汇编-贪吃蛇

运行环境

- 本次项目运行于实模式下以及使用大量dos中断来完成的，而Windows10并不支持dos程序，因此使用了dosbox作为运行环境
- 汇编器：nasm
- 汇编指令 `nasm -o ./snake.com ./main.asm`

实现功能

- 实现贪吃蛇的移动，吃食物等操作
- 小蛇随着吃掉食物数目的增加，移动速度加快，身体变长
- 支持暂停功能
- 支持分数统计
- 支持多种难度
- 使用菜单界面完成游戏的一些选项操作

操作方式

- 游戏以'**w**'，'**a**'，'**s**'，'**d**'来控制小蛇的移动方向
- 使用'**Esc**'键完成游戏的暂停功能
- 使用'**CTRL+c**'退出游戏（也可使用菜单界面退出）
- 使用鼠标来操作右边的菜单界面

代码设计

一些约定

- 所有汇编程序的函数调用时都应保存通用寄存器现场，并在返回前还原现场，使得调用起来不存在心理负担
- 在有返回值的函数中，返回值一律存放在ax寄存器中
- 使用压栈的方式传递参数

计时器

- 对于一个需要不断刷新的游戏来说，首先就是需要一个计时器来完成控制刷新频率的工作，本程序使用死循环加监听系统时间的方式来实现。
- 系统时间可通过dos调用int21h的2ch号功能来获取，最小单位为0.01秒，于是我决定以每0.03秒作为计时器的单位时间。
- 使用TimerCheck函数记录下当前时间并与上一次时间进行比较，当差值大等于0.03秒时便返回检测成功，通知控制程序进行界面的更新。
- 除此之外，考虑到后一次的时间肯定比前一次时间大，因此我们只需将前一次系统时间中以0.01秒为单位的部分，存放于变量`last_time`，然后记录当前时间中以0.01秒为单位的部分，若该部分小于`last_time`，则加上100在进行比较，若结果大于3，则返回检测成功。

随机数生成器

- 考虑到食物应当随机出现于各个位置，因此就需要一个随机数生成器，这里选择使用混合同余法生成随机数

- 使用公式`Xn+1=(LAMDA*Xn+C) MOD mod_value`生成随机数，这里令`LAMDA =91 C=7 mod_value=2^16-1`，X0同样使用系统时间中以0.01秒为单位的部分

保存现场

- 游戏的界面绘制需要对整个屏幕进行操作，因此采取的手段是直接向25*80彩色显示模式下的显存0xB8000 ~ 0xBFFFF直接写入数据，由于dosbox始终运行在第一页，因此我们只需对第一页的显存进行操作。
- 由于游戏的绘制会改变整个屏幕，因此在清空屏幕前我们先把原屏幕的信息保存起来，待程序退出之后将页面还原。
- 同时还由于界面还存在不美观的光标，通过对IO端口0x3d4h以及0x3d5h进行读写将光标隐藏起来，待退出时还原。

地图绘制

- 本游戏的地图较为简单，就是由四面墙壁构成的一个方盒。
- 本程序封装了函数`DrawLine`用于绘制矩形，接受矩形左上角的横纵坐标，长度和宽度以及填充的字符作为参数，由于墙壁的宽度和填充字符都是一样的，因此在函数`DrawLine`基础上再次封装了`DrawHorizontalWall`和`DrawVerticalWall`函数绘制水平和竖直方向的墙壁。而将绘制函数分为水平和竖直方向是由于25*80彩色显示模式单个字符的宽高比约为1:2，因此水平和竖直方向填充的字符数有所不同。

食物

- 使用随机数生成器生成两个随机数，之后再对生成的随机数使用游戏场景的长度和宽度进行取模，即得到食物的横纵坐标。
- 然后检测该位置是否可放置食物，若不能重复上述操作
- 经测试绝大部分情况下最多经过150*250次循环就可生成场景内的任意位置的横纵坐标，当然在一般情况下只需要10次循环以内的时间即可，但为避免极端情况导致的死循环，决定设置生成坐标次数的最大值。
- 将横纵坐标转换为线性地址然后将食物字符写入对应的显存中
- 同时保存了生成的食物所在的线性地址，从而判断食物是否存在。
- 提供了`CheckFood`函数，先检查食物是否存在，然后设置新的食物，并调用相关函数更新分数统计

贪吃蛇设计

节点结构

- 对于贪吃蛇来说，首先考虑到其形状是不规则的，于是我想到可以将其拆分成多条线的组合，每条线用一个结构体维护，结构体的结构如下
  `loc dw, dir db, len db`
- 由于贪吃蛇移动时只会存在头部和尾部向前移动，因此只需要对头部节点和尾部节点进行操作，因此可使用队列的形式维护

转弯

- 我们可以发现，若我们将四个方向按顺时针排序，可得到**左上右下**，依次将其赋值为1，2，3，4，可以发现
  `(dir_val.turnleft().val()-1)=(dir_val.val()-1-1) MOD 4,`
  `(dir_val.turnright().val()-1)=(dir_val.val()-1+1) MOD 4`
- 这样我们就得到转弯时方向的计算公式了
- 当蛇须要转弯时，我们可以先创建一个新的节点进入队列尾部，位置于原队尾一致，长度为0，方向可通过上述公式计算得到
- 这些操作都封装于函数`CheckDirection`

碰撞检测

- 在绘制移动的头部前检测该位置的字符是否为蛇身或是墙壁，若是墙壁则游戏结束
- 这部分操作封装于函数`CheckWall`

绘制移动过程

- 由于贪吃蛇的移动只涉及到头部和尾部节点，故只需操作头部节点和尾部节点
- 我们可通过队列的队首及队尾索引得到贪吃蛇的头部和尾部节点的结构体
- 我们通过头部节点的信息计算出下一个到达的位置的显存地址并填充上贪吃蛇的颜色，之后调用`CheckWall`检测是否碰到墙壁或自身，这一部分操作封装于函数`SnakeHeaderMove`
- 我们通过尾部节点的信息计算出下一个离开位置的显存地址并填充上空字符，这一部分操作封装于函数`SnakeEndMove`

变长

- 贪吃蛇在吃掉一定食物后会变长
- 因此我们记录吃掉的食物数目`food_num`，当达到一定数目时，我们只需要调用`SnakeHeaderMove`函数并将`food_num`清零即可完成一次增长操作
- 判断是否应该增长以及增长操作已经封装于函数`Enlongate`

移动逻辑

- 首先调用`CheckDirection`判断贪吃蛇接下来的移动方向
- 调用`Enlongate`，选择在这个地方调用该函数是出于以下考虑，若玩家在吃掉了食物时即将碰撞到墙壁或蛇身，若在此时调用该函数可能会使蛇身变长导致游戏结束，影响游戏体验，而在此时调用可让玩家有改变方向的缓冲期。
- 调用`SnakeEndMove`移动尾部，若尾部长度为0，则将其移出队列
- 调用`SnakeHeaderMove`移动头部，将移动头部放在移动尾部后是为了避免本应头尾错开却应先移动头部导致游戏结束
- 调用`CheckFood`判断食物是否被吃，并更新分数和放置新食物。
- 这部分逻辑封装于函数`Move`中

速率控制

- 采用进度条式的方式控制速率，每次增加的进度为`init_v+score/50`，其中`init_v`只与游戏开始时选择的难度有关，当进度达到`MAX_RATE`时，清零进度并调用`Move`
- 该部分封装于函数`RealMove`，这也是游戏控制器实际上调用的函数

菜单界面

- 游戏实现了菜单界面，玩家可通过点击相应的菜单调用相应的功能
- 每个菜单选项都是直接像显存写入数据实现的
- 在绘制菜单时需要注意的是，dos系统中鼠标会改变悬浮位置的字符颜色，并在移开后复原，推测这是鼠标驱动同样使用了保存现场和还原现场的机制，但这样导致的在绘制菜单时，鼠标所覆盖的位置将绘制失败，因此在绘制前先将鼠标隐藏起来

鼠标事件监听

- 鼠标采用dos系统提供的int33h中断进行配置
- 和计时器一样，鼠标同样采用死循环的方式进行事件监听，每个循环体内调用 int33h中断的06h号功能判断按键是否有松开按键的操作
- 当按键松开时，则判断按键位置从而调用相应函数
- 由于鼠标位置采用横纵坐标的方式，同时一个字符为8*8的大小，因此要先对齐进行转换才能得到对应的显存地址。

- 得到显存地址后，则判断该地址是否在某一按钮占用的地址之间，同时由于在某些情况，一些按钮的功能没有启用，因此代码中还储存了一些状态变量来辅助判断，如`is_pause wait_select`，当`is_pause==1`时，才启用**继续**按钮，当`wait_select==1`时，则进行难度选择功能

键盘事件监听

- 监听方式与鼠标类似，采用死循环加判断键盘缓存区是否有输入获取用户输入
- 当获取到的输入不是程序提供的功能按键时，则什么也不做
- 当获取到方向键时，将其存入变量`now_dir`中供贪吃蛇的移动函数使用
- 当获取到暂停键或退出键时，直接执行相应函数

游戏主控制器

- 使用死循环调用键盘监听和鼠标监听函数
- 同时判断游戏是否结束或是否暂停
- 如果游戏处在开始阶段且不处在暂停状态，调用计时器
- 计时器判断成功时，调用`RealMove`进行游戏

运行效果