ECE241 Final Project Report

Tank Duels

Team Members:

Armando Bablanian (999384063)

Lyndon Chan (999875817)

Course: ECE241H1 F

Lab Section: PRA0104

Lab Station: 38

Due Date: December 2, 2013

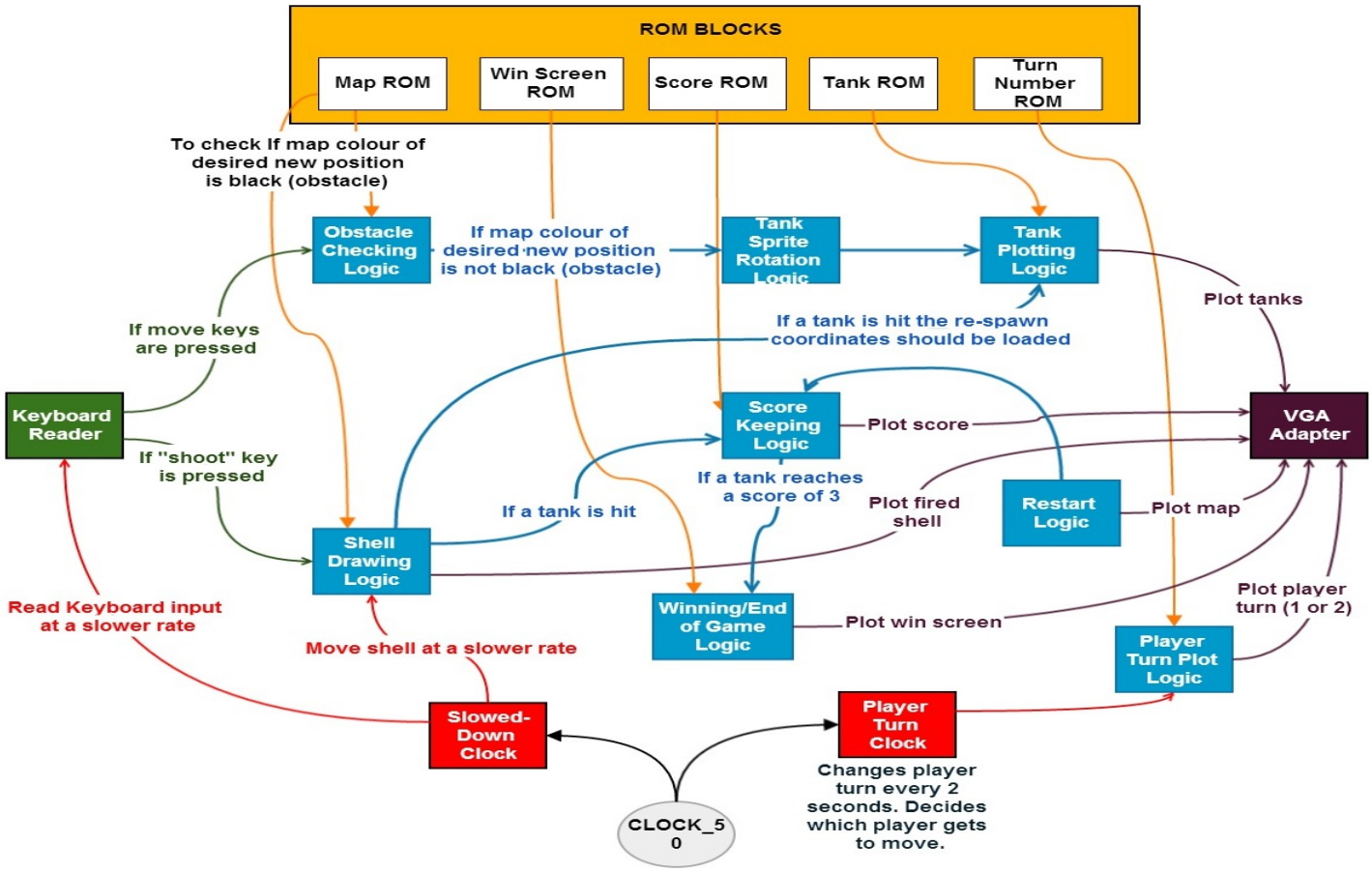
Table of Contents

I. Introduction

From the beginning, gaming has been tied to the latest computing advances. In 1958, one of the first computer games with a graphical display - *Tennis for Two* - was developed. Then in 1967, *Chase Game* - the first video game to be displayed on a television - was produced. In 1974, Kee Games released an arcade game called *Tank* - the first to store graphic data with IC-based ROM chips. In honour of *Tank*'s legacy, we have created *Tank Duels* - the FPGA version of the classic arcade game.

Similar to the original game, *Tank Duels* stores graphic data in ROM's and allows two players to shoot at each other in a maze. However, *Tank Duels* uses a PS/2 keyboard instead of four joysticks for input and an 8-colour VGA display instead of a black-and-white television screen. Additionally, since a PS/2 keyboard cannot simultaneously detect multiple keystrokes, Tank Duels operates with 2-second turns. Although what were then cutting-edge technologies in 1974 are now basic undergraduate concepts, we believe that building *Tank Duels* would push the boundaries of our knowledge of digital systems concepts as we apply them to a hands-on project.

II. The Design



Keyboard Reader Block

- Detects the current keypress on the PS/2 keyboard (if any key is currently pressed).

VGA Adapter Block

- Updates a pixel displayed on the monitor once it receives an indication that a pixel should be updated (i.e. written[1] to the screen) by changing the colour of the pixel at a given position.

ROM Block

- **Tank ROM:**
  - Stores the sprites of both Player 1's and Player 2's tanks and when given a pixel address, it outputs the colour of the requested pixel in the ROM.
- **Map ROM**
  - Stores the map background image and when given a pixel address, it outputs the colour of the requested pixel in the ROM.
- **Turn Number ROM**
  - Stores images of the turn numbers (a red "1" and a blue "2") and when given a pixel address, it outputs the colour of the requested pixel in the ROM.
- **Score ROM**
  - Stores images of three white numbers - "1", "2", "3" - to display player scores and when given a pixel address, it outputs the colour of the requested pixel in the ROM.
- **Win Screen ROM**
  - Stores the images of two win screen images - one for each player - and when given a pixel address, it outputs the colour of the requested pixel in the ROM.

Clock Block

- **Slowed-Down Clock:**
  - Produces a slowed-down clock pulse once every 0.01 s to slow down the sensitivity of keyboard input detection as well as the speed of the shell.
- **Player Turn Clock:**
  - Produces two symmetrical square waves (each the inverse of the other) with periods of two seconds - each one keeps track of whether it is that player's turn.

Gameplay Block

- **Obstacle Checking Logic**
  - Receives tank movement direction from keyboard input and checks all seven pixels in the immediate path of the incoming tank for obstacles.
  - If none of the seven pixels are black, then the tank is free to move in the desired direction - otherwise, if at

[1] In this document, the word "writing" refers to any sort of pixel update to the VGA adapter and comprises of both "plotting" (which refers to writing a pixel with a non-background colour) and "erasing" (which refers to writing a pixel with a background colour)

least one pixel is black, the tank does not move.

- **Tank Sprite Rotation Logic**
  - Based on the direction of last movement, decides which way to traverse the Tank ROM addresses in order to plot the tank facing the correct direction.
- **Tank Plotting Logic**
  - Erases the tank by writing a 7x7 pixel white square over its previous location.
  - Then the tank position is moved by one pixel in desired direction (if not blocked).
  - Plots the tank in the new position using pixel colours from Tank ROMs.
  - If a tank is hit, it is erased in previous location and plotted at its respawn position.
- **Shell Drawing Logic**
  - If a player presses their shoot key during their turn, a 1x1-pixel magenta shell is plotted at the front side of the tank.
  - The pixel is then continuously erased, moved in the shooting direction, and plotted until it hits a tank, an obstacle, or the edge of the game map - then, it is erased but not plotted.
- **Scorekeeping Logic**
  - Every time a player hits and "kills" their opponent, the killing player's score is incremented.
  - The new score is then plotted on top of the old score at the side of the screen under the scoring player's name using pixel colours from the score ROM (which holds the numbers "1", "2", "3").
  - The number "0" - which is the default score at the beginning of every game - is part of the background map and disappears as new scores are plotted on top of it.
- **Winning / End of Game Logic**
  - Once either player achieves a score of 3, a winning message is plotted on the screen for that player, using pixel colours from the Win Screen ROM.
  - Once plotting has finished, the game ends and waits for the user to restart.
- **Player Turn Plot Logic**
  - Alternates between plotting a number 1 or 2 every two seconds at the top-right of the screen to show which player's turn it is; the numbers are plotted using pixel colours from the Turn Number ROMs.
- **Restart Logic**
  - Once "KEY0" on the DE2 board is pressed, resets player positions to respawn points and scores to zero.

III. Report on Success

i. Tank Movement (see Appendix B.i)

- Once *Tank Duels* was loaded, the map background was displayed with both tanks invisible at their "respawn points" (top-left corner for Player 1, bottom-right corner for Player) until they made their first move.
- The turns alternated between players every two seconds and their scores were initialized to 0.
- During their turns, each player was able to move his/her tank across the map by pressing one key on the keyboard (for Player 1, "W" for up, "A" for left", "S" for down, and "D" for right; and for Player 2, "I" for up, "J" for left, "K" for down, and "L" for right) and was stopped by the other, the black obstacles and the edges of the map.

ii. Shell Shooting (see Appendix B.ii)

- Once the players pressed their shoot keys ("C" for Player 1 and "N" for Player 2) during their turn (even while moving), they fired a single magenta shell in the direction of their last movement.
- The tank disappeared once the shell hit any tank (including itself) and shells could only be fired if no other shells were currently moving.
- The magenta shell also disappeared once it hit an obstacle or the edge of the map.

iii. Killing and Respawning (see Appendix B.iii)

- Once either tank was killed, it respawned in its original corner of the screen.
- The killing player's score increased by one and his/her tank stayed in the same position.

iv. Winning (see Appendix B.iv)

- Once the score count of either player reached 3, gameplay stopped completely (no movement or shooting was possible) and a win message was displayed.

v. Restarting (see Appendix B.v)

- If "KEY0" was pressed at any point during or after gameplay on the DE2 board, the game restarted - both players were relocated to their respawn points and both player scores reverted to 0.

IV. What would we do differently

If we had another chance at doing this project, we would have made six key improvements:

1	**Document the logic of the project more**	● In many situations, a bug would appear after making a series of changes to a working version of the program. ● To pinpoint the change that caused the problem, we tried to make sense of all the program outputs, program structure, and potential explanations in our heads - this was exhausting and inefficient. ● Had we documented and tested even minor changes, recorded when a bug was reproducible, and worked out potential reasons for the bug on paper, we could have significantly simplified debugging.
2	**Draw more diagrams to help with visualization**	● Due to rapidly-changing code, diagrams describing the program were few and outdated. ● More diagrams would have greatly helped with visualizing where variables were being altered and how they were used. ● This would have streamlined the code and reduced the number of potential bugs.
3	**Make code more modular to improve code sharing**	● In order to more easily make changes to the entire program, we decided to use a single large module to contain all of the source code. ● This meant that changes to a single section would not necessitate tedious changes to the inputs and outputs of each module but combining separate versions of code was exhausting and error-prone. ● A good compromise would be to record which sections had changed and which had not after each revision - this would facilitate combining separate versions of code.
4	**Take appropriate breaks to reduce mental fatigue and frustration**	● Whenever we were hopelessly stuck on a problem, productivity quickly declined as we became frustrated and unable to think clearly. ● Once we implemented short breaks after unproductive periods - such as taking a walk, getting a drink, or eating a small snack - we were able to look at previously unsolvable problems from a new angle and solve them in a fraction of the time.
5	**Set more concrete meeting plans and internal deadlines**	● To combat the lack of coordination at team meetings, we should have: ○ Established the time, location, and agenda ahead of every meeting. ○ Brought in laptops to eliminate our reliance on vacant lab computers. ● We should also have set a fixed long-term schedule for general internal deadlines and a flexible short-term schedule for specific internal deadlines.
6	**Ask for advice when needed**	● At first, our team experienced much hesitation in asking others (whether classmates or the teaching assistants) for help whenever we were stuck. ● However, doing so more often would have given us fresh viewpoints and allowed us to solve problems more quickly .

APPENDIX A: Verilog Code

```
module final_project_working
(
	CLOCK_50,
	CLOCK_27,
	KEY,
	SW,
	PS2_DAT,
	PS2_CLK,
	AUD_ADCDAT,
	GPIO_0,
	GPIO_1,
	AUD_BCLK,
	AUD_ADCLRCK,
	AUD_DACLRCK,
	I2C_SDAT,
	AUD_XCK,
	AUD_DACDAT,
	I2C_SCLK,
	LEDR,
	LEDG,
	HEX0,
	HEX1,
	HEX2,
	HEX3,
	HEX4,
	HEX5,
	HEX6,
	HEX7,
	VGA_CLK,
	VGA_HS,
	VGA_VS,
	VGA_BLANK,
	VGA_SYNC,
	VGA_R,
	VGA_G,
	VGA_B
);

	input		CLOCK_50;
	input				CLOCK_27;
	input	[3:0]	KEY;
	input	[17:0]	SW;
	input			PS2_DAT;
	input			PS2_CLK;
```

```
input                           AUD_ADCDAT;
inout       [35:0]       GPIO_0;
inout       [35:0]       GPIO_1;
inout                           AUD_BCLK;
inout                           AUD_ADCLRCK;
inout                           AUD_DACLRCK;
inout                           I2C_SDAT;
output                  AUD_XCK;
output                  AUD_DACDAT;
output                  I2C_SCLK;
output      [17:0]       LEDR;
output      [7:0]  LEDG;
output      [6:0]        HEX0;
output      [6:0]  HEX1;
output      [6:0]  HEX2;
output      [6:0]  HEX3;
output      [6:0]  HEX4;
output      [6:0]  HEX5;
output      [6:0]  HEX6;
output      [6:0]  HEX7;
output          VGA_CLK;
output          VGA_HS;
output          VGA_VS;
output          VGA_BLANK;
output          VGA_SYNC;
output   [9:0]      VGA_R;
output   [9:0]      VGA_G;
output   [9:0]      VGA_B;

// TANK PRINTING VARIABLES UNIQUE TO PLAYER 1
// Immediately-previous tank location
integer x_1_old = 0;
integer y_1_old = 0;
// Instanteously-inputted tank location
integer x_1 = 0;
integer y_1 = 0;
// Saved previous tank location
integer x_1_old_const = 0;
integer y_1_old_const = 0;
// Saved new tank location
integer x_1_const = 0;
integer y_1_const = 0;
// Tank ROM counter variables
integer i_1;
integer j_1;
// Tank ROM address
integer ROM_address_1;
// Tank Size specifications
```

```
integer tankWidth_1 = 7;
integer tankHeight_1 = 7;
// Tank ROM output colour
wire [2:0] pixel_colour_1;
// Obstacle check counter
integer obstacleCheck_1;
// Flags whether Player 2 is blocked by obstacle
reg isReallyBlocked_1 = 0;

// TANK PRINTING VARIABLES UNIQUE TO PLAYER 2
// Immediately-previous tank location
integer x_2_old = 159-(7)+1-40;
integer y_2_old = 119-(7)+1;
// Instantaneously-inputted tank location
integer x_2 = 159-(7)+1-40;
integer y_2 = 119-(7)+1;
// Saved previous tank location
integer x_2_old_const = 159-(7)+1-40;
integer y_2_old_const = 119-(7)+1;
// Saved new tank location
integer x_2_const = 159-(7)+1-40;
integer y_2_const = 119-(7)+1;
// Tank ROM counter variables
integer i_2;
integer j_2;
// Tank ROM address
integer ROM_address_2;
// Tank Size specifications
integer tankWidth_2 = 7;
integer tankHeight_2 = 7;
// Tank ROM output colour
wire [2:0] pixel_colour_2;
// Obstacle check counter
integer obstacleCheck_2;
// Flags whether Player 2 is blocked by obstacle
reg isReallyBlocked_2 = 0;

// TANK PRINTING (i.e. ERASE / PLOT) VARIABLES SHARED BY BOTH PLAYERS
// State variables
reg [58:0] currState = STR;
reg [58:0] nextState;
// Reset variable
wire resetn = KEY[0];
// Flags whether to write (i.e. erase or plot) a pixel to VGA
reg isWritePixel;
// Position of pixel to be written to VGA
integer writeX;
integer writeY;
```

```
    // Colour of pixel to be written to VGA
    reg [2:0]colourReg;
    // PS/2 Keyboard Input Code
    wire [15:0] key_code;
    // Map ROM counter variables
    integer iMap;
    integer jMap;
    // Map ROM output colour
    wire [2:0] map_pixel_colour;

    assign LEDR[0] = has_P1_lost;
    assign LEDR[1] = has_P2_lost;
    assign LEDR[14:11] = currState[54:51];

    // SHELL STATES VARIABLES
    // Flags whether shell is moving
    reg [0:0] isShellLive = 0;
    // Shell position
    integer shellX;
    integer shellY;
    // Flags whether Player 1, 2 have been hit
    reg [0:0] is_P1_hit = 0;
    reg [0:0] is_P2_hit = 0;
    // Flags whether Player 1, 2 should respawn
reg [0:0] should_respawn_1 = 0;
    reg [0:0] should_respawn_2 = 0;
// Counts number of times Player 1, 2 have died
    integer P1_death_count = 0;
    integer P2_death_count = 0;
    // Flags whether Player 1 or 2 shooting
    reg [0:0] is_P1_shooting = 0;
    reg [0:0] is_P2_shooting = 0;
    // Flags whether Player 1 or 2 have lost
    reg [0:0] has_P1_lost = 0;
    reg [0:0] has_P2_lost = 0;
    // Stores direction of shell movement (equivalent to direction
    // of last tank movement), where 1 is Up, 2 is Left, 3 is Down, and 4 is Right
    integer shellDirection_1 = 3;
    integer shellDirection_2 = 1;

    // TURN STATES VARIABLES
    integer i_trn;
    integer j_trn;
    wire [2:0]turn1_colour;
    wire [2:0]turn2_colour;

    // SCORE-KEEPING STATES VARIABLES
    integer y_scr = 0;
```

```
        integer i_scr = 0;
        integer j_scr = 0;
        wire[2:0] SCR1_colour;
        wire[2:0] SCR2_colour;
        wire[2:0] SCR3_colour;

        // WIN STATES VARIABLES
        integer j_win = 0;
    integer i_win = 0;
        wire[2:0] win1_colour;
        wire[2:0] win2_colour;

        // Keyboard Reader Module
        keyboard_reader KEYBOARD (.CLOCK(CLOCK_50), .KEY(KEY), .SW(SW), .HEX0(HEX0), .HEX1(HEX1),
.HEX2(HEX2),
                                                                                .HEX3(HEX3),
.HEX4(HEX4), .HEX5(HEX5), .HEX6(HEX6), .HEX7(HEX7),

.key_code(key_code),
                                                                                .PS2_DAT(PS2_DAT),
.PS2_CLK(PS2_CLK), .GPIO_0(GPIO_0), .GPIO_1(GPIO_1));

        // VGA Adapter Module
        vga_adapter VGA(
                                        .resetn(resetn),
                                        .clock(CLOCK_50),
                                        .colour(colourReg),
                                        .x(writeX),
                                        .y(writeY),
                                        .plot(isWritePixel),
                                        /* Signals for the DAC to drive the monitor. */
                                        .VGA_R(VGA_R),
                                        .VGA_G(VGA_G),
                                        .VGA_B(VGA_B),
                                        .VGA_HS(VGA_HS),
                                        .VGA_VS(VGA_VS),
                                        .VGA_BLANK(VGA_BLANK),
                                        .VGA_SYNC(VGA_SYNC),
                                        .VGA_CLK(VGA_CLK));
                                        defparam VGA.RESOLUTION = "160x120";
                                        defparam VGA.MONOCHROME = "FALSE";
                                        defparam VGA.BITS_PER_COLOUR_CHANNEL = 1;
                                        defparam VGA.BACKGROUND_IMAGE = "gamemap.mif";

        // ROM Modules for Player 1, 2 Tank Sprites
        ROM_Player1 ROM_Player1_inst (.address ( ROM_address_1 ),

.clock( CLOCK_50 ),
```

```
                                                                                .q (
pixel_colour_1 )
                                                                                );
       ROM_Player2 ROM_Player2_inst (.address ( ROM_address_2 ),

.clock( CLOCK_50 ),
                                                                                .q (
pixel_colour_2 )
                                                                                );

       // ROM Module for Map Background Image
       NEWMAP_ROM NEWMAP_ROM_inst (.address((160 * jMap) + iMap),
                                                                    .clock(CLOCK_50),
                                                                    .q(map_pixel_colour));

       // ROM Modules for Player Turn Number Images
       ROM_NEWNUM1 ROM_PLYRNUM1_inst1 (.address((6 * j_trn) + i_trn),

.clock(CLOCK_50),

.q(turn1_colour)
                                                                                );
       ROM_NEWNUM2 ROM_PLYRNUM2_inst2 (.address((6 * j_trn) + i_trn),

.clock(CLOCK_50),

.q(turn2_colour)
                                                                                );

       // ROM Modules for Player 1, 2 Score-keeping Displays
       ROM_SCR1 ROM_SCR1_inst (.address((5 * j_scr) + i_scr),
                                    .clock (CLOCK_50),
                                    .q(SCR1_colour)
                                    );

   ROM_SCR2 ROM_SCR2_inst (.address((5 * j_scr) + i_scr),
                                    .clock (CLOCK_50),
                                    .q(SCR2_colour)
                                    );

   ROM_SCR3 ROM_SCR3_inst (.address((5 * j_scr) + i_scr),
                                    .clock (CLOCK_50),
                                    .q(SCR3_colour)
                                    );

       // ROM Modules for Player 1, 2 Win Screens
       ROM_WIN1 ROM_WIN1_inst (.address((80 * j_win) + i_win),
                                                                    .clock (CLOCK_50),
```

```
                                                                    .q(win1_colour)
                                                                    );
   ROM_WIN2 ROM_WIN2_inst (.address((80 * j_win) + i_win),
                                                                           .clock (CLOCK_50),
                                                                           .q(win2_colour)
                                                                           );

       parameter   // Reset States
                                    RST =
59'b00000000000000000000000000000000000000000000000000000000001,
                                    RSTA=
59'b00000000000000000000000000000000000000000000000000000000010,
                                    RSTB=
59'b00000000000000000000000000000000000000000000000000000000100,
                                    RSTC=
59'b00000000000000000000000000000000000000000000000000000001000,

                                    // Turn Display States
                                    STR =
59'b00000000000000000000000000000000000000000000000000000010000,
               trnD= 59'b00000000000000000000000000000000000000000000000000000100000,
               trnE= 59'b00000000000000000000000000000000000000000000000000001000000,
               trnE1=59'b00000000000000000000000000000000000000000000000000010000000,
               trnF= 59'b00000000000000000000000000000000000000000000000000100000000,

                                    // Crossroads State 1
               A =   59'b00000000000000000000000000000000000000000000000001000000000,

                                    // Player 1 Obstacle Detection States
                                    L =
59'b00000000000000000000000000000000000000000000000010000000000,
                                    wL=
59'b00000000000000000000000000000000000000000000000100000000000,
                                    aL=
59'b00000000000000000000000000000000000000000000001000000000000,
                                    sL=
59'b00000000000000000000000000000000000000000000010000000000000,
                                    dL=
59'b00000000000000000000000000000000000000000000100000000000000,
                                    LW=
59'b00000000000000000000000000000000000000000001000000000000000,
                                    LA=
59'b00000000000000000000000000000000000000000010000000000000000,
                                    LS=
59'b00000000000000000000000000000000000000000100000000000000000,
                                    LD=
59'b00000000000000000000000000000000000000001000000000000000000,
```

```
                                    // Player 2 Obstacle Detection States
                                    M =
59'b0000000000000000000000000000000000000001000000000000000000,
                                    iM=
59'b0000000000000000000000000000000000000010000000000000000000,
                                    jM=
59'b0000000000000000000000000000000000000100000000000000000000,
                                    kM=
59'b0000000000000000000000000000000000001000000000000000000000,
                                    lM=
59'b0000000000000000000000000000000000010000000000000000000000,
                                    MI=
59'b0000000000000000000000000000000000100000000000000000000000,
                                    MJ=
59'b0000000000000000000000000000000001000000000000000000000000,
                                    MK=
59'b0000000000000000000000000000000010000000000000000000000000,
                                    ML=
59'b0000000000000000000000000000000100000000000000000000000000,

                                    // Player 1 Erase / Plot States
                B0=   59'b0000000000000000000000000000001000000000000000000000000000,
                                    B =
59'b0000000000000000000000000000010000000000000000000000000000,
                                    B1=
59'b0000000000000000000000000000100000000000000000000000000000,
                                    C =
59'b0000000000000000000000000001000000000000000000000000000000,
                                    D =
59'b0000000000000000000000000010000000000000000000000000000000,
                                    E =
59'b0000000000000000000000000100000000000000000000000000000000,
                                    E1=
59'b0000000000000000000000001000000000000000000000000000000000,
                                    F =
59'b0000000000000000000000010000000000000000000000000000000000,

                                    // Player 2 Erase / Plot States
                                    G0=
59'b0000000000000000000000100000000000000000000000000000000000,
                                    G1=
59'b0000000000000000000001000000000000000000000000000000000000,
                                    G =
59'b0000000000000000000010000000000000000000000000000000000000,
                                    H =
59'b0000000000000000000100000000000000000000000000000000000000,
                                    I =
59'b0000000000000000001000000000000000000000000000000000000000,
```

```
                                J =
59'b000000000000000010000000000000000000000000000000000000000000000,
                                J1=
59'b000000000000000010000000000000000000000000000000000000000000000,
                                K =
59'b000000000000000010000000000000000000000000000000000000000000000,

                                //Shell States
                                P =
59'b000000000000001000000000000000000000000000000000000000000000000,
                                shA=
59'b000000000000010000000000000000000000000000000000000000000000000,
                                shB=
59'b000000000000100000000000000000000000000000000000000000000000000,
                                shC=
59'b000000000001000000000000000000000000000000000000000000000000000,
                                shD=
59'b000000000010000000000000000000000000000000000000000000000000000,
                                shB1=
59'b000000001000000000000000000000000000000000000000000000000000000,
                                shE=
59'b000000010000000000000000000000000000000000000000000000000000000,

                                //SCORE States
                                SCR=
59'b000000100000000000000000000000000000000000000000000000000000000,
                                SCRA=
59'b000001000000000000000000000000000000000000000000000000000000000,
                                SCRB=
59'b000010000000000000000000000000000000000000000000000000000000000,
                                SCRC=
59'b000100000000000000000000000000000000000000000000000000000000000,

                                // Win States
                                WINA=
59'b001000000000000000000000000000000000000000000000000000000000000,
                                WINB=
59'b010000000000000000000000000000000000000000000000000000000000000,
                                WINC=
59'b100000000000000000000000000000000000000000000000000000000000000,
                                END =
59'b100000000000000000000000000000000000000000000000000000000000000;

        // Slow Clock Logic - this creates a clock signal with a positive edge every 500,000 50-MHz
cycles
       reg [25:0] clock_count = 26'd0;
       reg slow_clock = 0;
       always @ (posedge CLOCK_50)
```

```
	begin
		if (clock_count == 26'd500000)
		begin
		   slow_clock <= 1;
		   clock_count <= 26'd0;
		end
		else
		begin
			slow_clock <= 0;
			clock_count <= clock_count + 1;
		end
	end

	// Player Turn Clock Logic (implements a clock to determine whose turn it is)
	reg is_P1_turn = 0;
	reg is_P2_turn = 0;
	integer turn_count = 0;
	always @ (posedge CLOCK_50)
	begin
		if (turn_count == 0)
		begin
		   is_P1_turn = 1;
		   is_P2_turn = 0;
		end
		else if (turn_count == 100000000)
		begin
		   is_P1_turn = 0;
		   is_P2_turn = 1;
		end
		else if (turn_count == 200000000)
		begin
			turn_count = -1;
		end
		turn_count = turn_count + 1;
	end

	// Plot-control Logic (determines when a pixel is written, whether to plot or erase, and where
to write)
	always @ (posedge CLOCK_50)
	begin
		// Controlling when to write pixel in reset states
		if (currState == RST)
      begin
         writeX = 0;
         writeY = 0;
      end
      else if (currState == RSTA)
      begin
```

```
         isWritePixel <= 1;
                     colourReg <= map_pixel_colour;
      end
              else if (currState == RSTB)
              begin
                     isWritePixel <= 0;
                     writeX = iMap;
                     writeY = jMap;
              end
              //Controlling when to write pixel in turn-display states
              else if (currState == trnE)
              begin
                     isWritePixel <= 1;
                     if (is_P1_turn == 1)begin
                            colourReg <= turn1_colour;
                     end else if (is_P2_turn == 1)begin
                            colourReg <= turn2_colour;
                     end
              end
              else if (currState == trnE1)
              begin
                 isWritePixel <= 0;
                 writeX = 137 + i_trn;
                 writeY = 23 + j_trn;
              end
              // Controlling when to write pixel in tank erase states
              else if (currState == B0)
              begin
                     writeX = x_1_old_const;
                     writeY = y_1_old_const;
              end
              else if (currState == G0)
              begin
                     writeX = x_2_old_const;
                     writeY = y_2_old_const;
              end
              else if (currState == B || currState == G)
              begin
                     // Only write erase pixel IF the desired location is not an obstacle AND has
either been hit or is moving (else, no need to erase)
                     if (map_pixel_colour != 3'b000 && map_pixel_colour != 3'b110)
                     begin
                            if (is_P1_hit == 1 || is_P2_hit == 1 || x_1_const != x_1_old_const ||
y_1_const != y_1_old_const || x_2_const != x_2_old_const || y_2_const != y_2_old_const)
                            begin
                                   isWritePixel <= 1;
                            end
                     end
```

```
        else
        begin
                isWritePixel <= 0;
        end
        colourReg <= 3'b111;
end
else if (currState == B1)
begin
        isWritePixel <= 0;
        writeX = x_1_old_const + i_1;
        writeY = y_1_old_const + j_1;
end
else if (currState == G1)
begin
        isWritePixel <= 0;
        writeX = x_2_old_const + i_2;
        writeY = y_2_old_const + j_2;
end
// Controlling when to write pixel in tank plot states
else if (currState == E || currState == J)
begin
        isWritePixel <= 1;
        if (currState == E)
        begin
                colourReg <= pixel_colour_1;
        end
        else if (currState == J)
        begin
                colourReg <= pixel_colour_2;
        end
end
else if (currState == E1)
begin
        isWritePixel <= 0;
        writeX = x_1_const + i_1;
        writeY = y_1_const + j_1;
end
else if (currState == J1)
begin
        isWritePixel <= 0;
        writeX = x_2_const + i_2;
        writeY = y_2_const + j_2;
end
// Controlling when to write pixel in shell states
else if (currState == shB || currState == shC)
begin
        // Only write pixel IF the desired location is not an obstacle
        if (map_pixel_colour != 3'b000 && map_pixel_colour != 3'b110)
```

```
                    begin
                           if ((is_P1_shooting == 1 && isReallyBlocked_1 == 0) || (is_P2_shooting == 1
&& isReallyBlocked_2 == 0))
                           begin
                                  isWritePixel <= 1;
                           end
                    end
                    else
                    begin
                           isWritePixel <= 0;
                    end
                    if (currState == shB) begin
                           colourReg <= 3'b101;
                    end
                    else if (currState == shC) begin
                           colourReg <= 3'b111;
                    end
             end
             else if (currState == shB1)
             begin
                    isWritePixel <= 0;
                    writeX = shellX;
                    writeY = shellY;
             end
             // Controlling when to write pixel in score-keeping states
             else if (currState == SCRA)
             begin
                    if (is_P1_hit == 1)
                    begin
                           if (P1_death_count == 1)
                           begin
                                  colourReg <= SCR1_colour;
                           end
                           else if (P1_death_count == 2)
                           begin
                                  colourReg <= SCR2_colour;
                           end
                           else if (P1_death_count == 3)
                           begin
                                  colourReg <= SCR3_colour;
                           end
                    end
                    else if (is_P2_hit == 1)
                    begin
                           if (P2_death_count == 1)
                           begin
                                  colourReg <= SCR1_colour;
                           end
```

```
                        else if (P2_death_count == 2)
                        begin
                                colourReg <= SCR2_colour;
                        end
                        else if (P2_death_count == 3)
                        begin
                                colourReg <= SCR3_colour;
                        end
                end
                isWritePixel <= 1;
        end
        else if (currState == SCRB)
        begin
                isWritePixel <= 0;
                writeX = 138 + i_scr;
                writeY = y_scr + j_scr;
        end
        // Controlling when to write pixel in Win Screen states
        else if (currState == WINA)
        begin
                isWritePixel <= 1;
                if (has_P2_lost == 1)begin //player 1 won
                        colourReg <= win1_colour;
                end else if (has_P1_lost == 1)begin //player 2 won
                        colourReg <= win2_colour;
                end
        end
        else if (currState == WINB)
        begin
                isWritePixel <= 0;
                writeX = 40 + i_win;
                writeY = 35 + j_win;
        end
        // If in any other states, reset the variables
        else
        begin
                isWritePixel <= 0;
                writeX = 0; // For safety
                writeY = 0; // For safety
        end
    end

// Tank Sprite Rotation Logic (sets appropriate ROM-reading counters for tank sprite .mif files)
always @ (*)
begin
        // Player 1 facing up
        if (shellDirection_1 == 1)
        begin
```

```
                ROM_address_1 = (tankWidth_1 * tankWidth_1 - 1) - (tankWidth_1 * j_1 + i_1);
            end
            // Player 1 facing left
            else if (shellDirection_1 == 2)
            begin
                  ROM_address_1 = (tankWidth_1 * tankWidth_1 - 1) - (tankWidth_1 * i_1 +
(tankWidth_1 - j_1 - 1));
            end
            // Player 1 facing down
            else if (shellDirection_1 == 3)
            begin
                ROM_address_1 = (tankWidth_1 * j_1) + i_1;
            end
            // Player 1 facing right
            else if (shellDirection_1 == 4)
            begin
                ROM_address_1 = tankWidth_1 * i_1 + (tankWidth_1 - j_1 - 1);
            end

            // Player 2 facing up
            if (shellDirection_2 == 1)
            begin
                ROM_address_2 = (tankWidth_2 * tankWidth_2 - 1) - (tankWidth_2 * j_2 + i_2);
            end
            // Player 2 facing left
            else if (shellDirection_2 == 2)
            begin
                ROM_address_2 = (tankWidth_2 * tankWidth_2 - 1) - (tankWidth_2 * i_2 + (tankWidth_2 -
j_2 - 1));
            end
            // Player 2 facing down
            else if (shellDirection_2 == 3)
            begin
                ROM_address_2 = (tankWidth_2 * j_2) + i_2;
            end
            // Player 2 facing right
            else if (shellDirection_2 == 4)
            begin
                ROM_address_2 = tankWidth_2 * i_2 + (tankWidth_2 - j_2 - 1);
            end
      end

      // Keyboard Input Logic (reads keyboard input and manages how the tank should move)
      always @ (posedge CLOCK_50)
      begin
            if (currState == RST)
            begin
                  x_1 = 0;
```

```
                        y_1 = 0;
                        x_1_old = 0;
                        y_1_old = 0;
                        x_2 = 159-(7)+1-40;
                        y_2 = 119-(7)+1;
                        x_2_old = 159-(7)+1-40;
                        y_2_old = 119-(7)+1;
                end
                // Reset tank coordinates to respawn points if respawn is imminent
                if (should_respawn_1 == 1)
                begin
                        x_1 = 0;
                        y_1 = 0;
                        x_1_old = 0;
                        y_1_old = 0;
                end
                else if (should_respawn_2 == 1)
                begin
                        x_2 = 159-(7)+1-40;
                        y_2 = 119-(7)+1;
                        x_2_old = 159-(7)+1-40;
                        y_2_old = 119-(7)+1;
                end
                // If a key is pressed at slowed down clock high-active level
                if (key_code[15:8] != 8'hF0 && slow_clock == 1)
        begin
                        // If it is Player 1's turn
           if (is_P1_turn == 1)
           begin
                                // Remember old location (needed for proper clearing)
              x_1_old = x_1;
              y_1_old = y_1;
                                // If 'W' pressed and Player 1 is not trying to move beyond upward screen
edge
                                if (key_code[7:0] == 8'h1D && y_1 != 0) // 'W'
                                begin
                                        // If blocked by Player 2, stop
                                        if (((y_1 == y_2 + tankHeight_2) && (x_1 > x_2 - tankWidth_2) &&
(x_1 < x_2 + tankWidth_2)))
                                        begin
                                                y_1 = y_1;
                                        end
                                        // If not blocked by obstacle, move up
                                        else if (isReallyBlocked_1 == 0)
                                        begin
                                                y_1 = y_1 - 1;
                                        end
                                end
```

```
                        // If 'A' pressed and Player 1 is not trying to move beyond leftward screen
edge
                        else if (key_code[7:0] == 8'h1C && x_1 != 0) // 'A'
                        begin
                                // If blocked by Player 2, stop
                                if (((x_1 == x_2 + tankWidth_2) && (y_1 > y_2 - tankHeight_2) &&
(y_1 < y_2 + tankHeight_2)))
                                begin
                                        x_1 = x_1;
                                end
                                // If not blocked by obstacle, move left
                                else if (isReallyBlocked_1 == 0)
                                begin
                                        x_1 = x_1 - 1;
                                end
                        end
                        // If 'S' pressed and Player 1 is not trying to move beyond downward screen
edge
                        else if (key_code[7:0] == 8'h1B && y_1 != (119-(tankHeight_1 - 1))) // 'S'
                        begin
                                // If blocked by Player 2, stop
                                if (((y_1 == y_2 - tankHeight_2) && (x_1 > x_2 - tankWidth_2) &&
(x_1 < x_2 + tankWidth_2)))
                                begin
                                        y_1 = y_1;
                                end
                                // If trying to move into Player 2 respawn point, stop
                                else if ((y_1 == (119-(tankHeight_1 - 1)-tankHeight_2)) && x_1 >
(159-(tankWidth_1 - 1)-tankWidth_2))
                                begin
                                        y_1 = y_1;
                                end
                                // If not blocked by obstacle, move down
                                else if (isReallyBlocked_1 == 0)
                                begin
                                        y_1 = y_1 + 1;
                                end
                        end
                        // If 'D' pressed and Player 1 is not trying to move beyond rightward
screen edge
                        else if (key_code[7:0] == 8'h23 && x_1 != (159-(tankWidth_1 - 1))) // 'D'
                        begin
                                // If blocked by Player 2, stop
                           if (((x_1 == x_2 - tankWidth_2) && (y_1 > y_2 - tankHeight_2) && (y_1 <
y_2 + tankHeight_2)))
                                begin
                                   x_1 = x_1;
                                end
```

```
                                    // If trying to move into Player 2 respawn point, stop
                                    else if ((x_1 == (159-(tankWidth_1 - 1)-tankWidth_2)) && y_1 >
(119-(tankHeight_1 - 1)-tankHeight_2))
                                    begin
                                       x_1 = x_1;
                                    end
                                    // If not blocked by obstacle, move right
                                    else if (isReallyBlocked_1 == 0)
                                    begin
                                       x_1 = x_1 + 1;
                                    end
                              end
                        end
                        // If it is Player 2's turn
         if (is_P2_turn == 1)
         begin
                              // Remember old location (needed for proper clearing)
                              x_2_old = x_2;
                              y_2_old = y_2;
                              // If 'I' pressed and Player 2 is not trying to move beyond upward screen
edge
                              if (key_code[7:0] == 8'h43 && y_2 != 0) // 'I'
                              begin
                                    // If blocked by Player 1, stop
                                    if ((y_2 == y_1 + tankHeight_1) && (x_2 > x_1 - tankWidth_1) && (x_2
< x_1 + tankWidth_1))
               begin
                  y_2 = y_2;
               end
                                    // If trying to move into Player 1 respawn point, stop

               else if ((y_2 == tankHeight_1) && (x_2 < tankWidth_1))
               begin
                  y_2 = y_2;
               end
                                    // If not blocked by obstacle, move right
               else if (isReallyBlocked_2 == 0)
               begin
                  y_2 = y_2 - 1;
               end
                              end
                              // If 'J' pressed and Player 2 is not trying to move beyond leftward screen
edge
            else if (key_code[7:0] == 8'h3B && x_2 != 0) // 'J'
            begin
                                    // If blocked by Player 1, stop

               if ((x_2 == x_1 + tankWidth_1) && (y_2 > y_1 - tankHeight_1) && (y_2 < y_1 +
```

```
tankHeight_1))
                begin
                   x_2 = x_2;
                end
                                    // If trying to move into Player 1 respawn point, stop

                else if ((x_2 == tankWidth_1) && (y_2 < tankHeight_1))
                                    begin
                                           x_2 = x_2;
                                    end
                                    // If not blocked by obstacle, move right
                else if (isReallyBlocked_2 == 0)
                begin
                   x_2 = x_2 - 1;
                end
            end
                            // If 'K' pressed and Player 2 is not trying to move beyond downward screen
edge
            else if (key_code[7:0] == 8'h42 && y_2 != (119-(tankHeight_2 - 1))) // 'K'
            begin
                                    // If blocked by Player 1, stop

                if ((y_2 == y_1 - tankHeight_1) && (x_2 > x_1 - tankWidth_1) && (x_2 < x_1 +
tankWidth_1))
                begin
                   y_2 = y_2;
                end
                                    // If not blocked by obstacle, move right
                else if (isReallyBlocked_2 == 0)
                begin
                   y_2 = y_2 + 1;
                end
            end
                            // If 'L' pressed and Player 2 is not trying to move beyond rightward
screen edge
            else if (key_code[7:0] == 8'h4B && x_2 != (159-(tankWidth_2-  1))) // 'L'
            begin
                                    // If blocked by Player 1, stop

                                    if ((x_2 == x_1 - tankWidth_1) && (y_2 > y_1 - tankHeight_1) && (y_2
< y_1 + tankHeight_1))
                begin
                   x_2 = x_2;
                end
                                    // If not blocked by obstacle, move right
                else if (isReallyBlocked_2 == 0)
                begin
                   x_2 = x_2 + 1;
```

```
            end
          end
        end
      end
   end

       // Write-Counter Logic (implements the counter logic needed to calculate the correct ROM address
to read from / write to)
   always @ (posedge CLOCK_50)
   begin
            if (currState == RST)
            begin
                  iMap = 0;
                  jMap = 0;
                  P1_death_count = 0;
                  P2_death_count = 0;
                  has_P1_lost <= 0;
                  has_P2_lost <= 0;
            end
            else if (currState == RSTB)
            begin
                  iMap = iMap + 1;
            end
            else if (currState == RSTC)
            begin
                  jMap = jMap + 1;
                  iMap = 0;
            end
      // Reset plot counters for turn-display and win-screen display
      else if (currState == STR)
            begin
                  i_trn = 0;
                  j_trn = 0;
                  i_win = 0;
                  j_win = 0;
            end
            // Reset plot counter for plotting turn display
            else if (currState == trnD)
            begin
                  i_trn = 0;
                  j_trn = 0;
            end
            // Increment column for plotting turn display
            else if (currState == trnE1)
            begin
              i_trn = i_trn + 1;
            end
            // Reset row and increment column for plotting turn display
```

```
		else if (currState == trnF)
		begin
			j_trn = j_trn + 1;
			i_trn = 0;
		end
		// Reset tank plot counters, respawn flags, win/loss flags
		else if (currState == A)
		begin
			i_1 = 0;
			j_1 = 0;
			i_2 = 0;
			j_2 = 0;
			should_respawn_1 <= 0;
			should_respawn_2 <= 0;
		end
		// To avoid updated keyboard input interference in middle of tank refresh,
		// save old location and latest location as constants
		// For Player 1
		else if (currState == L)		// Store old and latest Player 1 location
		begin
			obstacleCheck_1 = 0;
			x_1_old_const = x_1_old;
			y_1_old_const = y_1_old;
			x_1_const = x_1;
			y_1_const = y_1;
		end
		else if (currState == wL) // Counter for 7x1-px strip above Player 1's path for obstacle
check
		begin
			iMap = x_1_const + obstacleCheck_1;
			jMap = y_1_const - 1;
			obstacleCheck_1 = obstacleCheck_1 + 1;
			if (isShellLive == 0)
			begin
				shellDirection_1 = 1;
			end
		end
		else if (currState == aL) // Counter for 7x1-px strip left of Player 1's path for
obstacle check
		begin
			iMap = x_1_const - 1;
			jMap = y_1_const + obstacleCheck_1;
			obstacleCheck_1 = obstacleCheck_1 + 1;
			if (isShellLive == 0)
			begin
				shellDirection_1 = 2;
			end
		end
```

```
		else if (currState == sL) // Counter for 7x1-px strip below Player 1's path for obstacle
check
		begin
			iMap = x_1_const + obstacleCheck_1;
			jMap = y_1_const + 7;
			obstacleCheck_1 = obstacleCheck_1 + 1;
			if (isShellLive == 0)
			begin
				shellDirection_1 = 3;
			end
		end
		else if (currState == dL) // Counter for 7x1-px strip right of Player 1's path for
obstacle check
		begin
			iMap = x_1_const + 7;
			jMap = y_1_const + obstacleCheck_1;
			obstacleCheck_1 = obstacleCheck_1 + 1;
			if (isShellLive == 0)
			begin
				shellDirection_1 = 4;
			end
		end
		// For Player 2
		else if (currState == M) // Store old and latest Player 2 location
		begin
			obstacleCheck_2 = 0;
			x_2_old_const = x_2_old;
			y_2_old_const = y_2_old;
			x_2_const = x_2;
			y_2_const = y_2;
		end
		else if (currState == iM) // Counter for 7x1-px strip above Player 2's path for obstacle
check
		begin
			iMap = x_2_const + obstacleCheck_2;
			jMap = y_2_const - 1;
			obstacleCheck_2 = obstacleCheck_2 + 1;
			if (isShellLive == 0)
			begin
				shellDirection_2 = 1;
			end
		end
		else if (currState == jM) // Counter for 7x1-px strip left of Player 2's path for
obstacle check
		begin
			iMap = x_2_const - 1;
			jMap = y_2_const + obstacleCheck_2;
			obstacleCheck_2 = obstacleCheck_2 + 1;
```

```
                        if (isShellLive == 0)
                        begin
                                shellDirection_2 = 2;
                        end
                end
                else if (currState == kM) // Counter for 7x1-px strip below Player 2's path for obstacle
check
                begin
                        iMap = x_2_const + obstacleCheck_2;
                        jMap = y_2_const + 7;
                        obstacleCheck_2 = obstacleCheck_2 + 1;
                        if (isShellLive == 0)
                        begin
                                shellDirection_2 = 3;
                        end
                end
                else if (currState == lM) // Counter for 7x1-px strip right of Player 2's path for
obstacle check
                begin
                        iMap = x_2_const + 7;
                        jMap = y_2_const + obstacleCheck_2;
                        obstacleCheck_2 = obstacleCheck_2 + 1;
                        if (isShellLive == 0)
                        begin
                                shellDirection_2 = 4;
                        end
                end
                // Reset write counters after erasing Player 1 in preparation for plotting Player 1
                else if (currState == D)
                begin
                        i_1 = 0;
                        j_1 = 0;
                        iMap = x_1;
                        jMap = y_1;
                        // If Player 1 was hit, then reset position to starting position (i.e. respawn)
                        if (is_P1_hit == 1)
                        begin
                                is_P1_hit <= 0;
                                should_respawn_1 <= 1;
                                x_1_const = 0;
                                y_1_const = 0;
                                shellDirection_1 = 3;
                        end
                end
                // Reset write counters after erasing Player 2 in preparation for plotting Player 2
                else if (currState == I)
                begin
                        i_2 = 0;
```

```
        j_2 = 0;
        iMap = x_2;
        jMap = y_2;
        // If Player 2 was hit, then reset position to starting position (i.e. respawn)
        if (is_P2_hit == 1)
        begin
                is_P2_hit <= 0;
                should_respawn_2 <= 1;
                x_2_const = 159-(7)+1-40;
                y_2_const = 119-(7)+1;
                shellDirection_2 = 1;
        end
end
// ERASE / PLOT OF PLAYER 1
// Set map address for first pixel of Player 1 erase
else if (currState == B0)
begin
        iMap = x_1_const + i_1;
        jMap = y_1_const + j_1;
end
// Set map address for regular Player 1 erase / plot and increment column
else if (currState == B1 || currState == E1)
begin
        iMap = x_1_const + i_1;
        jMap = y_1_const + j_1;
        i_1 = i_1 + 1;
end
// Reset row and increment column for Player 1 erase / plot
else if (currState == C || currState == F)
begin
        j_1 = j_1 + 1;
        i_1 = 0;
end
// ERASE / PLOT OF PLAYER 2
// Set map address for first pixel of Player 2 erase
else if (currState == G0)
begin
        iMap = x_2_const + i_2;
        jMap = y_2_const + j_2;
end
// Set map address for regular Player 2 erase / plot and increment column
else if (currState == G1 || currState == J1)
begin
        iMap = x_2_const + i_2;
        jMap = y_2_const + j_2;
        i_2 = i_2 + 1;
end
// Reset row and increment column for Player 2 erase / plot
```

```
else if (currState == H || currState == K)
begin
	j_2 = j_2 + 1;
	i_2 = 0;
end
// SHELL STATES
// Initialize shell position
else if (currState == shA)begin
	isShellLive <= 1;
	// Initialize shell position in front of Player 1's barrel
	if (is_P1_turn == 1 && is_P1_shooting == 1)begin
		if (shellDirection_1 == 1)begin //UP
			shellX = x_1 + 3;
			shellY = y_1 - 1;
		end
		else if (shellDirection_1 == 2)begin //LEFT
			shellX = x_1 - 1;
			shellY = y_1 + 3;
		end
		else if (shellDirection_1 == 3)begin //DOWN
			shellX = x_1 + 3;
			shellY = y_1 + 7;
		end
		else if (shellDirection_1 == 4)begin //RIGHT
			shellX = x_1 + 7;
			shellY = y_1 + 3;
		end
	end
	// Initialize shell position in front of Player 2's barrel
	else if (is_P2_turn == 1 && is_P2_shooting == 1)begin
		if (shellDirection_2 == 1)begin //UP
			shellX = x_2 + 3;
			shellY = y_2 - 1;
		end
		else if (shellDirection_2 == 2)begin //LEFT
			shellX = x_2 - 1;
			shellY = y_2 + 3;
		end
		else if (shellDirection_2 == 3)begin //DOWN
			shellX = x_2 + 3;
			shellY = y_2 + 7;
		end
		else if (shellDirection_2 == 4)begin //RIGHT
			shellX = x_2 + 7;
			shellY = y_2 + 3;
		end
	end
```

```
		end
		// Check map for obstacles at shell position
		else if (currState == shB1) begin
			iMap = shellX;
			jMap = shellY;
		end
		// Move shell position in firing direction
		else if (currState == shD)
		begin
			// If Player 1 is shooting
			if (is_P1_shooting == 1)
			begin
				if (shellDirection_1 == 1)begin //UP
					shellY = shellY - 1;
				end
				else if (shellDirection_1 == 2)begin //LEFT
					shellX = shellX - 1;
				end
				else if (shellDirection_1 == 3)begin //DOWN
					shellY = shellY + 1;
				end
				else if (shellDirection_1 == 4)begin //RIGHT
					shellX = shellX + 1;
				end
			end
			// If Player 2 is shooting
			else if (is_P2_shooting == 1)
			begin
				if (shellDirection_2 == 1)begin //UP
					shellY = shellY - 1;
				end
				else if (shellDirection_2 == 2)begin //LEFT
					shellX = shellX - 1;
				end
				else if (shellDirection_2 == 3)begin //DOWN
				   shellY = shellY + 1;
				end
				else if (shellDirection_2 == 4)begin //RIGHT
				   shellX = shellX + 1;
				end
			end
			// Check map for obstacles at shell position
			iMap = shellX;
			jMap = shellY;
		end
		// Shell has reached end of move iteration (either hit a tank, obstacle, map border, or
nothing)
		else if (currState == shE)
```

```
		begin
			// If Player 1 is hit
			if ((shellX >= x_1 && shellX < x_1 + tankWidth_1) && (shellY >= y_1 && shellY <
y_1 + tankHeight_1))
			begin
				isShellLive <= 0;
				is_P1_hit <= 1;
				P1_death_count = P1_death_count + 1;
				if (P1_death_count == 3)
				begin
					has_P1_lost <= 1;
				end
				else
				begin
					has_P1_lost <= 0;
				end
			end
			// If Player 2 is hit
			else if ((shellX >= x_2 && shellX < x_2 + tankWidth_2) && (shellY >= y_2 && shellY
< y_2 + tankHeight_2))
			begin
				isShellLive <= 0;
				is_P2_hit <= 1;
				P2_death_count = P2_death_count + 1;
				if (P2_death_count == 3)
				begin
					has_P2_lost <= 1;
				end
				else
				begin
					has_P2_lost <= 0;
				end
			end
			// If map borders hit
			else if (shellX == 0 || shellX == 159 || shellY == 0 || shellY == 119)
			begin
				isShellLive <= 0;
			end
			// If obstacle is hit
			else if (map_pixel_colour == 3'b000 || map_pixel_colour == 3'b110)
			begin
				// Set the shell to be not live
				isShellLive <= 0;
			end
			// If nothing is hit, do nothing
		end
		// Initialize for Score-keeping
		else if (currState == SCR)
```

```
	begin
		if (is_P1_hit == 1) y_scr = 103;
		else if (is_P2_hit == 1) y_scr = 78;
		i_scr = 0;
		j_scr = 0;
	end
	// Advance column for score-keeping
	else if (currState == SCRB)
	begin
		i_scr = i_scr + 1;
	end
	//Advance row for score-keeping
	else if (currState == SCRC)
	begin
		j_scr = j_scr + 1;
		i_scr = 0;
	end
	// Plot counter logic for win-screen display
	else if (currState == WINB)
	begin
		i_win = i_win + 1;
	end
	else if (currState == WINC)
	begin
		j_win = j_win + 1;
		i_win = 0;
	end
end

// Next State Logic
always @ (*)
begin
	case (currState)
		RST:   nextState <= RSTA;
		RSTA: if (iMap < 159) nextState <= RSTB;
				else if (iMap == 159) nextState <= RSTC;
		RSTB: nextState <= RSTA;
		RSTC: if (jMap < 119) nextState <= RSTA;
				else if (jMap == 119) nextState <= STR;
		STR:  nextState <= trnD;
		trnD: nextState <= trnE;
		trnE: if (i_trn < 6) begin // Traverse 6-px wide row of turn display for plot
					nextState <= trnE1;
				end else if (i_trn == 6) begin
					nextState <= trnF;
				end
		trnE1: nextState <= trnE; // ROM buffer state
		trnF: if (j_trn < 11) begin // Traverse 11-px wide column of turn display for plot
```

```
                                        nextState <= trnE;
                                end else if (j_trn == 11) begin
                                        nextState <= A;
                                end
                        A: begin
                                if              ((key_code[15:8] != 8'hF0 && // If any of WASD keys
pressed in Player 1's turn or Player 1 hit
                                                        (key_code[7:0] == 8'h1D ||
                                                         key_code[7:0] == 8'h1C ||
                                                         key_code[7:0] == 8'h1B ||
                                                         key_code[7:0] == 8'h23) &&
                                                         is_P1_turn == 1) ||
                                                         is_P1_hit == 1)
                                begin
                                        nextState <= L;
                                end
                                else if         ((key_code[15:8] != 8'hF0 && // If any of IJKL keys
pressed in Player 2's turn or Player 2 hit
                                                        (key_code[7:0] == 8'h43 ||
                                                        key_code[7:0] == 8'h3B ||
                                                        key_code[7:0] == 8'h42 ||
                                                        key_code[7:0] == 8'h4B) &&
                                                        is_P2_turn == 1) ||
                                                        is_P2_hit == 1)
                                begin
                                        nextState <= M;
                                end
                                // If Player 1 has initialized shell-firing sequence
                                else if (is_P1_turn == 1 && key_code[15:8] != 8'hF0 && key_code[7:0]
== 8'h21 && isShellLive == 0)
                                begin
                                        nextState <= shA;
                                        is_P1_shooting <= 1;
                                        is_P2_shooting <= 0;
                                end
                                // If Player 2 has initialized shell-firing sequence
                                else if (is_P2_turn == 1 && key_code[15:8] != 8'hF0 && key_code[7:0]
== 8'h31 && isShellLive == 0)
                                begin
                                        nextState <= shA;
                                        is_P1_shooting <= 0;
                                        is_P2_shooting <= 1;
                                end
                                // If shell is already live and moving, then skip shell position
initialization
                                else if (isShellLive == 1)
                                begin
                                        nextState <= P;
```

```
                                end
                                else // Otherwise, loop back to display the appropriate player's
turn
                                begin
                                        nextState <= STR;
                                        is_P1_shooting <= 0;
                                        is_P2_shooting <= 0;
                                end
                         end
                    // Store Player 1 old and new location and redirect to appropriate tank direction
                    L:    if ((key_code[15:8] != 8'hF0) && (key_code[7:0] == 8'h1D)) nextState <= wL;
                                else if ((key_code[15:8] != 8'hF0) && (key_code[7:0] == 8'h1C))
nextState <= aL;
                                else if ((key_code[15:8] != 8'hF0) && (key_code[7:0] == 8'h1B))
nextState <= sL;
                                else if ((key_code[15:8] != 8'hF0) && (key_code[7:0] == 8'h23))
nextState <= dL;
                    // Traverse 7x1-px strip of map in front of Player 1's path for obstacles
                    // ROM buffer states (sets map-reading indices to check for obstacles)
                    wL:   nextState <= LW;
                    aL:   nextState <= LA;
                    sL:   nextState <= LS;
                    dL:   nextState <= LD;
                    // Decision making states (sets isReallyBlocked_1 to 1 if at least one obstacle
found; if all clear, sets to 0)
                    LW:   if (map_pixel_colour == 3'b000 || map_pixel_colour == 3'b110)
                                begin
                                        isReallyBlocked_1 <= 1;
                                        nextState <= B0;
                                end
                                else if (obstacleCheck_1 <8)
                                begin
                                        nextState <= wL;
                                end
                                else if (obstacleCheck_1 == 8)
                                begin
                                        isReallyBlocked_1 <= 0;
                                        nextState <= B0;
                                end
                    LA:   if (map_pixel_colour == 3'b000 || map_pixel_colour == 3'b110)
                                begin
                                        isReallyBlocked_1 <= 1;
                                        nextState <= B0;
                                end
                                else if (obstacleCheck_1 <8)
                                begin
                                        nextState <= aL;
                                end
```

```
                        else if (obstacleCheck_1 == 8)
                        begin
                                isReallyBlocked_1 <= 0;
                                nextState <= B0;
                        end
                LS:   if (map_pixel_colour == 3'b000 || map_pixel_colour == 3'b110)
                        begin
                                isReallyBlocked_1 <= 1;
                                nextState <= B0;
                        end
                        else if (obstacleCheck_1 <8)
                        begin
                                nextState <= sL;
                        end
                        else if (obstacleCheck_1 == 8)
                        begin
                                isReallyBlocked_1 <= 0;
                                nextState <= B0;
                        end
                LD:   if (map_pixel_colour == 3'b000 || map_pixel_colour == 3'b110)
                        begin
                                isReallyBlocked_1 <= 1;
                                nextState <= B0;
                        end
                        else if (obstacleCheck_1 <8)
                        begin
                                nextState <= dL;
                        end
                        else if (obstacleCheck_1 == 8)
                        begin
                                isReallyBlocked_1 <= 0;
                                nextState <= B0;
                        end
                // Store Player 2 old and new location and redirect to appropriate tank direction

                M:    if ((key_code[15:8] != 8'hF0) && (key_code[7:0] == 8'h43)) nextState <= iM;
                        else if ((key_code[15:8] != 8'hF0) && (key_code[7:0] == 8'h3B))
nextState <= jM;
                        else if ((key_code[15:8] != 8'hF0) && (key_code[7:0] == 8'h42))
nextState <= kM;
                        else if ((key_code[15:8] != 8'hF0) && (key_code[7:0] == 8'h4B))
nextState <= lM;
                // ROM buffer states (sets map-reading indices to check for obstacles)

                iM:   nextState <= MI;
                jM:   nextState <= MJ;
                kM:    nextState <= MK;
                lM:    nextState <= ML;
```

```
                    // Decision making states (sets isReallyBlocked_2 to 1 if at least one obstacle
found; if all clear, sets to 0)
                    MI:    if (map_pixel_colour == 3'b000 || map_pixel_colour == 3'b110)
                                  begin
                                        isReallyBlocked_2 <= 1;
                                        nextState <= G0;
                                  end
                                  else if (obstacleCheck_2 <8)
                                  begin
                                        nextState <= iM;
                                  end
                                  else if (obstacleCheck_2 == 8)
                                  begin
                                        isReallyBlocked_2 <= 0;
                                        nextState <= G0;
                                  end
                    MJ:   if (map_pixel_colour == 3'b000 || map_pixel_colour == 3'b110)
                                  begin
                                        isReallyBlocked_2 <= 1;
                                        nextState <= G0;
                                  end
                                  else if (obstacleCheck_2 <8)
                                  begin
                                        nextState <= jM;
                                  end
                                  else if (obstacleCheck_2 == 8)
                                  begin
                                        isReallyBlocked_2 <= 0;
                                        nextState <= G0;
                                  end
                    MK:   if (map_pixel_colour == 3'b000 || map_pixel_colour == 3'b110)
                                  begin
                                        isReallyBlocked_2 <= 1;
                                        nextState <= G0;
                                  end
                                  else if (obstacleCheck_2 <8)
                                  begin
                                        nextState <= kM;
                                  end
                                  else if (obstacleCheck_2 == 8)
                                  begin
                                        isReallyBlocked_2 <= 0;
                                        nextState <= G0;
                                  end
                    ML:   if (map_pixel_colour == 3'b000 || map_pixel_colour == 3'b110)
                                  begin
                                        isReallyBlocked_2 <= 1;
                                        nextState <= G0;
```

```
                end
                else if (obstacleCheck_2 <8)
                begin
                        nextState <= lM;
                end
                else if (obstacleCheck_2 == 8)
                begin
                        isReallyBlocked_2 <= 0;
                        nextState <= G0;
                end
// ERASE PLAYER 1
// Map index initialization ROM buffer state
B0:   nextState <= B;
// Traverse tank width for erase
B:    if (i_1 < tankWidth_1) begin
                        nextState <= B1;
                end else if (i_1 == tankWidth_1) begin
                        nextState <= C;
                end
// ROM buffer state
B1:   nextState <= B;
// Traverse tank height for erase
C:    if (j_1 < tankHeight_1 - 1) begin
                        nextState <= B1;
                end else if (j_1 == tankHeight_1 - 1) begin
                         nextState <= D;
                end
// Tank-drawing counter reset state
D:    nextState <= E;
// PLOT PLAYER 1
// Traverse tank width for plot
E:    if (i_1 < tankWidth_1) begin
                        nextState <= E1;
                end else if (i_1 == tankWidth_1) begin
                        nextState <= F;
                end
// ROM buffer state
E1:   nextState <= E;
// Traverse tank height for plot
F:    if (j_1 < tankHeight_1 - 1) begin
                        nextState <= E;
                end else if (j_1 == tankHeight_1 - 1) begin
                        nextState <= P;
                end
// ERASE PLAYER 2
// Map index initialization ROM buffer state
G0:   nextState <= G;
// Traverse tank width for erase
```

```
G:    if (i_2 < tankWidth_2) begin
                        nextState <= G1;
                end else if (i_2 == tankWidth_2) begin
                        nextState <= H;
                end
// ROM buffer state
G1:   nextState <= G;
// Traverse tank height for erase
H:    if (j_2 < tankHeight_2 - 1) begin
                        nextState <= G1;
                end else if (j_2 == tankHeight_2 - 1) begin
                        nextState <= I;
                end
// Tank-drawing counter reset state
I:    nextState <= J;
// PLOT PLAYER 2
// Traverse tank width for plot
J:    if (i_2 < tankWidth_2) begin
                        nextState <= J1;
                end else if (i_2 == tankWidth_2) begin
                        nextState <= K;
                end
// ROM buffer state
J1:   nextState <= J;
// Traverse tank height for plot
K:    if (j_2 < tankHeight_2 - 1) begin
                        nextState <= J1;
                end else if (j_2 == tankHeight_2 - 1) begin
                        nextState <= P;
                end
// Junction state between returning to displaying win screen, turn display, or
shell
P:     if (has_P1_lost == 1 || has_P2_lost == 1)
                begin
                        nextState <= SCR;
                end
                else if (isShellLive == 1)
                begin
                        nextState <= shB1;
                end
                else if (isShellLive == 0)
                begin
                        nextState <= STR;
                end
// SHELL STATES
// Initialize shell position
shA:   nextState <= shB1;
// ROM buffer state
```

```
shB1: nextState <= shB;
// Plot shell (at slowed-down clock)
shB:  if (clock_count != 1) nextState <= shB;
          else if (clock_count == 1) nextState <= shC;
// Erase shell
shC:  nextState <= shD;
// Move shell
shD:  nextState <= shE;
// Terminate shell sequence and make decision
shE:  begin
                    if (is_P1_hit == 1 || is_P2_hit == 1)
                    begin
                          is_P1_shooting <= 0;
                          is_P2_shooting <= 0;
                    end
                    nextState <= SCR;
              end
// SCORE-KEEPING STATES
SCR:  if (is_P1_hit == 1 || is_P2_hit == 1) begin
                    nextState <= SCRA;
              end else begin
                    nextState <= STR;
              end
SCRA: if (i_scr < 4) nextState <= SCRB;
              else if (i_scr == 4) nextState <= SCRC;
SCRB: nextState <= SCRA;
SCRC: if (j_scr < 8) nextState <= SCRA;
              else if (j_scr == 8)
              begin
                    if (has_P1_lost == 1 || has_P2_lost == 1)
                    begin
                          nextState <= WINA;
                    end
                    else
                    begin
                          nextState <= STR;
                    end
              end
// WIN STATES
WINA: begin
                    if (i_win < 79) begin // Traverse width of Win Screen
                          nextState <= WINB;
                    end else if (i_win == 79) begin
                          nextState <= WINC;
                    end
              end
WINB: nextState <= WINA; // ROM buffer state
WINC: if (j_win < 49) begin // Traverse height of Win Screen
```

```
                                        nextState <= WINA;
                                end else if (j_win == 49) begin
                                        nextState <= END;
                                end
                        END:  begin
                                        nextState <= END; // Infinite game-over state
                                end
                endcase
        end

        // Current State Logic
        always @ (posedge CLOCK_50 or negedge resetn)
        begin
                if (resetn == 0)
                begin
                        currState <= RST;
                end
                else
                begin
                        currState <= nextState;
                end
        end

endmodule
```

APPENDIX B: Screenshots

i. Tank Movement

Figure B.i.1: At the beginning of *Tank Duels*, both tanks are invisible, the turns alternate between 1 and 2, and both scores are 0.

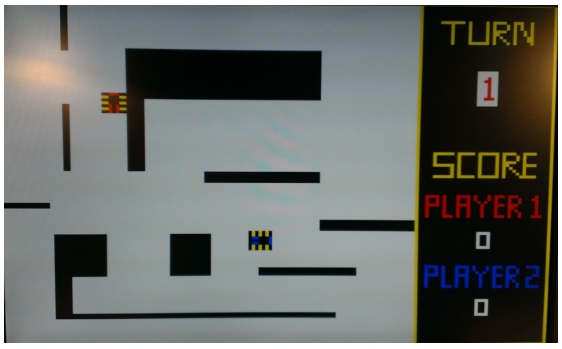


Figure B.i.2: Both tanks move across the map if unimpeded by each other, obstacles, or the map edges.

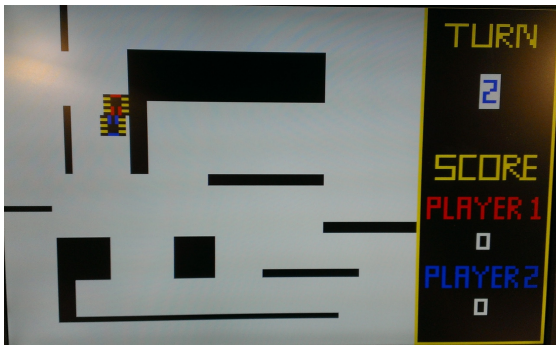


Figure B.i.3: Here, neither tank can move through the other.

ii. Shell Shooting

Figure B.ii.1: Player 1 fires a shell towards Player 2 during its turn.

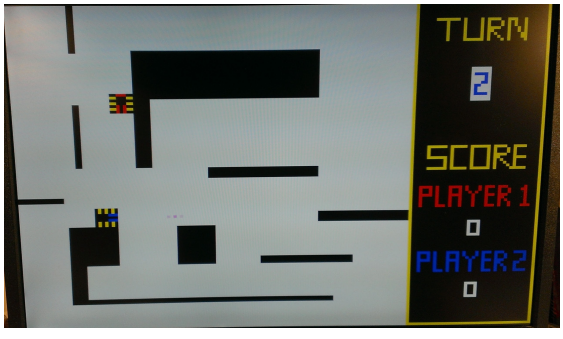


Figure B.ii.2: Player 2 fires a shell towards an obstacle during its turn.

iii. Killing and Respawning

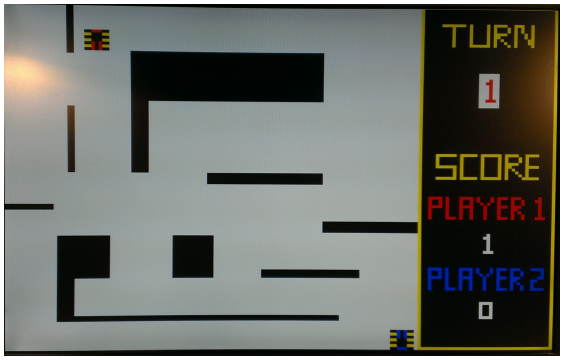


Figure B.iii.1: Player 1 has just "killed" Player 2 (see Figure B.ii.1) and Player 2 has respawned at the bottom-right corner of the map - Player 1's score has increased by 1.

iv. Winning

Figure B.iv.1: Player 1 has just "killed" Player 2 three times and won - a winning message is displayed and the game is stopped. The game will remain stopped until it is restarted.

v. Restarting

Figure B.v.1: "KEY0" has just been pressed on the DE2 board, thus restarting the game (both players relocate to their respawn points and both scores revert to 0).