ECE 540 Final Project Report

World of Tank

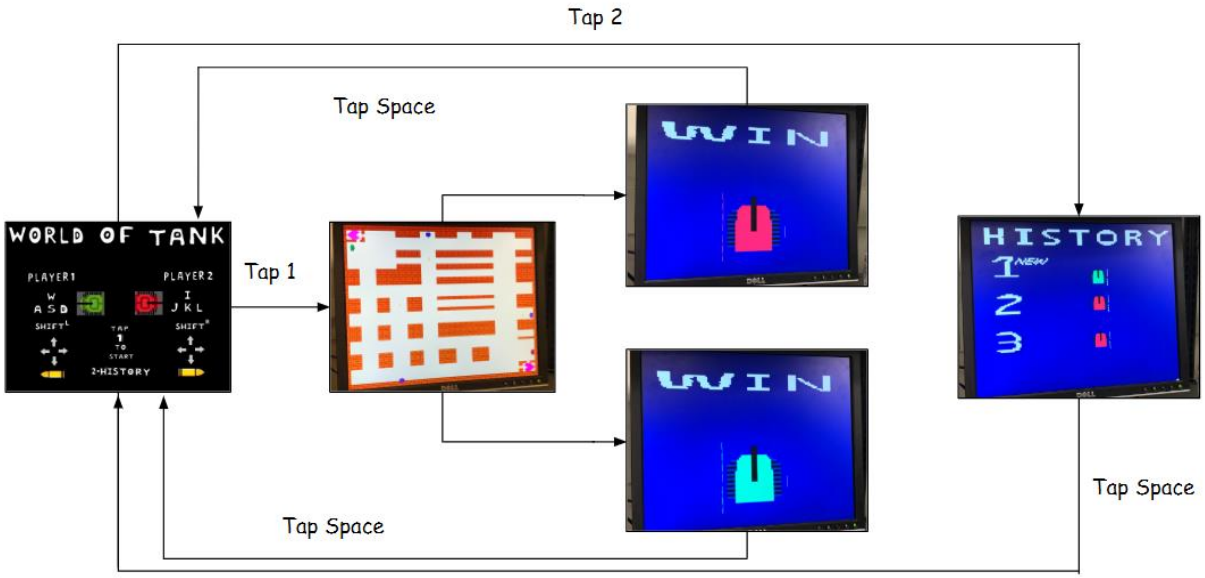
Winter 2019

Zhe Lu
Ting Wang
Xiaoqiao Mu
Ming Ma

Introduction

In this project, we want to implement a classic game: World of Tank by using Nexys A7 DDR board and MIPS FPGA. The hardware will control the display of the game and the software will control the game rules. This game is a competitive game between 2 players and the control interface is the keyboard. Each player will control a separated tank (Red Tank or Green Tank) to try to shoot each other. Also, there are three static monsters on the map. They will shoot bullets automatically. In order to win this game, you need to make sure you aren't shoot by the opponent's tank or monsters. Then, try your best to shoot the opponent's tank or destroy the opponent's base. There are two ways to win this game: destroy the opponent's tank 3 times or destroy the opponent's base 1 time. Then, we will introduce some design details of our project in the following part.

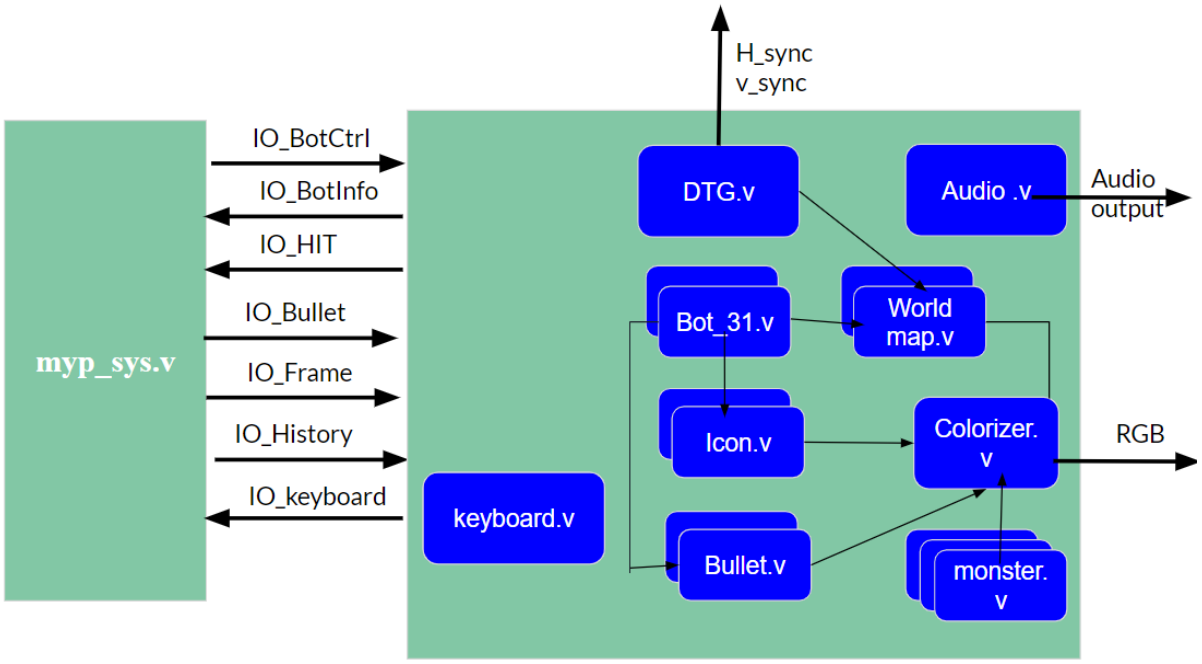
Game Procedures



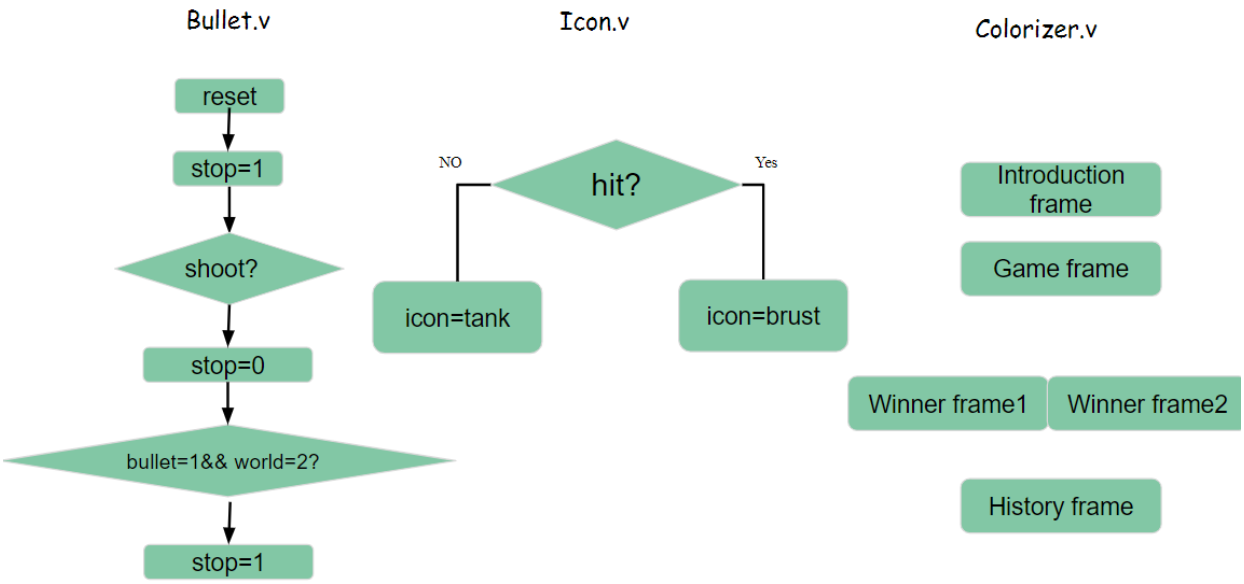
We have five frames. In the initial screen, the players can tap 1 to start the game or tap 2 to check the game history. When one of the player tap 1 to start playing the tank game, players can't go back to the Initial screen. Once one of the tank wins, the WIN screen will show up automatically. In the WIN screen, the player can tap space to go back to the Initial screen. In the history screen, the player can also tap space to go back to the Initial screen.

Hardware

Flowchart



This is the basic flowchart of our hardware part. The first thing we do is that we change the initial position, initial orientation and speed of the Rojobot (Red Tank and Green Tank). The Icon module and Bullet module is based on project 2. And the world map is generated by the tool that is given by Professor Roy. Then, we put all modules together as shown in the above picture. Then, we will talk about some details about our hardware part.



In Bullet module, we design a path that the bullet will move automatically along with path. After reset, we set a stop flag as 1 which means no bullet shows up. After getting the shoot signal, we set the stop flag back to 0. When the stop signal is 0, the starting position of bullet will increase after some delays. So, the bullet will move along with a straight path. The bullet will stop also when it hits the wall, player's tank or player's base. For example, the bullet will stop if the bullet flag (which is a signal to determine when and where to show the bullet) is 1 and world map pixel is 01(red base), 10(wall) or 11(green base) at the same time. Meanwhile, the stop signal will set back to 1. The following picture is an example to show how the stop signal is set.

```
// Get stop and burst signal
always @(posedge clock) begin
if (!reset)begin
 stop <= 1'b1;                          //At beginning, the stop flag is 1
 burst <= 1'b0;                         //At beginning, the hit signal is 0
end
else if (biu==1'b1) begin               //When the tank shoot
 stop <= 1'b0;                          //When the stop flag is 0
end
else if ((bullet == 2'b1) && (world_pixel == 2'b10)) begin  //When the stop keeps being 0 until the bullet meet the wall
 stop <= 1'b1;
end
else if ((bullet == 2'b1) && (world_pixel == 2'b11))begin   //When the stop keeps being 0 until the bullet meet the red base
 stop <= 1'b1;
end
else if ((bullet == 2'b1) && (world_pixel == 2'b01))begin   //When the stop keeps being 0 until the bullet meet the green base
 stop <= 1'b1;
end
else if ((bullet == 2'b1) && (icon_op == 2'b01))begin       //When the stop keeps being 0 until the bullet meet the opponemt tank
 stop <= 1'b1;
 burst <= 1'b1;                                             //When bullet meets the oppoment tank, set the burst signal to show the opponemt is hit
end
else begin
 stop <= stop;
 burst <= 1'b0;
end
end
```

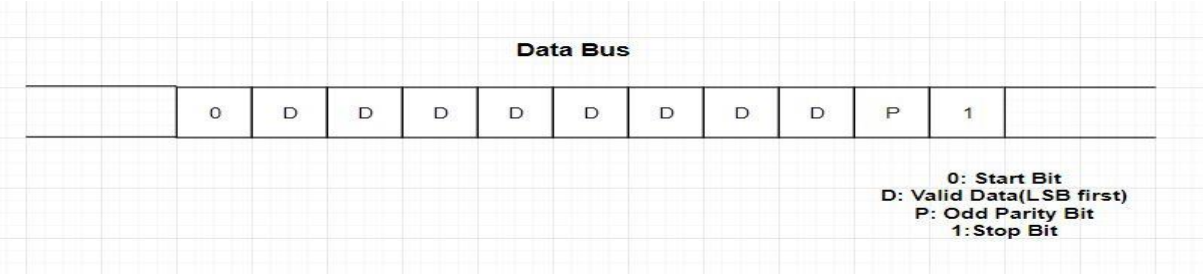
We use similar ways to determine if the one of tank is hit. When the bullet flag is 1 and the flag of tank is 1 at the same time, it means tanks are hit by the bullet. We will set a signal to show the status that the tank is hit. In the icon module, we will the burst image if this signal sets. Otherwise, the icon is the image of tank.

The monsters are static and are putted in a fix locations. They have the same shape with player's tank, but with different color.

For the colorizer module, the data read from rom is 12 bits RGB data and it's hard to determine which image should show up. So, we add extra flags to help us to determine where to show the correct images.

We determine which frames show up based on the IO_Frame signal. We have 5 frames in our project. The first frame is the introduction frame (initial frame) and this frame is a whole picture that we read from a ROM. The second frame is game frame. There are two players' tank, three monsters, bullets, two bases and world map in this frame and there are some case statements to determine which object should display. The obstruction of the world map is set to bricks because we add an extra module to do this placement. Two winner frames are consisted of two part: words and tanks. The history frame is consisted of numbers and tanks. According to the IO_History signal which record the winners of last three games, we determine which tank displays.

Keyboard

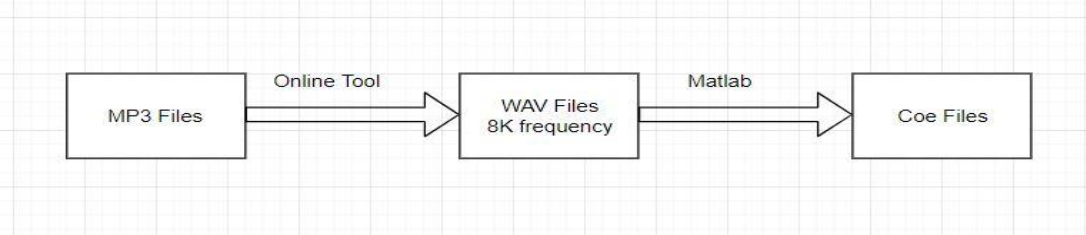


For the keyboard part, we use PS/2 type keyboard to work with Nexys A7 DDR board. This kind of keyboard is just basic keyboard. When user press keys of the keyboard, the keyboard will send scan data to the FPGA. The scan data is 11 bits data and the format of data is shown in the picture above. The valid data is only 8 bits.

```
//Store the 11-bit scan data from keyboard into register.
always@(negedge(kclkf))begin
    case(cnt)
    0:;//Start bit
    1:datacur[0]<=kdataf;
    2:datacur[1]<=kdataf;
    3:datacur[2]<=kdataf;
    4:datacur[3]<=kdataf;
    5:datacur[4]<=kdataf;
    6:datacur[5]<=kdataf;
    7:datacur[6]<=kdataf;
    8:datacur[7]<=kdataf;
    9:flag<=1'b1; //valid 8-bit scan data is ready.
    10:flag<=1'b0;
    endcase
    if(cnt<=9) cnt<=cnt+1;
    else if(cnt==10) cnt<=0;
end
```

We use the code which is shown above to capture the valid 8-bit scan data. Then, we will put these 8-bit data to different bits of keycodeout which is the output of keyboard module based on which player press keys. The keys that player 1 press is stored in keycodeout[7:0] and the keys that player 2 press is stored in keycodeout[15:8]. You can see the PS2Receiver module for more information.

Audio

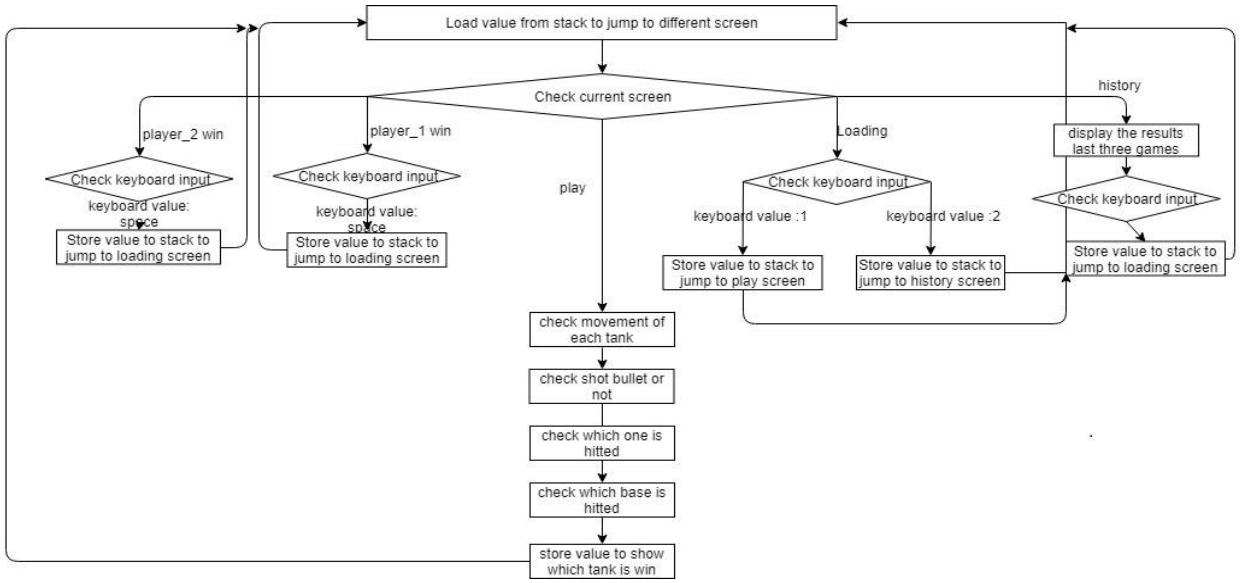


In order to store the audio files into FPGA, we need to do some conversions as shown in the above picture. The first conversion is how to convert from mp3 files to WAV files and the frequency of the WAV files is 8 KHz. When I start to do this part, I write the Matlab code to do this. However, I find this will increase some extra noise into the files. So, I find an online tool to help me to do this conversion. Then, the next part is how to convert from WAV files to coe files. For this part, I search online and I find someone's code to do this conversion. The code will load WAV files into Matlab first and then generate the sound plot. We can see what range of signal is good and put the range of data in the Matlab code to generate more efficient coe files. The width of coe files is 8 bits and the depth of coe files is 22400.

```
if(audio_counter1 >= audio_data1) //Compare the audio data of initial BGM with its audio counter
    AUD_PWM_I <= 1'b0;
else
    AUD_PWM_I <= 1'b1;
```

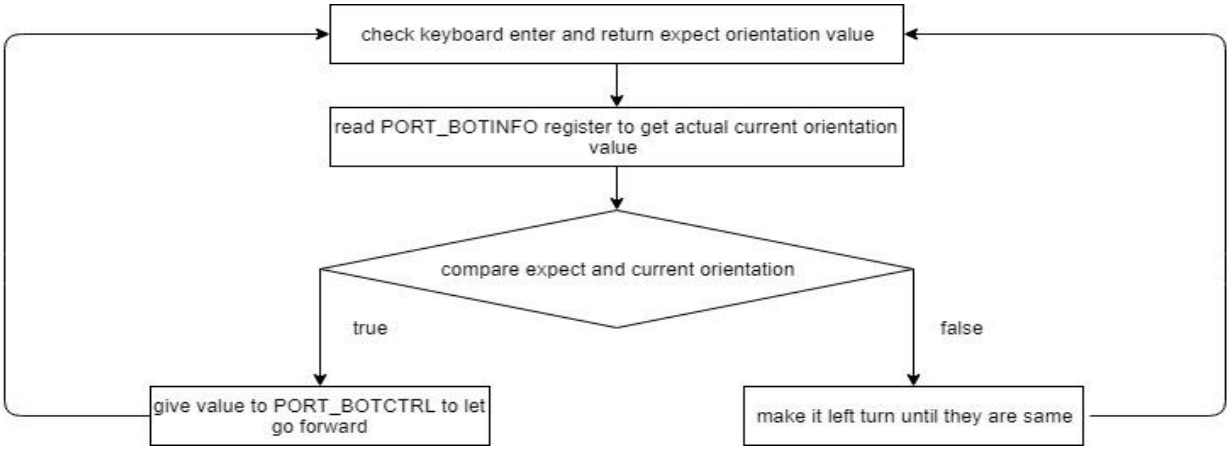
This FPGA only contains a simple amplifier. So, we only need to use two ports to control the audio part: AUD_PWM and AUD_SD. Basically, we set a PWM counter and this counter will increase by 1 on every positive edge of clock. Then, the read data from coe file will be compared with the value of this counter as shown in the picture above. The AUD_SD signal stands for shutdown and we don't need to worry about this. So, we just assign this part with a constant value: 1. For this part, we forget to play our BGM during our demo section. So, we include a short video in the project folder to display our audio part. The audios that we play have some noise and the sound is loud. So, we don't use speaker to play the audios because we don't that become too annoying. We only connect our earphones to the FPGA and we think it's enough to show the audio part. Please take a look at that if you have some questions.

Software

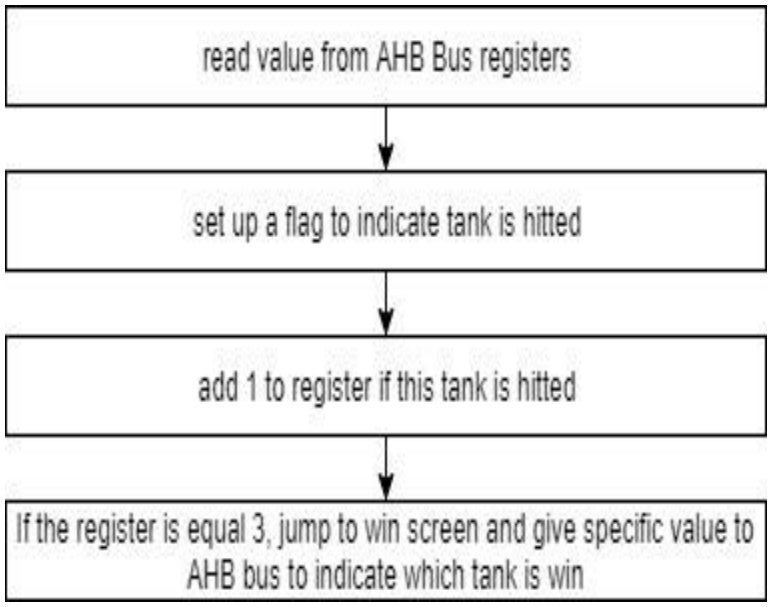


The software of this project is used to read the keyboard input to control the screen jumping and tanks movement, also it will read the data of which tank or tank's base is hit from hardware and then count how many time they has been hit.

There is a infinite loop for this program, and the first thing for this loop is loading the value from a stack which store the value for the screen it should jump to and then store this value to the register, this register is a AHB lite that will give the enable to different images which is stored in hardware. Because we always want the first screen is the loading screen, I give the value which will jump to loading screen before this loop so that it will always go to the loading screen first. Then this program will check which screen it is right now. If right now it is loading screen, it will stay this screen until key 1 or key 2 is pressed. If key 1 is pressed, the value for jumping to play screen will store to the stack and then go back to the beginning of the loop. As similar as key 2 is pressed, the value for jumping to history screen will store to stack and go back to the beginning of the loop. If the current screen is either green tank win screen, red tank win screen or history screen, it will do similar things as what it did in the loading screen. However, the more things they did rather than loading screen is they will give value to AHB bus to enable either red or green tank should display in that screen. If the current screen is play screen, it will read the input from keyboard to control the movement of the tanks, also control shooting bullet, then it will read value from register to check which tank is hit and count how many times it has been hit to decide which one is win.



The diagram below shows how to control the movement of the tank. First it will read the data from keyboard via AHB lite, after read the data, each key enter indicates different orientation that tank should move. For example, if the key w is entered, it will store value 0x00 which means orientation north, it is the orientation the tank should be. Then it will read BOTINFO register to get current orientation. By comparing these two orientations, if they are same, the tank will go forward, otherwise it will turn left until they are same. After we implement this algorithm, it should be more convenient for players that they can just tab key w for going up and s for going down.



After control the movement and shooting bullet, this program will check which tank is hit and count how many time it has been hit. First it will read the value from AHB bus register, this register has 4 bits which the first and third bits indicate either green or red tank has been hit, and second and fourth bits indicate either red or green base has been hit. If the tank hit bits are high, it will set up a flag to indicate which tank is hit and then add 1 more time to the value which store how many time the tank has been hit, if the counter is equal to 3, it will jump to the win screen and there is another register through AHB lite as well to indicate which tank image should be displayed in the screen.

Problem and Solution

Zhe Lu: At the beginning when we implement the movement algorithm, we just did similar thing as what we did in the project 2, the key w and s control the tank go forward or go backforward, and a and d controls to do left or right turn. However, it is to complicate for player to control the tank. After that, we think we need to improve this algorithm, then we implemented the algorithm that I talked above. Right now players don't need worry about turns any more, after they enter the key, the tank will turn automatically to the orientation they want and go forward.

Ting Wang: The first problem I met is the hit signal, which is to show one of tank is hit by bullet, is too short to be caught by MIPS. The solution is that we convert the burst signal into MIPS, which is used to show burst image and is hit signal with some delay. The second problem is the orientation of bullet is always change according to tanks' orientation. The solution is that we record the orientation of tanks at the shoot timing and set that as the orientation of bullet.

Xiaoqiao Mu: The problem I met is Matlab code and memory issues. For the Matlab code, I haven't found a perfect code when I searched online. I combined part code from different examples together. And I also learn something about color transform. The .coe files which converted from .jpg files always use a lot of memory. Because of the limited memory, I change some simple pictures such as tanks into world_map style (00 - white; 01 - black; 10 - green; 11 - gray). It will save a lot of memory!

Ming Ma: The biggest problem that I met in this project was the memory issue. We have already consumed 97% BRAM of the FPGA before I put the audio files into our project. Then, I find that I can store the audio file into distributed rom which will only consume LUTs of FPGA. However, each coe file will take us 50 minutes to synthesis. So, we only put two audio files in our project to save our time. Another issue is the audio contains some noise that I cannot eliminate. I can make the audio sound better by increasing the width of the coe file, but that will consume us lots of memory. So, we don't implement that and the sound is enough to listen.

Conclusion

After about three week's work, we finished our project on time. All functions that we listed in the proposal are satisfied and our game works pretty well. All of us learn lots of new skills and knowledge in this class. If we have time, we can add Wifi module to our project and add some Android things. So, we can control our tanks by Android phone also! Anyway, we all done!

Contribution

Zhe Lu: Game algorithm part (software)
Ting Wang: Icon, Rojobot, Bullet, Colorizer and Monster module
Xiaoqiao Mu: Image generation and hardware integration
Ming Ma: Audio module and keyboard interface

Reference

1. Demo Video: https://github.com/ECE540-winter2019/finalproj-zhel_mingm_xiaoqiaom_tingw/tree/master/ECE540_Video_Presentation
2. http://www.unm.edu/~tbeach/IT145/color.html
3. https://www.mathworks.com/matlabcentral/fileexchange/39805-image-to-coe-converter
4. https://reference.digilentinc.com/learn/programmable-logic/tutorials/nexys-4-ddr-keyboard-demo/start
5. https://github.com/gajjanag/6111_Project/tree/master/assets/audio_convert