Министерство образования Республики Беларусь

Учреждение образования Белорусский государственный университет информатики и радиоэлектроники

Кафедра ЭВМ

Отчёт по лабораторной работе №4

“Создание видеоигры”

Проверил: Выполнил:

ассистент каф. ЭВМ студент гр.030501

Туровец Николай Олегович Вайтехович П.В.

Минск 2022

**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.

**Алгоритм**

* использование прямого доступа к памяти
* обработка нажатия кнопок клавиатуры
* доступ к системным часам
* создание управления курсором

Кроме использования прерываний DOS, описанных в лабораторной работе №2, программа может выводить текст на экран с помощью пересылки данных в специальную область памяти, связанную с видеоадаптером – видеопамять.

Этот вариант вывода более быстр, чем при выводе символов через прерывания, а также позволяет формировать в консоли определенные эффекты, часто не используемые в режиме вывода в позицию курсора. В большинстве текстовых видеорежимов под видеопамять отводится специальная область памяти, начинающаяся с абсолютного адреса B800h:0000h и заканчивающаяся адресом B800h:FFFFh.

Все, что программа запишет в эту область памяти, будет пересылаться в память видеоадаптера и отображаться на экране. В текстовых режимах для хранения каждого изображенного символа ис?пользуются два байта. Для установки требуемого программе видеорежима используется прерывание 10h (видеосервис) BIOS. Видеорежимы отличаются друг от друга разрешением (для графических) и количеством строк и столбцов (для текстовых), а также количеством возможных цветов. В данной лабораторной работе использование графических режимов видеоадаптера не требуется, поэтому в описании прерываний эта информация будет опущена. Курсор не является символом из набора ASCII-кодов.

Компьютер имеет собственное аппаратное обеспечение для управления видом курсора. Обычно символ курсора похож на символ подчеркивания и всегда мерцает. Обработка нажатий на клавиатуру может производиться различными способами: -- с помощью прерываний ввода символов DOS; -- с помощью прерываний ввода символов BIOS; -- с помощью прямого доступа к буферу клавиатуры -- с помощью доступа к портам ввода-вывода клавиатуры. Ввод символов с помощью функций прерывания DOS 21h рассмотрен ранее в лабораторной работе №2. По сравнению с функциями DOS, прерывание BIOS 16h предоставляет больше возможностей для считывания данных и управления клавиатурой и такой доступ практически эквивалентен по производительности прямому доступу к буферу клавиатуры. Персональный компьютер содержит два устройства для управления про- 33 цессами: -- часы реального времени (RTC) – имеют автономное питание, используются для чтения/установки текущих даты и времени, установки будильника и для вызова прерывания IRQ8 (INT 4Ah) каждую миллисекунду; -- системный таймер – используется одновременно для управления контроллером прямого доступа к памяти, для управления динамиком и как генератор импульсов, вызывающий прерывание IRQ0 (INT 8h) 18,2 раза в секунду. Для видеоигры, создаваемой в данной лабораторной работе, указанные выше устройства лучше всего использовать на уровне функций DOS или BIOS как средство для определения текущего времени, организации задержек и формирования случайных чисел.

**Листинг программы**

clearScreen MACRO

push ax ; save ax

mov ax, 0003h ; 00 - set a video mode, clear window. 03h - mode 80x25

int 10h ; Calling an interrupt to execute a command

pop ax ; Restoring the value of the ax register

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 ; width of conole

ySize equ 25 ; height of console

xField equ 50 ; width of field

yField equ 21 ; height of field

oneMemoBlock equ 2 ; size of console cell

scoreSize equ 4 ; length of the reference block

;

videoStart dw 0B800h ; video buffer offset

dataStart dw 0000h ;

timeStart dw 0040h ;

timePosition dw 006Ch ;

;

space equ 0020h ; an empty block with a black background

snakeBodySymbol equ 0A40h ; snake body symbol

appleSymbol equ 0B0Fh ; apple symbol

VWallSymbol equ 0FBAh ; vertical wall symbol

HWallSymbol equ 0FCDh ; horizontal wall symbol

BWallSymbol equ 4020h ;

VWallSpecialSymbol equ 0FCCh ; the symbol of the crossing of the walls

fieldSpacingBad equ space, VWallSymbol, xField dup(space)

fieldSpacing equ fieldSpacingBad, VWallSymbol

rbSym equ 077DCh ; white block with black background

rbSpc equ 04F20h ; whitespace with red background and white color of symbols

ylSym equ 06FDCh ; white block with yellow background

ylSpc equ 06F20h ; whitespace with yellow background

grSym equ 02FDBh ; white block with green background

grSpc equ 02F20h ; empty block with white background

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)

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 ; loading the initial data

mov ax, videoStart ; we upload the code of the beginning of the output to the video buffer to ax

mov es, ax ; loading ax in es

xor ax, ax ; make es null

;

clearScreen ; clear the console

;

call initAllScreen ; initializing window

;

call mainGame ; we go into the main cycle of the game

;

to\_close: ;

call printBanner ;

mov ah,7h ; 7h - console input without echo (waiting for a key to exit the application)

int 21h

int 21h ;

esc\_exit:

clearScreen

mov ah, 4ch

int 21h

;AH = scan-code

CheckBuffer MACRO

mov ah, 01h

int 16h

ENDM

ReadFromBuffer MACRO

mov ah, 00h

int 16h

ENDM

;Ðåçóëüòàò â cx:dx

GetTimerValue MACRO

push ax

mov ax, 00h

int 1Ah

pop 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]

add si, PointSize

call CalcOffsetByPoint

mov di, bx

mov ax, BWallSymbol

stosw

loop loopBrickWall

pop bx

pop cx

ret

ENDP

destroyWall PROC

push cx

mov cx, brickWallSize

mov si, offset brickWallTrue

loopDestroyWall:

mov bx, [si]

add si, PointSize

call CalcOffsetByPoint

mov di, bx

mov ax, space

stosw

loop loopDestroyWall

pop cx

ret

ENDP

initAllScreen PROC

mov si, offset screen

xor di, di

mov cx, xSize\*ySize

rep movsw

xor ch, ch

mov cl, snakeSize

mov si, offset snakeBody

loopInitSnake:

mov bx, [si]

add si, PointSize

call CalcOffsetByPoint

mov di, bx

mov ax, snakeBodySymbol

stosw

loop loopInitSnake

call GenerateRandomApple

ret

ENDP

CalcOffsetByPoint PROC

push ax

push dx

xor ah, ah

mov al, bl

mov dl, xSize

mul dl

mov dl, bh

xor dh, dh

add ax, dx

mov dx, oneMemoBlock

mul dx

mov bx, ax

pop dx

pop ax

ret

ENDP

MoveSnake PROC

push ax

push bx

push cx

push si

push di

push es

mov al, snakeSize

xor ah, ah

mov cx, ax

mov bx, PointSize

mul bx

mov di, offset snakeBody

add di, ax

mov si, di

sub si, PointSize

push di

mov es, videoStart

mov bx, ds:[si]

call CalcOffsetByPoint

mov di, bx

mov ax, space

stosw

pop di

mov es, dataStart

std

rep movsw

mov bx, snakeBody

add bh, Bmoveright

add bl, Bmovedown

mov snakeBody, bx

pop es

pop di

pop si

pop cx

pop bx

pop ax

ret

ENDP

mainGame PROC

push ax

push bx

push cx

push dx

push ds

push es

checkAndMoveLoop:

CheckBuffer

jnz skipJmp2

jmp far ptr noSymbolInBuff

skipJmp2:

ReadFromBuffer

cmp ah, KExit

jne skipJmp

jmp far ptr esc\_exit

skipJmp:

cmp ah, KMoveLeft

je setMoveLeft

cmp ah, KMoveRight

je setMoveRight

cmp ah, KMoveUp

je setMoveUp

cmp ah, KMoveDown

je setMoveDown

cmp ah, KUpSpeed

je setSpeedUp

cmp ah, KDownSpeed

je setSpeedDown

jmp noSymbolInBuff

setMoveLeft:

mov al, Bmoveright

cmp al, forwardVal

jne setMoveLeft\_ok

jmp noSymbolInBuff

setMoveLeft\_ok:

mov Bmoveright, backwardVal

mov Bmovedown, stopVal

jmp noSymbolInBuff

setMoveRight:

mov al, Bmoveright

cmp al, backwardVal

jne setMoveRight\_ok

jmp noSymbolInBuff

setMoveRight\_ok:

mov Bmoveright, forwardVal

mov Bmovedown, stopVal

jmp noSymbolInBuff

setMoveUp:

mov al, Bmovedown

cmp al, forwardVal

jne setMoveUp\_ok

jmp noSymbolInBuff

setMoveUp\_ok:

mov Bmoveright, stopVal

mov Bmovedown, backwardVal

jmp noSymbolInBuff

setMoveDown:

mov al, Bmovedown

cmp al, backwardVal

jne setMoveDown\_ok

jmp noSymbolInBuff

setMoveDown\_ok:

mov Bmoveright, stopVal

mov Bmovedown, forwardVal

jmp noSymbolInBuff

setSpeedUp:

mov ax, waitTime

cmp ax, minWaitTime

je noSymbolInBuff

sub ax, deltaTime

mov waitTime, ax

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

mov bx, snakeBody

checkSymbolAgain:

call CalcOffsetByPoint

mov es, videoStart

mov ax, es:[bx]

cmp ax, appleSymbol

je AppleIsNext

cmp ax, snakeBodySymbol

je SnakeIsNext

cmp ax, HWallSymbol

je PortalUpDown

cmp ax, VWallSymbol

je PortalLeftRight

cmp ax, BWallSymbol

je SnakeIsNext

cmp ax, VWallSpecialSymbol

je PortalLeftRight

jmp GoNextIteration

AppleIsNext:

call destroyWall

call incSnake

call GenerateRandomApple

call incScore

jmp GoNextIteration

SnakeIsNext:

jmp endLoop

PortalUpDown:

mov bx, snakeBody

sub bl, yField

cmp bl, 0

jg writeNewHeadPos

add bl, yField\*2

writeNewHeadPos:

mov snakeBody, bx

jmp checkSymbolAgain

PortalLeftRight:

mov bx, snakeBody

sub bh, xField

cmp bh, 0

jg writeNewHeadPos

add bh, xField\*2

jmp writeNewHeadPos

GoNextIteration:

mov bx, snakeBody

call CalcOffsetByPoint

mov di, bx

mov ax, snakeBodySymbol

stosw

call Sleep

jmp checkAndMoveLoop

endLoop:

pop es

pop ds

pop dx

pop cx

pop bx

pop ax

ret

ENDP

Sleep PROC

push ax

push bx

push cx

push dx

GetTimerValue

add dx, waitTime

mov bx, dx

checkTimeLoop:

GetTimerValue

cmp dx, bx

jl checkTimeLoop

pop dx

pop cx

pop bx

pop ax

ret

ENDP

GenerateRandomApple PROC

push ax

push bx

push cx

push dx

push es

mov ah, 2Ch

int 21h

mov al, dl

mul dh

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

int 21h

mov al, dl

mul dh

xor dx, dx

mov cx, xField

div cx

add dx, 2

mov bh, dl

xor dx, dx

mov cx, yField

div cx

add dx, 2

mov bl, dl

push bx

call CalcOffsetByPoint

mov es, videoStart

mov ax, es:[bx]

pop bx

cmp ax, space

jne loop\_random

mov cx, brickWallSize

mov si, offset brickWallTemplate

loopRandomWall:

push bx

add bx, [si]

push bx

call CalcOffsetByPoint

mov es, videoStart

mov ax, es:[bx]

pop bx

pop bx

cmp ax, space

jne loop\_random

add si, PointSize

loop loopRandomWall

mov cx, brickWallSize

mov si, offset brickWallTemplate

mov di, offset brickWallTrue

loopCreateWall:

push ax

mov ax, [si]

add ax, bx

mov [di], ax

add si, PointSize

add di, PointSize

pop ax

loop loopCreateWall

call drawBrickWall

push bx

call CalcOffsetByPoint

mov es, videoStart

mov ax, appleSymbol;

mov es:[bx], ax

pop bx

pop es

pop dx

pop cx

pop bx

pop ax

ret

ENDP

;save tail of snake if no overloading

incSnake PROC

push ax

push bx

push di

push es

mov al, snakeSize

cmp al, snakeMaxSize

je return

inc al

mov snakeSize, al

dec al

mov bl, PointSize

mul bl

mov di, offset snakeBody

add di, ax

mov es, dataStart

mov bx, es:[di]

call CalcOffsetByPoint

mov es, videoStart

mov es:[bx], snakeBodySymbol

return:

pop es

pop di

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

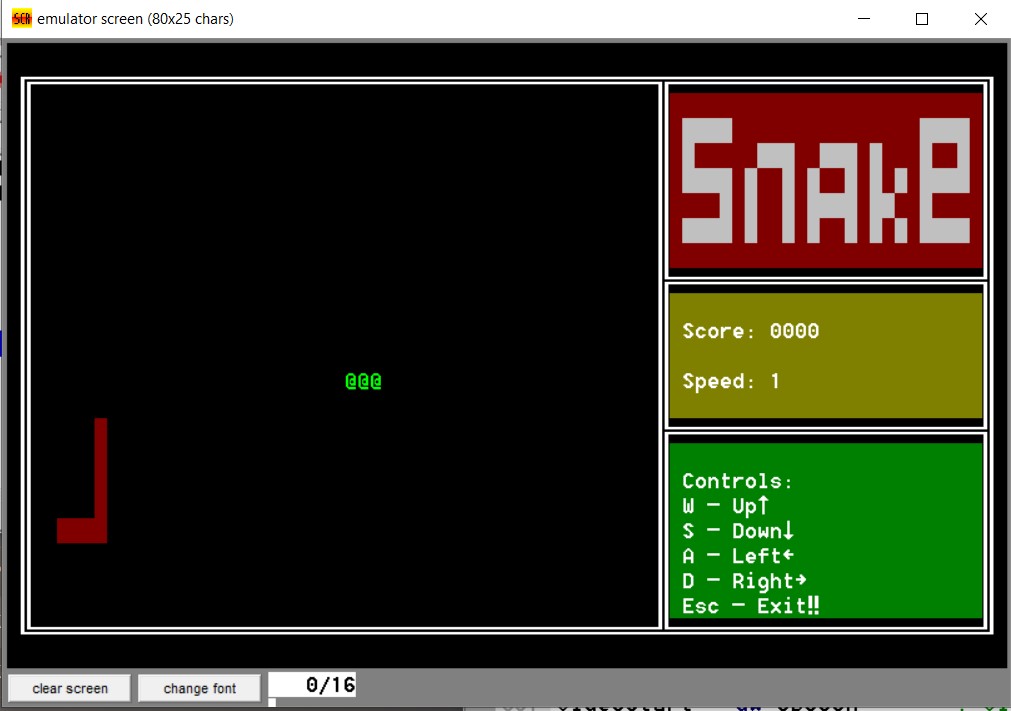
pop es

pop ax

ret

ENDP

end main

**Тест**

*Рисунок 1 –* результат выполнения программы

**Вывод:** В данной лабораторной работе разработана игра «змейка».