北京理工大學

本科生课程设计

汇编语言与接口技术组队实验报告——PACMAN

学　　院：	计算机学院
专　　业：	计算机科学与技术
学生姓名：	张驰、魏慧聪、曹哲瑀、王浚哲
学　　号：	1120191600、1120191866、1120191407、1120191197
指导教师：	李元章

2022 年 6 月 22 日

目 录

第 1 章 实验目的

本次实验通过使用ASM汇编语言编写PACMAN游戏，并在基础PACMAN游戏中加入一些创新点，从而掌握Windows系统汇编程序的基本结构，掌握基本的汇编指令。通过本次实验，使队伍更加熟练进行汇编程序的编写和调试，学会调用C库函数，提高汇编语言编程能力，加强对汇编指令的理解。

第 2 章 实验环境

本次实验在Windows10系统下进行，使用MASM,以Visual Studio 2019作为开发的IDE。引入了Irvine32库，该库在本此实验中的主要作用是变换颜色、清空窗口和在特定位置输出。

第 3 章 PACMAN 游戏与程序简介

3.1 PACMAN 游戏简介

吃豆人是电子游戏历史上的经典街机游戏，由Namco公司的岩谷彻设计并由Midway Games在1980年发行。PacMan被认为是80年代最经典的街机游戏之一。Pacman创造了第一个活生生的游戏角色，也是第一个引入了AI的游戏。2020年是Pacman诞生的40周年，这一年有诸多关于Pacman的有趣故事。到现在，依旧可以在各种电影中看到Pacman的身影，Pacman的形象已经成为了一种大众文化符号。游戏中有四个幽灵，分别是Blinky、Pinky、Inky和Clyde。最初的吃豆人游戏中有三个模式，分别是追逐模式，分散模式和惊吓模式。

本次组队实验编写的游戏采用了初代吃豆人的经典地图，在实现了追逐模式的基础上将分散模式融入道具，并加入了其他功能道具。

3.2 游戏规则

游戏界面如下图所示：

图 3-1 游戏开始界面展示

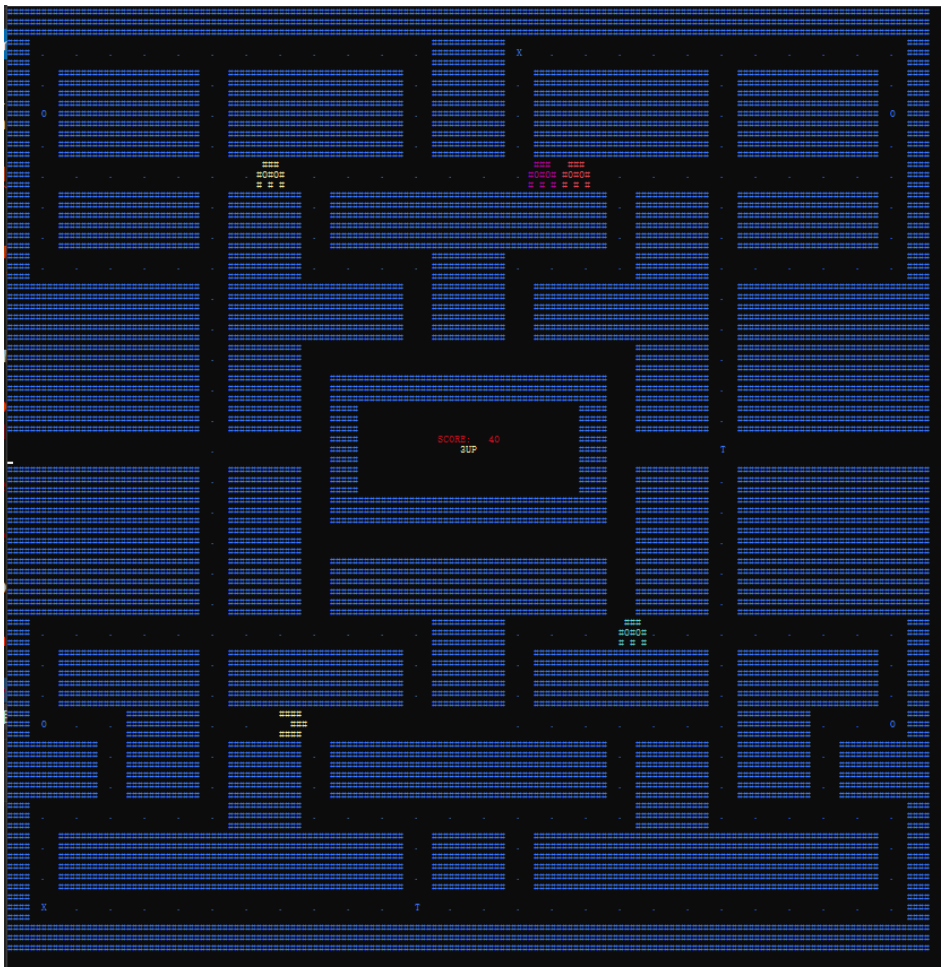

图 3-2 游戏主体界面展示

使用W、A、S、D控制吃豆人吃掉游戏迷宫中所有的豆子，且在路径中尽可能避免被幽灵吃掉。

在所有生命消耗完毕前，吃掉场上的所有豆子，即获得胜利。生命值消耗殆尽，游戏结束。

游戏中包含以下道具：

道具O：该道具干扰鬼的逻辑，在一定时间内幽灵会尽可能避免追逐PACMAN

道具X：该道具可以为PACMAN增加一点生命

道具T：该道具可以让场上的所有鬼暂停移动

3.3 程序运行方式

该游戏运行在cmd窗口中，为了游戏大小窗口合适，需要设置窗口属性的字体为3*5的点阵字体。

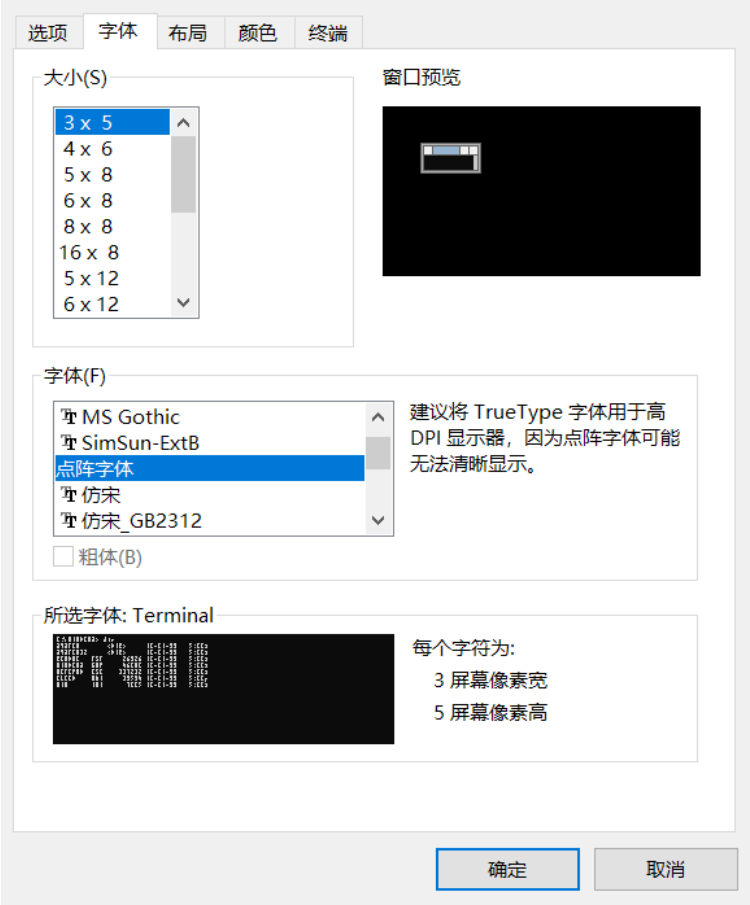

图 3-3 程序运行方式描述

3.4 组队实验分工介绍：

本次实验过程中包含界面设计、控制逻辑设计、碰撞检测、四只追逐PACMAN的幽灵追逐方式的设计、道具设计等多个模块。

在本次组队设计中，魏慧聪同学完成了游戏中幽灵追逐PACMAN的方案构思与代码实现、道具设计、地图和人物设计模块。曹哲瑀同学完成了游戏中幽灵与PACMAN的距离计算、幽灵移动方向计算、幽灵移动实现模块。张驰同学完成了游戏中控制逻辑以及碰撞检测、游戏过程的地图改动、道具设计模块。王浚哲同学完成了游戏中界面设计显示、帧数刷新、分数计算模块。

第 4 章 游戏的分模块介绍

4.1 数据结构

为了更加清晰有条理，程序的数据部分统一定义于文件 define.inc，在此将重要的变量进行介绍。

4.1.1 pacman 表示

```
pacChar dd ?
pacx SBYTE 78
pacy SBYTE 71
pacdir db 2        ;1 = north, 2 = east, 3 = south, 4 = west
```

1）pacx,pacy：pacman 的位置坐标，起始位置（78.71）
2）pacdir:pacman 的移动方向，用 1234 代表北东南析
3）pacChar:pacman 的显示形态

4.1.2 幽灵表示

```
ghos1x SBYTE 72
ghos1y SBYTE 36
ghos1dir db 1
ghos2x SBYTE 78
ghos2y SBYTE 36
ghos2dir db 1
```

```
ghos2targx db 0
ghos2targy db 0
ghos3x SBYTE 84
ghos3y SBYTE 36
ghos3dir db 1
ghos3targx db 0
ghos3targy db 0
ghos4x SBYTE 90
ghos4y SBYTE 36
ghos4dir db 1
ghos4targx db 0
ghos4targy db 0
```

游戏中四个幽灵具有不同的逻辑，在表示时采用四组不同的变量

1）ghos*x,ghos*y（*代表 1/2/3/4）：分别表示四个鬼当前的位置

2）ghos*dir（*代表 1/2/3/4）：分别表示四个鬼当前的方向

以上两组变量是为了确定鬼的行动状态，在移动函数中计算，在显示函数中使用来确定幽灵的形态

3）ghos*targx,ghos*targy(*代表 2/3/4)：基于幽灵 2 幽灵 3 幽灵 4 的不同逻辑，表示追逐的目标坐标，通过距离函数结合不同的算法计算。

4.1.3 方向表示

对于幽灵来说，需要在四个方向做选择。首先判断方向的合法性（不碰撞，不回头），进而根据幽灵坐标与追逐目标坐标计算得出一个距离，该距离值表征若沿某个方向追逐的追逐快慢，距离值小，更容易追上。

```
northlegal db 0   ;0 = false, 1 = true
eastlegal db 0
southlegal db 0
westlegal db 0
northdist db 0
eastdist db 0
southdist db 0
westdist db 0
```

1）northlegal eastlegal southlegal westlegal：四个方向的合法性。若合法则为1，不合法，则为0

2）northdist eastdist southdist westdist：四个方向的距离值

4.1.4 方向选择向量

```
dirchoose db 4 dup(0)
```

结合上面的方向合法性。例如，若北面合法，其余均不合法，则 dirchoose=(1000)

4.1.5 样式数据

内置了地图，人物图形，启动界面的样式串。

4.2 模块 1——界面设计与显示方式

该部分负责初始显示界面（图1）和游戏页面背景的显示（图2蓝色部分）。

图一主要由两部分组成，第一部分显示作者信息：

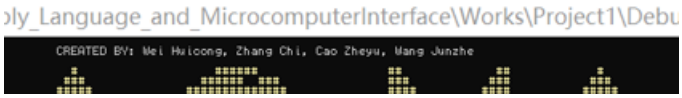

图 4-1 作者信息展示

第二部分显示主要图像内容。

图1输出函数如下：

```
splashScreen PROC
    Call Crlf
    mov edx, OFFSET createdBy  ;输出作者行
    Call WriteString
    Call Crlf
    Call pacManWord  ;主要输出函数
    Call Crlf
    ret
splashScreen ENDP
```

进入main函数先调用splashScreen函数，splashScreen函数3、4行输出作者行，作者信息已先行写入define.inc中，以字节的形式保存：

```
createdBy BYTE "                                CREATED BY: Wei Huicong,
Zhang Chi, Cao Zheyu, Wang Junzhe", 0ah, 0
```

第二部分的输出由pacManWord函数完成

```
pacManWord PROC       ;生成初始界面
    Call yellowText
```

```
mov ecx, 13
    mov edx, OFFSET pac-138
printPacLoop:
    add edx, 139
    Call writeStrWithSP ;从pac第一行开始打印，每行打印完接换行
    loop printPacLoop

    mov ecx, 17
print14to30: ;打印第一部分
    mov esi, -1
    add edx, 139
begin1:
    inc esi  ;esi + 1
    .if esi == 139
        dec ecx ;ecx - 1
        jmp loop1
    .endif
    .if esi < 52   ;从这里开始打印
        mov eax, 14
        call SetTextColor
        mov al, byte ptr[edx+esi] ;相对基址变址寻址
        call WriteChar
    .elseif byte ptr[edx+esi] == '0' ;edx+esi字节中的值为 0， 0打印为白色
        mov eax, 15
        call SetTextColor
        mov al, byte ptr[edx+esi]
        call WriteChar
    .elseif byte ptr[edx+esi] == '1'
        mov eax, 0 ;1打印为黑色
        call SetTextColor
        mov al, byte ptr[edx+esi]
        call WriteChar
    .elseif esi >= 52
        .if esi < 100
            mov eax, 4
            call SetTextColor
            mov al, byte ptr[edx + esi]
            call WriteChar
        .else
            mov eax, 3
            call SetTextColor
            mov al, byte ptr[edx + esi]
            call WriteChar
        .endif
```

```
    .endif
    jmp begin1
loop1:
    ……
begin2:
    ……
    jmp begin2
loop2:
    ……
loop3:
    add edx, 139
    Call writeStrWithSP
    loop loop3
    call Crlf
pacManWord ENDP
```

图1的显示是由字符打印实现的，原始字符已经存入define.inc中的pac中：

pacManWord函数分行对pac进行打印，区分每个字符以不同的颜色，从而最终实现图1的结果。

按图1要求输入任意字符后，利用Clrscr函数清空屏幕

图2输出的内容一直保持直达达成失败或胜利条件，此间，每次每次帧率刷新更新一次吃豆人和四个幽灵的位置，但是地图保持不变，所用函数为drawStringBoard:

```
drawStringBoard proc
    Call valuesToDrawBoard
    mov ecx, 93
printMap:
    add edx, 164
    Call writeStrWithSP
    loop printMap
doneWithMap:
    ret
drawStringBoard endp
```

valueToDrawBoard函数将输出位置改至左上角第一个字符，并将字体颜色设置为蓝色，最后把地图row的初地址移至edx中准备开始打印row中所存储地图：

```
valuesToDrawBoard PROC
    mov dx, 0
    Call Gotoxy
```

```
    mov eax, 9
    Call SetTextColor
    mov edx, offset row-164
    ret
valuesToDrawBoard ENDP
```

row存储地图的方式和上文所提的pac相同。

writeStrWithSP为简单的WriteString函数和一次换行。

当满足胜利或失败条件后，延迟200ms后，清空屏幕，输出战果：

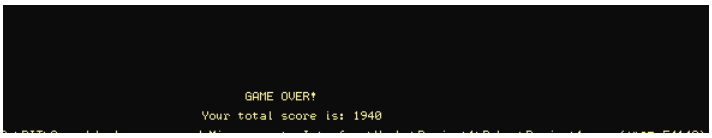

图 4-2 游戏失败界面提示

youWon函数和gameover函数比较简单且结构完全相同，区别仅有输出内容不同：

```
gameOver proc
    Call Clrscr
    mov dh, 12
    mov dl, 34
    Call Gotoxy
    mov edx, offset thatsAllSheWrote
    Call WriteString
    mov dh, 14
    mov dl, 28
    Call Gotoxy
    mov edx, offset result
    Call WriteString
    mov eax, totalScore
    Call WriteDec
    Call Crlf
    mov eax, 1000
    call Delay
    ret
gameOver endp
```

4.3 模块 2——控制逻辑与碰撞检测

控制逻辑的实现比较简单，即按下WASD按钮，控制PACMAN向上下左右四个方向移动。每次移动后，将PACMAN的x位置和y位置存放进入到BH和BL寄存器中。根据WASD的

按键，判断是否遇到了墙体，如果遇到的不是墙体，计算下一次的PACMAN位置，进行第一轮的碰撞检测。比较特殊的一点在于，在移动的过程中，要注意地图上特殊位置的检测（传送点位置的传送检测），要对位置进行调整。同时，在控制逻辑模块中，也记录了当PACMAN遭遇到相应的道具时，设置相应的增益时间。

相应的代码如下所示：(其中省区了一些重复的代码)

```
pacManControls PROC
    Call ReadKey
    mov BH, pacx
    mov BL, pacy
    ;向下走，记方向为 3
    .IF AL == 's' || AL == 'S'
        inc BL
        Call charAtXY
        mov pacdir, 3
        mov pacChar, offset pacmanEatShape+36
        .IF AL != '#'
            inc pacy
            call ghostsVsPacman
            .if al == 'O'
                add bufftime, 50
            .endif
            .if al == 'T'
                add pausetime, 60
            .endif
            .if al == 'X'
                inc totalLives
            .endif
        .ENDIF
    ;向上走，记方向为 1
    .ELSEIF AL == 'w' || AL == 'W'
        ...
    ;向右走，计方向为 2
    .ELSEIF AL == 'd' || AL == 'D'
        ...
    ;向左走，记方向为 4
    .ELSEIF AL == 'a' || AL == 'A'
        ...
    .ENDIF
    cmp pacx, 0
```

```
    je teleportPacMan1
    cmp pacx, 162
    je teleportPacMan2
    jmp nothingHere
```

碰撞检测模块ghostsVsPacman需要调用两次，分别是第一次碰撞检测和第二次碰撞检测。其中第一次碰撞检测的调用位于控制逻辑模块之中。

碰撞检测的过程需要一次比较pacman的中心点位置与四只Ghosts的x位置与y位置是否相等，如果相等，则判定为碰撞成功，死亡，减少一次生命并重置地图。需要进行两次碰撞的原因在于需要避免人与鬼对冲而过导致未检测到的情况。第一次碰撞检测的目的是检测对冲而过导致人与鬼进行碰撞的情况，第二轮碰撞检测则是检测正常的相遇情况。第一次碰撞检测时，只有人朝目标方向进行了移动，是为了防止人与鬼对冲的情况；第二次碰撞检测时，人与鬼均朝目标方向进行了移动，是正常的检测逻辑。

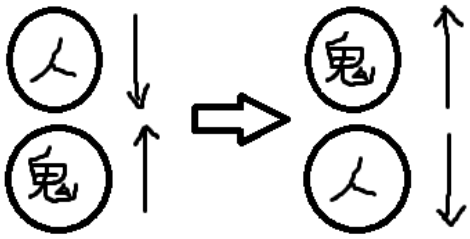

图 4-3 第一次碰撞避免情况的相关描述

碰撞检测的部分代码如下：（删去了重复部分）

```
ghostsVsPacman proc uses eax
    mov ah, ghos1x
    cmp pacx, ah
    je pacmanY1
pacmanX2:
    ...
pacmanX3:
    ...
pacmanX4:
    ...
pacmanY1:
    mov ah, ghos1y
    cmp pacy, ah
```

```
    je DeadMan
    jmp pacmanX2
pacmanY2:
    ...
pacmanY3:
    ...
pacmanY4:
    ...
DeadMan:
    Call caughtInTheAct
noDice:
    ret
ghostsVsPacman endp
```

4.4 幽灵追逐逻辑模块

幽灵追逐逻辑是本实验最为复杂的部分，每个幽灵的运动逻辑互不相同。

四只幽灵分别名为Blinky、Pinky、Inky和Clyde：

1) Blinky的移动逻辑最简单，它总是以玩家Pacman为目标点进行移动；
2) Pinky的移动逻辑与Blinky类似，但它的目标点是玩家Pacman当前移动方向的前方四格的位置（三步为1格）；
3) Inky的移动逻辑最复杂，它会以当前Blinky的位置和Pacman前方两格的一倍延长线终点为目标移动；
4) Clyde的移动最特殊，当它距离Pacman较远（大于8格）时，以和Blinky一样以玩家为目标追逐，但是一旦距离Pacman变近（小于等于8格），就会向着远离Pacman的方向移动

这种逻辑使得Inky有时和Blinky一起在后方追逐玩家，有时和Pinky一起在前方堵截玩家。而Clyde更像是气氛组，由于方向选择中设置了幽灵不能直接掉头的机制，因此Clyde也可以距离玩家很近，在没有分叉的道路上也有可能吃掉玩家。

整体移动逻辑如下图所示：

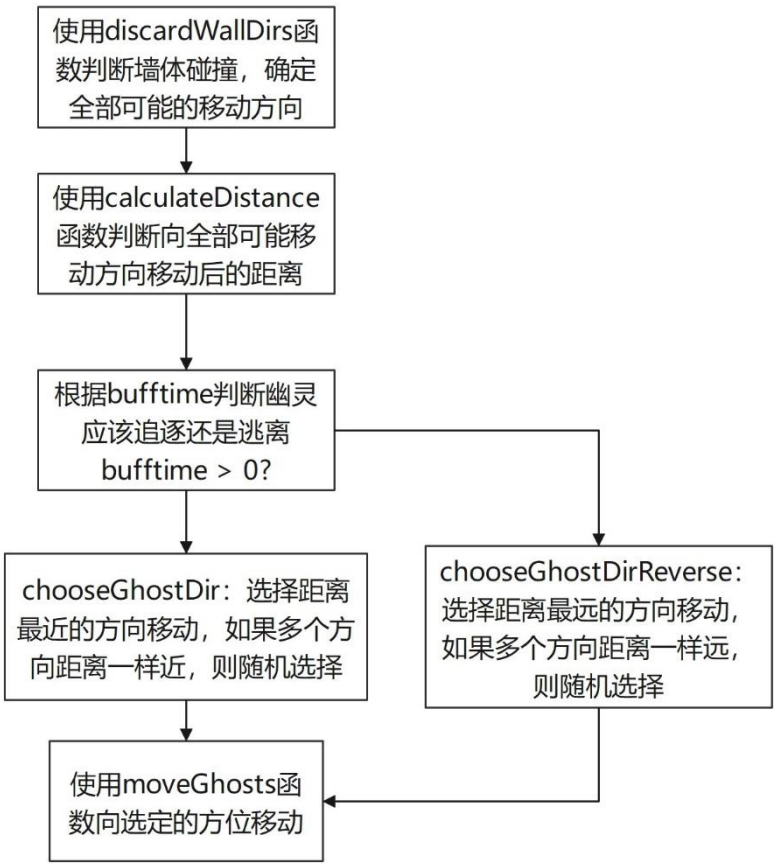

图4-4 幽灵移动逻辑

由于在追逐逻辑不同，在chooseGhostDir中，每个幽灵的方向选择有所不同。而在chooseGhostDirReverse函数中，四只幽灵的移动逻辑相同，只要向着远离方向移动即可。

4.4.1 判断可能的移动方向 discardWallDirs

游戏开始时设置初始移动方向，每时每刻都会有幽灵都会有一个当前的移动方向，在discardWallDirs函数中，首先将当前方向的反方向设置为不可移动，再依次比较其他三个方向是否有墙体阻挡，如果有则视为该方向不可移动，否则可以移动。结果存储在northlegal、eastlegal、southlegal和westlegal四个变量中，为1则可以移动，为0则不可移动。

4.4.2 计算向可能方向移动之后的距离 calculateDistance

该函数的工作逻辑是：分别判断四个方向是否可移动，如果可以移动，则对幽灵当前的(X、Y)坐标进行移动的改变，用absoluteDistance函数计算与目标点坐标的欧

氏距离（实际是两对X与Y的差值之和，与欧氏距离的大小关系等价），将结果保存在northdist、eastdist、southdist和westdist四个变量中。

```
;eax存储距离绝对值
absoluteDistance proc
    cmp al, bl
    jb yolo
    sub al, bl
    movzx eax, al
    jmp returndist
    yolo:
    sub bl, al
    movzx eax, bl
    returndist:
    ret
absoluteDistance endp
```

4.4.3 目标点的确定

幽灵所有的移动方式都是向着某个目标点移动或者远离某目标点移动的。Blinky的目标点选取最简单，只需要将玩家(X,Y)的坐标直接当作目标点用calculateDistance计算距离即可。但是Pinky和Inky的目标点选择较为复杂，需要一些计算，分别在acquireG2Target和acquireG3Target中完成。这部分不是难点，但是需要的参数较多，为了方便传递参数，仅仅使用了寄存器传递，因此只能使用8位的数据表示，使用了无符号数计算，需要考虑到越界的情况，及时判断并作出纠正，例如在Pinky的目标点确定函数acquireG2Target中：

```
westcheck:     ;如果是向左移动
    mov al, pacx
    .if al < 26
        mov al, 2
    .else
        sub al, 24 ;横向一格为6，前方四格就是-24
    .endif
    mov ghos2targx, al
    mov al, pacy
    mov ghos2targy, al    ;y的坐标无需变化
```

4.4.4 选择移动方向 chooseGhostDir

选择最小的距离可以使用依次比较的办法选出最小的一个距离，但是这种方法会导致在相等的情况下优先选择某个方向，经过实际测试，容易在游戏过程中在特定位置容易出现幽灵绕圈圈的情况，也就是可能会出现绝对安全的位置。因此需要将每个方向的距离都与其他三个方向依次比较，如果该方向距离最小，则存入dirchoose数组当中。

需要注意的是有些方向无法移动，即不可达，不能简单地将dist设置为某个大数，因为在幽灵逃离的过程中会选择距离最大的方向逃跑，这样设置会使得逃离的逻辑出现错误。

所以在实际编程的过程中，需要先判断该方向是否可以移动，再判断该方向距离是否最小，如果最小则将该方向加入到dirchoose数组，并用dircount记录数组大小，最后采用一种随机方法选择一个方向。随机值取当前的帧计数的值对数组大小取模，由于玩家在某一位置可能是任意一帧，结果已经相当不可预测，该随机选取方法具有一定的可靠性，随机选取的汇编代码如下：

```
mov ax, framecount
div dircount
movzx edi, ah ;edi存储了余数信息
mov esi, offset dirchoose
add esi, edi
movzx eax, byte ptr[esi] ;eax 存储随机选择的方向
```

在Clyde的逻辑模块，需要判断当前幽灵与玩家的欧氏距离来判断靠近还是远离：

```
mov ch, ghos4x
mov cl, ghos4y
mov dl, ghos4dir
Call discardWallDirs
mov ch, ghos4x
mov cl, ghos4y
mov dh, pacx
mov dl, pacy
Call calculateDistance
Call calculateDistanceG4FromPacman ;距离存储在dl中
.if dl < 24        ;如果在8格以内，则远离
    Call chooseGhostDirReverse
.else          ;否则靠近
    Call chooseGhostDir
.endif
mov ghos4dir, al
```

选择远离方向chooseGhostDirReverse同样以距离为判断的条件，为了简化，不需要从几个一样大的距离中随机选择。设置幽灵逃离的目的就是为了让玩家在紧张刺激的追逐中有短暂的喘息之机，由于这个状态下幽灵不会去追逐玩家而且时间较短，所以幽灵在某些特殊位置可能暂时循环绕圈也无关紧要。

4.5 道具设计与道具效果实现模块

如上游戏简介中所示，本游戏中设置了三种道具：

道具O：该道具干扰鬼的逻辑，在一定时间内幽灵会尽可能避免追逐PACMAN

道具X：该道具可以为PACMAN增加一点生命

道具T：该道具可以让场上的所有鬼暂停移动

道具的设计体现在地图之中，地图中的符号’ •’表示吃豆人要吃到的豆子，而地图上的符号’O’ ,’X’ ,’T’分别表示不同的符号。

其中，O道具和T道具均为增益道具，具有相应的时长，在吃到道具后时长设置为固定值，之后随着帧的变化时长逐渐减少。当减少到0时，道具的效果结束。X道具直接使当前的生命值增加一点即可。

其中，判定吃到道具的代码已经在控制逻辑模块中进行了展示。下面展示三种道具的效果体现：

```
;幽灵的移动函数
moveGhosts proc
    .if pausetime > 0
    jmp movedone
    .endif
    ...
movedone:
    ret
moveGhosts endp
    .if bufftime > 0
        dec bufftime
    .endif
    .if pausetime > 0
        dec pausetime
    .endif
    .if bufftime > 0
        ;决定下一次向哪个方向移动
        Call directGhostsReverse
    .else
        call directGhosts
    .endif
```

其中，增加生命道具的效果体现已经在控制逻辑模块进行了展示。

第 5 章 实验心得体会

在本次实验中，我们队伍四人通力协作，完成了本次组队实验——PACMAN游戏的编写。通过这次实验，我们队伍四人对各类汇编指令的使用更加熟练。同时，通过参考网络上的相关文档，学习到了Irvine32库的相关指令，对汇编语言程序设计有了进一步的理解。

此外，本次的实验项目也非常有趣，PACMAN本身的制作与游戏过程就是一件非常快乐的事情。希望在之后的学习生活中，我们可以继续在学习中获得快乐，在快乐中完成学业。