**TFT显示贪吃蛇**

第一部分 硬件电路连接

|  |  |  |  |
| --- | --- | --- | --- |
| TFT彩屏引脚名称 | 对应功能描述 | 对应单片机引脚 | 与STM32的连接 |
| GND | 接地引脚 |  | 接GND |
| VDD | 屏幕供电引脚 |  | 接VCC |
| LED-A | 背光+ |  | 接VCC |
| LED-K | 背光- |  | 接GND |
|  |  |  |  |
| X+X-Y+Y- | 触摸控制 |  | 悬空 |
| DB0-DB7 | 低八位数据总线 |  | 悬空 |
| NC | 无用引脚 |  | 悬空 |
| T-xxx | 触摸相关信号线 |  | 悬空 |
|  |  |  |  |
| DB8-DB15 | 高八位数据总线 |  | PA0-7 |
| DCX | 选择数据或寄存器 | RS 1 | PB0 (CN5\_3) |
| RDX | 并口读（不用时拉高） | RD 2 | PB1 (CN8\_4) |
| WRX | 并口写（不用时拉高） | WR 3 | PB2 (CN10\_22) |
| RESX | 复位（低电平有效） | RST 4 | PB3 (CN9\_4) |
| CSX | 片选（低电平有效） | CS 5 | PB4 (CN9\_6) |

第二部分 两种显示模式的构建

1. TFT彩屏原有显示模式分析

TFT显示模式与LCM12864不同。

LCM12864自带两种显示模式。LCM12864在字符显示模式下LCM12864可以调用其内部的字符库进行显示。LCM1284在图像显示模式下，其内部自带显存，只需要修改其显存对应位（0与1）即可改变该像素点的亮灭状态（单色显示）。其显存为16位存储模式，即横向的16个像素点存储在一个16位数据中，改变其中一个需要读取该组所有像素点信息。

TFT彩屏显示其只有一种显示模式，且不带显存，配合控制总线时序逻辑通过8位数据总线对其单个像素点发送数据即可改变其颜色（单个彩色显示，不同于LCM12864的分组单色显示），其对像素的赋值不需要指定地址，而是扫描模式，对上一个像素赋值完毕则自动跳到下一个像素。

1. 屏幕设计问题分析

首先TFT分辨率要高很多，每个像素点都特别小，不能再用一个像素代表贪吃蛇的一格。所以，在显示贪吃蛇图像时需要对屏幕进行分块，每一块显示一格贪吃蛇。

其次，贪吃蛇行进过程中并不是每个点都在变化，如果每次刷新都要重新计算显示之后再向TFT发送数据，无疑对MCU资源造成了很大浪费，其刷新图像的速率（帧率）就特别低下，程序运行效率相当低。本设计中的贪吃蛇算法用到，去某个位置改变对应像素值的函数，以此高效率实现贪吃蛇平移，但是TFT原有的扫描显示明显不支持此种操作。所以，构建一个图像显存，贪吃蛇运行逻辑改变图像显存的值即可，输出时根据显存的数据输出到屏幕。

最后，TFT内部无字符库。所以，在游戏开始之前输出提示性语句无法完成，所以需要自己构建字符库。字符库构建完成，如何告诉MCU在哪个位置显示哪个字符呢？借用上述思路，搭建字符模式的显存，同样在主函数中更改字符模式显存，之后调用字符模式显示函数即可在对应位置显示对应字符。

对于屏幕显示的设计总结如下：字符库、图像模式显存与字符模式显存、图像模式显示函数与字符模式显示函数、更改图像显存与更改字符显存、显存清除等。

1. 屏幕设计思路

屏幕划分：原屏幕分辨率240\*320，划分为20\*20块（块坐标如下图左），每一块分辨率12\*16（像素点坐标如下图右）。在代码中，“H”与“L”表示块坐标，“h”与“l”表示像素坐标。

15行0列

15行11列

19行19列

19行0列

0行19列

0行0列

0行11列

0行0列

图像模式显存：对于贪吃蛇来说，该模式不需要显示很精细的图像，所以定义数组TCSK[20][20]来存储20行20列的图像模式显存。该数组中存储不同的整型数据，每个不同的数值代表对应的每个16\*12像素显示区域不同的颜色。

图像模式显示函数display\_TCS：先发送相关命令设置起始像素位置等进入显示模式，用“H”、“L”、“h”和”l”四层for循环实现显示屏扫描，如果图像模式显存TCSK[H][L]为0则write\_data(0x00,0x00)显示黑色，否则write\_data(0x00+TCSK[H][L]\*20,0x00+TCSK[H][L]\*20)即根据显存存储的数据改变该像素的值。

主函数中改变图像模式显存：此逻辑设计简单，只需要将原来去某个位置（x，y）改变对应显示的函数，替换为给显存TCSK[x][y]赋值即可。

字符库创建：同样用上述屏幕划分原理，所以每个字符块的大小同样为12\*16像素（16行12列）。字符库ZFK[27][16][12]中对27个字符（包含26个英文字母大写与一个空格建模），字符只有单色显示模式，故其内部存储的值只有两种0灭1亮。

字符模式显存：所以定义数组WZK [20][20]来存储20行20列的字符模式显存。该数组中存储不同的整型数据（范围0-26），每个不同的数值代表对应的每个12\*16像素显示区域显示不同的字符。

字符模式显示函数display\_ZF：同样先发送相关命令设置起始像素位置等进入显示模式，用“H”、“L”、“h”和”l”四层for循环实现显示屏扫描，如果字符模式显存WZK[H][L]不满足小于27的要求则显示黑色，否则利用字符库ZFK[WZK[H][L]][h][l]判断该像素点是否需要亮起，该对应字符库为1则write\_data(0x00,0xFF)亮起，否则write\_data(0x00,0x00)显示黑色。

主函数中改变字符模式显存：此逻辑设计较为复杂，思路如下：首先便于用户操作，必然是直接写入一个连续的字符串，比如strcpy(tstr,"DOUBLE COOPERATION")这就将需要显示的字符串写入了字符串类型的缓存数组char tstr[20]（该数组长20，和屏幕每行20列的划分模式一一对应）。其次需要用构建函数printff(int hang,char tstr[])，用户通过此函数传入的参数hang实现定位到某一行，利用for循环（20次即一行）根据缓存字符类型数组tstr对应位置的字符显存赋值整型数据（例如“hang==8”，“tstr[5]==A”则WZK[8][5]赋值为1），以配合后续显示工作。

显存清除函数的设计最容易，只需用for循环将图像显存二维数组或字符显存二维数组清零即可。

1. 游戏逻辑设计

（1）数据存储：二维数组“flg[4][2]”存储蛇头位置变化（4个方向，2个坐标）；两个变量“m/n”存储可视化窗口大小；变量“tim”存储游戏执行延时（即游戏速度）；用结构体“tan”存储每条蛇每个坐标点的位置，结构体“food”记录两个食物的坐标；“tot”记录蛇的长度；“f”记录当前运行状态；食物存在点矩阵“vis”；记录得分、食物个数、背景颜色、游戏模式等。

（2）随机数的产生：在控制台用C语言实现贪吃蛇时，涉及随机投喂的问题，电脑内没有集成随机数模块，产生不出随机数，只能通过代码使其产生伪码，故单独应用rand函数，产生的数字都是相同的。那么怎么获取一个随机数呢？不停向前的时间可以帮助我们解决此问题，用GetTickCount函数来获取自系统启动后的毫秒数，之后用srand函数种下随机数种子，之后再调用rand函数即可产生伪码。每次GetTickCount返回的时间值不同，用Rand函数得到的伪码就不同。

（3）模式选择模块：输出提示信息，根据按键改变全局变量“ms”的值来选择模式。

（4）边框显示模块：在电脑中一个方块字符“▇”占用一行两列，所以需要根据显示窗口大小计算每个边框点的坐标值。因边框固定，所以初始化的时候显示一下，之后就不需要再刷新了。

（5）小蛇数据初始化：根据对应模式，选择玩家一、玩家二、机器人一以及机器人二中的一条或者两条小蛇进行初始化（颜色与位置）。

（6）食物的随机投喂：判断如果其中一个食物不见了，那么生成两个随机数，根据窗口大小取余运算之后得到X和Y坐标，再检测投喂点是否为空，如果不是则生成新坐标，如果是则去对应位置投放食物，并更新与食物相关的数据。

（7）没有按键按下时小蛇自动运行：根据不同的小蛇设置显示颜色，如果是机器人则先判断按下一步前进方向自动寻路，如果是玩家则保持当前运行状态。根据此运行状态更新蛇头的位置。之后用for循环（蛇的长度为循环次数），更新蛇身每个坐标点的数据，最后擦除蛇尾。如果在此过程中，小蛇有吃掉一个食物则在尾巴处显示方块字符“▇”并更新蛇的长度与食物数据并判断是否得分足够胜利。判断：如果一方撞到蛇的身体或者墙壁则游戏结束。都没有撞到的话，显示蛇头（蛇头颜色与身子不同便于识别，故需要单独显示）。

（8）退出：GetForegroundWindow函数获取用户正在激活的窗口。该函数设置指定窗口的显示状态 hWnd指窗口句柄 SW\_HIDE隐藏窗口（大小不变激活状态不变）。最后调用exit(0)退出。

（9）暂替：巧用循环，while循环中获取输入，如果输入的不是按键“P”，程序将困在此循环中继续等待下次输入。

（10）检查某个点是否为空（随机投喂食物用到）：设置布尔型变量“t”初始值为一，如果该点在边界外将t置为0，如果该点为蛇身体的一个点，也将t置为0，然后返回t的值。函数返回值为1即为空。

（11）机器人寻路模块：先根据蛇头与两个食物横纵坐标之差的绝对值，来判断距离哪个食物最近，然后根据坐标只差的正负确定8个方位，前方有障碍则运行other函数转变方向继续判断，根据上述过程计算出相应运行状态。

（12）游戏结果模块：游戏结束方式分为“一方积分足够”或者“一方撞到障碍物”，依据不同结束方式、游戏模式以及得分显示相应提示内容。

（13）游戏结束之后：如过玩家还想再来一局，则直接调用main主函数。

（14）main主函数内容：种下随机数种子，用系统函数system设置窗口标题以及可视化窗口大小，运行system("cls")函数清除屏幕，清零得分、小蛇以及食物相关数据，模式选择，设置游戏延迟（速度），对应模式小蛇初始化，至此所有初始化未完成，进入while（1）循环。当没有按键按下时，根据模式选择的不同，调用change函数实现状态刷新。如果有按键按下，则获取按键信息，判断是否为快捷功能键“E”退出“P”暂停，都不是则根据按键与当前模式改变玩家一（玩家二）小蛇的运行方向，继续回到无按键按下时的状态刷新。

1. 寻路算法解析

寻路本质就是寻找合适的方向：计算蛇头与两个食物距离（坐标之差绝对值在求和|x1-x2|+|y1-y2|）。如果0号食物比1号食物近或者1号食物消失，则将目标设定为0号食物，否则设定为1号食物。For循环遍历4个方向，首先用if确保此方向不是朝向身后（剩下3个方向），其次用if确保蛇头按照此方向前进一格之后不是障碍物，如果都满足则计算蛇头按照此方向前进一格之后和目标食物的距离，如果此距离小于之前计算出的最小距离，则将此方向更改为新的前进方向，并更新最小距离。

该循环结束后，将会选择出距离目标食物距离最小的方向，并确保了不向后，且不主动撞障碍物。当蛇头3其他三个方向都是障碍物，进入死胡同的之后，遍历完四个方向则不会改变原来的前进方向，被迫无奈撞墙死亡（不会造成死循环）。

1. 优化

时序上的问题：外设带给他的延时很大，即便没有单独加延时，每一次调用显示输出函数都占用好多时间资源，造成程序整体运行具有很大延时。所以需要尽可能减少输出函数的调用，故不是每次更改显存都输出显示，而是仅在以下时刻输出：

复位后显示提示性字符后调用字符模式输出。

中断来来临对应模式初始化后暂不调用输出，因为初始化之后将立即进入while循环之中开启游戏模式调用图像模式输出。在投喂、无按键按下的自动运行不调用输出，因为游戏模式循环一次说明一个操作周期过去了，之前已经实现一个操作周期刷新一次，故不需要在投喂、无按键按下的自动运行调用输出。

游戏结束之后，调用字符模式输出显示游戏结果，调用字符模式输出。