

课 程 设 计 报 告

课程名称： 微机系统与接口技术

班 级：计算机科学与技术183，182

学 号： 201820030124，201808140223

姓 名： 盛洪涛，韩文康

**2021年1月**

**目录**

**1.总体介绍------------------------------------------------------3**

**2.大体功能模块------------------------------------------------4**

**2.1地图绘制-------------------------------------------------4**

**2.2基于键盘中断处理程序的小车水平移动版块----7**

**2.3碰撞检测------------------------------------------------11**

**2.4胜利判定------------------------------------------------12**

**2.5结果画面------------------------------------------------13**

**3.运用到的所学知识-----------------------------------------14**

**4.流程图--------------------------------------------------------15**

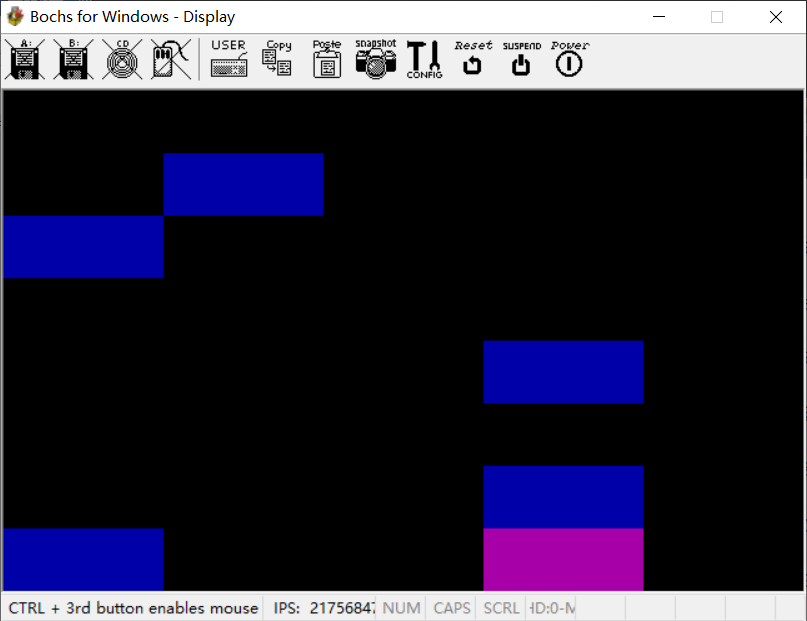
**5.心得体会-----------------------------------------------------16**

**6.程序代码-----------------------------------------------------16**

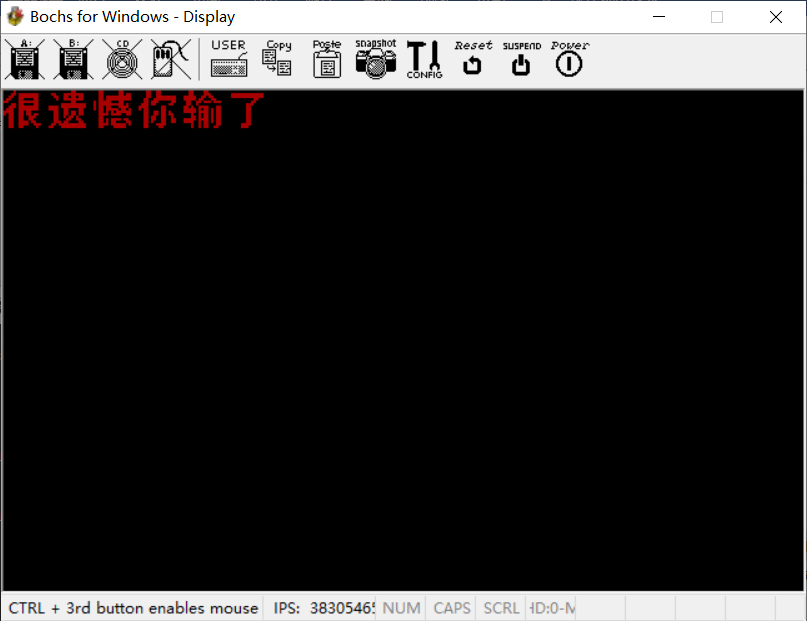
**一.程序总体介绍**

一个赛车躲避游戏，蓝色的矩阵代表障碍物，红色的矩阵代表赛车，每过一段时间矩阵都会往下降落，玩家通过键盘的A，D来左右控制赛车的位置，躲避迎面撞来的障碍物，碰撞检测存在两种，一种是左右移动车子时不能撞到同层的障碍物，另一种时障碍物下落时，车子不能被上层的障碍物撞击到.

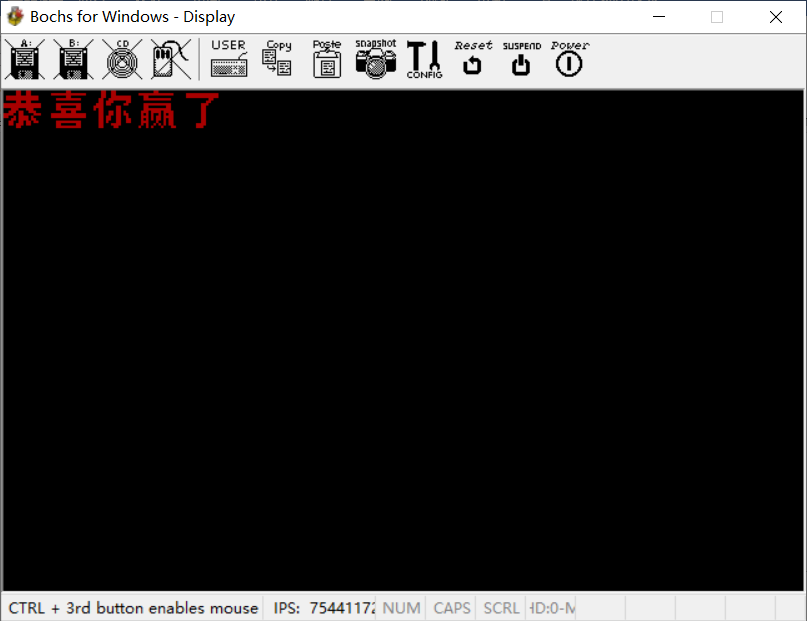
进入游戏的画面



迎面撞上障碍物 则失败



跑过一定长度的赛道后



**二.大体功能模块**

1.地图绘制

地图信息我们采用了固定生成的方式，用一个数组去存放我们地图的信息，虽然数组是一维的，但我们在实际使用中，以5个为一组当作一层的障碍信息，所以使用时是二维的逻辑结构，例如10010，它其实代表着在这一层第1列和第4列存在障碍物，我们的显示地图尺寸是7行5列，我们采用了双重循环来输出这个矩阵地图，

map\_a dw 0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,1

Show\_barrier:

mov si,map\_a

mov bx,[ds:iter]

mov cx,7;地图是7行5列双循环生成

FOR1:

mov ax,cx

mov dx,320

imul ax,dx

imul ax,25

mov [ds:x\_place],ax;把横坐标的位置换算成像素后存在xplace里

mov dx,0

FOR2: mov ax,[si+bx]

cmp ax,0

je GOTO

mov [ds:y\_place],dx;把纵坐标的列数放在yplace里

push ax

push bx

push cx

push dx

call Tangle;画矩阵

pop dx

pop cx

pop bx

pop ax

GOTO: add bx,2

add dx,1

cmp dx,5;五次循环

jnge FOR2

sub cx,1

cmp cx,1;七次循环

jge FOR1

mov bx,[ds:iter]

add bx,10;把指针的位置往后挪5格,注意此时只是对寄存器做了加法，还没对真正的指针变量进行修改

cmp word[if\_change\_map],0;检测当前是处于需要下降还是不需要下降

je show\_end;0就不要变

mov [ds:iter],bx;修改指针变量

show\_end:

ret

具体到如何绘制，我们把障碍物画成一个矩阵，对于矩阵的实现，我们调用了我们自己定义了的画线函数，通过调用多次画线来实现矩阵的生成。

line:

mov ax,0

col:

cmp ax, 64

jae bk

mov byte[es:bx], 1

add ax, 1

add bx, 1

jmp col

bk:

ret

Tangle:

mov bx,[ds:x\_place]

mov ax,64

mov dx,[ds:y\_place]

imul ax,dx

add bx,ax

mov cx,320

imul cx,175

add cx,[ds:position]

cmp cx,bx;冲撞分析

jne Ys

mov word[lose],1;设为失败

Ys: mov dx,0

draww: call line

sub bx,64

add bx,320

add dx,1

cmp dx,25

jl draww

ret

游戏过程中，车会不断前进，相对来看地图会不断下移，每一次下移，我们就把指向最底层的指针往后挪5个位置，以此来实现地图的整体移动。

2.基于键盘中断处理程序的小车水平移动版块

首先在每400\*1000个计算单位调用一次键盘中断处理程序，这里可选择自定义的中断处理程序，在本文代码段中有对应的自定义键盘中断处理程序。

Code：

time\_layback:

mov word[wa],0

here2: add word[wa],1

call time\_layback2

call Horizontal\_movement;每隔400\*1000个单位检测键盘缓冲区的数据做出相应

cmp word[wa],10

jb here2

ret

然后在Horizontal\_movement板块中，首先调用int16的1号功能号，去检测键盘缓冲区，对键盘扫描且不等待，并设置ZF标志。若有按键操作（即键盘缓冲区不空），则ZF＝0，AL中存放的是输入的ASCII码，AH中存放输入字符的扩展码。若无键按下，则标志位ZF＝1。显然如果键盘缓冲区为空，那么就不做小车坐标修改的操作。否则就调用int16的0号功能号，将键盘缓冲队列队头的扫描码送入ah，ASCII码则被送入al，并弹出队头，如此一来我们就可以根据al中存储的ASCII码与对应’a’和’d’的ASCII码进行对比，确定要做的操作（向左或向右移动小车的左上角坐标），当然小车的移动是有限制的，它的左上角纵坐标不得小于0或者大于256，因为这里我们设置了5个车位，也就是每个车位64个像素点。当然该表小车的坐标后就要对图形界面做响应的更新。这里我们写了draw\_car函数。

Horizontal\_movement Code：

Horizontal\_movement:

jmp here

keep\_right:

mov word[position],256

jmp back2

here:

mov ah,0x1;判断缓冲区是否为空

int 0x16

jz event\_end;zf标志寄存器,空就不做操作，否则实时更新 车的位置

mov ah, 0x0;从键盘缓冲区读出

int 0x16

;以下是车的坐标的更新

;int16处理后，ah存放扫描码，al存放ascii码

cmp al,'a';向左移动

jnz right

cmp word[position],0;限制车的位置不得小于0

je event\_end

add word[position],-64

jmp event\_end

right:

cmp al,'d';向右移动

jnz event\_end

add word[position],64

cmp word[position],256;限制车的位置不得大于256

jae keep\_right

back2: jmp event\_end

event\_end:

call draw\_car

ret

在更新图形界面上小车的位置时，首先要做的是清空整个屏幕，然后再画出之前的地图障碍物的位置，然后画出小车，并判断车是否在这次响应中撞上了障碍物，如果撞上就将全局变量lose设为1，那么程序就会在下个时间单位时检测到这个信号，并输出很遗憾失败的信息。

draw\_car Code:

;响应更新车的位置的操作

draw\_car:

call clear\_all;清空屏幕

mov word[if\_change\_map],0;设置成不要改变地图的位置，因为这是在1秒内的操作，而不是第k秒

call Show\_barrier

call get\_pos

mov cx,0

mov dx,0

iiii:

cmp cx,word[car\_height]

je draw\_car\_end

inc cx

add si,256

mov dx,0

jmp jjjj

jjjj:

cmp dx,word[car\_width]

je iiii

mov byte[es:si],5

inc dx

inc si

jmp jjjj

mov bx,[ds:iter]

mov si,map\_a

mov cx,0

jmp here5

;冲撞分析

judge:

mov dx,cx

imul dx,32

cmp word[position],dx

je set\_lose

jmp here5

;设置成错误

set\_lose:

mov word[lose],1

here5:

cmp dx,[si+bx]

cmp dx,1

je judge

add cx,2

add bx,2

cmp cx,8

jbe here5

draw\_car\_end:

ret

3.碰撞检测

1.地图下移时的碰撞检测

该碰撞检测我们直接在下移画障碍物的时候进行实现，因为我们车子和障碍物的初始点位都是一个矩阵的最左上角，因此只要存在障碍物的初始点与车辆相同，则说明两个矩阵重合既发生了碰撞，此时我们置定义的失败变量为1并跳出游戏循环显示结果画面。

cmp cx,bx;冲撞分析

jne Ys

mov word[lose],1;设为失败

cmp word[lose],1;如果发生了冲撞输出失败信息

je put\_lose

2.车辆左右移动时的碰撞检测

与上面原理相似，当我们左右移动车辆后，将车辆的初始点位与地图所有的矩阵循环判断，是否存在相同的情况，存在则发生了碰撞，置失败变量为1，跳出循环，显示结果画面。

;响应更新车的位置的操作

draw\_car:

call clear\_all;清空屏幕

mov word[if\_change\_map],0;设置成不要改变地图的位置，因为这是在1秒内的操作，而不是第k秒

call Show\_barrier

call get\_pos

mov cx,0

mov dx,0

iiii:

cmp cx,word[car\_height]

je draw\_car\_end

inc cx

add si,256

mov dx,0

jmp jjjj

jjjj:

cmp dx,word[car\_width]

je iiii

mov byte[es:si],5

inc dx

inc si

jmp jjjj

mov bx,[ds:iter]

mov si,map\_a

mov cx,0

jmp here5

;冲撞分析

judge:

mov dx,cx

imul dx,32

cmp word[position],dx

je set\_lose

jmp here5

;设置成错误

set\_lose:

mov word[lose],1

4.胜利判定

我们定义了一个游戏进行层数的变量，每一次地图的下移就对该变量+1，当超过我们设置的阈值，则成功通关，跳出游戏循环，显示结果画面。\

for:

add word[now\_level],1

call draw\_all

call time\_layback;限制一段的时间，这段时间内车可以移动，然后每隔固定时间，地图下移

cmp word[lose],1;如果发生了冲撞输出失败信息

je put\_lose

cmp word[now\_level],16;当通过一定长度的地图时就胜利了

je put\_win

5.结果画面

我们采用了点阵的方式打印字体，显示结果。

代码：

put\_lose:

call clear\_all

mov si,mat2

mov bx,0

call get\_ed

mov ax,0

call write\_a\_word

mov bx,18

call get\_ed

mov ax,0

call write\_a\_word

mov bx,36

call get\_ed

mov ax,0

call write\_a\_word

mov bx,54

call get\_ed

mov ax,0

call write\_a\_word

mov bx,72

call get\_ed

mov ax,0

call write\_a\_word

mov bx,90

call get\_ed

mov ax,0

call write\_a\_word

jmp end

put\_win:

call clear\_all

mov si,mat

mov bx,0

call get\_ed

mov ax,0

call write\_a\_word

mov bx,18

call get\_ed

mov ax,0

call write\_a\_word

mov bx,36

call get\_ed

mov ax,0

call write\_a\_word

mov bx,54

call get\_ed

mov ax,0

call write\_a\_word

mov bx,72

call get\_ed

mov ax,0

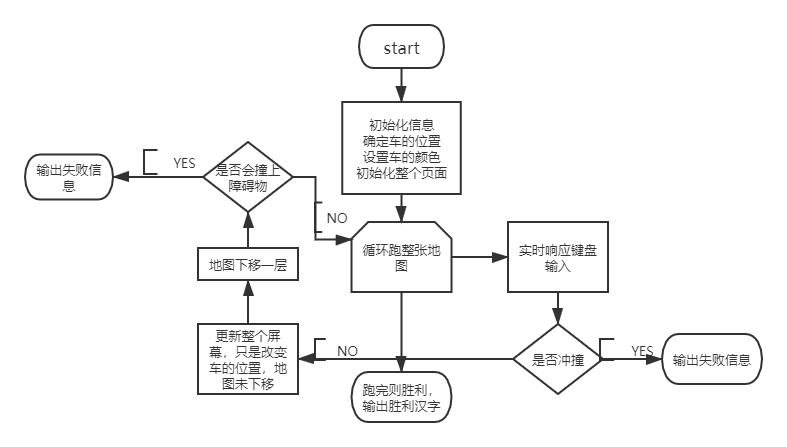
call write\_a\_word

end:

**三：运用到的所学知识**

**我们在具体实现中，运用了颜色的定义，画线及运用画线生成矩阵，采用点阵的方式打印汉字，键盘响应，栈实现寄存器的复用，类时间中断等知识。**

**四.流程图**

****

**五．心得体会**

最早学习C语言的时候，就对程序实现细节颇有好奇，这些代码实现这种操作背后是什么原理，通过整个学期的微机学习，帮助了我们更好的理解了编程的更底层的知识，知识不再是原先漂浮在空中的工具，更加接地气，老师平时上课让我们自由发挥，也大大加强了我们的学习能力和实践能力，在此次课设的进行过程中，这些能力都发挥很很大的作用，每当我们遇到问题的时候，例如寄存器急缺，程序出错，循环失败等等，我们通过自己查阅资料，和不断的中断程序检测结果，找到了问题的解决方法和程序的BUG所在,这一切都要感谢老师的细心教导。

**六.程序代码**

org 0x8400

jmp start

;以下是汉字矩阵存储

;恭喜你赢了

mat db 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0

db 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0

db 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0

db 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0

db 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0

db 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0

db 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0

db 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0

db 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0

db 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0

db 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0

db 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0

db 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0

db 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0

db 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0

db 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0

db 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0

db 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0

db 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0

db 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0

db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0

db 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0

db 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0

db 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0

db 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0

db 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0

db 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0

db 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0

db 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0

db 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0

db 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0

db 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0

db 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0

db 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0

db 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0

db 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0

db 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0

db 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0

db 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0

db 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0

db 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0

db 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0

db 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0

db 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0

db 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0

db 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0

db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

db 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0

db 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

;很遗憾你输了

mat2 db 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0

db 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0

db 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0

db 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0

db 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0

db 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0

db 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0

db 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0

db 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0

db 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0

db 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0

db 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0

db 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0

db 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0

db 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0

db 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0

db 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0

db 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0

db 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0

db 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0

db 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0

db 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0

db 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0

db 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0

db 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0

db 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0

db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0

db 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0

db 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0

db 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0

db 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0

db 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0

db 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0

db 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0

db 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0

db 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0

db 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0

db 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0

db 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0

db 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0

db 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0

db 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0

db 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0

db 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0

db 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0

db 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0

db 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0

db 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0

db 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0

db 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0

db 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0

db 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0

db 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0

db 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0

db 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0

db 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0

db 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

db 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0

db 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0

db 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0

db 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0

db 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0

db 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0

db 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0

db 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0

db 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

db 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0

db 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0

db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

color db 120,200,0;车的颜色

position dw 128;初始化车的左上角的坐标

car\_width dw 64;车的宽度

car\_height dw 25;车的高度

base dw 55744

da db 1,40,0,0

map\_lenth dw 16;地图长度

now\_level dw 0;当前是第几层

map\_a dw 0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,1

;地图数据

iter dw 0;地图指针

x\_place dw 0

y\_place dw 0

lose dw 0;是否输了

if\_change\_map dw 1;标志是否要下移地图

wa dw 0

wb dw 0

wc dw 0

;以下是画地图的版块

line:

mov ax,0

col:

cmp ax, 64

jae bk

mov byte[es:bx], 1

add ax, 1

add bx, 1

jmp col

bk:

ret

Tangle:

mov bx,[ds:x\_place]

mov ax,64

mov dx,[ds:y\_place]

imul ax,dx

add bx,ax

mov cx,320

imul cx,175

add cx,[ds:position]

cmp cx,bx;冲撞分析

jne Ys

mov word[lose],1;设为失败

Ys: mov dx,0

draww: call line

sub bx,64

add bx,320

add dx,1

cmp dx,25

jl draww

ret

Show\_barrier:

mov si,map\_a

mov bx,[ds:iter]

mov cx,7;地图是7行5列双循环生成

FOR1:

mov ax,cx

mov dx,320

imul ax,dx

imul ax,25

mov [ds:x\_place],ax;把横坐标的位置换算成像素后存在xplace里

mov dx,0

FOR2: mov ax,[si+bx]

cmp ax,0

je GOTO

mov [ds:y\_place],dx;把纵坐标的列数放在yplace里

push ax

push bx

push cx

push dx

call Tangle;画矩阵

pop dx

pop cx

pop bx

pop ax

GOTO: add bx,2

add dx,1

cmp dx,5;五次循环

jnge FOR2

sub cx,1

cmp cx,1;七次循环

jge FOR1

mov bx,[ds:iter]

add bx,10;把指针的位置往后挪5格,注意此时只是对寄存器做了加法，还没对真正的指针变量进行修改

cmp word[if\_change\_map],0;检测当前是处于需要下降还是不需要下降

je show\_end;0就不要变

mov [ds:iter],bx;修改指针变量

show\_end:

ret

;主函数

start:

mov ah, 00h

mov al, 13h

int 10h ;图形模式

mov ax, 0xa000

mov es, ax;确定车的位置

mov si,0

mov si, color

mov cl, 3

;根据需要决定是否要自定义键盘中断处理程序

;mov ax,0

;mov ds,ax

;CLI

;mov word[ds:0x24],int\_key

;mov word[ds:0x26],0

;STI

call setting;设置颜色

call draw\_car;初始化下画下车的位置

for:

add word[now\_level],1

call draw\_all

call time\_layback;限制一段的时间，这段时间内车可以移动，然后每隔固定时间，地图下移

cmp word[lose],1;如果发生了冲撞输出失败信息

je put\_lose

cmp word[now\_level],16;当通过一定长度的地图时就胜利了

je put\_win

jmp for

;基于汉字区位码的汉字点阵的输出

put\_lose:

call clear\_all

mov si,mat2

mov bx,0

call get\_ed

mov ax,0

call write\_a\_word

mov bx,18

call get\_ed

mov ax,0

call write\_a\_word

mov bx,36

call get\_ed

mov ax,0

call write\_a\_word

mov bx,54

call get\_ed

mov ax,0

call write\_a\_word

mov bx,72

call get\_ed

mov ax,0

call write\_a\_word

mov bx,90

call get\_ed

mov ax,0

call write\_a\_word

jmp end

put\_win:

call clear\_all

mov si,mat

mov bx,0

call get\_ed

mov ax,0

call write\_a\_word

mov bx,18

call get\_ed

mov ax,0

call write\_a\_word

mov bx,36

call get\_ed

mov ax,0

call write\_a\_word

mov bx,54

call get\_ed

mov ax,0

call write\_a\_word

mov bx,72

call get\_ed

mov ax,0

call write\_a\_word

end:

jmp $

;根据需要确定是否需要自定义键盘中断处理程序

int\_key:

jmp here10

keep\_rightt:

mov word[position],256

jmp exit

here10:

mov dx,0x60

in al,dx

cmp al,0x1e

jnz tmpd

;对a的响应

cmp word[position],0

je exit

add word[position],-64

jmp exit

tmpd:

cmp al,0x20

;对d的响应

jnz exit

add word[position],64

cmp word[position],256

jae keep\_rightt

jnz exit

exit:

mov dx, 0x20

mov al, 0x61

out dx, al

call draw\_car

iret

get\_ed:

mov dx,bx

add dx,4815

ret

;跑一遍点阵

write\_a\_word:

jmp here6

neline:

mov ax,0

add bx,304

jmp back6

write\_a\_dot:

mov byte [es:bx],4

jmp back5

here6:

cmp byte[ds:si],1

je write\_a\_dot

back5:

inc si

inc bx

inc ax

cmp ax,16

je neline

back6:

cmp bx,dx

jbe here6

ret

;自定义方便debug的函数

debug:

mov cx,0

mov dx,0

mov si,100

ii:

cmp cx,word[car\_height]

je debug\_end;

inc cx

add si,256

mov dx,0

jmp jj

jj:

cmp dx,word[car\_width]

je ii

mov byte[es:si],5

inc dx

inc si

jmp jj

debug\_end:

ret

;根据int16中断程序的0号和1号功能区完成键盘输入响应

Horizontal\_movement:

jmp here

keep\_right:

mov word[position],256

jmp back2

here:

mov ah,0x1;判断缓冲区是否为空

int 0x16

jz event\_end;zf标志寄存器,空就不做操作，否则实时更新 车的位置

mov ah, 0x0;从键盘缓冲区读出

int 0x16

;以下是车的坐标的更新

;int16处理后，ah存放扫描码，al存放ascii码

cmp al,'a';向左移动

jnz right

cmp word[position],0;限制车的位置不得小于0

je event\_end

add word[position],-64

jmp event\_end

right:

cmp al,'d';向右移动

jnz event\_end

add word[position],64

cmp word[position],256;限制车的位置不得大于256

jae keep\_right

back2: jmp event\_end

event\_end:

call draw\_car

ret

;通过当前车的左上角的相对坐标找到它的真实坐标

get\_pos:

mov si,word[base]

add si,word[position]

ret

;用黑色点覆盖整个屏幕

clear\_all:

mov si,0

mov dx,64000

clear\_dot:

mov word[es:si],0

inc si

cmp si,dx

jb clear\_dot

ret

;响应更新车的位置的操作

draw\_car:

call clear\_all;清空屏幕

mov word[if\_change\_map],0;设置成不要改变地图的位置，因为这是在1秒内的操作，而不是第k秒

call Show\_barrier

call get\_pos

mov cx,0

mov dx,0

iiii:

cmp cx,word[car\_height]

je draw\_car\_end

inc cx

add si,256

mov dx,0

jmp jjjj

jjjj:

cmp dx,word[car\_width]

je iiii

mov byte[es:si],5

inc dx

inc si

jmp jjjj

mov bx,[ds:iter]

mov si,map\_a

mov cx,0

jmp here5

;冲撞分析

judge:

mov dx,cx

imul dx,32

cmp word[position],dx

je set\_lose

jmp here5

;设置成错误

set\_lose:

mov word[lose],1

here5:

cmp dx,[si+bx]

cmp dx,1

je judge

add cx,2

add bx,2

cmp cx,8

jbe here5

draw\_car\_end:

ret

;在这个版块中使地图下降

draw\_all:

call clear\_all

mov word[if\_change\_map],1;标志地图下降

call Show\_barrier

call get\_pos

mov cx,0

mov dx,0

iii:

cmp cx,word[car\_height]

je draw\_all\_end

inc cx

add si,256

mov dx,0

jmp jjj

jjj:

cmp dx,word[car\_width]

je iii

mov byte[es:si],5

inc dx

inc si

jmp jjj

draw\_all\_end:

ret

setting:

mov dx, 0x3c8

mov al, cl

out dx, al

mov dx, 0x3c9

mov al, [ds:si]

out dx, al

mov al, [ds:si+1]

out dx, al

mov al, [ds:si+2]

out dx, al

ret

;分时间2

time\_layback3:

mov word[wc],0

here4: add word[wc],1

cmp word[wc],1000

jb here4

ret

;分时间1

time\_layback2:

mov word[wb],0

here3: add word[wb],1

call time\_layback3

cmp word[wb],400

jb here3

ret

time\_layback:

mov word[wa],0

here2: add word[wa],1

call time\_layback2

call Horizontal\_movement;每隔400\*1000个单位检测键盘缓冲区的数据做出相应

cmp word[wa],10

jb here2

ret