Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Лабораторная работа № 4

по дисциплине ПнаЯА

Вариант 1

Выполнил студент гр. 150501: Барило К.С.

Проверил: Туровец Н.О.

Минск 2022

Тема работы: Создание видеоигры

Цель работы: Ознакомиться в рамках создания видеоигры с обработкой нажатий кнопок клавиатуры, рассмотреть прямой доступ к видеопамяти с целью формирования игрового поля и информации для пользователя.

Теоретические сведения:

1. Видеорежимы и управление ими.

2. Прямой доступ к видеопамяти.

3. Управление курсором и вывод данных на экран прерыванием BIOS. 4. Обработка нажатий кнопок клавиатуры прерыванием BIOS.

Буфер клавиатуры.

5. Доступ к системным часам.

Задание:

Игра «Змейка».

Цель: ползаем, едим случайно появляющиеся в свободных местах яблочки и растем от этого в длину, выход за границы экрана означает возврат в поле с противоположной стороны (круглый мир).

Окончание: проигрыш – укус самого себя ил врезание в препятствие, выигрыш – нет.

Информация: счет.

Код программы:

|  |
| --- |
| clearScreen MACRO ; |
|  | push ax ; Сохраняем значение ax |
|  | mov ax, 0003h ; 00 - установить видеорежим, очистить экран. 03h - режим 80x25 |
|  | int 10h ; Вызов прерывания для исполнения команды |
|  | pop ax ; Восстанавливаем значение регистра ax |
|  | ENDM ; |
|  | ;end macro help |
|  |  |
|  | .model small |
|  |  |
|  | .stack 100h |
|  |  |
|  | .data |
|  |  |
|  | ;key bindings (configuration) |
|  | KUpSpeed equ 48h ; Up key |
|  | KDownSpeed equ 50h ; Down key |
|  | KMoveUp equ 11h ; W key |
|  | KMoveDown equ 1Fh ; S key |
|  | KMoveLeft equ 1Eh ; A key |
|  | KMoveRight equ 20h ; D key |
|  | KExit equ 01h ; ESC key |
|  | ; |
|  | xSize equ 80 ; Ширина консоли |
|  | ySize equ 25 ; Высота консоли |
|  | xField equ 50 ; Ширина поля |
|  | yField equ 21 ; Высота поля |
|  | oneMemoBlock equ 2 ; Размер одной "клетки" консоли |
|  | scoreSize equ 4 ; Длина блока счета |
|  | ; |
|  | videoStart dw 0B800h ; Смещение видеобуффера |
|  | dataStart dw 0000h ; |
|  | timeStart dw 0040h ; |
|  | timePosition dw 006Ch ; |
|  | ; |
|  | space equ 0020h ; Пустой блок с черным фоном |
|  | snakeBodySymbol equ 0A40h ; Символ тела змейки |
|  | appleSymbol equ 0B0Fh ; Символ яблока |
|  | VWallSymbol equ 0FBAh ; Символ вертикальной стены |
|  | HWallSymbol equ 0FCDh ; Символ горизонтальной стены |
|  | BWallSymbol equ 4020h ; |
|  | VWallSpecialSymbol equ 0FCCh ; Символ перекрещивания стен |
|  |  |
|  | fieldSpacingBad equ space, VWallSymbol, xField dup(space) |
|  | fieldSpacing equ fieldSpacingBad, VWallSymbol |
|  | rbSym equ 077DCh ; Белый блок с белым фоном |
|  | rbSpc equ 04F20h ; Пробел с красным фоном и белым цветом символов |
|  | ylSym equ 06FDCh ; Белый блок с желтым фоном |
|  | ylSpc equ 06F20h ; Пробел с желтым фоном |
|  | grSym equ 02FDBh ; Белый блок с зеленым фоном |
|  | grSpc equ 02F20h ; Пустой блок с белым фоном |
|  |  |
|  | screen dw xSize dup(space) |
|  | dw space, 0FC9h, xField dup(HWallSymbol), 0FCBh, xSize - xField - 5 dup(HWallSymbol), 0FBBh, space |
|  | firstBl dw fieldSpacing, xSize - xField - 5 dup(rbSpc), VWallSymbol, space |
|  | dw fieldSpacing, rbSpc, 4 dup(rbSym), 15 dup(rbSpc), 4 dup(rbSym), rbSpc, VWallSymbol, space |
|  | dw fieldSpacing, rbSpc, rbSym, 5 dup(rbSpc), 3 dup(rbSym), 2 dup(rbSpc), 3 dup(rbSym), rbSpc, rbSym, 3 dup(rbSpc), rbSym, 2 dup(rbSpc), rbSym, rbSpc, VWallSymbol, space |
|  | dw fieldSpacing, rbSpc, 4 dup(rbSym), rbSpc, rbSym, 2 dup(rbSpc), rbSym, rbSpc, rbSym, 2 dup(rbSpc), 3 dup(rbSym, rbSpc), 4 dup(rbSym), rbSpc, VWallSymbol, space |
|  | dw fieldSpacing, 4 dup(rbSpc), rbSym, rbSpc, rbSym, 2 dup(rbSpc), rbSym, rbSpc, 4 dup(rbSym), rbSpc, 2 dup(rbSym), 2 dup(rbSpc), rbSym, 4 dup(rbSpc), VWallSymbol, space |
|  | dw fieldSpacing, rbSpc, 4 dup(rbSym), rbSpc, rbSym, 2 dup(rbSpc), rbSym, rbSpc, rbSym, 2 dup(rbSpc), 3 dup(rbSym, rbSpc), 4 dup(rbSym), rbSpc, VWallSymbol, space |
|  | dw fieldSpacing, xSize - xField - 5 dup(rbSpc), VWallSymbol, space |
|  | delim1 dw fieldSpacingBad, 0FCCh, xSize - xField - 5 dup(HWallSymbol), 0FB9h, space |
|  | secondF dw fieldSpacing, xSize - xField - 5 dup(ylSpc), VWallSymbol, space |
|  | dw fieldSpacing, ylSpc, 06F53h, 06F63h, 06F6Fh, 06F72h, 06F65h, 06F3Ah, ylSpc |
|  | score dw scoreSize dup(06F30h), xSize - xField - scoreSize - 13 dup(ylSpc), VWallSymbol, space |
|  | dw fieldSpacing, xSize - xField - 5 dup(ylSpc), VWallSymbol, space |
|  | dw fieldSpacing, ylSpc, 06F53h, 06F70h, 2 dup(06F65h), 06F64h, 06F3Ah, ylSpc |
|  | speed dw 06F31h, 16 dup(ylSpc), VWallSymbol, space |
|  | dw fieldSpacing, xSize - xField - 5 dup(ylSpc), VWallSymbol, space |
|  | delim2 dw fieldSpacingBad, 0FCCh, xSize - xField - 5 dup(HWallSymbol), 0FB9h, space |
|  | thirdF dw fieldSpacing, xSize - xField - 5 dup(grSpc), VWallSymbol, space |
|  | dw fieldSpacing, grSpc, 02F43h, 02F6Fh, 02F6Eh, 02F74h,02F72h, 02F6Fh, 02F6Ch,02F73h, 02F3Ah, 15 dup(grSpc), VWallSymbol, space |
|  | dw fieldSpacing, grSpc, 02F57h, grSpc, 02FC4h, grSpc, 02F55h, 02F70h, 02F18h, 17 dup(grSpc), VWallSymbol, space |
|  | dw fieldSpacing, grSpc, 02F53h, grSpc, 02FC4h, grSpc, 02F44h, 02F6Fh, 02F77h ,02F6Eh, 02F19h, 15 dup(grSpc), VWallSymbol, space |
|  | dw fieldSpacing, grSpc, 02F41h, grSpc, 02FC4h, grSpc, 02F4Ch, 02F65h, 02F66h ,02F74h, 02F1Bh, 15 dup(grSpc), VWallSymbol, space |
|  | dw fieldSpacing, grSpc, 02F44h, grSpc, 02FC4h, grSpc, 02F52h, 02F69h, 02F67h ,02F68h, 02F74h, 02F1Ah, 14 dup(grSpc), VWallSymbol, space |
|  | dw fieldSpacing, grSpc, 02F45h, 02F73h,02F63h, grSpc, 02FC4h, grSpc, 02F45h, 02F78h, 02F69h ,02F74h, 02F13h, xSize - xField - 17 dup(grSpc), VWallSymbol, space |
|  | dw space, 0FC8h, xField dup(HWallSymbol), 0FCAh, xSize - xField - 5 dup(HWallSymbol), 0FBCh, space |
|  | dw xSize dup(space) |
|  |  |
|  | ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | ; BANNER |
|  | ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | widthOfBanner equ 40 ; |
|  | allWidth equ 80 ; |
|  | black equ 0020h ; |
|  | white equ 4020h ; |
|  | black equ 0020h ; |
|  |  |
|  | blackVWallSymbol equ 00FBAh |
|  | blackHWallSymbol equ 00FCDh |
|  |  |
|  | wastedBanner dw 00FC9h, widthOfBanner-2 dup(blackHWallSymbol), 0FBBh |
|  | dw blackVWallSymbol, widthOfBanner-2 dup(black), blackVWallSymbol |
|  | dw blackVWallSymbol, 4 dup(black), white, 5 dup(black), white, 2 dup(black), 2 dup(white), black, 4 dup(white), black, 3 dup(white), black, 3 dup(white), black, 3 dup(white), 6 dup(black), blackVWallSymbol |
|  | dw blackVWallSymbol, 4 dup(black), white, 5 dup(black), white, black, white, black, white, black,white, black, black, black, black, black, white, 2 dup(black), white, 2 dup(black), black, white, black, black, white, 5 dup(black), blackVWallSymbol |
|  | dw blackVWallSymbol, 5 dup(black), 3 dup(white, black), black, 3 dup(white), black, 4 dup(white), 2 dup(black), white, 2 dup(black), 2 dup(white), 2 dup(black), white, 2 dup(black), white, 5 dup(black), blackVWallSymbol |
|  | dw blackVWallSymbol, 5 dup(black), 3 dup(white, black), black, white, black, white, 4 dup(black), white, 2 dup(black), white, 2 dup(black), white, 2 dup(black), black, white, 2 dup(black), white, 5 dup(black), blackVWallSymbol |
|  | dw blackVWallSymbol, 6 dup(black), 2 dup(white, black), 2 dup(black), white, black, white, black, 4 dup(white), 2 dup(black), white, 2 dup(black), 3 dup(white), black, 3 dup(white), 6 dup(black), blackVWallSymbol |
|  | dw blackVWallSymbol, widthOfBanner-2 dup(black), blackVWallSymbol |
|  | dw blackVWallSymbol, 7 dup(black) ,08F50h, 08F72h, 08F65h, 08F73h, 08F73h, 08F00h, 08F61h, 08F6Eh, 08F79h, 08F00h, 08F6Bh, 08F65h, 08F79h, 08F00h, 08F74h, 08F6Fh, 08F00h, 08F65h, 08F78h, 08F69h, 08F74h, 10 dup(black), blackVWallSymbol |
|  | dw 0FC8h, widthOfBanner-2 dup(blackHWallSymbol), 0FBCh |
|  |  |
|  | snakeMaxSize equ 30 |
|  | snakeSize db 3 |
|  | PointSize equ 2 |
|  |  |
|  | snakeBody dw 1D0Dh, 1C0Dh, 1B0Dh, snakeMaxSize-2 dup(0000h) |
|  |  |
|  | brickWallSize equ 9 |
|  |  |
|  | brickWall1 dw 0303h, 0302h, 0301h, 0300h, 02FFh, 0203h, 0103h, 0003h, 0FF03h |
|  | brickWall2 dw 0103h, 0003h, 0FF03h, 0FE03h, 0FD03h, 0FD02h, 0FD01h, 0FD00h, 0FCFFh |
|  | brickWall3 dw 01FEh, 00FEh, 0FFFEh, 0FEFEh, 0FDFEh, 0FD01h, 0FD00h, 0FCFFh, 0FCFEh |
|  | brickWall4 dw 01FEh, 00FEh, 0FFFEh, 0FEFEh, 002FEh, 00401h, 00400h, 003FFh, 003FEh |
|  |  |
|  | brickWallTemplate dw brickWallSize dup(0) |
|  |  |
|  | brickWallTrue dw brickWallSize dup(0) |
|  |  |
|  | stopVal equ 00h |
|  | forwardVal equ 01h |
|  | backwardVal equ -1 |
|  |  |
|  | Bmoveright db 01h |
|  | Bmovedown db 00h |
|  |  |
|  | minWaitTime equ 1 |
|  | maxWaitTime equ 9 |
|  | waitTime dw maxWaitTime |
|  | deltaTime equ 1 |
|  |  |
|  | .code |
|  |  |
|  | main: |
|  | mov ax, @data ; |
|  | mov ds, ax ; |
|  | mov dataStart, ax ; Загружаем начальные данные |
|  | mov ax, videoStart ; Загружаем в ax код начала вывода в видеобуффер |
|  | mov es, ax ; Загружаем ax в es |
|  | xor ax, ax ; Обнуляем ax |
|  | ; |
|  | clearScreen ; Очищаем консоль |
|  | ; |
|  | call initAllScreen ; Инициализируем экран |
|  | ; |
|  | call mainGame ; Переходим в основной цикл игры |
|  | ; |
|  | to\_close: ; |
|  | call printBanner ; |
|  | mov ah,7h ; 7h - консольный ввод без эха (ожидаем нажатия клавиши для выхода из приложения) |
|  | int 21h ; |
|  |  |
|  | esc\_exit: |
|  |  |
|  | clearScreen ; |
|  | ; |
|  | mov ah, 4ch ; |
|  | int 21h ; |
|  | ; |
|  | ; |
|  | ;ZF = 1 - Буффер пуст ; |
|  | ;AH = scan-code ; |
|  | CheckBuffer MACRO ; Проверяем - был ли введен символ с клавиатуры |
|  | mov ah, 01h ; |
|  | int 16h ; |
|  | ENDM ; |
|  | ; |
|  | ReadFromBuffer MACRO ; Считываем нажатую клавишу |
|  | mov ah, 00h ; |
|  | int 16h ; |
|  | ENDM ; |
|  | ; |
|  | ;Результат в cx:dx ; |
|  | GetTimerValue MACRO ; |
|  | push ax ; Сохраняем значения регистра ax |
|  | ; |
|  | mov ax, 00h ; Получаем значение времени |
|  | int 1Ah ; |
|  | ; |
|  | pop ax ; Восстанавливаем значение регистра ax |
|  | ENDM ; |
|  | ; |
|  |  |
|  | printBanner PROC |
|  | push es ; |
|  | push 0B800h ; |
|  | ; 0b800h |
|  | pop es ; ES=0B800h |
|  | ; |
|  | mov di, 7\*allWidth\*2 + (allWidth - widthOfBanner) ; |
|  | mov si, offset wastedBanner ; |
|  | mov cx, 10 ; |
|  | cld ; |
|  | loopPrintBanner: ; |
|  | ; |
|  | push cx ; |
|  | ; |
|  | mov cx, widthOfBanner ; |
|  | rep movsw ; |
|  | ; |
|  | add di, 2\*(allWidth - widthOfBanner); |
|  | ; |
|  | pop cx ; |
|  | loop loopPrintBanner ; |
|  | std ; |
|  | pop es ; |
|  | ret ; |
|  | ENDP |
|  |  |
|  | drawBrickWall PROC |
|  | push cx |
|  | push bx |
|  | mov cx, brickWallSize |
|  |  |
|  | mov si, offset brickWallTrue |
|  | loopBrickWall: |
|  | mov bx, [si] ; Загружаем в si очередной символ |
|  | add si, PointSize ; |
|  |  |
|  | ; Получаем позицию в видеобуффере(bh + (bl \* xSize))\*oneMemoBlock |
|  | call CalcOffsetByPoint ; Получаем смещение выводимого символа в видеобуффере |
|  | ; |
|  | mov di, bx ; загружаем в di позицию |
|  | ; |
|  | mov ax, BWallSymbol ; Загружаем в ax выводимый символ |
|  | stosw ; Выводим |
|  | loop loopBrickWall |
|  | pop bx |
|  | pop cx |
|  | ret |
|  | ENDP |
|  |  |
|  | destroyWall PROC |
|  | push cx |
|  | mov cx, brickWallSize |
|  |  |
|  | mov si, offset brickWallTrue |
|  | loopDestroyWall: |
|  | mov bx, [si] ; Загружаем в si очередной символ |
|  | add si, PointSize ; |
|  |  |
|  | call CalcOffsetByPoint ; Получаем смещение выводимого символа в видеобуффере |
|  | ; |
|  | mov di, bx ; загружаем в di позицию |
|  | ; |
|  | mov ax, space ; Загружаем в ax выводимый символ |
|  | stosw ; Выводим |
|  | loop loopDestroyWall |
|  |  |
|  | pop cx |
|  | ret |
|  | ENDP |
|  | ; |
|  | initAllScreen PROC ; |
|  | mov si, offset screen ; В si загружаем |
|  | xor di, di ; Обнуляем di |
|  | ; Теперь ds:si указывает на символы, которые мы будем выводить |
|  | ; а es:di на di'ый символ консоли |
|  | mov cx, xSize\*ySize ; Загружаем в cx кол-во символов в консоли, т.е. 80x25 |
|  | rep movsw ; Переписываем последовательно все cx символов из ds:si в консоль es:di |
|  | ; |
|  | ; |
|  | xor ch, ch ; Обнуляем ch |
|  | mov cl, snakeSize ; Загружаем в cl размер змейки |
|  | mov si, offset snakeBody; В si загружаем смещения начала тела змейки |
|  | ; |
|  | loopInitSnake: ; Цикл, в котором мы выводим тело змейки |
|  | mov bx, [si] ; Загружаем в si очередной символ тела змейки |
|  | add si, PointSize ; Добавляем к si PointSize, т.е. 2, т.к. каждая точка занимает 2 байта (цвет + символ) |
|  |  |
|  | call CalcOffsetByPoint ; Получаем смещение выводимого символа в видеобуффере |
|  | ; |
|  | mov di, bx ; загружаем в di позицию |
|  | ; |
|  | mov ax, snakeBodySymbol ; Загружаем в ax выводимый символ |
|  | stosw ; Выводим |
|  | loop loopInitSnake ; |
|  | ; |
|  | call GenerateRandomApple; Генерируем яблоко в случайных координатах |
|  | ; |
|  | ret ; |
|  | ENDP ; |
|  |  |
|  | ;Получаем смещение видеобуффера как (bh + (bl \* xSize))\*oneMemoBlock |
|  | ;input: Координаты (x,y) в bx |
|  | ;output: Смещение в bx |
|  | CalcOffsetByPoint PROC ; |
|  | push ax ; Сохраняем значения регистров ax и dx |
|  | push dx ; |
|  | ; |
|  | xor ah, ah ; Обнуляем ah |
|  | mov al, bl ; Загружаем в al bl |
|  | mov dl, xSize ; В dl загружаем xSize - размер строки |
|  | mul dl ; Умножаем al на dl |
|  | mov dl, bh ; Загружаем в dl bh |
|  | xor dh, dh ; Обнуляем dh |
|  | add ax, dx ; Добавляем к ax dx |
|  | mov dx, oneMemoBlock ; Загружаем в dx oneMemoBlock - длину каждого блока |
|  | mul dx ; Умножаем на размер блока |
|  | mov bx, ax ; Загружаем ax в bx |
|  | ; |
|  | pop dx ; Восстанавливаем значения регистров dx и ax |
|  | pop ax ; |
|  | ret ; |
|  | ENDP ; |
|  |  |
|  | ;Сдвигаем тело змейки в массиве |
|  | ;Удаляем старый последний элемент |
|  | ;Закрашиваем последний элемент |
|  | MoveSnake PROC ; |
|  | push ax ; |
|  | push bx ; |
|  | push cx ; |
|  | push si ; Сохраняем значения регистров |
|  | push di ; |
|  | push es ; |
|  | ; |
|  | mov al, snakeSize ; В al загружаем длину змейки |
|  | xor ah, ah ; Обнуляем ah |
|  | mov cx, ax ; Загружаем в cx ax |
|  | mov bx, PointSize ; Загружаем в bx размер точки на экране |
|  | mul bx ; Теперь в ax реальная позиция в памяти относительно начала массива |
|  | mov di, offset snakeBody; Загружаем в di смещение головы змейки |
|  | add di, ax ; di - адрес следующего после последнего элемента массива |
|  | mov si, di ; Загружаем di в si |
|  | sub si, PointSize ; si - адрес последнего элемента массива |
|  | ; |
|  | push di ; Сохраняем значение di |
|  | ; Удаляем конец змейки с экрана |
|  | mov es, videoStart ; Загружаем в es смещение видеобуффера |
|  | mov bx, ds:[si] ; Загружаем в bx последний элемент змейки |
|  | call CalcOffsetByPoint ; Вычисляем ее позицию на экране |
|  | mov di, bx ; Заносим позицию, которую будем очищать в di |
|  | mov ax, space ; Загружаем в ax пустую клетку |
|  | stosw ; Записываем (пересылаем содерджимое ax в es:di) |
|  | ; |
|  | pop di ; Восстанавливаем di |
|  | ; |
|  | mov es, dataStart ; Для работы с данными (до этого es указывал на видеобуффер) |
|  | std ; Идем от конца к началу |
|  | rep movsw ; Переписываем символы из ds:si в es:di (si - предпоследний элемент змейки, di - последний элемент) |
|  | ; Таким образом смещаем всю змейку на 1 шаг |
|  | ; |
|  | mov bx, snakeBody ; Загружаем в bx позицию головы змейки |
|  | ; |
|  | add bh, Bmoveright ; Обновляем координаты головы |
|  | add bl, Bmovedown ; |
|  | mov snakeBody, bx ; сохраняем новую позицию головы |
|  | ; |
|  | ; теперь все тело в памяти сдвинуто ; |
|  | pop es ; |
|  | pop di ; |
|  | pop si ; |
|  | pop cx ; Восстанавливаем значения регистров |
|  | pop bx ; |
|  | pop ax ; |
|  | ret ; |
|  | ENDP ; |
|  |  |
|  | mainGame PROC |
|  | push ax ; |
|  | push bx ; |
|  | push cx ; |
|  | push dx ; Сохраняем значения регистров |
|  | push ds ; |
|  | push es ; |
|  | ; |
|  | checkAndMoveLoop: ; |
|  | ; |
|  | CheckBuffer ; Проверяем - был ли введен символ |
|  | jnz skipJmp2 ; Если да - skipJmp2 |
|  | jmp far ptr noSymbolInBuff ; Иначе noSymbolInBuff |
|  | ; |
|  | skipJmp2: ; |
|  | ReadFromBuffer ; Считываем символ из буффера |
|  | ; |
|  | cmp ah, KExit ; Если была нажата кнопка выхода |
|  | jne skipJmp ; Иначе skipJmp |
|  | ; |
|  | jmp far ptr esc\_exit ; Заканчиваем игру, прыгая в endLoop |
|  | ; |
|  | skipJmp: ; |
|  | cmp ah, KMoveLeft ; Если была нажата кнопка "влево" |
|  | je setMoveLeft ; |
|  | ; |
|  | cmp ah, KMoveRight ; Если была нажата кнопка "вправо" |
|  | je setMoveRight ; |
|  | ; |
|  | cmp ah, KMoveUp ; Если была нажата кнопка "вверх" |
|  | je setMoveUp ; |
|  | ; |
|  | cmp ah, KMoveDown ; Если была нажата кнопка "вниз" |
|  | je setMoveDown ; |
|  | ; |
|  | cmp ah, KUpSpeed ; move up key is pressed |
|  | je setSpeedUp ; |
|  | ; |
|  | cmp ah, KDownSpeed ; move down key is pressed |
|  | je setSpeedDown ; |
|  | ; |
|  | jmp noSymbolInBuff ; |
|  | ; |
|  | setMoveLeft: ; |
|  | mov al, Bmoveright ; Проверка на попытку изменения направления на противоположное |
|  | cmp al, forwardVal ; |
|  | jne setMoveLeft\_ok ; |
|  | jmp noSymbolInBuff ; |
|  | ; |
|  | setMoveLeft\_ok: ; |
|  | ; |
|  | mov Bmoveright, backwardVal ; Направление вправо - отрицательное |
|  | mov Bmovedown, stopVal ; Направление вниз - нулевое |
|  | jmp noSymbolInBuff ; |
|  | ; |
|  | setMoveRight: ; |
|  | mov al, Bmoveright ; Проверка на попытку изменения направления на противоположное |
|  | cmp al, backwardVal ; |
|  | jne setMoveRight\_ok ; |
|  | jmp noSymbolInBuff ; |
|  | ; |
|  | setMoveRight\_ok: ; |
|  | ; |
|  | mov Bmoveright, forwardVal ; Направление вправо - положительное |
|  | mov Bmovedown, stopVal ; Направление вправо - нулевое |
|  | jmp noSymbolInBuff ; |
|  | ; |
|  | setMoveUp: ; |
|  | mov al, Bmovedown ; Проверка на попытку изменения направления на противоположное |
|  | cmp al, forwardVal ; |
|  | jne setMoveUp\_ok ; |
|  | jmp noSymbolInBuff ; |
|  | ; |
|  | setMoveUp\_ok: ; |
|  | ; |
|  | mov Bmoveright, stopVal ; Направление вниз - отрицательное |
|  | mov Bmovedown, backwardVal ; |
|  | jmp noSymbolInBuff ; |
|  | ; |
|  | setMoveDown: ; |
|  | mov al, Bmovedown ; Проверка на попытку изменения направления на противоположное |
|  | cmp al, backwardVal ; |
|  | jne setMoveDown\_ok ; |
|  | jmp noSymbolInBuff ; |
|  | ; |
|  | setMoveDown\_ok: ; |
|  | ; |
|  | mov Bmoveright, stopVal ; Направление вправо - нулевое |
|  | mov Bmovedown, forwardVal ; Направление вниз - положительное |
|  | jmp noSymbolInBuff ; |
|  | ; |
|  | setSpeedUp: ; |
|  | mov ax, waitTime ; Загружаем в ax значение задержки |
|  | cmp ax, minWaitTime ; Сравниваем его с минимальным |
|  | je noSymbolInBuff ; Если равно минимальному - пропускаем |
|  | ; |
|  | sub ax, deltaTime ; Уменьшаем время задержки |
|  | mov waitTime, ax ; Обновляем значение задержки |
|  | ; |
|  | mov es, videoStart ; |
|  | mov di, offset speed - offset screen ; |
|  | mov ax, es:[di] ; |
|  | inc ax ; |
|  | mov es:[di], ax ; |
|  | ; |
|  | jmp noSymbolInBuff ; |
|  | ; |
|  | setSpeedDown: ; |
|  | mov ax, waitTime ; |
|  | cmp ax, maxWaitTime ; |
|  | je noSymbolInBuff ; |
|  | ; |
|  | add ax, deltaTime ; |
|  | mov waitTime, ax ; |
|  | ; |
|  | mov es, videoStart ; |
|  | mov di, offset speed - offset screen ; |
|  | mov ax, es:[di] ; |
|  | dec ax ; |
|  | mov es:[di], ax ; |
|  | ; |
|  | jmp noSymbolInBuff ; |
|  | ; |
|  | noSymbolInBuff: ; |
|  | call MoveSnake ; Передвигаем змейку на экране |
|  | ; |
|  | mov bx, snakeBody ; В помещаем в bx голову змеи |
|  | checkSymbolAgain: ; |
|  | call CalcOffsetByPoint ; В bx теперь смещение ячейки консоли с новой головой змейки |
|  | ; |
|  | mov es, videoStart ; Загружаем в es смещение видеобуффера |
|  | mov ax, es:[bx] ; Загружаем в ax символ куда должна стать змейка |
|  | ; |
|  | cmp ax, appleSymbol ; Если этот символ - яблоко |
|  | je AppleIsNext ; |
|  | ; |
|  | cmp ax, snakeBodySymbol ; Если этот символ - тело змейки |
|  | je SnakeIsNext ; |
|  | ; |
|  | cmp ax, HWallSymbol ; Если этот символ - горизонтальная стена |
|  | je PortalUpDown ; |
|  | ; |
|  | cmp ax, VWallSymbol ; Если этот символ - верникальная стена |
|  | je PortalLeftRight ; |
|  | ; |
|  | cmp ax, BWallSymbol ; Если этот символ - горизонтальная стена |
|  | je SnakeIsNext ; |
|  | ; |
|  | cmp ax, VWallSpecialSymbol ; |
|  | je PortalLeftRight ; |
|  | ; |
|  | jmp GoNextIteration ; |
|  | ; |
|  | AppleIsNext: ; |
|  | call destroyWall |
|  | call incSnake ; Увеличиваем длину змейки |
|  | call GenerateRandomApple ; Генерируем новое яблоко |
|  | call incScore ; Увеличиваем счет |
|  | jmp GoNextIteration ; Переходим к следующей итерации |
|  | SnakeIsNext: ; |
|  | jmp endLoop ; Заканчиваем игру |
|  | PortalUpDown: ; |
|  | mov bx, snakeBody ; Загружаем в bx голову змейки |
|  | sub bl, yField ; Отнимаем от y координаты высоту консоли |
|  | cmp bl, 0 ; Определяем верхняя это или нижняя граница |
|  | jg writeNewHeadPos ; Перерисовываем голову змейки |
|  | ; |
|  | ; Если это была верхняя стена |
|  | add bl, yField\*2 ; Корректируем координаты |
|  | ; |
|  | writeNewHeadPos: ; |
|  | mov snakeBody, bx ; Записываем новое значение головы |
|  | jmp checkSymbolAgain ; и отправляем его заново на сравнение |
|  | ; |
|  | PortalLeftRight: ; |
|  | mov bx, snakeBody ; |
|  | sub bh, xField ; |
|  | cmp bh, 0 ; |
|  | jg writeNewHeadPos ; Аналогично обрабатываем случай с вертикальной стеной |
|  | ; |
|  | add bh, xField\*2 ; |
|  | jmp writeNewHeadPos ; |
|  | ; |
|  | GoNextIteration: ; |
|  | mov bx, snakeBody ; Загружаем в bx новое начало змейки |
|  | call CalcOffsetByPoint ; Вычисляем ее позицию |
|  | mov di, bx ; Теперь в di смещение позиции bx в консоли |
|  | mov ax, snakeBodySymbol ; Записываем в ax символ змейки |
|  | stosw ; Записываем в консоль |
|  | ; |
|  | call Sleep ; Задержка |
|  | ; |
|  | jmp checkAndMoveLoop ; |
|  | ; |
|  | endLoop: ; |
|  | pop es ; |
|  | pop ds ; |
|  | pop dx ; Восстанавливаем значения регистров |
|  | pop cx ; |
|  | pop bx ; |
|  | pop ax ; |
|  | ret ; |
|  | ENDP ; |
|  | ; |
|  | Sleep PROC ; |
|  | push ax ; |
|  | push bx ; Сохраняем регистры |
|  | push cx ; |
|  | push dx ; |
|  | ; |
|  | GetTimerValue ; Получаем текущее значение времени |
|  | ; |
|  | add dx, waitTime ; Добавляем к dx значение задержки |
|  | mov bx, dx ; Загружаем его в bx |
|  | ; |
|  | checkTimeLoop: ; |
|  | GetTimerValue ; Получаем текузее значение времени |
|  | cmp dx, bx ; ax - current value, bx - needed value |
|  | jl checkTimeLoop ; Если еще рано - уходим на следующую итерацию |
|  | ; |
|  | pop dx ; |
|  | pop cx ; |
|  | pop bx ; Восстанавливаем значения регистров |
|  | pop ax ; |
|  | ret ; |
|  | ENDP ; |
|  |  |
|  | GenerateRandomApple PROC ; |
|  | push ax ; |
|  | push bx ; |
|  | push cx ; Сохраняем значения регистров |
|  | push dx ; |
|  | push es ; |
|  | ; |
|  | mov ah, 2Ch ; Считываем текущее время |
|  | int 21h ; ch - час, cl - минуты, dh - секунды, dl - мсек |
|  |  |
|  | mov al, dl |
|  | mul dh ; Теперь в ax число для рандома |
|  |  |
|  | xor dx, dx |
|  |  |
|  | mov cx, 04h |
|  | div cx |
|  | mov bh, dl |
|  |  |
|  | cmp bh, 0 |
|  | jne rnd1 |
|  | mov si, offset brickWall1 |
|  | jmp writeToTemplate |
|  |  |
|  | rnd1: |
|  |  |
|  | cmp bh, 1 |
|  | jne rnd2 |
|  | mov si, offset brickWall2 |
|  | jmp writeToTemplate |
|  |  |
|  | rnd2: |
|  |  |
|  | cmp bh, 2 |
|  | jne rnd3 |
|  | mov si, offset brickWall3 |
|  | jmp writeToTemplate |
|  |  |
|  | rnd3: |
|  |  |
|  | mov si, offset brickWall4 |
|  | jmp writeToTemplate |
|  |  |
|  | writeToTemplate: |
|  | mov di, offset brickWallTemplate |
|  | mov cx, brickWallSize |
|  |  |
|  | toTemplate: |
|  | push ax |
|  | mov ax, [si] |
|  | mov [di],ax |
|  | pop ax |
|  |  |
|  | add di, PointSize |
|  | add si, PointSize |
|  | loop toTemplate |
|  |  |
|  | loop\_random: ; |
|  | mov ah, 2Ch ; Считываем текущее время |
|  | int 21h ; ch - час, cl - минуты, dh - секунды, dl - мсек |
|  | ; |
|  | mov al, dl ; Получаем случайное число |
|  | mul dh ; Теперь в ax число для рандома |
|  | ; |
|  | xor dx, dx ; Обнуляем dx |
|  | mov cx, xField ; В cx загружаем ширину поля |
|  | div cx ; Получаем номер строки яблока |
|  | add dx, 2 ; Добавляем смещение от начала оси |
|  | mov bh, dl ; Сохраняем координату x |
|  | ; |
|  | xor dx, dx ; |
|  | mov cx, yField ; |
|  | div cx ; Аналогично получаем y координату |
|  | add dx, 2 ; |
|  | mov bl, dl ; Теперь в bx находится координата яблока |
|  | ; |
|  | push bx |
|  | call CalcOffsetByPoint; Расситываем смещение |
|  | mov es, videoStart ; Загружаем в es начало видеобуффера |
|  | mov ax, es:[bx] ; В ax загружаем символ, который расположен по координатам, в которых мы хотим расположить яблоко |
|  | pop bx |
|  | ; |
|  | cmp ax, space ; Сравниваем их с пробелом(т.е. пустой клеткой). |
|  | jne loop\_random ; Если в клетке что-то есть - генерируем новые координаты |
|  |  |
|  | mov cx, brickWallSize |
|  | mov si, offset brickWallTemplate |
|  | loopRandomWall: |
|  | push bx ; Цикл, в котором мы выводим тело змейки |
|  | add bx, [si] ; Загружаем в si очередной символ тела змейки |
|  |  |
|  | push bx |
|  | call CalcOffsetByPoint; Расситываем смещение |
|  | mov es, videoStart ; Загружаем в es начало видеобуффера |
|  | mov ax, es:[bx] ; В ax загружаем символ, который расположен по координатам, в которых мы хотим расположить яблоко |
|  | pop bx |
|  |  |
|  | pop bx |
|  |  |
|  | cmp ax, space |
|  |  |
|  | jne loop\_random |
|  |  |
|  | add si, PointSize ; Добавляем к si PointSize, т.е. 2, т.к. каждая точка занимает 2 байта (цвет + символ) |
|  | loop loopRandomWall |
|  |  |
|  | mov cx, brickWallSize |
|  | mov si, offset brickWallTemplate |
|  | mov di, offset brickWallTrue |
|  | loopCreateWall: |
|  | push ax ; Цикл, в котором мы выводим тело змейки |
|  | mov ax, [si] ; Загружаем в si очередной символ тела змейки |
|  | add ax, bx |
|  | mov [di], ax |
|  |  |
|  | add si, PointSize |
|  | add di, PointSize |
|  | pop ax ; Выводим |
|  | loop loopCreateWall |
|  |  |
|  | call drawBrickWall |
|  |  |
|  | push bx |
|  | call CalcOffsetByPoint; Расситываем смещение |
|  | mov es, videoStart ; Загружаем в es начало видеобуффера |
|  | mov ax, appleSymbol; |
|  | mov es:[bx], ax ; Выводим символ яблока |
|  | pop bx |
|  | ; |
|  | pop es ; |
|  | pop dx ; |
|  | pop cx ; Восстанавливаем регистры |
|  | pop bx ; |
|  | pop ax ; |
|  | ret ; |
|  | ENDP ; |
|  |  |
|  | ;save tail of snake if no overloading |
|  | incSnake PROC ; |
|  | push ax ; |
|  | push bx ; Сохраняем значения регистров |
|  | push di ; |
|  | push es ; |
|  | ; |
|  | mov al, snakeSize ; Загружаем в ax текущий размер змейки |
|  | cmp al, snakeMaxSize ; Сравниваем его с макисимальным размером змейки |
|  | je return ; Если достигли максимума - выходим |
|  | ; |
|  | ; Увеличиваем длину змейки в массиве |
|  | inc al ; Увеличиваем al на 1 |
|  | mov snakeSize, al ; Обновляем размер змейки |
|  | dec al ; Уменьшаем al на 1. Для дальнейшей работы удобнее старая длина змейки |
|  | ; |
|  | ; |
|  | mov bl, PointSize ; Восстанавливаем конец |
|  | mul bl ; Получили в ax нужное для восстановления смещение |
|  | ; |
|  | mov di, offset snakeBody |
|  | add di, ax ; di теперь укаывает на точку для восстановления |
|  | ; |
|  | mov es, dataStart ; Загружаем в es данные |
|  | mov bx, es:[di] ; Загружаем в bx восстанавливаемую точку |
|  | call CalcOffsetByPoint; Получаем ее координаты |
|  | ; |
|  | mov es, videoStart ; Загружаем в es смещение видеобуффера |
|  | mov es:[bx], snakeBodySymbol ; Записываем в точку символ тела змейки |
|  | ; |
|  | return: ; |
|  | pop es ; |
|  | pop di ; Восстанавливаем значения регистров |
|  | pop bx ; |
|  | pop ax ; |
|  | ret ; |
|  | ENDP ; |
|  | ; |
|  | incScore PROC ; |
|  | push ax ; |
|  | push es ; |
|  | push si ; |
|  | push di ; |
|  | mov es, videoStart ; |
|  | mov cx, scoreSize ; ;max pos value |
|  | mov di, offset score + (scoreSize - 1)\*oneMemoBlock - offset screen ;???????? ???????? ?????????? ??????? ????? |
|  | ; |
|  | loop\_score: ; |
|  | mov ax, es:[di] ; |
|  | cmp al, 39h ;'9' symbol |
|  | jne nineNotNow ; |
|  | ; |
|  | sub al, 9 ; |
|  | mov es:[di], ax ; |
|  | ; |
|  | sub di, oneMemoBlock ;return to symbol back |
|  | ; |
|  | loop loop\_score ; |
|  | jmp return\_incScore ; |
|  | ; |
|  | nineNotNow: ; |
|  | inc ax ; |
|  | mov es:[di], ax ; |
|  | return\_incScore: ; |
|  | pop di ; |
|  | pop si ; |
|  | pop es ; |
|  | pop ax ; |
|  | ret ; |
|  | ENDP ; |
|  | end main ; |