

Part 4

Lesson

4

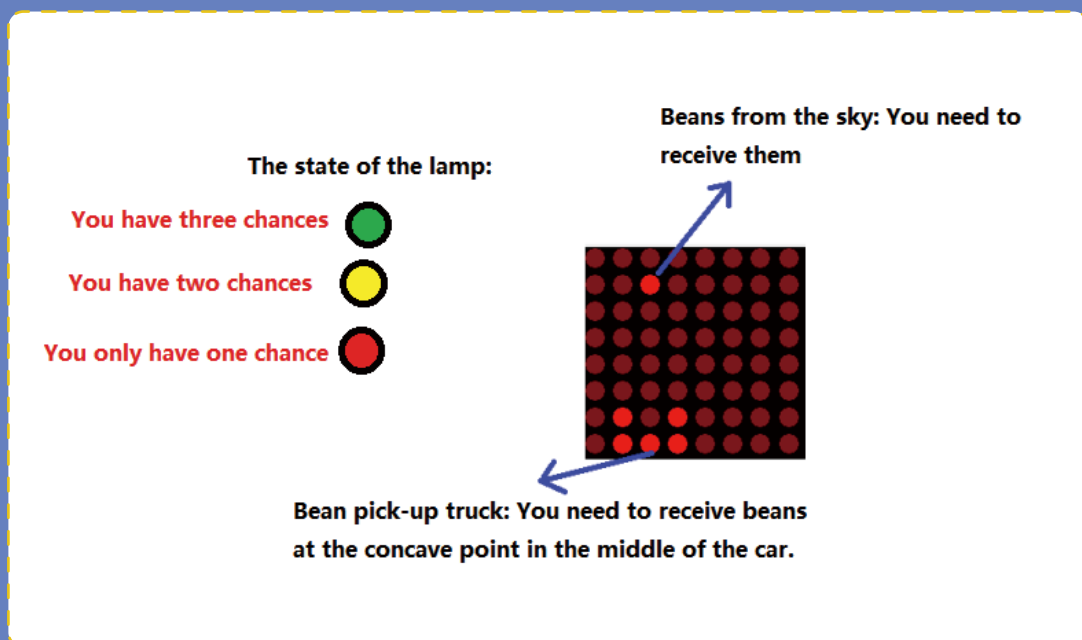
Fruit Forecast

Overview

Through the practical learning of the game project of receive bean, we can strengthen the understanding of the practical application of MAX7219 lattice screen module, MPU6050 module and buzzer, and cultivate the programming idea and feel the charm of Arduino.

Component Required:

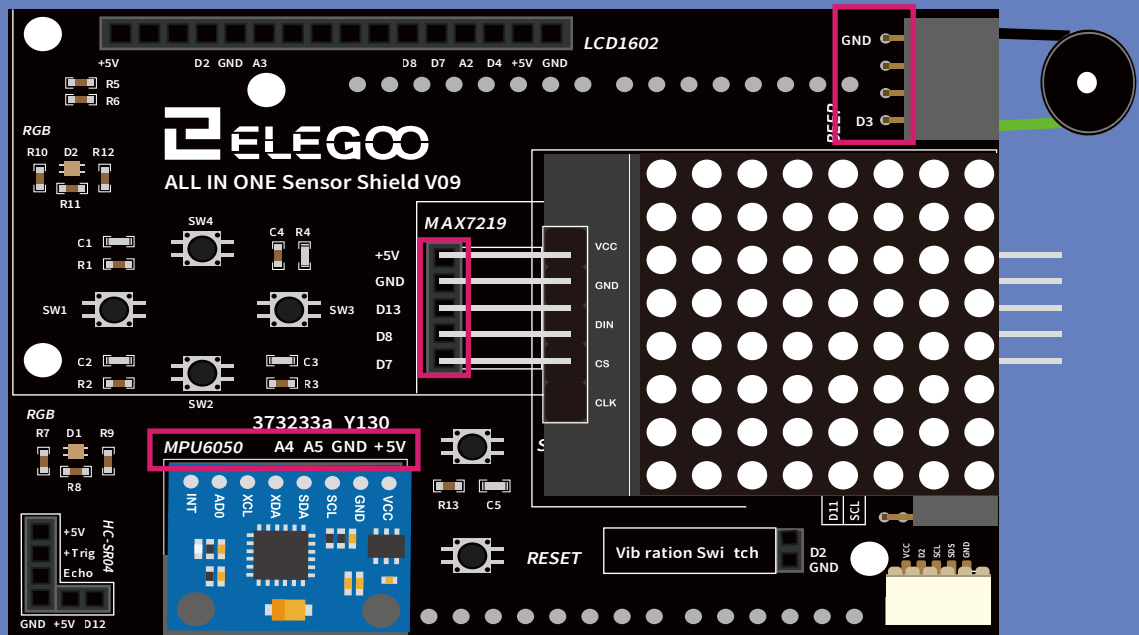
- (1) X Elegoo UNO R3
- (1) X MPU 6050 module
- (1) X MAX7219 lattice screen module
- (1) X RGB LED
- (1) X Passive Buzzer



Shake the extensions board left and right to control the movement of the car. The beans will be caught if the center concave is aligned with beans. If you succeed in catching more than 10 beans, it will show a smiling face pattern, otherwise it will show game over. When the game is over, shake the expansion up and down to restart the game.

Connection schematic diagram:

Tips: Please insert the extended board into UNO.



Make sure you add libraries before downloading the program

Part1 (demo1):

Control the movement of the car

/**/Please open **demo1** which in the **code**

First we need to make a car displayed on the MAX7219, so we need to define an array of cars.

Int car [2]={B00000111, B00000101}.

Because the subscript "[num]" counts from zero.

Car [0] is "B00000111"

Car [1] is "B00000101"

There are two. So num is 2.

So far, the car have been made

```
void game_init() //car init
{
    lc.setRow(0,7,car[0]);
    lc.setRow(0,6,car[1]);
}
```

Theory (PWM)

Next, programming to control the car to move left and right.

Details can be found in the following function:

Control_right()

Control_left()

To achieve vehicle movement, we can use "<<" to move B00000111 to the left.

Example: **B00000111<<1** becomes **B00001110**

 **B00000111**

 **B00001110**

So when it's on the screen, it moves one bit.

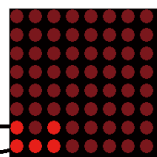
5. When we are able to control the vehicles to move left and right, we need to consider several issues.

We need to run an edge collision test every time the vehicles were moving. Otherwise it may move to the edge of left /right or disappear.

So when the vehicle moves to the following position, we should cancel this movement and reset the position of vehicle.

The first part of the detailed code, please open Demo1 in code.

car [1] : B00000101
Car [0] : B00000111



```
void Control_right()
{
    .....
    else{
        car[0]=car[0]<<1;
        car[1]=car[1]<<1;
    }
    .....
}
```

```
void Control_right()
{
    if(AcY>5000)
    {
        if(car[0]==B00000011)
        {
            car[0]=B00000111;
            car[1]=B00000101;
        }
        if(car[0]==B11000000)
        {
            car[0]=B00000011;
            car[1]=B00000010;
        }
    }
    .....
}
```

Part2 (demo2):

Bean Falling

/**/Please open **demo2** which in the **code**

For details, please refer to our `pro_random_beans()` function.

```
void random_init()
{randomSeed(analogRead(0));} //random init
```

1. Because the dropping of beans is random, we need to add `randomSeed(analogRead(0))` to produce a random number and select it as the column that will generate a beans.

2. Then set the range of random numbers.

```
uint8_t bean_x = 0; //Storage location: At which point16_t the bean falls on the x-axis
uint8_t bean_y = 0; //Storage location: At which point16_t the bean falls on the y-axis
void beans_init()
{bean_y = random(7);
 bean_x=0;}
```

Because our screen has eight columns(0~7)

3. When the beans are identified in which column they are produced, they will gradually fall from the sky.

We can make sense as following: the lights on the same column will turn on and turn off in sequence.

We use the **for** loop here.

Incidentally, a variable **bean_speed** is set as the interval between the lights on and off in the same column in order, that is, the speed at which the beans fall.

```
void pro_random_beans()//produce ramdon beans
{
  beans_init();
  for(bean_x=0;bean_x<=7;)
  {
    beans_falling();
  }
}
void beans_falling()
{
  lc.setLed(0,bean_x,7-bean_y,true);
  delay(bean_flick_speed);
  lc.setLed(0,bean_x,7-bean_y,false);
  delay(bean_flick_speed);
  delay(300);
  bean_x++;
}
```

The second part of the detailed code, please open **demo2** in code.

Part 3(demo 3):

Receive bean

- `/**/`Please open `demo 3` which in the `code`
- At this time, we add the `control_right()` function , `Control_left()` function and `game_init()` function written in Part1 to the `pro_random_beans()` function in part2 to generate the effect of picking up beans.
- The third part of the detailed code, please open `demo3` in code.

```
void pro_random_beans()//produce ramdon beans
{
    beans_init();
    for(bean_x=0;bean_x<=7;)
    {
        update_Ac();
        Control_right();
        Control_left();
        game_init();
        beans_falling();
    }
}
```

Part 4(demo 4):

Animation and Sound Production

- `/**/`Please open `demo 4` which in the `code`
`/**/` We can change the following variable to modify the animation and sound speed.
- This part of the code mainly realizes the sound effect, lighting effect and screen display in the game of receiving up beans.
- The forth part of the detailed code, please open `demo 4` in code.

```
int16_t er_show_speed=150; //enlarge_reduce_display speed
int16_t heart_show_speed=150;
int16_t game_show_speed=80;
int16_t music_speed=100;
int16_t car_display_speed=85;
```

Part 5 (receive_beans_G):

receive_beans_G

- `/**/`Please open `receive_beans_G` in the `code` folder
- The last part is a combination of the previous parts. In this part, we mainly learn how to realize the mechanism of receiving beans.
- Principle:
Judge whether `car [0]` is in the same place(`x[]`) when the beans fall to the last row.

```
//// When the beans fall to the last grid, the car must be in the X [] position to receive the beans correctly.
uint8_t x[]={B00000011,B00000111,B00001110,B00011100,B00111000,B01110000,B11100000,B11000000};
```

- If `car [0]==x [bean_f]` picks up beans successfully
If `car [0] != x [bean_f]` failed to pick up beans
- The fifth part of the detailed code, please open `receive_beans_G` in code.

```
void case_ending_judge()
{
    if(bean_x>7)//When the beans fall to the last grid
    {
        switch(bean_y)//Determine which column is the last grid
        {
            case x1:if(car[0]!=x[bean_y]) {case_run();}
                    else {success_receive_voice();
                        beans_received_num++; } break;
            .....
        }
    }
}
```