Gomoku Project Report

1. Overview

1.1 Introduction Gomoku

Gomoku, also called *Five in a Row*, is an abstract strategy board game. It's traditionally played with Go pieces (black and white stones) on a 15×15 board, as shown in Figure 1. Two players take turns to place a stone of their color on an empty intersection. The winner is the first player to form an unbroken line of five stones of their color horizontally, vertically, or diagonally.

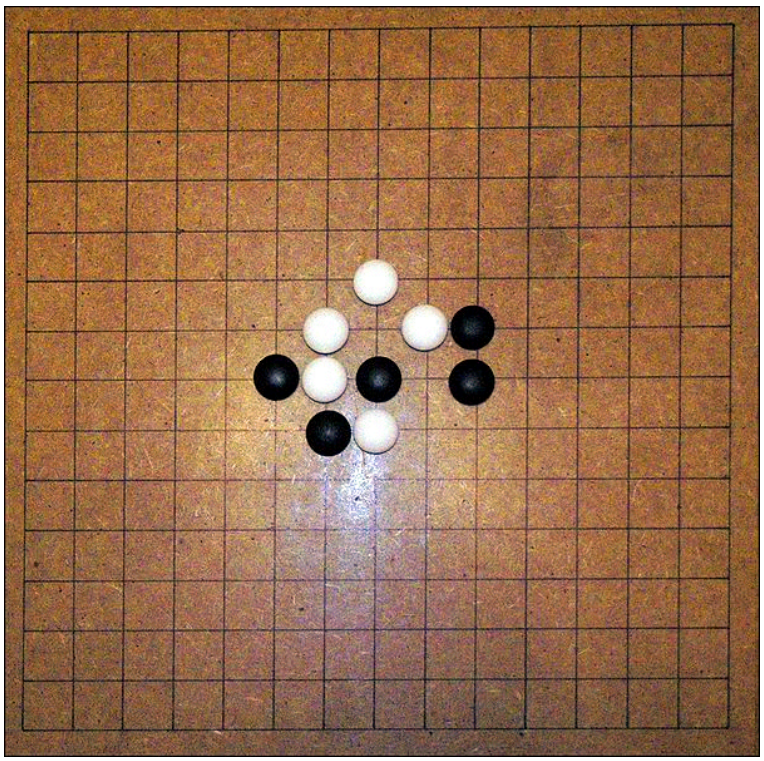


Figure 1

1.2 Project overview

In this project, we developed a FPGA-based Gomoku Application which incorporates essential Gomoku logic, AI algorithms, and network functionality, which offers players a variety of modes to choose from. Additionally, we implemented the device driver for both Xbox and Touchpad, so the game supports these two kinds of controllers, as shown in Figure 2. All of these improve the overall user experience.

Figure 2

The game offers three different modes at the start: Player vs. Player (PvP) Mode, Network Mode, and Player vs. Environment (PvE) Mode, as shown in Figure 2. In PvP Mode, two players play against each other on the same device, taking turns to make their moves. In Network Mode, players can connect and play over a network by either creating a new room or joining an existing one, allowing them to compete remotely. PvE Mode pits a single player against the computer, and the color of the player will be randomly assigned by the game. Once a mode is selected and the necessary preparations are made, the game begins.

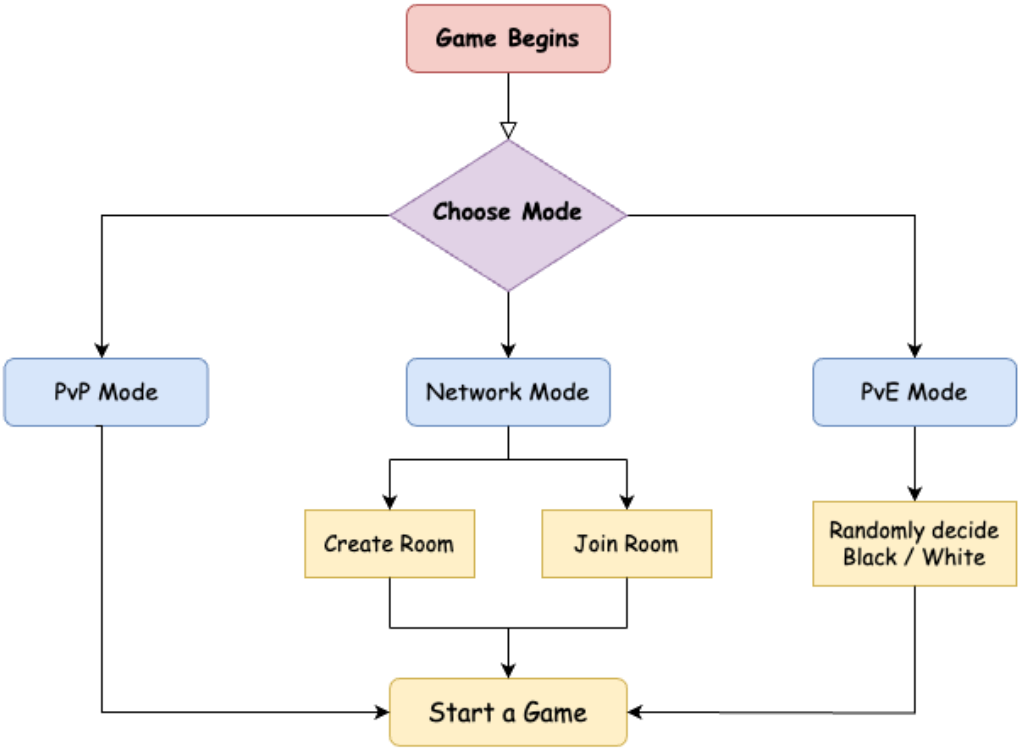


Figure 3

During one game, players have the option to regret a move, ask for AI's hint based on the current board situation, and resign the game. Additionally, players have nice-looking profiles on the top right corner, together with a colored stone indicating whether they are playing as Black or White, as shown in Figure 5.

2. Hardware

2.1 Graphics

2.1.1 UI Design

Our display components are mainly distributed across the menu interface, in-game interface, and pop-up message interface.

Menu interface

The menu interface includes the "GOMOKU" game title, as well as five options, which are shown in Figure 4.

- "START PVP"
    - allows players to engage in player-versus-player mode on the same computer
- "START PVE"
    - the player is randomly assigned black or white stones to play against an AI opponent
- "CREATE ROOM"
    - allows the player to create a local area network room to play against other players on different computers within the same local network
- "JOIN ROOM"
    - after a local area network room is created, this option allows the player to join that room. Upon joining, a "SCANNING..." pop-up message appears; if no room is found, the pop-up closes automatically after a few seconds, returning to the menu interface
- "EXIT"
    - exit the game

Figure 4

In-game interface

The in-game interface consists of a 15x15 board (with pixel dimensions of 500x480) and an information bar and options bar on the right, as shown in Figure 5. In addition to displaying black and white pieces, the board area also shows a selected mark and a last piece mark. The former is used to determine the next move's position, while the latter identifies the opponent's previous move.

The upper half of the right side is the player information bar, which includes the following information:

- player's avatar
- the color of their pieces
- identity (P1, P2, or AI)
- a current turn indicator (showing whose turn it is to make a move).

The lower half on the right side contains the in-game options, consisting of four functions:

- "REGRET"
    - undo move, clicking this will undo the last moves made by both players, reverting to the previous board state
- "HINT"
    - AI assistance, when unsure of the next move, this can be used to request a hint, and the AI will display a golden piece on the recommended position
- "RESIGN"
    - resign game, if not wanting to continue the match, this can be used to immediately end the game, displaying a message indicating your defeat/opponent's victory
- "EXIT"
    - exit the current match and return to the main menu

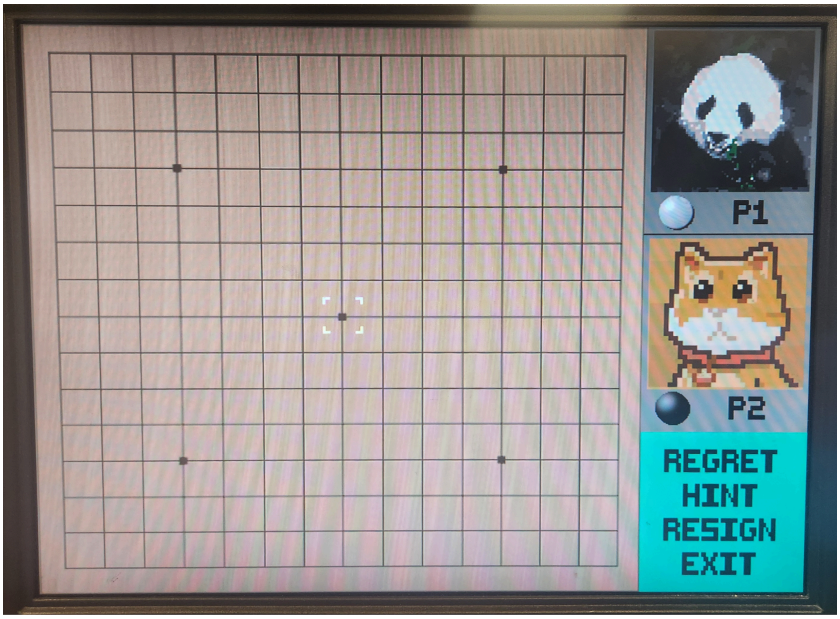


Figure 5

pop-up message interface

The last part is the pop-up message interface. On the menu interface, selecting to join a local area network room will trigger a "SCANNING..." pop-up window.

- In the game interface, when one side successfully achieves a line of five pieces, those five pieces forming the line will sequentially turn golden in color, after which

the corresponding result window will pop up. all of these pop-ups are centered with an "EXIT" option at the bottom.

- if playing against an AI (PVE) and winning, a "YOU WIN!" the victory window will pop up;
- if losing, a "YOU LOSE" window will appear.
- If playing against another player (PVP), a "P1 WIN!" the window will pop up if P1 wins, and a "P2 WIN!" window if P2 wins.
- when choosing to exit the game, an "ARE YOU SURE?" a confirmation window will appear with "YES" and "NO" options near the bottom.
  - selecting "YES" will return to the menu interface
  - selecting "NO" will close the confirmation window.

2.1.2 Image Display Logic

To reduce the memory cost of images, we use the color palette method, which uses one palette value to represent one RGB value. By doing so, we can reduce the memory usage of one pixel from 24 bits to the bit length of a palette value. For example, we use a 16-color, 4-bit palette for our board, reducing the memory usage to ⅙. And we use the 256-color, 8-bit palette for the rest of our images, decreasing the image size to ⅓. Figure 6 shows how we generate the image MIFs and display the image.

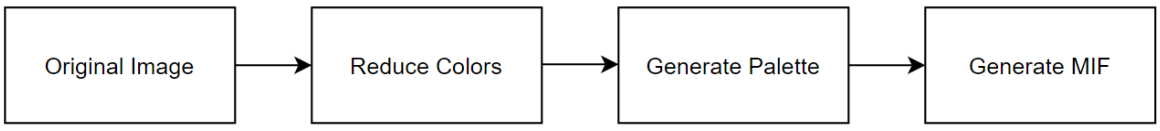


Image MIF generation procedure

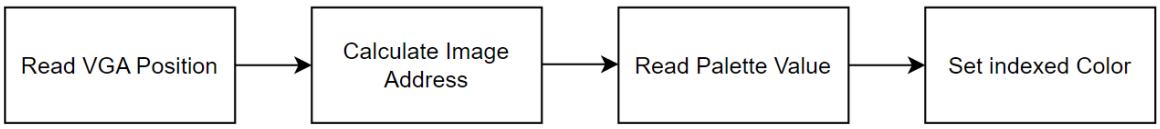


Image display procedure

Figure 6

It is worth mentioning that we reuse the white piece image for the highlighted piece. To achieve this, we set the original white piece Blue value to 128 so that the piece looks to be highlighted but without adding an extra piece image.

2.1.3 Text Display Logic

To display various fixed-position texts of different font sizes, we designed logic that reuses the character ROM data and uses very little memory.
For each character, we store 8x5 bits of data in the ROM. Each bit represents a display segment that shows the same color according to the bit value. We have 8 segments in a row and 5 segments in a column, which then construct a display segment matrix for a character. Therefore, as long as the width and height of the display segment are correctly set, we can display characters of various sizes with a memory cost of only 5 bytes for each character. Figure 7 shows how we translate the data to show character “A”.

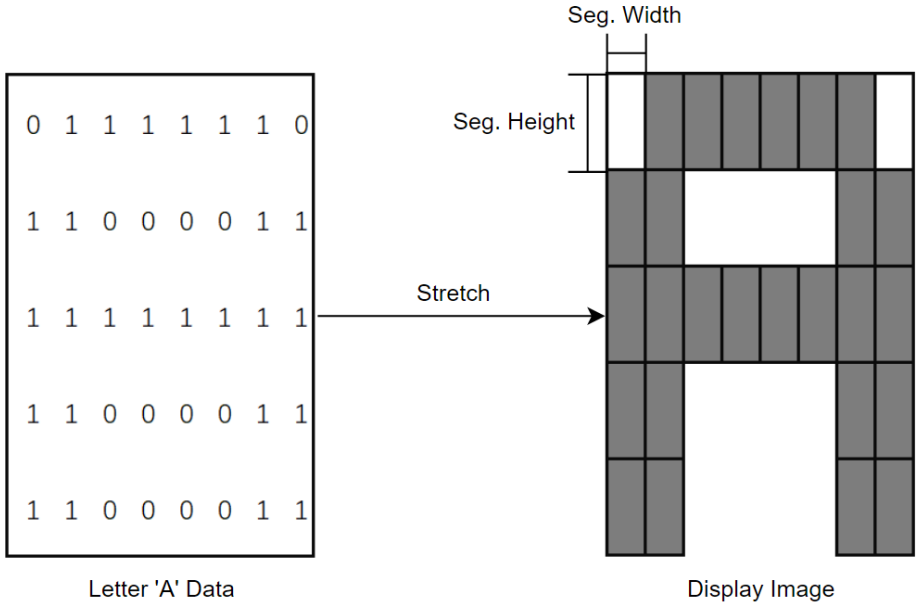


Figure 7

To display text, we reuse the character data in the ROM and calculate the ROM address using segment width, segment height, and the character. In addition, if we set the width and height to the power of 2, we can calculate the font ROM address more easily by using the bit shift operation.

We also wrote a Python script to help generate the SystemVerilog code. Here is a snippet of the generated code for displaying “EXIT.”

```
Unset
/**
* message: EXIT
* group 1: menu
* group_index: 5
* select_index: 6
* h_start: 288
* v_start: 370
* font_width: 16
* font_height: 20
*/
if((hcount[10:1] >= 10'd288) && (hcount[10:1] < 10'd352) && (vcount >= 10'd370)
&& (vcount < 10'd390) && msg_visible_menu[5]) begin
       msg_display_menu[5] <= 1;
       cur_msg_selected <= (msg_selected==6);
       case((hcount[10:1]-288)>>4)
              8'd0: font_addr <= 8'd20+((vcount[9:0]-370)>>2);     // 'E'
              8'd1: font_addr <= 8'd115+((vcount[9:0]-370)>>2);    // 'X'
              8'd2: font_addr <= 8'd40+((vcount[9:0]-370)>>2);     // 'I'
              8'd3: font_addr <= 8'd95+((vcount[9:0]-370)>>2);     // 'T'
              default:;
       endcase
       font_pix_idx <= (hcount[10:1]-288)>>1;
end
else
       msg_display_menu[5] <= 0;
```

2.2 Input Device

We use 2 kinds of USB input devices for this project, the controller and the touchpad. We use libusb-1.0 C library to read the USB message, and we have reversed the message protocol for both devices.

2.2.1 Controller

Figure 8

Figure TODO shows the controller we use for this project. This controller has 2 joysticks, 4 direction buttons, ABXY buttons, LB/RB buttons, LT/RT triggers, and 3 function buttons. For this project, we use direction buttons for change selection and the A button to confirm selection.

The message protocol for this device is as follows.
VENDOR_ID: 0x045E
PRODUCT_ID: 0x028E
Message length: 20 Bytes

Byte Offset	Value type	Description
0-1	uint_16	Device message header, 0x1400
2-3	uint_16	Button press status, each bit represents one button. 0 for unpressed, 1 for pressed. **Bit offset** **Button** 0 UP 1 DOWN 2 LEFT 3 RIGHT 4 START 5 BACK 6 LSTICK 7 RSTICK 8 LB 9 RB 10 MENU 11 UNKNOWN

		12 A 13 B 14 X 15 Y
4	uint_8	Left trigger press level
5	uint_8	Right trigger press level
6-8	int_16	**Left** stick horizontal direction Positive value - stick pushes right Negative value -stick pushes left
9-10	int_16	**Left** stick vertical direction Positive value - stick pushes up Negative value -stick pushes down
11-12	int_16	**Right** stick horizontal direction Positive value - stick pushes right Negative value -stick pushes left
13-14	int_16	**Right** stick vertical direction Positive value - stick pushes up Negative value -stick pushes down
15-19	uint_8	Unknown

2.2.2 Touchpad

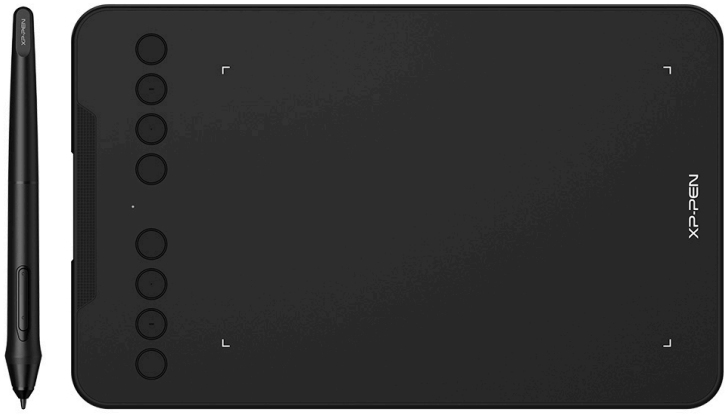


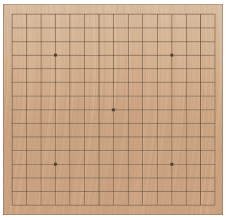
Figure 9

Figure 9 shows the touchpad we used in this project. We use this pad as a mouse. When the pen is placed or pressed on the pad, a message is sent via USB. We then retrieve the pen position, convert it to the VGA screen location, and display the cursor or trigger mouse press event.

The message protocol for this device is as follows.
VENDOR_ID: 0x28BD
PRODUCT_ID: 0x0928
Message length: 10 Bytes

Byte Offset	Value type	Description
0	uint_8	Device message header, 0x07
1	uint_8	The higher 4 bits indicate whether the pen is detected: 0xC - Pen not detected 0xA - Pen detected The lower 4 bits are the press status, each bit represents one button. 0 for unpressed, 1 for pressed. **Bit offset** **Button** 0 PEN_TOUCH 1 PEN_BUTTON1 2 PEN_BUTTON2 3 UNKNOWN
2-3	uint_16	Pen horizontal position
4-5	uint_16	Pen vertical position
6-7	uint_16	Pen pressure level
8-9	uint_8	Unknown

2.3 Memory

Name	Graphic	Size (pixels)	Bits per Pixel	Total Bytes
Board		500 x 480	4	120,000
Black Piece		33 x 31	8	1023
White Piece		33 x 31	8	1023
Fonts		45pcs x 8 x 5	1	225
Player profile		3pcs x 128 x 128	4	24,576
		Memory Budget (bits)		1,174,776

The fonts include 26 capitalized alphabets, 10 numeric characters, and 9 special characters:
`' '`, `'!'`, `'?'`, `','`, `'.'`, `'+'`, `'-'`, `'×'`, `'/'`

3. Software

3.1 Game Logic

The basic game logic and AI algorithm is written in C++, with the core variables and methods implemented in a class named ***Gomoku***. In this section, we will introduce some of the key classes that are crucial for implementing both the game.

- ***Board Validity Checks***: Methods like on_board and valid_move that checks the validity of a move based on the rule of Gomoku.
- ***Game Progression***: The make_move method updates the game state with each new move made by the player (or the AI).
- ***Win Conditions***: Methods like check, check_win verifies if either player has met the winning condition—forming a contiguous line of five stones.
- ***Game Status Updates***: Functions such as is_draw and switchPlayers manage the game's progress, determining draw conditions and alternating turns between players.
- ***Board Display***: The displayBoard is for visualization, called when simulating an actual game between player and computer or between players.
- ***Reset Game***: Reset the game to its original state, called when a game has ended, and the program waits for the user to start another game.
- ***Regret Move***: Called when a player wants to regret a move. This updates the corresponding board states, and reverts to the previous move made by the player’s opponent.
- ***Record Game***: Record the game to a file when it has ended.

3.2 AI

3.2.1 Overview

In this section, we will introduce the AI algorithm used in our Gomoku application. The AI algorithm is based on Minimax. Additionally, to address the high computational demands of the original Minimax, Alpha-Beta Pruning is used for efficiency.

MiniMax is a commonly used decision tree-based algorithm. Decision tree represents the state space of the game, with each branch representing a player making a move, and each node representing a unique kind of board configuration. We devised a method of evaluating the current board status for each player, which will be presented in the form as a score. By switching between maximizing the AI's score and minimizing the opponent's score, AI optimizes the game strategy by simulating these movements using the Minimax method.

3.2.2 Minimax

Minimax is a typical decision tree algorithm for games like Gomoku, where each node, as described above, represents a possible state in the game. At its core, Minimax seeks to maximize the potential scores for the player (the maximizer) while minimizing the gains for the opponent (the minimizer), here is the workflow of this algorithm.

1. Look at all the places where you can make a legal move, make a temporary move at each possible location, and begin simulation for that move.
2. For each move, think ahead about possible responses by the white player (opponent). Each of these responses creates further branches in the tree. Specifically, in the maximum layer, you are making moves which lead to the highest score for you, while in the minimum layer, your opponent does the very opposite thing, making moves that are worst for you, minimizing your score.
3. When the simulation for the move(mentioned in 1) is done, undo the move, move to the next possible position, and begin simulation.

4. Choose the move with the highest score.

pseudo code for MiniMax

```
Unset

function  minimax( node, depth, maximizingPlayer ) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max( value, minimax( child, depth - 1, FALSE ) )
        return value
    else (* minimizing player *)
        value := +∞
        for each child of node do
            value := min( value, minimax( child, depth - 1, TRUE ) )
        return value
```

3.2.2 Alpha-Beta Pruning

MiniMax is a brute force solution. Its time complexity is exponential, which is unacceptable for real-life applications like Gomoku. Alpha-beta pruning helps accelerate the process by deleting parts of the decision tree that don't affect the final decision. Specifically, for the maximizing player, any branch with a value lower than the current best (alpha) is ignored, for the minimizing player, branches with values higher than the current worst (beta) are disregarded. This selective process can significantly reduce the number of branches evaluated.

pseudo code for alpha-beta pruning

```
function alphabeta(node, depth, α, β, maximizingPlayer) is
    if depth == 0 or node is terminal then
        return the heuristic value of node
    if maximizingPlayer then
```

```
        value := -∞
        for each child of node do
            value := max(value, alphabeta(child, depth - 1, α, β, FALSE))
            if value > β then
                break (* β cutoff *)
            α := max(α, value)
        return value
    else
        value := +∞
        for each child of node do
            value := min(value, alphabeta(child, depth - 1, α, β, TRUE))
            if value < α then
                break (* α cutoff *)
            β := min(β, value)
        return value
```

3.3 Network Function

Network function allows two users to play against each other over a LAN. As shown in Figure 4, the “create room” option and “join room” option are for this mode. To play a game over the network, one player creates a room, and the other player joins it. It is worth noting that one cannot join the room without another player already creating a room. Otherwise, a “scanning” message will appear on the screen for 3 seconds before indicating that no room is available.

Network functionality consists of two phases: the UDP broadcast phase and the TCP connection phase.

UDP broadcast phase

As shown above, once a player chooses the “create room” or “join room” option, he/she begins to look for other opponents. Behind the scenes, the program is broadcasting UDP packets containing the current player’s IP address. Once the other player receives the packet, the two of them initiate TCP connections with each other and move to the TCP connection phase.

TCP connection phase

The player who creates the room acts as the server, while the player who joins is the client. They establish a TCP connection with each other over the network, sending the current board status after making a move, as well as other messages, such as “regret” and “resign”. Additionally, there are also some mechanisms to gracefully handle the situations where a player exits the game on purpose while the game is still in progress.

3.4 Avalon Bus Interface

Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Details
00					Sound						MSG	PF	T	P	C	M	Display
02						Black Piece	White Piece		Piece Y Position				Piece X Position				Piece information
04	Selected mark Y location				Selected mark X Location				Last Piece Y Location				Last Piece X Position				Selected mark, Last piece mark
06	Message group index						V	V	V	V	V	V	V	V	V	V	Message Visibility Info
08	Message Index, 0 means no message is selected.																Selected Message Info
10	Touchpad mark X position																Touchpad mark Info
12	Touchpad mark Y position																

Address 00, offset:

- 0 - Whether to show the menu page or the board page
- 1 - Whether to show touchpad cursor
- 2 - Player UI, player piece information. Whether player 1 takes the black piece
- 3 - Player UI, current player mark. Whether it is player 1’s turn.
- 4 - Player UI, player 2 profile image. Whether player 2’s profile image is a cat or dog
- 5 - Message box UI. Whether to show the popup message
- 8 - Victory Sound
- 9 - Defeat sound
- 10 - Show menu sound
- 11 - Piece move sound

Address 06:

- We categorize messages into groups so that we can set the visibility of a group of messages simultaneously
- 0~9 - Visibility status of each message in one group. So there are at most 10 messages in a group.
- 11~15 - Message group index. At most 64 groups
- So we could have at most 640 messages to show using this interface

4. Contribution

Each member: Design, debug, and help each other throughout the project.

- Yunzhou Li:
    - Hardware code implementation.
    - Software display logic.
    - Code structure Reconstruction.
    - Touchpad driver.
    - Part of the network functionality.
- Hongyu Sun:
    - Basic Gomoku game logic.
    - Alpha-beta pruning MiniMax Algorithm.
    - Part of network function.
- Zhiwei Xie:
    - Graphical user interface design.
    - Part of hardware display logic.
- Zixuan Fang:
    - Xbox driver.
    - Part of the software input device logic.

5. Source Code

5.1 vga_ball.sv

```
Unset
/*
 * Avalon memory-mapped peripheral that generates VGA
 *
 * Stephen A. Edwards
 * Columbia University
 */

module vga_ball(input logic             clk,
                          input logic         reset,
                          input logic [15:0]  writedata,
                          input logic         write,
                          input                       chipselect,
                          input logic [2:0]   address,

                          input                       L_READY,
                          input                       R_READY,
                          output logic [15:0] L_DATA,
                          output logic [15:0] R_DATA,
                          output logic        L_VALID,
                          output logic        R_VALID,

                          output logic [7:0]  VGA_R, VGA_G, VGA_B,
                          output logic        VGA_CLK, VGA_HS, VGA_VS,
                                                      VGA_BLANK_n,
                          output logic        VGA_SYNC_n);

       logic [10:0]        hcount;
       logic [9:0]               vcount;
       vga_counters counters(.clk50(clk), .*);

       /*****************************************************************/
       /********************* Audio Unfinished ***********************/
       /*****************************************************************/

       logic [11:0] move_addr;
       logic [15:0] move_data;
       logic play_move;
       logic move_sound_stop;

       logic [12:0] victory_addr;
       logic [15:0] victory_data;
       logic play_victory;
       logic victory_sound_stop;

       logic [10:0] menu_addr;
       logic [15:0] menu_data;
```

```
	logic play_menu;
	logic menu_sound_stop;

	logic [13:0] defeat_addr;
	logic [15:0] defeat_data;
	logic play_defeat;
	logic defeat_sound_stop;

	reg [11:0] counter;

	soc_system_move_sound
move(.address(move_addr),.clk(clk),.clken(1),.reset_req(0),.readdata(move_data));
	soc_system_victory_sound
victory(.address(victory_addr),.clk(clk),.clken(1),.reset_req(0),.readdata(victory_data));
	soc_system_menu_sound
menu(.address(menu_addr),.clk(clk),.clken(1),.reset_req(0),.readdata(menu_data));
	soc_system_defeat_sound
defeat(.address(defeat_addr),.clk(clk),.clken(1),.reset_req(0),.readdata(defeat_data));

	always_ff @(posedge clk) begin

		if (reset) begin
			counter <= 0;
			move_sound_stop <= 1;
			victory_sound_stop <=1;
			menu_sound_stop <= 1;
			defeat_sound_stop <= 1;
		end
		else if (L_READY == 1 && R_READY == 1 && counter < 3125) begin
			counter <= counter + 1;
			L_VALID <= 0;
			R_VALID <= 0;
		end
		else if (L_READY == 1 && R_READY == 1 && counter >= 3125) begin
			counter <= 0;
			L_VALID <= 1;
			R_VALID <= 1;
			if(play_move || move_sound_stop==1'b0) begin
				if(move_addr < 12'd2871) begin
					move_addr <= move_addr + 1;
					move_sound_stop <= 1'b0;
				end
				else begin
					move_addr <= 0;
					move_sound_stop <= 1'b1;
				end
				L_DATA <= move_data;
				R_DATA <= move_data;
			end
			else if(play_victory || victory_sound_stop==1'b0) begin
				if(victory_addr < 13'd6628) begin
					victory_addr <= victory_addr + 1;
					victory_sound_stop <= 1'b0;
				end
```

```
                else begin
                        victory_addr <= 0;
                        victory_sound_stop <= 1'b1;
                end
                L_DATA <= victory_data;
                R_DATA <= victory_data;
            end
            else if(play_menu || menu_sound_stop==1'b0) begin
                if(menu_addr < 11'd1497) begin
                        menu_addr <= menu_addr + 1;
                        menu_sound_stop <= 1'b0;
                end
                else begin
                        menu_addr <= 0;
                        menu_sound_stop <= 1'b1;
                end
                L_DATA <= menu_data;
                R_DATA <= menu_data;
            end
            else if(play_defeat || defeat_sound_stop==1'b0) begin
                if(defeat_addr < 14'd11096) begin
                        defeat_addr <= defeat_addr + 1;
                        defeat_sound_stop <= 1'b0;
                end
                else begin
                        defeat_addr <= 0;
                        defeat_sound_stop <= 1'b1;
                end
                L_DATA <= defeat_data;
                R_DATA <= defeat_data;
            end
            else begin
                move_addr <= 0;
                victory_addr <= 0;
                menu_addr <= 0;
                defeat_addr <= 0;
                L_DATA <= 0;
                R_DATA <= 0;
            end
        end
        else begin
            L_VALID <= 0;
            R_VALID <= 0;
        end
    end

    /////////////////////////////////////////////////////
    ////////////////// Board Display ////////////////////
    /////////////////////////////////////////////////////

    logic  is_menu;                  // Whether current page is menu
    logic  is_board;                 // Whether current page is board

    ////////// Board & piece //////////
```

```
      // Board Piece information array 15*15*info_length
      // Index - board position index, x is vertial, y is horizontal
      // 0,1,2 - piece info
      //    index 0 - show highlighted piece
      //    index 1 - show white piece
      //    index 2 - show black piece
      logic [14:0][14:0][2:0]    board;
      logic[4:0]   piece_v;            // Piece image offset v
      logic[5:0]   piece_h;            // Piece image offset h
      logic[3:0]   piece_x;            // Piece coordinate x
      logic[3:0]   piece_y;            // Piece coordinate y
      logic[2:0]   piece_info;         // Piece image information
      logic[3:0]   selected_x;         // Selected mark x position
      logic[3:0]   selected_y;         // Selected mark y position
      logic[3:0]   last_piece_x;// Last piece mark x position
      logic[3:0]   last_piece_y;// Last piece mark y position
      logic        is_piece_area;
      logic        is_board_area;
      logic        is_last_piece_area;
      logic        is_selected_area;

      assign piece_info = board[piece_x][piece_y];
      // Board Area [0,499]
      assign is_board_area = (hcount[10:1]>=10'd0) && (hcount[10:1]<=10'd500);
      // Piece 33x31
      // Piece Area: h[4,498] v[7,472]
      assign is_piece_area = (hcount[10:1]>=10'd4) && (hcount[10:1]<=10'd498) && (vcount[9:0]>=10'd7)
&& (vcount[9:0]<=10'd472);
      assign is_selected_area  = ((piece_h <= 6'd1) || (piece_h >= 6'd31) || (piece_v <= 5'd1) ||
(piece_v >= 5'd29)) && (piece_h <= 6'd5 || piece_h >= 6'd27) && (piece_v <= 5'd5 || piece_v >= 5'd25)
&& is_piece_area;
      assign is_last_piece_area = (piece_h >=6'd16) && (piece_v >= 5'd15) && (piece_h+piece_v <= 6'd43)
&& is_piece_area;

      /************ Board & piece ROM ************/
      logic [17:0] bg_addr;     // Background image ROM address
      logic [7:0]  bg_rgb;             // Background image ROM output, each output has 2 palette
values
      logic [3:0]  bg_palette;  // Background image palette value
      logic [9:0]  piece_addr;  // Piece image ROM address
      logic [7:0]  black_rgb;   // Black image ROM output
      logic [7:0]  white_rgb;   // Piece image ROM output

      assign bg_addr = is_board_area?((vcount[9:0]*17'd500+hcount[10:1])>>1):0;
      assign bg_palette = hcount[1]?bg_rgb[7:4]:bg_rgb[3:0];

      // Image ROMs
      soc_system_background
background(.address(bg_addr),.clk(clk),.clken(1),.reset_req(0),.readdata(bg_rgb));
      soc_system_white_piece
white(.address(piece_addr),.clk(clk),.clken(1),.reset_req(0),.readdata(white_rgb));
      soc_system_black_piece
black(.address(piece_addr),.clk(clk),.clken(1),.reset_req(0),.readdata(black_rgb));
```

```
// Init board info
initial begin
	for(int i=0;i<15;i++) begin
		for(int j=0;j<15;j++) begin
			board[i][j] <= 3'b0;
		end
	end
	selected_y = 4'h7;
	selected_x = 4'h7;
	last_piece_x = 4'hf;
	last_piece_y = 4'hf;
	msg_visible_ingame_options <= -1;
	msg_visible_ingame_players <= -1;
	msg_visible_ingame_message <= -1;
	msg_visible_ingame_confirm <= -1;
	//msg_display_menu <= 0;
	//msg_display_ingame <= 0;
	msg_selected = 9; // P1 selected
	//msg_visible_ingame_players[1] <= 0;
	//msg_visible_ingame_message[0] <= 1;
	black_on_top <= 0;
	board[7][7] <= 3'b1;
end

// Set piece counter
always_ff @(posedge clk) begin
	// piece left 4
	if(hcount[10:1]<=10'd4) begin
		piece_h <= 6'd0;
		piece_y <= 4'd0;
	end
	// piece right 498
	else if(hcount[10:1]>=10'd498) begin
		piece_h <= 6'd32;
		piece_y <= 4'd14;
	end
	else if(hcount[0]==0) begin
		if(piece_h==6'd32) begin
			piece_y <= piece_y + 4'd1;
			piece_h <= 0;
		end else
			piece_h <= piece_h + 6'd1;
	end
	if(hcount[10:0]==11'd0)
		// piece top 7
		if(vcount[9:0]<=10'd7) begin
			piece_v <= 5'd0;
			piece_x <= 4'd0;
		end
		// piece bottom 472
		else if(vcount[9:0]>=10'd472) begin
			piece_v <= 5'd30;
			piece_x <= 4'd14;
		end
```

```
                else if(piece_v==5'd30) begin
                        piece_x <= piece_x + 4'd1;
                        piece_v <= 0;
                end
                else begin
                        piece_v <= piece_v + 5'd1;
                end
        end



        /*****************************************************/
        /***************** Player UI Display *****************/
        /*****************************************************/

        /*********** Piece Icon **********/
        logic[4:0] icon_v;              // Player piece icon image offset v
        logic[5:0] icon_h;              // Player piece icon image offset h
        logic  is_piece_top;            // Piece for Player 1, top piece
        logic  is_piece_bottom;    // Piece for Player 1, bottom piece
        logic  black_on_top;       // Whether Player 1 takes black piece
        logic  is_black_icon_area;
        logic  is_white_icon_area;

        assign is_black_icon_area = black_on_top?is_piece_top:is_piece_bottom;
        assign is_white_icon_area = black_on_top?is_piece_bottom:is_piece_top;

        // piece top area: h=[510,543],v=[137(0+3+128+6),165(137+31))
        assign is_piece_top = (hcount[10:1]>=10'd510) && (hcount[10:1]<=10'd543) &&
(vcount[9:0]>=10'd137) && (vcount[9:0]<=10'd168);
        // piece bottom area: h=[510,543],v=[304(170+3+128+3),335(304+31))
        assign is_piece_bottom = (hcount[10:1]>=10'd510) && (hcount[10:1]<=10'd543) &&
(vcount[9:0]>=10'd304) && (vcount[9:0]<=10'd335);

        /********* Piece Icon end ********/

        /******* Player Turn Mark ********/
        logic is_p1_turn;
        logic is_p1_turn_area;
        logic is_p2_turn_area;
        logic is_turn_area;

        // horizontal [553,563), vertical [142, 162), v+h <= 553+162, v >= h + 142-553
        assign is_p1_turn_area = (hcount[10:1] >= 10'd553) && (hcount[10:1] <= 10'd715 - vcount[9:0]) &&
(vcount[9:0]>= hcount[10:1]-10'd411);
        // horizontal [553,563), vertical [309, 329), v+h <= 553+329, v >= h + 309-553
        assign is_p2_turn_area = (hcount[10:1] >= 10'd553) && (hcount[10:1] <= 10'd882 - vcount[9:0]) &&
(vcount[9:0]>= hcount[10:1]-10'd244);
        assign is_turn_area = is_p1_turn?is_p1_turn_area:is_p2_turn_area;
        /***** Player Turn Mark End ******/

        /********* Player profile *********/
        logic [12:0] p1_profile_addr;    // Player 1 profile Image ROM address
        logic [4:0]  p1_profile_palette; // Player 1 profile Image palette value
```

```
	logic [12:0] p2_profile_addr;    // Player 2 profile Image ROM address
	logic [4:0]  p2_profile_palette; // Player 2 profile Image palette value

	logic [7:0]  panda_rgb;                // Panda Image ROM output, contains 2 palette values
	logic [7:0]  dog_rgb;                  // Dog Image ROM output, contains 2 palette values
	logic [7:0]  cat_rgb;                  // Cat Image ROM output, contains 2 palette values

	logic p2_profile_cat;                  // Whether P2 profile is cat or dog
	logic is_p1_profile_area;
	logic is_p2_profile_area;

	/* p2_profile_palette is assign in the always comb below. */
	assign p1_profile_palette = hcount[1]?panda_rgb[7:4]:panda_rgb[3:0];
	assign p1_profile_addr = is_p1_profile_area? ((((vcount[9:0] - 13'd3) << 7) + (hcount[10:1] -
13'd506)) >> 1):0;
	assign p2_profile_addr = is_p2_profile_area? ((((vcount[9:0] - 13'd173) << 7) + (hcount[10:1] -
13'd506)) >> 1):0;
	/*
		Player 1 Picture 128x128
		Pic 1 Area: h[506,634] v[3,134]
	*/
	assign is_p1_profile_area = (hcount[10:1]>=10'd506) && (hcount[10:1]<=10'd634) &&
(vcount[9:0]>=10'd3) && (vcount[9:0]<=10'd134);
	/*
		Player 2 Picture 128x128
		Pic 2 Area: h[506,634] v[174,302]
	*/
	assign is_p2_profile_area = (hcount[10:1]>=10'd506) && (hcount[10:1]<=10'd634) &&
(vcount[9:0]>=10'd174) && (vcount[9:0]<=10'd302);

	soc_system_panda_avatar
panda(.address(p1_profile_addr),.clk(clk),.clken(1),.reset_req(0),.readdata(panda_rgb));
	soc_system_dog_avatar
dog(.address(p2_profile_addr),.clk(clk),.clken(1),.reset_req(0),.readdata(dog_rgb));
	soc_system_cat_avatar
cat(.address(p2_profile_addr),.clk(clk),.clken(1),.reset_req(0),.readdata(cat_rgb));

	/******* Player profile end *********/

	/************* Cursor *************/
	logic is_cursor_visible;
	logic [9:0] cursor_v;
	logic [9:0] cursor_h;
	logic is_cursor_area;

	assign is_cursor_area = (hcount[10:1]>=(cursor_h-2)) && (hcount[10:1]<=(cursor_h+2)) &&
(vcount[9:0]>=(cursor_v-2)) && (vcount[9:0]<=(cursor_v+2));

	/*********** Cursor End ***********/

	/************* Message box *************/
	// Message
	logic is_scan_area;              // Scanning message box
	logic is_message_area;           // Other message box
```

```
      logic display_message_box; // Whether show message box

      // Message Area: h[121(22+3*33),381(481-3*33)], v[179(24+31x5),270(179+31x3-2)]
      assign is_message_area = (hcount[10:1]>=10'd121) && (hcount[10:1]<=10'd382) &&
(vcount[9:0]>=10'd179) && (vcount[9:0]<=10'd270);
      // Scan Area: h[190,451], v[195,286]
      assign is_scan_area = (hcount[10:1]>=10'd190) && (hcount[10:1]<=10'd451) &&
(vcount[9:0]>=10'd195) && (vcount[9:0]<=10'd286);

      /*********** Message box end ***********/

      /********** Player UI Layout **********/
      logic  is_player1_area;    // Player 1 UI area
      logic  is_player2_area;    // Player 2 UI area
      logic  is_option_area;     // Board option area
      logic  division_line;             // Division Line

      // player 1 - top area: h=[500,680],v=[0,170)
      assign is_player1_area = (hcount[10:1]>=10'd500) && (hcount[10:1]<=10'd680) &&
(vcount[9:0]>=10'd0) && (vcount[9:0]<=10'd170);
      // player 2 - bottom area: h=[500,680],v=[170,340)
      assign is_player2_area = (hcount[10:1]>=10'd500) && (hcount[10:1]<=10'd680) &&
(vcount[9:0]>=10'd170) && (vcount[9:0]<=10'd340);
      // option area: h=[500,680],v=[340,480]
      assign is_option_area = (hcount[10:1]>=10'd500) && (hcount[10:1]<=10'd680) &&
(vcount[9:0]>=10'd340) && (vcount[9:0]<=10'd480);
      // division line
      assign division_line = (vcount[9:0] == 10'd170) && (hcount[10:1] >= 10'd500) && (hcount[10:1] <
10'd680);

      /******** Player UI Layout End ********/




      /*******************************************************/
      /**** Message display block, generated by sv_gen.py ****/
      /*******************************************************/

    // Message visibility and display signal of group menu
      logic [5:0] msg_visible_menu;
      logic [5:0] msg_display_menu;
    // Message visibility and display signal of group ingame_options
      logic [3:0] msg_visible_ingame_options;
      logic [3:0] msg_display_ingame_options;
    // Message visibility and display signal of group ingame_players
      logic [2:0] msg_visible_ingame_players;
      logic [2:0] msg_display_ingame_players;
    // Message visibility and display signal of group ingame_message
      logic [5:0] msg_visible_ingame_message;
      logic [5:0] msg_display_ingame_message;
    // Message visibility and display signal of group ingame_confirm
      logic [2:0] msg_visible_ingame_confirm;
      logic [2:0] msg_display_ingame_confirm;
```

```
	// Selected message index, 0 means none is selected
	logic [5:0] msg_selected;
	// Whether the current message is selected
	logic cur_msg_selected;
	// Font pixel index of a line, [0,8)
	logic [2:0] font_pix_idx;
	logic [7:0] font_addr;
	logic [7:0] font_val;
	soc_system_font font(.address(font_addr),.clk(clk),.clken(1),.reset_req(0),.readdata(font_val));

	always_ff @(posedge clk) begin
		/**
		* message: GOMOKU
		* group 1: menu
		* group_index: 0
		* select_index: 1
		* h_start: 128
		* v_start: 128
		* font_width: 64
		* font_height: 80
		*/
		if((hcount[10:1] >= 10'd128) && (hcount[10:1] < 10'd512) && (vcount >= 10'd128) && (vcount
< 10'd208) && msg_visible_menu[0]) begin
			msg_display_menu[0] <= 1;
			cur_msg_selected <= (msg_selected==1);
			case((hcount[10:1]-128)>>6)
				8'd0: font_addr <= 8'd30+((vcount[9:0]-128)>>4);
				8'd1: font_addr <= 8'd70+((vcount[9:0]-128)>>4);
				8'd2: font_addr <= 8'd60+((vcount[9:0]-128)>>4);
				8'd3: font_addr <= 8'd70+((vcount[9:0]-128)>>4);
				8'd4: font_addr <= 8'd50+((vcount[9:0]-128)>>4);
				8'd5: font_addr <= 8'd100+((vcount[9:0]-128)>>4);
				default:;
			endcase
			font_pix_idx <= (hcount[10:1]-128)>>3;
		end
		else
			msg_display_menu[0] <= 0;

		/**
		* message: START PVP
		* group 1: menu
		* group_index: 1
		* select_index: 2
		* h_start: 248
		* v_start: 250
		* font_width: 16
		* font_height: 20
		*/
		if((hcount[10:1] >= 10'd248) && (hcount[10:1] < 10'd392) && (vcount >= 10'd250) && (vcount
< 10'd270) && msg_visible_menu[1]) begin
			msg_display_menu[1] <= 1;
			cur_msg_selected <= (msg_selected==2);
```

```
                case((hcount[10:1]-248)>>4)
                        8'd0: font_addr <= 8'd90+((vcount[9:0]-250)>>2);
                        8'd1: font_addr <= 8'd95+((vcount[9:0]-250)>>2);
                        8'd2: font_addr <= 8'd0+((vcount[9:0]-250)>>2);
                        8'd3: font_addr <= 8'd85+((vcount[9:0]-250)>>2);
                        8'd4: font_addr <= 8'd95+((vcount[9:0]-250)>>2);
                        8'd5: font_addr <= 8'd180+((vcount[9:0]-250)>>2);
                        8'd6: font_addr <= 8'd75+((vcount[9:0]-250)>>2);
                        8'd7: font_addr <= 8'd105+((vcount[9:0]-250)>>2);
                        8'd8: font_addr <= 8'd75+((vcount[9:0]-250)>>2);
                        default:;
                endcase
                font_pix_idx <= (hcount[10:1]-248)>>1;
        end
        else
                msg_display_menu[1] <= 0;

        /**
        * message: START PVE
        * group 1: menu
        * group_index: 2
        * select_index: 3
        * h_start: 248
        * v_start: 280
        * font_width: 16
        * font_height: 20
        */
        if((hcount[10:1] >= 10'd248) && (hcount[10:1] < 10'd392) && (vcount >= 10'd280) && (vcount
< 10'd300) && msg_visible_menu[2]) begin
                msg_display_menu[2] <= 1;
                cur_msg_selected <= (msg_selected==3);
                case((hcount[10:1]-248)>>4)
                        8'd0: font_addr <= 8'd90+((vcount[9:0]-280)>>2);
                        8'd1: font_addr <= 8'd95+((vcount[9:0]-280)>>2);
                        8'd2: font_addr <= 8'd0+((vcount[9:0]-280)>>2);
                        8'd3: font_addr <= 8'd85+((vcount[9:0]-280)>>2);
                        8'd4: font_addr <= 8'd95+((vcount[9:0]-280)>>2);
                        8'd5: font_addr <= 8'd180+((vcount[9:0]-280)>>2);
                        8'd6: font_addr <= 8'd75+((vcount[9:0]-280)>>2);
                        8'd7: font_addr <= 8'd105+((vcount[9:0]-280)>>2);
                        8'd8: font_addr <= 8'd20+((vcount[9:0]-280)>>2);
                        default:;
                endcase
                font_pix_idx <= (hcount[10:1]-248)>>1;
        end
        else
                msg_display_menu[2] <= 0;

        /**
        * message: CREATE ROOM
        * group 1: menu
        * group_index: 3
        * select_index: 4
        * h_start: 232
```

```
            * v_start: 310
            * font_width: 16
            * font_height: 20
            */
            if((hcount[10:1] >= 10'd232) && (hcount[10:1] < 10'd408) && (vcount >= 10'd310) && (vcount
< 10'd330) && msg_visible_menu[3]) begin
                  msg_display_menu[3] <= 1;
                  cur_msg_selected <= (msg_selected==4);
                  case((hcount[10:1]-232)>>4)
                        8'd0: font_addr <= 8'd10+((vcount[9:0]-310)>>2);
                        8'd1: font_addr <= 8'd85+((vcount[9:0]-310)>>2);
                        8'd2: font_addr <= 8'd20+((vcount[9:0]-310)>>2);
                        8'd3: font_addr <= 8'd0+((vcount[9:0]-310)>>2);
                        8'd4: font_addr <= 8'd95+((vcount[9:0]-310)>>2);
                        8'd5: font_addr <= 8'd20+((vcount[9:0]-310)>>2);
                        8'd6: font_addr <= 8'd180+((vcount[9:0]-310)>>2);
                        8'd7: font_addr <= 8'd85+((vcount[9:0]-310)>>2);
                        8'd8: font_addr <= 8'd70+((vcount[9:0]-310)>>2);
                        8'd9: font_addr <= 8'd70+((vcount[9:0]-310)>>2);
                        8'd10: font_addr <= 8'd60+((vcount[9:0]-310)>>2);
                        default:;
                  endcase
                  font_pix_idx <= (hcount[10:1]-232)>>1;
            end
            else
                  msg_display_menu[3] <= 0;

            /**
            * message: JOIN ROOM
            * group 1: menu
            * group_index: 4
            * select_index: 5
            * h_start: 248
            * v_start: 340
            * font_width: 16
            * font_height: 20
            */
            if((hcount[10:1] >= 10'd248) && (hcount[10:1] < 10'd392) && (vcount >= 10'd340) && (vcount
< 10'd360) && msg_visible_menu[4]) begin
                  msg_display_menu[4] <= 1;
                  cur_msg_selected <= (msg_selected==5);
                  case((hcount[10:1]-248)>>4)
                        8'd0: font_addr <= 8'd45+((vcount[9:0]-340)>>2);
                        8'd1: font_addr <= 8'd70+((vcount[9:0]-340)>>2);
                        8'd2: font_addr <= 8'd40+((vcount[9:0]-340)>>2);
                        8'd3: font_addr <= 8'd65+((vcount[9:0]-340)>>2);
                        8'd4: font_addr <= 8'd180+((vcount[9:0]-340)>>2);
                        8'd5: font_addr <= 8'd85+((vcount[9:0]-340)>>2);
                        8'd6: font_addr <= 8'd70+((vcount[9:0]-340)>>2);
                        8'd7: font_addr <= 8'd70+((vcount[9:0]-340)>>2);
                        8'd8: font_addr <= 8'd60+((vcount[9:0]-340)>>2);
                        default:;
                  endcase
                  font_pix_idx <= (hcount[10:1]-248)>>1;
```

```
            end
            else
                  msg_display_menu[4] <= 0;

            /**
            * message: EXIT
            * group 1: menu
            * group_index: 5
            * select_index: 6
            * h_start: 288
            * v_start: 370
            * font_width: 16
            * font_height: 20
            */
            if((hcount[10:1] >= 10'd288) && (hcount[10:1] < 10'd352) && (vcount >= 10'd370) && (vcount
< 10'd390) && msg_visible_menu[5]) begin
                  msg_display_menu[5] <= 1;
                  cur_msg_selected <= (msg_selected==6);
                  case((hcount[10:1]-288)>>4)
                        8'd0: font_addr <= 8'd20+((vcount[9:0]-370)>>2);
                        8'd1: font_addr <= 8'd115+((vcount[9:0]-370)>>2);
                        8'd2: font_addr <= 8'd40+((vcount[9:0]-370)>>2);
                        8'd3: font_addr <= 8'd95+((vcount[9:0]-370)>>2);
                        default:;
                  endcase
                  font_pix_idx <= (hcount[10:1]-288)>>1;
            end
            else
                  msg_display_menu[5] <= 0;

            /**
            * message: Regret
            * group 2: ingame_options
            * group_index: 0
            * select_index: 7
            * h_start: 520
            * v_start: 355
            * font_width: 16
            * font_height: 20
            */
            if((hcount[10:1] >= 10'd520) && (hcount[10:1] < 10'd616) && (vcount >= 10'd355) && (vcount
< 10'd375) && msg_visible_ingame_options[0]) begin
                  msg_display_ingame_options[0] <= 1;
                  cur_msg_selected <= (msg_selected==7);
                  case((hcount[10:1]-520)>>4)
                        8'd0: font_addr <= 8'd85+((vcount[9:0]-355)>>2);
                        8'd1: font_addr <= 8'd20+((vcount[9:0]-355)>>2);
                        8'd2: font_addr <= 8'd30+((vcount[9:0]-355)>>2);
                        8'd3: font_addr <= 8'd85+((vcount[9:0]-355)>>2);
                        8'd4: font_addr <= 8'd20+((vcount[9:0]-355)>>2);
                        8'd5: font_addr <= 8'd95+((vcount[9:0]-355)>>2);
                        default:;
                  endcase
                  font_pix_idx <= (hcount[10:1]-520)>>1;
```

```
            end
            else
                  msg_display_ingame_options[0] <= 0;

            /**
            * message: HINT
            * group 2: ingame_options
            * group_index: 1
            * select_index: 8
            * h_start: 536
            * v_start: 385
            * font_width: 16
            * font_height: 20
            */
            if((hcount[10:1] >= 10'd536) && (hcount[10:1] < 10'd600) && (vcount >= 10'd385) && (vcount
< 10'd405) && msg_visible_ingame_options[1]) begin
                  msg_display_ingame_options[1] <= 1;
                  cur_msg_selected <= (msg_selected==8);
                  case((hcount[10:1]-536)>>4)
                        8'd0: font_addr <= 8'd35+((vcount[9:0]-385)>>2);
                        8'd1: font_addr <= 8'd40+((vcount[9:0]-385)>>2);
                        8'd2: font_addr <= 8'd65+((vcount[9:0]-385)>>2);
                        8'd3: font_addr <= 8'd95+((vcount[9:0]-385)>>2);
                        default:;
                  endcase
                  font_pix_idx <= (hcount[10:1]-536)>>1;
            end
            else
                  msg_display_ingame_options[1] <= 0;

            /**
            * message: RESIGN
            * group 2: ingame_options
            * group_index: 2
            * select_index: 9
            * h_start: 520
            * v_start: 415
            * font_width: 16
            * font_height: 20
            */
            if((hcount[10:1] >= 10'd520) && (hcount[10:1] < 10'd616) && (vcount >= 10'd415) && (vcount
< 10'd435) && msg_visible_ingame_options[2]) begin
                  msg_display_ingame_options[2] <= 1;
                  cur_msg_selected <= (msg_selected==9);
                  case((hcount[10:1]-520)>>4)
                        8'd0: font_addr <= 8'd85+((vcount[9:0]-415)>>2);
                        8'd1: font_addr <= 8'd20+((vcount[9:0]-415)>>2);
                        8'd2: font_addr <= 8'd90+((vcount[9:0]-415)>>2);
                        8'd3: font_addr <= 8'd40+((vcount[9:0]-415)>>2);
                        8'd4: font_addr <= 8'd30+((vcount[9:0]-415)>>2);
                        8'd5: font_addr <= 8'd65+((vcount[9:0]-415)>>2);
                        default:;
                  endcase
                  font_pix_idx <= (hcount[10:1]-520)>>1;
```

```
            end
            else
                  msg_display_ingame_options[2] <= 0;

            /**
            * message: EXIT
            * group 2: ingame_options
            * group_index: 3
            * select_index: 10
            * h_start: 536
            * v_start: 445
            * font_width: 16
            * font_height: 20
            */
            if((hcount[10:1] >= 10'd536) && (hcount[10:1] < 10'd600) && (vcount >= 10'd445) && (vcount
< 10'd465) && msg_visible_ingame_options[3]) begin
                  msg_display_ingame_options[3] <= 1;
                  cur_msg_selected <= (msg_selected==10);
                  case((hcount[10:1]-536)>>4)
                        8'd0: font_addr <= 8'd20+((vcount[9:0]-445)>>2);
                        8'd1: font_addr <= 8'd115+((vcount[9:0]-445)>>2);
                        8'd2: font_addr <= 8'd40+((vcount[9:0]-445)>>2);
                        8'd3: font_addr <= 8'd95+((vcount[9:0]-445)>>2);
                        default:;
                  endcase
                  font_pix_idx <= (hcount[10:1]-536)>>1;
            end
            else
                  msg_display_ingame_options[3] <= 0;

            /**
            * message: P1
            * group 3: ingame_players
            * group_index: 0
            * select_index: 11
            * h_start: 573
            * v_start: 142
            * font_width: 16
            * font_height: 20
            */
            if((hcount[10:1] >= 10'd573) && (hcount[10:1] < 10'd605) && (vcount >= 10'd142) && (vcount
< 10'd162) && msg_visible_ingame_players[0]) begin
                  msg_display_ingame_players[0] <= 1;
                  cur_msg_selected <= (msg_selected==11);
                  case((hcount[10:1]-573)>>4)
                        8'd0: font_addr <= 8'd75+((vcount[9:0]-142)>>2);
                        8'd1: font_addr <= 8'd135+((vcount[9:0]-142)>>2);
                        default:;
                  endcase
                  font_pix_idx <= (hcount[10:1]-573)>>1;
            end
            else
                  msg_display_ingame_players[0] <= 0;
```

```
            /**
            * message: P2
            * group 3: ingame_players
            * group_index: 1
            * select_index: 12
            * h_start: 573
            * v_start: 309
            * font_width: 16
            * font_height: 20
            */
            if((hcount[10:1] >= 10'd573) && (hcount[10:1] < 10'd605) && (vcount >= 10'd309) && (vcount
< 10'd329) && msg_visible_ingame_players[1]) begin
                  msg_display_ingame_players[1] <= 1;
                  cur_msg_selected <= (msg_selected==12);
                  case((hcount[10:1]-573)>>4)
                        8'd0: font_addr <= 8'd75+((vcount[9:0]-309)>>2);
                        8'd1: font_addr <= 8'd140+((vcount[9:0]-309)>>2);
                        default:;
                  endcase
                  font_pix_idx <= (hcount[10:1]-573)>>1;
            end
            else
                  msg_display_ingame_players[1] <= 0;

            /**
            * message: AI
            * group 3: ingame_players
            * group_index: 2
            * select_index: 13
            * h_start: 573
            * v_start: 309
            * font_width: 16
            * font_height: 20
            */
            if((hcount[10:1] >= 10'd573) && (hcount[10:1] < 10'd605) && (vcount >= 10'd309) && (vcount
< 10'd329) && msg_visible_ingame_players[2]) begin
                  msg_display_ingame_players[2] <= 1;
                  cur_msg_selected <= (msg_selected==13);
                  case((hcount[10:1]-573)>>4)
                        8'd0: font_addr <= 8'd0+((vcount[9:0]-309)>>2);
                        8'd1: font_addr <= 8'd40+((vcount[9:0]-309)>>2);
                        default:;
                  endcase
                  font_pix_idx <= (hcount[10:1]-573)>>1;
            end
            else
                  msg_display_ingame_players[2] <= 0;

            /**
            * message: You Win!
            * group 4: ingame_message
            * group_index: 0
            * select_index: 14
            * h_start: 129
```

```
            * v_start: 187
            * font_width: 32
            * font_height: 40
            */
            if((hcount[10:1] >= 10'd129) && (hcount[10:1] < 10'd385) && (vcount >= 10'd187) && (vcount
< 10'd227) && msg_visible_ingame_message[0]) begin
                  msg_display_ingame_message[0] <= 1;
                  cur_msg_selected <= (msg_selected==14);
                  case((hcount[10:1]-129)>>5)
                        8'd0: font_addr <= 8'd120+((vcount[9:0]-187)>>3);
                        8'd1: font_addr <= 8'd70+((vcount[9:0]-187)>>3);
                        8'd2: font_addr <= 8'd100+((vcount[9:0]-187)>>3);
                        8'd3: font_addr <= 8'd180+((vcount[9:0]-187)>>3);
                        8'd4: font_addr <= 8'd110+((vcount[9:0]-187)>>3);
                        8'd5: font_addr <= 8'd40+((vcount[9:0]-187)>>3);
                        8'd6: font_addr <= 8'd65+((vcount[9:0]-187)>>3);
                        8'd7: font_addr <= 8'd185+((vcount[9:0]-187)>>3);
                        default:;
                  endcase
                  font_pix_idx <= (hcount[10:1]-129)>>2;
            end
            else
                  msg_display_ingame_message[0] <= 0;

            /**
            * message: You Lose
            * group 4: ingame_message
            * group_index: 1
            * select_index: 15
            * h_start: 125
            * v_start: 187
            * font_width: 32
            * font_height: 40
            */
            if((hcount[10:1] >= 10'd125) && (hcount[10:1] < 10'd381) && (vcount >= 10'd187) && (vcount
< 10'd227) && msg_visible_ingame_message[1]) begin
                  msg_display_ingame_message[1] <= 1;
                  cur_msg_selected <= (msg_selected==15);
                  case((hcount[10:1]-125)>>5)
                        8'd0: font_addr <= 8'd120+((vcount[9:0]-187)>>3);
                        8'd1: font_addr <= 8'd70+((vcount[9:0]-187)>>3);
                        8'd2: font_addr <= 8'd100+((vcount[9:0]-187)>>3);
                        8'd3: font_addr <= 8'd180+((vcount[9:0]-187)>>3);
                        8'd4: font_addr <= 8'd55+((vcount[9:0]-187)>>3);
                        8'd5: font_addr <= 8'd70+((vcount[9:0]-187)>>3);
                        8'd6: font_addr <= 8'd90+((vcount[9:0]-187)>>3);
                        8'd7: font_addr <= 8'd20+((vcount[9:0]-187)>>3);
                        default:;
                  endcase
                  font_pix_idx <= (hcount[10:1]-125)>>2;
            end
            else
                  msg_display_ingame_message[1] <= 0;
```

```
            /**
            * message: P1 Win!
            * group 4: ingame_message
            * group_index: 2
            * select_index: 16
            * h_start: 150
            * v_start: 187
            * font_width: 32
            * font_height: 40
            */
            if((hcount[10:1] >= 10'd150) && (hcount[10:1] < 10'd374) && (vcount >= 10'd187) && (vcount
< 10'd227) && msg_visible_ingame_message[2]) begin
                  msg_display_ingame_message[2] <= 1;
                  cur_msg_selected <= (msg_selected==16);
                  case((hcount[10:1]-150)>>5)
                        8'd0: font_addr <= 8'd75+((vcount[9:0]-187)>>3);
                        8'd1: font_addr <= 8'd135+((vcount[9:0]-187)>>3);
                        8'd2: font_addr <= 8'd180+((vcount[9:0]-187)>>3);
                        8'd3: font_addr <= 8'd110+((vcount[9:0]-187)>>3);
                        8'd4: font_addr <= 8'd40+((vcount[9:0]-187)>>3);
                        8'd5: font_addr <= 8'd65+((vcount[9:0]-187)>>3);
                        8'd6: font_addr <= 8'd185+((vcount[9:0]-187)>>3);
                        default:;
                  endcase
                  font_pix_idx <= (hcount[10:1]-150)>>2;
            end
            else
                  msg_display_ingame_message[2] <= 0;

            /**
            * message: P2 Win!
            * group 4: ingame_message
            * group_index: 3
            * select_index: 17
            * h_start: 150
            * v_start: 187
            * font_width: 32
            * font_height: 40
            */
            if((hcount[10:1] >= 10'd150) && (hcount[10:1] < 10'd374) && (vcount >= 10'd187) && (vcount
< 10'd227) && msg_visible_ingame_message[3]) begin
                  msg_display_ingame_message[3] <= 1;
                  cur_msg_selected <= (msg_selected==17);
                  case((hcount[10:1]-150)>>5)
                        8'd0: font_addr <= 8'd75+((vcount[9:0]-187)>>3);
                        8'd1: font_addr <= 8'd140+((vcount[9:0]-187)>>3);
                        8'd2: font_addr <= 8'd180+((vcount[9:0]-187)>>3);
                        8'd3: font_addr <= 8'd110+((vcount[9:0]-187)>>3);
                        8'd4: font_addr <= 8'd40+((vcount[9:0]-187)>>3);
                        8'd5: font_addr <= 8'd65+((vcount[9:0]-187)>>3);
                        8'd6: font_addr <= 8'd185+((vcount[9:0]-187)>>3);
                        default:;
```

```
                endcase
                font_pix_idx <= (hcount[10:1]-150)>>2;
            end
            else
                msg_display_ingame_message[3] <= 0;


            /**
            * message: Are You Sure?
            * group 4: ingame_message
            * group_index: 4
            * select_index: 18
            * h_start: 150
            * v_start: 190
            * font_width: 16
            * font_height: 20
            */
            if((hcount[10:1] >= 10'd150) && (hcount[10:1] < 10'd358) && (vcount >= 10'd190) && (vcount
< 10'd210) && msg_visible_ingame_message[4]) begin
                msg_display_ingame_message[4] <= 1;
                cur_msg_selected <= (msg_selected==18);
                case((hcount[10:1]-150)>>4)
                    8'd0: font_addr <= 8'd0+((vcount[9:0]-190)>>2);
                    8'd1: font_addr <= 8'd85+((vcount[9:0]-190)>>2);
                    8'd2: font_addr <= 8'd20+((vcount[9:0]-190)>>2);
                    8'd3: font_addr <= 8'd180+((vcount[9:0]-190)>>2);
                    8'd4: font_addr <= 8'd120+((vcount[9:0]-190)>>2);
                    8'd5: font_addr <= 8'd70+((vcount[9:0]-190)>>2);
                    8'd6: font_addr <= 8'd100+((vcount[9:0]-190)>>2);
                    8'd7: font_addr <= 8'd180+((vcount[9:0]-190)>>2);
                    8'd8: font_addr <= 8'd90+((vcount[9:0]-190)>>2);
                    8'd9: font_addr <= 8'd100+((vcount[9:0]-190)>>2);
                    8'd10: font_addr <= 8'd85+((vcount[9:0]-190)>>2);
                    8'd11: font_addr <= 8'd20+((vcount[9:0]-190)>>2);
                    8'd12: font_addr <= 8'd190+((vcount[9:0]-190)>>2);
                    default:;
                endcase
                font_pix_idx <= (hcount[10:1]-150)>>1;
            end
            else
                msg_display_ingame_message[4] <= 0;


            /**
            * message: Scanning...
            * group 4: ingame_message
            * group_index: 5
            * select_index: 19
            * h_start: 240
            * v_start: 230
            * font_width: 16
            * font_height: 20
            */
```

```
            if((hcount[10:1] >= 10'd240) && (hcount[10:1] < 10'd416) && (vcount >= 10'd230) && (vcount
< 10'd250) && msg_visible_ingame_message[5]) begin
                  msg_display_ingame_message[5] <= 1;
                  cur_msg_selected <= (msg_selected==19);
                  case((hcount[10:1]-240)>>4)
                        8'd0: font_addr <= 8'd90+((vcount[9:0]-230)>>2);
                        8'd1: font_addr <= 8'd10+((vcount[9:0]-230)>>2);
                        8'd2: font_addr <= 8'd0+((vcount[9:0]-230)>>2);
                        8'd3: font_addr <= 8'd65+((vcount[9:0]-230)>>2);
                        8'd4: font_addr <= 8'd65+((vcount[9:0]-230)>>2);
                        8'd5: font_addr <= 8'd40+((vcount[9:0]-230)>>2);
                        8'd6: font_addr <= 8'd65+((vcount[9:0]-230)>>2);
                        8'd7: font_addr <= 8'd30+((vcount[9:0]-230)>>2);
                        8'd8: font_addr <= 8'd200+((vcount[9:0]-230)>>2);
                        8'd9: font_addr <= 8'd200+((vcount[9:0]-230)>>2);
                        8'd10: font_addr <= 8'd200+((vcount[9:0]-230)>>2);
                        default:;
                  endcase
                  font_pix_idx <= (hcount[10:1]-240)>>1;
            end
            else
                  msg_display_ingame_message[5] <= 0;


            /**
            * message: EXIT
            * group 5: ingame_confirm
            * group_index: 0
            * select_index: 20
            * h_start: 220
            * v_start: 241
            * font_width: 16
            * font_height: 20
            */
            if((hcount[10:1] >= 10'd220) && (hcount[10:1] < 10'd284) && (vcount >= 10'd241) && (vcount
< 10'd261) && msg_visible_ingame_confirm[0]) begin
                  msg_display_ingame_confirm[0] <= 1;
                  cur_msg_selected <= (msg_selected==20);
                  case((hcount[10:1]-220)>>4)
                        8'd0: font_addr <= 8'd20+((vcount[9:0]-241)>>2);
                        8'd1: font_addr <= 8'd115+((vcount[9:0]-241)>>2);
                        8'd2: font_addr <= 8'd40+((vcount[9:0]-241)>>2);
                        8'd3: font_addr <= 8'd95+((vcount[9:0]-241)>>2);
                        default:;
                  endcase
                  font_pix_idx <= (hcount[10:1]-220)>>1;
            end
            else
                  msg_display_ingame_confirm[0] <= 0;

            /**
            * message: yes
            * group 5: ingame_confirm
            * group_index: 1
```

```
            * select_index: 21
            * h_start: 164
            * v_start: 244
            * font_width: 16
            * font_height: 20
            */
            if((hcount[10:1] >= 10'd164) && (hcount[10:1] < 10'd212) && (vcount >= 10'd244) && (vcount
< 10'd264) && msg_visible_ingame_confirm[1]) begin
                  msg_display_ingame_confirm[1] <= 1;
                  cur_msg_selected <= (msg_selected==21);
                  case((hcount[10:1]-164)>>4)
                        8'd0: font_addr <= 8'd120+((vcount[9:0]-244)>>2);
                        8'd1: font_addr <= 8'd20+((vcount[9:0]-244)>>2);
                        8'd2: font_addr <= 8'd90+((vcount[9:0]-244)>>2);
                        default:;
                  endcase
                  font_pix_idx <= (hcount[10:1]-164)>>1;
            end
            else
                  msg_display_ingame_confirm[1] <= 0;

            /**
            * message: no
            * group 5: ingame_confirm
            * group_index: 2
            * select_index: 22
            * h_start: 311
            * v_start: 244
            * font_width: 16
            * font_height: 20
            */
            if((hcount[10:1] >= 10'd311) && (hcount[10:1] < 10'd343) && (vcount >= 10'd244) && (vcount
< 10'd264) && msg_visible_ingame_confirm[2]) begin
                  msg_display_ingame_confirm[2] <= 1;
                  cur_msg_selected <= (msg_selected==22);
                  case((hcount[10:1]-311)>>4)
                        8'd0: font_addr <= 8'd65+((vcount[9:0]-244)>>2);
                        8'd1: font_addr <= 8'd70+((vcount[9:0]-244)>>2);
                        default:;
                  endcase
                  font_pix_idx <= (hcount[10:1]-311)>>1;
            end
            else
                  msg_display_ingame_confirm[2] <= 0;
      end

      /*****************************************************/
      /************** Message display block end **************/
      /*****************************************************/


      /*****************************************************/
      /****************** Avalon Interface *******************/
      /*****************************************************/
```

```
/*
 * Whether the game starts
 * If the game hasn't started,
 * don't show P2 profile information.
 */
logic game_started;

// When P2 player name is shown, game starts
assign game_started = (msg_visible_ingame_players[2:1] > 0);

always_ff @(posedge clk)
	/************ Initial **************/
	if (reset) begin
		is_menu <= 1;		// Show Menu on boot
		is_board <= 0;		// Show Board on boot

		// Clear board
		for(int i=0;i<15;i++) begin
			for(int j=0;j<15;j++) begin
				board[i][j] <= 3'b0;
			end
		end
		play_move <= 0;
		play_victory <= 0;
		play_menu <= 0;
		play_defeat <= 0;
	end
	/************ Initial end *************/
	else if (chipselect && write) begin
		case (address)
			3'h0 : begin
				// Reset command
				if(writedata==16'hffff) begin
					for(int i=0;i<15;i++) begin
						for(int j=0;j<15;j++) begin
							board[i][j] <= 3'b0;
						end
					end
					selected_y = 4'h7;
					selected_x = 4'h7;
					last_piece_x = 4'hf;
					last_piece_y = 4'hf;
				end
				else begin // Display options
					is_menu <= writedata[0];
					is_cursor_visible <= writedata[1];
					black_on_top <= writedata[2];
					is_p1_turn <= writedata[3];
					p2_profile_cat <= writedata[4];
					display_message_box <= writedata[5];
					play_victory <= writedata[8];
					play_defeat <= writedata[9];
					play_menu <= writedata[10];
```

```
                                play_move <= writedata[11];
                        end
                end
                3'h1 : board[writedata[3:0]][writedata[7:4]][2:0] <= writedata[10:8];

                3'h2 : {selected_y,selected_x,last_piece_y,last_piece_x} <= writedata[15:0];
                3'h3 : begin
                        case(writedata[15:10])
                                6'h1:msg_visible_menu <= writedata[5:0];
                                6'h2:msg_visible_ingame_options <= writedata[3:0];
                                6'h3:msg_visible_ingame_players <= writedata[2:0];
                                6'h4:msg_visible_ingame_message <= writedata[5:0];
                                6'h5:msg_visible_ingame_confirm <= writedata[2:0];
                                default:;
                        endcase
                end
                3'h4 : msg_selected <= writedata;
                3'h5 : cursor_h <= ((writedata>2)?writedata:2);
                3'h6 : cursor_v <= ((writedata>2)?writedata:2);
                default:;
        endcase
    end

    /*****************************************************/
    /************** Main Display Logic *******************/
    /*****************************************************/

always_comb begin
        {VGA_R, VGA_G, VGA_B} = 24'h0;
        /* Piece Image address */
        piece_addr = piece_v*33+piece_h;
        icon_h = 0;
        icon_v = 0;
        if(is_piece_top) begin
                icon_h = hcount[10:1] - 510;
                icon_v = vcount[9:0] - 137;
                piece_addr = icon_v*33+icon_h;
        end
        else if (is_piece_bottom) begin
                icon_h = hcount[10:1] - 510;
                icon_v = vcount[9:0] - 304;
                piece_addr = icon_v*33+icon_h;
        end
        /* p2_profile palette */
        if(p2_profile_cat)
                p2_profile_palette = hcount[10:1]?cat_rgb[7:4]:cat_rgb[3:0];
        else
                p2_profile_palette = hcount[10:1]?dog_rgb[7:4]:dog_rgb[3:0];

        /************** Draw Graphics *************/
        if (VGA_BLANK_n) begin
                /*************** Cursor ***************/
                if(is_cursor_area && is_cursor_visible) begin
                        {VGA_R, VGA_G, VGA_B} = 24'hf5a6d0;
```

```
                    end
                    /************** Menu Page *************/
                    else if (is_menu) begin //draw menu
                           // Menu background
                           {VGA_R, VGA_G, VGA_B} = 24'h00DDAA;
                           /************* Scan Message Box ***********/
                           if(is_scan_area && display_message_box) begin
                                  // Message background: light grey
                                  {VGA_R, VGA_G, VGA_B} = 24'he4e6ed;
                                  // Draw ingame_message
                                  if(msg_display_ingame_message && font_val[font_pix_idx]) begin

                                         {VGA_R, VGA_G, VGA_B} = 24'h000000;
                                  end
                           end
                           // Draw menu message
                           else if(msg_display_menu && font_val[font_pix_idx]) begin
                                  // Font selected
                                  if(cur_msg_selected)
                                         {VGA_R, VGA_G, VGA_B} = 24'h00ffff;
                                  else
                                         {VGA_R, VGA_G, VGA_B} = 24'h000000;
                           end
                    end
                    /************** Board Page *************/
                    else if(is_board) begin
                           /************** Message Box *************/
                           if(is_message_area && display_message_box) begin
                                  // Message background
                                  {VGA_R, VGA_G, VGA_B} = 24'he4e6ed;
                                  // Draw ingame_message
                                  if(msg_display_ingame_message && font_val[font_pix_idx]) begin

                                         {VGA_R, VGA_G, VGA_B} = 24'h000000;
                                  end
                                  // Draw ingame_confirm message
                                  else if(msg_display_ingame_confirm && font_val[font_pix_idx]) begin
                                         // Font selected
                                         if(cur_msg_selected)
                                                {VGA_R, VGA_G, VGA_B} = 24'h00ffff;
                                         else
                                                {VGA_R, VGA_G, VGA_B} = 24'h000000;
                                  end
                           end
                           /************** Board Area *************/
                           // Draw selected border
                           else if(is_selected_area && (piece_x==selected_x) && (piece_y==selected_y))
begin
                                  {VGA_R, VGA_G, VGA_B} =
(is_p1_turn^black_on_top)?24'hffffff:24'h000000;
                           end
                           // Draw last piece mark
                           else if (is_last_piece_area && (piece_x==last_piece_x) &&
(piece_y==last_piece_y)) begin
```

```
                        {VGA_R, VGA_G, VGA_B} = 24'h256bd3;
                    end
                    // Draw White or Highlighted Piece, but don't draw transparent
                    // Or draw player piece icon
                    else if((((piece_info[2]==1'b1 || piece_info[0]==1'b1) && is_piece_area) ||
(is_white_icon_area && game_started)) && white_rgb!=8'h00) begin
                        case(white_rgb)
                            8'h00 : {VGA_R, VGA_G, VGA_B} = 24'h000000;
                            8'h01 : {VGA_R, VGA_G, VGA_B} = 24'hcecece;
                            8'h02 : {VGA_R, VGA_G, VGA_B} = 24'hc5c5c5;
                            8'h03 : {VGA_R, VGA_G, VGA_B} = 24'hd8d8d8;
                            8'h04 : {VGA_R, VGA_G, VGA_B} = 24'hdfdfdf;
                            8'h05 : {VGA_R, VGA_G, VGA_B} = 24'hdcdcdc;
                            8'h06 : {VGA_R, VGA_G, VGA_B} = 24'hc6c6c6;
                            8'h07 : {VGA_R, VGA_G, VGA_B} = 24'ha8a8a8;
                            8'h08 : {VGA_R, VGA_G, VGA_B} = 24'h8a8a8a;
                            8'h09 : {VGA_R, VGA_G, VGA_B} = 24'hbababa;
                            8'h0a : {VGA_R, VGA_G, VGA_B} = 24'he7e7e7;
                            8'h0b : {VGA_R, VGA_G, VGA_B} = 24'hdadada;
                            8'h0c : {VGA_R, VGA_G, VGA_B} = 24'hdedede;
                            8'h0d : {VGA_R, VGA_G, VGA_B} = 24'he0e0e0;
                            8'h0e : {VGA_R, VGA_G, VGA_B} = 24'hc7c7c7;
                            8'h0f : {VGA_R, VGA_G, VGA_B} = 24'hc8c8c8;
                            8'h10 : {VGA_R, VGA_G, VGA_B} = 24'hd2d2d2;
                            8'h11 : {VGA_R, VGA_G, VGA_B} = 24'hcccccc;
                            8'h12 : {VGA_R, VGA_G, VGA_B} = 24'ha2a2a2;
                            8'h13 : {VGA_R, VGA_G, VGA_B} = 24'h777777;
                            8'h14 : {VGA_R, VGA_G, VGA_B} = 24'he9e9e9;
                            8'h15 : {VGA_R, VGA_G, VGA_B} = 24'he4e4e4;
                            8'h16 : {VGA_R, VGA_G, VGA_B} = 24'hd5d5d5;
                            8'h17 : {VGA_R, VGA_G, VGA_B} = 24'hd9d9d9;
                            8'h18 : {VGA_R, VGA_G, VGA_B} = 24'hc4c4c4;
                            8'h19 : {VGA_R, VGA_G, VGA_B} = 24'hd7d7d7;
                            8'h1a : {VGA_R, VGA_G, VGA_B} = 24'hd4d4d4;
                            8'h1b : {VGA_R, VGA_G, VGA_B} = 24'hcacaca;
                            8'h1c : {VGA_R, VGA_G, VGA_B} = 24'ha4a4a4;
                            8'h1d : {VGA_R, VGA_G, VGA_B} = 24'hdddddd;
                            8'h1e : {VGA_R, VGA_G, VGA_B} = 24'hecec ec;
                            8'h1f : {VGA_R, VGA_G, VGA_B} = 24'hdbdbdb;
                            8'h20 : {VGA_R, VGA_G, VGA_B} = 24'hebebeb;
                            8'h21 : {VGA_R, VGA_G, VGA_B} = 24'he2e2e2;
                            8'h22 : {VGA_R, VGA_G, VGA_B} = 24'hd3d3d3;
                            8'h23 : {VGA_R, VGA_G, VGA_B} = 24'hcfcfcf;
                            8'h24 : {VGA_R, VGA_G, VGA_B} = 24'hd1d1d1;
                            8'h25 : {VGA_R, VGA_G, VGA_B} = 24'hbbbbbb;
                            8'h26 : {VGA_R, VGA_G, VGA_B} = 24'he8e8e8;
                            8'h27 : {VGA_R, VGA_G, VGA_B} = 24'hdedede;
                            8'h28 : {VGA_R, VGA_G, VGA_B} = 24'he5e5e5;
                            8'h29 : {VGA_R, VGA_G, VGA_B} = 24'hf2f2f2;
                            8'h2a : {VGA_R, VGA_G, VGA_B} = 24'hd0d0d0;
                            8'h2b : {VGA_R, VGA_G, VGA_B} = 24'hb7b7b7;
                            8'h2c : {VGA_R, VGA_G, VGA_B} = 24'hf6f6f6;
                            8'h2d : {VGA_R, VGA_G, VGA_B} = 24'hf8f8f8;
                            8'h2e : {VGA_R, VGA_G, VGA_B} = 24'hf1f1f1;
```

```
8'h2f : {VGA_R, VGA_G, VGA_B} = 24'hfafafa;
8'h30 : {VGA_R, VGA_G, VGA_B} = 24'hd6d6d6;
8'h31 : {VGA_R, VGA_G, VGA_B} = 24'hc1c1c1;
8'h32 : {VGA_R, VGA_G, VGA_B} = 24'hc3c3c3;
8'h33 : {VGA_R, VGA_G, VGA_B} = 24'hbdbdbd;
8'h34 : {VGA_R, VGA_G, VGA_B} = 24'hc9c9c9;
8'h35 : {VGA_R, VGA_G, VGA_B} = 24'hffffff;
8'h36 : {VGA_R, VGA_G, VGA_B} = 24'hfefefe;
8'h37 : {VGA_R, VGA_G, VGA_B} = 24'hfcfcfc;
8'h38 : {VGA_R, VGA_G, VGA_B} = 24'hf9f9f9;
8'h39 : {VGA_R, VGA_G, VGA_B} = 24'hefefef;
8'h3a : {VGA_R, VGA_G, VGA_B} = 24'he1e1e1;
8'h3b : {VGA_R, VGA_G, VGA_B} = 24'hbcbcbc;
8'h3c : {VGA_R, VGA_G, VGA_B} = 24'hb5b5b5;
8'h3d : {VGA_R, VGA_G, VGA_B} = 24'h898989;
8'h3e : {VGA_R, VGA_G, VGA_B} = 24'hcdcdcd;
8'h3f : {VGA_R, VGA_G, VGA_B} = 24'hf0f0f0;
8'h40 : {VGA_R, VGA_G, VGA_B} = 24'hf7f7f7;
8'h41 : {VGA_R, VGA_G, VGA_B} = 24'hfbfbfb;
8'h42 : {VGA_R, VGA_G, VGA_B} = 24'h7a7a7a;
8'h43 : {VGA_R, VGA_G, VGA_B} = 24'ha7a7a7;
8'h44 : {VGA_R, VGA_G, VGA_B} = 24'hbebebe;
8'h45 : {VGA_R, VGA_G, VGA_B} = 24'ha3a3a3;
8'h46 : {VGA_R, VGA_G, VGA_B} = 24'h363636;
8'h47 : {VGA_R, VGA_G, VGA_B} = 24'he3e3e3;
8'h48 : {VGA_R, VGA_G, VGA_B} = 24'hcbcbcb;
8'h49 : {VGA_R, VGA_G, VGA_B} = 24'hc0c0c0;
8'h4a : {VGA_R, VGA_G, VGA_B} = 24'hb2b2b2;
8'h4b : {VGA_R, VGA_G, VGA_B} = 24'hb0b0b0;
8'h4c : {VGA_R, VGA_G, VGA_B} = 24'h565656;
8'h4d : {VGA_R, VGA_G, VGA_B} = 24'hb9b9b9;
8'h4e : {VGA_R, VGA_G, VGA_B} = 24'h676767;
8'h4f : {VGA_R, VGA_G, VGA_B} = 24'h353535;
8'h50 : {VGA_R, VGA_G, VGA_B} = 24'heaeaea;
8'h51 : {VGA_R, VGA_G, VGA_B} = 24'he6e6e6;
8'h52 : {VGA_R, VGA_G, VGA_B} = 24'hc2c2c2;
8'h53 : {VGA_R, VGA_G, VGA_B} = 24'h6f6f6f;
8'h54 : {VGA_R, VGA_G, VGA_B} = 24'h313131;
8'h55 : {VGA_R, VGA_G, VGA_B} = 24'hb8b8b8;
8'h56 : {VGA_R, VGA_G, VGA_B} = 24'hb6b6b6;
8'h57 : {VGA_R, VGA_G, VGA_B} = 24'h797979;
8'h58 : {VGA_R, VGA_G, VGA_B} = 24'h252525;
8'h59 : {VGA_R, VGA_G, VGA_B} = 24'hbfbfbf;
8'h5a : {VGA_R, VGA_G, VGA_B} = 24'hb3b3b3;
8'h5b : {VGA_R, VGA_G, VGA_B} = 24'h787878;
8'h5c : {VGA_R, VGA_G, VGA_B} = 24'h222222;
8'h5d : {VGA_R, VGA_G, VGA_B} = 24'h6c6c6c;
8'h5e : {VGA_R, VGA_G, VGA_B} = 24'h272727;
8'h5f : {VGA_R, VGA_G, VGA_B} = 24'hadadad;
8'h60 : {VGA_R, VGA_G, VGA_B} = 24'hababab;
8'h61 : {VGA_R, VGA_G, VGA_B} = 24'h5d5d5d;
8'h62 : {VGA_R, VGA_G, VGA_B} = 24'h292929;
8'h63 : {VGA_R, VGA_G, VGA_B} = 24'h9e9e9e;
8'h64 : {VGA_R, VGA_G, VGA_B} = 24'hb4b4b4;
```

```
                                    8'h65 : {VGA_R, VGA_G, VGA_B} = 24'ha6a6a6;
                                    8'h66 : {VGA_R, VGA_G, VGA_B} = 24'h404040;
                                    8'h67 : {VGA_R, VGA_G, VGA_B} = 24'h323232;
                                    8'h68 : {VGA_R, VGA_G, VGA_B} = 24'hafafaf;
                                    8'h69 : {VGA_R, VGA_G, VGA_B} = 24'h8c8c8c;
                                    8'h6a : {VGA_R, VGA_G, VGA_B} = 24'h2e2e2e;
                                    8'h6b : {VGA_R, VGA_G, VGA_B} = 24'h333333;
                                    8'h6c : {VGA_R, VGA_G, VGA_B} = 24'haeaeae;
                                    8'h6d : {VGA_R, VGA_G, VGA_B} = 24'hacacac;
                                    8'h6e : {VGA_R, VGA_G, VGA_B} = 24'ha9a9a9;
                                    8'h6f : {VGA_R, VGA_G, VGA_B} = 24'h5c5c5c;
                                    8'h70 : {VGA_R, VGA_G, VGA_B} = 24'h2f2f2f;
                                    8'h71 : {VGA_R, VGA_G, VGA_B} = 24'h8e8e8e;
                                    8'h72 : {VGA_R, VGA_G, VGA_B} = 24'hb1b1b1;
                                    8'h73 : {VGA_R, VGA_G, VGA_B} = 24'ha0a0a0;
                                    8'h74 : {VGA_R, VGA_G, VGA_B} = 24'h515151;
                                    8'h75 : {VGA_R, VGA_G, VGA_B} = 24'h2c2c2c;
                                    8'h76 : {VGA_R, VGA_G, VGA_B} = 24'haaaaaa;
                                    8'h77 : {VGA_R, VGA_G, VGA_B} = 24'h686868;
                                    8'h78 : {VGA_R, VGA_G, VGA_B} = 24'h2d2d2d;
                                    8'h79 : {VGA_R, VGA_G, VGA_B} = 24'h303030;
                                    8'h7a : {VGA_R, VGA_G, VGA_B} = 24'h8f8f8f;
                                    8'h7b : {VGA_R, VGA_G, VGA_B} = 24'h909090;
                                    8'h7c : {VGA_R, VGA_G, VGA_B} = 24'h2b2b2b;
                                    8'h7d : {VGA_R, VGA_G, VGA_B} = 24'h525252;
                                    8'h7e : {VGA_R, VGA_G, VGA_B} = 24'ha5a5a5;
                                    8'h7f : {VGA_R, VGA_G, VGA_B} = 24'h8d8d8d;
                                    8'h80 : {VGA_R, VGA_G, VGA_B} = 24'h585858;
                                    8'h81 : {VGA_R, VGA_G, VGA_B} = 24'h373737;
                                    8'h82 : {VGA_R, VGA_G, VGA_B} = 24'h282828;
                                    8'h83 : {VGA_R, VGA_G, VGA_B} = 24'h737373;
                                    8'h84 : {VGA_R, VGA_G, VGA_B} = 24'h5e5e5e;
                                    8'h85 : {VGA_R, VGA_G, VGA_B} = 24'h424242;
                                    default:{VGA_R, VGA_G, VGA_B} = 24'hffffff;
                             endcase
                             // Highlighted white piece
                             if(piece_info[0]==1'b1)
                                    VGA_B = 8'd128;
                      end
                      // Draw Black Piece
                      // Or draw player piece icon
                      else if(((piece_info[1]==1'b1 && is_piece_area) || (is_black_icon_area &&
game_started)) && black_rgb!=8'h00) begin
                             case(black_rgb)
                                    8'h00 : {VGA_R, VGA_G, VGA_B} = 24'h000000;
                                    8'h01 : {VGA_R, VGA_G, VGA_B} = 24'h535353;
                                    8'h02 : {VGA_R, VGA_G, VGA_B} = 24'h4b4b4b;
                                    8'h03 : {VGA_R, VGA_G, VGA_B} = 24'h454545;
                                    8'h04 : {VGA_R, VGA_G, VGA_B} = 24'h3d3d3d;
                                    8'h05 : {VGA_R, VGA_G, VGA_B} = 24'h373737;
                                    8'h06 : {VGA_R, VGA_G, VGA_B} = 24'h343434;
                                    8'h07 : {VGA_R, VGA_G, VGA_B} = 24'h323232;
                                    8'h08 : {VGA_R, VGA_G, VGA_B} = 24'h272727;
                                    8'h09 : {VGA_R, VGA_G, VGA_B} = 24'h5e5e5e;
```

```
                8'h0a : {VGA_R, VGA_G, VGA_B} = 24'h5c5c5c;
                8'h0b : {VGA_R, VGA_G, VGA_B} = 24'h575757;
                8'h0c : {VGA_R, VGA_G, VGA_B} = 24'h515151;
                8'h0d : {VGA_R, VGA_G, VGA_B} = 24'h4a4a4a;
                8'h0e : {VGA_R, VGA_G, VGA_B} = 24'h424242;
                8'h0f : {VGA_R, VGA_G, VGA_B} = 24'h383838;
                8'h10 : {VGA_R, VGA_G, VGA_B} = 24'h2f2f2f;
                8'h11 : {VGA_R, VGA_G, VGA_B} = 24'h282828;
                8'h12 : {VGA_R, VGA_G, VGA_B} = 24'h202020;
                8'h13 : {VGA_R, VGA_G, VGA_B} = 24'h1c1c1c;
                8'h14 : {VGA_R, VGA_G, VGA_B} = 24'h666666;
                8'h15 : {VGA_R, VGA_G, VGA_B} = 24'h6b6b6b;
                8'h16 : {VGA_R, VGA_G, VGA_B} = 24'h6f6f6f;
                8'h17 : {VGA_R, VGA_G, VGA_B} = 24'h6e6e6e;
                8'h18 : {VGA_R, VGA_G, VGA_B} = 24'h676767;
                8'h19 : {VGA_R, VGA_G, VGA_B} = 24'h606060;
                8'h1a : {VGA_R, VGA_G, VGA_B} = 24'h585858;
                8'h1b : {VGA_R, VGA_G, VGA_B} = 24'h4e4e4e;
                8'h1c : {VGA_R, VGA_G, VGA_B} = 24'h444444;
                8'h1d : {VGA_R, VGA_G, VGA_B} = 24'h393939;
                8'h1e : {VGA_R, VGA_G, VGA_B} = 24'h252525;
                8'h1f : {VGA_R, VGA_G, VGA_B} = 24'h161616;
                8'h20 : {VGA_R, VGA_G, VGA_B} = 24'h1f1f1f;
                8'h21 : {VGA_R, VGA_G, VGA_B} = 24'h777777;
                8'h22 : {VGA_R, VGA_G, VGA_B} = 24'h7a7a7a;
                8'h23 : {VGA_R, VGA_G, VGA_B} = 24'h7d7d7d;
                8'h24 : {VGA_R, VGA_G, VGA_B} = 24'h767676;
                8'h25 : {VGA_R, VGA_G, VGA_B} = 24'h656565;
                8'h26 : {VGA_R, VGA_G, VGA_B} = 24'h5a5a5a;
                8'h27 : {VGA_R, VGA_G, VGA_B} = 24'h363636;
                8'h28 : {VGA_R, VGA_G, VGA_B} = 24'h2a2a2a;
                8'h29 : {VGA_R, VGA_G, VGA_B} = 24'h212121;
                8'h2a : {VGA_R, VGA_G, VGA_B} = 24'h181818;
                8'h2b : {VGA_R, VGA_G, VGA_B} = 24'h111111;
                8'h2c : {VGA_R, VGA_G, VGA_B} = 24'h0d0d0d;
                8'h2d : {VGA_R, VGA_G, VGA_B} = 24'h707070;
                8'h2e : {VGA_R, VGA_G, VGA_B} = 24'h7c7c7c;
                8'h2f : {VGA_R, VGA_G, VGA_B} = 24'h828282;
                8'h30 : {VGA_R, VGA_G, VGA_B} = 24'h888888;
                8'h31 : {VGA_R, VGA_G, VGA_B} = 24'h8b8b8b;
                8'h32 : {VGA_R, VGA_G, VGA_B} = 24'h646464;
                8'h33 : {VGA_R, VGA_G, VGA_B} = 24'h494949;
                8'h34 : {VGA_R, VGA_G, VGA_B} = 24'h3c3c3c;
                8'h35 : {VGA_R, VGA_G, VGA_B} = 24'h303030;
                8'h36 : {VGA_R, VGA_G, VGA_B} = 24'h1b1b1b;
                8'h37 : {VGA_R, VGA_G, VGA_B} = 24'h131313;
                8'h38 : {VGA_R, VGA_G, VGA_B} = 24'h0c0c0c;
                8'h39 : {VGA_R, VGA_G, VGA_B} = 24'h090909;
                8'h3a : {VGA_R, VGA_G, VGA_B} = 24'h6d6d6d;
                8'h3b : {VGA_R, VGA_G, VGA_B} = 24'h868686;
                8'h3c : {VGA_R, VGA_G, VGA_B} = 24'h8f8f8f;
                8'h3d : {VGA_R, VGA_G, VGA_B} = 24'h959595;
                8'h3e : {VGA_R, VGA_G, VGA_B} = 24'h989898;
                8'h3f : {VGA_R, VGA_G, VGA_B} = 24'h949494;
```

```
8'h40 : {VGA_R, VGA_G, VGA_B} = 24'h8d8d8d;
8'h41 : {VGA_R, VGA_G, VGA_B} = 24'h858585;
8'h42 : {VGA_R, VGA_G, VGA_B} = 24'h797979;
8'h43 : {VGA_R, VGA_G, VGA_B} = 24'h6c6c6c;
8'h44 : {VGA_R, VGA_G, VGA_B} = 24'h5f5f5f;
8'h45 : {VGA_R, VGA_G, VGA_B} = 24'h505050;
8'h46 : {VGA_R, VGA_G, VGA_B} = 24'h353535;
8'h47 : {VGA_R, VGA_G, VGA_B} = 24'h292929;
8'h48 : {VGA_R, VGA_G, VGA_B} = 24'h1e1e1e;
8'h49 : {VGA_R, VGA_G, VGA_B} = 24'h151515;
8'h4a : {VGA_R, VGA_G, VGA_B} = 24'h0e0e0e;
8'h4b : {VGA_R, VGA_G, VGA_B} = 24'h080808;
8'h4c : {VGA_R, VGA_G, VGA_B} = 24'h818181;
8'h4d : {VGA_R, VGA_G, VGA_B} = 24'h9e9e9e;
8'h4e : {VGA_R, VGA_G, VGA_B} = 24'ha1a1a1;
8'h4f : {VGA_R, VGA_G, VGA_B} = 24'ha0a0a0;
8'h50 : {VGA_R, VGA_G, VGA_B} = 24'h9c9c9c;
8'h51 : {VGA_R, VGA_G, VGA_B} = 24'h8c8c8c;
8'h52 : {VGA_R, VGA_G, VGA_B} = 24'h7f7f7f;
8'h53 : {VGA_R, VGA_G, VGA_B} = 24'h737373;
8'h54 : {VGA_R, VGA_G, VGA_B} = 24'h555555;
8'h55 : {VGA_R, VGA_G, VGA_B} = 24'h464646;
8'h56 : {VGA_R, VGA_G, VGA_B} = 24'h2b2b2b;
8'h57 : {VGA_R, VGA_G, VGA_B} = 24'h171717;
8'h58 : {VGA_R, VGA_G, VGA_B} = 24'h101010;
8'h59 : {VGA_R, VGA_G, VGA_B} = 24'h0a0a0a;
8'h5a : {VGA_R, VGA_G, VGA_B} = 24'h060606;
8'h5b : {VGA_R, VGA_G, VGA_B} = 24'h5d5d5d;
8'h5c : {VGA_R, VGA_G, VGA_B} = 24'ha4a4a4;
8'h5d : {VGA_R, VGA_G, VGA_B} = 24'ha7a7a7;
8'h5e : {VGA_R, VGA_G, VGA_B} = 24'ha6a6a6;
8'h5f : {VGA_R, VGA_G, VGA_B} = 24'ha2a2a2;
8'h60 : {VGA_R, VGA_G, VGA_B} = 24'h9b9b9b;
8'h61 : {VGA_R, VGA_G, VGA_B} = 24'h909090;
8'h62 : {VGA_R, VGA_G, VGA_B} = 24'h838383;
8'h63 : {VGA_R, VGA_G, VGA_B} = 24'h484848;
8'h64 : {VGA_R, VGA_G, VGA_B} = 24'h3a3a3a;
8'h65 : {VGA_R, VGA_G, VGA_B} = 24'h2d2d2d;
8'h66 : {VGA_R, VGA_G, VGA_B} = 24'h222222;
8'h67 : {VGA_R, VGA_G, VGA_B} = 24'h050505;
8'h68 : {VGA_R, VGA_G, VGA_B} = 24'h232323;
8'h69 : {VGA_R, VGA_G, VGA_B} = 24'ha9a9a9;
8'h6a : {VGA_R, VGA_G, VGA_B} = 24'ha8a8a8;
8'h6b : {VGA_R, VGA_G, VGA_B} = 24'ha3a3a3;
8'h6c : {VGA_R, VGA_G, VGA_B} = 24'h919191;
8'h6d : {VGA_R, VGA_G, VGA_B} = 24'h848484;
8'h6e : {VGA_R, VGA_G, VGA_B} = 24'h686868;
8'h6f : {VGA_R, VGA_G, VGA_B} = 24'h020202;
8'h70 : {VGA_R, VGA_G, VGA_B} = 24'h757575;
8'h71 : {VGA_R, VGA_G, VGA_B} = 24'h9d9d9d;
8'h72 : {VGA_R, VGA_G, VGA_B} = 24'h969696;
8'h73 : {VGA_R, VGA_G, VGA_B} = 24'h717171;
8'h74 : {VGA_R, VGA_G, VGA_B} = 24'h626262;
8'h75 : {VGA_R, VGA_G, VGA_B} = 24'h0f0f0f;
```

```
                8'h76 : {VGA_R, VGA_G, VGA_B} = 24'h787878;
                8'h77 : {VGA_R, VGA_G, VGA_B} = 24'h404040;
                8'h78 : {VGA_R, VGA_G, VGA_B} = 24'h333333;
                8'h79 : {VGA_R, VGA_G, VGA_B} = 24'h1d1d1d;
                8'h7a : {VGA_R, VGA_G, VGA_B} = 24'h030303;
                8'h7b : {VGA_R, VGA_G, VGA_B} = 24'h7b7b7b;
                8'h7c : {VGA_R, VGA_G, VGA_B} = 24'h636363;
                8'h7d : {VGA_R, VGA_G, VGA_B} = 24'h2e2e2e;
                8'h7e : {VGA_R, VGA_G, VGA_B} = 24'h242424;
                8'h7f : {VGA_R, VGA_G, VGA_B} = 24'h1a1a1a;
                8'h80 : {VGA_R, VGA_G, VGA_B} = 24'h121212;
                8'h81 : {VGA_R, VGA_G, VGA_B} = 24'h040404;
                8'h82 : {VGA_R, VGA_G, VGA_B} = 24'h000000;
                8'h83 : {VGA_R, VGA_G, VGA_B} = 24'h3e3e3e;
                8'h84 : {VGA_R, VGA_G, VGA_B} = 24'h808080;
                8'h85 : {VGA_R, VGA_G, VGA_B} = 24'h595959;
                8'h86 : {VGA_R, VGA_G, VGA_B} = 24'h4d4d4d;
                8'h87 : {VGA_R, VGA_G, VGA_B} = 24'h010101;
                8'h88 : {VGA_R, VGA_G, VGA_B} = 24'h313131;
                8'h89 : {VGA_R, VGA_G, VGA_B} = 24'h262626;
                8'h8a : {VGA_R, VGA_G, VGA_B} = 24'h0b0b0b;
                8'h8b : {VGA_R, VGA_G, VGA_B} = 24'h070707;
                8'h8c : {VGA_R, VGA_G, VGA_B} = 24'h474747;
                8'h8d : {VGA_R, VGA_G, VGA_B} = 24'h191919;
                default:{VGA_R, VGA_G, VGA_B} = 24'hffffff;
        endcase
end
// Draw board
else if(is_board_area) begin
        case(bg_palette)
                4'h0 : {VGA_R, VGA_G, VGA_B} = 24'h816951;
                4'h1 : {VGA_R, VGA_G, VGA_B} = 24'ha0866d;
                4'h2 : {VGA_R, VGA_G, VGA_B} = 24'hc9a081;
                4'h3 : {VGA_R, VGA_G, VGA_B} = 24'hc6ab91;
                4'h4 : {VGA_R, VGA_G, VGA_B} = 24'hd3b69a;
                4'h5 : {VGA_R, VGA_G, VGA_B} = 24'hd9bb9d;
                4'h6 : {VGA_R, VGA_G, VGA_B} = 24'hdcc0a5;
                4'h7 : {VGA_R, VGA_G, VGA_B} = 24'hd7b694;
                4'h8 : {VGA_R, VGA_G, VGA_B} = 24'h513f29;
                4'h9 : {VGA_R, VGA_G, VGA_B} = 24'hd4b191;
                4'ha : {VGA_R, VGA_G, VGA_B} = 24'hd1ac89;
                4'hb : {VGA_R, VGA_G, VGA_B} = 24'hd0a988;
                4'hc : {VGA_R, VGA_G, VGA_B} = 24'hd2ae8b;
                4'hd : {VGA_R, VGA_G, VGA_B} = 24'hcca383;
                4'he : {VGA_R, VGA_G, VGA_B} = 24'hcfa585;
                default:{VGA_R, VGA_G, VGA_B} = 24'hffffff;
        endcase
end
/************** Player UI Area *************/
// Player UI Division line
else if (division_line) begin
        {VGA_R, VGA_G, VGA_B} = 24'h000000;  // Black color
end
// Draw Player turn mark
```

```
else if(is_turn_area && game_started) begin
	{VGA_R, VGA_G, VGA_B} = 24'ha8eb34;
end
// Draw profile image
else if(is_p1_profile_area)
	/* Panda palette */
	case(p1_profile_palette)
		4'h0 : {VGA_R, VGA_G, VGA_B} = 24'h111111;
		4'h1 : {VGA_R, VGA_G, VGA_B} = 24'h49484e;
		4'h2 : {VGA_R, VGA_G, VGA_B} = 24'hfefbf5;
		4'h3 : {VGA_R, VGA_G, VGA_B} = 24'h616160;
		4'h4 : {VGA_R, VGA_G, VGA_B} = 24'hfdede4;
		4'h5 : {VGA_R, VGA_G, VGA_B} = 24'h15340e;
		4'h6 : {VGA_R, VGA_G, VGA_B} = 24'hadadb0;
		4'h7 : {VGA_R, VGA_G, VGA_B} = 24'hdacbbe;
		4'h8 : {VGA_R, VGA_G, VGA_B} = 24'h838588;
		4'h9 : {VGA_R, VGA_G, VGA_B} = 24'h00833b;
		4'ha : {VGA_R, VGA_G, VGA_B} = 24'h004719;
		default:{VGA_R, VGA_G, VGA_B} = 24'hffffff;
	endcase
else if(is_p2_profile_area && game_started) begin
	if(p2_profile_cat)
	/* Cat palette */
	case(p2_profile_palette)
		4'h0 : {VGA_R, VGA_G, VGA_B} = 24'hded6c6;
		4'h1 : {VGA_R, VGA_G, VGA_B} = 24'hd8cdb9;
		4'h2 : {VGA_R, VGA_G, VGA_B} = 24'he2bd85;
		4'h3 : {VGA_R, VGA_G, VGA_B} = 24'hd2a25a;
		4'h4 : {VGA_R, VGA_G, VGA_B} = 24'hd69d4a;
		4'h5 : {VGA_R, VGA_G, VGA_B} = 24'hd59833;
		4'h6 : {VGA_R, VGA_G, VGA_B} = 24'hcb9d50;
		4'h7 : {VGA_R, VGA_G, VGA_B} = 24'hbd9a75;
		4'h8 : {VGA_R, VGA_G, VGA_B} = 24'hd69d3f;
		4'h9 : {VGA_R, VGA_G, VGA_B} = 24'hc3904d;
		4'ha : {VGA_R, VGA_G, VGA_B} = 24'hc08748;
		4'hb : {VGA_R, VGA_G, VGA_B} = 24'hb87f36;
		4'hc : {VGA_R, VGA_G, VGA_B} = 24'hd86353;
		4'hd : {VGA_R, VGA_G, VGA_B} = 24'h985321;
		4'he : {VGA_R, VGA_G, VGA_B} = 24'h58402e;
		4'hf : {VGA_R, VGA_G, VGA_B} = 24'h1d1715;
	endcase
	else
	/* Dog palette */
	case(p2_profile_palette)
		4'h0 : {VGA_R, VGA_G, VGA_B} = 24'he1d1bd;
		4'h1 : {VGA_R, VGA_G, VGA_B} = 24'hdfb771;
		4'h2 : {VGA_R, VGA_G, VGA_B} = 24'hd7ad64;
		4'h3 : {VGA_R, VGA_G, VGA_B} = 24'hddbd83;
		4'h4 : {VGA_R, VGA_G, VGA_B} = 24'hcc9b53;
		4'h5 : {VGA_R, VGA_G, VGA_B} = 24'hc28b3e;
		4'h6 : {VGA_R, VGA_G, VGA_B} = 24'hb59770;
		4'h7 : {VGA_R, VGA_G, VGA_B} = 24'ha97230;
		4'h8 : {VGA_R, VGA_G, VGA_B} = 24'ha5835b;
		4'h9 : {VGA_R, VGA_G, VGA_B} = 24'h967551;
```

```
                                    4'ha : {VGA_R, VGA_G, VGA_B} = 24'h8d6c49;
                                    4'hb : {VGA_R, VGA_G, VGA_B} = 24'h4c86b4;
                                    4'hc : {VGA_R, VGA_G, VGA_B} = 24'h935a1c;
                                    4'hd : {VGA_R, VGA_G, VGA_B} = 24'h715944;
                                    4'he : {VGA_R, VGA_G, VGA_B} = 24'h67462a;
                                    4'hf : {VGA_R, VGA_G, VGA_B} = 24'h191412;
                              endcase
                        end
                        // Draw in-game options
                        else if(is_option_area) begin
                              // Option background
                              {VGA_R, VGA_G, VGA_B} = 24'h00DDAA;
                              // Draw ingame_options message
                              if(msg_display_ingame_options && font_val[font_pix_idx]) begin
                                    // Font selected
                                    if(cur_msg_selected)
                                          {VGA_R, VGA_G, VGA_B} = 24'h00ffff;
                                    else
                                          {VGA_R, VGA_G, VGA_B} = 24'h000000;
                              end
                        end
                        // Player Profile Message & Background
                        else if(is_player1_area || is_player2_area) begin
                              // Profile background
                              {VGA_R, VGA_G, VGA_B} = 24'h96968C;
                              // Draw ingame_players message
                              if(msg_display_ingame_players && font_val[font_pix_idx]) begin

                                    {VGA_R, VGA_G, VGA_B} = 24'h000000;
                              end
                        end
                  end
            end
      end
endmodule

module vga_counters(
input  logic        clk50, reset,
output logic [10:0] hcount,  // hcount[10:1] is pixel columnvga_piece_painter
output logic [9:0]  vcount,  // vcount[9:0] is pixel row
output logic        VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n);

      /*
      * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
      *
      * HCOUNT 1599 0             1279       1599 0
      *             _______________              ________
      * ___________|    Video      |____________|  Video
      *
      *
      * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
      *       _______________________      _____________
      * |____|       VGA_HS          |____|
      */
```

```
// Parameters for hcount
parameter    HACTIVE      = 11'd 1280,
                  HFRONT_PORCH = 11'd 32,
                  HSYNC        = 11'd 192,
                  HBACK_PORCH  = 11'd 96,
                  HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC +
                                       HBACK_PORCH; // 1600

// Parameters for vcount
parameter    VACTIVE      = 10'd 480,
                  VFRONT_PORCH = 10'd 10,
                  VSYNC        = 10'd 2,
                  VBACK_PORCH  = 10'd 33,
                  VTOTAL       = VACTIVE + VFRONT_PORCH + VSYNC +
                                       VBACK_PORCH; // 525

logic endOfLine;

always_ff @(posedge clk50 or posedge reset)
       if (reset)          hcount <= 0;
       else if (endOfLine) hcount <= 0;
       else                hcount <= hcount + 11'd 1;

assign endOfLine = hcount == HTOTAL - 1;

logic endOfField;

always_ff @(posedge clk50 or posedge reset)
       if (reset)                 vcount <= 0;
       else if (endOfLine)
              if (endOfField) vcount <= 0;
              else                vcount <= vcount + 10'd 1;

assign endOfField = vcount == VTOTAL - 1;

// Horizontal sync: from 0x520 to 0x5DF (0x57F)
// 101 0010 0000 to 101 1101 1111
assign VGA_HS = !( (hcount[10:8] == 3'b101) &
                          !(hcount[7:5] == 3'b111));
assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);

assign VGA_SYNC_n = 1'b0; // For putting sync on the green signal; unused

// Horizontal active: 0 to 1279     Vertical active: 0 to 479
// 101 0000 0000  1280            01 1110 0000  480
// 110 0011 1111  1599            10 0000 1100  524
assign VGA_BLANK_n = !( hcount[10] & (hcount[9] | hcount[8]) ) &
                                  !( vcount[9] | (vcount[8:5] == 4'b1111) );

/* VGA_CLK is 25 MHz
*             __    __    __
* clk50    __|  |__|  |__|
*
*             _____       __
```

```
       * hcount[0]__|     |_____|
       */
       assign VGA_CLK = hcount[0]; // 25 MHz clock: rising edge sensitive

endmodule
```

5.2 vga_kmod/vga_gomoku.h

```
C/C++
#ifndef _vga_gomoku_H
#define _vga_gomoku_H

#include <linux/ioctl.h>
#include <stddef.h>
#include <linux/types.h>

typedef struct {
       uint16_t params[8];
} vga_gomoku_arg_t;


#define VGA_GOMOKU_MAGIC 'q'
#define VGA_DRIVER_FILENAME "/dev/vga_ball"

/* ioctls and their arguments */
#define VGA_GOMOKU_WRITE _IOW(VGA_GOMOKU_MAGIC, 1, vga_gomoku_arg_t *)
#define VGA_GOMOKU_READ  _IOR(VGA_GOMOKU_MAGIC, 2, vga_gomoku_arg_t *)

#endif
```

5.3 vga_kmod/vga_gomoku.c

```
C/C++
/* * Device driver for the VGA video generator
 *
 * A Platform device implemented using the misc subsystem
 *
 * Stephen A. Edwards
 * Columbia University
 *
 * References:
 * Linux source: Documentation/driver-model/platform.txt
 *               drivers/misc/arm-charlcd.c
 * http://www.linuxforu.com/tag/linux-device-drivers/
 * http://free-electrons.com/docs/
```

```
 *
 * "make" to build
 * insmod vga_gomoku.ko
 *
 * Check code style with
 * checkpatch.pl --file --no-tree vga_gomoku.c
 */

#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "vga_gomoku.h"

#define DRIVER_NAME "vga_ball"

/* Device registers */
// #define CIR_RED(x) (x)
// #define CIR_GREEN(x) ((x)+1)
// #define CIR_BLUE(x) ((x)+2)
// #define CIR_XH(x) ((x)+3)
// #define CIR_XL(x) ((x)+4)
// #define CIR_YH(x) ((x)+5)
// #define CIR_YL(x) ((x)+6)
// #define CIR_R(x) ((x)+7)
// #define BG_RED(x) ((x)+8)
// #define BG_GREEN(x) ((x)+9)
// #define BG_BLUE(x) ((x)+10)

/*
 * Information about our device
 */
struct vga_gomoku_dev {
	struct resource res; /* Resource: our registers */
	void __iomem *virtbase; /* Where registers can be accessed in memory */
} dev;

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
static void write_data(vga_gomoku_arg_t *arg)
{
	int i;
	for(i=0;i<8;++i)
```

```
		iowrite16(arg->params[i], dev.virtbase+i*2);
}

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */
static long vga_gomoku_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
	vga_gomoku_arg_t vla;

	switch (cmd) {
	case VGA_GOMOKU_WRITE:
		if (copy_from_user(&vla, (vga_gomoku_arg_t *) arg,
				   sizeof(vga_gomoku_arg_t)))
			return -EACCES;
		write_data(&vla);
		// write_circle_color(&vla.c_color);
		// write_circle(&vla.circle);
		// write_bg_color(&vla.bg_color);
		break;

	// case vga_gomoku_READ:
	//     vla.c_color = dev.c_color;
	//     vla.circle = dev.circle;
	//     vla.bg_color = dev.bg_color;
	//     if (copy_to_user((vga_gomoku_arg_t *) arg, &vla,
	//                      sizeof(vga_gomoku_arg_t)))
	//         return -EACCES;
	//     break;
	default:
		return -EINVAL;
	}

	return 0;
}

/* The operations our device knows how to do */
static const struct file_operations vga_gomoku_fops = {
	.owner		= THIS_MODULE,
	.unlocked_ioctl = vga_gomoku_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice vga_gomoku_misc_device = {
	.minor		= MISC_DYNAMIC_MINOR,
	.name		= DRIVER_NAME,
	.fops		= &vga_gomoku_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
```

```
 */
static int __init vga_gomoku_probe(struct platform_device *pdev)
{
	int ret;

	/* Register ourselves as a misc device: creates /dev/vga_gomoku */
	ret = misc_register(&vga_gomoku_misc_device);

	/* Get the address of our registers from the device tree */
	ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
	if (ret) {
		ret = -ENOENT;
		goto out_deregister;
	}

	/* Make sure we can use these registers */
	if (request_mem_region(dev.res.start, resource_size(&dev.res),
			       DRIVER_NAME) == NULL) {
		ret = -EBUSY;
		goto out_deregister;
	}

	/* Arrange access to our registers */
	dev.virtbase = of_iomap(pdev->dev.of_node, 0);
	if (dev.virtbase == NULL) {
		ret = -ENOMEM;
		goto out_release_mem_region;
	}

	return 0;

out_release_mem_region:
	release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
	misc_deregister(&vga_gomoku_misc_device);
	return ret;
}

/* Clean-up code: release resources */
static int vga_gomoku_remove(struct platform_device *pdev)
{
	iounmap(dev.virtbase);
	release_mem_region(dev.res.start, resource_size(&dev.res));
	misc_deregister(&vga_gomoku_misc_device);
	return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id vga_gomoku_of_match[] = {
	{ .compatible = "csee4840,vga_ball-1.0" },
	{},
};
MODULE_DEVICE_TABLE(of, vga_gomoku_of_match);
```

```
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver vga_gomoku_driver = {
	.driver	= {
		.name  = DRIVER_NAME,
		.owner = THIS_MODULE,
		.of_match_table = of_match_ptr(vga_gomoku_of_match),
	},
	.remove	= __exit_p(vga_gomoku_remove),
};

/* Called when the module is loaded: set things up */
static int __init vga_gomoku_init(void)
{
	pr_info(DRIVER_NAME ": init\n");
	return platform_driver_probe(&vga_gomoku_driver, vga_gomoku_probe);
}

/* Calball when the module is unloaded: release resources */
static void __exit vga_gomoku_exit(void)
{
	platform_driver_unregister(&vga_gomoku_driver);
	pr_info(DRIVER_NAME ": exit\n");
}

module_init(vga_gomoku_init);
module_exit(vga_gomoku_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
MODULE_DESCRIPTION("VGA ball driver");
```

5.4 main.cpp

```C/C++
#include "game/game.h"
#include "game/gomoku.h"
#include "display/display.h"
#include "input/touchpad.h"
#include "input/xboxcont.h"
#include "display/message_group_init.h"
#include <cstdio>
#include <vector>

int main()
{
    Gomoku game(1);
    GameMenu menu(&game);
	GMKDisplayMessageGroup msg_group;
	GMKDisplay display;
```

```
        XboxController controller;
        Touchpad touchpad;

        init_message_group(msg_group);
        display.set_message_group(&msg_group);
        menu.setMessageGroup(&msg_group);

        if(!display.open_display()){
                perror("Open VGA display");
        }
        else {
                touchpad.set_display(&display);
                menu.setDisplay(&display);
        }

        if(!touchpad.open_device()){
                printf("Open touchpad device failed.\n");
        }
        else{
                touchpad.set_input_handler(&menu);
                touchpad.create_handling_thread();
        }

        if(!controller.open_device()){
                printf("Open controller device failed.\n");
        }
        else {
                controller.set_input_handler(&menu);
                controller.create_handling_thread();
        }

        //while (1);
    menu.gameStart();
        controller.stop_handling_thread();
        touchpad.stop_handling_thread();
        controller.close_device();
        touchpad.close_device();
    return 0;
}
```

5.5 game/game.h

```
C/C++
#ifndef _MAIN_MENU_HH
#define _MAIN_MENU_HH

#include "gomoku.h"
#include "gomokuAI.h"
#include "players.h"
#include "../display/display.h"
#include "../input/input.h"
```

```
#include <cstdint>

class GameMenu : public InputEventHandler{
private:
    Gomoku *game;
       GMKDisplay *display=NULL;
    int currentMode;
       uint16_t board_x, board_y;
       uint16_t selected_msg_index;
       GMKDisplayMessageGroup *msg_group;
       bool block_input=false;
       bool cancel = false;
       bool waiting = false;

public:
    GameMenu(Gomoku *game) : game(game), currentMode(0) {}

    void displayMode();          // Display mode options.
    int  getChoice();            // Get user's choice.

    // Different modes
    void PvPMode();              // Player vs player. Integrated with pvp.cpp. Mode code : 1
    void PvEMode(int player);    // Player vs AI. Integrated with pve.cpp, evp.cpp. Mode code : 2, 3
    void networkMode(bool server);      // Players play over the network. Intergrated with network.cpp.
Mode code : 4

       void handle_input_press(InputEvent event) override;
       void handle_input_release(InputEvent event) override {}

    void gameStart();            // Game start.

       void showMenu();
       void showBoard(bool top_first=true);
       void playAnime();

       void setDisplay(GMKDisplay *display);
       void setMessageGroup(GMKDisplayMessageGroup *group){this->msg_group=group;}

       bool wait_for_confirm();         // Wait for confirm message. True means confirm, False means
cancel.
};

#endif
```

5.6 game/game.cpp

```
C/C++
#include "game.h"
#include <cassert>
#include <cstdint>
#include <cstdio>
```

```
#include <string>
#include <type_traits>
#include <unistd.h>
#include "../network/network.h"

void* findPlayersWait(void *arg)
{
	sleep(3);
	return NULL;
}

void GameMenu::setDisplay(GMKDisplay *display)
{
	if(game)
		game->set_display(display);
	this->display=display;
}

void GameMenu::showMenu()
{
	if(display){
		display->show_game_result(0,false);
		display->show_menu();
	}
	else{
		msg_group->update_group_visibility(0, false);
		msg_group->update_group_visibility(1, true);
		msg_group->update_group_visibility(2, false);
		msg_group->update_group_visibility(3, false);
	}
	selected_msg_index = msg_group->first_selectable_message();
	if(display)
		display->update_select(selected_msg_index);
	//msg_group->display_selectable();
}

void GameMenu::showBoard(bool top_first)
{
	if(display){
		display->show_game_result(0,false);
		display->show_confirm_message(false);
		display->clear_board();
		display->set_player_piece(top_first,false);
		display->set_turn_mark(top_first,false);
		display->show_board(false);
	}
	else{
		msg_group->update_group_visibility(0, true);
		msg_group->update_group_visibility(1, false);
		msg_group->update_group_visibility(2, true);
		msg_group->update_group_visibility(3, true);
	}
	selected_msg_index = 0x7700;
	board_x = 8;
```

```
	board_y = 8;
	if(display)
		display->update_select(selected_msg_index);
	//msg_group->display_selectable();
}

void GameMenu::displayMode()
{
    std::cout << "Select Game Mode:\n";
	for(auto select: msg_group->selectable){
		printf("%d. %s\n",select.index,select.content.c_str());
	}
}

int GameMenu::getChoice()
{
    int choice;
    std::cin >> choice;
    return choice;
}

void GameMenu::gameStart()
{
    while (1) {
		showMenu();
        std::cout << "\n\nChoose a mode to play. " <<std::endl;
        displayMode();
        uint16_t mode = wait_for_command();
		srand(time(NULL));
        switch(mode) {
			case 6: // EXIT
				return;
            case 3: // PVE
				if((rand()%4)>=2)
					PvEMode(1);
				else
					PvEMode(2);
                break;
            case 2: // PVP
                PvPMode();
                break;
            case 4: // CREATE LAN
                networkMode(1);
                break;
            case 5: // JOIN LAN
                networkMode(0);
                break;
            default:
                std::cout << "Invalid mode. Type again\n";
        }
    }
}

void GameMenu::PvEMode(int player)
```

```
{
    /*
     * Player plays : black (if player == 1) , white (if player == 2)
     */
    game->mode = 1;
    Player p1(game, player);
    GomokuAI ai(game, 1);
    int aiFirst = (player == 1) ? 0 : 1;
    string playerColor = (player == 1) ? "black" : "white";
    pair<int, int> aiMove = make_pair(-1, -1);  // ai moves

       showBoard(!aiFirst);
       if(display)
             display->update_p2_profile(0);
    std::cout << "\n\nGame started. You Play as " << playerColor << endl;
    std::cout << "\nType in (board_x, board_y) to make a move. (0, 0) to regret a move.\
    (-1, -1) to return to main menu. " << playerColor << endl;

    if (aiFirst) {
        aiMove = ai.findBestMove();
        ai.makeMove(aiMove);
    }
    game->displayBoard();

       while (1) {
        uint16_t command = wait_for_command();
             switch (command) {
             case 10: // Quit
                   // should add display message                  //     continue;
                   if(display)
                         display->show_confirm_message();
                   selected_msg_index=msg_group->first_selectable_message();
                   if(!wait_for_confirm()){
                         if(display)
                               display->show_confirm_message(false);
                         selected_msg_index=msg_group->first_selectable_message();
                         continue;
                   }
                   std::cout << "Returning to main menu ..." << std::endl;
            game->resetGame();  // NOTE: modified from clearBoard method in class Gomoku.
            return;
             case 0: // Move
                   break;
             case 7: // Regret
            if (!game->regret_move()) {
                std::cout << "regret a move, please continue" << endl;
                game->displayBoard();
            }
            continue;
             case 9:
                   // Resign without confirm
                   game->displayBoard();
            std::cout << "You lose!" << endl;
                   game->end_game(!(player==1));
```

```
                if(display)
                        display->show_game_result(1);
                selected_msg_index = msg_group->first_selectable_message();
                // Wait for confirm command, quit to main menu.
                while(wait_for_confirm()){
                        game->resetGame();
                        return;
                }
                continue;
        case 8:{
                std::pair<int, int> ai_hint = ai.findBestMove();
                if(display)
                        display->show_hint(ai_hint.first, ai_hint.second);
                // std::cout << "Hint: " << hint.first + 1 << ", " << hint.second + 1 << endl;
                continue;
        }
        default:
                printf("Invalid command!\n");
                continue;
        }

    // NOTE: (0, 0) now refers to regretting a move, (-1, -1) stands for returning to main menu.
         if (p1.makeMove(make_pair(board_x - 1, board_y- 1))) {
        std::cout << "Invalid move!" << '\n' << endl;
        continue;
    }
         board_x=-1;
         board_y=-1;

    if (game->state == 1) {
        game->displayBoard();
                playAnime();
        std::cout << "You win!" << endl;
                if(display)
                        display->show_game_result(0);
                selected_msg_index = msg_group->first_selectable_message();
                // Wait for confirm command, quit to main menu.
                while(wait_for_confirm()){
                        game->resetGame();
                        return;
                }
    }
    game->displayBoard();

    // AI makes a move.
         assert(game->current_player+player==3);
    aiMove = ai.findBestMove();
    ai.makeMove(aiMove);

    std::cout << "You made a move at " << board_x << ", " << board_y << ", " << "AI made a move at
"
    << aiMove.first + 1 << ", " << aiMove.second + 1 << "\n" << endl;

    if (game->state == 1) {
```

```
            game->displayBoard();
                    playAnime();
            std::cout << "You lose!" << endl;
                    if(display)
                           display->show_game_result(1);
                    selected_msg_index = msg_group->first_selectable_message();
                    // Wait for confirm command, quit to main menu.
                    bool confirm = wait_for_confirm();
                    if(confirm){
                           game->resetGame();
                           return;
                    }
        }
        game->displayBoard();
    }
}

void GameMenu::PvPMode()
{
    game->mode = 0;
    Player p1(game, 1);
    Player p2(game, 2);

       showBoard();
       if(display)
              display->update_p2_profile(1);
    cout << "\n\nGame started. " << endl;
    game->displayBoard();

    while (1) {
        uint16_t command = wait_for_command();
             switch (command) {
             case 10: // Quit without confirm
                    std::cout << "Returning to main menu ..." << std::endl;
            game->resetGame();  // NOTE: modified from clearBoard method in class Gomoku.
            return;
             case 0: // Move
                    break;
             case 7: // Regret
            if (!game->regret_move()) {
                std::cout << ((game->current_player == 2) ? "Black" : "White") <<" regrets a move,
please continue" << endl;
                game->displayBoard();
            }
            continue;
             case 8:
                    // HINT. There should not be a hint in PVP mode.
                    printf("Tip not implemented.\n");
                    continue;
             default:
                    printf("Invalid command!\n");
                    continue;
             }
```

```
            std::string str_player = "White ";
            Player *p = &p2;
            if(game->current_player == 2){
                  str_player = "Black ";
                  p=&p1;
            }
        if (p->makeMove(make_pair(board_x - 1, board_y- 1))) {
            cout << "Invalid move!" << '\n' << endl;
            continue;
        }
        cout << str_player << "made a move at " << board_x << ", " << board_y << endl;
             board_x=-1;
             board_y=-1;

        if (game->state == 1) {
            game->displayBoard();
                  playAnime();
                  if(display)
                        // current_player=1 -> black -> p1 -> result=2
                        // current_player=2 -> white -> p2 -> result=3
                        display->show_game_result(game->current_player+1);
                  selected_msg_index=msg_group->first_selectable_message();
            cout << str_player << "wins! Returning to main menu. " << endl;
                  // Wait for confirm command, quit to main menu.
                  bool confirm = wait_for_confirm();
                  if(confirm){
                        game->resetGame();
                        return;
                  }
        }
        game->displayBoard();
    }
}

void GameMenu::networkMode(bool server)
{
    GomokuAI ai(game, 1); // AI is here to provide hints.
       std::pair<int, int> ai_hint;
       game->mode=1;

    if (server) {
             GMKServer server(game);
             server.set_cancel(&cancel);
             server.set_event_handler(this);

             if(!server.create())
                   return;
             showBoard();
             if(display)
                   display->update_p2_profile(2);
             waiting = true;
             if(!server.wait_for_player()){
                   printf("Wait for player failed.\n");
                   game->resetGame();
```

```
                waiting = false;
                command_type_=0;
                return;
            }
            bool top_first = server.start_game();
            showBoard(top_first);
            if(display)
                display->update_p2_profile(1);
            while(server.check_game_result()<0){
                uint16_t command = wait_for_command();
                switch (command) {
                case 10: // Quit
                    if(display)
                        display->show_confirm_message();
                    selected_msg_index=msg_group->first_selectable_message();
                    if(!wait_for_confirm())
                        continue;
                    server.resign();
                    std::cout << "Returning to main menu ..." << std::endl;
                    game->resetGame();  // NOTE: modified from clearBoard method in class Gomoku.
                    return;
                case 0: // Move
                    // Make move
                    if (!server.make_move(board_x - 1, board_y- 1)){
                        printf("Invalid move!\n");
                    }
                    board_x=-1;
                    board_y=-1;
                    break;
                case 7: // Regret
                    // if (!game->regret_move()) {
                    //     std::cout << ((game->current_player == 2) ? "Black" : "White") <<"
regrets a move, please continue" << endl;
                    //     game->displayBoard();
                    // }
                    printf("Regret disabled in net work mode.\n");
                    break;
                case 9:
                    // Resign without confirm
                    server.resign();
                    break;
                case 8:
                    // HINT
                    ai_hint = ai.findBestMove();
                    // std::cout << "Hint: " << hint.first + 1 << ", " << hint.second + 1 <<
endl;
                    if(display)
                        display->show_hint(ai_hint.first, ai_hint.second);
                    break;
                default:
                    printf("Invalid command!\n");
                    break;
                }
                // Remote player resign check, end the loop
```

```
                if(game->state==1)
                        break;
        }

        // Connection closed
        if(server.check_game_result()==3){
                printf("Connection lost!\n");
                // Probably print you win.
                if(display)
                        display->show_game_result(0);
                selected_msg_index=msg_group->first_selectable_message();
        }
        else if(server.check_game_result()==1) {
                printf("You win!\n");
                playAnime();
                if(display)
                        display->show_game_result(0);
                selected_msg_index=msg_group->first_selectable_message();
        }
        else{
                printf("You lose!\n");
                playAnime();
                if(display)
                        display->show_game_result(1);
                selected_msg_index=msg_group->first_selectable_message();
        }

        // Wait for confirm command, quit to main menu.
        bool confirm = wait_for_confirm();
        if(confirm){
                game->resetGame();
                return;
        }
}
else {
        GMKClient client(game);
        client.set_event_handler(this);
        client.set_cancel(&cancel);
        client.send_server_discover();

        string null;
        string server_status[3] = {"Down","Gaming","Ready"};
        GMKServerInfo info;
        bool found=false;

        printf("Discovering server...\n");
        if(display)
                display->show_scanning_message();
        // Wait for 3 seconds but without blocking the main thread.
        client.wait_for_scan();
        display->show_scanning_message(false);

        if(client.get_server_list().empty()){
                printf("No server discovered!\n");
```

```
                return;
            }
            printf("Server List\n%-15s\t%-4s\t%s\n","IP address","Port","Status");
            for(auto server:client.get_server_list()){

printf("%s\t%-4u\t%-6s\n",server.address.c_str(),server.port,server_status[server.status].c_str());
                if(server.status==2){
                    info = server;
                    found = true;
                }
            }
            if(!found){
                printf("No available server found!\n");
                return;
            }
            printf("Connecting %s:%u\n",info.address.c_str(),info.port);
            if(!client.connect_to(info))
                return;
            //client.connect_to("128.59.65.165");

            /* Connected. Show board page. */
            showBoard();
            if(display)
                display->update_p2_profile(1);
            while(client.check_game_result()<0){
                uint16_t command = wait_for_command();
                switch (command) {
                case 10: // Quit confirm
                    if(display)
                        display->show_confirm_message();
                    selected_msg_index=msg_group->first_selectable_message();
                    if(!wait_for_confirm())
                        continue;
                    std::cout << "Returning to main menu ..." << std::endl;
                    game->resetGame();  // NOTE: modified from clearBoard method in class Gomoku.
                    showMenu();
                    return;
                case 0: // Move or remote resigns
                    if (!client.make_move(board_x - 1, board_y - 1)){
                    }
                    board_x=-1;
                    board_y=-1;
                    break;
                case 7: // Regret
                    // if (!game->regret_move()) {
                    //     std::cout << ((game->current_player == 2) ? "Black" : "White") <<"
regrets a move, please continue" << endl;
                    //     game->displayBoard();
                    // }
                    printf("Regret disabled in net work mode.\n");
                    continue;
                case 9:
                    // Resign without confirm
                    client.resign();
```

```
                        break;
                case 8:
                        // HINT
                        ai_hint = ai.findBestMove();
                        // std::cout << "Hint: " << hint.first + 1 << ", " << hint.second + 1 <<
endl;
                        if(display)
                                display->show_hint(ai_hint.first, ai_hint.second);
                        continue;
                default:
                        printf("Invalid command!\n");
                        break;
                }
                // Resign check, end the loop
                if (game->state == 1)
                        break;
        }

        if(client.check_game_result()==3){ // Connection closed
                printf("Connection lost!\n");
                // Probably print you win.
                printf("You win!\n");
                if(display)
                        display->show_game_result(0);
                selected_msg_index=msg_group->first_selectable_message();
        }
        else if(client.check_game_result()==1){
                printf("You win!\n");
                playAnime();
                if(display)
                        display->show_game_result(0);
                selected_msg_index=msg_group->first_selectable_message();
        }
        else{
                printf("You lose!\n");
                playAnime();
                if(display)
                        display->show_game_result(1);
                selected_msg_index=msg_group->first_selectable_message();
        }

        bool confirm = wait_for_confirm();
        if(confirm){
                game->resetGame();
                return;
        }
    }
}

void GameMenu::playAnime()
{
    if(!display)
        return;
    block_input=true;
```

```
        // Hide last mark
        display->update_register(2, display->get_register(2)|0x00ff);
        int count=0;
        for(auto piece:game->winArray){
                display->update_piece_info(piece.first, piece.second, 3,0);
                usleep(500*1000);
                ++count;
                if(count>=5)
                        break;
        }
        block_input=false;
}

void GameMenu::handle_input_press(InputEvent event){
        if(block_input)
                return;
        uint16_t cursor;
        switch (event.type) {
        case XBOX_UP:
        case XBOX_DOWN:
        case XBOX_LEFT:
        case XBOX_RIGHT: // Direction buttons
                // Next message
                selected_msg_index = msg_group->next_message_by_direction(selected_msg_index, event.type);
                board_x = 0xf;
                board_y = 0xf;
                if(msg_group->is_board_selected(selected_msg_index)){
                        printf("(%d,%d)
selected!\n",(selected_msg_index>>12),((selected_msg_index>>8)&0xf));
                        board_x = (selected_msg_index>>12)+1;
                        board_y = ((selected_msg_index>>8)&0xf)+1;
                }
                else
                        printf("%s
selected!\n",msg_group->messages[msg_group->get_message_command(selected_msg_index)].content.c_str());
                if(display)
                        display->update_select(selected_msg_index);
                break;
        case XBOX_A: // Confirm selection
                command_type_ = msg_group->get_message_command(selected_msg_index);
                if(command_type_==10 && waiting)
                        cancel = true;
                else
                        command_received_ = true;
                printf("%s
selected!\n",msg_group->messages[msg_group->get_message_command(selected_msg_index)].content.c_str());
                break;
        case XBOX_B: // TODO: Cancel buttion
        case XBOX_X:
        case XBOX_Y:
        case PAD_MOUSE_LEFT: // Click uses current cursor position
                cursor = msg_group->message_select_by_vga_xy(event.vga_x, event.vga_y);
                // command_type_ = msg_group->get_message_command(cursor);
                if(cursor==0xffff){
```

```
                        printf("None selected! (%d,%d)\n",event.vga_x,event.vga_y);
                        break;
                }
                selected_msg_index = cursor;
                board_x = 0xf;
                board_y = 0xf;
                if(msg_group->is_board_selected(selected_msg_index)){
                        printf("(%d,%d)
selected!\n",(selected_msg_index>>12),((selected_msg_index>>8)&0xf));
                        board_x = (selected_msg_index>>12)+1;
                        board_y = ((selected_msg_index>>8)&0xf)+1;
                }
                else
                        printf("%s
selected!\n",msg_group->messages[msg_group->get_message_command(cursor)].content.c_str());
                command_type_ = msg_group->get_message_command(selected_msg_index);
                if(command_type_==10 && waiting)
                        cancel = true;
                else
                        command_received_ = true;
                break;
        case PAD_MOUSE_RIGHT:
        case PAD_MOUSE_MID:
        case NONE:
                command_type_ = 0xFF;
                command_received_ = true;
        default:
                break;
    }

}

bool GameMenu::wait_for_confirm()
{
        // Wait for confirm command, quit to main menu.
        uint16_t command;
        while(true){
                command = wait_for_command();
                switch (command) {
                case 20: // MESSAGE_BOX_EXIT
                        return true;
                case 21: // MESSAGE_YES
                        return true;
                case 22: // MESSAGE_NO
                        return false;
                default:
                        printf("Invalid command! %d\n",command);
                        break;
                };
        }
}
```

5.7 game/gomoku.h

```C/C++
#ifndef _GOMOKU_HH
#define _GOMOKU_HH

#include <vector>
#include <iostream>
#include "../display/display.h"

class Player;
using namespace std;

class Gomoku {
public:
    int state;                          // 0 for ongoing, 1 for we have a winner already.
    int board_size;                     // The standard Gomoku board is 15x15.
    int current_player;                 // 1 for black's turn, 2 for white's turn.
    int winner;                         // It's useful to know who wins in MCTS. Same as
current_player.
    int WIN_LENGTH;                       // Ending condition: Form an unbroken line of five
stones.
    int mode;                           // 0 for pvp, 1 for pve. This is useful when we regret a
move.
    int regretTimes;                    // Times user has regreted. Used only when online gaming
and playing with AI.
    vector<vector<int>> board;          // The board.
    vector<pair<int, int>> record;      // Game record.
    vector<pair<int, int>> winArray;    // For End of game animation.
       GMKDisplay *display=NULL;                // Display driver pointer
       bool record_game = true;                 // Whether to record game

    Gomoku(int mode) : state(0), board_size(15), current_player(1),
    WIN_LENGTH(5), mode(mode), regretTimes(0), board(board_size, vector<int>(board_size, 0)) {}

    bool on_board(int x, int y);        // Check if (x, y) is a valid position.
    bool valid_move(int x, int y);      // Check if the move at (x, y) is valid.
    int make_move(pair<int, int> move, bool fake=false);// Make a move at (x, y).    vector<pair<int,
int>> getLegalMoves();
    int regret_move();                  // Regret a move. If it's pvp, dial back one move. If it's
pve, dial back 2 moves.
       void end_game(bool black_win);           // Set the winner and end the game.

    bool check_win(int x, int y);       // Check if the winning condition is satisfied.
    bool is_draw();                     // Check if players draw.
    void switchPlayers(bool fake=false);    // Switch Black and white.
    void displayBoard();                // Display board in terminal.

       void set_display(GMKDisplay *display)   // Set display driver
             {this->display=display;}

    void resetGame();                   // Clear board, reset everything.
    void recordGame();                  // Record game in a record/
```

```
    template<int x_step, int y_step>
    bool check(int x, int y) {
        int cur_x = x + x_step*(1-WIN_LENGTH);
        int cur_y = y + y_step*(1-WIN_LENGTH);
        int cumulative_stone = 0;

        for (int i=0; i<2*WIN_LENGTH-1; i++) {
            if (!on_board(cur_x, cur_y)) {
                cur_x += x_step;
                cur_y += y_step;
                continue;
            }

            if (this->board[cur_x][cur_y] != current_player)
                cumulative_stone = 0;
            else cumulative_stone++;

            /* TODO: Rigorously, six (or more) in a row is forbidden. */
            if (cumulative_stone == WIN_LENGTH) {
                for (int j=WIN_LENGTH-1; j>=0; j--)
                    winArray.push_back(make_pair(cur_x-j*x_step, cur_y-j*y_step));
                return true;
            }
            cur_x += x_step;
            cur_y += y_step;
        }

        return false;
    }
};

#endif
```

5.8 game/gomoku.cpp

```
C/C++
#include <iostream>
#include <fstream>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <unistd.h>
#include <vector>
#include "gomoku.h"

bool Gomoku::on_board(int x, int y)
{
    return x >= 0 && x < board_size && y >= 0 && y < board_size;
}

bool Gomoku::valid_move(int x, int y)
{
```

```
       /* Game not ongoing. */
       if(state==1)
              return false;
    return on_board(x, y) && board[x][y] == 0;
}

void Gomoku::end_game(bool black_win)
{
       winner = black_win?1:2;
       state = 1;
       return;
}

int Gomoku::make_move(pair<int, int> move,bool fake)
{
    int x = move.first;
    int y = move.second;

    if (!valid_move(x, y))  return 1;
    board[x][y] = current_player;
       if(display && !fake){
              // Piece value is different from current_player
              display->update_piece_info(x, y, current_player);
              display->play_sound(3);
       }
       if(record_game && !fake)
              record.push_back(make_pair(x, y));
    if (check_win(x, y)) {
              end_game(current_player==1);
        return 0;
    }

    switchPlayers(fake);
    return 0;
}

/*
 * [WIN_LENGTH] of stones in a row.
 * Can be horizontally, vertically, or diagonally.
 */
bool Gomoku::check_win(int x, int y)
{
    // check horizontally
    if (check<1, 0>(x, y))
        return true;

    // check vertically
    if (check<0, 1>(x, y))
        return true;

    // check diagonally
    if (check<1, 1>(x, y) || check<-1, 1>(x, y))
        return true;
```

```
    return false;
}

bool Gomoku::is_draw()
{
    if (state == 1) {
        // Shouldn't reach here.
        cout << "The winning condition is satisfied already." << endl;
        return false;
    }

    for (int i=0; i<board_size; i++)
        for (int j=0; j<board_size; j++)
            if (board[i][j] == 0) return false;

    return true;
}

void Gomoku::switchPlayers(bool fake)
{
    current_player = 3 - current_player;
       if(display && !fake)
             display->switch_turn_mark();
}

void Gomoku::displayBoard()
{
    // [n] -    -    -    -    O    X    -    -    -    -    -    -    -    -    -
    // n is row index, '-' refers to there's no stone on this intercection.
    // 'X' refers to there is a black stone, 'O' refers to white.
    // '@' refers to the current move made by any player.

    string filler = "   ";
    pair<int, int> lastMove = make_pair(-1, -1);
    if (record.size() > 0)
        lastMove = record.back();

    cout << '\n' << filler << "  ";
    for (int i = 0; i < board_size; i++) {
        if (i < 9)
            cout << i + 1 << filler;
        else
            cout << i + 1 << "  ";
    }
    cout << '\n';

    for (int i = 0; i < board_size; i++) {
        if (i < 9)
            cout << ' ' << i + 1 << filler;
        else
            cout << i + 1 << filler;

        for (int j = 0; j < board_size; j++) {
            if (i == lastMove.first && j == lastMove.second)
```

```
                cout << '@' << filler;
            else
            switch (board[i][j]) {
                case 0: {cout << '-' << filler; break;}
                case 1: {cout << 'X' << filler; break;}
                case 2: {cout << 'O' << filler; break;}
            }
        }
        cout << "\n\n";
    }
    // cout << endl;
       // if(vga_gomoku_fd!=-1)
       //     return;
       // const char *filename = VGA_DRIVER_FILENAME;
       // if ((vga_gomoku_fd = open(filename, O_RDWR)) == -1) {
       //     fprintf(stderr, "could not open %s\n", filename);
       //     exit(-1);
       // }
}

void Gomoku::resetGame()
{
    this->board = vector<vector<int>>(board_size, vector<int>(board_size, 0));
    this->record.erase(record.begin(), record.end());
    this->state = 0;
    this->current_player = 1;
    this->winner = 0;
    this->regretTimes = 0;
       this->winArray.clear();
       if(display){
              display->show_confirm_message(false);
              display->clear_board();
       }
}

int Gomoku::regret_move()
{
    // PvP
    if (mode == 0) {
        if (record.size() == 0) {
            cerr << "Cannot regret a move. " << endl;
            return 1;
        }
        auto rit = record.rbegin();
        pair<int, int> lastMove = *rit;
        board[lastMove.first][lastMove.second] = 0;

        // clear the record.
        if(record_game){
                    record.erase(record.end());
             }

             if(display){
                    display->update_piece_info(lastMove.first, lastMove.second, 0);
```

```
                    if(record.size()){ // Set last mark
                          const pair<int, int> &lastMove = *(record.rbegin());

display->update_piece_info(lastMove.first,lastMove.second,board[lastMove.first][lastMove.second]);
                    }
                    else { // No more piece on board, hide last mark
                          printf("No piece Params[2]_:%04x\n",display->get_register(2)|0x00ff );
                          display->update_register(2, display->get_register(2)|0x00ff );
                    }
              }

              switchPlayers();
    }

    // PvE
    else {
        if (record.size() < 2) {
            cerr << "Cannot regret a move. " << endl;
            return 1;
        }
        auto rit = record.rbegin();
        pair<int, int> lastMove = *rit;
        pair<int, int> secondLastMove = *(rit + 1);
        board[lastMove.first][lastMove.second] = 0;
        board[secondLastMove.first][secondLastMove.second] = 0;

        // clear the record.
             if(record_game)
             record.erase(record.end()-2, record.end());
        regretTimes++;

             if(display){
                    display->update_piece_info(lastMove.first, lastMove.second, 0);
                    display->update_piece_info(secondLastMove.first, secondLastMove.second, 0);
                    if(record.size()){ // Set last mark
                          const pair<int, int> &lastMove = *(record.rbegin());

display->update_piece_info(lastMove.first,lastMove.second,board[lastMove.first][lastMove.second]);
                    }
                    else { // No more piece on board, hide last mark
                          printf("Params[2]_:%04x\n",display->get_register(2)|0x00ff );
                          display->update_register(2, display->get_register(2)|0x00ff );
                    }
              }

        if (regretTimes >= 3) {
            cout << "Think before you make a move!" << endl;
        }

        /* TODO: We might disable regret function if player regrets too often. */
    }

    return 0;
}
```

```
void Gomoku::recordGame()
{
    string filename = "record.txt";
    ofstream file(filename, ios::out | ios::app);
    if (!file) {
        cerr << "Error opening file: " << filename << endl;
        return;
    }

    // Record game into the record.txt (append mode)
    file << "========================== New Game ==========================";
    file << "\n\n";

    for (auto move:record) {
        int first = move.first;
        int second = move.second;

        file << first;
        file << ",";
        file << second;
        file << "   ";
    }

    file.close();

    // Should it fail
    if (file.fail()) {
        cerr << "Error while closing record.tx." << endl;
    } else {
        cout << "Written to record.txt." << endl;
    }
}
```

5.9 game/players.h

C/C++

```
#ifndef _PLAYERS_HH
#define _PLAYERS_HH
#include "gomoku.h"

struct PlayerInfo{
       int pid;                        // Player ID :)
       char name[120];
};

class Player {
    public:
    Gomoku *game;                   // Player holds reference to the game.
       PlayerInfo info{0,"Player"};     // Player information
       bool black;                                  // Player piece color
```

```
    Player(Gomoku *game, int id) : game(game) {info.pid=id;}     // the current player is pid == 1?
Black : White.

    int makeMove(pair<int, int>);   // Player makes a move at (x, y).
       int regretMove();                        // Player regrets move
       int resign();                            // Player resigns
};


#endif
```

5.10 game/players.cpp

```
C/C++
#include <iostream>
#include "players.h"

using namespace std;

int Player::makeMove(pair<int, int> move)
{
    if (!game->make_move(move)) {
        // cout << "Player " << pid << " makes move at " << move.first << ", " << move.second << endl;
        return 0;
    }
    else {
        // cout << "Player " << pid << " makes move at " << move.first << ", " << move.second << endl;
        return 1;
    }
}

int Player::regretMove()
{
       if(!game->regret_move())
              return 0;
       return 1;
}

int Player::resign()
{
       game->end_game(!black);
       return 0;
}
```

5.11 game/gomokuAI.h

C/C++

```
#ifndef __GOMOKUAI_HH
#define __GOMOKUAI_HH

#include "gomoku.h"
#include <climits>
#include <map>
#include <cassert>
#include <algorithm>


// Weights for each position. Closer to the center, higher the weight.
const int posWeights[15][15] =
{
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0},
    {0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0},
    {0, 1, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 1, 0},
    {0, 1, 2, 3, 4, 4, 4, 4, 4, 4, 4, 3, 2, 1, 0},
    {0, 1, 2, 3, 4, 6, 6, 6, 6, 6, 4, 3, 2, 1, 0},
    {0, 1, 2, 3, 4, 6, 7, 7, 7, 6, 4, 3, 2, 1, 0},
    {0, 1, 2, 3, 4, 6, 7, 8, 7, 6, 4, 3, 2, 1, 0},
    {0, 1, 2, 3, 4, 6, 7, 7, 7, 6, 4, 3, 2, 1, 0},
    {0, 1, 2, 3, 4, 6, 6, 6, 6, 6, 4, 3, 2, 1, 0},
    {0, 1, 2, 3, 4, 4, 4, 4, 4, 4, 4, 3, 2, 1, 0},
    {0, 1, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 1, 0},
    {0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0},
    {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
};


/*
 * TODO: Maybe need to consider more. (Nah dont think so.)
 *
 * RENJU
 * OPEN_FOURS
 * HALF_OPEN_FOURS
 * OPEN_THREES
 * HALF_OPEN_THREES
 * OPEN_TWOS
 * HALF_OPEN_TWOS
 *
 *
 * 1 for stone in our color.
 * 0 for stone in opponent's color.
 * # for blank intersection.
 * I for invalid places (out of board).
 * XXX: Didn't really take I into consideration which will probably cause bugs
 * when the game reaches the edge but it should be very unlikely to happen.
 */


// http://gomokuworld.com/gomoku/1
struct shapesLookup {
```

```
int RENJU_SCORE,
    OFOUR_SCORE,
    HFOUR_SCORE,
    OTHREE_SCORE,
    HTHREE_SCORE,
    OTWO_SCORE,
    HTWO_SCORE;

string  RENJU,
    OFOUR,
    HFOUR_0, HFOUR_1, HFOUR_2, HFOUR_3, HFOUR_4,
    OTHREE_0, OTHREE_1, OTHREE_2, OTHREE_3, OTHREE_4,
    HTHREE_0, HTHREE_1, HTHREE_2, HTHREE_3, HTHREE_4,
    HTHREE_5, HTHREE_6, HTHREE_7, HTHREE_8, HTHREE_9,
    OTWOS_0, OTWOS_1, OTWOS_2,
    HTWOS_0, HTWOS_1, HTWOS_2, HTWOS_3, HTWOS_4, HTWOS_5, HTWOS_6;

shapesLookup() :
// In the case in <branch debug> debug.cpp, the threshold is 33809.
// 33809 ❌ 33810 ✅
    RENJU_SCORE(5000000),
    OFOUR_SCORE(100000),
    HFOUR_SCORE(50000),
    OTHREE_SCORE(10000),
    HTHREE_SCORE(700),
    OTWO_SCORE(80),
    HTWO_SCORE(20),

    RENJU("11111"),
    OFOUR("#1111#"),

    HFOUR_0("01111#"),
    HFOUR_1("#11110"),
    HFOUR_2("111#1"),
    HFOUR_3("1#111"),
    HFOUR_4("11#11"),

    OTHREE_0("##111##"),
    OTHREE_1("0#111##"),
    OTHREE_2("##111#0"),
    OTHREE_3("#1#11#"),
    OTHREE_4("#11#1#"),

    HTHREE_0("##1110"),
    HTHREE_1("#11#10"),
    HTHREE_2("011#1#"),
    HTHREE_3("0111##"),
    HTHREE_4("1##11"),
    HTHREE_5("11##1"),
    HTHREE_6("1#1#1"),
    HTHREE_7("0#111#0"),
    HTHREE_8("#1#110"),
    HTHREE_9("01#11#"),
```

```
        OTWOS_0("##11##"),
        OTWOS_1("#1#1#"),
        OTWOS_2("1##1"),

        HTWOS_0("###110"),
        HTWOS_1("1###1"),
        HTWOS_2("011###"),
        HTWOS_3("##1#10"),
        HTWOS_4("#1##10"),
        HTWOS_5("01##1#"),
        HTWOS_6("01#1##")
    {}
};

struct VectorComparer {
    bool operator()(const vector<pair<int, int>> v1, const vector<pair<int, int>> v2) const {
    // v1 < v2 return true. otherwise return false.
        if (v1.size() != v2.size()) {
            return v1.size() < v2.size();
        }
        auto sorted_v1 = v1;
        auto sorted_v2 = v2;

        sort(sorted_v1.begin(), sorted_v1.end());
        sort(sorted_v2.begin(), sorted_v2.end());

        return sorted_v1 < sorted_v2;
    }
};

class GomokuAI {
public:
    Gomoku *game;                                           // AI holds reference to the game
instance.
    shapesLookup shapeTable;                                // AI knows all the valid shapes for
evaluating.
    int  maxDepth;                                          // Max calculating depth per move.
    bool applyWeight;                                       // Apply weight to the board.
    int  strategy;                                          // The aggresive degree of AI. (1->3)

    GomokuAI(Gomoku *game, int strategy):
    game(game), maxDepth(5), applyWeight(true), strategy(strategy) {}

    vector<pair<int, int>> getLegalMoves();                 // Get valid intersections on board.
    vector<pair<int, int>> getLegalMoves(bool heuristic);   // Focus on the possible areas to
reduce overhead.
    string posToStr(int x, int y);                          // Turn a (x, y) pair to str for
filling record.
    int evaluate(int player);                               // Evaluate the current board.
    int evaluate(int player, int heuristic);                // Heuristicly evaluate for optimizing.
    int ratePos(int x, int y, int player);                  // Rate the value of one color in a
given position.
    int makeMove(pair<int, int> move, bool fake=false);     // AI makes a move at (x, y).
```

```
    int undoMove(pair<int, int> move);                          // Undo a move at (x, y). Used when
searching.
    pair<int, int> findBestMove();                              // Find the best move, return a (x, y)
pair.
    int MiniMax(int depth, int alpha, int beta, bool isMax);    // Alpha Beta Prunning.
    int getScorefromTable(string s);                            // Look up shapesLookup table to get
score.

    // Beginnings.
    pair<int, int> decideThirdMove();   // AI plays black and it's the third move.

    // The fourth move. Apply the strongest known defense.
    // https://zhuanlan.zhihu.com/p/549399379
    // [花月, Kagetsu], [雨月, Ugetsu]
    pair<int, int> decideFourthMove();
    pair<int, int> isKagestu(pair<int, int> bestMove);
    pair<int, int> isUgetsu(pair<int, int> bestMove);
    pair<int, int> finishMove();

    // Another way of doing all this.
    map<vector<pair<int, int>>, pair<int, int>, VectorComparer> OpeningMap {
        // ai as white
        {{{6, 7}, {7, 7}, {7, 6}}, {5, 6}},
        {{{6, 7}, {7, 7}, {7, 8}}, {5, 8}},
        {{{8, 7}, {7, 7}, {7, 6}}, {9, 6}},
        {{{8, 7}, {7, 7}, {7, 8}}, {9, 8}},
        {{{6, 6}, {7, 7}, {6, 8}, {6, 7}, {5, 7}}, {5, 6}},

        // ai as black
        {{{7, 7}, {6, 6}, {6, 8}, {8, 6}}, {9, 7}},
        {{{7, 7}, {6, 6}, {8, 6}, {8, 8}}, {6, 8}},
        {{{7, 7}, {6, 8}, {8, 8}, {6, 6}}, {7, 5}}
    };

    /* vertical line.
     * (1, 0)
     * (0, 1)   horizontal line.
     * (1, 1)   diagonal line from lt to rb.
     * (1, -1)  diagonal line from rt to lb.
     */
    template<int x_dir, int y_dir>
    string getStrFromPos(int x, int y, int player)              // Get a string consisted of 9 char as
(x, y) in the middle.
    {
        string ret  = "";
        int dir_len = game->WIN_LENGTH - 1;
        int r_begin = x - x_dir*dir_len, c_begin = y - y_dir*dir_len;
        int r_end   = x + x_dir*dir_len, c_end   = y + y_dir*dir_len;
        int cur_r   = r_begin, cur_c = c_begin;
        // cout << "r_begin: " << r_begin <<" c_begin: "<<c_begin<<" r_end: "<< r_end <<" c_end:
"<<c_end<< endl;

        while (cur_r != r_end + x_dir || cur_c != c_end + y_dir) {
            if (game->on_board(cur_r, cur_c)) {
```

```
                if (game->board[cur_r][cur_c] == 0)
                    ret += '#';
                else if (game->board[cur_r][cur_c] == player)
                    ret += '1';
                else
                    ret += '0';
            }
            else
                ret += "I";    // Invalid place.
            cur_r += x_dir;
            cur_c += y_dir;
        }

        return ret;
    }

};

#endif
```

5.12 game/gomokuAI.cpp

```
C/C++
#include "gomokuAI.h"


string GomokuAI::posToStr(int x, int y)
{
    return to_string(x) + to_string(y);
}

vector<pair<int, int>> GomokuAI::getLegalMoves()
{
    vector<pair<int, int>> legalMoves;
    for (int row=0; row<game->board_size; row++)
        for (int col=0; col<game->board_size; col++)
            if (!game->board[row][col])
                legalMoves.push_back(make_pair(row, col));

    return legalMoves;
}

vector<pair<int, int>> GomokuAI::getLegalMoves(bool heuristic)
{
    /* So traverse the board in a spiral way from inside out.
     *
     * This is basically https://leetcode.cn/problems/spiral-matrix/ in reversing direction.
     */

    vector<pair<int, int>> legalMoves;
    int row = game->board_size, col = game->board_size;
```

```
    int x = int((row-1)/2), y = int((col-1)/2);
    int steps = 1;   // Initial step size
    int di = 0;      // Start direction index, begin with moving right
    std::vector<std::pair<int, int>> directions = {
    {0, 1},  // Right
    {1, 0},  // Down
    {0, -1}, // Left
    {-1, 0}  // Up
    };
    int moves = 1;  // Number of handled places.

    if(!game->board[x][y])
        legalMoves.push_back(make_pair(x, y));

    while (moves < row*col) {
        for (int i=0; i<2; i++) {
            for (int j=0; j<steps; j++) {
                int nx = x + directions[di].first;
                int ny = y + directions[di].second;
                if (game->on_board(nx, ny)) {
                    if (!game->board[nx][ny])
                        legalMoves.push_back(make_pair(nx, ny));
                    moves ++;
                    x = nx;
                    y = ny;
                }
            }
            if (moves >= row*col) {
                return legalMoves;
            }
            di = (di+1) % 4;
        }
        steps += 1;
    }

    return legalMoves;
}

int GomokuAI::getScorefromTable(string s)
{
    int score = 0;

    // RENJU
    if (s.find(shapeTable.RENJU) != string::npos)
        score += shapeTable.RENJU_SCORE;

    // OPEN_FOURS
    if (s.find(shapeTable.OFOUR) != string::npos)
        score += shapeTable.OFOUR_SCORE;

    // HALF_OPEN_FOURS
    if (s.find(shapeTable.HFOUR_0) != string::npos ||
        s.find(shapeTable.HFOUR_1) != string::npos ||
        s.find(shapeTable.HFOUR_2) != string::npos ||
```

```
        s.find(shapeTable.HFOUR_3) != string::npos ||
        s.find(shapeTable.HFOUR_4) != string::npos)
        {score += shapeTable.HFOUR_SCORE;}

    // OPEN_THREES
    if (s.find(shapeTable.OTHREE_0) != string::npos ||
        s.find(shapeTable.OTHREE_1) != string::npos ||
        s.find(shapeTable.OTHREE_2) != string::npos ||
        s.find(shapeTable.OTHREE_3) != string::npos ||
        s.find(shapeTable.OTHREE_4) != string::npos)
        {score += shapeTable.OTHREE_SCORE;}

    // HALF_OPEN_THREES
    if (s.find(shapeTable.HTHREE_0) != string::npos||
        s.find(shapeTable.HTHREE_1) != string::npos||
        s.find(shapeTable.HTHREE_2) != string::npos||
        s.find(shapeTable.HTHREE_3) != string::npos||
        s.find(shapeTable.HTHREE_4) != string::npos||
        s.find(shapeTable.HTHREE_5) != string::npos||
        s.find(shapeTable.HTHREE_6) != string::npos||
        s.find(shapeTable.HTHREE_7) != string::npos||
        s.find(shapeTable.HTHREE_8) != string::npos||
        s.find(shapeTable.HTHREE_9) != string::npos)
        {score += shapeTable.HTHREE_SCORE;}


    // OPEN_TWOS
    if (s.find(shapeTable.OTWOS_0) != string::npos ||
        s.find(shapeTable.OTWOS_1) != string::npos ||
        s.find(shapeTable.OTWOS_2) != string::npos)
        {score += shapeTable.OTWO_SCORE;}

    // HALF_OPEN_TWOS
    if (s.find(shapeTable.HTWOS_0) != string::npos ||
        s.find(shapeTable.HTWOS_1) != string::npos ||
        s.find(shapeTable.HTWOS_2) != string::npos ||
        s.find(shapeTable.HTWOS_3) != string::npos ||
        s.find(shapeTable.HTWOS_4) != string::npos ||
        s.find(shapeTable.HTWOS_5) != string::npos ||
        s.find(shapeTable.HTWOS_6) != string::npos)
        {score += shapeTable.HTWO_SCORE;}

    return score;
}

int GomokuAI::ratePos(int x, int y, int player)
{
    // int dirs[4][4] = {{1, 0}, {0, 1}, {1, 1}, {1, -1}};

    int score = 0;
    int weight = 1;
    if (applyWeight && game->record.size() < 8)
        weight = posWeights[x][y];
    string s;
```

```
    s = getStrFromPos<1, 0>(x, y, player);
    score += getScorefromTable(s);
    s = getStrFromPos<0, 1>(x, y, player);
    score += getScorefromTable(s);
    s = getStrFromPos<1, 1>(x, y, player);
    score += getScorefromTable(s);
    s = getStrFromPos<1, -1>(x, y, player);
    score += getScorefromTable(s);

    return weight*score;
}

int GomokuAI::evaluate(int player)
{
    /* TODO: May need rethinking */
    int my_score = 0, op_score = 0;
    int state = 0;

    for (int row = 0; row < game->board_size; row++)
        for (int col = 0; col < game->board_size; col++) {
            state = game->board[row][col];
            if (!state)
                continue;
            else if (state == player)
                my_score += ratePos(row, col, player);
            else
                op_score += ratePos(row, col, 3-player);
        }

    switch(strategy) {
        case 1: {
            return  my_score - 3*op_score;
            break;
        }

        case 2: {
            return  my_score - op_score;
            break;
        }

        case 3: {
            return  3*my_score - op_score;
            break;
        }

        default: {
            // should not get here.
            cout << "Warning: Set a strategy for AI." << endl;
            return my_score - op_score;
        }
    }

}
```

```cpp
int GomokuAI::evaluate(int player, int heuristic)
{
    /* TODO: Heuristic evaluation. */
    return 0;
}

int GomokuAI::makeMove(pair<int, int> move, bool fake)
{
    if (!game->make_move(move, fake)) {
        // cout << "AI makes move at " << move.first << ", " << move.second << endl;
        return 0;
    }
    else {
        // cout << "Invalid move at " << move.first << ", " << move.second << endl;
        return 1;
    }
}

int GomokuAI::undoMove(pair<int, int> move)
{
    int x = move.first, y = move.second;
    game->board[x][y] = 0;

    if (game->state == 1) {
        game->state = 0;
    }

    game->switchPlayers(true);
    return 0;
}

pair<int, int> GomokuAI::decideThirdMove()
{
    // The first move is fixed as (7, 7).
    pair<int, int> secondMove = game->record[1];

    // Out of touch
    if (secondMove.first >= 9 || secondMove.first <= 5 || secondMove.second >= 9 || secondMove.second
<= 5) {
        vector<pair<int, int>> response = {make_pair(6, 8), make_pair(8, 6), make_pair(8, 8),
make_pair(6, 6)};
        srand(time(NULL));
        return response[rand() % 4];
    }

    // Direct4
    else if (secondMove.first + secondMove.second == 15)
        return make_pair(8, 8);

    else if (secondMove.first + secondMove.second == 13)
        return make_pair(6, 6);

    else {
```

```
        int x = secondMove.first - 7;
        int y = secondMove.second - 7;
        vector<pair<int, int>> response = {make_pair(secondMove.first - 2*x, secondMove.second),
                                           make_pair(secondMove.first, secondMove.second - 2*y)};
        srand(time(NULL));
        return response[rand() % 2];
    }
}

/*
 * XXX: May need to revise the logic.
 */
pair<int, int> GomokuAI::isKagestu(pair<int, int> bestMove)
{
    if (game->record.size() != 3) {
        cerr << "Incorrectly entered function isKagestu. " << endl;
        return bestMove;
    }

    // 花月 e.g. (7,7) -> (7,8) -> (8,7)
    pair<int, int> firstMove = game->record[0];
    pair<int, int> secondMove = game->record[1];
    pair<int, int> thirdMove = game->record[2];

    if (firstMove != make_pair(7, 7))
        return bestMove;

    else if ((secondMove == make_pair(7, 8) && thirdMove == make_pair(6, 8))||
    (secondMove == make_pair(8, 7) && thirdMove == make_pair(8, 6)))   { bestMove = make_pair(6, 6);}

    else if ((secondMove == make_pair(6, 7) && thirdMove == make_pair(6, 8))||
    (secondMove == make_pair(7, 6) && thirdMove == make_pair(8, 6)))   { bestMove = make_pair(8, 8);}

    else if ((secondMove == make_pair(8, 7) && thirdMove == make_pair(8, 8))||
    (secondMove == make_pair(7, 6) && thirdMove == make_pair(6, 6)))   { bestMove = make_pair(6, 8);}

    else if ((secondMove == make_pair(7, 8) && thirdMove == make_pair(8, 8))||
    (secondMove == make_pair(6, 7) && thirdMove == make_pair(6, 6)))   { bestMove = make_pair(8, 6);}

    return bestMove;
}

pair<int, int> GomokuAI::isUgetsu(pair<int, int> bestMove)
{
    if (game->record.size() != 3) {
        cerr << "Incorrectly entered function isUgetsu. " << endl;
        return bestMove;
    }

    // 花月 e.g. (7,7) -> (7,6) -> (6,7)
    pair<int, int> firstMove = game->record[0];
    pair<int, int> secondMove = game->record[1];
    pair<int, int> thirdMove = game->record[2];
```

```
    if (firstMove != make_pair(7, 7))
        return bestMove;

    else if ((secondMove == make_pair(7, 6) && thirdMove == make_pair(6, 7))||
    (secondMove == make_pair(7, 8) && thirdMove == make_pair(6, 7)))   { bestMove = make_pair(5, 7);}

    else if ((secondMove == make_pair(7, 8) && thirdMove == make_pair(8, 7))||
    (secondMove == make_pair(7, 6) && thirdMove == make_pair(8, 7)))   { bestMove = make_pair(9, 7);}

    else if ((secondMove == make_pair(6, 7) && thirdMove == make_pair(7, 6))||
    (secondMove == make_pair(8, 7) && thirdMove == make_pair(7, 6)))   { bestMove = make_pair(7, 5);}

    else if ((secondMove == make_pair(6, 7) && thirdMove == make_pair(7, 8))||
    (secondMove == make_pair(8, 7) && thirdMove == make_pair(7, 8)))   { bestMove = make_pair(7, 9);}

    return bestMove;
}

pair<int, int> GomokuAI::finishMove()
{
    /*  Merging with <tsuki> commit a1f5bce698f8f3730fb9b12cde732eb0bf9bb2a0
     *
     *  AI has some bugs such as this scenario:
     *  (Assuming AI is playing as 1, and its opponent is playing as 0)
     *  01111
     *  10000
     *  Now it's AI's turn. Instead of placing a stone to form a RENJU
     *  and end the game, it tends to defend, which is not a good idea.
     *  So let's hardcode this situation.
     */

    pair<int, int> bestMove = make_pair(-1, -1);
    int size = game->record.size();
    if (size < 8)
        return bestMove;
    pair<int, int> scdlstMove = game->record[size-2];  // second to last

    // If AI has a Half four. Don't hesitate!
    int x = scdlstMove.first, y = scdlstMove.second, player = game->current_player;
    vector<string> s_list = {getStrFromPos<1, 0>(x, y, player), getStrFromPos<0, 1>(x, y, player),
    getStrFromPos<1, 1>(x, y, player), getStrFromPos<1, -1>(x, y, player)};

    for (auto s: s_list) {
        if (s.find(shapeTable.HFOUR_0) != string::npos ||
        s.find(shapeTable.HFOUR_1) != string::npos ||
        s.find(shapeTable.HFOUR_2) != string::npos ||
        s.find(shapeTable.HFOUR_3) != string::npos ||
        s.find(shapeTable.HFOUR_4) != string::npos ||
        s.find(shapeTable.OFOUR) != string::npos)
        // We are sure that one single move leads to success.
        {
            for (auto& move:getLegalMoves()) {
                int x = move.first, y = move.second;
                game->board[x][y] = game->current_player;  // temporarily set.
```

```
                if (game->check_win(x, y)) {
                    game->board[x][y] = 0;
                    return move;
                }
                game->board[x][y] = 0;
            }
        }
    }

    return bestMove;
}

pair<int, int> GomokuAI::decideFourthMove()
{
    pair<int, int> bestMove = make_pair(-1, -1);

    // If the player keeps playing at the corners or edges, either
    // (s)he is stupid or does this on purpose.
    pair<int, int> firstMove = game->record[0];
    pair<int, int> thirdMove = game->record[2];
    if (firstMove != make_pair(7, 7) &&
        (thirdMove.first >= 9 || thirdMove.first <= 5 || thirdMove.second >= 9 || thirdMove.second <=
5))
    {
        applyWeight = false;
        vector<pair<int, int>> response = {make_pair(6, 8), make_pair(8, 6), make_pair(8, 8),
make_pair(6, 6),
        make_pair(6, 7), make_pair(7, 6), make_pair(8, 7), make_pair(7, 8)};
        srand(time(NULL));

        while (true) {
            pair<int, int> tmpMove = response[rand() % 8];
            if (game->valid_move(tmpMove.first, tmpMove.second)) {
                bestMove = tmpMove;
                break;
                // Impossible to reach isKagestu and isUgetsu so it's fine.
            }
        }
    }

    bestMove = isKagestu(bestMove);
    if (bestMove.first != -1)
        return bestMove;

    bestMove = isUgetsu(bestMove);
    if (bestMove.first != -1)
        return bestMove;

    return bestMove;
}

pair<int, int> GomokuAI::findBestMove()
{
    /* HACK:TODO: This is a temporary fix of this issue. Not a good idea. */
```

```
    int cur_player = game->current_player;
    int bestScore = INT_MIN;
    pair<int, int> bestMove = {-1, -1};

    if (this->OpeningMap.find(game->record) != this->OpeningMap.end()) {
        bestMove = this->OpeningMap[game->record];
        return bestMove;
    }

    // HACK:TODO: Terrible idea. Fix this if I have time!!!!
    // If there is a [Half Four] in our (others') sructure. handle this immediately!!
    bestMove = finishMove();
    assert(cur_player == game->current_player);
    if (bestMove != make_pair(-1, -1)) {
        if (game->current_player != cur_player)
            game->switchPlayers(); /// FUUUUUUUCKKKKKKKK!
        //game->switchPlayers();
        return bestMove;
    }

    // This is very important!!!
    assert(cur_player == game->current_player);

    /* Hard code for the first several moves. */
    // Hard code for the first move. The best move for the first move is always (7, 7)
    if (game->record.size() == 0)
        return make_pair(7, 7);

    // Hard code for the second move.
    if (game->record.size() == 1) {
        pair<int, int> firstMove = game->record.back();
        int x = firstMove.first, y = firstMove.second;
        if (x == 7 && y == 7) {
        // Several opening choices. Randomly choose one.
            vector<pair<int, int>> response = {make_pair(6, 8), make_pair(8, 6), make_pair(8, 8),
make_pair(6, 6),
                                               make_pair(6, 7), make_pair(7, 6), make_pair(8, 7),
make_pair(7, 8)};
            srand(time(NULL));
            return response[rand() % 8];
        }
        else
            return make_pair(7, 7);
    }

    // Hard code for the third move. This is deterministic so don't need to check validity.
    if (game->record.size() == 2)
        return decideThirdMove();

    // Hard code for the fourth move.
    if (game->record.size() == 3) {
        pair<int, int> move = decideFourthMove();
        if (move.first != -1)
            return move;
```

```
    }

    for(auto& move:getLegalMoves()) {
        makeMove(move, true);
        int score = MiniMax(maxDepth, INT_MAX, INT_MIN, true);
        undoMove(move);
        if (score > bestScore) {
            bestScore = score;
            bestMove = move;
        }
    }

    /* HACK:FIXME: As stated above. */
    if (game->current_player != cur_player)
        game->switchPlayers();

    return bestMove;
}

int GomokuAI::MiniMax(int depth, int alpha, int beta, bool isMax)
{
    if (!depth || game->state == 1) {
        return evaluate(game->current_player);
    }

    if (isMax) {
        int maxEval = INT_MIN;
        for (auto& move:getLegalMoves()) {
            makeMove(move, true);
            int eval = MiniMax(depth-1, alpha, beta, false);   // Now minimizing.
            undoMove(move);
            maxEval = max(maxEval, eval);
            alpha = max(alpha, maxEval);   // Update alpha.
            if (beta <= alpha)
                break;      // Beta prunning.
        }
        return maxEval;
    }
    else {
        int minEval = INT_MAX;
        for (auto& move:getLegalMoves()) {
            makeMove(move, true);
            int eval = MiniMax(depth-1, alpha, beta, true);    // Now maximizing.
            undoMove(move);
            minEval = min(minEval, eval);
            beta = min(beta, minEval);     // Update beta
            if (beta <= alpha)
                break;      // Alpha prunning.
        }
        return minEval;
    }
}
```

5.13 network/network.h

```C/C++
#pragma once

#include "../game/gomoku.h"
#include "../game/players.h"
#include <cstdint>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#include <thread>
#include <vector>
#include "../display/display.h"
#include "../input/input.h"

#define GMK_UDP_PORT 33261
#define GMK_SERVER_PORT 18253
#define GMK_SERVER_PORT_MAX_OFFSET 100
#define GMK_MSG_PLAYER_INFO 1
#define GMK_MSG_GAME_INFO 2
#define GMK_MSG_GAME_START 3
#define GMK_MSG_MOVE_INFO 4
#define GMK_MSG_REQ_PLAYER_INFO 5
#define GMK_MSG_MOVE_REGRET 6
#define GMK_MSG_GAME_RESIGN 7

#define GMK_MSG_UDP_DISCOVER 0xF1
#define GMK_MSG_UDP_READY   0xF2
#define GMK_MSG_UDP_BUSY    0xF3
#define GMK_UDP_DATA_OFFSET (sizeof(struct sockaddr_in))
#define IS_GMK_UDP_MSG(x)  ((x)&0xF0)

struct GMKNetMessage{
       // Gomoku message magic 0x474D4B4D
       uint32_t magic = 0x474D4B4D;
       u_char type;
       // For UDP Message, data should start from msg+sizeof(struct sockaddr_in).
       // Because the first several bytes are preserved for source IP.
       u_char msg[251];
       GMKNetMessage():magic(0x474D4B4D){}
};

struct GMKGameInfo{
       uint32_t board_size;
       int32_t win_length;
};

struct GMKMoveInfo{
       uint32_t idx;
```

```
        int x,y;
};

struct GMKServerInfo{
        string address;
        uint16_t port;
        // Server status
        // 0 - down
        // 1 - busy
        // 2 - ready
        uint16_t status;
};

class GMKNetBase{
public:
        GMKNetBase(Gomoku *game) :
        game_(game),
        local_player_(game_, 0),
        remote_player_(game_, 1)
        {
                game_->state=2;
                // Set game mode to PVP
                game_->mode=0;
        }
        bool make_move(int x, int y);
        bool regret_move();
        bool resign();
        int check_game_result();
        void reset_board();
        int start_game_loop();
        void set_cancel(bool *cancel){cancel_=cancel;}
        bool is_connected(){return connected_;}
        void set_event_handler(InputEventHandler *handler){handler_=handler;}

protected:
        bool send_player_info(const PlayerInfo &info);

        void create_receive_thread();
        virtual void handle_message(const GMKNetMessage &msg) {};
        void receive_thread_callback();
        // TCP thread functions
        void receive_thread_func();
        // UDP thread functions
        bool create_udp_socket();
        void udp_receive_thread_func();

        void start_local_game();
        bool is_move_valid(const GMKMoveInfo &info, const Player &player);
        bool is_players_turn(const Player &player);

        void handle_remote_move(const GMKMoveInfo &info);
        void handle_remote_regret();
        void handle_remote_resign();
```

```
        int remote_fd_=-1, udp_fd_=-1;
        std::thread recv_thread_, udp_recv_thread_;
        Gomoku *game_;
        Player local_player_, remote_player_;
        uint32_t piece_count_;
        uint16_t udp_port_;
        bool loop_stopped_ = true;

        bool connected_ = false;
        bool *cancel_ = NULL;
        InputEventHandler *handler_=NULL;
};

class GMKServer : public GMKNetBase{
public:
        GMKServer(Gomoku *game) : GMKNetBase(game) {}
        // create Server
        bool create();
        // Wait for a player to join the game.
        bool wait_for_player();
        // Send information of this game, board size, etc.
        bool send_game_info();
        // Send game start message
        bool start_game();
        ~GMKServer();
protected:
        // Wait for the player to send player info.
        bool wait_for_player_info();
        void request_player_info();
        void handle_message(const GMKNetMessage &msg) override;
        //void receive_thread_callback() override;
        // Return local black
        bool assign_pieces();
        int local_fd_ = -1;
        uint16_t server_port_;
        bool is_player_joined_ = false;
};

class GMKClient : public GMKNetBase{
public:
        GMKClient(Gomoku *game) : GMKNetBase(game) {server_list_.clear();}
        bool connect_to(const char *ip);
        bool connect_to(const GMKServerInfo &info);
        //bool Disconnect();
        // UDP Broadcast;
        bool send_server_discover();
        std::vector<GMKServerInfo> get_server_list() const {return server_list_;}
        bool wait_for_scan();
        ~GMKClient();
protected:
        void handle_message(const GMKNetMessage &msg) override;
        void update_server_list(const GMKServerInfo &info);
        void reset_server_list();
```

```
        std::vector<GMKServerInfo> server_list_;
        //void receive_thread_callback() override;
};
```

5.14 network/network.cpp

```C/C++
#include "network.h"
#include <cstddef>
#include <cstdint>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <netinet/in.h>
#include <strings.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <thread>
#include <type_traits>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/poll.h>
#include <vector>
#include <fcntl.h>

using namespace std;

bool GMKNetBase::send_player_info(const PlayerInfo &info)
{
        GMKNetMessage msg;
        msg.magic = 0x474D4B4D;
        msg.type = GMK_MSG_PLAYER_INFO;
        memcpy(msg.msg, &info, sizeof(info));
        write(remote_fd_, &msg, sizeof(msg));
        return true;
}

void GMKNetBase::reset_board()
{
        game_->state=2;
        game_->resetGame();
}

void GMKNetBase::create_receive_thread()
{
        recv_thread_=std::thread(&GMKNetBase::receive_thread_func,this);
}

void GMKNetBase::receive_thread_callback()
{
        // Notify the event_handler
```

```
	if(handler_){
		InputEvent event;
		event.type = NONE;
		handler_->handle_input_press(event);
	}
}

void GMKNetBase::receive_thread_func()
{
	GMKNetMessage msg;
	printf("TCP Message handling thread created!\n");
	struct pollfd fds[1];
	fds[0].fd = remote_fd_;
	fds[0].events = POLLIN;
	int ret;
	while((ret=poll(fds,1, 1000))>=0){
		if(!ret)
			continue;
		bzero(&msg, sizeof(msg));
		if(read(remote_fd_,&msg,sizeof(msg))<0)
			break;
		// printf("Message received. %x %d %02x %02x\n",msg.magic,msg.type,msg.msg[0],msg.msg[1]);
		if(msg.magic!=0x474D4B4D){
			printf("Message magic invalid, close connection.\n");
			close(remote_fd_);
			break;
		}
		handle_message(msg);
	}
	printf("connection closed!\n");
	connected_ = false;
	// Do some work after thread exits
	receive_thread_callback();
	return;
}

bool GMKNetBase::create_udp_socket()
{
    struct sockaddr_in anyAddr;
	int reuse_port_val = 0;
	int result = 1;
	if ((udp_fd_ = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
	 perror("UDP Socket");
		udp_fd_=-1;
	 return false;
    }
	if (fcntl(udp_fd_, F_SETFL, O_NONBLOCK) < 0) {
	 perror("server create fcntl");
    }
    if (setsockopt(udp_fd_, SOL_SOCKET, SO_REUSEPORT, &reuse_port_val, sizeof(reuse_port_val)) < 0) {
	 perror("UDP setsockopt(SO_REUSEPORT) failed");
	 return false;
    }
```

```
    memset(&anyAddr, 0, sizeof(anyAddr));
    anyAddr.sin_family = AF_INET;
    anyAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    anyAddr.sin_port = htons(GMK_UDP_PORT);
       udp_port_ = GMK_UDP_PORT;

       for(int i=0;i<GMK_SERVER_PORT_MAX_OFFSET;++i){
              result = bind(udp_fd_, (struct sockaddr*)&anyAddr, sizeof(anyAddr));
              if(!result)
                     break;
              anyAddr.sin_port = htons(GMK_UDP_PORT+i);
              udp_port_ = GMK_UDP_PORT + i;
       }
    if (result != 0) {
        perror("UDP bind address");
              udp_fd_=-1;
        return false;
    }

       printf("UDP Socket created on port %u!\n",udp_port_);

       // Create UDP Receive thread
       udp_recv_thread_=std::thread(&GMKNetBase::udp_receive_thread_func,this);
       return true;
}

void GMKNetBase::udp_receive_thread_func()
{
    struct sockaddr_in srcAddr;
    socklen_t addrLen = sizeof(srcAddr);
       GMKNetMessage msg;
       int ret;

       printf("UDP Message handling thread created!\n");

       memset(&srcAddr, 0, sizeof(srcAddr));
       bzero(&msg, sizeof(msg));
       while(true){
              ret=recvfrom(udp_fd_,
                                  &msg,
                                       sizeof(GMKNetMessage),
                                 0,
                                  (struct sockaddr *)&srcAddr,
                           &addrLen);
              if(ret<0){
                     if (errno == EWOULDBLOCK || errno == EAGAIN){
                            usleep(1000); // Sleep briefly to prevent high CPU usage
                continue;
                     }
                     break;
              }
              // Drop non GMK message
              if(ret!=sizeof(GMKNetMessage))
                     continue;
```

```
            if(msg.magic==0x474D4B4D && IS_GMK_UDP_MSG(msg.type)){
                  printf("UDP message received type %d.\n",msg.type);
                  // Copy source Address, used for later reply.
                  memcpy(msg.msg, &srcAddr, addrLen);
                  handle_message(msg);
            }
            memset(&srcAddr, 0, sizeof(srcAddr));
            bzero(&msg, sizeof(msg));
      }

      close(udp_fd_);
      printf("UDP socket closed!\n");
      if(ret<0){
            perror("UDP Recv:");
      }
      receive_thread_callback();
      return;
}

void GMKNetBase::start_local_game()
{
      game_->resetGame();
      printf("Game started! You take %s piece!\n",
                  local_player_.black?"Black":"White"
      );
      piece_count_=0;
      game_->state=0;
      game_->current_player=1;
      game_->displayBoard();
}

bool GMKNetBase::is_players_turn(const Player &player)
{
      // Not player's turn
      // It is players's turn when:
      // 1. current number of pieces is even and player holds black pieces.
      // 2. current number of pieces is odd and player holds white pieces.
      // That is (piece_count_%2) XOR local_player_.black
      if(!(piece_count_%2^player.black)){
            printf("It's not your turn.\n");
            return false;
      }
      return true;
}

bool GMKNetBase::is_move_valid(const GMKMoveInfo &info, const Player &player)
{
      // Game not started
      if(game_->state>0){
            printf("Game not started.\n");
            return false;
      }
```

```
	if(!is_players_turn(player))
		return false;

	if(!game_->valid_move(info.x,info.y))
		return false;
	return true;
}

bool GMKNetBase::make_move(int x, int y)
{
	GMKMoveInfo info;
	info.idx=piece_count_;
	info.x=x;
	info.y=y;

	if(!is_move_valid(info, local_player_))
		return false;

	// Local player takes move
	local_player_.makeMove(make_pair(x, y));
	++piece_count_;
	// Send move message
	GMKNetMessage msg;
	msg.magic = 0x474D4B4D;
	msg.type = GMK_MSG_MOVE_INFO;
	memcpy(msg.msg, &info, sizeof(info));
	write(remote_fd_, &msg, sizeof(msg));

	printf("%d:%s (You) take a move at (%d,%d)\n", piece_count_,
		local_player_.black?"Black":"White",
		x,y
	);
	game_->displayBoard();
	return true;
}

bool GMKNetBase::regret_move()
{
	// Only allow local player to regret when it's player's turn
	if(!is_players_turn(local_player_))
		return false;
	// Local player regrets
	game_->regret_move();
	// Fall back 2 moves
	piece_count_-=2;

	// Send move message
	GMKNetMessage msg;
	msg.magic = 0x474D4B4D;
	msg.type = GMK_MSG_MOVE_REGRET;
	write(remote_fd_, &msg, sizeof(msg));

	printf("%d:%s (You) regret a move!\n", piece_count_,
		local_player_.black?"Black":"White"
```

```
        );
        game_->displayBoard();
        return true;
}

bool GMKNetBase::resign()
{
        // local player resigns
        local_player_.resign();

        // Send move message
        GMKNetMessage msg;
        msg.magic = 0x474D4B4D;
        msg.type = GMK_MSG_GAME_RESIGN;
        write(remote_fd_, &msg, sizeof(msg));

        printf("%d:%s (You) resigned the game!\n", piece_count_,
                  local_player_.black?"Black":"White"
        );
        game_->displayBoard();
        return true;
}

int GMKNetBase::check_game_result()
{
        if(!connected_)
                return 3;
        // Game not started/ended.
        if(game_->state!=1)
                return -1;
        return local_player_.black^(game_->winner==2);
}

void GMKNetBase::handle_remote_move(const GMKMoveInfo &info)
{
        if(!is_move_valid(info, remote_player_)){
                // FIXME:Remote move not valid
                return ;
        }
        // Remote player takes move
        remote_player_.makeMove(make_pair(info.x, info.y));
        ++piece_count_;
        printf("%d:%s takes a move at (%d,%d)\n", piece_count_,
                remote_player_.black?"Black":"White",
                info.x,info.y
        );
        game_->displayBoard();

        if(handler_){
                InputEvent event;
                event.type = NONE;
                handler_->handle_input_press(event);
        }
}
```

```
void GMKNetBase::handle_remote_regret()
{
	if(!is_players_turn(remote_player_)){
		return ;
	}
	if(remote_player_.regretMove()){
	// if return value is not 0, an error occurs
		return;
	}
	piece_count_-=2;
	printf("%d:%s regrets a move!\n", piece_count_,
		remote_player_.black?"Black":"White"
	);
	game_->displayBoard();
	if(handler_){
		InputEvent event;
		event.type = NONE;
		handler_->handle_input_press(event);
	}
}

void GMKNetBase::handle_remote_resign()
{
	remote_player_.resign();
	printf("%d:%s resigns!\n", piece_count_,
		remote_player_.black?"Black":"White"
	);
	game_->displayBoard();
	// Notify the main thread
	if(handler_){
		InputEvent event;
		event.type = NONE;
		handler_->handle_input_press(event);
	}
}

void GMKServer::handle_message(const GMKNetMessage &msg)
{
	if(msg.type==GMK_MSG_PLAYER_INFO){// Player info
		PlayerInfo info;
		memcpy(&info, msg.msg, sizeof(info));
		remote_player_.info=info;
		is_player_joined_ = true;
	}
	else if(msg.type==GMK_MSG_MOVE_INFO){ // Move info
		GMKMoveInfo info;
		memcpy(&info, msg.msg, sizeof(info));
		handle_remote_move(info);
	}
	else if(msg.type==GMK_MSG_MOVE_REGRET){ // Remote player regrets
		handle_remote_regret();
	}
	else if(msg.type==GMK_MSG_GAME_RESIGN){ // Remote player resigns
```

```
            handle_remote_resign();
      }
      else if(msg.type==GMK_MSG_UDP_DISCOVER){ // UDP Server discover
            GMKNetMessage reply;
            sockaddr_in addr;

            // Read reply address
            memcpy(&addr, msg.msg, sizeof(struct sockaddr_in));
            //addr.sin_port = GMK_UDP_PORT;
            // Send server status
            // Playing or ready to play
            reply.type = connected_?GMK_MSG_UDP_BUSY:GMK_MSG_UDP_READY;
            // Copy Server Port
            memcpy(reply.msg+GMK_UDP_DATA_OFFSET, &server_port_, sizeof(uint16_t));
            sendto(udp_fd_, &reply, sizeof(reply),0,(struct sockaddr *)&addr, sizeof(addr));
            printf("Sending server status to %s:%u\n",inet_ntoa(addr.sin_addr),addr.sin_port);
      }
}


bool GMKServer::create()
{
    // socket create and verification
    local_fd_ = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    struct sockaddr_in servaddr;
      if (local_fd_ == -1) {
       printf("Gomoku Server: Socket creation failed...\n");
       return false;
    }
      if (fcntl(local_fd_, F_SETFL, O_NONBLOCK) < 0) {
       perror("server create fcntl");
    }
    printf("Gomoku Server: Socket successfully created..\n");
      bzero(&servaddr, sizeof(servaddr));

      // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(GMK_SERVER_PORT);
      server_port_ = GMK_SERVER_PORT;

    // Binding newly created socket to given IP and verification
      int result = 1;
      int i=0;
      for(;i<GMK_SERVER_PORT_MAX_OFFSET;++i){
            result = bind(local_fd_, (struct sockaddr*)&servaddr, sizeof(servaddr));
            if(!result)
                  break;
            servaddr.sin_port = htons(GMK_SERVER_PORT+i);
            server_port_ = GMK_SERVER_PORT+i;
      }
    if (result != 0) {
       printf("Gomoku Server: Socket bind failed...\n");
       return false;
```

```
    }
    printf("Gomoku Server: Socket successfully binded..\n");

    // Now server is ready to listen and verification
    if ((listen(local_fd_, 5)) != 0) {
        printf("Gomoku Server: Listen failed...\n");
        return false;
    }
       printf("Gomoku Server: Server listening at port %d..\n",GMK_SERVER_PORT+i);
       create_udp_socket();
       return true;
}
GMKServer::~GMKServer()
{
       printf("close(remote_fd_);\n");
       if(remote_fd_>0)
              close(remote_fd_);
       printf("close(local_fd_);\n");
       if(local_fd_>0)
              close(local_fd_);
       printf("close(udp_fd_);\n");
       if(udp_fd_>0)
              close(udp_fd_);
       printf("udp_recv_thread_.join();\n");
       if(udp_recv_thread_.joinable())
              udp_recv_thread_.join();
       printf("recv_thread_.join();\n");
       if(recv_thread_.joinable())
              recv_thread_.join();
}

bool GMKServer::wait_for_player()
{
       struct sockaddr_in client;
       uint len;
       len = sizeof(client);

       if(recv_thread_.joinable())
              recv_thread_.join();
       printf("Gomoku Server: Waiting for player to join...\n");
       if(cancel_)
              *cancel_=false;
       // Wait a player to join
       while(true){
              // Accept the data packet from client and verification
              remote_fd_ = accept(local_fd_, (struct sockaddr*)&client, &len);
              if (remote_fd_ < 0) {
                     if (errno == EWOULDBLOCK || errno == EAGAIN) {
                            if(cancel_)
                                   if(*cancel_){
                                          printf("Gomoku Server: Wait for player cancelled!\n");
                                          return false;
                                   }
                       // No pending connections, continue to next iteration
```

```
                usleep(1000); // Sleep briefly to prevent high CPU usage
                continue;
            }
                    printf("Gomoku Server: Server accept failed...\n");
                    return false;
             }
             printf("Gomoku Server: Server accept the client...\n");

             // Create message handling thread
             create_receive_thread();
             request_player_info();
             /*
             * After connection established,
             * the gomoku client must send player info in 5s,
             * or the connection will be closed.
             */
             if(!wait_for_player_info()){
                    printf("Gomoku Server: Not a valid player.\n");
                    close(remote_fd_);
                    recv_thread_.join();
             }
             else
                    break;
       }
       connected_ = true;
       return true;
}

void GMKServer::request_player_info()
{
       GMKNetMessage msg;
       msg.magic = 0x474D4B4D;
       msg.type = GMK_MSG_REQ_PLAYER_INFO;
       write(remote_fd_, &msg, sizeof(msg));
}

bool GMKServer::wait_for_player_info()
{
       printf("Gomoku Server: Waiting for remote player info...\n");
       if(cancel_)
              *cancel_=false;
       for(int i=0;i<100;++i){
              usleep(50000);
              if(cancel_)
                     if(*cancel_)
                            break;
              if(is_player_joined_){
                     printf("Gomoku Server: Player info received!\n");
                     // Send server player info
                     send_player_info(local_player_.info);
                     // Send server game info
                     send_game_info();
                     return true;
              }
```

```
        }
        printf("Gomoku Server: Player info timeout!\n");
        return false;
}

bool GMKServer::send_game_info()
{
        GMKNetMessage msg;
        bzero(&msg, sizeof(msg));
        msg.magic = 0x474D4B4D;
        msg.type = GMK_MSG_GAME_INFO;

        GMKGameInfo info;
        info.board_size = game_->board_size;
        info.win_length = game_->WIN_LENGTH;
        memcpy(msg.msg, &info, sizeof(info));
        write(remote_fd_, &msg, sizeof(msg));
        return true;
}

bool GMKServer::start_game()
{
        bool local_first = assign_pieces();

        // Send start signal
        GMKNetMessage msg;
        bzero(&msg, sizeof(msg));
        msg.magic = 0x474D4B4D;
        msg.type = GMK_MSG_GAME_START;
        msg.msg[0]=  local_player_.black;
        msg.msg[1] = remote_player_.black;
        write(remote_fd_, &msg, sizeof(msg));

        // Wait for remote acknowledge?
        start_local_game();
        return local_first;
}

bool GMKServer::assign_pieces()
{
        srand(time(NULL));
        local_player_.black=((rand()%4)>=2);
        remote_player_.black=!local_player_.black;
        return local_player_.black;
}

bool GMKClient::connect_to(const char *ip)
{
        GMKServerInfo info;
        info.address = ip;
        info.port = GMK_SERVER_PORT;
        return connect_to(info);
}
```

```
bool GMKClient::connect_to(const GMKServerInfo &info)
{
       struct sockaddr_in servaddr;
       // socket create and verification
    remote_fd_ = socket(AF_INET, SOCK_STREAM, 0);
    if (remote_fd_ == -1) {
        printf("Gomoku Client: Socket creation failed...\n");
        return false;
    }
       printf("Gomoku Client: Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr(info.address.c_str());
    servaddr.sin_port = htons(info.port);

    // connect the client socket to server socket
    if (connect(remote_fd_, (struct sockaddr*)&servaddr, sizeof(servaddr))
        != 0) {
        printf("Gomoku Client: connect_toion with the server failed...\n");
        return false;
    }
    printf("Gomoku Client: connect_toed to the server..\n");
       connected_ = true;
       create_receive_thread();
       return true;
}

void GMKClient::reset_server_list()
{
       for(auto server: server_list_){
              server.status=0;
       }
}

bool GMKClient::send_server_discover()
{
       struct sockaddr_in broadcastAddr;
       int broadcastPermission = 1;
       struct GMKNetMessage msg;

       if(udp_fd_<0){
              if(!create_udp_socket())
                     return false;
       }

    if (setsockopt(udp_fd_, SOL_SOCKET, SO_BROADCAST, &broadcastPermission,
sizeof(broadcastPermission)) < 0) {
        perror("UDP setsockopt");
        close(udp_fd_);
        return false;
    }
```

```
    // Reset server list
       reset_server_list();

       memset(&broadcastAddr, 0, sizeof(broadcastAddr));
       broadcastAddr.sin_family = AF_INET;
       broadcastAddr.sin_addr.s_addr = htonl(INADDR_BROADCAST);
       bzero(&msg, sizeof(msg));
       msg.magic = 0x474D4B4D;
       msg.type = GMK_MSG_UDP_DISCOVER;
       for(int i=0;i<GMK_SERVER_PORT_MAX_OFFSET;++i){
              broadcastAddr.sin_port = htons(GMK_UDP_PORT+i);
              sendto(udp_fd_, &msg, sizeof(GMKNetMessage), 0, (struct sockaddr *)&broadcastAddr,
sizeof(broadcastAddr));
       }

       return true;
}

void GMKClient::update_server_list(const GMKServerInfo & info)
{
       for(auto element: server_list_){
              if(element.address==info.address&&element.port==info.port){
                     element.status=info.status;
                     return;
              }
       }
       server_list_.push_back(info);
}


void GMKClient::handle_message(const GMKNetMessage &msg)
{
       if(msg.type==GMK_MSG_GAME_INFO){// Game info
              GMKGameInfo info;
              memcpy(&info, msg.msg, sizeof(info));
              game_->board_size=info.board_size;
              game_->WIN_LENGTH=info.win_length;
              printf("Game info received! Waiting game to start...\n");
       }
       else if(msg.type==GMK_MSG_PLAYER_INFO){
              PlayerInfo info;
              memcpy(&info, msg.msg, sizeof(info));
              remote_player_.info=info;
       }
       else if(msg.type==GMK_MSG_GAME_START){// Game start signal
              remote_player_.black=msg.msg[0];
              local_player_.black=msg.msg[1];
              // FIXME: Move display logic
              if(game_->display){
                     game_->display->set_player_piece(local_player_.black);
                     game_->display->set_turn_mark(local_player_.black);
              }
              start_local_game();
       }
```

```
	else if(msg.type==GMK_MSG_MOVE_INFO){ // Move info
		GMKMoveInfo info;
		memcpy(&info, msg.msg, sizeof(info));
		handle_remote_move(info);
	}
	else if(msg.type==GMK_MSG_MOVE_REGRET){ // Remote player regrets
		handle_remote_regret();
	}
	else if(msg.type==GMK_MSG_GAME_RESIGN){ // Remote player resigns
		handle_remote_resign();
	}
	else if(msg.type==GMK_MSG_REQ_PLAYER_INFO){ // Server requests player info
		send_player_info(local_player_.info);
		printf("Server requests player info\n");
	}
	else if(IS_GMK_UDP_MSG(msg.type)){ // Handle UDP message
		// Read server port
		// Read Server Port
		GMKServerInfo info;
		sockaddr_in addr;

		// Read server address
		memcpy(&addr, msg.msg, sizeof(struct sockaddr_in));
		info.address = inet_ntoa(addr.sin_addr);
		// Read server port
		memcpy(&info.port, msg.msg+GMK_UDP_DATA_OFFSET, sizeof(uint16_t));

		switch (msg.type) {
		case GMK_MSG_UDP_READY: info.status=2; break;
		case GMK_MSG_UDP_BUSY: info.status=1;  break;
		default: return;    //Drop Other messages
		}
		update_server_list(info);
	}
}
GMKClient::~GMKClient()
{
	if(remote_fd_>0)
		close(remote_fd_);
	if(udp_fd_>0)
		close(udp_fd_);
	if(udp_recv_thread_.joinable())
		udp_recv_thread_.join();
	if(recv_thread_.joinable())
		recv_thread_.join();
}

bool GMKClient::wait_for_scan()
{
	if(cancel_)
		*cancel_ = false;
	for(size_t i=0;i<50;++i){
		if(cancel_)
			if(*cancel_)
```

```
                        break;
                usleep(100000);
                for(auto server:get_server_list()){
                        if(server.status==2){
                                return true;
                        }
                }
        }
        if(server_list_.empty())
                return false;
        return true;
}
```

5.15 input/input.h

```C/C++
#ifndef INPUT_H
#define INPUT_H

#include <cstdint>
#include <libusb-1.0/libusb.h>
#include <stdint.h>
#include <thread>
#include "../display/display.h"

#define INPUT_WAIT_INTERVAL_US 10000

enum InputEventType : unsigned char{
        // 0x0000xxxx - Xbox Event
        XBOX_UP = 0,
        XBOX_DOWN = 1,
        XBOX_LEFT = 2,
        XBOX_RIGHT = 3,
        XBOX_START = 4,
        XBOX_BACK = 5,
        XBOX_LSTICK = 6,
        XBOX_RSTICK = 7,
        XBOX_LB = 8,
        XBOX_LT = 9,
        XBOX_MENU = 10,
        XBOX_A = 12,
        XBOX_B = 13,
        XBOX_X = 14,
        XBOX_Y = 15,
        // 0x0001xxxx - Pad event
        PAD_MOUSE_LEFT = 16,
        PAD_MOUSE_RIGHT = 17,
        PAD_MOUSE_MID = 18,
        NONE = 0xFF
};
```

```
struct InputEvent{
	InputEventType type;
	uint16_t vga_x,vga_y;
};

class InputEventHandler{
public:
	virtual void handle_input_press(InputEvent event){
		printf("%d:pressed\n",event.type);
		command_received_ = true;
	}
	virtual void handle_input_release(InputEvent event){
		printf("%d:released\n",event.type);
	}

	uint16_t wait_for_command();

protected:
	bool command_received_ = false;
	uint16_t command_type_ = 0;
};

class BaseInputDevice{
public:
	BaseInputDevice(){};
	~BaseInputDevice(){close_device();}
	bool open_device();
	void close_device();

	void print_touchpad_message(struct XPPenMessage msg);

	virtual void create_handling_thread() = 0;
	void stop_handling_thread();

	void set_display(GMKDisplay *display) {display_=display;}
	GMKDisplay *get_display() {return display_;}
	void set_input_handler(InputEventHandler *handler){input_handler_=handler;}

protected:
	virtual void handle_message_func() = 0;

	GMKDisplay *display_=NULL;
	libusb_device_handle *handle_=NULL;
	InputEventHandler *input_handler_=NULL;
	libusb_context *context_=NULL;

	std::string device_name_ = "";
	uint16_t vendor_id_ = 0;
	uint16_t product_id_ = 0;
	int interface_ = 0;
	unsigned char endpoint_ = 0;

	std::thread thread_;
	int thread_stopped_ = 0;
```

```
	const int usb_timeout_ = 2000;
};

class Console: public BaseInputDevice{
public:
	void create_handling_thread();
protected:
	void handle_message_func();
};

#endif
```

5.16 input/input.cpp

```
C/C++
#include "input.h"
#include <cstdint>
#include <unistd.h>

uint16_t InputEventHandler::wait_for_command()
{
	command_received_ = false;
	while(!command_received_)
		usleep(INPUT_WAIT_INTERVAL_US);

	return command_type_;
}


bool BaseInputDevice::open_device()
{
	printf("%u %u %d %02x\n",vendor_id_,product_id_,interface_,endpoint_);
	int r;

    r = libusb_init(&context_);
    if (r < 0) {
        fprintf(stderr, "fail to init libusb: %d\n", r);
        return false;
    }

	handle_ = libusb_open_device_with_vid_pid(context_,vendor_id_,product_id_);
	if(!handle_){
		std::string error = "Open "+device_name_+" usb device";
		perror(error.c_str());
		goto ERR;
	}

	// Use interface 0
	r = libusb_set_auto_detach_kernel_driver(handle_,1);
	if(r!=LIBUSB_SUCCESS){
		printf("Detach kernel driver failed.\n");
```

```
                goto ERR;
        }
        if (libusb_kernel_driver_active(handle_, interface_) == 1) {
                r = libusb_detach_kernel_driver(handle_, interface_);
                if (r == 0) {
                        printf("Kernel Driver Detached\n");
                } else {
                        fprintf(stderr, "Error detaching kernel driver: %d\n", r);
                }
        }

        return true;

ERR:
        close_device();
        return false;
}

void BaseInputDevice::close_device()
{
        if(handle_!=NULL)
                libusb_close(handle_);
        if(context_)
                libusb_exit(context_);
}

void BaseInputDevice::stop_handling_thread()
{
        thread_stopped_ = 1;
        if(thread_.joinable())
                thread_.join();
}

void Console::create_handling_thread()
{
        thread_stopped_ = 0;
        thread_ = std::thread(&Console::handle_message_func,this);
}

void Console::handle_message_func()
{
        while(!thread_stopped_){

        }
}
```

5.17 input/touchpad.h

```
C/C++
#ifndef TOUCHPAD_HH
#define TOUCHPAD_HH
```

```cpp
#include "../display/display.h"
#include <cstdint>
#include <libusb-1.0/libusb.h>
#include <stdint.h>
#include <thread>
#include "input.h"

#define TOUCHPAD_VENDOR_ID 0x28bd
#define TOUCHPAD_PRODUCT_ID 0x0928
#define TOUCHPAD_INTERFACE 1
#define TOUCHPAD_ENDPOINT 0x82
#define TOUCHPAD_SHOW_CURSOR(status) (((status)&0xf0)==0xa0)

#define TOUCHPAD_MOUSE_EVENT_COUNT 3
#define TOUCHPAD_MOUSE_LEFT 0
#define TOUCHPAD_MOUSE_RIGHT 1
#define TOUCHPAD_MOUSE_MID 2

struct libusb_device_handle;
struct XPPenMessage{
	// magic number 0x07
	unsigned char magic;
	// mouse status
	// 0xc0 - pen not detected
	// 0xa0 - pen detected
	// 0xa1 - left mouse down
	// 0xa2 - right mouse down
	// 0xa3 - left & right mouse down
	unsigned char status;

	// Coordinates, need to convert to display coordinate
	// Horizontal - (uint16_t)((horizontal)/32768.0*480)
	// Vertical - (uint16_t)((vertical)/51.2)

	// Horizontal, similar to hcount, 0~32767
	// 0 at the side that is close to the buttons
	uint16_t horizontal;
	// Vertical, similar to vcount, 0~32767
	uint16_t vertical;
	// pen pressure level
	uint16_t pressure;
	// currently unknown
	uint16_t unknown;
};

class Touchpad : public BaseInputDevice{
public:
	Touchpad()
	{
		device_name_ = "Touchpad";
		vendor_id_ = TOUCHPAD_VENDOR_ID;
		product_id_ = TOUCHPAD_PRODUCT_ID;
		interface_ = TOUCHPAD_INTERFACE;
```

```
            endpoint_ = TOUCHPAD_ENDPOINT;
        }
        void print_touchpad_message(struct XPPenMessage msg);

        void create_handling_thread() override;

protected:
        void handle_message_func() override;
        void handle_touchpad_pen_event(struct XPPenMessage *msg);
};

#endif
```

5.18 input/touchpad.cpp

```
C/C++
#include <cstdint>
#include <libusb-1.0/libusb.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <thread>
#include "input.h"
#include "touchpad.h"

void Touchpad::print_touchpad_message(struct XPPenMessage msg)
{
        const char *touchpad_status_str[]={"NO PEN","PEN","LEFT","RIGHT","LEFT&RIGHT","UNKNOWN"};
        int status;
        switch (msg.status) {
        case 0xc0: status=0;break;
        case 0xa0: status=1;break;
        case 0xa1: status=2;break;
        case 0xa2: status=3;break;
        case 0xa3: status=4;break;
        default: status=5;break;
        }
        printf("Info:\n");
        printf("Magic:0x%02x\n",msg.magic);
        printf("Status:0x%02x %s\n",msg.status,touchpad_status_str[status]);

printf("Coordinate:%u,%u\n",(uint16_t)((msg.horizontal)/51.2),(uint16_t)((msg.vertical)/32768.0*480));
        printf("Pressure:%u\n",msg.pressure);
        printf("Unknown:%u\n",msg.unknown);
}

void Touchpad::create_handling_thread()
{
        thread_stopped_ = 0;
        std::thread t(&Touchpad::handle_message_func,this);
```

```
	thread_.swap(t);
}

void Touchpad::handle_message_func()
{
	int data_len = 0;
	int r = 0;
	uint16_t vga_x,vga_y;
	bool show_cursor=true;
	XPPenMessage data;

	r = libusb_claim_interface(handle_, interface_);
	if (r < 0) {
			fprintf(stderr, "claim interface error %d - %s\n", r,
libusb_strerror((libusb_error)r));
			goto OUT;
	}
	printf("claimed interface %d\n",interface_);

	while(!thread_stopped_){
		// Read data
		r = libusb_interrupt_transfer(handle_, endpoint_, (unsigned char *)&data, sizeof(data),
&data_len, usb_timeout_);
		show_cursor = true;
		switch (r) {
		case LIBUSB_ERROR_TIMEOUT: show_cursor = false; break;
		case LIBUSB_ERROR_PIPE: printf("LIBUSB_ERROR_PIPE\n"); goto OUT;
		case LIBUSB_ERROR_OVERFLOW: printf("LIBUSB_ERROR_OVERFLOW\n"); goto OUT;
		case LIBUSB_ERROR_NO_DEVICE: printf("LIBUSB_ERROR_NO_DEVICE\n"); goto OUT;
		case LIBUSB_ERROR_BUSY: printf("LIBUSB_ERROR_BUSY\n"); goto OUT;
		case LIBUSB_ERROR_INVALID_PARAM: printf("LIBUSB_ERROR_INVALID_PARAM\n"); goto OUT;
		}
		// printf("\nData: length %d\n",data_len);
		// print_touchpad_message(data);
		// printf("\n");
		// TODO: Message handling
		vga_x = ((data.horizontal)/51.2);
		vga_y = ((data.vertical)/32768.0*480);
		if(display_){
			if(!display_->update_touchpad_cursor(vga_x,vga_y,show_cursor))
				perror("update touchpad cursor");
		}
		if(input_handler_)
			handle_touchpad_pen_event(&data);
	}
OUT:
	thread_stopped_ = 1;
	close_device();
	return;
}

void Touchpad::handle_touchpad_pen_event(struct XPPenMessage *msg)
{
	const int debounce_threshold = 0;
```

```
        static unsigned int debounce = 0;
        static unsigned int debounce_status = 0xc0;
        static unsigned char prev_status = 0xc0;
        if(msg->status==debounce_status)
                ++debounce;
        else {
                debounce_status = msg->status;
                debounce = 0;
        }
        if(debounce>=debounce_threshold && prev_status!=msg->status){
                // DEBUG
                print_touchpad_message(*msg);
                printf("\n\n");

                InputEvent event;
                // Set event PAD type 1<<5
                unsigned char event_type=16;

                event.vga_x = ((msg->horizontal)/51.2);
                event.vga_y = ((msg->vertical)/32768.0*480);
                // Check pen press status
                for(int i=0;i<TOUCHPAD_MOUSE_EVENT_COUNT;++i){
                        unsigned char pressed = (msg->status>>i)%2;
                        if(pressed!=((prev_status>>i)%2)){
                                // Set press/release and button status
                                event_type+=i;
                                event.type = InputEventType(event_type);
                                // Send pen input event
                                if(pressed)
                                        input_handler_->handle_input_press(event);
                                else
                                        input_handler_->handle_input_release(event);
                        }
                }

                prev_status = msg->status;
        }
}
```

5.19 input/xboxcont.h

C/C++

```
// xboxcont.h
#ifndef XBOXCONT_H
#define XBOXCONT_H

#include <cstdint>
#include <stdio.h>
#include <libusb-1.0/libusb.h>
#include <sys/types.h>
#include <time.h>
```

```
#include <unistd.h>

#include "../display/display.h"
#include "input.h"

#define CONTROLLER_VENDOR_ID 0x045e
#define CONTROLLER_PRODUCT_ID 0x028e
#define CONTROLLER_ENDPOINT 0x81

struct XboxMessage{
	uint16_t header;
	uint16_t buttons;
	unsigned char lt_level;
	unsigned char rt_level;
	int16_t lstick_dir_right;
	int16_t lstick_dir_up;
	int16_t rstick_dir_right;
	int16_t rstick_dir_up;
	unsigned char unknown[9];
};

class XboxController: public BaseInputDevice{
public:
	XboxController()
	{
		device_name_ = "Xbox controller";
		vendor_id_ = CONTROLLER_VENDOR_ID;
		product_id_ = CONTROLLER_PRODUCT_ID;
		interface_ = 0;
		endpoint_ = CONTROLLER_ENDPOINT;
	}
	void create_handling_thread() override;
	void print_xbox_message(XboxMessage msg);

protected:
	void handle_message_func() override;
	void handle_xbox_button_event(uint16_t status);
};

#endif // XBOXCONT_H
```

5.20 input/xboxcont.cpp

```
C/C++
#include <cstdint>
#include <stdio.h>
#include <libusb-1.0/libusb.h>
#include <time.h>
#include <unistd.h>

#include "xboxcont.h"
```

```
#include "input.h"

void XboxController::create_handling_thread()
{
	thread_stopped_ = 0;
	std::thread t(&XboxController::handle_message_func,this);
	thread_.swap(t);
}

void XboxController::handle_message_func()
{
	int data_len = 0;
	int r = 0;
	XboxMessage data;

	r = libusb_claim_interface(handle_, interface_);
	if (r < 0) {
			fprintf(stderr, "claim interface error %d - %s\n", r,
libusb_strerror((libusb_error)r));
			goto OUT;
	}
	printf("claimed interface %d\n",interface_);

	while(!thread_stopped_){
		// Read data
		r = libusb_interrupt_transfer(handle_, endpoint_, (unsigned char *)&data,
sizeof(XboxMessage), &data_len, usb_timeout_);
		switch (r) {
		case LIBUSB_ERROR_TIMEOUT: break;
		case LIBUSB_ERROR_PIPE: printf("LIBUSB_ERROR_PIPE\n"); goto OUT;
		case LIBUSB_ERROR_OVERFLOW: printf("LIBUSB_ERROR_OVERFLOW\n"); goto OUT;
		case LIBUSB_ERROR_NO_DEVICE: printf("LIBUSB_ERROR_NO_DEVICE\n"); goto OUT;
		case LIBUSB_ERROR_BUSY: printf("LIBUSB_ERROR_BUSY\n"); goto OUT;
		case LIBUSB_ERROR_INVALID_PARAM: printf("LIBUSB_ERROR_INVALID_PARAM\n"); goto OUT;
		}
		// print_xbox_message(data);
		if(input_handler_)
			handle_xbox_button_event(data.buttons);
	}
OUT:
	thread_stopped_ = 1;
	close_device();
	return;
}

void XboxController::handle_xbox_button_event(uint16_t status)
{
	const int debounce_threshold = 1;
	static unsigned int debounce = 0;
	static unsigned int debounce_status = 0xc0;
	static uint16_t prev_status = 0xc0;
	// If 1, only sends up/down events
	// If 2, only sends left/right events
	static int pad_dir = 0;
```

```
	if(status==debounce_status)
		++debounce;
	else {
		debounce_status = status;
		debounce = 0;
	}
	if(debounce>=debounce_threshold && prev_status!=status){
		InputEvent event;
		unsigned char event_type=0;
		// Check pen press status
		for(int i=0;i<16;++i){
			unsigned char pressed = (status>>i)%2;
			if(pressed!=((prev_status>>i)%2)){
				// Up/Down
				if(i==0||i==1){
					if(pad_dir==2)
						continue;
					pad_dir=pressed?1:0;
				}
				// Left/Right
				if(i==2||i==3){
					if(pad_dir==1)
						continue;
					pad_dir=pressed?2:0;
				}
				event_type=i;
				event.type = InputEventType(event_type);
				// Handle pen input event
				if(pressed)
					input_handler_->handle_input_press(event);
				else
					input_handler_->handle_input_release(event);
			}
		}

		prev_status = status;
	}
}

void XboxController::print_xbox_message(XboxMessage msg)
{
	printf("Header: 0x%04x\n",msg.header);
	printf("Buttons: 0x%04x\n",msg.buttons);
	printf("lt_level: %u\n",msg.lt_level);
	printf("rt_level: %u\n",msg.rt_level);
	printf("lstick_dir_right: %d\n",msg.lstick_dir_right);
	printf("lstick_dir_up: %d\n",msg.lstick_dir_up);
	printf("rstick_dir_right: %d\n",msg.rstick_dir_right);
	printf("rstick_dir_up: %d\n\n",msg.rstick_dir_up);
}
```

5.21 display/display.h

```C/C++
#ifndef _DISPLAY_HH
#define _DISPLAY_HH

#include <cstdint>
#include <stdint.h>
#include <string>
#include <sys/types.h>
#include <utility>
#include <vector>
#define VGA_DRIVER_FILENAME "/dev/vga_ball"

struct BBOX{
	uint16_t left=0xffff,up=0xffff,right=0xffff,bottom=0xffff;
	BBOX expand_by(BBOX &bbox);
	BBOX(){};
	BBOX(uint16_t _left,uint16_t _up,uint16_t _right, uint16_t _bottom):
		left(_left), up(_up), right(_right), bottom(_bottom){}
	bool in(uint16_t x, uint16_t y);
	void reset();
};

struct GMKDisplayMessageInfo{
	std::string content="";
	uint16_t group=0;
	uint16_t group_idx=0;
	uint16_t index=0;
	uint16_t up=0xffff, down=0xffff, left=0xffff, right=0xffff;
	bool selectable=false;
	bool visible=false;
	bool disabled=false;
	BBOX bounding_box=BBOX(0xFFFF,0xFFFF,0xFFFF,0xFFFF);
};


class GMKDisplayMessageGroup{
public:
	uint16_t get_message_command(uint16_t index);
	uint16_t message_select_by_vga_xy(uint16_t vga_x,uint16_t vga_y);
	uint16_t next_message_by_direction(uint16_t current_message, int direction);
	uint16_t first_selectable_message();

	void update_message_selectable(uint16_t index, bool selectable, bool update_cache=false);
	void update_selectable_cache();
	void generate_visibility();
	void update_group_visibility(uint16_t group, bool visible);

	void display_selectable();

	inline static bool is_board_selected(uint16_t message_index){return !(message_index&0x00ff);}

	std::vector<GMKDisplayMessageInfo> messages;
```

```
        std::vector<GMKDisplayMessageInfo> selectable;
        std::vector<uint16_t> group_visibility;

protected:
        uint16_t next_board_by_direction(uint16_t board_message,int direction);

        BBOX selected_area_;
};


class GMKDisplay{
        public:
                GMKDisplay()
                        {hint_.first=15;hint_.second=15;};
                // piece 0:No piece, 1:white piece, 2:black piece
                bool update_piece_info(int x,int y,int piece, int current=1, bool sync=true);
                bool update_select(int board_x,int board_y, bool sync=true);
                bool update_select(uint16_t message_index, bool sync=true);
                bool update_register(unsigned int index,uint16_t val, bool sync=true);
                uint16_t get_register(uint16_t index){return params_[index];}
                bool update_touchpad_cursor(uint16_t vga_x, uint16_t vga_y, bool visible=true, bool
sync=true);
                bool show_menu();
                bool show_board(bool clear);
                bool clear_board();
                bool sync();
                bool open_display();

                bool set_player_piece(bool top_black,bool sync=true);
                bool set_turn_mark(bool top_turn,bool sync=true);
                bool switch_turn_mark(bool sync=true);

                void set_message_group(GMKDisplayMessageGroup *group){msg_group_ = group;}
                bool update_message_visibility(uint16_t index, bool visible, bool sync=true);
                bool update_group_visibility(uint16_t group, bool visible, bool sync=true);
                bool update_group_visibility(uint16_t group, uint16_t val, bool sync=true);

                bool update_p2_profile(int profile, bool sync=true);
                void show_game_result(int result, bool show=true);
                void show_confirm_message(bool show=true);
                void show_scanning_message(bool show=true);
                void show_hint(int x_index,int y_index);

                // Play sound
                // 0 - victory
                // 1 - defeat
                // 2 - menu
                // 3 - move
                void play_sound(int index);

        protected:

                uint16_t params_[8];
                int vga_gomoku_fd_=-1;
```

```
            const char *dev_name_=VGA_DRIVER_FILENAME;
            std::pair<uint16_t, uint16_t> hint_;

            GMKDisplayMessageGroup *msg_group_;
};

#endif
```

5.22 display/display.cpp

```C/C++
#include "display.h"
#include <cstddef>
#include <cstdint>
#include <cstdio>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <unistd.h>
#include <iostream>
#include <stdint.h>

#include "../vga_kmod/vga_gomoku.h"

bool BBOX::in(uint16_t x, uint16_t y)
{
      //printf("left %u, x %u, right %u, up %u, y %u, bottom %u\n",left,x,right,up,y,bottom);
      return (left<=x && x<=right && up<=y && y<= bottom);
}
void BBOX::reset()
{
      left=0xffff;
}

BBOX BBOX::expand_by(BBOX &bbox)
{
      BBOX ret;
      if(left==0xffff){
            ret = bbox;
            return ret;
      }
      ret.left = bbox.left<left?bbox.left:left;
      ret.right = bbox.right>right?bbox.right:right;
      ret.up = bbox.up<up?bbox.up:up;
      ret.bottom = bbox.bottom>bottom?bbox.bottom:bottom;
      return ret;
}

uint16_t GMKDisplayMessageGroup::message_select_by_vga_xy(uint16_t vga_x, uint16_t vga_y)
{
      if(messages[0].visible && !messages[0].disabled && vga_x>=4 && vga_x<499 && vga_y >=7 &&
vga_y<472){
```

```
                uint16_t piece_x, piece_y;
                piece_x=(vga_x-4)/33;
                piece_y=(vga_y-7)/31;
                return (piece_x<<12)|(piece_y<<8);
        }
        if(!selected_area_.in(vga_x, vga_y))
                return 0xFFFF;
        for(auto msg:selectable){
                if(msg.bounding_box.in(vga_x, vga_y))
                        return msg.index|0xFF00;
        }
        return 0xFFFF;
}

void GMKDisplayMessageGroup::update_selectable_cache()
{
        selectable.clear();
        selected_area_.reset();
        for(GMKDisplayMessageInfo &msg: messages){
                if(msg.selectable && msg.visible && !msg.disabled){
                        selectable.push_back(msg);
                        selected_area_ = selected_area_.expand_by(msg.bounding_box);
                        printf("%s selectable\n",msg.content.c_str());
                }
        }
}

uint16_t GMKDisplayMessageGroup::first_selectable_message()
{
        update_selectable_cache();
        if(selectable.empty())
                return 0xffff;
        return selectable[0].index;
}

uint16_t GMKDisplayMessageGroup::next_board_by_direction(uint16_t board_message,int direction)
{
        uint16_t x = board_message>>12;
        uint16_t y = (board_message>>8)&0xf;
        if(direction==0){ // UP
                if(y>0)
                        --y;
        }
        else if(direction==1){ // DOWN
                if(y<14)
                        ++y;
        }
        else if(direction==2){ // LEFT
                if(x>=0)
                        --x;
        }
        else if(direction==3){ // RIGHT
                if(x==14)
                        return messages[0].right;
```

```
            ++x;
        }
        return (x<<12)|(y<<8);
}

uint16_t GMKDisplayMessageGroup::next_message_by_direction(uint16_t current_message, int direction)
{
        uint16_t next = current_message;
        if(is_board_selected(current_message))
                return next_board_by_direction(current_message, direction);
        if(direction==0){ // UP
                // Iterate next message
                next = messages[current_message].up;
                while(next!=0xffff){ // 0xffff - no next message
                        if(is_board_selected(next)){
                                if(messages[0].selectable && messages[0].visible && !messages[0].disabled)
                                        // Return Piece board
                                        return next;
                                return current_message;
                        }
                        if(messages[next].selectable && messages[next].visible && !messages[next].disabled)
                                // Return Selectable message
                                return next;
                        next = messages[current_message].up;
                }
        }
        else if(direction==1){ // DOWN
                // Iterate next message
                next = messages[current_message].down;
                while(next!=0xffff){ // 0xffff - no next message
                        if(is_board_selected(next)){
                                if(messages[0].selectable && messages[0].visible && !messages[0].disabled)
                                        // Return Piece board
                                        return next;
                                return current_message;
                        }
                        if(messages[next].selectable && messages[next].visible && !messages[next].disabled)
                                // Return Selectable message
                                return next;
                        next = messages[current_message].down;
                }
        }
        else if(direction==2){ // LEFT
                // Iterate next message
                next = messages[current_message].left;
                while(next!=0xffff){ // 0xffff - no next message
                        if(is_board_selected(next)){
                                if(messages[0].selectable && messages[0].visible && !messages[0].disabled)
                                        // Return Piece board
                                        return next;
                                return current_message;
                        }
                        if(messages[next].selectable && messages[next].visible && !messages[next].disabled)
                                // Return Selectable message
```

```
                        return next;
                    next = messages[current_message].left;
                }
        }
        else if(direction==3){ // RIGHT
                // Iterate next message
                next = messages[current_message].right;
                while(next!=0xffff){ // 0xffff - no next message
                    if(is_board_selected(next)){
                        if(messages[0].selectable && messages[0].visible && !messages[0].disabled)
                            // Return Piece board
                            return next;
                        return current_message;
                    }
                    if(messages[next].selectable && messages[next].visible && !messages[next].disabled)
                        // Return Selectable message
                        return next;
                    next = messages[current_message].right;
                }
        }
        // No selectable message, return current
        return current_message;
}

uint16_t GMKDisplayMessageGroup::get_message_command(uint16_t index)
{
        if(is_board_selected(index)) // Selected a board position
                // Return make move command
                return 0;
        return index & 0x00FF;
}

void GMKDisplayMessageGroup::update_message_selectable(uint16_t index, bool selectable, bool update)
{
        messages[index].selectable = selectable;
        if(update)
                update_selectable_cache();
}

void GMKDisplayMessageGroup::generate_visibility()
{
        int group=0;
        group_visibility.push_back(0);
        for(auto msg:messages){
                if(msg.group!=group){
                    group_visibility.push_back(msg.group<<10);
                    group = msg.group;
                }
                group_visibility[group]|=(msg.visible<<msg.group_idx);
        }
}

void GMKDisplayMessageGroup::update_group_visibility(uint16_t group, bool visible)
{
```

```
	if(visible){
		group_visibility[group] |= 0x03ff;
		for(GMKDisplayMessageInfo &msg:messages){
			if(msg.group==group)
				msg.visible = true;
		}
	}
	else{
		group_visibility[group] &= ~(0x03ff);
		for(GMKDisplayMessageInfo &msg:messages){
			if(msg.group==group)
				msg.visible = false;
		}
	}
	update_selectable_cache();
}

void GMKDisplayMessageGroup::display_selectable()
{
	printf("Selectable\n");
	for(auto msg:messages){
		printf("%d:%s\n",msg.index,msg.content.c_str());
	}
	printf("\n");
}

bool GMKDisplay::update_register(unsigned int index,uint16_t val, bool sync)
{
	if(index>8)
		return false;
	params_[index]=val;
	if(sync)
		return this->sync();
	return true;
}

bool GMKDisplay::update_p2_profile(int profile, bool sync)
{
	if(profile==0){     // AI Profile
		printf("AI profile\n");
		update_group_visibility(3,(uint16_t)0b101,false);
		params_[0]&=~(1<<4);
	}else if(profile==1){ // P2 Profile
		printf("P2 profile\n");
		update_group_visibility(3,(uint16_t)0b011,false);
		params_[0]|=(1<<4);
	}
	else if(profile==2){  // No player
		printf("No Player\n");
		update_group_visibility(3,(uint16_t)0b001,false);
		params_[0]|=(1<<4);
	}
	if(sync)
		return this->sync();
```

```
	return true;
}

bool GMKDisplay::update_message_visibility(uint16_t index, bool visible, bool sync)
{
	GMKDisplayMessageInfo &msg = msg_group_->messages[index];
	msg.visible = visible;
	uint16_t group_idx = msg.group_idx;
	uint16_t &param = msg_group_->group_visibility[msg.group];

	if(visible)
		param |= (1<<group_idx);
	else
		param &= ~(1<<group_idx);

	params_[3] = param;
	if(sync)
		return this->sync();
	return true;
}

bool GMKDisplay::update_group_visibility(uint16_t group, uint16_t val, bool sync)
{
	bool visible[10];
	for(size_t i=0;i<10;++i)
		visible[i]=(val>>i)&1;
	val = (val & 0x03FF) | (group << 10);
	msg_group_->group_visibility[group] = val;
	for(GMKDisplayMessageInfo &msg:msg_group_->messages){
		if(msg.group==group)
			msg.visible=visible[msg.group_idx];
	}

	params_[3] = val;
	printf("%s by val:%04x\n",__func__,params_[3]);
	if(sync)
		return this->sync();
	return true;
}

bool GMKDisplay::update_group_visibility(uint16_t group, bool visible, bool sync)
{
	msg_group_->update_group_visibility(group, visible);

	params_[3] = msg_group_->group_visibility[group];
	printf("Visibility\n");
	for(size_t i=0;i<msg_group_->group_visibility.size();++i){
		printf("0x%04x ",msg_group_->group_visibility[i]);
	}
	printf("\n\n");
	if(sync)
		return this->sync();
	return true;
}
```

```
bool GMKDisplay::update_touchpad_cursor(uint16_t vga_x, uint16_t vga_y,bool visible, bool sync)
{
	//printf("update touchpad cursor: x %u, y %u, visible %d\n",x,y,visible);
	params_[0] = ((params_[0] & 0xFFFD) | (visible<<1));
	//printf("param0 0x%04x\n",params_[0]);
	params_[5] = vga_x;
	params_[6] = vga_y;

	// If cursor visible, update select
	if(visible){
		uint16_t index = msg_group_->message_select_by_vga_xy(vga_x, vga_y);
		update_select(index,false);
	}

	if(sync)
		return this->sync();
	return true;
}

bool GMKDisplay::show_menu()
{
	play_sound(2);
	// Dialog set to 1
	params_[0]|=1;
	update_group_visibility(0, false);
	update_group_visibility(1, true);
	update_group_visibility(2, false);
	update_group_visibility(3, false);
	update_group_visibility(4, false);
	return update_group_visibility(5, false);
}
bool GMKDisplay::show_board(bool clear)
{
	bool ret=true;
	// Dialog set to 0
	params_[0]&=(~1);
	update_group_visibility(0, true);
	update_group_visibility(1, false);
	update_group_visibility(2, true);
	update_group_visibility(3, true);
	update_group_visibility(4, false);
	update_group_visibility(5, false);
	// Select board center
	ret = update_select(7, 7,false);
	if(clear)
		return clear_board();
	return ret;
}

bool GMKDisplay::open_display()
{
	for(int i=0;i<8;++i)
		params_[i]=0;
```

```
	params_[2]=0x77ff;
	if ((vga_gomoku_fd_= open(this->dev_name_, O_RDWR)) == -1) {
		fprintf(stderr, "could not open %s\n", this->dev_name_);
		return false;
	}
	return true;
}

bool GMKDisplay::update_piece_info(int x,int y, int piece, int current, bool sync)
{
	if(hint_.first!=15&&hint_.second!=15){
		// Hide hint
		params_[1] = hint_.second|(hint_.first<<4);
		hint_.first=15;
		hint_.second=15;
		this->sync();
		return update_piece_info(x, y, piece,sync);
	}
	if(piece==3) // Highlighted mark
		params_[1] = y|(x<<4)|(1<<8);
	else
		params_[1] = y|(x<<4)|(piece<<9);
	if(current)
		params_[2] = (params_[2]&0xff00)|(y|(x<<4));
	printf("piece_info3:0x%04x\n",params_[1]);
	printf("piece_info_params_[2]:%04x\n",params_[2]);
	if(sync)
		return this->sync();
	return true;
}

bool GMKDisplay::update_select(int x,int y, bool sync)
{
	// Update board selection
	params_[2] = ((x<<4)|y)<<8|(params_[2]&0x00ff);
	printf("select_params_[2]:%04x\n",params_[2]);
	// Unselect message
	params_[4] = 0;
	if(sync)
		return this->sync();
	return true;
}

bool GMKDisplay::clear_board()
{
	// Reset board command
	params_[0] = 0xffff;
	// Piece Info register
	params_[1] = 0;
	// Selected register
	params_[2] = 0xFFFF;
	bool ret = this->sync();
	params_[0] = 0x0000;
	return ret;
```

```
}

bool GMKDisplay::sync()
{
	if (ioctl(vga_gomoku_fd_, VGA_GOMOKU_WRITE, params_)){
		perror("ioctl(VGA_GOMOKU_WRITE) failed");
		return false;
	}
	return true;
}

bool GMKDisplay::update_select(uint16_t message_index, bool sync)
{
	if(message_index&0xFF00)
		// Update board selection
		params_[2] = (message_index&0xff00)|(params_[2]&0x00ff);
	else
		params_[2] = (0xff00)|(params_[2]&0x00ff);
	// Update message selection
	params_[4] = message_index&0x00ff;
	if(sync)
		return this->sync();
	return true;
}

bool GMKDisplay::set_player_piece(bool top_black,bool sync)
{

	params_[0] = ((params_[0] & ~(top_black<<2)) | (top_black<<2));
	if(sync)
		return this->sync();
	return true;
}

bool GMKDisplay::set_turn_mark(bool top_turn,bool sync)
{
	params_[0] = ((params_[0] & ~(top_turn<<3)) | (top_turn<<3));
	if(sync)
		return this->sync();
	return true;
}

bool GMKDisplay::switch_turn_mark(bool sync)
{
	params_[0] ^= (1 << 3);
	if(sync)
		return this->sync();
	return true;
}

// 0 - YOU WIN
// 1 - YOU LOSE
// 2 - P1 WIN
// 3 - P2 WIN
```

```
void GMKDisplay::show_game_result(int result, bool show)
{
	if(!show){
		for(GMKDisplayMessageInfo &msg:msg_group_->messages){
			msg.disabled=false;
		}
		msg_group_->update_selectable_cache();
		update_register(0, get_register(0)&~(1<<5));
		return;
	}
	update_group_visibility(4,(uint16_t)(1<<result));
	update_group_visibility(5,(uint16_t)(1));
	for(GMKDisplayMessageInfo &msg:msg_group_->messages){
		msg.disabled=true;
	}
	msg_group_->messages[20].disabled=false;
	GMKDisplayMessageInfo &msg = msg_group_->messages[20];
	printf("%s:selectable %d, visible %d, disabled
%d\n",msg.content.c_str(),msg.selectable,msg.visible,msg.disabled);
	msg_group_->update_selectable_cache();
	update_register(0, get_register(0)|(1<<5));
	if(result==1)
		play_sound(1);
	else
		play_sound(0);
}

void GMKDisplay::show_confirm_message(bool show)
{
	if(!show){
		for(GMKDisplayMessageInfo &msg:msg_group_->messages){
			msg.disabled=false;
		}
		msg_group_->update_selectable_cache();
		update_register(0, get_register(0)&~(1<<5));
		return ;
	}
	update_group_visibility(4,(uint16_t)(1<<4));
	update_group_visibility(5,(uint16_t)(0b110));
	for(GMKDisplayMessageInfo &msg:msg_group_->messages){
		msg.disabled=true;
	}
	msg_group_->messages[21].disabled=false;
	msg_group_->messages[22].disabled=false;
	GMKDisplayMessageInfo &msg = msg_group_->messages[20];
	printf("%s:selectable %d, visible %d, disabled
%d\n",msg.content.c_str(),msg.selectable,msg.visible,msg.disabled);
	msg_group_->update_selectable_cache();
	update_register(0, get_register(0)|(1<<5));
}

void GMKDisplay::show_scanning_message(bool show)
{
	if(!show){
```

```
            for(GMKDisplayMessageInfo &msg:msg_group_->messages){
                  msg.disabled=false;
            }
            msg_group_->update_selectable_cache();
            update_register(0, get_register(0)&~(1<<5));
            return;
      }
      update_group_visibility(4,(uint16_t)(1<<5));
      update_group_visibility(5,(uint16_t)(0b000));
      for(GMKDisplayMessageInfo &msg:msg_group_->messages){
            msg.disabled=true;
      }
      // EXIT
      msg_group_->messages[20].disabled=false;
      msg_group_->update_selectable_cache();
      // Enable message box
      update_register(0, get_register(0)|(1<<5));
}

void GMKDisplay::show_hint(int x_index,int y_index)
{
      hint_.first = x_index;
      hint_.second = y_index;
      params_[1] = y_index|(x_index<<4)|(1<<8);
      this->sync();
}

void GMKDisplay::play_sound(int index)
{
      params_[0] &= 0x00FF;
      this->sync();
      params_[0] |= (1<<(index+8));
      this->sync();
      usleep(50000);
      params_[0] &= 0x00FF;
      this->sync();
}
```

5.23 display/message_group_init.h

```
C/C++
#ifndef _MESSAGE_GROUP_INIT_HH
#define _MESSAGE_GROUP_INIT_HH
#include "display.h"
inline void init_message_group(GMKDisplayMessageGroup &group)
{
      GMKDisplayMessageInfo info[23];

      info[0].content="BOARD";
      info[0].group=0;
      info[0].group_idx=1;
```

```
info[0].index=0;
info[0].selectable=1;
info[0].bounding_box=BBOX(0,0,40,5);
info[0].right=7;
group.messages.push_back(info[0]);

info[1].content="GOMOKU";
info[1].group=1;
info[1].group_idx=0;
info[1].index=1;
info[1].selectable=0;
info[1].bounding_box=BBOX(128,128,512,208);
group.messages.push_back(info[1]);

info[2].content="START PVP";
info[2].group=1;
info[2].group_idx=1;
info[2].index=2;
info[2].selectable=1;
info[2].bounding_box=BBOX(248,250,392,270);
info[2].down=3;
group.messages.push_back(info[2]);

info[3].content="START PVE";
info[3].group=1;
info[3].group_idx=2;
info[3].index=3;
info[3].selectable=1;
info[3].bounding_box=BBOX(248,280,392,300);
info[3].up=2;
info[3].down=4;
group.messages.push_back(info[3]);

info[4].content="CREATE ROOM";
info[4].group=1;
info[4].group_idx=3;
info[4].index=4;
info[4].selectable=1;
info[4].bounding_box=BBOX(232,310,408,330);
info[4].up=3;
info[4].down=5;
group.messages.push_back(info[4]);

info[5].content="JOIN ROOM";
info[5].group=1;
info[5].group_idx=4;
info[5].index=5;
info[5].selectable=1;
info[5].bounding_box=BBOX(248,340,392,360);
info[5].up=4;
info[5].down=6;
group.messages.push_back(info[5]);

info[6].content="EXIT";
```

```
info[6].group=1;
info[6].group_idx=5;
info[6].index=6;
info[6].selectable=1;
info[6].bounding_box=BBOX(288,370,352,390);
info[6].up=5;
group.messages.push_back(info[6]);

info[7].content="Regret";
info[7].group=2;
info[7].group_idx=0;
info[7].index=7;
info[7].selectable=1;
info[7].bounding_box=BBOX(520,355,616,375);
info[7].down=8;
info[7].left=0xea00;
group.messages.push_back(info[7]);

info[8].content="HINT";
info[8].group=2;
info[8].group_idx=1;
info[8].index=8;
info[8].selectable=1;
info[8].bounding_box=BBOX(536,385,600,405);
info[8].up=7;
info[8].down=9;
info[8].left=0xeb00;
group.messages.push_back(info[8]);

info[9].content="RESIGN";
info[9].group=2;
info[9].group_idx=2;
info[9].index=9;
info[9].selectable=1;
info[9].bounding_box=BBOX(520,415,616,435);
info[9].up=8;
info[9].down=10;
info[9].left=0xec00;
group.messages.push_back(info[9]);

info[10].content="EXIT";
info[10].group=2;
info[10].group_idx=3;
info[10].index=10;
info[10].selectable=1;
info[10].bounding_box=BBOX(536,445,600,465);
info[10].up=9;
info[10].left=0xed00;
group.messages.push_back(info[10]);

info[11].content="P1";
info[11].group=3;
info[11].group_idx=0;
info[11].index=11;
```

```
info[11].selectable=0;
info[11].bounding_box=BBOX(573,142,605,162);
group.messages.push_back(info[11]);

info[12].content="P2";
info[12].group=3;
info[12].group_idx=1;
info[12].index=12;
info[12].selectable=0;
info[12].bounding_box=BBOX(573,309,605,329);
group.messages.push_back(info[12]);

info[13].content="AI";
info[13].group=3;
info[13].group_idx=2;
info[13].index=13;
info[13].selectable=0;
info[13].bounding_box=BBOX(573,309,605,329);
group.messages.push_back(info[13]);

info[14].content="YOUWIN";
info[14].group=4;
info[14].group_idx=0;
info[14].index=14;
info[14].selectable=0;
info[14].bounding_box=BBOX(129,187,385,227);
group.messages.push_back(info[14]);

info[15].content="YOULOSE";
info[15].group=4;
info[15].group_idx=1;
info[15].index=15;
info[15].selectable=0;
info[15].bounding_box=BBOX(125,187,381,227);
group.messages.push_back(info[15]);

info[16].content="P1WIN";
info[16].group=4;
info[16].group_idx=2;
info[16].index=16;
info[16].selectable=0;
info[16].bounding_box=BBOX(150,187,374,227);
group.messages.push_back(info[16]);

info[17].content="P2WIN";
info[17].group=4;
info[17].group_idx=3;
info[17].index=17;
info[17].selectable=0;
info[17].bounding_box=BBOX(125,187,381,227);
group.messages.push_back(info[17]);

info[18].content="AREYOUSURE";
info[18].group=4;
```

```
	info[18].group_idx=4;
	info[18].index=18;
	info[18].selectable=0;
	info[18].bounding_box=BBOX(150,190,358,210);
	group.messages.push_back(info[18]);

	info[19].content="SCANNING...";
	info[19].group=4;
	info[19].group_idx=5;
	info[19].index=19;
	info[19].selectable=0;
	info[19].bounding_box=BBOX(240,230,416,250);
	group.messages.push_back(info[19]);

	info[20].content="EXIT";
	info[20].group=5;
	info[20].group_idx=0;
	info[20].index=20;
	info[20].selectable=1;
	info[20].bounding_box=BBOX(220,241,284,261);
	group.messages.push_back(info[20]);

	info[21].content="YES";
	info[21].group=5;
	info[21].group_idx=1;
	info[21].index=21;
	info[21].selectable=1;
	info[21].right=22;
	info[21].bounding_box=BBOX(164,244,212,264);
	group.messages.push_back(info[21]);

	info[22].content="NO";
	info[22].group=5;
	info[22].group_idx=2;
	info[22].index=22;
	info[22].selectable=1;
	info[22].left=21;
	info[22].bounding_box=BBOX(311,244,343,264);
	group.messages.push_back(info[22]);

	group.generate_visibility();
	for(auto msg:group.messages){
		printf("%s\n",msg.content.c_str());
	}
	for(auto gp:group.group_visibility){
		printf("Vis: 0x%04x\n",gp);
	}
}

#endif //_MESSAGE_GROUP_INIT_HH
```