Simple Car Collision-Avoidance Game Project Report

Zheng Zou, College of Engineering, University of California, Davis

***Abstract*-This article describes the full details of constructing a simple car collision avoidance game on the SSD1331 OLED screen powered by the TI CC3200 board.**

I.INTRODUCTION

This final lab incorporates all the knowledge I learned in the EEC172 class, including SPI, I2C, and AWS to produce an engaging game on the OLED screen. Firstly, I used SPI and carefully managed the spawning and moving down of new cars, represented by yellow rectangles, for users to avoid. Secondly, with the help of I2C, the user's input can be seamlessly transferred to the control of the user's own red rectangular car. To make the game more interesting, I added a stealth mode which enables the temporal invincible ability to the user's car, so that users will have more choices of avoiding collisions. Finally, the AWS feature provides users their scores sent by the CC3200 board and they can decide whether to replay the game by sending messages to the AWS.

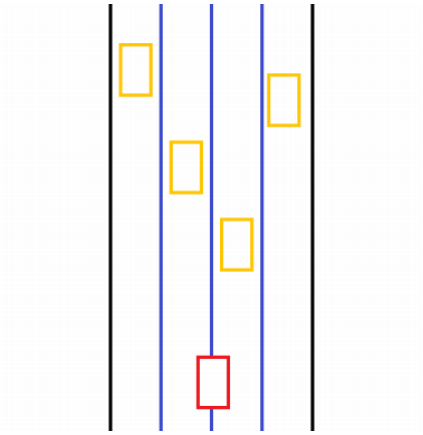
II. PROCESS OF THE GAME PROGRAM

```
main(){
   playGame();
}
playGame(){
   Initialize variables and parameters;
   Display life count;
   Initialize all the cars' data;
   Loop{
      Determine speed of the game;
      Increase score;
      Read button input and determine stealth mode;
      Show stealth cooldown time if enabled;
      Read I2C value and draw user's car;
      Determine collision and reduce life if collided;
      Spawn new car objects if conditions are met;
      Move and draw all the car objects;
      Print remaining life;
      Go to endGame() if life is zero, else go to Loop
   }
}
endGame(){
   Display "Game Over" on Oled screen;
   Send score to user and wait user's input;
   If user decide to replay, go to playGame();
}
```

III. CONSTRUCT THE GAME

A. Build the Car Objects to Avoid

To make the game as intuitive to play as possible, the car objects will spawn in four lanes with appropriate probabilities, so that there will be just enough car objects on the screen with appropriate distance from each other for users to avoid.



Basic Idea of The Game
Each lane is 15px wide, car object is 10x16px.

For the actual coding, I first struct and define the car object and lane object.

```
struct cars{
    uint8_t enabled;
    int x, y, prevX, prevY;
};
```

The car object has x and y coordinates and previous x and y coordinates for wiping the previous car object. Enabled is indicating if the car object is activated.

```
struct lanes {
    uint8_t num_cars;
    uint8_t coordX;
    struct cars car[MAXCARALLOWED];
}; // MAXCARALLOWED = 2
```

The lane object keeps track of how many cars on the lane and its x coordinate on the OLED display. Since each lane will have the same x for all the cars, it is better to define them in the first place. Finally, I hold the two cars objects' place in the memory.

I start the game by initializing all the parameters of 8 possible cars in all the lanes.

Then, I can start spawning the car object on the OLED screen. Because there have 4 lanes, I wrote a for loop to go over all the 4 lanes.

```
for (i = 0; i < NUMLANES; i++)
// NUMLANES = 4
```

For each lane, I have a probability to determine if the car object is spawned. I first determine the function to generate a random number.

```
#define randnum(min, max) {((rand() %
(int)(((max) + 1) - (min))) + (min))}
```

Then get the number.

```
uint8_t randnum = randnum(0,100);
```

I then have to check the probability, if randnum is greater than diff(98), spawn the car. But there is another thing to do: check if the number of car objects is under 2, so that the oled have room to spawn the car.

```
if (randnum > diff && lane[i].num_cars <
MAXCARALLOWED)
```

After completing the loop, we draw them out and should see some car objects on the screen. Similarly, I loop through the 4 lanes and draw car objects out if the car object is enabled.

```
if(lane[i].car[0].enabled) // then draw
```

After the program completes a cycle, I have to move the existing car objects down. The speed is determined by trial and error. The SPI has a fixed speed so that the cars are moving faster when there is less object need to draw and vice versa. The best combination is to move 1 pixel down if less than 3 car objects present, and 2 pixels if more than 3 car objects.

To move the car down, I first check if it is enabled, then I can wipe the previous car object and draw a new one after applying speed offset. Another important point to note is if the car moves out of bound (y > 127) I have to disable it and reset its parameters.

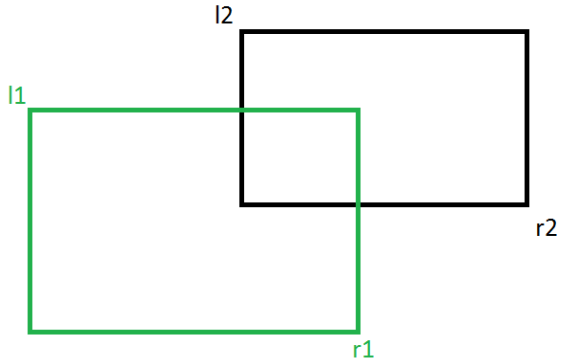
```
if (lane[i].car[0].y > 127) {
lane[i].car[0].enabled = 0;
Lane[i].num_cars--;
...
```

Then, there we have a nice generating and moving pattern of the car objects moving down the screen.

B. Draw User Car and Determine Collision

Similar to Lab3 where I drew the ball using input from the accelerometer on the CC3200 board, the user's car's movement is determined by the data from the accelerometer. There is not much code to change rather than setting the boundary(34<x<85)

Then, I have to determine the collision, which was quite difficult to achieve when I first worked on it. But luckily Google helped me out. The idea was that the two rectangles do not overlap each other if 1) One is above the top edge of another, 2) One is on the left side of another. Otherwise, they overlap.[1]



Picture from geeksforgeeks.org

The l1, r1, l2, and r2 all have their (x,y) coordinates. The condition is then

```
If (l1.x >= r2.x || l2.x >= r1.x || l1.y <=
r2.y || l2.y <= r1.y) {
// does not collide, continue; }
else {
// collide, with other codes }
```

After that, the users can play the game and try to avoid collision by moving their own cars. Also, the program can determine if users have crushed the car, and reduce the life remain of users. There was one problem remain, which is there might be more cycles where user car touches the other car, so that I have to set a cooldown time for reducing life. The time is when cars have moved 40 pixels.

IV. ADDING OTHER FEATURES

A. Adding Stealth Mode

The stealth mode is trigger by the button SW2 on the CC3200 board. Because users can push the button at any time, I decided to poll the button at each cycle.

```
int32_t pinRead = GPIOPinRead(GPIOA2_BASE,
0x40);
```

After trigger the button and stealth mode is not in

cooldown, we enter the stealth mode and set other variables.

```
if (pinRead == 64 && stealthCD < 0 ) {
          stealth = 1;
          stealthCD = 200;
          stealthTime = 34;
}
```

The stealth state is tracked by the stealth variable, if stealth is 1, the users' car changes to white color and ignores all the collisions. After stealthTime is up, which is 34 pixels move, the stealth mode is off and enters cooldown mode. After 200 pixels (4-5s), the cooldown is complete and the user can use it again.

In addition, there is an indication of "CD" text on the top-right corner to indicate if the skill is in cooldown.

B. Display Remaining Life

Initially, the users have 3 lives, after they collide once, the life count reduces by one. When there are 0 lives left, the game ends.

The life counter is located on the top-left corner of the OLED display, with different colors: green, yellow, and red to make users more engaged in the game.

If 0 lives remain, the game stops, and the program goes to the function endGame(), where the text "Game Over" is showed on the display and the score is sent to the user by AWS service.

C. The AWS service

To send the score info to the user, I reused the code in lab4. After creating the shadow CC3200 and sending keys to the board, I used RESTful API to connect to my AWS endpoint. By sending the data below to the AWS, the board triggers the topic I created in lab4. The shadow is updated, and the score information is sent to the user.

```
{\"state\": {\n\r\"desired\" :
{\n\r\"message\" : \"Game Ended! Your Score
is %i. Thanks for playing!\"\n\r}}}\n\r\n\r",
score);
```

Below is the message received.

```
{"message":"Game Ended! Your Score is 6076.
Thanks for playing!"}
```

To let users send back messages to the board, I need the 2-way SMS function. I have to set a lambda function[2] first so that the AWS can recognize and react to what users send to the AWS. The keyword is "replay". After shadow is updated, the board should restart the game.

To send a message to the board, we should update the state in the lambda function. The keyword for pinpoint 2-way SMS should set to "replay."

```
update_state = { 'state':{ 'desired': {
'message' : 'restart', } } }
client = boto3.client('iot-data',
region_name='us-west-2',
endpoint_url='a1i90nzfq510vq-ats.iot.us-west-
2.amazonaws.com')
response = client.update_thing_shadow(
thingName='CC3200', payload=
json.dumps(update_state) )
```

To get the shadow update from the lambda function, the CC3200 board must get its shadow for the AWS using RESTful API and use http_post() to communicate.

```
"GET /things/CC3200/shadow HTTP/1.1\n\r"
```

Then it can get the message part and compare if it is equal to "replay," if yes, it will call playGame() to restart the game.

The SMS service is not available in my region but I went through the whole process of creating it and should work.

V. Result and Conclusion

I accomplished all the design ideas that were in my project proposals, including the basic interface, spawning and moving car objects, determining collision, implementing stealth feature and life counting, and also the AWS communication. Though I did not have a chance to try to send back messages to AWS because of lacking service in my region, I have dived into the lambda functions and pinpoint and studied how to communicate with the AWS server. Through using SPI, I2C, and RESTful API, I have designed an engaging game that interacts with the user. The experience was very educative and interesting.

By learning and exploring the embedded system, I have a glimpse of how capable and powerful it is, especially when IoT comes into place that connects everything to the internet.

REFERENCES

[1] GeeksforGeeks, "*Find if two rectangles overlap.*" Available:https://www.geeksforgeeks.org/find-two-rectangles-overlap/

[2] Amazon AWS, "AWS Lambda Developer Guide." Available:https://docs.aws.amazon.com/lambda/latest/dg/welcome.html