

单片机大作业

单片机大作业

设计目的

设计所需器件

设计流程

模块划分

实现流程

器件硬件配置

一、STM32核心板

二、摇杆

三、OLED

器件软件设计

一、STM32核心板

二、摇杆

三、OLED

游戏主体设计

设计思想

头文件定义

变量与常量定义

游戏地图初始化

初始化蛇身

刷新蛇身显示

蛇的移动及相关信息的更新

随机产生新食物

地图数组映射OLED显示

游戏状态控制

蛇移动控制

用户交互控制

增加游戏趣味性

实现效果

初始游戏界面

游戏进行界面

游戏结束界面 (按下摇杆实现restart)

总结

设计目的

- 简单游戏贪吃蛇在单片机上的实现

设计所需器件

- STM32F103C8T6核心板
- 0.96寸OLED(68*128)
- 摆杆

设计流程

模块划分

模块	作用
MAIN	单片机主程序
STM32_CONFIG	STM32核心板基础配置
TIMER_CONFIG	计数器TIM4配置、获取当前计数器的值、判断操作时间间隔
ADC_SET	配置摇杆IO、ADC与DMA配置、获取摇杆当前状态
OLED_SET	配置OLED的IO、字符在OLED上的显示函数、显存数据发送至OLED显示
SNAKE_GAME	游戏设计主体，包含页面设计、游戏逻辑交互

实现流程

1. STM32核心版运行的基础配置
2. 外设OLED与摇杆的IO接口配置
3. 设计OLED与摇杆的基础函数，便于SNAKE模块对外设的控制
4. 测试1：核心板与外设连线，设计简单程序，测试单片机能否正常通过摇杆控制、通过OLED屏幕显示
5. 游戏设计：页面设计、游戏交互逻辑设计
6. 测试2：游戏软体在PC端测试
7. 测试3：软硬体结合，调整参数，调试BUG，并减少因信号采集判断而导致的误触

器件硬件配置

一、STM32核心板

1. 配置系统时钟

```
// 配置串口1—(USART1) 时钟
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1 | RCC_APB2Periph_GPIOA, ENABLE);
```

2. 配置NVIC和GPIO

```

// 配置串口1—接收终端的优先级
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);
NVIC_InitStructure.NVIC IRQChannel = USART1 IRQn;
NVIC_InitStructure.NVIC IRQChannelPreemptionPriority = 1;
NVIC_InitStructure.NVIC IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);

// 配置串口1—发送引脚(PA.09)
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOA, &GPIO_InitStructure);

// 配置串口1 接收引脚 (PA.10)
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOA, &GPIO_InitStructure);

// 配置串口1—工作模式 (USART1 mode)
USART_InitStructure.USART_BaudRate = 115200;
USART_InitStructure.USART_WordLength = USART_WordLength_8b; // 数据位为8个字节
USART_InitStructure.USART_StopBits = USART_StopBits_1; // 一位停止位
USART_InitStructure.USART_Parity = USART_Parity_No ; // 无校验位
USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None; // 无需流控制
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx; // 配置接收发送模式
USART_Init(USART1, &USART_InitStructure);

USART_ITConfig(USART1, USART_IT_RXNE, ENABLE); // 开启中断
USART_Cmd(USART1, ENABLE); // 使能串口

```

3.配置TIMER

```

void TIME_Configuration(void) // 配置TIM4
{
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
    // 使能TIM4时钟
    TIM_TimeBaseStructure.TIM_Period = 0xFFFF; // 范围 0x0000-0xFFFF
    // 设置在下一个更新事件装入活动的自动重装载寄存器周期的值
    TIM_TimeBaseStructure.TIM_Prescaler = (36000-1);
    // 设置用来作为 TIM4 时钟频率除数的预分频值
    TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);
    // 根据指定的参数初始化 TIM4 的时钟基数单位
    TIM_Cmd(TIM4, ENABLE);
    // 使能 TIM4
}

```

二、摇杆

引脚配置

摇杆	STM32核心板
GND	电源地
VCC	接5V或3.3V
VRX	PA.2
VRY	PA.1
SW	PA.0

GPIO配置

```
void Adc_GPIO_Config()
{
    GPIO_InitTypeDef GPIO_InitStructure;
    // 使能GPIO和ADC1通道时钟
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_ADC1, ENABLE);

    // 将PA0设置为模拟输入
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;

    // 将GPIO设置为模拟输入
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;

    // 将GPIO设置为模拟输入
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;

    // 将GPIO设置为模拟输入
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```

ADC配置

```
// 72M/6=12,ADC最大时间不能超过14M
RCC_ADCCLKConfig(RCC_PCLK2_Div6);

// 将外设 ADC1 的全部寄存器重设为默认值
ADC_DeInit(ADC1);
// ADC工作模式:ADC1和ADC2工作在独立模式
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
// 模数转换工作在单通道模式
ADC_InitStructure.ADC_ScanConvMode = ENABLE;
// 模数转换工作在单次转换模式
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
// ADC转换由软件而不是外部触发启动
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
// ADC数据右对齐
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
// 顺序进行规则转换的ADC通道的数目
ADC_InitStructure.ADC_NbrOfChannel = 3;
// 根据ADC_InitStruct中指定的参数初始化外设ADCx的寄存器
ADC_Init(ADC1, &ADC_InitStructure);

ADC-RegularChannelConfig(ADC1, ADC_Channel_0, 1, ADC_SampleTime_239Cycles5 );
ADC-RegularChannelConfig(ADC1, ADC_Channel_1, 2, ADC_SampleTime_239Cycles5 );
ADC-RegularChannelConfig(ADC1, ADC_Channel_2, 3, ADC_SampleTime_239Cycles5 );
// 使能指定的ADC1
ADC_Cmd(ADC1, ENABLE);
ADC_DMACmd(ADC1, ENABLE);

// 重置指定的ADC1的校准寄存器
ADC_ResetCalibration(ADC1);
// 获取ADC1重置校准寄存器的状态,设置状态则等待
while(ADC_GetResetCalibrationStatus(ADC1));
// 开始指定ADC1的校准
ADC_StartCalibration(ADC1);
// 获取指定ADC1的校准程序,设置状态则等待
while(ADC_GetCalibrationStatus(ADC1));
```

DMA配置

```
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);

DMA_InitStructure.DMA_PeripheralBaseAddr = (u32)&(ADC1->DR);
DMA_InitStructure.DMA_MemoryBaseAddr =(u32)&ADC_ConvertedValue;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
DMA_InitStructure.DMA_BufferSize = 30;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;

DMA_Init(DMA1_Channel1, &DMA_InitStructure);
DMA_Cmd(DMA1_Channel1, ENABLE);
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
```

三、OLED

引脚配置

OLED	STM32核心板
GND	电源地
VCC	接5V或3.3V电源
D0(CLK)	PA.5(CLK)
D1(MOSI)	PA.7(DIN)
DC	PB.0
CS	PB.1
RES	接3.3V电源，接地时会导致复位

GPIO配置

```
void GPIO_Config(void) // IO端口初始化
{
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

    //PB.0->DC引脚 PB.1->CS引脚
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 ;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    //PA.5->CLK引脚 PA.7->MOSI引脚
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_7 ;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```

写指令到LCD模块

```
// 写指令到LCD模块
void OLED_TX_Command(int data)
{
    char i;
    lcd_dc(0), lcd_cs(0);
    for (i = 0; i < 8; i++)
    {
        lcd_sclk(0);

        if (data & 0x80) lcd_sid(1);
        else lcd_sid(0);

        lcd_sclk(1), __nop();
        data <<= 1;
    }
    lcd_dc(1), lcd_cs(1);
}
```

写数据到LCD模块

```
// 写数据到LCD模块
void OLED_TX_Data(int data)
{
    char i;
    lcd_dc(1), lcd_cs(0);
    for (i = 0; i < 8; i++)
    {
        lcd_sclk(0);

        if (data & 0x80) lcd_sid(1);
        else lcd_sid(0);

        lcd_sclk(1), __nop();
        data <<= 1;
    }
    lcd_dc(1), lcd_cs(1);
}
```

OLED配置

```

// OLED 初始化
void OLED_Init(void)
{
    delay_ms(800);
    GPIO_Config();
    lcd_cs(0); // 片选引脚激活

    OLED_TX_Command(0xAE); // display off
    OLED_TX_Command(0x20); // Set Memory Addressing Mode
    OLED_TX_Command(0x10); // 00,Horizontal Addressing Mode;01,Vertical Addressing Mode;10,Page Addressing Mode (RESET);11,Invalid
    OLED_TX_Command(0xb0); // Set Page Start Address for Page Addressing Mode,0-7
    OLED_TX_Command(0xc8); // Set COM Output Scan Direction
    OLED_TX_Command(0x00); //---set low column address
    OLED_TX_Command(0x10); //---set high column address
    OLED_TX_Command(0x40); //---set start line address
    OLED_TX_Command(0x81); //---set contrast control register
    OLED_TX_Command(0XF);
    OLED_TX_Command(0xa1); //---set segment re-map 0 to 127
    OLED_TX_Command(0xa6); //---set normal display
    OLED_TX_Command(0xa8); //---set multiplex ratio(1 to 64)
    OLED_TX_Command(0x3F); //
    OLED_TX_Command(0xa4); //0xa4,Output follows RAM content;0xa5,Output ignores RAM content
    OLED_TX_Command(0xd3); //---set display offset
    OLED_TX_Command(0x00); //---not offset
    OLED_TX_Command(0xd5); //---set display clock divide ratio/oscillator frequency
    OLED_TX_Command(0xf0); //---set divide ratio
    OLED_TX_Command(0xd9); //---set pre-charge period
    OLED_TX_Command(0x22); //
    OLED_TX_Command(0xda); //---set com pins hardware configuration
    OLED_TX_Command(0x12);
    OLED_TX_Command(0xdb); //---set vcomh
    OLED_TX_Command(0x20); //0x20,0.77xVcc
    OLED_TX_Command(0x8d); //---set DC-DC enable
    OLED_TX_Command(0x14); //
    OLED_TX_Command(0xaf); //---turn on oled panel
}

```

器件软件设计

一、STM32核心板

重定向C中的fputc至串口

```

int fputc(int ch, FILE *f)
// 重定向c库中fputc至串口，使得printf打印的信息在串口上发送，PC上用串口助手则接收信息
{
    // 将Printf内容发往串口
    USART_SendData(USART1, (unsigned char) ch);
    while (USART_GetFlagStatus(USART1, USART_FLAG_TC) != SET);
    return (ch);
}

```

获取当前计数器的值

```

//获取当前计时器的数值
void Tick_Update(uint32_t* tick)
{
    *tick = TIM_GetCounter(TIM4);
}

```

判断前后两次同类操作发生的时间，是否在允许的时间间隔内

```
//限制操作发生的时间间隔
int Check_TimeOut(uint32_t* oldtick, uint32_t diff_tick)
// return 1 表示超时  return 0 表示没有超时
{
    uint32_t ticknow;
    uint32_t diff;

    ticknow = TIM_GetCounter(TIM4);
    diff = ticknow - (*oldtick);
    return diff > diff_tick;
}
```

二、摇杆

查询当前摇杆状态

```
//查询当前摇杆状态
unsigned char QueryState()
{
    uint8_t temp1, temp2;
    uint8_t event = NON;

    get_Average();

    // 计算X坐标的偏移
    temp1 = (ConvertedValue[2] & 0xF00) >> 8;
    if (temp1 < 7) // 向左
        event = TURN_LEFT_EV;
    else if (temp1 > 8) // 向右
        event = TURN_RIGHT_EV;

    // 计算Y坐标的偏移
    temp2 = (ConvertedValue[1] & 0xF00) >> 8;
    if (temp2 < 7) // 向上
        event = TURN_UP_EV;
    else if (temp2 > 8) // 向下
        event = TURN_DOWN_EV;

    // 确定SW按键的状态
    if (ConvertedValue[0] < 0x03)
        // 缩小范围至刚好按下能够reset, 减少误触几率
        event = RESTART_EV;

    return event;
}
```

三、OLED

在OLED对应坐标上描点

```
// OLED对应坐标上描点
void LCD_DrawPoint(uint8_t x,uint8_t y,uint8_t color)
{
    uint8_t temp, idx, page;

    if (color > 1) color = 1;
    idx = y % 8; // idx=0
    page = y / 8;

    temp = LCD_cache[page][x];
    temp &= ~(0x01 << idx); // clean bit
    temp |= color << idx; // write bit
    LCD_cache[page][x] = temp;
}
```

在OLED特定位置显示一个字符

```
//在OLED特定的位置显示一个字符
void GUI_DrawChar(uint16_t x0, uint16_t y0, unsigned char ch)
{
    uint8_t xcnt, ycnt, color, count = 0;
    const uint8_t* pData;

    pData = &GUI_Font8x8ASCII_Data[8 * (ch - 0x20)];

    for (xcnt = 0; xcnt < 8; ++xcnt)
    {
        count = 0;
        for (ycnt = 0; ycnt < 8; ++ycnt)
        {
            color = ((*pData) & (0x80 >> count)) > 0 ? 1 : 0;
            LCD_DrawPoint(y0 + ycnt, 63 - (x0 + xcnt), color);
            count++;
        }
        pData++;
    }
}
```

在OLED指定位置显示字符串

```
// 在指定的位置显示字符串
void OLED_DispString(int x0, int y0, char *str)
{
    while (*str)
    {
        GUI_DrawChar(x0, y0, *str);
        str++, x0 += 6;
        if (x0 >= 64)
            x0 = 0, y0 += 10;
    }
}
```

将显存数据发送到OLED进行显示

```
//将显存里的数据发送到oled进行显示
void OLED_Update()
{
    unsigned char i, j;

    lcd_cs(0);
    for (i = 0; i < 8; ++i)
    {
        OLED_TX_Command(0xb0 + i);
        OLED_TX_Command(0x00);
        OLED_TX_Command(0x10);
        for (j = 0; j < 128; ++j)
            OLED_TX_Data(LCD_cache[i][j]);
    }
    lcd_cs(1);
}
```

游戏主体设计

设计思想

- 我们设计的贪吃蛇界面由OLED屏幕的64120像素点组成，实现了1630大小的地图供贪吃蛇活动，每一个格子对应于OLED屏幕上4*4的像素点。
- 游戏分为三个状态：游戏未开始，游戏进行中，游戏结束。贪吃蛇碰到自己身体或者墙则进入“游戏结束”状态，按restart则重新进入“游戏进行中”状态。
- 贪吃蛇游戏界面包含两个代码文件，SNAKE_GAME.h头文件定义了需要的数据类型和参数，SNAKE_GAME.c具体实现了头文件定义的各种函数。
- 下面结合具体代码介绍实现的代码的主要思想。

头文件定义

```
#ifndef __SNAKE_H__
#define __SNAKE_H__


#define HEIGHT 30
#define WIDTH 16
#define GAME_RUNING 0
#define GAME_PAUSE 1

#define NULL 0x00

typedef struct SnakeNode{
    int x;
    int y;
    struct SnakeNode *next, *prev;
}SNAKELIST;
//存储蛇的节点，数据结构为双向循环链表
//头节点的前驱为尾节点，尾节点的后继为空
```

- SNAKE_GAME.h头文件定义了贪吃蛇游戏的主要函数原型，以及存储贪吃蛇结点的数据结构，以及定义了一些重要游戏参数常量。
- ifndef的预编译指令保证了该头文件只会被一次，避免该头文件被多重定义导致出现错误。
- 定义了游戏状态常量GAME_RUNING为0，表示游戏进行。GAME_PAUSE为1，表示游戏暂停（游戏结束或未开始）。定义了地图宽度WIDTH为16，高度HEIGHT为30。
- SnakeNode存储了贪吃蛇身体的一个结点，表示贪吃蛇身体的其中一节。(x,y)表示贪吃蛇这一节身体在地图上的坐标。next,prev是指向SnakeNode数据类型的指针，存储时按类似双向循环链表存储。对于非头结点，其prev为其在蛇身体上的前一结点，头结点的prev为蛇的尾结点（方便定位蛇的尾结点）。对于任意一个非尾结点，其next为蛇身体上的后一结点，尾节点的next为NULL。

变量与常量定义

```
SNAKELIST SnakeHeadNode;
// 存储蛇的链表头节点
SNAKELIST SnakeNodeBuffer[HEIGHT*WIDTH];
// 不能使用malloc动态分配内存 通过预先申请的静态内存进行内存管理
int SnakeNodeBufferIndex = 0, SnakeDirection = TURN_UP_EVENTENT;
// SnakeNodeBufferIndex记录静态内存区使用的结点编号
unsigned int speed_turn = 0, speed_restart = 0;
unsigned int speed_move = 0, speed_max = 200, speed_move=400;
unsigned char Game_Map[HEIGHT][WIDTH] = {};
// 游戏图像的像素缓存
int GameScore = 0, GameStatus = GAME_PAUSE;
```

- SnakeHeadNode存储蛇的双向链表的头结点，通过这个头结点可以定位到蛇身体的所有点
- 由于单片机在这种情况下不能使用malloc的内存管理函数来实现动态分配内存，所以这里采用了内存池的方法，预先申请了一批静态内存结点，自己实现内存管理，在插入和删除蛇结点的时候自

行分配给一个内存池中的结点。由于蛇结点数目最多为地图的大小，所以静态内存池的大小设置为HEIGHT*WIDTH。

- SnakeNodeBufferIndex是用于内存池管理的辅助变量，存储当前未用的内存池结点在内存池中的编号。每次需要一个新的结点时，就取出SnakeNodeBufferIndex编号对应的结点使用，然后SnakeNodeBufferIndex自增1。
- SnakeDirection表示贪吃蛇的行进方向。
- speed_turn, speed_restart, speed_move, speed_max用于用户交互的管理，用于判定是否触发转向、重启等事件。
- Game_Map存储了游戏的地图的当前状态。
- GameScore记录游戏当前分数，GameStatus记录游戏当前状态。

游戏地图初始化

```
void GAME_BackgroundInit(unsigned char (*Game_Map)[WIDTH])
// 游戏地图初始化
{
    int i, j;
    memset(Game_Map, 0, HEIGHT*WIDTH);

    for (i = 0; i < WIDTH; ++i)
    { //地图上下边界
        Game_Map[HEIGHT-1][i] = 1;
        Game_Map[0][i] = 1;
    }

    for (i = 0; i < HEIGHT; ++i)
    { //地图上左右界
        Game_Map[i][WIDTH-1] = 1;
        Game_Map[i][0] = 1;
    }
}
```

- GAME_BackgroundInit实现了游戏地图的初始化，将地图的边界设置为障碍(1表示蛇的身体或者障碍)，其他地方设置为0(可以通行)。这里采用了传数组指针的编程技巧。

初始化蛇身

```
void GAME_New_Init_Snake(SNAKELIST* SnakeList) //初始化链表，初始化蛇长度，4+1节
{
    int x = 6, y = 12, i;//设定好蛇的初始位置

    SnakeList->x = x;
    SnakeList->y = y++;
    SnakeList->prev = SnakeList;
    SnakeList->next = SnakeList;

    for (i = 0; i < 4; i++)
        GAME_Snake_List_Add_A_Node(SnakeList, x, y + i);

    GAME_Snake_Fill_In_The_Game_Map(Game_Map, SnakeList);
}
```

- 该函数初始化了蛇身体。先设定蛇头的初始位置，然后申明蛇头，然后开辟4个身体结点并插入，并将游戏地图进行初始化。

刷新蛇身显示

```
void GAME_Snake_Fill_In_The_Game_Map(unsigned char (*Game_Map)[WIDTH], SNAKELIST* SnakeList)
// 将蛇所在位置填入游戏图像缓存
{
    while (n->next != NULL)//遍历蛇每个结点
    {
        Game_Map[SnakeList->y][SnakeList->x] = 1;
        n = n->next;
    }
}
```

- GAME_Snake_Fill_In_The_Game_Map函数实现了将蛇的身体填入地图缓存。从链表头开始遍历整个链表，将每个结点在地图上都标记为1(在这里表示蛇的身体)。

蛇的移动及相关信息的更新

- 蛇的移动首先需要清理蛇尾结点在地图的缓存空间，然后需要修改蛇所有结点的x与y坐标，最后考虑是否需要插入新的结点。

```
void GAME_Snake_Destroy_In_The_Game_Map(unsigned char (*Game_Map)[WIDTH], SNAKELIST* SnakeList)
// 移动前将当前蛇尾去掉
{
    Game_Map[SnakeList->prev->y][SnakeList->prev->x] = 0;
}
```

- 该函数实现了蛇在移动过程中，将当前蛇尾巴在地图缓存中清理掉的功能。这种功能展现让蛇当前尾巴结点在地图上消失，实现了蛇尾巴移动的下效果。

```
void GAME_Snake_List_Add_A_Node(SNAKELIST* SnakeList, int x, int y)
// 后部添加节点，吃食物时给链表添加节点
{
    SNAKELIST *h = SnakeList, *p = SnakeList->prev;//头结点的前驱为尾结点 尾结点的后继为空

    SnakeNodeBuffer[SnakeNodeBufferIndex].x = x;
    SnakeNodeBuffer[SnakeNodeBufferIndex].y = y;
    SnakeNodeBuffer[SnakeNodeBufferIndex].next = NULL;
    SnakeNodeBuffer[SnakeNodeBufferIndex].prev = p;

    h->prev = &SnakeNodeBuffer[SnakeNodeBufferIndex]; // 头节点的前继节点指向申请的新结点
    p->next = &SnakeNodeBuffer[SnakeNodeBufferIndex]; // 原尾巴节点的后继节点指向申请的新结点

    SnakeNodeBufferIndex++; // 指向SnakeNodeBuffer下一个未用节点，为下一次做准备
}
```

- 该函数实现了在存储蛇身体的链表中添加结点的功能。先用h和p指针分别指向当前蛇头和蛇尾，然后建立新的蛇身体结点，让h前驱和p后继指向新结点，并让新节点的前驱为p，后继为空，实现了在蛇身体这一类双向链表的数据结构中的插入删除。SnakeNodeBufferIndex++表明之前新申请的结点已经使用，该参数指向下一个未使用过的蛇结点，实现了自己的内存管理。

随机产生新食物

```

void GAME_Produce_New_Food(unsigned char (*Game_Map)[WIDTH]) //在地图上随机产生新的食物
{
    unsigned int seed1, seed2;
    int x, y;

    while(1)
    {
        Tick_Updater(&seed1); //取当前时间(相当于随机数)作为随机数
        Tick_Updater(&seed2);
        x = seed1%WIDTH;
        y = seed2%HEIGHT;
        if (!Game_Map[y][x]){//不为蛇身体或者食物
            Game_Map[y][x] = 2;
            break;
        }
    }
}

```

- 该函数实现了在地图上随机产生新的食物。地图产生新食物需要随机产生才能保证游戏的趣味性。如果不引入一些随机信号，只是按照代码实现的固定策略生成随机数，那么可能出现几次游戏中生成的食物都是固定的位置的情况，这样会导致游戏趣味性下降很多，对用户不太友好。因此需要引入一些随机信号来产生随机数。

单片机产生随机数的几种方法

- 使用C语言stdlib.h头文件中的srand方法和rand函数产生伪随机数。但经过测试发现达到比较好效果的伪随机数计算需要一些时间，而单片机计算能力相对较小，贪吃蛇游戏进行速度又较快。为了达到游戏反应速度比较快，延时比较小，用户体验比较好，我们没有采用这种方法产生随机数。
- 使用单片机ADC噪声来产生随机数。但这有可能会受到环境潮湿或电路短路等一些其他因素影响，导致这个采集到的信号可能不变，导致随机数产生失败，因此也未用这种方法。
- 有的单片机自带随机数产生器，可以使用自带的随机数产生器产生随机数。
- 使用自定义的随机数表来产生随机数。这种方法类似于老师上课讲解的，在单片机中用表存储二进制位为1数目函数以及sin等科学函数来加快单片机一些计算的方法。这些基本函数在普通PC机上计算不是问题，通常也都是直接统计或调用库函数，不需要考虑耗时问题。但单片机计算能力相对较小，如果每次都需要计算这些本身也需要一定时间计算的函数，在这些函数大量调用的情况下，系统可能会反应比较慢。因此我们需要根据单片机的特点想出解决方法，我们需要用空间换时间，将这些基本函数的计算结果存储在单片机的存储单元里，调用函数时直接查存储单元中的表即可。硬件层面实现伪随机数也是比较困难的，在单片机上用自定义算法实现伪随机数也是比较麻烦。我们可以先预先将伪随机数表放入存储单元，然后直接调用来实现随机数功能。
- 将单片机和环境，用户交互的一些信号，或者时间信号，来作为随机信号。这里采用了这个方法，取当前系统时间作为随机数seed1,seed2(时间一直均匀流逝，触发事件时随机选定一个时间相当于随机的信号)，然后将这两个原始随机数进行取模，得到新产生食物的坐标。如果当前位置已经有物品(1表示障碍或蛇，2表示食物)，那么就重新随机产生食物。

地图数组映射OLED显示

```

extern volatile unsigned char Cache_LCD[8][128];
// volatile取消编译优化 extern引用外部文件的变量

void Game_Map_To_LCD_Cache(void)
//对游戏图像 (16*30) 进行放大然后填入lcd下方 (64*120) 的显存，游戏界面图像的一个像素转变成lcd的4*4个像素
{
    int i, j, map_X = 15, map_Y = 0;

    for (i = 0; i < 16; ++i)
    {
        map_Y = 0;
        for (j = 8; j < 128; ++j)
        { // 行映射
            if (Game_Map[map_Y][map_X])
            {
                if (!(i & 1))
                    Cache_LCD[i >> 1][j] |= 0x0F; // 低四位置1
                else
                    Cache_LCD[i >> 1][j] |= 0xF0; // 高四位置1
            }
            else
            {
                if (!(i & 1))
                    Cache_LCD[i >> 1][j] &= 0xF0; // 低四位置0
                else
                    Cache_LCD[i >> 1][j] &= 0x0F; // 高四位置0
            }
            if ((j & 3) == 0) map_Y++;
        }
        map_X--;
    }
}

```

- Cache_LCD存储了LCD显示屏的缓存，决定了外设显示屏的显示情况。
- Game_Map_To_LCD_Cache实现了将游戏地图当前信息转换为显示屏输出，由于单片机计算能力相对较小，为了加快单片机运行速度，这里的运算都使用了简单的+1, -1运算，以及位运算。
- 地图存储的元素为1/2，则表示该处有物品(障碍或蛇或事物)，将其映射到LCD缓存数组的对应位置，这里将11空间映射为显示屏上44大小的显示空间，让用户游戏中观察得更清楚，也优化了用户体验。
- Cache_LCD存储数据类型为unsigned char，每一个unsigned char可以存储8个bit，因此根据行号的奇偶性，地图对应于存储LCD缓存的char的高四位或低四位，用和十六进制F的位运算即可实现置0或者置1。

游戏状态控制

```

void GameReady(void) // 游戏初始界面，按下restart按键开始游戏
{
    memset(Cache_LCD, 0, sizeof(Cache_LCD));
    OLED_DispString(0, 35, "Start!");
    OLED_DispString(0, 45, "Please");
    OLED_DispString(0, 55, "Press the");
    OLED_DispString(0, 65, "button");
    OLED_Update();
}

```

- GameReady函数定义了游戏初始界面的显示

```

void GameOver(void) // 游戏结束界面, 按下restart按键重新开始游戏
{
    memset(Cache_LCD, 0, sizeof(Cache_LCD));
    OLED_DispString(0, 35, "You fail!");
    OLED_DispString(0, 45, "Press");
    OLED_DispString(0, 55, "the button");
    OLED_DispString(0, 65, "to restart!");
    OLED_Update();
}

```

- GameOver函数定义了游戏结束界面的显示

蛇移动控制

- GAME_Snake_A_Move是最为重要的函数之一, 用来实现蛇的一次移动: 蛇按照dir方向进行一次移动, Game_Map和SnakeList和gamescore会发生相应变化。

```

unsigned char GAME_Snake_A_Move(unsigned char (*Game_Map)[WIDTH], SNAKELIST* SnakeList, int dir, unsigned int *gamescore)
// 蛇移动, dir=方向
{
    SNAKELIST *p,*h;
    unsigned char val, result = 1;

    GAME_Snake_Destroy_In_The_Game_Map(Game_Map, SnakeList); // 移动前去掉原来的蛇尾

    h = SnakeList;
    p = SnakeList->prev; // 指向前驱节点, 头节点的前驱节点是尾巴

    // 蛇移动
    while(p!=SnakeList)
    { // 从最后一个节点开始, 后继结点的x、y等于前继节点的x、y, 直到头节点为止, 头节点的x、y先不变
        p->x = p->prev->x; // 等于该节点的前继节点的x
        p->y = p->prev->y; // 等于该节点的前继节点的y
        p = p->prev;
    }
}

```

- 首先先清理掉原来的蛇尾结点对应的地图单元, 然后让蛇除了头结点外的所有结点, 其坐标等于其前驱的坐标。

```

switch(dir)
{
    case TURN_DOWN_EVENT: // 下
        h->y++;
        break;
    case TURN_LEFT_EVENT: // 左
        h->x--;
        break;
    case TURN_UP_EVENT: // 上
        h->y--;
        break;
    case TURN_RIGHT_EVENT: // 右
        h->x++;
        break;
    default:
        break;
}

```

- 然后根据蛇的具体移动方向, 修改蛇头结点的坐标。

```

val = Game_Map[h->y][h->x]; // 蛇头将要经过的点

switch(val)
{
    case 0: // 可以移动
        break;
    case 1: // 碰到障碍物或者蛇身
        GameStatus = GAME_PAUSE;
        GameOver();
        printf("GAME OVER!\r\n");
        result = 0; // 游戏失败
        break;
    case 2: // 吃到食物
        GAME_Snake_List_Add_A_Node(SnakeList, h->x, h->y);
        GAME_Produce_New_Food(Game_Map);
        (*gamescore) += 1;
        if ((*gamescore)%4==0) speed_max--;
        break;
    default:
        break;
}

```

- 然后根据蛇头即将经过的位置状态不同，决定触发不同的事件。碰到障碍物或者蛇身体，触发游戏结束事件；吃到食物则触发添加蛇结点步兵累加得分的事件。

```

    GAME_Snake_Fill_In_The_Game_Map(Game_Map, SnakeList);
    return result;
}

```

- 最后刷新蛇在地图上的显示，并返回该函数是否处理成功的结果

用户交互控制

- Game_Handle_Input也是最重要的函数之一，处理了对用户交互信号的反应。

```

unsigned char Game_Handle_Input(unsigned char event) // 对输入按键事件的处理
{
    unsigned char result = 0;

    if (GameStatus == GAME_PAUSE && event != RESTART_EVENT)
        // 游戏状态为未进行游戏时，除非按下restart，否则不进入
        return 0;

    if (event + SnakeDirection == 5)
        // 按下蛇前进的相反方向时，忽略
        event = NON;
}

```

- result同样表示是否成功执行该函数，在最后返回该结果。
- 游戏状态不在进行中时，忽略用户的输入，除非用户的输入是重新开始游戏。
- 如果用户输入的方向和蛇的前进方向相反，由于游戏规则中蛇不可能直接反向，所以当成未发生事件处理。

```

switch(event)
{
    case NON:
        speed_max = 200;
        if (Check_TimeOut(&speed_move, speed_move)) // 自动按原方向前进
            Tick_Update(&speed_move),
            event = SnakeDirection;
        else break;
    case TURN_DOWN_EVENT: // 蛇向下移动
    case TURN_UP_EVENT: // 蛇向上移动
    case TURN_RIGHT_EVENT: // 蛇向右移动
    case TURN_LEFT_EVENT: // 蛇向左移动
        if (Check_TimeOut(&speed_turn, speed_max))
        {
            Tick_Update(&speed_turn);
            speed_max = 160;
            SnakeDirection = event;
            result = GAME_Snake_A_Move(Game_Map, &SnakeHeadNode, event, &GameScore);
            printf("EVENT LOG=%d\r\n", event);
        }
        break;
}

```

- speed_move为上一次按原方向前进的时间，如果当前时间比上一次按原方向前进的时间还多
- SPEED_MOVE(SPEED_MOVE为自动按原方向移动的时间阈值)，则优先触发继续按原方向移动的事件。

```

case RESTART_EVENT: // 游戏复位
    if (Check_TimeOut(&speed_restart, SPEED_RESTART_MAX))
    {
        Tick_Update(&speed_restart);
        GameScore = 0;
        SnakeDirection = TURN_UP_EVENT;
        GameStatus = GAME_RUNNING;
        GAME_BackgroundInit(Game_Map);
        GAME_New_Init_Snake(&SnakeHeadNode);
        speed_move=400;//蛇在400ms的间隔自动前进一步
        GAME_Produce_New_Food(Game_Map);
        printf("EVENT LOG=%d\r\n", event);
        result = 1;
    }
    break;
default:
    break;
}
return result;
}

```

- speed_turn为上一次用户移动的时间，如果当前时间比上一次用户移动的时间多speed_max，才可以触发上一次用户移动事件，(speed_max<SPEED_MOVE，为转向时间阈值，为经过这一时间蛇不能转向)。如果触发了移动的事件，则更新上一次按原方向前进时间和上一次用户移动时间，并让蛇尝试移动一步。
- 如果用户的输入是RESTART_EVENT，则表明用户重新开始游戏，重新初始化游戏的所有设置。

增加游戏趣味性

- 为了增加游戏的趣味性，我们考虑可以实现根据用户得分增加而增加难度，让蛇的移动速度变得越来越快，经过测试，我们设置每吃4个食物，蛇的自动移动阈值就减少1。

```

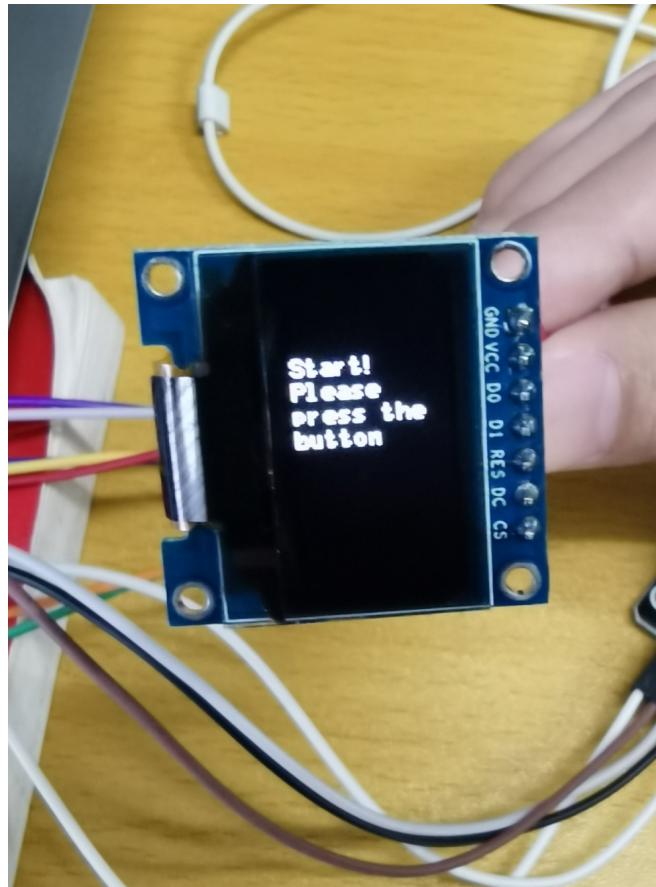
if ((*gamescore)%4==0) speed_max--;

```

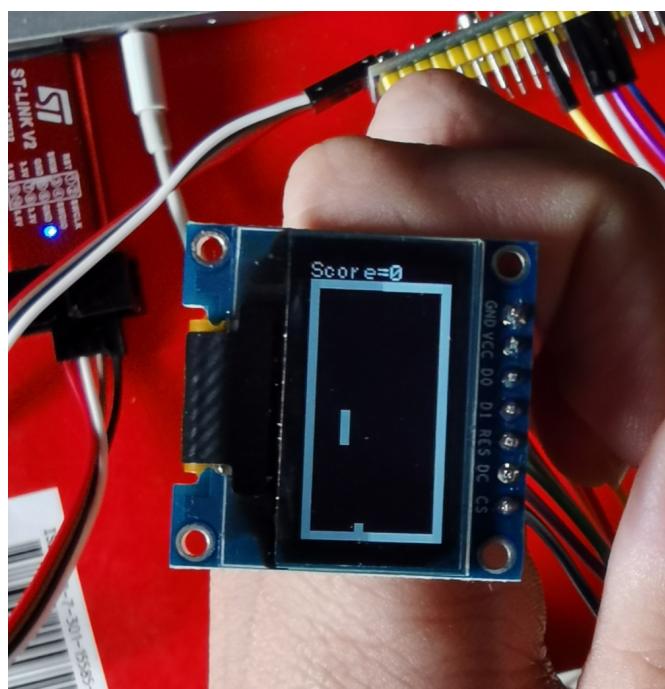
- 此外，我们还增加了一些新的游戏功能，如吃到特殊奖励食物，可以得分更多，或者可以在短时间内减慢蛇的自动移动速度，或者吃到有害食物，就会导致蛇的自动移动速度短时间内变快，或者游戏结束。我们之后也会进一步考虑如何增加游戏趣味性，以达到更好的用户体验。

实现效果

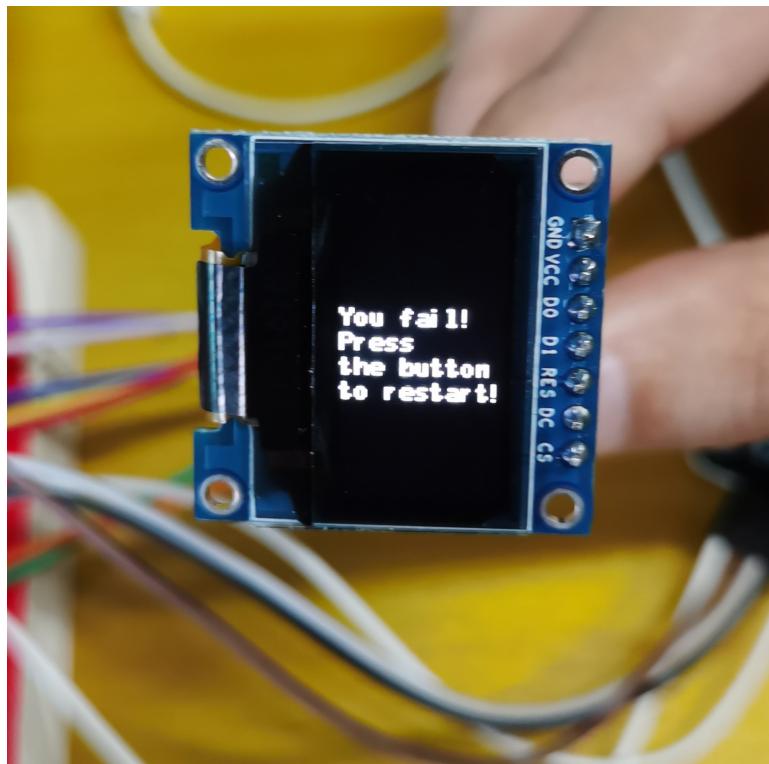
初始游戏界面



游戏进行界面



游戏结束界面 (按下摇杆实现restart)



总结
