CSEE 4840 Embedded Systems
Final Project Report
TankGO! VideoGame

May 11 2024

Yiyang Peng (yp2655)
Xuanbo Xu (xx2440)
JiaYi Wang (jw4462)
YiZhi Wang (yw4174)

Contents

1 Introduction

1.1 Game Overview

We designed a traditional tank game based on the original Tank arcade game developed in 1974 by a subsidiary of Atari. Two players control tanks in three possible mazes. Player's tank positions are viewed from above. The game utilizes joystick inputs for tank navigation and a button press to shoot. Bullets bounce off walls up to 15 times, and tanks lose health points (HP) upon being hit, with the game ending when a tank's HP reaches zero. Tanks can accidentally be shot by their own bullets. Each player has 8 HP's in total. Once the game is over, the game directs you back to the map selection stage and is in an infinite loop.

The project demonstrates the integration of real-time control systems and graphical output, emphasizing interrupt handling, collision detection, and input processing. This document outlines the system's architecture, challenges in implementation, and solutions that ensure functional and responsive game play on an embedded platform.

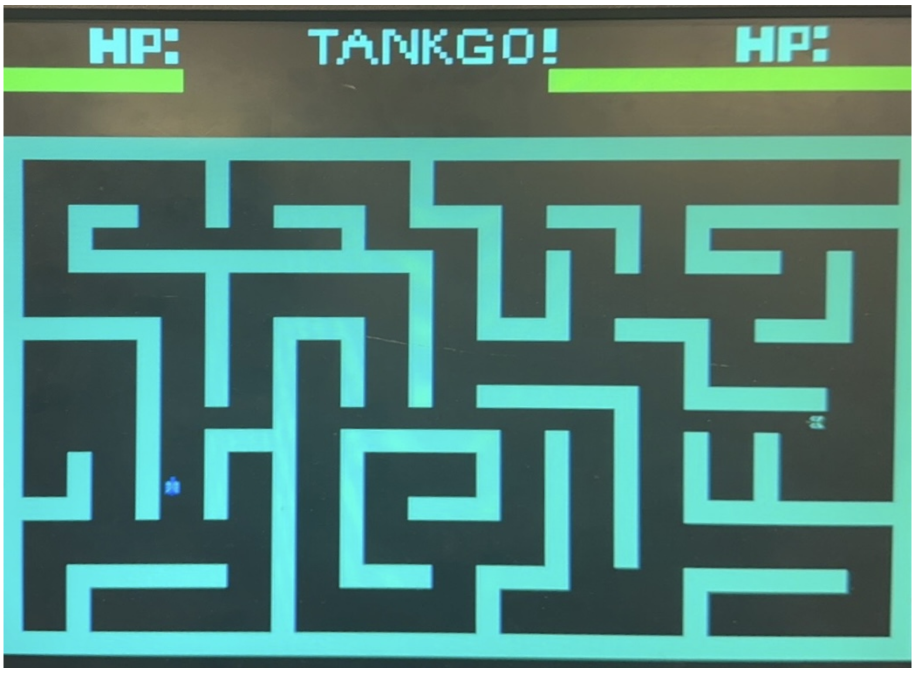


Figure 1: Image of our game

1.2 System Architecture

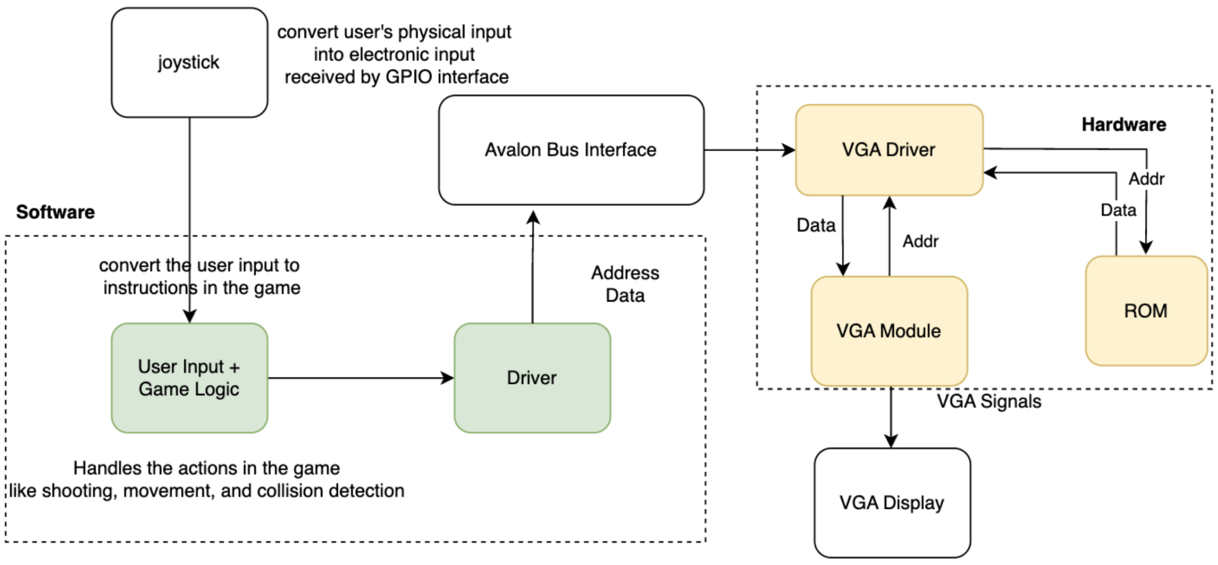

joystick
convert user's physical input into electronic input received by GPIO interface
Avalon Bus Interface
Hardware
VGA Driver
Addr
Data
Data
Addr
Software
convert the user input to instructions in the game
Address Data
VGA Module
ROM
User Input + Game Logic
Driver
VGA Signals
Handles the actions in the game like shooting, movement, and collision detection
VGA Display

Figure 2: Systems Block Diagram

Players controls the game using joysticks, interfacing with the software controlling the game logic by communicating through the USB protocol. The signals from the joystick is then communicated to the software where the game logic is performed. The software communicates with the hardware with a device driver. The hardware handles communication with the VGA monitor by sending signals on how to display the graphics.

The joystick.c file collects the joystick inputs by communicating through the USB protocol and libusb library. This is accessed by the hello.c file where majority of the game logic is written. This includes tank movement, bullet movement, collision detection, and score calculations. The vga_ball.c file is a device driver which communicates with the vga module vga_ball.sv. This communication passes the avalon bus interface addresses of bytes to be sent over to the hardware.

The hardware handles the graphics and displays on the VGA monitor based on location and other information sent through the avalon bus by the software. It also consists of the on-chip memory ROMs that we placed inside the FPGA. This stores all the sprites and other condensed memories such as maps in bit arrays and letters for displays.

The vga_ball.sv module sends requested addresses to the ROMs, where the ROMs returns the requested output data for displays per pixel on the VGA monitor. Finally, signals are passed through to the VGA display in order to display the right color on the right pixel at the right time.

2 Hardware

2.1 Graphics

2.1.1 Sprites

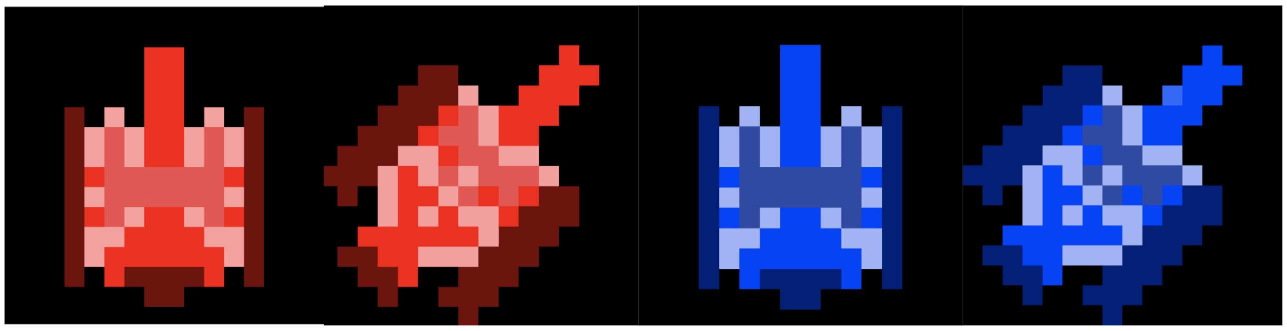


Figure 3: Sprites we created ourselves

We used matrix translations on mifs to achieve 8 directions with only four sprites in ROM for two tanks. This can save us a lot of ROM space. This is done mathematically shown in the following diagram:

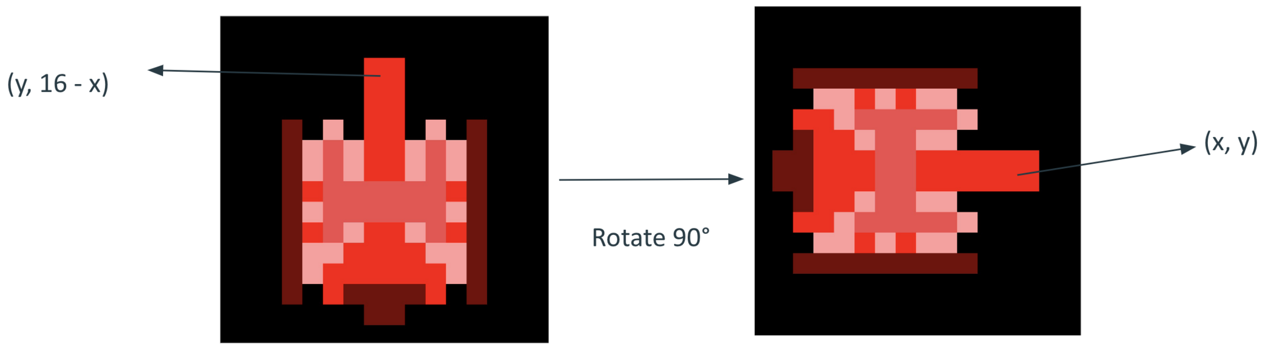


Figure 4: Sprites we created ourselves

2.1.2 Memory Allocation

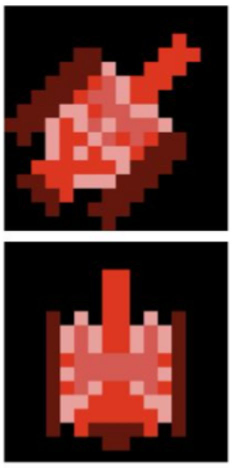
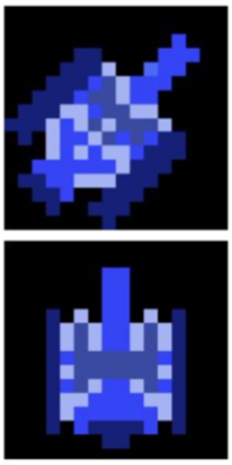
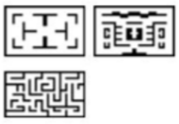
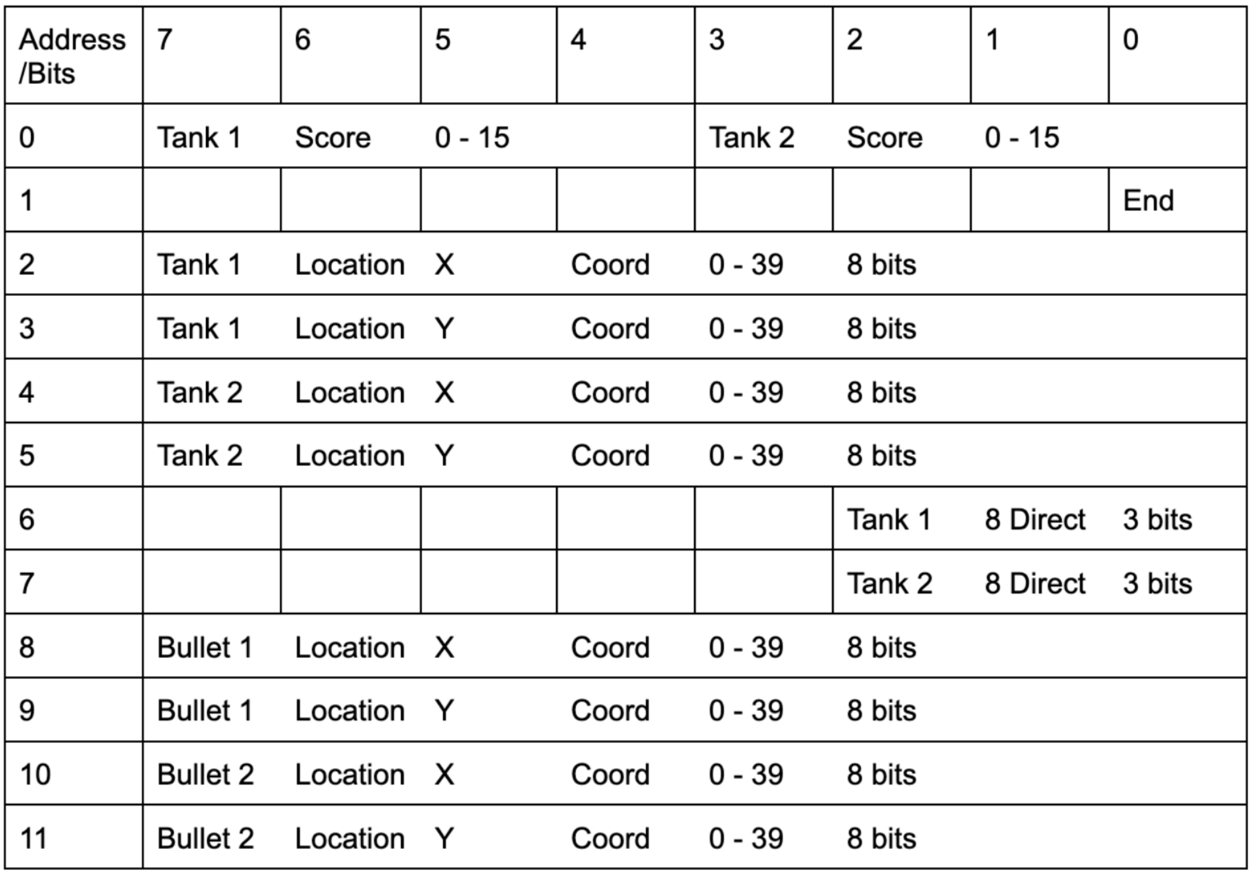
Category	Graphics	Size(bits)	# of images	Total Size(bits)
Tank 1		16*16	2	12288
Tank 2		16*16	2	12288
Text 1	TANKGO!	48*8	1	384
Text 2	GAMEOVER	60*8	1	480
Text 3	HP:	16*8	1	128
Map		40*25	3	3000

Figure 5: Memory Budget Calculations, totals 28568 bits

3 Avalon Bus Interface



Address /Bits	7	6	5	4	3	2	1	0
0	Tank 1	Score	0 - 15		Tank 2	Score	0 - 15	
1								End
2	Tank 1	Location	X	Coord	0 - 39	8 bits		
3	Tank 1	Location	Y	Coord	0 - 39	8 bits		
4	Tank 2	Location	X	Coord	0 - 39	8 bits		
5	Tank 2	Location	Y	Coord	0 - 39	8 bits		
6						Tank 1	8 Direct	3 bits
7						Tank 2	8 Direct	3 bits
8	Bullet 1	Location	X	Coord	0 - 39	8 bits		
9	Bullet 1	Location	Y	Coord	0 - 39	8 bits		
10	Bullet 2	Location	X	Coord	0 - 39	8 bits		
11	Bullet 2	Location	Y	Coord	0 - 39	8 bits		

Figure 6: Avalon Bus Interface

We have settled on byte sized addresses, and required 12 bytes per pass through in total.

A tank's HP is split into 16 levels. When all 16 HPs are lost, game is over. Thus, we require four bits for each tank's HP status. This totals 8 bits and are stored in the first address.

Game over signal is required as when this is high, the hardware displays the game over page for indication that this round of battle has terminated. This is an independent signal thus occupied the last bit of the next address.

Tank's location on the VGA monitor is determined by a 120 times 160 grid. This is not the size of the pixels as precisely locating each pixel requires a very wide location coordinate. Thus we have simplified this process to just having a 120 times 160 grid. At this count, an 8 bit address is perfect for one axis coordinate. Thus we required four bytes in total, where address 02 and 03 determines the x and y coordinates of tank 1, and addresses 04 and 05 determines the x and y coordinates of tank 2.

Each tank requires a direction definition. This is to do with displaying the right tank direction on the VGA monitor as tanks is able to rotate in 8 directions in our game. Thus, in order to differentiate 8 directions in total, three bits for each tank is required to define their directions. This is done in address 06 and 07 as independent addresses are simply easier to define in the software.

Lastly, because our bullets are perfect circles in plain color, our bullets from each tank only requires their location coordinates and not their directions. Thus, we applied similar practices as the location of the tank, to have four address representing the location coordinates for two bullets.

4 Software

4.1 User Input

4.1.1 Overview

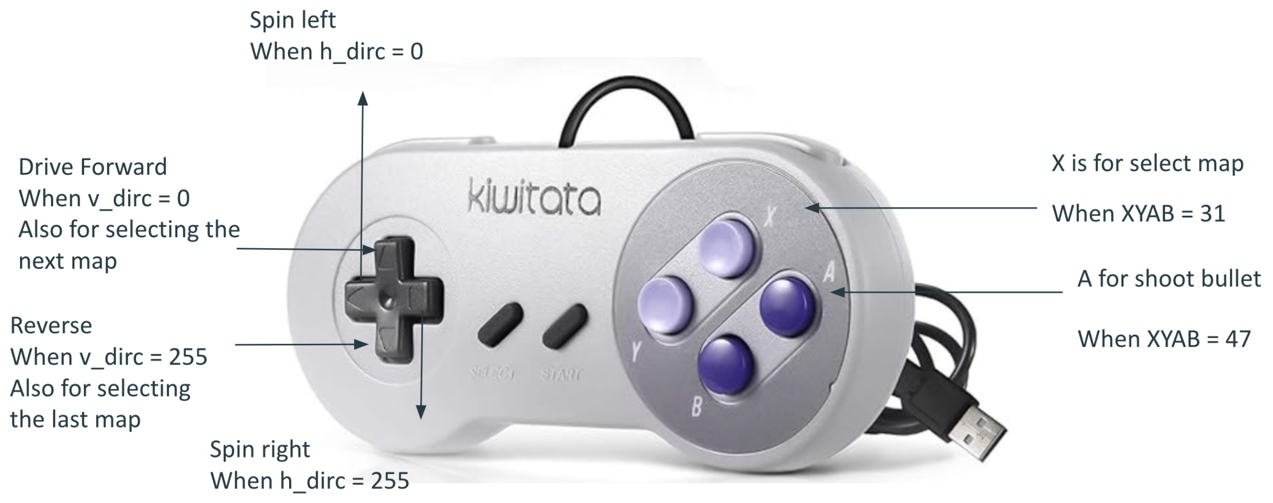


Figure 7: Our joysticks

Players interact with each other with a set of joysticks.The joysticks connect via USB, identified by an idProduct of 17, and feature a single interface. Players use the arrow keys on the controllers to move the tanks and the A button to fire bullets. We implemented counter variables for each control to manage input speed and effectively debounce the switches, ensuring that each physical press corresponds to a single software input. Such debounce counters are generally set to 300 for continuous movements of the tanks, and 600 for discrete rotations of the tank. Moving around the map is considered to be continuous movements and rotating and switching direction is considered to be more discrete. Input from both controllers is managed within a single loop, as threading was not required to achieve low input latency. However threading is used in bullet firing which we will talk about later.

We adapted the joystick.c skeleton to suit our setup, modifying the function that opens the keyboard to return a structure with details about both controllers and restricting it to only recognize our specified controllers. Using libusb_interrupt_transfer, we read 7-byte protocol messages from each controller into a structure designed to capture each field. This data is then processed in the game-loop to update tank movements and shooting actions.

4.1.2 Communication Protocol

Each controller uses a 7-byte protocol consisting of three initial constant fields that define the protocol type. Since our controllers do not match any built-in libusb protocols, these fields are set to 255, denoting 'protocol 0'. Protocols recognized by libusb, like those for keyboards, would be indicated by 'protocol 1'.

Constant	Constant	Constant	h_dirc	v_dirc	XYAB	Other keys

Figure 8: 7 byte protocol

The protocol also includes fields for directional inputs and buttons. The left/right direction field is set by default to 127, changing to 0 for left and 255 for right. Similarly, the up/down direction field starts at 127 and adjusts to 0 for up and 255 for down.

Button presses are encoded in a single field for X, Y, A, B, where the integer values represent different combinations of button presses:

- 15 for none
- 31 for X
- 47 for A
- 63 for A and X
- 79 for B
- 95 for B and X
- 111 for A and B
- 127 for A, B, and X
- etc.

This encoding is cumulative, with each additional button press adding to the integer value of the field. The fields for the Left Bumper, Right Bumper, Start, and Select buttons operate similarly, substituting the A, B, X, Y keys with their respective functionalities. Only X and A are used in our game design thus we only explored this far of the joystick return values.

4.2 Game Logic

4.2.1 Overview

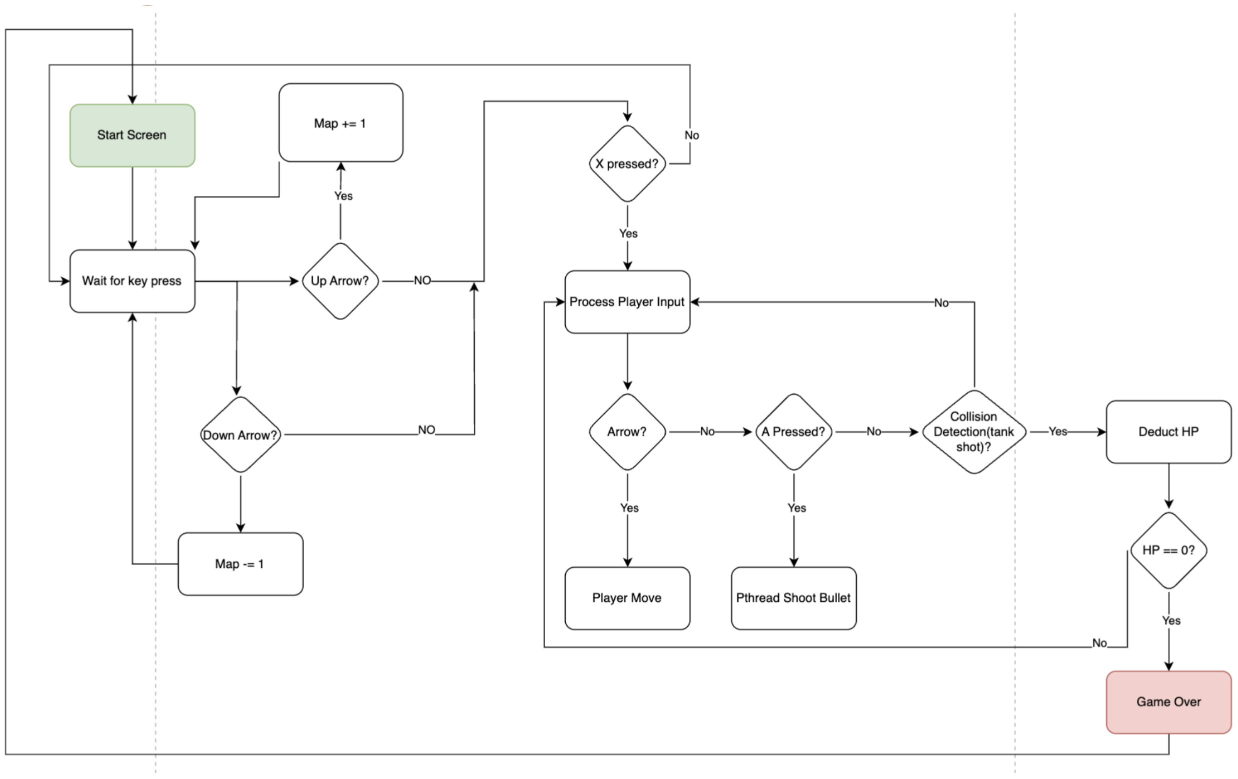


Figure 9: Game Logic Flow Chart

Our game logic consists of three important loops. 1) loop for game to be in an infinity loop 2) loop to select maps before playing the game 3) loop to play the game until one tank dies.

At start screen, we wait for key presses to select different maps. If up arrow or down arrow is pressed, we allocate the right map onto the display. When player confirms their wanted map, they press the X button. Thus upon pressing the X button, game logic starts to listen to player inputs of tank movement or bullet shoot. Tank movement is navigated by arrow keys, and bullets are shot by A buttons.

If players move, then we update the location of display of the tank through the hardware onto the VGA display. If players shoot, we started a parallel thread to process bullet's position in parallel until bullets hit a tank or bullet bounces wall after 15 times and disappears.

Collision detection is happening simultaneously throughout this loop. For every movement of the tank or the bullet, collision detection is happening and also calculates results of each collision detected. If it is tank with wall, then tank movement is restricted to the three other directions. If bullet collides with wall then bouncing direction is calculated. If bullet collides with tank then HP deduction is also conducted in this loop.

If HP is zero, the program prints the game over page and returns to the start screen after a brief second of halt.

4.2.2 Tank Movement

We designed our tanks to have 8 directions to move in. This allows the bouncing bullet feature off the walls. Every bounce is shot at either 45 degrees or 90 degrees. Since we only have arrow controls, and we want to achieve freedom of movement in 8 directions, we have adopted the following control technique.

Left and right arrows are reserved for directional determination only. So pressing the left arrow rotates the tank at the same spot counterclockwise. Similarly, pressing the right arrow rotates the tank at the same spot clockwise. This can be shown in the following diagram:

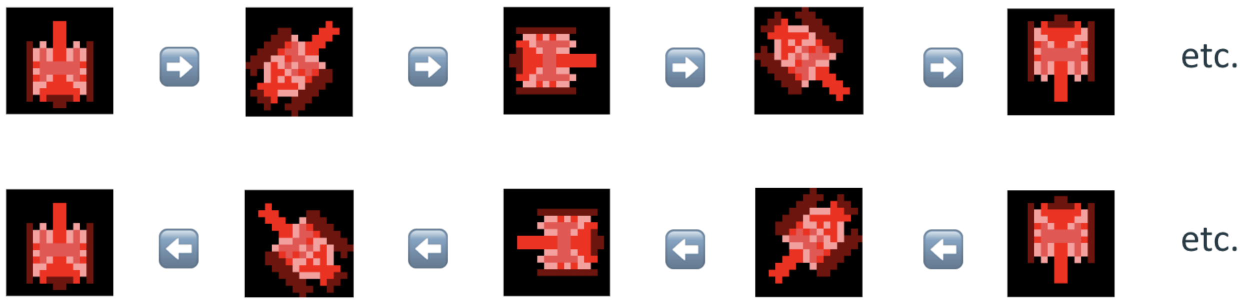


Figure 10: Tank Rotational Movements

For translational movements, up and down arrows are used to move the tank in the direction it is facing. If the tank is facing east, then pressing the up arrow corresponds to moving the tank towards the east, i.e. driving forward. If the down arrow is pressed, this corresponds to moving the tank towards the west, i.e. in reverse mode.

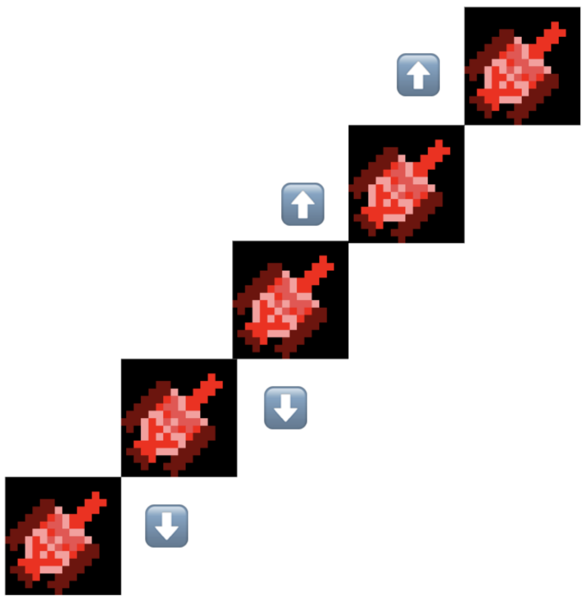


Figure 11: Tank Translational Movements

4.2.3 Collision Detection Overview

The VGA screen is in 480 times 640 pixels. This is of too high resolution for us to precisely place tanks and bullets as coordinates may be too wide. Thus, we have decided to decrease its resolution by 4 times, making it a grid of 120 times 160. This is the set of grid that tank and bullet movements are working with. Every frame the tank or bullet moves one grid distance.

Furthermore, this grid is still considered to be too high resolution. In order to create maps with bolder outlines, we divided this further into 30 times 40 tiles. We designed map shapes based on this tile design. And the size of our tank also corresponds to one of such tile. Thus, our map is stored as a bit array of 1200 in length.

4.2.4 Collision Detection (Tank with Map)

The tank cannot intersect with the map. If they touch, the tank is not allowed to move further into the maps.

The tank sprite is a 4 times 4 grid. Thus, its corners are important to locate thus calculating whether any of these corners intersect with the map.

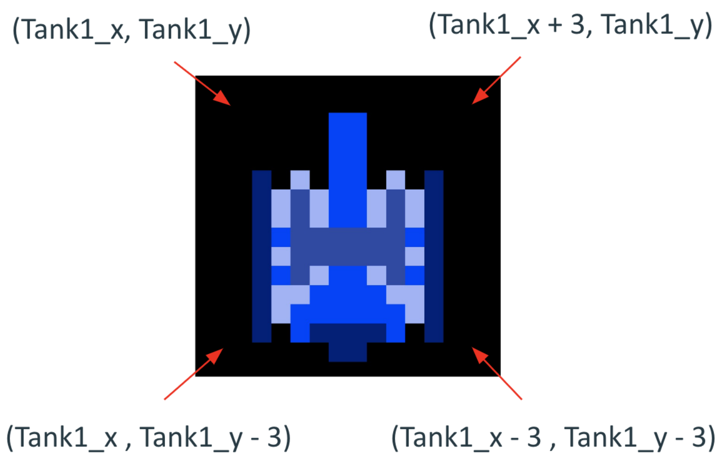
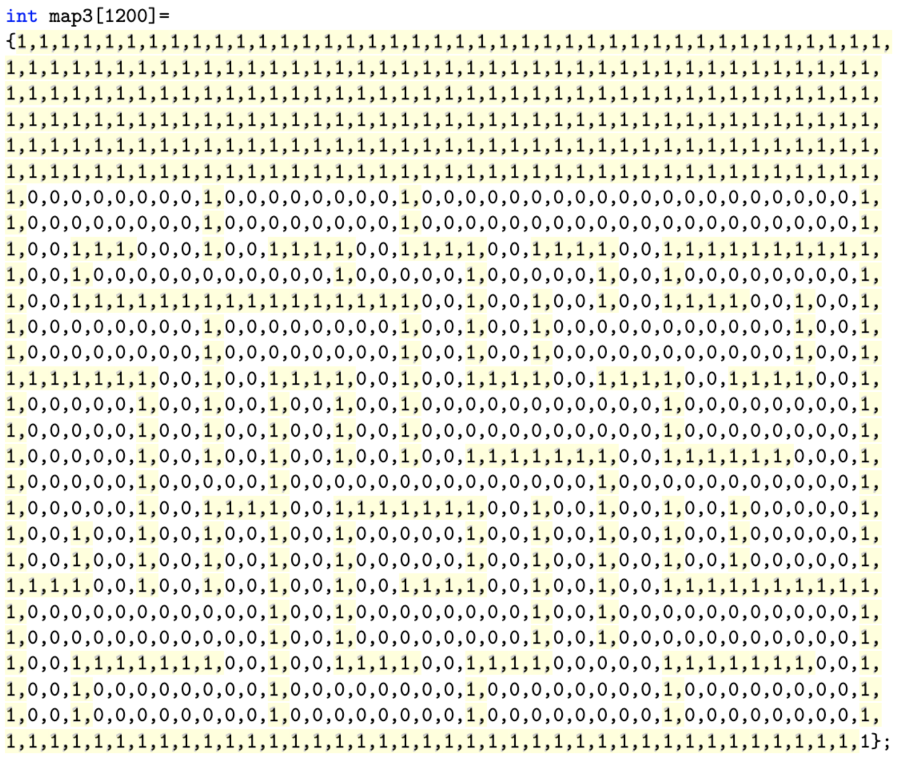


Figure 12: Tank Corner Coordinate Calculations

We saved a bit array of 1200 in the software corresponding to the shape of our map. 1 in the array represents where a wall is present, and 0 means it is space that the tank can travel through.



```
int map3[1200]=
{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,1,1,1,0,0,0,1,0,0,1,1,1,1,0,0,1,1,1,1,0,0,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,
1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,1,
1,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,0,0,1,0,0,1,0,0,1,1,1,1,0,0,1,0,0,1,
1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,
1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,
1,1,1,1,1,1,1,0,0,1,0,0,1,1,1,1,0,0,1,0,0,1,1,1,1,0,0,1,1,1,1,0,0,1,1,1,1,0,0,1,
1,0,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,0,0,0,1,
1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,1,0,0,1,1,1,1,0,0,1,1,1,1,1,1,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0,0,0,1,
1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0,0,0,1,
1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0,0,0,1,
1,1,1,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,1,1,1,0,0,1,0,0,1,0,0,1,1,1,1,1,1,1,1,1,1,
1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,1,1,1,1,1,1,1,0,0,1,0,0,1,1,1,1,0,0,1,1,1,1,0,0,0,0,0,1,1,1,1,1,1,1,0,0,1,
1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,
1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
```

Figure 13: Map duplicate in software

The following algorithm is used to calculate whether the tank will collide with the map:

```
int index = x + y * 8;
int index1 = x+3 + y * 8;
int index2 = x + (y-3) * 8;
int index3 = x-3 + (y-3) * 8;

if (index >= 0 && index < 64 && (map[index] == 1||
map[index1] == 1 || map[index2] == 1 || map[index3] == 1)) {
   return 1;
} else {
   return 0;
}
```

Figure 14: Collision Detection Tank with Map Algorithm

If returns 1, then it indicates a collision. If returns 0, then the tank can continue to move at its assigned direction.

4.2.5 Collision Detection (Bullet with Tank)

Bullet coordinates are in grids of 120 times 160. Tanks are in tiles of 4 times 4 grids in size. Thus this collision detection is simple: if bullet x coordinate is between the max and min x coordinate of tank, AND if bullet y coordinate is between the max and min y coordinate of tank, then a collision has happened.

```
if bullet_x between Tank1_x and Tank1_x - 3,
AND bullet_y between Tank1_y and Tank1_y - 3) {
    return 1;
}
    return 0;
}
```

Figure 15: Collision Detection Bullet with Tank Algorithm

4.2.6 Bullet Firing and Bouncing

The tank movements are detected in one loop. However, bullet firing can no longer be done in one loop because latency in detection will be much longer due to the length of the infinite for loop. Thus we have used pthreads to achieve parallel calculations of bullet location for both bullets.

Bullet bounces off walls at 90 degrees if the direction that it was shot out at was either North, South, East or West. Bullets will bounce in straight lines horizontally or vertically 15 times until it disappears or until it hits a tank.

Bullet bounces off walls at 45 degrees if the direction that it was shot out at was either North-East, SouthEast, NorthWest or SouthWest. Bullets will bounce in diagonal lines 15 times until it disappears or until it hits a tank. This allows the bullet to travel through the map as it continuously bounces off walls.

4.2.7 Collision Detection (Bullet with Wall)

If the direction of tank is North, South, East or West, then the bullet is shot at the same direction.

If a bullet is originally shot at East direction. This collision can be detected by: if bullet x coordinate + 1 collides with map, then bullet direction changes from East to West.

If a bullet is at West, then if bullet x coordinate - 1 collides with map, then bullet direction changes to East.

If a bullet is at North, then if bullet y coordinate + 1 collides with map, then bullet direction changes to South.

If bullet is facing South, then if bullet y coordinate - 1 collides with map, then bullet direction changes to North.

However, for 45 degree bouncing, the case is a little more complicated. For each direction where the bullet is shot, there exists two possibilities shown in the diagrams below:

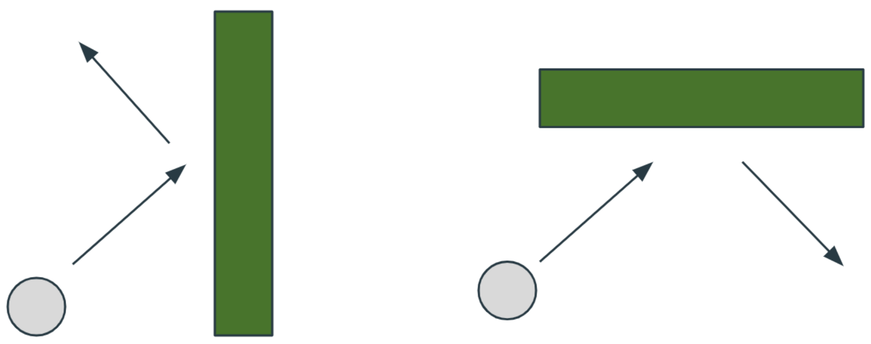


Figure 16: Bullet Bouncing Diagram

In order to tell which is happening and which is not, we can calculate by the following example:

If the direction of bullet is originally NorthEast, then if bullet x coordinate + 1 collides with map, then NorthEast changes to NorthWest. if bullet y coordinate + 1 collides with map, then NorthEast changes to SouthEast.

4.2.8 End Game

When the HP of any of the tank is zero, end bit is high and game displays gameover. After usleep(4000000), the game restarts at the map selection stage.

5 Discussion

5.1 Challenges

- We originally wanted some very pretty tanks. However, due to only allocating 16 times 16 pixels for our tank, the resolution of the tanks decreased very heavily when they are converted to mif files as some colors were merged and looked blurry. Thus, in order to begin with high resolution, we hand drawn some tank sprites using piskel https://www.piskelapp.com/ at specifically 16 times 16 sizes. This allowed tanks to look relatively high resoluted.
- Upon realisation that two bullet movement and two tank movements cannot be calculated in the same loop, we adopted pthreads so that no latency was seen. This is very successful as bullets can move parallel without any latency.
- For tank direction to match the sprite direction took us some time. It was hard matching shoot direction, facing direction, and rotating direction so that it is all correct. It took individual testing and verification before the next step was carried out.

5.2 Further Improvements

- Tank and bullet movements are right now every 4 pixel. There exist a pixel jump and although it looks smooth enough, it is not the smoothest possible. Thus, using a 16 bit address for each x coordinate can eliminate this issue for a 480 times 640 display.

- We can make our design more coherent in terms of color usage. Currently we only have two main colors for the game. This is not visually too pleasing.
- Lags exists when both tanks are moving at the same time. Although not a lot. Another pthread can be used in this context to eliminate this issue. This may require some logic alteration.
- we can use gifs for making the display of the game look cooler. We have not yet experimented with this and could be done in the future. Such as when tanks are being hit and creates an explosion.

5.3 Workload Distribution

- Yiyang (Annie) Peng and Yizhi Wang worked on software.
- Xuanbo Xu and Jiayi Wang worked on hardware.

6 References

https://www.piskelapp.com/
https://www.cs.columbia.edu/ sedwards/classes/2023/4840-spring/reports/Tank-report.pdf

7 Codes

1. hello.c

```c
#include <stdio.h>
#include "vga_ball.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <libusb-1.0/libusb.h>
#include "joystick.h"
#include <pthread.h>

int vga_ball_fd;

vga_ball_color_t color;

int map = 0;

pthread_t pthread1, pthread2;

int tank1_hit = 0;
int tank2_hit = 0;
int bullet1_gone = 0;
int bullet2_gone = 0;

unsigned int tank1_score = 0;
unsigned int tank2_score = 0;

struct bullet_packet {
    unsigned char bulletloc_x;
    unsigned char bulletloc_y;
};

struct bullet_packet entry1;
struct bullet_packet entry2;



unsigned char game_info_1 = 0b00000000;
unsigned char game_info_2 = 0b00000000;
unsigned char game_info_3 = 0b00000000;
unsigned char tank1loc_x = 0b00010000;
unsigned char tank1loc_y = 0b01000100;
unsigned char tank2loc_x = 0b10001100;
unsigned char tank2loc_y = 0b01000100;
unsigned char tank1_dirc = 0b00000010;
unsigned char tank2_dirc = 0b00000110;
unsigned char bullet1loc_x = 0b00000000;
unsigned char bullet1loc_y = 0b00000000;
unsigned char bullet2loc_x = 0b00000000;
unsigned char bullet2loc_y = 0b00000000;
unsigned char bullet1_dirc = 0b00000000; // bullet enable
unsigned char bullet2_dirc = 0b00000000;
unsigned char explo_loc_x = 0b00000000;
unsigned char explo_loc_y = 0b00000000;

int bullet1_en = 0;
int bullet2_en = 0;

int map1[1200]=
{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,1,1,1,1,1,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,1,1,1,1,1,0,0,0,1,
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,
1,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,1,
1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,
1,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,1,
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,
1,0,0,0,1,1,1,1,1,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,1,1,1,1,1,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};

int map2[1200]=
{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,1,
1,0,0,0,0,1,0,1,0,0,1,1,1,1,1,1,1,1,0,0,0,0,1,1,1,1,1,1,1,1,0,0,1,0,1,0,0,0,0,1,
1,0,0,0,0,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,1,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,1,1,1,1,1,1,1,1,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,1,1,1,0,0,1,0,0,0,0,0,1,1,1,0,0,1,1,1,0,0,0,0,0,1,0,0,1,1,1,0,0,0,0,1,
1,0,0,0,0,1,0,1,0,0,1,0,0,0,0,0,1,1,1,0,0,1,1,1,0,0,0,0,0,1,0,0,1,0,1,0,0,0,0,1,
1,0,0,0,0,1,0,0,0,0,1,1,1,1,0,0,1,1,0,0,0,0,1,1,0,0,1,1,1,1,0,0,0,0,1,0,0,0,0,1,
1,0,0,0,0,1,0,1,0,0,1,0,0,0,0,0,1,1,1,0,0,1,1,1,0,0,0,0,0,1,0,0,1,0,1,0,0,0,0,1,
1,0,0,0,0,1,1,1,0,0,1,0,0,0,0,0,1,1,1,0,0,1,1,1,0,0,0,0,0,1,0,0,1,1,1,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,1,
1,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1,1,1,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};

int map3[1200]=
{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,1,1,1,0,0,0,1,0,0,1,1,1,1,0,0,1,1,1,1,0,0,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,
1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,1,
1,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,0,0,1,0,0,1,0,0,1,1,1,1,0,0,1,0,0,1,
1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,
1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,
1,1,1,1,1,1,1,0,0,1,0,0,1,1,1,1,0,0,1,0,0,1,1,1,1,0,0,1,1,1,1,0,0,1,1,1,1,0,0,1,
1,0,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,0,0,0,1,
1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,1,0,0,1,1,1,1,0,0,1,1,1,1,1,1,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0,0,0,1,
1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0,0,0,1,
1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0,0,0,1,
1,1,1,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,1,1,1,0,0,1,0,0,1,0,0,1,1,1,1,1,1,1,1,1,1,
1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,1,1,1,1,1,1,1,0,0,1,0,0,1,1,1,1,0,0,1,1,1,1,0,0,0,0,0,1,1,1,1,1,1,1,0,0,1,
1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,
1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};

/* Set the background color */
```

```c
void set_background_color(const vga_ball_color_t *c){
    vga_ball_arg_t vla;
    vla.background = *c;
    if (ioctl(vga_ball_fd, VGA_BALL_WRITE_BACKGROUND, &vla)) {
        perror("ioctl(VGA_BALL_SET_BACKGROUND) failed");
        return;
    }
}

int collision_wall(char coordinate_x, char coordinate_y) {

    int x = (unsigned char)coordinate_x;
    int x1 = (unsigned char) coordinate_x + 1;
    int x2 = (unsigned char) coordinate_x + 2;
    int x3 = (unsigned char) coordinate_x + 3;
    int y = (unsigned char)coordinate_y;
    int y1 = (unsigned char) coordinate_y + 1;
    int y2 = (unsigned char) coordinate_y + 2;
    int y3 = (unsigned char) coordinate_y + 3;
    int block_x = x / 4;
    int block_y = y / 4;
    int block_x3 = x3 / 4;
    int block_y3 = y3 / 4;
    int map_index = block_x + block_y * 40;
    int map_index1 = block_x + block_y3 * 40;
    int map_index2 = block_x3 + block_y * 40;
    int map_index3 = block_x3 + block_y3 * 40;
    if (map == 0){
        if (map_index >= 0 && map_index < 1200 && (map1[map_index] == 1||
            map1[map_index1] == 1 || map1[map_index2] == 1 || map1[map_index3] ==
            1)) {
            return 1;
        } else {
            return 0;
        }
    } else if (map == 1) {
        if (map_index >= 0 && map_index < 1200 && (map2[map_index] == 1||
            map2[map_index1] == 1 || map2[map_index2] == 1 || map2[map_index3] ==
            1)) {
            return 1;
        } else {
            return 0;
        }
    } else if (map == 2) {
   if (map_index >= 0 && map_index < 1200 && (map3[map_index] == 1||
       map3[map_index1] == 1 || map3[map_index2] == 1 || map3[map_index3] == 1)) {
            return 1;
        } else {
            return 0;
        }
    }
```

```
}

// coordinate 1 is bullet and coordinate 2 is tank.
int collision_tank(unsigned char coordinate_x_1, unsigned char coordinate_y_1,
    unsigned char coordinate_x_2, unsigned char coordinate_y_2) {
    int x1 = coordinate_x_1;
    int y1 = coordinate_y_1;
    int x2 = coordinate_x_2;
    int y2 = coordinate_y_2;

    int tank_right = x2 + 5;
    int tank_bottom = y2 + 5;

    if (x1 < tank_right && x1 > x2 && y1 < tank_bottom && y1 > y2) {
        return 1;
    }
    return 0;
}

int collision_bullettank(unsigned char coordinate_x_1, unsigned char
    coordinate_y_1, unsigned char coordinate_x_2, unsigned char coordinate_y_2) {

    unsigned char tank_x_max = coordinate_x_2 + 3;
    unsigned char tank_y_min = coordinate_y_2 + 3;

    if (coordinate_x_1 >= coordinate_x_2 && coordinate_x_1 <= tank_x_max &&
        coordinate_y_1 <= tank_y_min && coordinate_y_1 >= coordinate_y_2) {
        return 1;
    }

    return 0;
}

int collision_bulletwall(unsigned char coordinate_x, unsigned char coordinate_y){
    int x = (unsigned char) coordinate_x;
    int y = (unsigned char) coordinate_y;
    int block_x = x / 4;
    int block_y = y / 4;
    int map_index = block_x + block_y * 40;
    if (map == 0){
        if (map_index >= 0 && map_index < 1200 && map1[map_index] == 1) {
            return 1;
        } else {
            return 0;
        }
    } else if (map == 1) {
        if (map_index >= 0 && map_index < 1200 && map2[map_index] == 1) {
            return 1;
        } else {
            return 0;
        }
    } else if (map == 2) {
   if (map_index >= 0 && map_index < 1200 && map3[map_index] == 1) {
```

```
            return 1;
        } else {
            return 0;
        }
    }
}

void *fire_bullet1(void *entry1){

    int bullet1_wallhit = 0;

    struct bullet_packet *bullet_packet = (struct bullet_packet *) entry1;
    unsigned char bullet1loc_x = bullet_packet->bulletloc_x;
    unsigned char bullet1loc_y = bullet_packet->bulletloc_y;
    unsigned char bullet1_dirc = tank1_dirc;

    if (tank1_dirc == 0b00000000){
        bullet1loc_x = tank1loc_x + 2;
        bullet1loc_y = tank1loc_y - 1;
    } else if (tank1_dirc == 0b00000001){
        bullet1loc_x = tank1loc_x + 4;
        bullet1loc_y = tank1loc_y - 1;
    } else if (tank1_dirc == 0b00000010){
        bullet1loc_x = tank1loc_x + 4;
        bullet1loc_y = tank1loc_y + 2;
    } else if (tank1_dirc == 0b00000011){
        bullet1loc_x = tank1loc_x + 4;
        bullet1loc_y = tank1loc_y + 4;
    } else if (tank1_dirc == 0b00000100){
        bullet1loc_x = tank1loc_x + 2;
        bullet1loc_y = tank1loc_y + 4;
    } else if (tank1_dirc == 0b00000101){
        bullet1loc_x = tank1loc_x - 1;
        bullet1loc_y = tank1loc_y + 4;
    } else if (tank1_dirc == 0b00000110){
        bullet1loc_x = tank1loc_x - 1;
        bullet1loc_y = tank1loc_y + 2;
    } else if (tank1_dirc == 0b00000111){
        bullet1loc_x = tank1loc_x - 1;
        bullet1loc_y = tank1loc_y - 1;
    }
    color.bullet1loc_x = bullet1loc_x;
    color.bullet1loc_y = bullet1loc_y;
    set_background_color(&color);

    for (;;){
        if (bullet1_wallhit > 15){
            bullet1_gone = 1;
            break;
        } else if (bullet1_dirc == 0b00000000){
            if (collision_bullettank(bullet1loc_x, bullet1loc_y, tank1loc_x,
                tank1loc_y) == 1){
                tank1_hit = 1;
                break;
            } else if (collision_bullettank(bullet1loc_x, bullet1loc_y, tank2loc_x,
                tank2loc_y) == 1){
                tank2_hit = 1;
                break;
            } else if (collision_bulletwall(bullet1loc_x, bullet1loc_y - 1) == 1){
                bullet1_dirc = 0b00000100;
                bullet1_wallhit += 1;
            } else {
                bullet1loc_y -= 1;
            }
        } else if (bullet1_dirc == 0b00000001){
            if (collision_bullettank(bullet1loc_x, bullet1loc_y, tank1loc_x,
                tank1loc_y) == 1){
                tank1_hit = 1;
                break;
            } else if (collision_bullettank(bullet1loc_x, bullet1loc_y, tank2loc_x,
                tank2loc_y) == 1){
                tank2_hit = 1;
                break;
            } else if (collision_bulletwall(bullet1loc_x+1, bullet1loc_y) == 1){
                bullet1_dirc = 0b00000111;
                bullet1_wallhit += 1;
            } else if (collision_bulletwall(bullet1loc_x, bullet1loc_y-1) == 1){
                bullet1_dirc = 0b00000011;
                bullet1_wallhit += 1;
            } else {
                bullet1loc_x += 1;
                bullet1loc_y -= 1;
            }
        } else if (bullet1_dirc == 0b00000010){
            if (collision_bullettank(bullet1loc_x, bullet1loc_y, tank1loc_x,
                tank1loc_y) == 1){
                tank1_hit = 1;
                break;
            } else if (collision_bullettank(bullet1loc_x, bullet1loc_y, tank2loc_x,
                tank2loc_y) == 1){
                tank2_hit = 1;
                break;
            } else if (collision_bulletwall(bullet1loc_x + 1, bullet1loc_y) == 1){
                bullet1_dirc = 0b00000110;
                bullet1_wallhit += 1;
            } else {
                bullet1loc_x += 1;
            }
        } else if (bullet1_dirc == 0b00000011){
            if (collision_bullettank(bullet1loc_x, bullet1loc_y, tank1loc_x,
                tank1loc_y) == 1){
                tank1_hit = 1;
                break;
            } else if (collision_bullettank(bullet1loc_x, bullet1loc_y, tank2loc_x,
                tank2loc_y) == 1){
                tank2_hit = 1;
                break;
            } else if (collision_bulletwall(bullet1loc_x+1, bullet1loc_y) == 1){
                bullet1_dirc = 0b00000101;
                bullet1_wallhit += 1;
            } else if (collision_bulletwall(bullet1loc_x, bullet1loc_y+1) == 1){
                bullet1_dirc = 0b00000001;
                bullet1_wallhit += 1;
            } else {
                bullet1loc_x += 1;
                bullet1loc_y += 1;
            }
        } else if (bullet1_dirc == 0b00000100){
            if (collision_bullettank(bullet1loc_x, bullet1loc_y, tank1loc_x,
                tank1loc_y) == 1){
                tank1_hit = 1;
                break;
            } else if (collision_bullettank(bullet1loc_x, bullet1loc_y, tank2loc_x,
                tank2loc_y) == 1){
                tank2_hit = 1;
                break;
            } else if (collision_bulletwall(bullet1loc_x, bullet1loc_y + 1) == 1){
                bullet1_dirc = 0b00000000;
                bullet1_wallhit += 1;
            } else {
                bullet1loc_y += 1;
            }
        } else if (bullet1_dirc == 0b00000101){
            if (collision_bullettank(bullet1loc_x, bullet1loc_y, tank1loc_x,
                tank1loc_y) == 1){
                tank1_hit = 1;
                break;
            } else if (collision_bullettank(bullet1loc_x, bullet1loc_y, tank2loc_x,
                tank2loc_y) == 1){
                tank2_hit = 1;
                break;
            } else if (collision_bulletwall(bullet1loc_x-1, bullet1loc_y) == 1){
                bullet1_dirc = 0b00000011;
                bullet1_wallhit += 1;
            } else if (collision_bulletwall(bullet1loc_x, bullet1loc_y+1) == 1){
                bullet1_dirc = 0b00000111;
                bullet1_wallhit += 1;
            } else {
                bullet1loc_x -= 1;
                bullet1loc_y += 1;
            }
        } else if (bullet1_dirc == 0b00000110){
            if (collision_bullettank(bullet1loc_x, bullet1loc_y, tank1loc_x,
                tank1loc_y) == 1){
                tank1_hit = 1;
                break;
            } else if (collision_bullettank(bullet1loc_x, bullet1loc_y, tank2loc_x,
                tank2loc_y) == 1){
                tank2_hit = 1;
```

```
                break;
            } else if (collision_bulletwall(bullet1loc_x-1, bullet1loc_y) == 1){
                bullet1_dirc = 0b00000010;
                bullet1_wallhit += 1;
            } else {
                bullet1loc_x -= 1;
            }
        } else if (bullet1_dirc == 0b00000111){
            if (collision_bullettank(bullet1loc_x, bullet1loc_y, tank1loc_x,
                tank1loc_y) == 1){
                tank1_hit = 1;
                break;
            } else if (collision_bullettank(bullet1loc_x, bullet1loc_y, tank2loc_x,
                tank2loc_y) == 1){
                tank2_hit = 1;
                break;
            } else if (collision_bulletwall(bullet1loc_x-1, bullet1loc_y) == 1){
                bullet1_dirc = 0b00000001;
                bullet1_wallhit += 1;
            } else if (collision_bulletwall(bullet1loc_x, bullet1loc_y-1) == 1){
                bullet1_dirc = 0b00000101;
                bullet1_wallhit += 1;
            } else {
                bullet1loc_x -= 1;
                bullet1loc_y -= 1;
            }
        }
        usleep(17000);
        color.bullet1loc_x = bullet1loc_x;
        color.bullet1loc_y = bullet1loc_y;
        set_background_color(&color);
    }
}

// if break without any hits, bullet disappear
// if break with tank 1 hit or tank2 hit, then do whatever needed end thread or smt
void *fire_bullet2(void *entry2){

    int bullet2_wallhit = 0;

    struct bullet_packet *bullet_packet = (struct bullet_packet *) entry2;
    unsigned char bullet2loc_x = bullet_packet->bulletloc_x;
    unsigned char bullet2loc_y = bullet_packet->bulletloc_y;
    unsigned char bullet2_dirc = tank2_dirc;

    if (tank2_dirc == 0b00000000){
        bullet2loc_x = tank2loc_x + 2;
        bullet2loc_y = tank2loc_y - 1;
    } else if (tank2_dirc == 0b00000001){
        bullet2loc_x = tank2loc_x + 4;
        bullet2loc_y = tank2loc_y - 1;
    } else if (tank2_dirc == 0b00000010){
        bullet2loc_x = tank2loc_x + 4;
    bullet2loc_y = tank2loc_y + 2;
} else if (tank2_dirc == 0b00000011){
    bullet2loc_x = tank2loc_x + 4;
    bullet2loc_y = tank2loc_y + 4;
} else if (tank2_dirc == 0b00000100){
    bullet2loc_x = tank2loc_x + 2;
    bullet2loc_y = tank2loc_y + 4;
} else if (tank2_dirc == 0b00000101){
    bullet2loc_x = tank2loc_x - 1;
    bullet2loc_y = tank2loc_y + 4;
} else if (tank2_dirc == 0b00000110){
    bullet2loc_x = tank2loc_x - 1;
    bullet2loc_y = tank2loc_y + 2;
} else if (tank2_dirc == 0b00000111){
    bullet2loc_x = tank2loc_x - 1;
    bullet2loc_y = tank2loc_y - 1;
}
color.bullet2loc_x = bullet2loc_x;
color.bullet2loc_y = bullet2loc_y;
set_background_color(&color);

for (;;){
    if (bullet2_wallhit > 15){
        bullet2_gone = 1;
        break;
    } else if (bullet2_dirc == 0b00000000){
        if (collision_bullettank(bullet2loc_x, bullet2loc_y, tank1loc_x,
            tank1loc_y) == 1){
            tank1_hit = 1;
            break;
        } else if (collision_bullettank(bullet2loc_x, bullet2loc_y, tank2loc_x,
            tank2loc_y) == 1){
            tank2_hit = 1;
            break;
        } else if (collision_bulletwall(bullet2loc_x, bullet2loc_y - 1) == 1){
            bullet2_dirc = 0b00000100;
            bullet2_wallhit += 1;
        } else {
            bullet2loc_y -= 1;
        }tank2_dirc = 0b00000000;
                color.tank2_dirc = 0b00000000;
                set_background_color(&color);

    } else if (bullet2_dirc == 0b00000001){
        if (collision_bullettank(bullet2loc_x, bullet2loc_y, tank1loc_x,
            tank1loc_y) == 1){
            tank1_hit = 1;
            break;
        } else if (collision_bullettank(bullet2loc_x, bullet2loc_y, tank2loc_x,
            tank2loc_y) == 1){
            tank2_hit = 1;
            break;
        } else if (collision_bulletwall(bullet2loc_x+1, bullet2loc_y) == 1){
            bullet2_dirc = 0b00000111;
            bullet2_wallhit += 1;
        } else if (collision_bulletwall(bullet2loc_x, bullet2loc_y-1) == 1){
            bullet2_dirc = 0b00000011;
            bullet2_wallhit += 1;
        } else {
            bullet2loc_x += 1;
            bullet2loc_y -= 1;
        }
    } else if (bullet2_dirc == 0b00000010){
        if (collision_bullettank(bullet2loc_x, bullet2loc_y, tank1loc_x,
            tank1loc_y) == 1){
            tank1_hit = 1;
            break;
        } else if (collision_bullettank(bullet2loc_x, bullet2loc_y, tank2loc_x,
            tank2loc_y) == 1){
            tank2_hit = 1;
            break;
        } else if (collision_bulletwall(bullet2loc_x + 1, bullet2loc_y) == 1){
            bullet2_dirc = 0b00000110;
            bullet2_wallhit += 1;
        } else {
            bullet2loc_x += 1;
        }
    } else if (bullet2_dirc == 0b00000011){
        if (collision_bullettank(bullet2loc_x, bullet2loc_y, tank1loc_x,
            tank1loc_y) == 1){
            tank1_hit = 1;
            break;
        } else if (collision_bullettank(bullet2loc_x, bullet2loc_y, tank2loc_x,
            tank2loc_y) == 1){
            tank2_hit = 1;
            break;
        } else if (collision_bulletwall(bullet2loc_x+1, bullet2loc_y) == 1){
            bullet2_dirc = 0b00000101;
            bullet2_wallhit += 1;
        } else if (collision_bulletwall(bullet2loc_x, bullet2loc_y+1) == 1){
            bullet2_dirc = 0b00000001;
            bullet2_wallhit += 1;
        } else {
            bullet2loc_x += 1;
            bullet2loc_y += 1;
        }
    } else if (bullet2_dirc == 0b00000100){
        if (collision_bullettank(bullet2loc_x, bullet2loc_y, tank1loc_x,
            tank1loc_y) == 1){
            tank1_hit = 1;
            break;
        } else if (collision_bullettank(bullet2loc_x, bullet2loc_y, tank2loc_x,
            tank2loc_y) == 1){
            tank2_hit = 1;
            break;
        } else if (collision_bulletwall(bullet2loc_x, bullet2loc_y + 1) == 1){
            bullet2_dirc = 0b00000000;
            bullet2_wallhit += 1;
        } else {
            bullet2loc_y += 1;
        }
    } else if (bullet2_dirc == 0b00000101){
        if (collision_bullettank(bullet2loc_x, bullet2loc_y, tank1loc_x,
            tank1loc_y) == 1){
            tank1_hit = 1;
            break;
        } else if (collision_bullettank(bullet2loc_x, bullet2loc_y, tank2loc_x,
            tank2loc_y) == 1){
            tank2_hit = 1;
            break;
        } else if (collision_bulletwall(bullet2loc_x-1, bullet2loc_y) == 1){
            bullet2_dirc = 0b00000011;
            bullet2_wallhit += 1;
        } else if (collision_bulletwall(bullet2loc_x, bullet2loc_y+1) == 1){
            bullet2_dirc = 0b00000111;
            bullet2_wallhit += 1;
        } else {
            bullet2loc_x -= 1;
            bullet2loc_y += 1;
        }
    } else if (bullet2_dirc == 0b00000110){
        if (collision_bullettank(bullet2loc_x, bullet2loc_y, tank1loc_x,
            tank1loc_y) == 1){
            tank1_hit = 1;
            break;
        } else if (collision_bullettank(bullet2loc_x, bullet2loc_y, tank2loc_x,
            tank2loc_y) == 1){
            tank2_hit = 1;
            break;
        } else if (collision_bulletwall(bullet2loc_x-1, bullet2loc_y) == 1){
            bullet2_dirc = 0b00000010;
            bullet2_wallhit += 1;
        } else {
            bullet2loc_x -= 1;
        }
    } else if (bullet2_dirc == 0b00000111){
        if (collision_bullettank(bullet2loc_x, bullet2loc_y, tank1loc_x,
            tank1loc_y) == 1){
            tank1_hit = 1;
            break;
        } else if (collision_bullettank(bullet2loc_x, bullet2loc_y, tank2loc_x,
            tank2loc_y) == 1){
            tank2_hit = 1;
            break;
        } else if (collision_bulletwall(bullet2loc_x-1, bullet2loc_y) == 1){
            bullet2_dirc = 0b00000001;
            bullet2_wallhit += 1;
        } else if (collision_bulletwall(bullet2loc_x, bullet2loc_y-1) == 1){
            bullet2_dirc = 0b00000101;
```

```
                bullet2_wallhit += 1;
            } else {
                bullet2loc_x -= 1;
                bullet2loc_y -= 1;
            }
        }
        usleep(17000);
        color.bullet2loc_x = bullet2loc_x;
        color.bullet2loc_y = bullet2loc_y;
        set_background_color(&color);
    }
}

int main(){

    vga_ball_arg_t vla;
    static const char filename[] = "/dev/vga_ball";

    static vga_ball_color_t colors[] = {
        { 0x00, 0x00, 0x00, 0x9f, 0x00, 0x9f, 0x00, 0x01 }, /* Red */
        { 0x00, 0xff, 0x00, 0x9f, 0x00, 0x9f, 0x00, 0x01 }, /* Green */
        { 0x00, 0x00, 0xff, 0x9f, 0x00, 0x9f, 0x00, 0x01 }, /* Blue */
        { 0xff, 0xff, 0x00, 0x9f, 0x00, 0x9f, 0x00, 0x01 }, /* Yellow */
        { 0x00, 0xff, 0xff, 0x90, 0x00, 0x9f, 0x00, 0x01 }, /* Cyan */
        { 0xff, 0x00, 0xff, 0xA0, 0x00, 0x9f, 0x00, 0x01 }, /* Magenta */
        { 0x80, 0x80, 0x80, 0xB0, 0x00, 0x9f, 0x00, 0x01 }, /* Gray */
        { 0x00, 0x00, 0x00, 0xC0, 0x00, 0x9f, 0x00, 0x01 }, /* Black */
        { 0xff, 0xff, 0xff, 0xD0, 0x00, 0x9f, 0x00, 0x01 } /* White */
    };

    # define COLORS 9

    printf("TANK GO! userspace program started\n");
    if ( (vga_ball_fd = open(filename, O_RDWR)) == -1) {
        fprintf(stderr, "could not open %s\n", filename);
        return -1;
    }

    struct joystick_admin joysticks = open_controllers();

    struct joystick_packet joystick1;
    struct joystick_packet joystick2;

    int length1 = sizeof(joystick1);
    int length2 = sizeof(joystick2);

    int array1;
    int array2;

    for (;;){

        map = 0;
        int map_up = 0;
        int map_down = 0;

        int tank1_rotleft = 0;
        int tank1_rotright = 0;
        int tank1_n = 0;
        int tank1_ne = 0;
        int tank1_e = 0;
        int tank1_se = 0;
        int tank1_s = 0;
        int tank1_sw = 0;
        int tank1_w = 0;
        int tank1_nw = 0;
        int tank1_fire = 0;
        int tank1_score = 0;

        int tank2_rotleft = 0;
        int tank2_rotright = 0;
        int tank2_n = 0;
        int tank2_ne = 0;
        int tank2_e = 0;
        int tank2_se = 0;
        int tank2_s = 0;
        int tank2_sw = 0;
        int tank2_w = 0;
        int tank2_nw = 0;
        int tank2_fire = 0;
        int tank2_score = 0;

    game_info_1 = 0b00000000;
    game_info_2 = 0b00000000;
    game_info_3 = 0b00000000;
    tank1loc_x = 0b00010000;
    tank1loc_y = 0b01000100;
    tank2loc_x = 0b10001100;
    tank2loc_y = 0b01000100;
    tank1_dirc = 0b00000010;
    tank2_dirc = 0b00000110;
    bullet1loc_x = 0b00000000;
    bullet1loc_y = 0b00000000;
    bullet2loc_x = 0b00000000;
    bullet2loc_y = 0b00000000;
    bullet1_dirc = 0b00000000; // bullet enable
    bullet2_dirc = 0b00000000;
    explo_loc_x = 0b00000000;
    explo_loc_y = 0b00000000;

        color.game_info_1 = game_info_1;
        color.game_info_2 = game_info_2;
        color.game_info_3 = game_info_3;
        color.tank1loc_x = tank1loc_x;
        color.tank1loc_y = tank1loc_y;
        color.tank2loc_x = tank2loc_x;
```

```
color.tank2loc_y = tank2loc_y;
color.tank1_dirc = tank1_dirc;
color.tank2_dirc = tank2_dirc;
color.bullet1loc_x = bullet1loc_x;
color.bullet1loc_y = bullet1loc_y;
color.bullet2loc_x = bullet2loc_x;
color.bullet2loc_y = bullet2loc_y;
color.bullet1_dirc = bullet1_dirc;
color.bullet2_dirc = bullet2_dirc;
color.explo_loc_x = explo_loc_x;
color.explo_loc_y = explo_loc_y;

set_background_color(&color);

for (;;) {

    libusb_interrupt_transfer(joysticks.joystick1, joysticks.joystick1_addr,
        (unsigned char *) &joystick1, length1, &array1, 0);

    if (array1 == 7){

        uint8_t button1 = joystick1.xyab;

        if (joystick1.v_dirc == 0) { // up

            map_up += 1;
            if (map_up >= 600){
                map_up = 0;
            }

            if (game_info_1 == 0b00000000){
                map = 1;
                game_info_1 = 0b00000001;
 color.game_info_1 = game_info_1;
 set_background_color(&color);
 usleep(150000);
            } else if (game_info_1 == 0b00000001){
                map = 2;
                game_info_1 = 0b00000010;
 usleep(150000);
            }
            color.game_info_1 = game_info_1;
            set_background_color(&color);

        } else if (joystick1.v_dirc == 255) { // down

            map_down += 1;
            if (map_down >= 600){
                map_down = 0;
            }

            if (game_info_1 == 0b00000010){
                map = 1;
                game_info_1 = 0b00000001;
   color.game_info_1 = game_info_1;
                set_background_color(&color);
   usleep(150000);
             } else if (game_info_1 == 0b00000001){
                map = 0;
                game_info_1 = 0b00000000;
   color.game_info_1 = game_info_1;
                set_background_color(&color);
   usleep(150000);
             }

             color.game_info_1 = game_info_1;
             set_background_color(&color);

         } else if (button1 == 31) {

             tank1_fire += 1;
             if (tank1_fire >= 600) {
                 tank1_fire = 0;
             }

             printf("Selected Map %d \n", map + 1);
             break;
         }
     }
  }

  for (;;){

 if (bullet1_gone == 1){
pthread_cancel(pthread1);
bullet1loc_x = 0b00000000;
bullet1loc_y = 0b00000000;
bullet1_en = 0;
bullet1_gone = 0;
color.bullet1loc_x = bullet1loc_x;
color.bullet1loc_y = bullet1loc_y;
set_background_color(&color);
 }
 if (bullet2_gone == 1){
         pthread_cancel(pthread2);
         bullet2loc_x = 0b00000000;
         bullet2loc_y = 0b00000000;
bullet2_en = 0;
bullet2_gone = 0;
         color.bullet2loc_x = bullet2loc_x;
         color.bullet2loc_y = bullet2loc_y;
         set_background_color(&color);
     }
 if (tank1_hit == 1 || tank2_hit == 1){
         if (tank1_hit == 1){
             tank1_score += 2;
         tank1_hit = 0;
pthread_cancel(pthread1);
         pthread_cancel(pthread2);

color.tank1_dirc = 0b00000000;
set_background_color(&color);
usleep(10000);
color.tank1_dirc = 0b00000001;
set_background_color(&color);
usleep(10000);
color.tank1_dirc = 0b00000010;
set_background_color(&color);
usleep(10000);
color.tank1_dirc = 0b00000011;
         set_background_color(&color);
         usleep(10000);
         color.tank1_dirc = 0b00000100;
         set_background_color(&color);
         usleep(10000);
         color.tank1_dirc = 0b00000101;
         set_background_color(&color);
         usleep(10000);
color.tank1_dirc = 0b00000110;
         set_background_color(&color);
         usleep(10000);
         color.tank1_dirc = 0b00000111;
         set_background_color(&color);
         usleep(10000);
color.tank1_dirc = 0b00000000;
set_background_color(&color);
usleep(10000);
         color.tank1_dirc = 0b00000001;
         set_background_color(&color);
         usleep(10000);
         color.tank1_dirc = 0b00000010;
         set_background_color(&color);
         usleep(10000);
         color.tank1_dirc = 0b00000011;
         set_background_color(&color);
         usleep(10000);
         color.tank1_dirc = 0b00000100;
         set_background_color(&color);
         usleep(10000);
         color.tank1_dirc = 0b00000101;
         set_background_color(&color);
         usleep(10000);
         color.tank1_dirc = 0b00000110;
         set_background_color(&color);
         usleep(10000);
         color.tank1_dirc = 0b00000111;
         set_background_color(&color);
         usleep(10000);
color.tank1_dirc = 0b00000000;
```

```
          set_background_color(&color);
          usleep(10000);
          color.tank1_dirc = 0b00000001;
          set_background_color(&color);
          usleep(10000);
          color.tank1_dirc = 0b00000010;
          set_background_color(&color);
          usleep(10000);
          color.tank1_dirc = 0b00000011;
          set_background_color(&color);
          usleep(10000);
          color.tank1_dirc = 0b00000100;
          set_background_color(&color);
          usleep(10000);
          color.tank1_dirc = 0b00000101;
          set_background_color(&color);
          usleep(10000);
          color.tank1_dirc = 0b00000110;
usleep(10000);
color.tank1_dirc = 0b00000111;
usleep(10000);
color.tank1_dirc = 0b00000000;
          set_background_color(&color);
          usleep(10000);
          color.tank1_dirc = 0b00000001;
          set_background_color(&color);
          usleep(10000);
          color.tank1_dirc = 0b00000010;
          set_background_color(&color);
          usleep(10000);
          color.tank1_dirc = 0b00000011;
          set_background_color(&color);
          usleep(10000);
          color.tank1_dirc = 0b00000100;
          set_background_color(&color);
          usleep(10000);
          color.tank1_dirc = 0b00000101;
          set_background_color(&color);
          usleep(10000);
          color.tank1_dirc = 0b00000110;
usleep(10000);
color.tank1_dirc = 0b00000111;
usleep(10000);
tank1_dirc = 0b00000000;
          color.tank1_dirc = 0b00000000;
          set_background_color(&color);

      }
      if(tank2_hit == 1){
          tank2_score += 2;
          tank2_hit = 0;
pthread_cancel(pthread1);
          pthread_cancel(pthread2);
```

```
color.tank2_dirc = 0b00000000;
         set_background_color(&color);
         usleep(10000);
         color.tank2_dirc = 0b00000001;
         set_background_color(&color);
         usleep(10000);
         color.tank2_dirc = 0b00000010;
         set_background_color(&color);
         usleep(10000);
         color.tank2_dirc = 0b00000011;
         set_background_color(&color);
         usleep(10000);
         color.tank2_dirc = 0b00000100;
         set_background_color(&color);
         usleep(10000);
         color.tank2_dirc = 0b00000101;
         set_background_color(&color);
         usleep(10000);
         color.tank2_dirc = 0b00000110;
         set_background_color(&color);
         usleep(10000);
         color.tank2_dirc = 0b00000111;
         set_background_color(&color);
usleep(10000);
color.tank1_dirc = 0b00000000;
         set_background_color(&color);
         usleep(10000);
         color.tank2_dirc = 0b00000001;
         set_background_color(&color);
         usleep(10000);
         color.tank2_dirc = 0b00000010;
         set_background_color(&color);
         usleep(10000);
         color.tank2_dirc = 0b00000011;
         set_background_color(&color);
         usleep(10000);
         color.tank2_dirc = 0b00000100;
         set_background_color(&color);
         usleep(10000);
         color.tank2_dirc = 0b00000101;
         set_background_color(&color);
         usleep(10000);
         color.tank2_dirc = 0b00000110;
         set_background_color(&color);
         usleep(10000);
         color.tank2_dirc = 0b00000111;
         set_background_color(&color);
usleep(10000);
color.tank2_dirc = 0b00000000;
         set_background_color(&color);
         usleep(10000);
         color.tank2_dirc = 0b00000001;
```

```
             set_background_color(&color);
             usleep(10000);
             color.tank2_dirc = 0b00000010;
             set_background_color(&color);
             usleep(10000);
             color.tank2_dirc = 0b00000011;
             set_background_color(&color);
             usleep(10000);
             color.tank2_dirc = 0b00000100;
             set_background_color(&color);
             usleep(10000);
             color.tank2_dirc = 0b00000101;
             set_background_color(&color);
             usleep(10000);
             color.tank2_dirc = 0b00000110;
             set_background_color(&color);
             usleep(10000);
             color.tank2_dirc = 0b00000111;
             set_background_color(&color);
    usleep(10000);
    color.tank2_dirc = 0b00000000;
             set_background_color(&color);
             usleep(10000);
             color.tank2_dirc = 0b00000001;
             set_background_color(&color);
             usleep(10000);
             color.tank2_dirc = 0b00000010;
             set_background_color(&color);
             usleep(10000);
             color.tank2_dirc = 0b00000011;
             set_background_color(&color);
             usleep(10000);
             color.tank2_dirc = 0b00000100;
             set_background_color(&color);
             usleep(10000);
             color.tank2_dirc = 0b00000101;
             set_background_color(&color);
             usleep(10000);
             color.tank2_dirc = 0b00000110;
             set_background_color(&color);
             usleep(10000);
             color.tank2_dirc = 0b00000111;
             set_background_color(&color);
    usleep(10000);
    tank2_dirc = 0b00000000;
    color.tank2_dirc = 0b00000000;
    set_background_color(&color);
         }

bullet1_en = 0;
bullet2_en = 0;
         bullet1loc_x = 0b00000000;
         bullet1loc_y = 0b00000000;
```

```
          color.bullet1loc_x = bullet1loc_x;
          color.bullet1loc_y = bullet1loc_y;
bullet2loc_x = 0b00000000;
          bullet2loc_y = 0b00000000;
          color.bullet2loc_x = bullet2loc_x;
          color.bullet2loc_y = bullet2loc_y;

          set_background_color(&color);

if (tank1_score >= 15 || tank2_score >= 15){
              game_info_3 = 0b00000001;
    color.game_info_3 = 0b00000001;
    set_background_color(&color);
    usleep(4000000);
    break;
          }


          tank1_score &= 0x0F;
          tank2_score &= 0x0F;

          game_info_2 = (tank1_score << 4) | tank2_score;

          color.game_info_2 = game_info_2;
          color.game_info_3 = game_info_3;
          color.explo_loc_x = explo_loc_x;
          color.explo_loc_y = explo_loc_y;
          set_background_color(&color);

          usleep(300000);
          explo_loc_x = 0b00000000;
          explo_loc_y = 0b00000000;
          color.explo_loc_x = explo_loc_x;
          color.explo_loc_y = explo_loc_y;
          set_background_color(&color);
 }
      libusb_interrupt_transfer(joysticks.joystick1, joysticks.joystick1_addr,
          (unsigned char *) &joystick1, length1, &array1, 0);
      libusb_interrupt_transfer(joysticks.joystick2, joysticks.joystick2_addr,
          (unsigned char *) &joystick2, length2, &array2, 0);

      if (array1 == 7 && array2 == 7){
          uint8_t button1 = joystick1.xyab;
          if (joystick1.h_dirc == 255){
              tank1_rotright += 1;
              if (tank1_rotright >= 600){
                  tank1_rotright = 0;
              }
              if (tank1_dirc == 0b00000000){
                  tank1_dirc = 0b00000001;
              } else if (tank1_dirc == 0b00000001){
                  tank1_dirc = 0b00000010;
              } else if (tank1_dirc == 0b00000010){
```

```
            tank1_dirc = 0b00000011;
        } else if (tank1_dirc == 0b00000011){
            tank1_dirc = 0b00000100;
        } else if (tank1_dirc == 0b00000100){
            tank1_dirc = 0b00000101;
        } else if (tank1_dirc == 0b00000101){
            tank1_dirc = 0b00000110;
        } else if (tank1_dirc == 0b00000110){
            tank1_dirc = 0b00000111;
        } else if (tank1_dirc == 0b00000111){
            tank1_dirc = 0b00000000;
        }
        color.tank1_dirc = tank1_dirc;
        set_background_color(&color);
        usleep(200000);
    }
    if (joystick1.h_dirc == 0){
        tank1_rotleft += 1;
        if (tank1_rotleft >= 600){
            tank1_rotleft = 0;
        }
        if (tank1_dirc == 0b00000000){
            tank1_dirc = 0b00000111;
        } else if (tank1_dirc == 0b00000111){
            tank1_dirc = 0b00000110;
        } else if (tank1_dirc == 0b00000110){
            tank1_dirc = 0b00000101;
        } else if (tank1_dirc == 0b00000101){
            tank1_dirc = 0b00000100;
        } else if (tank1_dirc == 0b00000100){
            tank1_dirc = 0b00000011;
        } else if (tank1_dirc == 0b00000011){
            tank1_dirc = 0b00000010;
        } else if (tank1_dirc == 0b00000010){
            tank1_dirc = 0b00000001;
        } else if (tank1_dirc == 0b00000001){
            tank1_dirc = 0b00000000;
        }
        color.tank1_dirc = tank1_dirc;
        set_background_color(&color);
        usleep(200000);
    }
    if (joystick1.v_dirc == 255){ // reversing
        if (tank1_dirc == 0b00000000){
            tank1_n += 1;
            if (tank1_n >= 300){
                tank1_n = 0;
            }
            usleep(70000);
            if (collision_wall(tank1loc_x, tank1loc_y+1) == 0){
                tank1loc_y += 1;
            }
            color.tank1loc_y = tank1loc_y;
```

```
} else if (tank1_dirc == 0b00000001){
    tank1_ne += 1;
    if (tank1_ne >= 300){
        tank1_ne = 0;
    }
    usleep(70000);
    if (collision_wall(tank1loc_x-1, tank1loc_y+1) == 0){
        tank1loc_y += 1;
        tank1loc_x -= 1;
    }
    color.tank1loc_y = tank1loc_y;
    color.tank1loc_x = tank1loc_x;
} else if (tank1_dirc == 0b00000010){
    tank1_e += 1;
    if (tank1_e >= 300){
        tank1_e = 0;
    }
    usleep(70000);
    if (collision_wall(tank1loc_x-1, tank1loc_y) == 0){
        tank1loc_x -= 1;
    }
    color.tank1loc_x = tank1loc_x;
} else if (tank1_dirc == 0b00000011){
    tank1_se += 1;
    if (tank1_se >= 300){
        tank1_se = 0;
    }
    usleep(70000);
    if (collision_wall(tank1loc_x-1, tank1loc_y-1) == 0){
        tank1loc_y -= 1;
        tank1loc_x -= 1;
    }
    color.tank1loc_y = tank1loc_y;
    color.tank1loc_x = tank1loc_x;
} else if (tank1_dirc == 0b00000100){
    tank1_s += 1;
    if (tank1_s >= 300){
        tank1_s = 0;
    }
    usleep(70000);
    if (collision_wall(tank1loc_x, tank1loc_y-1) == 0){
        tank1loc_y -= 1;
    }
    color.tank1loc_y = tank1loc_y;
} else if (tank1_dirc == 0b00000101){
    tank1_sw += 1;
    if (tank1_sw >= 300){
        tank1_sw = 0;
    }
    usleep(70000);
    if (collision_wall(tank1loc_x+1, tank1loc_y-1) == 0){
        tank1loc_y -= 1;
        tank1loc_x += 1;
```

```
            }
            color.tank1loc_y = tank1loc_y;
            color.tank1loc_x = tank1loc_x;
        } else if (tank1_dirc == 0b00000110){
            tank1_w += 1;
            if (tank1_w >= 300){
                tank1_w = 0;
            }
            usleep(70000);
            if (collision_wall(tank1loc_x+1, tank1loc_y) == 0){
                tank1loc_x += 1;
            }
            color.tank1loc_x = tank1loc_x;
        } else if (tank1_dirc == 0b00000111){
            tank1_nw += 1;
            if (tank1_nw >= 300){
                tank1_nw = 0;
            }
            usleep(70000);
            if (collision_wall(tank1loc_x+1, tank1loc_y+1) == 0){
                tank1loc_y += 1;
                tank1loc_x += 1;
            }
            color.tank1loc_y = tank1loc_y;
            color.tank1loc_x = tank1loc_x;
        }
    }
    if (joystick1.v_dirc == 0){ // drive
        if (tank1_dirc == 0b00000000){
            tank1_n += 1;
            if (tank1_n >= 300){
                tank1_n = 0;
            }
            usleep(70000);
            if (collision_wall(tank1loc_x, tank1loc_y-1) == 0){
                tank1loc_y -= 1;
            }
            color.tank1loc_y = tank1loc_y;
        } else if (tank1_dirc == 0b00000001){
            tank1_ne += 1;
            if (tank1_ne >= 300){
                tank1_ne = 0;
            }
            usleep(70000);
            if (collision_wall(tank1loc_x+1, tank1loc_y-1) == 0){
                tank1loc_y -= 1;
                tank1loc_x += 1;
            }
            color.tank1loc_y = tank1loc_y;
            color.tank1loc_x = tank1loc_x;
        } else if (tank1_dirc == 0b00000010){
            tank1_e += 1;
            if (tank1_e >= 300){
```

```
        tank1_e = 0;
    }
    usleep(70000);
    if (collision_wall(tank1loc_x+1, tank1loc_y) == 0){
        tank1loc_x += 1;
    }
    color.tank1loc_x = tank1loc_x;
} else if (tank1_dirc == 0b00000011){
    tank1_se += 1;
    if (tank1_se >= 300){
        tank1_se = 0;
    }
    usleep(70000);
    if (collision_wall(tank1loc_x+1, tank1loc_y+1) == 0){
        tank1loc_y += 1;
        tank1loc_x += 1;
    }
    color.tank1loc_y = tank1loc_y;
    color.tank1loc_x = tank1loc_x;
} else if (tank1_dirc == 0b00000100){
    tank1_s += 1;
    if (tank1_s >= 300){
        tank1_s = 0;
    }
    usleep(70000);
    if (collision_wall(tank1loc_x, tank1loc_y+1) == 0){
        tank1loc_y += 1;
    }
    color.tank1loc_y = tank1loc_y;
} else if (tank1_dirc == 0b00000101){
    tank1_sw += 1;
    if (tank1_sw >= 300){
        tank1_sw = 0;
    }
    usleep(70000);
    if (collision_wall(tank1loc_x-1, tank1loc_y+1) == 0){
        tank1loc_y += 1;
        tank1loc_x -= 1;
    }
    color.tank1loc_y = tank1loc_y;
    color.tank1loc_x = tank1loc_x;
} else if (tank1_dirc == 0b00000110){
    tank1_w += 1;
    if (tank1_w >= 300){
        tank1_w = 0;
    }
    usleep(70000);
    if (collision_wall(tank1loc_x-1, tank1loc_y) == 0){
        tank1loc_x -= 1;
    }
    color.tank1loc_x = tank1loc_x;
} else if (tank1_dirc == 0b00000111){
    tank1_nw += 1;
```

```
            if (tank1_nw >= 300){
                tank1_nw = 0;
            }
            usleep(70000);
            if (collision_wall(tank1loc_x-1, tank1loc_y-1) == 0){
                tank1loc_y -= 1;
                tank1loc_x -= 1;
            }
            color.tank1loc_y = tank1loc_y;
            color.tank1loc_x = tank1loc_x;
        }
        set_background_color(&color);
    } else if (button1 == 47 || button1 == 63 || button1 == 111 ||
        button1 == 127 || button1 == 175 || button1 == 191 || button1 ==
        239 || button1 == 255) {

        tank1_fire += 1;
        if (tank1_fire >= 600) {
            tank1_fire = 0;
        }

        if (bullet1_en == 0){
            bullet1_en = 1;
            pthread_create(&pthread1, NULL, &fire_bullet1, (void *)
                &entry1);
        }
    }
    set_background_color(&color);
    uint8_t button2 = joystick2.xyab;
    if (joystick2.h_dirc == 255){
        tank2_rotright += 1;
        if (tank2_rotright >= 600){
            tank2_rotright = 0;
        }
        if (tank2_dirc == 0b00000000){
            tank2_dirc = 0b00000001;
        } else if (tank2_dirc == 0b00000001){
            tank2_dirc = 0b00000010;
        } else if (tank2_dirc == 0b00000010){
            tank2_dirc = 0b00000011;
        } else if (tank2_dirc == 0b00000011){
            tank2_dirc = 0b00000100;
        } else if (tank2_dirc == 0b00000100){
            tank2_dirc = 0b00000101;
        } else if (tank2_dirc == 0b00000101){
            tank2_dirc = 0b00000110;
        } else if (tank2_dirc == 0b00000110){
            tank2_dirc = 0b00000111;
        } else if (tank2_dirc == 0b00000111){
            tank2_dirc = 0b00000000;
        }
        color.tank2_dirc = tank2_dirc;
        set_background_color(&color);
        usleep(200000);
    }
    if (joystick2.h_dirc == 0){
        tank2_rotleft += 1;
        if (tank2_rotleft >= 600){
            tank2_rotleft = 0;
        }
        if (tank2_dirc == 0b00000000){
            tank2_dirc = 0b00000111;
        } else if (tank2_dirc == 0b00000111){
            tank2_dirc = 0b00000110;
        } else if (tank2_dirc == 0b00000110){
            tank2_dirc = 0b00000101;
        } else if (tank2_dirc == 0b00000101){
            tank2_dirc = 0b00000100;
        } else if (tank2_dirc == 0b00000100){
            tank2_dirc = 0b00000011;
        } else if (tank2_dirc == 0b00000011){
            tank2_dirc = 0b00000010;
        } else if (tank2_dirc == 0b00000010){
            tank2_dirc = 0b00000001;
        } else if (tank2_dirc == 0b00000001){
            tank2_dirc = 0b00000000;
        }
        color.tank2_dirc = tank2_dirc;
        set_background_color(&color);
        usleep(200000);
    }
    if (joystick2.v_dirc == 255){ // reversing
        if (tank2_dirc == 0b00000000){
            tank2_n += 1;
            if (tank2_n >= 300){
                tank2_n = 0;
            }
            usleep(70000);
            if (collision_wall(tank2loc_x, tank2loc_y+1) == 0){
                tank2loc_y += 1;
            }
            color.tank2loc_y = tank2loc_y;
        } else if (tank2_dirc == 0b00000001){
            tank2_ne += 1;
            if (tank2_ne >= 300){
                tank2_ne = 0;
            }
            usleep(70000);
            if (collision_wall(tank2loc_x-1, tank2loc_y+1) == 0){
                tank2loc_y += 1;
                tank2loc_x -= 1;
            }
            color.tank2loc_y = tank2loc_y;
            color.tank2loc_x = tank2loc_x;
        } else if (tank2_dirc == 0b00000010){
            tank2_e += 1;
            if (tank2_e >= 300){
                tank2_e = 0;
            }
            usleep(70000);
            if (collision_wall(tank2loc_x-1, tank2loc_y) == 0){
                tank2loc_x -= 1;
            }
            color.tank2loc_x = tank2loc_x;
        } else if (tank2_dirc == 0b00000011){
            tank2_se += 1;
            if (tank2_se >= 300){
                tank2_se = 0;
            }
            usleep(70000);
            if (collision_wall(tank2loc_x-1, tank2loc_y-1) == 0){
                tank2loc_y -= 1;
                tank2loc_x -= 1;
            }
            color.tank2loc_y = tank2loc_y;
            color.tank2loc_x = tank2loc_x;
        } else if (tank2_dirc == 0b00000100){
            tank2_s += 1;
            if (tank2_s >= 300){
                tank2_s = 0;
            }
            usleep(70000);
            if (collision_wall(tank2loc_x, tank2loc_y-1) == 0){
                tank2loc_y -= 1;
            }
            color.tank2loc_y = tank2loc_y;
        } else if (tank2_dirc == 0b00000101){
            tank2_sw += 1;
            if (tank2_sw >= 300){
                tank2_sw = 0;
            }
            usleep(70000);
            if (collision_wall(tank2loc_x+1, tank2loc_y-1) == 0){
                tank2loc_y -= 1;
                tank2loc_x += 1;
            }
            color.tank2loc_y = tank2loc_y;
            color.tank2loc_x = tank2loc_x;
        } else if (tank2_dirc == 0b00000110){
            tank2_w += 1;
            if (tank2_w >= 300){
                tank2_w = 0;
            }
            usleep(70000);
            if (collision_wall(tank2loc_x+1, tank2loc_y) == 0){
                tank2loc_x += 1;
            }
            color.tank2loc_x = tank2loc_x;
        } else if (tank2_dirc == 0b00000111){
            tank2_nw += 1;
            if (tank2_nw >= 300){
                tank2_nw = 0;
            }
            usleep(70000);
            if (collision_wall(tank2loc_x+1, tank2loc_y+1) == 0){
                tank2loc_y += 1;
                tank2loc_x += 1;
            }
            color.tank2loc_y = tank2loc_y;
            color.tank2loc_x = tank2loc_x;
        }
        set_background_color(&color);
    }
    if (joystick2.v_dirc == 0){ // drive
        if (tank2_dirc == 0b00000000){
            tank2_n += 1;
            if (tank2_n >= 300){
                tank2_n = 0;
            }
            usleep(70000);
            if (collision_wall(tank2loc_x, tank2loc_y-1) == 0){
                tank2loc_y -= 1;
            }
            color.tank2loc_y = tank2loc_y;
        } else if (tank2_dirc == 0b00000001){
            tank2_ne += 1;
            if (tank2_ne >= 300){
                tank2_ne = 0;
            }
            usleep(70000);
            if (collision_wall(tank2loc_x+1, tank2loc_y-1) == 0){
                tank2loc_y -= 1;
                tank2loc_x += 1;
            }
            color.tank2loc_y = tank2loc_y;
            color.tank2loc_x = tank2loc_x;
        } else if (tank2_dirc == 0b00000010){
            tank2_e += 1;
            if (tank2_e >= 300){
                tank2_e = 0;
            }
            usleep(70000);
            if (collision_wall(tank2loc_x+1, tank2loc_y) == 0){
                tank2loc_x += 1;
            }
            color.tank2loc_x = tank2loc_x;
        } else if (tank2_dirc == 0b00000011){
            tank2_se += 1;
            if (tank2_se >= 300){
                tank2_se = 0;
            }
            usleep(70000);
            if (collision_wall(tank2loc_x+1, tank2loc_y+1) == 0){
                tank2loc_y += 1;
                tank2loc_x += 1;
            }
            color.tank2loc_y = tank2loc_y;
            color.tank2loc_x = tank2loc_x;
        } else if (tank2_dirc == 0b00000100){
            tank2_s += 1;
            if (tank2_s >= 300){
                tank2_s = 0;
            }
            usleep(70000);
            if (collision_wall(tank2loc_x, tank2loc_y+1) == 0){
                tank2loc_y += 1;
            }
            color.tank2loc_y = tank2loc_y;
        } else if (tank2_dirc == 0b00000101){
            tank2_sw += 1;
            if (tank2_sw >= 300){
                tank2_sw = 0;
            }
            usleep(70000);
            if (collision_wall(tank2loc_x-1, tank2loc_y+1) == 0){
                tank2loc_y += 1;
                tank2loc_x -= 1;
            }
            color.tank2loc_y = tank2loc_y;
            color.tank2loc_x = tank2loc_x;
        } else if (tank2_dirc == 0b00000110){
            tank2_w += 1;
            if (tank2_w >= 300){
                tank2_w = 0;
            }
            usleep(70000);
            if (collision_wall(tank2loc_x-1, tank2loc_y) == 0){
                tank2loc_x -= 1;
            }
            color.tank2loc_x = tank2loc_x;
        } else if (tank2_dirc == 0b00000111){
            tank2_nw += 1;
            if (tank2_nw >= 300){
                tank2_nw = 0;
            }
            usleep(70000);
            if (collision_wall(tank2loc_x-1, tank2loc_y-1) == 0){
                tank2loc_y -= 1;
                tank2loc_x -= 1;
            }
            color.tank2loc_y = tank2loc_y;
            color.tank2loc_x = tank2loc_x;
        }
        set_background_color(&color);
    } else if (button2 == 47 || button2 == 63 || button2 == 111 ||
                    button2 == 127 || button2 == 175 || button2 == 191 || button2 ==
                    239 || button2 == 255) {

                    tank2_fire += 1;
                    if (tank2_fire >= 600) {
                        tank2_fire = 0;
                    }

                    if (bullet2_en == 0){
                        bullet2_en = 1;
                        pthread_create(&pthread2, NULL, &fire_bullet2, (void *)
                            &entry2);
                    }
                }
            }
        }
    }
    return 0;
}
```

2. joystick.h

```c
#ifndef _JOYSTICK_H
#define _JOYSTICK_H

#include <stdio.h>
#include <stdlib.h>
#include <libusb-1.0/libusb.h>

struct joystick_admin {
    struct libusb_device_handle *joystick1;
    struct libusb_device_handle *joystick2;
    uint8_t joystick1_addr;
    uint8_t joystick2_addr;
};

struct joystick_packet {

    uint8_t c1;
    uint8_t c2;
    uint8_t c3;
    uint8_t h_dirc;
    uint8_t v_dirc;
    uint8_t xyab;
    uint8_t rl;
};

struct args_list {
    struct joystick_admin joysticks;
    char *buttons;
    int mode;
    int print;
};

extern struct joystick_admin open_controllers();

#endif
```

3. joystick.c

```
#include "joystick.h"
#include <libusb-1.0/libusb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct joystick_admin open_controllers() {

       printf("Searching for USB connections...\n");

       uint8_t endpoint_address = 0;
       struct joystick_admin joysticks;
       libusb_device **devs;
       struct libusb_device_descriptor desc;
       struct libusb_device_handle *controller = NULL;
       ssize_t num_devs;


       // Boot libusb library
       if (libusb_init(NULL) != 0) {
              printf("\nERROR: libusb failed to boot");
              exit(1);
       }

       if  ((num_devs = libusb_get_device_list(NULL, &devs)) < 0) {
              printf("\nERROR: no controllers found");
              exit(1);
       }

       //printf("Detected %d devices...\n", num_devs);
       int connection_count = 0;
       for (int i = 0; i < num_devs; i++) {

              libusb_device *dev = devs[i];

              if (libusb_get_device_descriptor(dev, &desc) < 0) {
                     printf("\nERROR: bad device descriptor.");
                     exit(1);
              }

              // Our controllers have idProduct of 17
              if (desc.idProduct == 17) {

                     //printf("FOUND: idProduct-%d ", desc.idProduct);
                     struct libusb_config_descriptor *config;
                     if ((libusb_get_config_descriptor(dev, 0, &config)) < 0) {
                            printf("\nERROR: bad config descriptor.");
                            exit(1);
                     }
                     //printf("interfaces-%d\n", config->bNumInterfaces);

                     // Our controllers only have a single interface, no need for
```

```
                            looping
                        // This interface also only has one .num_altsetting, no need
                            for looping

                        int r;
                        const struct libusb_interface_descriptor *inter =
                            config->interface[0].altsetting;
                        if ((r = libusb_open(dev, &controller)) != 0) {
                                printf("\nERROR: couldn't open controller");
                                exit(1);
                        }
                        if (libusb_kernel_driver_active(controller, 0)) {
                                libusb_detach_kernel_driver(controller, 0);
                        }
                        libusb_set_auto_detach_kernel_driver(controller, 0);
                        if ((r = libusb_claim_interface(controller, 0)) != 0) {
                                printf("\nERROR: couldn't claim controller.");
                                exit(1);
                        }

                        endpoint_address = inter->endpoint[0].bEndpointAddress;
                        connection_count++;

                        if (connection_count == 1) {
                                joysticks.joystick1 = controller;
                                joysticks.joystick1_addr = endpoint_address;
                        } else {
                                joysticks.joystick2 = controller;
                                joysticks.joystick2_addr = endpoint_address;
                                //printf("%d:%d,%d:%d\n",devices.device1,devices.device1_addr,devi
                                                     goto found;
                        }
                }
        }

        if (connection_count < 2) {
                printf("ERROR: couldn't find 2 controllers.");
                exit(1);
        }

        found:
                printf("Connected %d controllers!\n", connection_count);
                libusb_free_device_list(devs, 1);

        return joysticks;
}

void detect_presses(struct joystick_packet pkt, char *buttons, int mode) {

        // Choose whether you want human-readable or binary output
        char vals[] = "LRUDA";
        if (mode == 1) {
                strcpy(buttons, "00000");
                strcpy(vals, "11111");
        } else {
                strcpy(buttons, "_____");
        }

        // Check left/right arrows (can only be one at a time)
        if (pkt.h_dirc == 0) {
                buttons[0] = vals[0];
        } else if (pkt.h_dirc == 255) {
                buttons[1] = vals[1];
        }

        // Check up/down arrows (can only be one at a time)
        if (pkt.v_dirc == 0) {
                buttons[2] = vals[2];
        } else if (pkt.v_dirc == 255) {
                buttons[3] = vals[3];
        }

        // Check if shoot button (A) is pressed
        uint8_t a = pkt.xyab;
        if (a == 47 || a == 63 || a == 111 || a == 127 || a == 175 || a == 191 || a
            == 239 || a == 255) {
                buttons[4] = vals[4];
        }

        //printf("\n%s", buttons);

}

void *listen_controllers(void *arg) {

        struct args_list *args_p = arg;
        struct args_list args = *args_p;
        struct joystick_admin joysticks = args.joysticks;

        struct joystick_packet pkt1, pkt2;
        int fields1, fields2;
        int size1 = sizeof(pkt1);
        int size2 = sizeof(pkt2);
        char buttons1[] = "_____";
        char buttons2[] = "_____";

        for (;;) {

                libusb_interrupt_transfer(joysticks.joystick1,
                    joysticks.joystick1_addr, (unsigned char *) &pkt1, size1,
                    &fields1, 0);
                libusb_interrupt_transfer(joysticks.joystick2,
                    joysticks.joystick2_addr, (unsigned char *) &pkt2, size2,
                    &fields2, 0);

                // 7 fields should be transferred for each packet
                if (fields1 == 7 && fields2 == 7) {
                        detect_presses(pkt1, buttons1, args.mode);
                        detect_presses(pkt2, buttons2, args.mode);
                        strcat(buttons1, buttons2);
                        strcpy(args.buttons, buttons1);
                        if (args.print == 1) {
                                printf("%s\n", args.buttons);
                        }
                }
        }
}
```

4. vga_ball.h

```
#ifndef _VGA_BALL_H
#define _VGA_BALL_H

#include <linux/ioctl.h>

typedef struct {
   unsigned char game_info_1,game_info_2,game_info_3, tank1loc_x,tank1loc_y,
       tank2loc_x,
               tank2loc_y,tank1_dirc,tank2_dirc,bullet1loc_x,bullet1loc_y,bullet2loc_x,
               bullet2loc_y,bullet1_dirc, bullet2_dirc, explo_loc_x,explo_loc_y;
} vga_ball_color_t;


typedef struct {
  vga_ball_color_t background;
} vga_ball_arg_t;

#define VGA_BALL_MAGIC 'q'

/* ioctls and their arguments */
#define VGA_BALL_WRITE_BACKGROUND _IOW(VGA_BALL_MAGIC, 1, vga_ball_arg_t *)
#define VGA_BALL_READ_BACKGROUND _IOR(VGA_BALL_MAGIC, 2, vga_ball_arg_t *)

#endif
```

5. vga_ball.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_ball.ko
 *
 * Check code style with
 * checkpatch.pl --file --no-tree vga_ball.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_ball.h"

#define DRIVER_NAME "vga_ball"

/* Device registers */
#define GAME_INFO_1(x)   (x)  // stage num->2, explosion num->2, map num->3
#define GAME_INFO_2(x)   ((x)+1) // player1score->4, player2score->4
#define GAME_INFO_3(x)       ((x)+2)
#define TANK1LOC_X(x)   ((x)+3)
#define TANK1LOC_Y(x)   ((x)+4)
#define TANK2LOC_X(x)   ((x)+5)
#define TANK2LOC_Y(x)   ((x)+6)
#define TANK1_DIRC(x)   ((x)+7) // first 2 tank1 dirc, next 2 tank2 dirc
#define TANK2_DIRC(x)       ((x)+8)

#define BULLET1LOC_X(x)  ((x)+9)
#define BULLET1LOC_Y(x)  ((x)+10)
#define BULLET2LOC_X(x)  ((x)+11)
#define BULLET2LOC_Y(x)  ((x)+12)
```

```
#define BULLET1_DIRC(x)      ((x)+13)
#define BULLET2_DIRC(x)      ((x)+14)
#define EXPLO_LOC_X(x)   ((x)+15)
#define EXPLO_LOC_Y(x)   ((x)+16) // 7 bits only

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

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
static void write_background(vga_ball_color_t *background)
{
   iowrite8(background->game_info_1,  GAME_INFO_1(dev.virtbase) );
   iowrite8(background->game_info_2,  GAME_INFO_2(dev.virtbase) );
   iowrite8(background->game_info_3,   GAME_INFO_3(dev.virtbase) );
   iowrite8(background->tank1loc_x,  TANK1LOC_X(dev.virtbase) );
   iowrite8(background->tank1loc_y,  TANK1LOC_Y(dev.virtbase) );
   iowrite8(background->tank2loc_x,  TANK2LOC_X(dev.virtbase) );
   iowrite8(background->tank2loc_y,  TANK2LOC_Y(dev.virtbase) );
   iowrite8(background->tank1_dirc,  TANK1_DIRC(dev.virtbase) );
   iowrite8(background->tank2_dirc,    TANK2_DIRC(dev.virtbase) );
   iowrite8(background->bullet1loc_x,  BULLET1LOC_X(dev.virtbase) );
   iowrite8(background->bullet1loc_y,  BULLET1LOC_Y(dev.virtbase) );
   iowrite8(background->bullet2loc_x,  BULLET2LOC_X(dev.virtbase) );
   iowrite8(background->bullet2loc_y,  BULLET2LOC_Y(dev.virtbase) );
   iowrite8(background->bullet1_dirc,  BULLET1_DIRC(dev.virtbase) );
   iowrite8(background->bullet2_dirc,  BULLET2_DIRC(dev.virtbase) );
   iowrite8(background->explo_loc_x,  EXPLO_LOC_X(dev.virtbase) );
   iowrite8(background->explo_loc_y,  EXPLO_LOC_Y(dev.virtbase) );
   dev.background = *background;
}

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

   switch (cmd) {
   case VGA_BALL_WRITE_BACKGROUND:
      if (copy_from_user(&vla, (vga_ball_arg_t *) arg,
               sizeof(vga_ball_arg_t)))
```

```
            return -EACCES;
         write_background(&vla.background);
         break;

      case VGA_BALL_READ_BACKGROUND:
         vla.background = dev.background;
         if (copy_to_user((vga_ball_arg_t *) arg, &vla,
                sizeof(vga_ball_arg_t)))
            return -EACCES;
         break;

      default:
         return -EINVAL;
      }

      return 0;
}

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

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

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init vga_ball_probe(struct platform_device *pdev)
{
       vga_ball_color_t beige = { 0xf9, 0xe4, 0xb7, 0x00, 0x00 }; //this place is
           different
   int ret;

   /* Register ourselves as a misc device: creates /dev/vga_ball */
   ret = misc_register(&vga_ball_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;
   }

   /* Set an initial color */
        write_background(&beige);

   return 0;

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

/* Clean-up code: release resources */
static int vga_ball_remove(struct platform_device *pdev)
{
   iounmap(dev.virtbase);
   release_mem_region(dev.res.start, resource_size(&dev.res));
   misc_deregister(&vga_ball_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_ball_of_match[] = {
   { .compatible = "csee4840,vga_ball-1.0" },
   {},
};
MODULE_DEVICE_TABLE(of, vga_ball_of_match);
#endif

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

/* Called when the module is loaded: set things up */
```

```
static int __init vga_ball_init(void)
{
   pr_info(DRIVER_NAME ": init\n");
   return platform_driver_probe(&vga_ball_driver, vga_ball_probe);
}

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

module_init(vga_ball_init);
module_exit(vga_ball_exit);

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

6. vga_ball.sv

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

module vga_ball(input logic     clk,
          input logic    reset,
      input logic [7:0] writedata,
      input logic      write,
      input         chipselect,
      input logic [4:0] address,

      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;

   logic [7:0]      map, score, map_end_en, tank1_x, tank1_y, tank2_x, tank2_y,
      tank1_dir, tank2_dir, bullet1_x, bullet1_y, bullet2_x, bullet2_y,
      bullet1_dir, bullet2_dir, explod_x, explod_y;

   logic [999:0] map1 =
      1000'b1111111111111111111111111111111111111111000000000000000000000000000000000000000110
   logic [999:0] map2 =
      1000'b1111111111111111111111111111111111111111000000000000000000000000000000000000000110
   logic [999:0] map3 =
      1000'b0000000000000000000000000000000000000000000000000000000000000000000000000000000011
    logic [383:0] logo =
       384'b0000000000000000000000000000000000000000000000000001100011100011111001000101000010
    logic [479:0] game_over =
       480'b0000000000000000000000000000000000000000000000000000000000000011001101111100001000
    logic [127:0] hp =
       128'b0000000000000000001100001101100110110000110110011000011110111111000010011011111101


    logic[2:0] counter = 0;
    logic[2:0] counter_y = 0;
    logic[10:0] real_x = 0;
    logic[10:0] real_y = 0;
    logic[10:0] real_tank_x1;
    logic[10:0] real_tank_x2;
    logic[10:0] real_tank_y1;
    logic[10:0] real_tank_y2;
    logic[10:0] real_bullet_x1;
    logic[10:0] real_bullet_x2;
    logic[10:0] real_bullet_y1;
    logic[10:0] real_bullet_y2;
```

```
logic[10:0] res;
 logic[10:0] res2;
logic[10:0] minus_tmp1;
logic[10:0] minus_tmp2;
logic[10:0] minus_tmp3;
logic[10:0] minus_tmp4;
logic[10:0] cur;
logic[10:0] cur_logo;
logic[10:0] cur_game_over;
logic[10:0] addr;
logic[10:0] cur_hp_l;
logic[10:0] cur_hp_r;
logic[23:0] q;
logic[20:0] tmp1;
logic[20:0] tmp2;
logic[20:0] tmp3;
logic[20:0] tmp4;
logic[30:0] result1;
logic[30:0] result2;

pic_rom pic_rom_inst(
  .address(addr),
  .clock(clk),
  .q(q)
);

parameter RED  = 24'b00000000_01001000_01110111;
parameter BLUE = 24'b01111100_00000000_00000000;
parameter WHITE = 24'b11111111_11111111_11111111;
parameter  MAP_COLOR = 24'b10000100_10100100_11000100;
parameter  DARK_BROWN = 24'b00110011_01000000_01011100;
parameter GREEN = 24'b00000000_11111111_00000000;
parameter BLACK = 24'b00000000_00000000_00000000;

vga_counters counters(.clk50(clk), .*);

assign res2 = 8'd16 * (16 - score[3:0]);
 assign res = 8'd16 * (16 - score[7:4]);
assign real_tank_x1 = tank1_x * 4;
assign real_tank_x2 = tank2_x * 4;
assign real_tank_y1 = tank1_y * 4;
assign real_tank_y2 = tank2_y * 4;
assign real_bullet_x1 = bullet1_x * 4;
assign real_bullet_x2 = bullet2_x * 4;
assign real_bullet_y1 = bullet1_y * 4;
assign real_bullet_y2 = bullet2_y * 4;
assign cur_logo = (vcount >> 2) * 8'd48 + ((hcount[10:1] - 8'd208) >> 2);
assign cur_game_over = ((vcount - 8'd224) >> 2) * 8'd60 + ((hcount[10:1] -
    8'd200) >> 2);
assign cur_hp_l = (vcount >> 2) * 8'd16 + ((hcount[10:1] - 8'd64) >> 2);
assign cur_hp_r = (vcount >> 2) * 8'd16 + ((hcount[10:1] - 8'd512) >> 2);
assign cur = ((vcount - 8'd80) >> 4) * 8'd40 + (hcount[10:1] >> 4);
assign minus_tmp1 = (hcount[10:1] > real_bullet_x1) ? hcount[10:1] -
```

```
    real_bullet_x1 : real_bullet_x1 - hcount[10:1];
 assign minus_tmp2 = (vcount > real_bullet_y1) ? vcount - real_bullet_y1 :
    real_bullet_y1 - vcount;
 assign minus_tmp3 = (hcount[10:1] > real_bullet_x2) ? hcount[10:1] -
    real_bullet_x2 : real_bullet_x2 - hcount[10:1];
 assign minus_tmp4 = (vcount > real_bullet_y2) ? vcount - real_bullet_y2 :
    real_bullet_y2 - vcount;
 assign tmp1 = minus_tmp1 * minus_tmp1;
 assign tmp2 = minus_tmp2 * minus_tmp2;
 assign tmp3 = minus_tmp3 * minus_tmp3;
 assign tmp4 = minus_tmp4 * minus_tmp4;
 assign result1 = tmp1 + tmp2;
 assign result2 = tmp3 + tmp4;

always_ff @(posedge clk)
  if (reset) begin
   map <= 8'h0;
   score <= 8'h0;
   map_end_en <= 8'h0;
   tank1_x <= 8'd40;
   tank1_y <= 8'd70;
   tank2_x <= 8'd80;
   tank2_y <= 8'd50;
   tank1_dir <= 8'd0;
   tank2_dir <= 8'd2;
   bullet1_x <= 8'd12;
   bullet1_y <= 8'd32;
   bullet2_x <= 8'd69;
   bullet2_y <= 8'd50;
   bullet1_dir <= 8'd0;
   bullet2_dir <= 8'd0;
   explod_x <= 8'd0;
   explod_y <= 8'd0;
  end else if (chipselect && write)
    case (address)
     5'd0 : map <= writedata;
     5'd1 : score <= writedata;
     5'd2 : map_end_en <= writedata;
     5'd3 : tank1_x <= writedata;
     5'd4 : tank1_y <= writedata;
     5'd5 : tank2_x <= writedata;
     5'd6 : tank2_y <= writedata;
     5'd7 : tank1_dir <= writedata;
     5'd8 : tank2_dir <= writedata;
     5'd9 : bullet1_x <= writedata;
     5'd10 : bullet1_y <= writedata;
     5'd11 : bullet2_x <= writedata;
     5'd12 : bullet2_y <= writedata;
     5'd13 : bullet1_dir <= writedata;
     5'd14 : bullet2_dir <= writedata;
     5'd15 : explod_x <= writedata;
     5'd16 : explod_y <= writedata;
    endcase
```

```
always_ff @(posedge clk) begin
   if (reset)
     {VGA_R, VGA_G, VGA_B} <= {8'h0, 8'h0, 8'h0};
   else if (VGA_BLANK_n )begin
     if(map_end_en[0] == 1'b1) begin
       if(vcount >= 224 && vcount < 256 && hcount[10:1] >=200 && hcount[10:1] <
           440) begin
         if(game_over[cur_game_over] == 1'b1) begin
           {VGA_R, VGA_G, VGA_B} <= WHITE;
         end else
           {VGA_R, VGA_G, VGA_B} <= {8'h0, 8'h0, 8'h0};
       end else
          {VGA_R, VGA_G, VGA_B} <= {8'h0, 8'h0, 8'h0};
     end else begin
       if(vcount < 10'd80) begin
         if(vcount < 32) begin
           if(hcount[10:1] >= 208 && hcount[10:1] < 400) begin
             if(logo[cur_logo] == 1'b1)
               {VGA_B, VGA_G, VGA_R} <= WHITE;
             else
               {VGA_B, VGA_G, VGA_R} <= BLACK;
           end else if(hcount[10:1] >= 64 && hcount[10:1] < 128) begin
             if(hp[cur_hp_l] == 1'b1) begin
               {VGA_B, VGA_G, VGA_R} <= WHITE;
             end else
               {VGA_B, VGA_G, VGA_R} <= BLACK;
           end else if(hcount[10:1] >= 512 && hcount[10:1] < 576) begin
             if(hp[cur_hp_r] == 1'b1) begin
               {VGA_B, VGA_G, VGA_R} <= WHITE;
             end else
               {VGA_B, VGA_G, VGA_R} <= BLACK;
           end else
             {VGA_B, VGA_G, VGA_R} <= BLACK;
         end else if(vcount >= 32 && vcount < 48) begin
           if((hcount[10:1] >=0 && hcount[10:1] < res) || (hcount[10:1] >= 640 -
               res2)) begin
             {VGA_B, VGA_G, VGA_R} <= GREEN;
           end else begin
             {VGA_B, VGA_G, VGA_R} <= BLACK;
           end
         end else begin
           {VGA_B, VGA_G, VGA_R} <= BLACK;
         end
       end else begin
         if(hcount[10:1] >= real_tank_x1 && hcount[10:1] < real_tank_x1 + 16 &&
             vcount >= real_tank_y1 && vcount < real_tank_y1+16) begin
           case(tank1_dir)
             8'b00000000: addr <= (10'd15 - (hcount[10:1] - real_tank_x1[9:0])) *
                 16 + (vcount - real_tank_y1[9:0]); //000
             8'b00000111: addr <= 10'd256 + (10'd15 - (hcount[10:1] -
                 real_tank_x1[9:0])) * 16 + (vcount - real_tank_y1[9:0]); //111
             8'b00000010: addr <= (vcount - real_tank_y1[9:0]) * 16 + (10'd15 -
        (hcount[10:1] - real_tank_x1[9:0])); //010
      8'b00000011: addr <= 10'd256 + (vcount - real_tank_y1[9:0]) * 16 +
        (10'd15 - (hcount[10:1] - real_tank_x1[9:0])); //011 correct
      8'b00000100: addr <= (hcount[10:1] - real_tank_x1[9:0]) * 16 +
        (10'd15 - (vcount - real_tank_y1[9:0])); //100
      8'b00000001: addr <= 10'd256 + (10'd15 - (vcount -
        real_tank_y1[9:0])) * 16 + (10'd15 - (hcount[10:1] -
        real_tank_x1[9:0])); //011
      8'b00000110: addr <= (vcount - real_tank_y1[9:0]) * 16 +
        (hcount[10:1] - real_tank_x1[9:0]);         //110
      8'b00000101: addr <= 10'd256 + (vcount - real_tank_y1[9:0]) * 16 +
        (hcount[10:1] - real_tank_x1[9:0]); //101 correct
      default: addr <= (vcount - real_tank_y1[9:0]) * 16 + (hcount[10:1] -
        real_tank_x1[9:0]);
    endcase
    {VGA_B, VGA_G, VGA_R} <= q;
  end else if(hcount[10:1] >= real_tank_x2 && hcount[10:1] < real_tank_x2
      + 16 && vcount >= real_tank_y2 && vcount < real_tank_y2+16) begin
    //addr <= 10'd512 + (vcount - real_tank_y2[9:0]) * 16 + (hcount[10:1]
      - real_tank_x2[9:0]);
    case(tank2_dir)
      8'b00000000: addr <= 10'd512 + (10'd15 - (hcount[10:1] -
        real_tank_x2[9:0])) * 16 + (vcount - real_tank_y2[9:0]);
      8'b00000111: addr <= 10'd768 + (10'd15 - (hcount[10:1] -
        real_tank_x2[9:0])) * 16 + (vcount - real_tank_y2[9:0]);
      8'b00000010: addr <= 10'd512 + (vcount - real_tank_y2[9:0]) * 16 +
        (10'd15 - (hcount[10:1] - real_tank_x2[9:0]));
      8'b00000011: addr <= 10'd768 + (vcount - real_tank_y2[9:0]) * 16 +
        (10'd15 - (hcount[10:1] - real_tank_x2[9:0]));
      8'b00000100: addr <= 10'd512 + (hcount[10:1] - real_tank_x2[9:0]) *
        16 + (10'd15 - (vcount - real_tank_y2[9:0]));
      8'b00000001: addr <= 10'd768 + (10'd15 - (vcount -
        real_tank_y2[9:0])) * 16 + (10'd15 - (hcount[10:1] -
        real_tank_x2[9:0]));
      8'b00000110: addr <= 10'd512 + (vcount - real_tank_y2[9:0]) * 16 +
        (hcount[10:1] - real_tank_x2[9:0]);
      8'b00000101: addr <= 10'd768 + (vcount - real_tank_y2[9:0]) * 16 +
        (hcount[10:1] - real_tank_x2[9:0]);
      default: addr <= 10'd512 + (vcount - real_tank_y2[9:0]) * 16 +
        (hcount[10:1] - real_tank_x2[9:0]);
    endcase
    {VGA_B, VGA_G, VGA_R} <= q;
  end else if(result1 < 9'd16)begin
    {VGA_B, VGA_G, VGA_R} <= GREEN;
  end else if(result2 < 9'd16) begin
    {VGA_B, VGA_G, VGA_R} <= GREEN;
  end else begin
    case(map)
      8'd0: begin
        if(map1[cur] == 1'b1)
          {VGA_B, VGA_G, VGA_R} <= MAP_COLOR;
        else
          {VGA_B, VGA_G, VGA_R} <= BLACK;
```

```
                end
                8'd1: begin
                  if(map2[cur] == 1'b1)
                    {VGA_B, VGA_G, VGA_R} <= MAP_COLOR;
                  else
                    {VGA_B, VGA_G, VGA_R} <= BLACK;
                end
                8'd2: begin
                  if(map3[cur] == 1'b1)
                    {VGA_B, VGA_G, VGA_R} <= MAP_COLOR;
                  else
                    {VGA_B, VGA_G, VGA_R} <= BLACK;
                end
                default: begin
                  {VGA_B, VGA_G, VGA_R} <= BLACK;
                end
              endcase
            end
          end
        end
      end else
        {VGA_R, VGA_G, VGA_B} <= {8'h0, 8'h0, 8'h0};
   end

endmodule

module vga_counters(
 input logic      clk50, reset,
 output logic [10:0] hcount, // hcount[10:1] is pixel column
 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
```