Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Отчет

Лабораторная работа №4

Выполнил студент гр. 150505: Лисичкин Д.А.

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.

**Вариант лабораторной работы:** Игра «Пакман». Цель: ходим по лабиринту и собираем случайно появляющиеся яблочки, убегаем от противника (один вид, 2 или 3 противника). Окончание: проигрыш – столкновение с противником, выигрыш – нет. Информация: счет.

**Код программы**

.model small

.stack 100h

.data

windiwSizeX equ 40

windiwSizeY equ 25

fieldSize equ 21

fieldOffsetX equ 2

fieldOffsetY equ 2

fieldOffsetInMemory equ 2 \* 2 \* (fieldOffsetY \* windiwSizeX + fieldOffsetX)

w equ 1110011000100001b

g equ 0000000000100001b

groundBlock dw 2 dup(g)

gameField dw fieldSize dup(w)

dw w,g,g,g,g,g,g,g,g,g,g,g,w,g,g,g,g,g,g,g,w

dw w,g,w,g,w,w,w,g,w,w,w,g,w,g,w,w,w,g,w,g,w

dw w,g,w,g,g,g,g,g,w,g,g,g,g,g,g,g,w,g,g,g,w

dw w,g,w,g,w,g,w,w,w,g,w,w,w,g,w,g,w,g,w,w,w

dw w,g,g,g,w,g,g,g,g,g,w,g,g,g,g,g,w,g,g,g,w

dw w,w,w,g,w,g,w,g,w,g,w,g,w,w,w,g,w,w,w,g,w

dw w,g,g,g,g,g,w,g,w,g,g,g,g,g,g,g,g,g,g,g,w

dw w,g,w,g,w,w,w,g,w,g,w,g,w,w,w,w,w,g,w,g,w

dw w,g,w,g,g,g,w,g,g,g,w,g,w,g,g,g,g,g,w,g,w

dw w,g,w,w,w,g,w,g,w,w,w,g,w,g,w,g,w,g,w,g,w

dw w,g,g,g,g,g,g,g,g,g,g,g,g,g,w,g,g,g,w,g,w

dw w,g,w,g,w,w,w,g,w,g,w,w,w,g,w,w,w,g,w,g,w

dw w,g,g,g,g,g,g,g,w,g,g,g,w,g,g,g,w,g,g,g,w

dw w,w,w,g,w,g,w,g,w,g,w,g,w,g,w,g,w,w,w,w,w

dw w,g,g,g,w,g,w,g,g,g,g,g,w,g,w,g,g,g,g,g,w

dw w,g,w,g,w,g,w,w,w,g,w,g,w,g,w,w,w,g,w,g,w

dw w,g,g,g,w,g,w,g,g,g,w,g,g,g,g,g,g,g,w,g,w

dw w,g,w,w,w,g,w,g,w,g,w,g,w,w,w,g,w,g,w,g,w

dw w,g,g,g,g,g,g,g,w,g,g,g,g,g,g,g,w,g,g,g,w

dw fieldSize dup (w)

scoreOffsetX equ 31

scoreOffsetY equ 8

scoreOffsetInMemory equ 2 \* 2 \* (scoreOffsetY \* windiwSizeX + scoreOffsetX)

scoreMessage db 'Score: '

scoreMessageOffsetX equ 30

scoreMessageOffsetY equ 6

scoreMessageOffsetInMemory equ 2 \* 2 \* (scoreMessageOffsetY \* windiwSizeX + scoreMessageOffsetX)

gameOverMessage db 'Game over!'

gameOverMessageOffsetX equ 29

gameOverMessageOffsetY equ 12

gameOverMessageOffsetInMemory equ 2 \* 2 \* (gameOverMessageOffsetY \* windiwSizeX + gameOverMessageOffsetX)

random db 010010111b

entropySourceWeak db ?

entropySourceStrong db ?

gameLoopBigPause equ 1

gameLoopSmallPause equ 0

ghostsCount equ 4

ghostsMaxDelay equ 3

ghostsDelayCounter db ?

ghostsX db ghostsCount dup(?)

ghostsY db ghostsCount dup(?)

ghostsDirection db ghostsCount dup(?)

ghostsColor db ghostsCount dup(?)

ghostBlue dw 0000100100100000b, 0000100100000010b

ghostGreen dw 0000101000100000b, 0000101000000010b

ghostPurple dw 0000110100100000b, 0000110100000010b

ghostGray dw 0000011100100000b, 0000011100000010b

packmanMaxDelay equ 3

packmanDelayCounter db ?

packmanX db ?

packmanY db ?

packmanDirection db ?

packmanNextDirection db ?

packmanUp dw 0000111001011100b, 0000111000101111b

packmanDown dw 0000111000101111b, 0000111001011100b

packmanLeft dw 0000111000111110b, 0000111000101101b

packmanRight dw 0000111000101101b, 0000111000111100b

appleCount db 0

applesNumber equ 5

mem dw ?

a dw 0000110000101000b, 0000110000101001b

applesX db applesNumber dup(?)

applesY db applesNumber dup(?)

appleCountString dw 4 dup(?)

.code

;;;;;;;;;;;;;;;;;;;;;;;

drawField proc

push si

push di

push ax

push cx

mov si, offset gameField

mov di, fieldOffsetInMemory

mov cx, fieldSize

drawFieldLoopOnStrings:

push cx

mov cx, fieldSize

drawFieldLoopOnColumns:

push cx

mov ax, ds:[si]

mov cx, 2

drawFieldCellLoop:

mov word ptr es:[di], ax

add di, 2

loop drawFieldCellLoop

add si, 2

pop cx

loop drawFieldLoopOnColumns

add di, 2 \* 2 \* (windiwSizeX - fieldSize)

pop cx

loop drawFieldLoopOnStrings

pop cx

pop ax

pop di

pop si

ret

drawField endp

;;;;;;;;;;;;;;;;;;;;;;;

drawMessage proc

mov cx, 10

drawMessageLoop:

mov ah, 00001111b

mov al, [si]

mov word ptr es:[di], ax

inc si

add di, 2

loop drawMessageLoop

ret

drawMessage endp

drawScoreMessage proc

mov si, offset scoreMessage

mov di, scoreMessageOffsetInMemory

call drawMessage

ret

drawScoreMessage endp

drawGameOverScoreMessage proc

mov si, offset gameOverMessage

mov di, gameOverMessageOffsetInMemory

call drawMessage

ret

drawGameOverScoreMessage endp

;;;;;;;;;;;;;;;;;;;;;;;

clearScreen macro

push ax

mov ax, 0000 0000 0000 0003

int 10h

pop ax

endm

sleep macro

push ax

push cx

push dx

mov ah, 86h

mov cx, gameLoopBigPause \* 0001h

mov dx, gameLoopSmallPause \* 1000h

int 15h

pop dx

pop cx

pop ax

endm

isKeyPressed macro

push ax

mov ah, 01h

int 16h

pop ax

endm

getKey macro

mov ah, 00h

int 16h

endm

clearKeyboardBuffer macro

push ax

mov ax,0c00h

int 21h

pop ax

endm

;;;;;;;;;;;;;;;;;;;;;;;;;;;;

updateEntropy macro

push dx

push cx

mov ah, 2ch

int 21h

mov entropySourceWeak, dh

mov entropySourceStrong, dl

pop cx

pop dx

endm

updateRandomParameter macro number shift1 multiplier summand shift2

push ax

mov al, number

ror al, shift1

mov ah, multiplier

mul ah

add al, summand

ror al, shift2

mov number, al

pop ax

endm

updateRandom macro

updateRandomParameter random 2 23 11 5

updateRandomParameter entropySourceWeak 1 7 4 3

updateRandomParameter entropySourceStrong 7 5 8 4

updateRandomParameter random 6 entropySourceWeak entropySourceStrong 1

endm

getRandomNumber macro limit

push bx

push dx

updateRandom

xor ax, ax

mov al, random

xor bx, bx

mov bl, limit

cwd

div bx

mov ax, dx

pop dx

pop bx

endm

;;;;;;;;;;;;;;;;;;;;;;;

calculateObjectOffsetRelativeToField macro sizeX

xor bx, bx

mov bl, ah

mov ah, 0h

mov dx, sizeX

mul dx

add ax, bx

mov dx, 2 \* 2

mul dx

endm

drawObject macro object

push si

push di

push cx

push bx

push dx

calculateObjectOffsetRelativeToField windiwSizeX

mov si, offset object

mov di, fieldOffsetInMemory

add di, ax

mov cx, 2

rep movsw

pop dx

pop bx

pop cx

pop di

pop si

endm

getObject macro

push si

push bx

push dx

calculateObjectOffsetRelativeToField fieldSize

mov bx, 2

div bx

mov si, offset gameField

add si, ax

mov ax, [si]

pop dx

pop bx

pop si

endm

getObjectOnDirection proc

cmp bl, 0

je checkObjectUp

cmp bl, 1

je checkObjectDown

cmp bl, 2

je checkObjectLeft

cmp bl, 3

je checkObjectRight

checkObjectUp:

dec al

jmp getNeighborObject

checkObjectDown:

inc al

jmp getNeighborObject

checkObjectLeft:

dec ah

jmp getNeighborObject

checkObjectRight:

inc ah

jmp getNeighborObject

getNeighborObject:

getObject

ret

getObjectOnDirection endp

checkMeetingPackmanAndGhost proc

push di

push cx

push ax

mov cx, ghostsCount

mov di, 0

checkMeetingPackmanAndGhostLoop:

mov ah, ghostsX[di]

mov al, ghostsY[di]

checkMeetingPackmanAndGhostOnX:

cmp ah, packmanX

je checkMeetingPackmanAndGhostOnY

jmp checkNextMeetingPackmanAndGhost

checkMeetingPackmanAndGhostOnY:

cmp al, packmanY

je endGame

jmp checkNextMeetingPackmanAndGhost

checkNextMeetingPackmanAndGhost:

inc di

loop checkMeetingPackmanAndGhostLoop

pop ax

pop cx

pop di

ret

checkMeetingPackmanAndGhost endp

checkMeetingPackmanAndApple proc

push ax

push di

push cx

mov cx, applesNumber

mov di, 0

checkMeetingPacmanApple:

checkMeetingPackmanAndAppleOnX:

mov ah, applesX[di]

mov al, packmanX

cmp ah, al

je checkMeetingPackmanAndAppleOnY

jmp checkMeetingPackmanNextApple

checkMeetingPackmanAndAppleOnY:

mov ah, applesY[di]

mov al, packmanY

cmp ah, al

je incAppleCount

jmp checkMeetingPackmanNextApple

checkMeetingPackmanNextApple:

inc di

loop checkMeetingPacmanApple

jmp endCheckMeetingPackmanAndApple

incAppleCount:

inc appleCount

call createApple

endCheckMeetingPackmanAndApple:

pop cx

pop di

pop ax

ret

checkMeetingPackmanAndApple endp

checkMeetingGhostAndApple proc

push ax

push di

push cx

mov cx, applesNumber

mov di, 0

chekMeetingGhostApple:

checkMeetingGhostAndAppleOnX:

mov ah, applesX[di]

mov al, ghostsX[si]

cmp ah, al

je checkMeetingGhostAndAppleOnY

jmp chekMeetingGhostNextApple

checkMeetingGhostAndAppleOnY:

mov ah, applesY[di]

mov al, ghostsY[si]

cmp ah, al

je redrawApple

jmp chekMeetingGhostNextApple

chekMeetingGhostNextApple:

inc di

loop chekMeetingGhostApple

jmp endCheckMeetingGhostAndApple:

redrawApple:

call drawApple

endCheckMeetingGhostAndApple:

pop cx

pop di

pop ax

ret

checkMeetingGhostAndApple endp

;;;;;;;;;;;;;;;;;;;;;;

createPackman proc

mov packmanDelayCounter, 0

mov packmanX, 1

mov packmanY, 1

mov packmanDirection, 2

mov packmanNextDirection, 2

call drawPackman

ret

createPackman endp

checkKeystroke proc

isKeyPressed

jnz keyWasPressed

ret

keyWasPressed:

getKey

clearKeyboardBuffer

cmp al, 'w'

je changePackmanNextDirectionOnUp

cmp al, 's'

je changePackmanNextDirectionOnDown

cmp al, 'a'

je changePackmanNextDirectionOnLeft

cmp al, 'd'

je changePackmanNextDirectionOnRight

;nazhato ne to

ret

changePackmanNextDirectionOnUp:

mov packmanNextDirection, 0

ret

changePackmanNextDirectionOnDown:

mov packmanNextDirection, 1

ret

changePackmanNextDirectionOnLeft:

mov packmanNextDirection, 2

ret

changePackmanNextDirectionOnRight:

mov packmanNextDirection, 3

ret

checkKeystroke endp

erasePackman proc

mov ah, packmanX

mov al, packmanY

drawObject groundBlock

ret

erasePackman endp

changePackmanCoordinate proc

cmp packmanDirection, 0

je movePackmanUp

cmp packmanDirection, 1

je movePackmanDown

cmp packmanDirection, 2

je movePackmanLeft

cmp packmanDirection, 3

je movePackmanRight

movePackmanUp:

dec packmanY

ret

movePackmanDown:

inc packmanY

ret

movePackmanLeft:

dec packmanX

ret

movePackmanRight:

inc packmanX

ret

changePackmanCoordinate endp

drawPackman proc

mov ah, packmanX

mov al, packmanY

cmp packmanDirection, 0

je drawPackmanUp

cmp packmanDirection, 1

je drawPackmanDown

cmp packmanDirection, 2

je drawPackmanLeft

cmp packmanDirection, 3

je drawPackmanRight

drawPackmanUp:

drawObject packmanUp

jmp endDrawPackman

drawPackmanDown:

drawObject packmanDown

jmp endDrawPackman

drawPackmanLeft:

drawObject packmanLeft

jmp endDrawPackman

drawPackmanRight:

drawObject packmanRight

jmp endDrawPackman

endDrawPackman:

ret

drawPackman endp

movePackman proc

push ax

push bx

call checkKeystroke

checkPackmanDelayCounter:

inc packmanDelayCounter

cmp packmanDelayCounter, packmanMaxDelay

jne endMovePackman

mov packmanDelayCounter, 0

checkNextDirection:

mov ah, packmanX

mov al, packmanY

mov bl, packmanNextDirection

call getObjectOnDirection

cmp ax, g

je setPackmanDirectionOnNext

cmp ax, w

je checkCurrentDirection

checkCurrentDirection:

mov ah, packmanX

mov al, packmanY

mov bl, packmanDirection

call getObjectOnDirection

cmp ax, g

je redrawPackman

cmp ax, w

je endMovePackman

setPackmanDirectionOnNext:

mov ah, packmanNextDirection

mov packmanDirection, ah

redrawPackman:

call erasePackman

call changePackmanCoordinate

call checkMeetingPackmanAndGhost

call checkMeetingPackmanAndApple

call drawPackman

endMovePackman:

pop bx

pop ax

ret

movePackman endp

;;;;;;;;;;;;;;;;;;;;;;;

createGhosts proc

updateEntropy

push cx

push si

mov cx, ghostsCount

mov si, 0

createGhostsLoop:

createGhost:

getRandomNumber fieldSize

mov ghostsX[si], al

getRandomNumber fieldSize

mov ghostsY[si], al

getRandomNumber 4

mov ghostsDirection[si], al

getRandomNumber 4

mov ghostsColor[si], al

mov ah, ghostsX[si]

mov al, ghostsY[si]

getObject

cmp ax, g

je createNextGhosts

cmp ax, w

je createGhost

createNextGhosts:

call drawGhost

inc si

loop createGhostsLoop

mov ghostsDelayCounter, 0

pop si

pop cx

ret

createGhosts endp

eraseGhost proc

mov ah, ghostsX[si]

mov al, ghostsY[si]

drawObject groundBlock

ret

eraseGhost endp

changeGhostCoordinate proc

cmp bl, 0

je moveGhostUp

cmp bl, 1

je moveGhostDown

cmp bl, 2

je moveGhostLeft

cmp bl, 3

je moveGhostRight

moveGhostUp:

dec ghostsY[si]

ret

moveGhostDown:

inc ghostsY[si]

ret

moveGhostLeft:

dec ghostsX[si]

ret

moveGhostRight:

inc ghostsX[si]

ret

changeGhostCoordinate endp

drawGhost proc

mov ah, ghostsX[si]

mov al, ghostsY[si]

mov bl, ghostsColor[si]

cmp bl, 0

je drawGhostBlue

cmp bl, 1

je drawGhostGreen

cmp bl, 2

je drawGhostPurple

cmp bl, 3

je drawGhostGray

drawGhostBlue:

drawObject ghostBlue

jmp endDrawGhost

drawGhostGreen:

drawObject ghostGreen

jmp endDrawGhost

drawGhostPurple:

drawObject ghostPurple

jmp endDrawGhost

drawGhostGray:

drawObject ghostGray

jmp endDrawGhost

endDrawGhost:

ret

drawGhost endp

getOppositeDirection proc

cmp bl, 2

jge OppositeOfLeftOrRight

jmp OppositeOfUpOrDown

OppositeOfLeftOrRight:

cmp bl, 2

je OppositeOfLeft

jmp OppositeOfRight

OppositeOfUpOrDown:

cmp bl, 1

je OppositeOfDown

jmp OppositeOfUp

OppositeOfUp:

mov bl, 1

ret

OppositeOfDown:

mov bl, 0

ret

OppositeOfLeft:

mov bl, 3

ret

OppositeOfRight:

mov bl, 2

ret

getOppositeDirection endp

moveGhosts proc

push ax

push bx

push cx

checkGhostsDelayCounter:

inc ghostsDelayCounter

cmp ghostsDelayCounter, ghostsMaxDelay

jne endMoveGhosts

mov ghostsDelayCounter, 0

updateEntropy

mov cx, ghostsCount

mov si, 0

moveGhostsLoop:

checkRandomDirection:

getRandomNumber 4

mov bl, al

mov ah, ghostsX[si]

mov al, ghostsY[si]

call getObjectOnDirection

cmp ax, g

je checkPreviousDirection

cmp ax, w

je checkRandomDirection

checkPreviousDirection:

call getOppositeDirection

mov bh, ghostsDirection[si]

cmp bh, bl

je checkRandomDirection

setGhostDirectionOnNext:

call getOppositeDirection

mov ghostsDirection[si], bl

redrawGhost:

call eraseGhost

call checkMeetingGhostAndApple

call changeGhostCoordinate

call checkMeetingPackmanAndGhost

call drawGhost

inc si

loop moveGhostsLoop

endMoveGhosts:

pop cx

pop bx

pop ax

ret

moveGhosts endp

;;;;;;;;;;;;;;;;;;;;;;;

numberToString proc

push bp

mov bp, sp

mov ax, [bp + 6]

mov si, [bp + 4]

xor cx, cx

mov bx, 10

pushDigits:

xor dx, dx

div bx

push dx

inc cx

cmp ax, 0

jne pushDigits

loopFillStr:

pop dx

add dx, 30h

mov dh, 00001111b

mov word ptr [si], dx

add si, 2

loop loopFillStr

pop bp

ret 4

numberToString endp

drawAppleCount proc

xor cx, cx

mov cl, appleCount

push cx

push offset appleCountString

call numberToString

mov si, offset appleCountString

mov di, scoreOffsetInMemory

mov cx, 4

rep movsw

ret

drawAppleCount endp

drawApple proc

mov ah, applesX[di]

mov al, applesY[di]

drawObject a

ret

drawApple endp

createApple proc

updateEntropy

xor ax, ax

setAppleCoordinates:

getRandomNumber fieldSize

mov applesX[di], al

getRandomNumber fieldSize

mov applesY[di], al

mov ah, applesX[di]

mov al, applesY[di]

getObject

cmp ax, a

je setAppleCoordinates

cmp ax, w

je setAppleCoordinates

cmp ax, g

je drawAppleAndAppleCount

drawAppleAndAppleCount:

call drawApple

mov mem, di

call drawAppleCount

mov di, mem

ret

createApple endp

createApples proc

push di

push cx

mov cx, applesNumber

mov di, 0

create:

push cx

call createApple

pop cx

inc di

loop create

pop cx

pop di

ret

createApples endp

;;;;;;;;;;;;;;;;;;;;;;;

start:

mov ax, @data

mov ds, ax

mov ax, 0B800h

mov es, ax

clearScreen

call drawField

call drawScoreMessage

call createPackman

call createGhosts

call createApples

gameLoop:

sleep

call movePackman

call moveGhosts

jmp gameLoop

endGame:

call drawGameOverScoreMessage

mov ax, 4c00h

int 21h

end start

**Вывод в консоли после компиляции**

