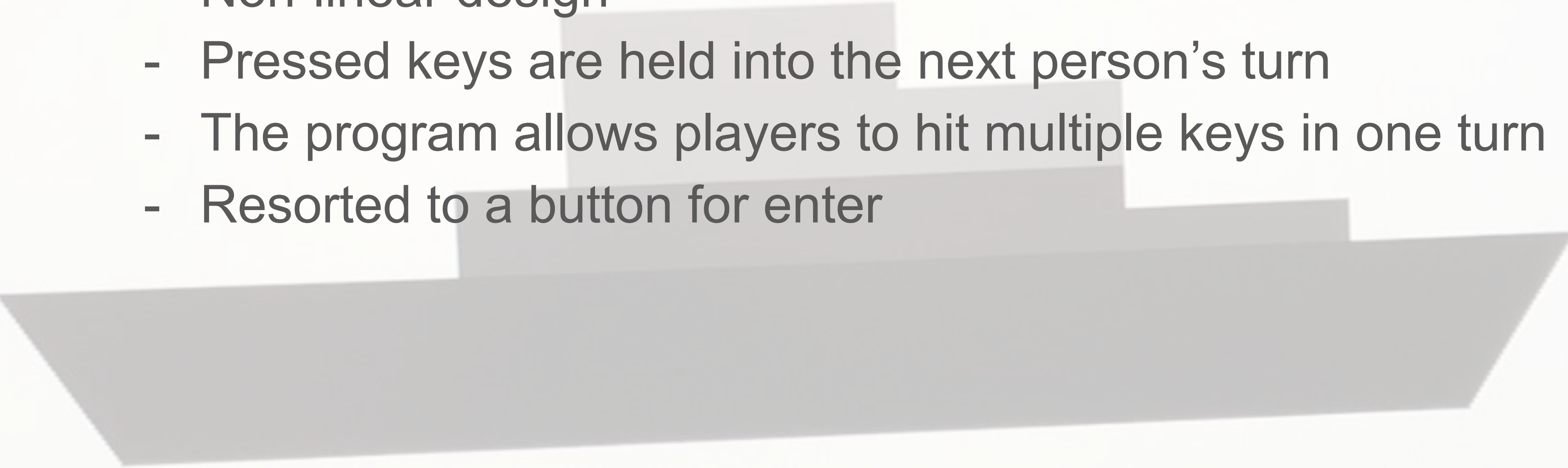
Code Snippet: `hit_or_miss.v`

```verilog
always @(posedge clk /* negedge reset*/) begin
    if (~reset) begin
        hit <= 0;
        miss <= 0;
        new_enemy_ships <= 36'b111111111111111111111111111111111111;
    end
    if (place)
        new_enemy_ships <= enemy_ships;
    else if (fire) begin
        for (i = 0; i < 36; i = i + 1) begin
            if (target[i] && enemy_ships[i]) begin
                new_enemy_ships[i] <= 0;
                hit_tracker[i] <= 1;
            end else begin
                hit_tracker[i] <= 0;
                //new_enemy_ships[i]
            end
        end
        if (hit_tracker > 0) hit <= 1
        if (hit == 0) miss <= 1; else
    end
end
```

When this register is all 0s, that means every ship location has been hit, and therefore the game is over.

Failures

- Non-linear design
- Pressed keys are held into the next person’s turn
- The program allows players to hit multiple keys in one turn
- Resorted to a button for enter



Successes

- Made a cool start screen logo
- Game takes in user input to set ships for each player via the keyboard
- VGA Hit or Miss screen correctly displays at the end of each player's turn
- Seven segment display shows whose turn it is/if players are setting ships
- When a player has no ships left, the VGA correctly displays the winner
- Goes through all the states
- Correct ships are knocked out
- Places ships for the correct keys
- Game works but there are some
  unexpected quirks