赵明泽 2151974

期末项目

目录

项目名称

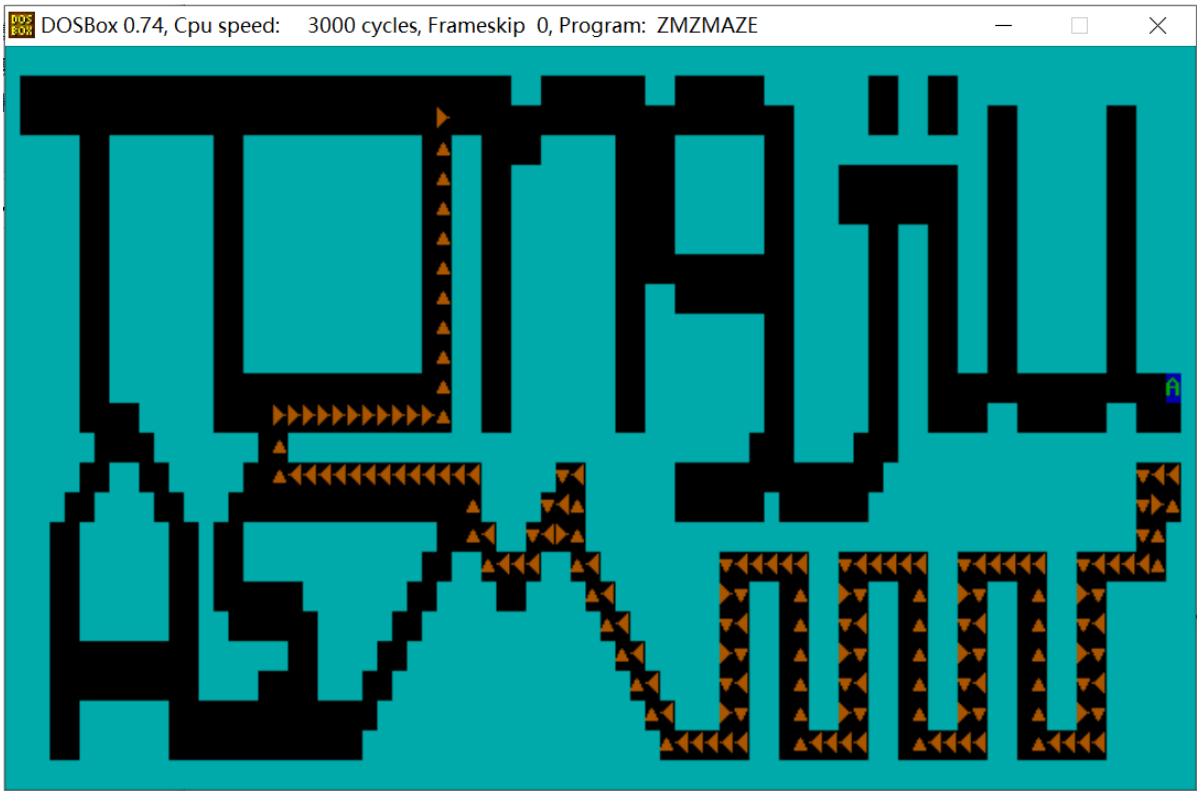
迷宫游戏

项目简介

这是一款结合两种游玩方式的走迷宫小游戏。

该项目在Windows10系统环境下开发，使用Visual Studio Code对代码进行编辑，使用MASM对汇编代码进行汇编、链接与执行。

程序运行截图，如下：



项目功能介绍

该程序分两种模式：1. 自动寻路模式；2. 手动输入模式。玩家进入程序时，需要先选择哪种模式，随后跳转到相应页面。其中，页面右侧偏下是起点，右侧偏上是终点（蓝绿色的A）。玩家需要寻找到合适的路径，从起点到达终点。

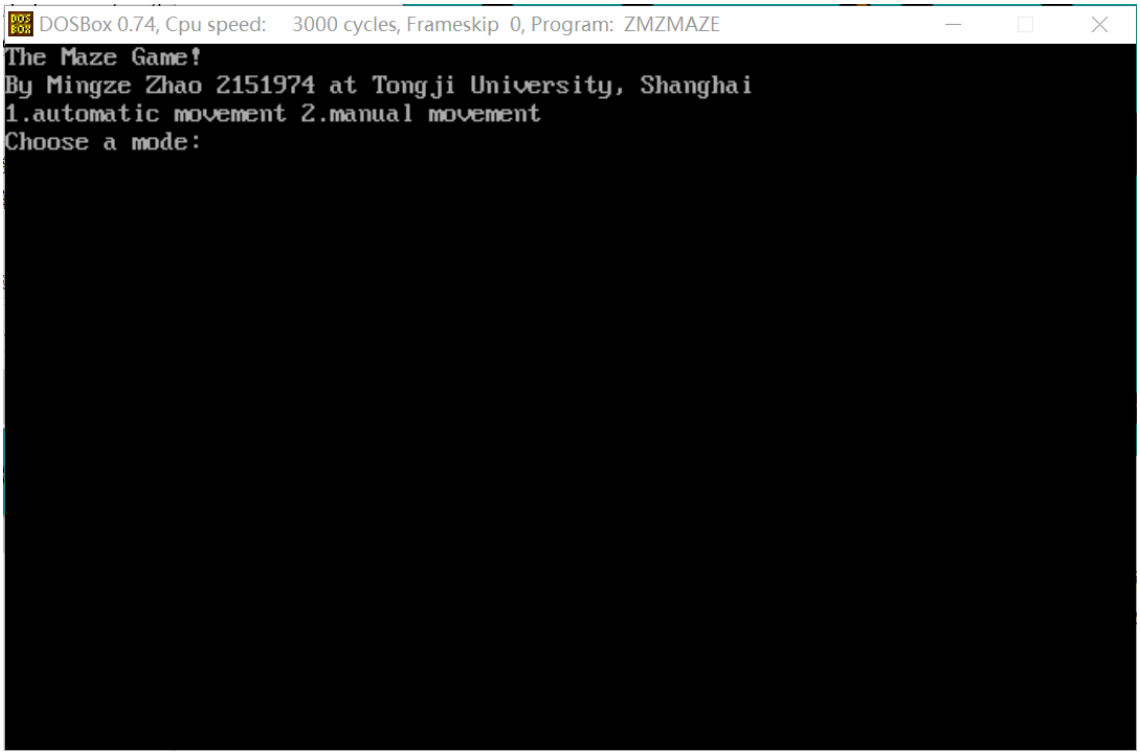
1. 模式选择

   打开程序，首先会看到如下图所示页面。此时玩家需要输入数字1或2来选择模式，选择1即为自动寻路模式，选择2即为手动操作模式。

   程序首先需要调用清屏，相关代码如下。这段代码的意思是`int 10`是8086系统中的中断向量类型码，而不是汇编BIOS中的类型码，而10~1F是给BIOS使用的，10号对应的就是屏幕显示I/O。

   ```
   clear:
       mov  al, 3h
       mov  ah, 0h
       int  10h
       ret
   ```

   用户输入一个数字进行选择，系统会根据数字判断，并跳转对应页面。

2. 自动寻路

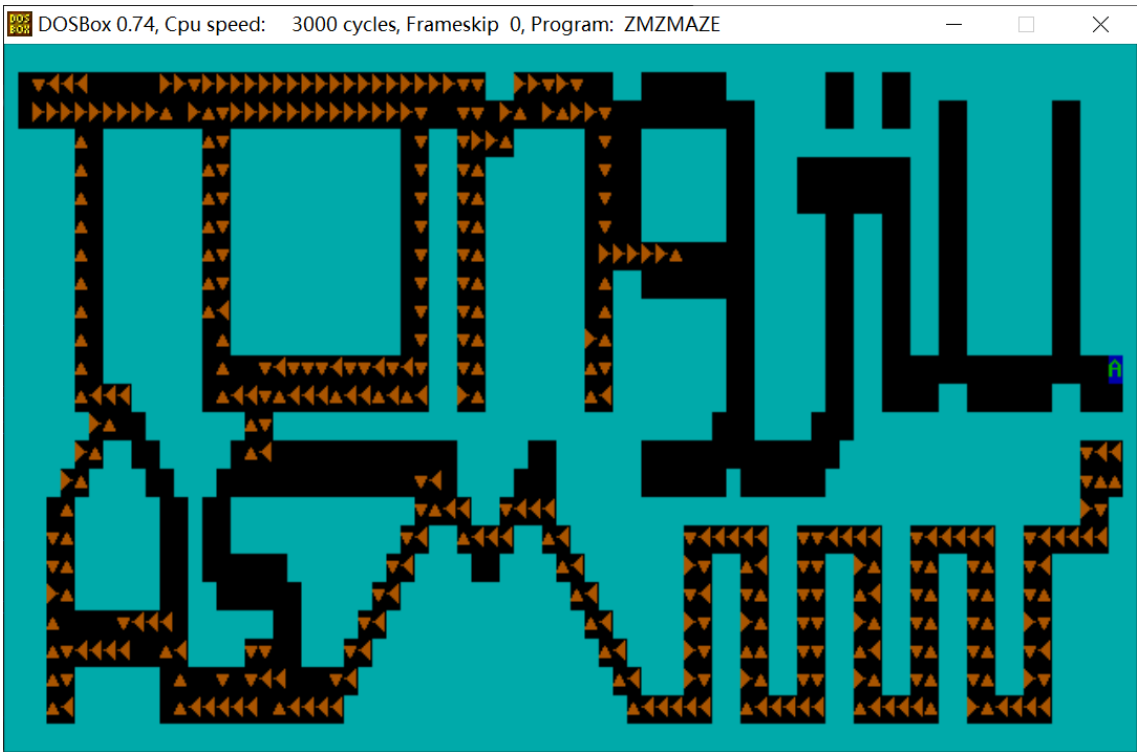
   用户选择1，进入自动寻路模式。自动寻路参考深度优先搜索（Depth-First Search，DFS）策略，其是十分常见的图搜索方法之一。深度优先搜索会沿着一条路径一直搜索下去，在无法搜索时，回退到刚刚访问过的节点。深度优先遍历按照深度优先搜索的方式对图进行遍历。并且每个节点只能访问一次。

   通过在数据段中定义一个图的数据结构，根据标号判断图访问路径，添加一些启发式策略，从而让图标能够适应各种不同的迷宫路径，达到自动寻路的效果。

3. 手动操作

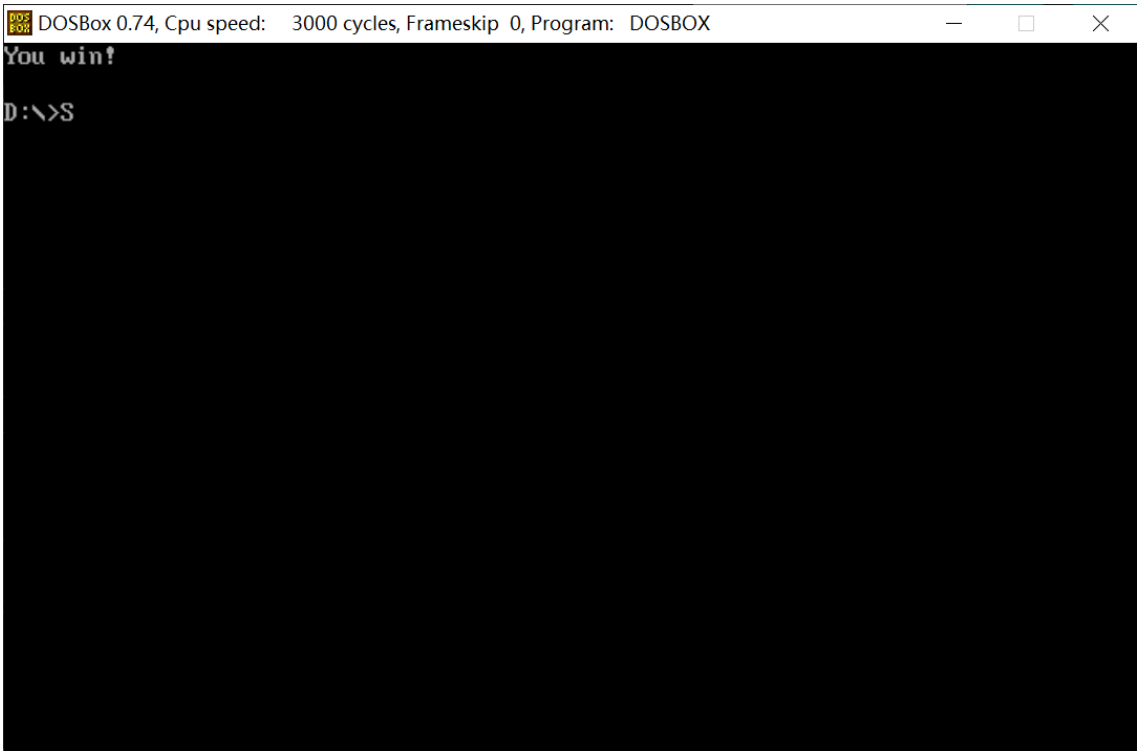
   用户选择2，进入手动操作模式。用户通过键盘输入`w`、`a`、`s`、`d`的方式，控制光标分别向上、左、下、右方向移动，留下移动的轨迹，如下图所示：

   当用户成功将光标移动到终点时，意味着该关卡已经成功，届时界面会进行跳转。



4. 成功页面

   成功到达终点后，页面会再次清空，显示`You win!`，如图所示：



项目开发流程

使用增量模型开发，项目初期设立目标，进行架构设计和系统分析，预计将要开发两种功能，一种是手动一种是自动。首先开始的是手动模式开发。相较于自动模式，手动模式的操控更加简单，部分基础的行走逻辑可以在后面自动模式复用。

数据结构方面，选用的是`byte`字节型大小的图结构，对于通路，标号为0；对于墙壁，标号为1；对于第一次走的路，标号为2；对于退回去的路，标号为3；对于终点，标号为4。

提示语单独放到一个段里，其中还有一些诸如选项之类的变量。

首先，把段的起始地址赋值到段基址寄存器。8086的显存是从`0b800h`开始的，8086的RAM一共有2^20byte，也就是0xFFFFFbyte。其中，0x0~0xA0000：作为常规内存；0xA0000~0xEFFFF：提供给外围设备；0xF0000~ 0xFFFFF：提供给ROM-BIOS芯片。提供给外围设备的0xA0000~0xEFFFF，其中0xB8000 ~0xBFFFF用作显存。

```
mov  ax, 0b800h
mov  es, ax
```

接下来是打印提示语，选择模式。LEA是微机8086/8088系列的一条指令，取自英语***Load effect address——取有效地址***，也就是取偏移地址。在微机8086/8088中有20位物理地址，由16位段基址向左偏移4位再与偏移地址之和得到。通过LEA指令把提示语的偏移地址放入`DX`寄存器，再通过DOS的21h号中断输出一条字符串。

```
lea  dx, msg1   ;打印提示语
mov  ah, 09h    ;将寄存器AH设置为09h，这是DOS的功能号，表示要执行字符串输出
int  21h        ;触发DOS中断21h，以执行字符串输出操作，输出提示语
```

接下来打印迷宫。显存有0xB8000 ~0xBFFFFbit，一共有32KB(8页)，一页的内容占据4KB，一个字符用ASCII码表示，占用一个字节，这个字节还需要一些属性，也需要占用一个字节。4KB / 2 = 2048，所以一页最多能够显示2048个字符。本项目实践中的图型数据结构共有26行（多了一行没删）80列，和显存的数值有些出入。

```
mov  word ptr es:[bx+di], 3020h
```

接下来是对起点、终点的设置和选择的判断。

首先说手动挡。一开始玩家需要输入一个字母来表示前进方向，这里要调用无回显方式的读键盘中断，对应程序如下：

```
mov  ah, 7          ;无回显方式读键
int  21h
```

对于读到对应的键盘字母，要分别跳转到上、下、左、右的可行性判断，因为如果上面是墙，而玩家偏偏选择向上走的话，就不能够把光标向上挪动；而如果是正常通路，光标又要随着字母对应方向移动。下面截取了向上走的可行性判断，程序如下：

```
;判断向上走可行性
shang:
    cmp  byte ptr ds:[bp-80], 1 ;墙
    je   manual_end             ;不处理
    mov  byte ptr ds:[bp], 2    ;走过的地方设置为2
    call shang_go               ;向上走
    jmp  manual_end             ;走完结束
```

这里注意是通过`ds`寻址还是通过`es`寻址，一个是对应一字节的字节型图，一个是对应两字节的字型显存。

接下来要跳转到处理显存的内容，使得在屏幕上显示出移动的轨迹，还要在这里判断一下是不是落进了终点。这一段的程序如下：

```
;处理显存
;右箭头=16;左箭头=17;上箭头=30;下箭头=31;
;dh+dl=当前位置
shang_go:
    mov  word ptr es:[di], 061eh
    dec  dh
    sub  bp, 80
    sub  di, 160
    call judge      ;判赢
    ret
```

然后说一下自动挡。这一块也比较简单，不需要对读取进行判断，但是需要对上、下、左、右四个格子的类型进行判断，大致判断的原理是：第一次按照右、上、左、下的顺序，对当前格子四周的格子进行检查，如果找到一条通路（即标号是0），就进入；如果没找到，进入第二次搜索，依然按照右、上、左、下的顺序，只不过这次检查的是走过一次的道路（即标号是2），如果找到退路，则原路返回，第二次走过的路标记为3；如果没找到，说明走进了死路，直接退出。以下是部分逻辑的摘取：

```
down:           ;下边有空地
    cmp  byte ptr ds:[bp+80], 0
    jne  right2
    mov  byte ptr ds:[bp], 2
    call xia_go
    jmp  auto_end
right2:         ;右边走过一次，可以回退
    cmp  byte ptr ds:[bp+1], 2
    jne  up2
    mov  byte ptr ds:[bp], 3    ;又走一次，标记为3
    call you_go
    jmp  auto_end
```

为了看的清楚，特地加了一个延时函数，如下：

```
;延时函数
delay_time:
    push cx
    mov  cx, 00ffh
dn1:
    push cx
    mov  cx, 00ffh
dn2:
    loop dn2
    pop  cx
    loop dn1
    pop  cx
    ret
```

判断赢了，即光标进入了终点，就跳转到赢了界面，清空屏幕并打印提示语。

```
;赢了
win:
    call clear
    mov  ax, var
    mov  ds, ax
    lea  dx, msg6
    mov  ah, 09h     ;将寄存器AH设置为09h，这是DOS的功能号，表示要执行字符串输出
    int  21h         ;触发DOS中断21h，以执行字符串输出操作，输出提示语
    jmp  finish
    ret
```

心得

通过这次汇编大项目的实践，对汇编有了更加深刻的认识，也深深体会到了汇编作为一种底层的语言，其速度之迅捷，功能之强大，可扩展性之强。如果能过将汇编与算法结合，或者在当今机器学习十分火热的背景下，作为硬件的编程语言提高其性能，也许可以帮助人类加速迈入AGI时代。

参考文献

1. https://blog.csdn.net/dc12499574/article/details/124064761
2. https://blog.csdn.net/tingyu_521/article/details/84600730
3. https://blog.csdn.net/sinat_42483341/article/details/88627341
4. https://blog.csdn.net/Backlight__/article/details/122092960
5. https://blog.csdn.net/Rong_Toa/article/details/88727741
6. https://blog.csdn.net/lilien1010/article/details/7481145
7. https://blog.csdn.net/ZipingPan/article/details/100678109
8. https://blog.csdn.net/rizero/article/details/107136574
9. https://blog.csdn.net/qq_37232329/article/details/79895350
10. https://blog.csdn.net/qq_37232329/article/details/79922994
11. https://blog.csdn.net/m0_59287327/article/details/128057288
12. https://blog.csdn.net/flq18210105507/article/details/127566006
13. https://blog.csdn.net/u012244479/article/details/129639316
14. 计算机语言_x86汇编语言：从实模式到保护模式（操作系统引导课）_原书作者李忠制作_少量字幕