Implementation of Whack A Mole in DOS using ASM Interrupt Functions

Kristine Y. Coscolluela   
*Mapúa University*  
*School of EECE – CpE Dept.*Manila, Philippines  
kycoscolluela@mymail.mapua.edu.ph

Jhon Paul Jaspe  
*Mapúa University*  
*School of EECE – CpE Dept.*Manila, Philippines  
jpejaspe@mymail.mapua.edu.ph

Danielle Nicole O. Jawod  
*Mapúa University*  
*School of EECE – CpE Dept.*Manila, Philippines dnojawod@mymail.mapua.edu.ph

# Introduction

# The designed software project is a game called Whack a Mole. It uses graphics implementation and ASM advanced topics such as the INT 21H, INT 16H and INT 10H functions which includes the different keyboard and video operations. The program itself is inspired from the game Whack a Mole, an arcade game wherein the players use a mallet to hit moles, that appear randomly and go back into their holes. The program first displays the main menu of the game which gives the user three options namely play, help and exit. The only tool to operate the said program is with the use of the keyboard. To choose from the three options, the arrow keys are to be used. Once the user presses the enter key, a selection will be made. In the play option, the Whack a Mole game will be implemented where in the user must use the numpad keys to operate the spawning of the mallet in its designated location. Once the game is started, a timer of 60 seconds will countdown before the game ends. The spawning of each mole is in random which is made possible by an algorithm included in the program. Each whacked mole increases the score which is displayed in the game. As time passes, the game eventually gets more difficult as the number of moles being spawned increases. Once the timer runs out, the game will end and the player has the option to restart the game pressing R or to direct him/her to the main menu by pressing ESC key. In the help option, the program displays the instruction and mechanics of how to play the game. In the exit option, the program will be terminated.

# Objectives

## Apply the concepts in Assembly Language

The group is determined to properly execute the code integration described in the previous section by applying the concepts learned about Computer System Organization with Assembly Language. In addition, this program is in the means of providing a reference of the mechanics and capabilities of the assembly language in a simple game development.

## Demonstrate error-free execution of combined program

Aside from successful integration of several interrupt functions, the project must be executed properly, without any errors or potential bugs and loopholes.

# Methodology

## Introduction

Probably one of the most famous interrupts on the x86 architecture, Interrupt 10h is probably best known for switching video modes. Int 10h can also do other nifty things like working with cursor position, setting the active video page (for modes that support it), and even reading light pen coordinates.

## Materials

To produce a machine program that would replicate the game Whack a Mole, int 10h functions were used for the graphics imeplementation. The programmers behind this software design used such tools to generate this program. Using the DOSBox, which is an emulator program that emulates an IBM PC compatible computer running a DOS operating system, the programmers were able to debug and run the said program. Designing and coding the machine program requires appropriate usage of interrupt functions to generate the desired result or outcome. With the use of INT 10H functions, graphics implementation and video operations were made possible. On the other hand, interaction with the designed program is necessary due to the fact that it is a game which requires user interaction. Using the INT 16H functions, keyboard operations for user interaction was made possible. Followed by the correct set of logical and structured coding, a well implemented machine program would produce the desired output.

## Procedure

To design this program, the programmers took a step by step procedure to arrive at a working program. The programmers first considered the design or the display of how the game would look like once implemented. They considered such important elements namely the background display, the moles, mallet, holes, text display, and arrangement of such elements. After designing the display and interface of the program, the programmers started implementing these elements into display by using video operations. Morever, designing the game Whack a Mole was done. The next thing they did was the logic of the game itself. The programmers took into consideration of how the game should work by having knowledge about it. To spawn the moles at random locations, the programmers used such codes of algorithm. A countdown timer was also made which displays a 60 second counter and a score board which increases in every whacked mole. In addition, the programmers used such keyboard operations to let the user take control of the game and selection of options.

## Results

From the experiments that the group had encountered, the concepts and basics they have learned about Computer System Organization with Assembly Language were applied in the development of this machine problem and made this project even more successful. As a result, the group have replicated the arcade game Whack a Mole with the use of graphic implementation and video and keyboard operations. While making this project, the group was able to familiarize themselves with the functions that are related to this project as well. Although they have encountered so many errors, their troubleshooting skills improved in some way. In addition, a further knowledge in fixing some faults in the project is needed.

# Source Code

;wam.asm

.model medium

.386

org 100h

Include font.asm

writeString macro string, length, x, y, color

pusha

mov ah, 13h

mov al, 0

mov bh, 0

mov bl, color

lea bp, string

mov cx, length

mov dl, x

mov dh, y

int 10h

popa

endm

FlushKeyboard macro

mov ax, 0c00h

int 21h

endm

setCurPos macro x, y

mov ah, 2

mov bh, 0

mov dl, x

mov dh, y

int 10h

endm

drawBlockSize macro posX, posY, sizeX, sizeY, color

pusha

mov xPos, posX

mov yPos, posY

mov xSize, sizeX

mov ySize, sizeY

mov al, color

mov bh, 0

mov xEnd, posX

add xEnd, sizeX

mov yEnd, posY

add yEnd, sizeY

call draw

popa

endm

drawBlock macro posX, posY, color

mov dx, HoleBlockSize

drawBlockSize posX, posY, dx, dx, color

endm

.stack 100h

.data

; CONSTANTS

; COLORS

Black equ 000h

White equ 00Fh

LightBlue equ 034h

DarkLightBlue equ 0C4h

; HOLES

BaseColor equ 0BAh

ShadowColor equ 072h

OuterHoleColor equ 006h

; MOLES

MoleBody equ 02Ah

MoleNose equ 028h

MoleOutline equ 072h

; GRASS

GrassSize equ 003h

DarkGrass equ 078h

LightGrass equ 02Eh

; TOP PANEL

SkyColor equ 04Dh

SunColor equ 02Ch

WoodColor equ 006h

SunOutline equ 074h

WoodOutline equ 0BAh

; HAMMER

WoodDark equ 006h

MetalDark equ 014h

MetalBase equ 016h

WoodLight equ 02Ah

SmashDark equ 070h

MetalLight equ 01Ch

SmashLight equ 028h

WoodOutline equ 0BAh

SmashEffect equ 00Bh

MetalOutline equ 011h

MetalLighter equ 01Dh

; SCORE

ScoreColor equ 02Ch

ScoreBackColor equ 06Fh

; MENU

Red equ 028h

Yellow equ 02Ch

; KEY CODES

UpKey equ 4800h

EscKey equ 011Bh

DownKey equ 5000h

EnterKey equ 1C0Dh

Numpad1 equ 04Fh

Numpad2 equ 050h

Numpad3 equ 051h

Numpad4 equ 04Bh

Numpad5 equ 04Ch

Numpad6 equ 04Dh

Numpad7 equ 047h

Numpad8 equ 048h

Numpad9 equ 049h

; FONT

LetSize equ 00Ah

LetColor equ 036h

LetShadCol equ 0C6h

LetShadSize equ 014h

; Drawing Blocks

xPos dw 00

yPos dw 00

xEnd dw 00

yEnd dw 00

xSize dw 00

ySize dw 00

; Hole Information

; Hole Name dw x, y ==> Center

Hole1 dw 215, 240

Hole2 dw 345, 240

Hole3 dw 475, 240

Hole4 dw 215, 365

Hole5 dw 345, 365

Hole6 dw 475, 365

Hole7 dw 215, 490

Hole8 dw 345, 490

Hole9 dw 475, 490

HoleColor db 00

HoleBlockSize dw 05

; MOLES

Mole1 dw 238, 240

Mole2 dw 368, 240

Mole3 dw 498, 240

Mole4 dw 238, 365

Mole5 dw 368, 365

Mole6 dw 498, 365

Mole7 dw 238, 490

Mole8 dw 368, 490

Mole9 dw 498, 490

; HAMMER

Hammer1 dw 240, 220

Hammer2 dw 370, 220

Hammer3 dw 500, 220

Hammer4 dw 240, 345

Hammer5 dw 370, 345

Hammer6 dw 500, 345

Hammer7 dw 240, 470

Hammer8 dw 370, 470

Hammer9 dw 500, 470

MissCount dw 00

HammerTracker dw 00

ActiveHammers dw 09 dup(-1)

; Font

FontPos dw 00, 00

FontColor db LightBlue

GetAlphaNum dw 36 dup(00)

; TIMER

TimeMsg db ' TIME '

TimeMsgLen dw $-TimeMsg

TimePrint db 05 dup(00)

GameTime dw 60

TempSec db 00

; MENU STUFF

InMenu db 01

InPlay db 00

InHelp db 00

Choice dw -1

MenuMole1 dw 245, 105

MenuMole2 dw 515, 105

ChoicerPos0 dw 215, 215

ChoicerPos1 dw 215, 335

ChoicerPos2 dw 215, 455

; GAME THINGS

GameOverMsg db 'GAME OVER!'

GameOverMsgLen dw $-GameOverMsg

RestartMsg db 'Press R To Restart'

RestartMsgLen dw $-RestartMsg

Score dw 00

Point dw 35

SpawnTime db 01

SpawnFlag db 00

SpawnCount dw 01

ScorePrint db 05 dup(00)

TimeDifficulty db 21

MoleTracker dw 03 dup(00)

ActiveMole dw 03 dup(-1)

MoleSpawn dw 00

; HELP

MoleHelp1 dw 160, 270

MoleHelp2 dw 210, 270

MoleHelp3 dw 185, 320

HammerHelp1 dw 170, 430

HammerHelp2 dw 220, 430

HammerHelp3 dw 195, 465

RowSpace db ' '

Row1 db ' 7 8 9 '

Row2 db ' 4 5 6 '

Row3 db ' 1 2 3 '

RowLen dw $-Row3

HelpSpace db 58 dup(' ')

NumpadMsg1 db ' Use the NUMPAD KEYS as the input to whack the moles '

NumpadMsg2 db ' Each key corresponds to a mole to whack. It is aligned '

NumpadMsg3 db ' so that the 1st row corresponds to keys 7, 8, and 9. '

NumpadMsg4 db ' The 2nd row corresponds to keys 4, 5, 6. And the 3rd '

NumpadMsg5 db ' row corresponds to keys 1, 2, and 3. '

MoleMsg1 db ' This is a mole and it is your enemy. You need to WHACK '

MoleMsg2 db ' as many moles, as you can in 60 seconds and score as '

MoleMsg3 db ' much points as you can. As the game progresses, more '

MoleMsg4 db ' moles will spawn and more points can be earned. '

HammerMsg1 db ' WHACK the moles with your mighty hammer of justice in '

HammerMsg2 db ' order to bring peace to what was once a mole - free '

HammerMsg3 db ' environment. Every whacked mole gives you points and '

HammerMsg4 db ' As the game goes on, whacked moles give more points '

HammerMsg5 db ' BE CAREFUL! If you miss, you will lose big points '

HelpLen dw $-HammerMsg5

EscSpace db ' '

EscMsg1 db ' [ ESC ] '

EscMsg2 db ' MAIN MENU '

EscLen dw $-EscMsg2

.code

main proc far

Start:

mov ax, @data

mov ds, ax

mov es, ax

FlushKeyboard

; Set video mode to SVGA

; 800x600 - 256 colors

mov ax, 04F02h

mov bx, 0103h

int 10h

; SET KEYBOARD STUFF

; Repeat Rate and Delay

mov ax, 0305h

mov bh, 0

mov bl, 01fh

int 16h

call putFontsToArray

MainMenu:

call drawTopPanel

call drawLeftPanel

call drawRightPanel

call drawMiddlePanel

call drawGrassDesign

call drawSun

call drawTitle

call drawCOEGroup

call drawMainMenu

WhileInMenu:

call MenuInputs

cmp InPlay, 1

je WhileInPlay

cmp InHelp, 1

je WhileInHelp

jmp WhileInMenu

WhileInHelp:

call drawHelp

HelpLoop:

mov ah, 10h

int 16h

push ax

FlushKeyboard

pop ax

cmp ax, EscKey

jz HelpToMenu

jmp HelpLoop

HelpToMenu:

mov InHelp, 0

mov InMenu, 1

mov Choice, 0FFFFh

jmp MainMenu

WhileInPlay:

; IN-GAME STUFF

call drawTopPanel

call drawLeftPanel

call drawRightPanel

call drawMiddlePanel

call drawGrassDesign

call drawSun

call drawWoodScore

call drawWhackVertical

call drawAMoleVertical

call drawAllHoles

WriteString TimeMsg, TimeMsgLen, 86, 1, White

WriteString TimeMsg, TimeMsgLen, 86, 6, White

call PrintEsc

call MainGameLogic

Exit:

drawBlockSize 0, 0, 800, 600, Black

mov ah, 04ch

int 21h

main endp

MenuInputs proc near

pusha

mov ah, 0

int 16h

push ax

FlushKeyboard

pop ax

cmp ax, UpKey

je UpChoice

cmp ax, DownKey

je DownChoice

cmp ax, EnterKey

je ProcessChoice

jmp MenuInputRet

UpChoice:

; Choice Can't Be Less Than 0

cmp Choice, 0

jle MenuInputRet

dec Choice

jmp ClearMenuChoices

DownChoice:

; Choice Can't Be Greater Than 2

cmp Choice, 2

jge MenuInputRet

inc Choice

jmp ClearMenuChoices

ProcessChoice:

cmp Choice, 0

jz EnterPlay

cmp Choice, 1

jz EnterHelp

cmp Choice, 2

jz Exit

jmp MenuInputRet

EnterPlay:

mov InMenu, 0

mov InPlay, 1

mov InHelp, 0

jmp MenuInputRet

EnterHelp:

mov InMenu, 0

mov InPlay, 0

mov InHelp, 1

ClearMenuChoices:

lea bx, ChoicerPos0

mov cx, 3

; Draw a Block to Cover the Choicer

ClearChoices:

mov si, [bx]

mov di, [bx+2]

sub si, 4

sub di, 4

drawBlockSize si, di, 85, 90, WoodColor

add bx, 4

loop ClearChoices

; CHOICES

mov ax, Choice

shl ax, 2

lea bx, ChoicerPos0

add bx, ax

mov si, [bx]

mov di, [bx+2]

; The Purplish/Magentish Pink? Arrow Thingy

call drawChoicer

MenuInputRet:

popa

ret

MenuInputs endp

MainGameLogic proc near

; Get Current Time And Store It

mov ah, 02Ch

int 21h

mov TempSec, dh

lea bx, Mole1

LoopForever:

call Timer

call GameInputs

cmp SpawnFlag, 1

jz SpawnMole

jmp LoopBack

; Every 20 Seconds The Difficult Increases By Increasing Spawn Count

; TimeDifficulty = 21, To give the next second time for transition

IncreaseSpawnCount:

inc SpawnCount

add Point, 50

mov TimeDifficulty, 21

jmp LoopBack

; There Are 9 Predefined Hammer Positions

; HammerTracker is Used to Compute the Index

SpawnHammer:

mov si, HammerTracker

lea bx, Hammer1 + si

; When a Hammer Spawns, It Checks The ActiveMole (Spawned Moles)

; If a Hammer is on a Mole Then it's a Hit

mov cx, SpawnCount

lea di, ActiveMole

mov ax, HammerTracker

repne scasw

je HammerHit

; Draws The Hammer Without Effect

; Store The Index of The Missed Hammer in ActiveHammers

; The ActiveHammers are Cleared in the Next Second

; Lose 100 Points

HammerMiss:

call drawHammer

mov dx, HammerTracker

mov bx, MissCount

mov cl, 1

shl bx, cl

lea di, ActiveHammers + bx

mov [di], dx

inc MissCount

sub Score, 100

cmp Score, 0

jle ZeroScore

jmp ContinueMiss

; Score Can't Go Below 0

ZeroScore:

mov Score, 0

; Update the Score

ContinueMiss:

call PrintScore

jmp LoopBack

; If it hits, Draws the Hammer With Effect

; Score Distribution Table

; 1 Spawns - 35 Points Each Mole

; 2 Spawns - 85 Points Each Mole

; 3 Spawns - 135 Points Each Mole

HammerHit:

push ax

call drawHammerWithEffect

mov ax, Point

add Score, ax

call PrintScore

pop ax

jmp LoopBack

; Every Second, Random Moles are Spawned

SpawnMole:

call SpawnRandomMole

; When Time Reaches 0, It's Game Over

; TimeDifficulty Decrements Every Second...

; When it's 0, Increase The Difficulty

LoopBack:

cmp GameTime, 0

jl GameOver

cmp TimeDifficulty, 0

jz IncreaseSpawnCount

jmp LoopForever

GameOver:

WriteString GameOverMsg, GameOverMsgLen, 45, 2, White

WriteString RestartMsg, RestartMsgLen, 41, 3, White

GameOverInput:

mov ah, 10h

int 16h

push ax

FlushKeyboard

pop ax

cmp al, 'r'

jz Restart

cmp al, 'R'

jz Restart

cmp ax, EscKey

jz GameOverToMenu

jmp GameOverInput

Restart:

call RestartData

jmp WhileInPlay

GameOverToMenu:

call GotoMainMenu

MainGameLogic endp

RestartData proc near

; Set the variables needed back to what it was when game started

cld

lea di, MoleTracker

xor ax, ax

mov cx, 3

rep stosw

lea di, ActiveMole

mov ax, -1

mov cx, 3

rep stosw

lea di, ActiveHammers

mov cx, 9

rep stosw

mov MoleSpawn, 0

mov TimeDifficulty, 21

mov HammerTracker, 0

mov SpawnCount, 1

mov SpawnTime, 1

mov SpawnFlag, 0

mov MissCount, 0

mov GameTime, 60

mov TempSec, 0

mov Point, 35

mov Score, 0

ret

RestartData endp

GoToMainMenu proc near

mov InPlay, 0

mov InMenu, 1

mov choice, 0FFFFh

call RestartData

jmp MainMenu

GoToMainMenu endp

GameInputs proc near

QueryExtendedKeyboard:

; Gets Keyboard Input Without Waiting

; Flush The Keyboard Buffer Every After Input...

; It Improves the Game's Responsiveness to Inputs

mov ah, 11h

int 16h

push ax

FlushKeyboard

pop ax

cmp ah, Numpad1

jz Num1

cmp ah, Numpad2

jz Num2

cmp ah, Numpad3

jz Num3

cmp ah, Numpad4

jz Num4

cmp ah, Numpad5

jz Num5

cmp ah, Numpad6

jz Num6

cmp ah, Numpad7

jz Num7

cmp ah, Numpad8

jz Num8

cmp ah, Numpad9

jz Num9

cmp ax, EscKey

jz PlayToMenu

jmp NumEnd

PlayToMenu:

call GoToMainMenu

; The Indexes of the Hammers are Stored in HammerTracker

Num1:

mov HammerTracker, 24

jmp SpawnHammer

Num2:

mov HammerTracker, 28

jmp SpawnHammer

Num3:

mov HammerTracker, 32

jmp SpawnHammer

Num4:

mov HammerTracker, 12

jmp SpawnHammer

Num5:

mov HammerTracker, 16

jmp SpawnHammer

Num6:

mov HammerTracker, 20

jmp SpawnHammer

Num7:

mov HammerTracker, 0

jmp SpawnHammer

Num8:

mov HammerTracker, 4

jmp SpawnHammer

Num9:

mov HammerTracker, 8

jmp SpawnHammer

NumEnd:

ret

GameInputs endp

Timer proc near

pusha

; Get the Current System Time

; Checks if the Current Second and The Previous Second...

; Are Not Equal, If they are, Then 1 Second Has Passed

mov SpawnFlag, 0

mov ah, 02Ch

int 21h

cmp TempSec, dh

jne \_1SecondPassed

jmp TimerReturn

; If the MissCount is not 0, Clear Out the Missed Hammers

; ActiveHammers Contains Indexes to Missed Hammer Positions

; We Draw A Hole In that Index Position

; It Gives The Illusion of Clearing the Hammer

\_1SecondPassed:

lea si, ActiveHammers

mov cx, MissCount

cmp cx, 0

je Continue

RemoveHammer:

lea bx, Hole1

add bx, [si]

call drawFullHole

add si, 2

loop RemoveHammer

mov MissCount, cx

Continue:

call PrintTime

dec TimeDifficulty

dec SpawnTime

cmp SpawnTime, 0

jz GoSpawnMole

jmp TimerReturn

GoSpawnMole:

mov SpawnFlag, 1

mov SpawnTime, 1

TimerReturn:

mov TempSec, dh

popa

ret

Timer endp

PrintTime proc near

pusha

mov ax, GameTime

lea di, TimePrint

call ConvertHexToBCD

mov FontColor, Black

mov dx, 700

; Draw BackGround Color to Give the Illusion of Deleting Previous Time

drawBlockSize dx, 40, 80, 50, SkyColor

; Print The Time

; We Only Need the Last 2 Digits of The TimePrint

cld

mov cx, 2

lea si, TimePrint + 3

PrintLoopTimer:

xor ah, ah

lodsb

mov bx, 2

mul bl

mov bx, ax

printAlphaNum dx, 40, bx

add dx, 40

loop PrintLoopTimer

dec GameTime

popa

ret

PrintTime endp

PrintScore proc near

pusha

mov ax, Score

lea di, ScorePrint

call ConvertHexToBCD

mov dx, 305

; Draw The Board to Remove Previous Score

call drawWoodScore

cld

mov cx, 5

lea si, ScorePrint

PrintLoopScore:

xor ah, ah

lodsb

mov bx, 2

mul bl

mov bx, ax

mov di, dx

add di, 5

mov FontColor, ScoreBackColor

printAlphaNum di, 45, bx

mov FontColor, ScoreColor

printAlphaNum dx, 40, bx

add dx, 40

loop PrintLoopScore

popa

ret

PrintScore endp

ConvertHexToBCD proc near

pusha

; Converts Up to WORD Sized Hex (MAX 65535)

; Put The Hex in AX

; Use DI, to Point in Memory

mov bx, 0ffffh

mov cx, 0ffffh

back:

inc cl

sub ax, 10000

jnc back

add ax, 10000

back1:

inc ch

sub ax, 1000

jnc back1

add ax, 1000

back2:

inc bl

sub ax, 100

jnc back2

add ax, 100

back3:

inc bh

sub ax, 10

jnc back3

add ax, 10

mov word ptr[di], cx

add di, 2

mov word ptr [di], bx

add di, 2

mov byte ptr [di], al

popa

ret

ConvertHexToBCD endp

SpawnRandomMole proc near

pusha

mov cx, SpawnCount

lea si, ActiveMole

RemoveMole:

; It will Remove the previously created moles

lea bx, Hole1

add bx, [si]

call drawFullHole

add si, 2

loop RemoveMole

mov cx, SpawnCount

lea si, ActiveMole

lea di, MoleTracker

SpawnMoleLoop:

; Get random number, spawn mole at random number

call GetRandomNumber

lea bx, Mole1

add bx, [di]

; Store the spawned mole index

mov dx, [di]

mov [si], dx

add si, 2

add di, 2

call drawMole

loop SpawnMoleLoop

popa

ret

SpawnRandomMole endp

GetRandomNumber proc near

pusha

push di

cld

; This is a Somewhat Random Number Generator

; It Gets a "Random" Number from 0-9

RandomLoop:

mov ah, 02ch

int 21h

xor ax, ax

mov bx, 9

mov al, dl

div bl

or al, dh

div bl

not dl

xor dh, dl

add al, dh

div bl

and ah, 0Fh

mov al, ah

xor ah, ah

; Multiply by 4 to make it the Index

mov bl, 4

mul bl

lea di, MoleTracker

mov cx, SpawnCount

repne scasw

je LoopBackRandom

pop di

mov [di], ax

jmp ReturnRandom

LoopBackRandom:

jmp RandomLoop

ReturnRandom:

popa

ret

GetRandomNumber endp

drawChoicer proc near

pusha

; 1

mov si, [bx]

mov di, [bx+2]

add si, 29

add di, 49

mov cx, 25

mov dx, 50

drawChoicer1:

drawBlockSize si, di, dx, 1, 0B6h

inc di

dec si

sub dx, 2

loop drawChoicer1

; 2

mov si, [bx]

mov di, [bx+2]

sub si, 4

add di, 24

mov cx, 25

mov dx, 1

drawChoicer2:

drawBlockSize si, di, dx, 1, 0B6h

inc di

inc si

add dx, 2

loop drawChoicer2

; 3

mov si, [bx]

mov di, [bx+2]

add si, 4

add di, 24

mov cx, 25

mov dx, 1

drawChoicer3:

drawBlockSize si, di, dx, 1, 0B6h

inc di

inc si

add dx, 2

loop drawChoicer3

; 4

mov si, [bx]

mov di, [bx+2]

add si, 25

add di, 45

mov cx, 25

mov dx, 50

drawChoicer4:

drawBlockSize si, di, dx, 1, 25h

inc di

dec si

sub dx, 2

loop drawChoicer4

; 5

mov si, [bx]

mov di, [bx+2]

add di, 20

mov cx, 25

mov dx, 1

drawChoicer5:

drawBlockSize si, di, dx, 1, 24h

inc di

inc si

add dx, 2

loop drawChoicer5

popa

ret

drawChoicer endp

drawHelp proc near

pusha

drawBlockSize 0, 0, 800, 600, 02h

drawBlockSize 130, 60, 670, 150, 78h

drawBlockSize 130, 230, 670, 150, 78h

drawBlockSize 130, 400, 670, 150, 78h

drawBlockSize 0, 120, 100, 410, 78h

WriteString RowSpace, RowLen, 20, 5, 0

WriteString Row1, RowLen, 20, 6, 4Bh

WriteString RowSpace, RowLen, 20, 7, 0

WriteString Row2, RowLen, 20, 8, 4Bh

WriteString RowSpace, RowLen, 20, 9, 0

WriteString Row3, RowLen, 20, 10, 4Bh

WriteString RowSpace, RowLen, 20, 11, 0

lea bx, MoleHelp1

call drawMole

lea bx, MoleHelp2

call drawMole

lea bx, MoleHelp3

call drawMole

lea bx, HammerHelp1

call drawHammer

lea bx, HammerHelp2

call drawHammer

lea bx, HammerHelp3

call drawHammer

WriteString HelpSpace, HelpLen, 40, 5, 0

WriteString NumpadMsg1, HelpLen, 40, 6, 4Bh

WriteString NumpadMsg2, HelpLen, 40, 7, 4Bh

WriteString NumpadMsg3, HelpLen, 40, 8, 4Bh

WriteString NumpadMsg4, HelpLen, 40, 9, 4Bh

WriteString NumpadMsg5, HelpLen, 40, 10, 4Bh

WriteString HelpSpace, HelpLen, 40, 11, 0

WriteString HelpSpace, HelpLen, 40, 16, 0

WriteString MoleMsg1, HelpLen, 40, 17, 4Bh

WriteString MoleMsg2, HelpLen, 40, 18, 4Bh

WriteString MoleMsg3, HelpLen, 40, 19, 4Bh

WriteString MoleMsg4, HelpLen, 40, 20, 4Bh

WriteString HelpSpace, HelpLen, 40, 21, 0

WriteString HelpSpace, HelpLen, 40, 26, 0

WriteString HammerMsg1, HelpLen, 40, 27, 4Bh

WriteString HammerMsg2, HelpLen, 40, 28, 4Bh

WriteString HammerMsg3, HelpLen, 40, 29, 4Bh

WriteString HammerMsg4, HelpLen, 40, 30, 4Bh

WriteString HammerMsg5, HelpLen, 40, 31, 4Bh

WriteString HelpSpace, HelpLen, 40, 32, 0

mov FontColor, 94h

PrintAlphaNum 35, 155, \_H

PrintAlphaNum 35, 255, \_E

PrintAlphaNum 35, 355, \_L

PrintAlphaNum 35, 455, \_P

mov FontColor, 4Ch

PrintAlphaNum 30, 150, \_H

PrintAlphaNum 30, 250, \_E

PrintAlphaNum 30, 350, \_L

PrintAlphaNum 30, 450, \_P

call PrintEsc

popa

ret

drawHelp endp

PrintEsc proc near

WriteString EscSpace, EscLen, 0, 0, 0

WriteString EscMsg1, EscLen, 0, 1, Red

WriteString EscMsg2, EscLen, 0, 2, Red

WriteString EscSpace, EscLen, 0, 3, 0

ret

PrintEsc endp

drawMainMenu proc near

; WOODS

drawBlockSize 210, 210, 360, 100, WoodColor

drawBlockSize 300, 220, 260, 80, WoodOutline

drawBlockSize 210, 330, 360, 100, WoodColor

drawBlockSize 300, 340, 260, 80, WoodOutline

drawBlockSize 210, 450, 360, 100, WoodColor

drawBlockSize 300, 460, 260, 80, WoodOutline

; RED

mov FontColor, Red

PrintAlphaNum 340, 230, \_P

PrintAlphaNum 390, 230, \_L

PrintAlphaNum 440, 230, \_A

PrintAlphaNum 490, 230, \_Y

PrintAlphaNum 340, 350, \_H

PrintAlphaNum 390, 350, \_E

PrintAlphaNum 440, 350, \_L

PrintAlphaNum 490, 350, \_P

PrintAlphaNum 340, 470, \_E

PrintAlphaNum 390, 470, \_X

PrintAlphaNum 440, 470, \_I

PrintAlphaNum 490, 470, \_T

; YELLOW

mov FontColor, Yellow

PrintAlphaNum 340, 235, \_P

PrintAlphaNum 390, 235, \_L

PrintAlphaNum 440, 235, \_A

PrintAlphaNum 490, 235, \_Y

PrintAlphaNum 340, 355, \_H

PrintAlphaNum 390, 355, \_E

PrintAlphaNum 440, 355, \_L

PrintAlphaNum 490, 355, \_P

PrintAlphaNum 340, 475, \_E

PrintAlphaNum 390, 475, \_X

PrintAlphaNum 440, 475, \_I

PrintAlphaNum 490, 475, \_T

ret

drawMainMenu endp

drawTitle proc near

; Shadow

mov FontColor, 68h

PrintAlphaNum 250, 20, \_W

PrintAlphaNum 320, 20, \_H

PrintAlphaNum 370, 20, \_A

PrintAlphaNum 420, 20, \_C

PrintAlphaNum 470, 20, \_K

PrintAlphaNum 540, 20, \_A

PrintAlphaNum 320, 90, \_M

PrintAlphaNum 370, 90, \_O

PrintAlphaNum 420, 90, \_L

PrintAlphaNum 470, 90, \_E

; Foreground

mov FontColor, 20h

PrintAlphaNum 250, 15, \_W

PrintAlphaNum 320, 15, \_H

PrintAlphaNum 370, 15, \_A

PrintAlphaNum 420, 15, \_C

PrintAlphaNum 470, 15, \_K

PrintAlphaNum 540, 15, \_A

PrintAlphaNum 320, 85, \_M

PrintAlphaNum 370, 85, \_O

PrintAlphaNum 420, 85, \_L

PrintAlphaNum 470, 85, \_E

lea bx, MenuMole1

call drawMole

lea bx, MenuMole2

call drawMole

ret

drawTitle endp

drawCOEGroup proc near

; Shadow

mov FontColor, DarkLightBlue

PrintAlphaNum 715, 185, \_G

PrintAlphaNum 715, 250, \_R

PrintAlphaNum 715, 315, \_O

PrintAlphaNum 715, 380, \_U

PrintAlphaNum 715, 445, \_P

PrintAlphaNum 715, 515, \_3

; GROUP 3

mov FontColor, LightBlue

PrintAlphaNum 710, 180, \_G

PrintAlphaNum 710, 245, \_R

PrintAlphaNum 710, 310, \_O

PrintAlphaNum 710, 375, \_U

PrintAlphaNum 710, 440, \_P

PrintAlphaNum 710, 510, \_3

; Shadow

mov FontColor, DarkLightBlue

PrintAlphaNum 68, 185, \_C

PrintAlphaNum 68, 245, \_O

PrintAlphaNum 68, 305, \_E

PrintAlphaNum 68, 360, \_1

PrintAlphaNum 68, 415, \_1

PrintAlphaNum 68, 475, \_9

PrintAlphaNum 71, 520, \_L

; COE119

mov FontColor, LightBlue

PrintAlphaNum 63, 180, \_C

PrintAlphaNum 63, 240, \_O

PrintAlphaNum 63, 300, \_E

PrintAlphaNum 63, 355, \_1

PrintAlphaNum 63, 410, \_1

PrintAlphaNum 63, 470, \_9

PrintAlphaNum 66, 515, \_L

ret

drawCOEGroup endp

drawHammer proc near

pusha

call drawHammerBase

call drawHammerDark

call drawHammerLight

call drawHammerOutline

call drawWoodHawakan

popa

ret

drawHammer endp

drawHammerWithEffect proc near

pusha

call drawSmashEffectBG

call drawSmashEffect

call drawHammerBase

call drawHammerDark

call drawHammerLight

call drawHammerOutline

call drawWoodHawakan

popa

ret

drawHammerWithEffect endp

drawSmashEffectBG proc near

; Triangle 1

mov si, [bx]

mov di, [bx+2]

add si, 50

add di, 4

mov cx, 20

mov dx, 1

drawTriangle1:

drawBlockSize si, di, dx, 1, SmashLight

inc di

dec si

inc dx

loop drawTriangle1

; Triangle 2

mov si, [bx]

mov di, [bx+2]

add si, 48

add di, 15

mov cx, 20

mov dx, 1

drawTriangle2:

drawBlockSize si, di, dx, 1, SmashLight

inc di

inc dx

loop drawTriangle2

; Triangle 3

mov si, [bx]

mov di, [bx+2]

add si, 48

add di, 15

mov cx, 20

mov dx, 20

drawTriangle3:

drawBlockSize si, di, dx, 1, SmashLight

inc di

dec dx

loop drawTriangle3

; Triangle 4

mov si, [bx]

mov di, [bx+2]

add si, 44

add di, 25

mov cx, 20

mov dx, 1

drawTriangle4:

drawBlockSize si, di, dx, 1, SmashLight

inc di

dec si

add dx, 2

loop drawTriangle4

; Triangle 5

mov si, [bx]

mov di, [bx+2]

add si, 33

add di, 31

mov cx, 22

mov dx, 1

drawTriangle5:

drawBlockSize si, di, dx, 1, SmashLight

inc di

dec si

add dx, 2

loop drawTriangle5

; Triangle 6

mov si, [bx]

mov di, [bx+2]

add si, 22

add di, 49

mov cx, 10

mov dx, 19

drawTriangle6:

drawBlockSize si, di, dx, 1, SmashLight

inc di

inc si

sub dx, 2

loop drawTriangle6

; Smash Dark

mov si, [bx]

mov di, [bx+2]

add si, 51

add di, 25

mov cx, 5

SmashDarkLoop1:

pusha

mov cx, 20

SmashDarkLoop2:

drawBlockSize si, di, 2, 2, SmashDark

dec si

inc di

loop SmashDarkLoop2

popa

inc si

inc di

loop SmashDarkLoop1

; Dark Triangle 1

mov si, [bx]

mov di, [bx+2]

add si, 25

add di, 36

mov cx, 12

mov dx, 1

drawDarkTriangle1:

drawBlockSize si, di, dx, 1, SmashDark

inc di

inc dx

loop drawDarkTriangle1

; Dark Triangle 2

mov si, [bx]

mov di, [bx+2]

add si, 41

add di, 19

mov cx, 12

mov dx, 12

drawDarkTriangle2:

drawBlockSize si, di, dx, 1, SmashDark

inc di

inc si

dec dx

loop drawDarkTriangle2

; Dark Triangle 3

mov si, [bx]

mov di, [bx+2]

add si, 43

add di, 30

mov cx, 12

mov dx, 1

drawDarkTriangle3:

drawBlockSize si, di, dx, 1, SmashDark

inc di

inc dx

loop drawDarkTriangle3

ret

drawSmashEffectBG endp

drawSmashEffect proc near

mov si, [bx]

mov di, [bx+2]

add si, 48

add di, 10

drawBlockSize si, di, 2, 6, SmashEffect

mov cx, 10

drawEffect1:

drawBlockSize si, di, 2, 1, SmashEffect

dec si

inc di

loop drawEffect1

add si, 15

drawBlockSize si, di, 10, 1, SmashEffect

drawBlockSize si, di, 2, 9, SmashEffect

mov cx, 10

drawEffect2:

drawBlockSize si, di, 2, 1, SmashEffect

dec si

inc di

loop drawEffect2

add si, 3

add di, 2

drawBlockSize si, di, 15, 1, SmashEffect

drawBlockSize si, di, 2, 13, SmashEffect

mov cx, 10

drawEffect3:

drawBlockSize si, di, 1, 2, SmashEffect

inc si

inc di

loop drawEffect3

sub di, 5

sub si, 15

drawBlockSize si, di, 1, 15, SmashEffect

add di, 12

drawBlockSize si, di, 8, 2, SmashEffect

sub di, 8

mov cx, 10

drawEffect4:

drawBlockSize si, di, 1, 1, SmashEffect

dec si

inc di

loop drawEffect4

drawBlockSize si, di, 2, 6, SmashEffect

sub di, 12

drawBlockSize si, di, 1, 12, SmashEffect

mov cx, 10

drawEffect5:

drawBlockSize si, di, 2, 1, SmashEffect

dec si

inc di

loop drawEffect5

sub si, 6

drawBlockSize si, di, 9, 1, SmashEffect

ret

drawSmashEffect endp

drawWoodHawakan proc near

mov si, [bx]

mov di, [bx+2]

add si, 14

add di, 30

mov cx, 8

WoodOutlineLoop1:

drawBlockSize si, di, 2, 2, WoodOutline

sub si, 2

add di, 2

loop WoodOutlineLoop1

drawBlockSize si, di, 2, 4, WoodOutline

add di, 4

drawBlockSize si, di, 6, 2, WoodOutline

add si, 4

mov cx, 8

WoodOutlineLoop2:

drawBlockSize si, di, 2, 2, WoodOutline

add si, 2

sub di, 2

loop WoodOutlineLoop2

mov si, [bx]

mov di, [bx+2]

add si, 12

add di, 34

mov cx, 7

WoodDarkLoop1:

drawBlockSize si, di, 2, 2, WoodDark

sub si, 2

add di, 2

loop WoodDarkLoop1

mov si, [bx]

mov di, [bx+2]

add si, 12

add di, 36

mov cx, 7

WoodLightLoop1:

drawBlockSize si, di, 4, 2, WoodLight

sub si, 2

add di, 2

loop WoodLightLoop1

ret

drawWoodHawakan endp

drawHammerBase proc near

mov si, [bx]

mov di, [bx+2]

add si, 45

add di, 24

mov cx, 7

HammerBaseLoop1:

drawBlockSize si, di, 2, 8, MetalBase

sub si, 2

add di, 2

loop HammerBaseLoop1

ret

drawHammerBase endp

drawHammerDark proc near

mov si, [bx]

mov di, [bx+2]

add si, 25

add di, 2

mov cx, 10

HammerDarkLoop1:

drawBlockSize si, di, 4, 2, MetalDark

sub si, 2

add di, 2

loop HammerDarkLoop1

mov cx, 10

HammerDarkLoop2:

drawBlockSize si, di, 10, 2, MetalDark

add si, 2

add di, 2

loop HammerDarkLoop2

mov cx, 10

HammerDarkLoop3:

drawBlockSize si, di, 4, 2, MetalDark

add si, 2

sub di, 2

loop HammerDarkLoop3

sub si, 4

mov cx, 10

HammerDarkLoop4:

drawBlockSize si, di, 6, 2, MetalDark

sub si, 2

sub di, 2

loop HammerDarkLoop4

ret

drawHammerDark endp

drawHammerLight proc near

mov si, [bx]

mov di, [bx+2]

add si, 17

add di, 2

mov cx, 5

HammerLightLoop1:

drawBlockSize si, di, 8, 2, MetalLight

sub si, 2

add di, 2

loop HammerLightLoop1

sub si, 2

add di, 2

mov cx, 5

HammerLightLoop2:

drawBlockSize si, di, 2, 8, MetalLight

add si, 2

sub di, 2

loop HammerLightLoop2

; Inner

mov si, [bx]

mov di, [bx+2]

add si, 23

add di, 8

mov cx, 9

HammerLightLoop3:

pusha

mov cx, 7

HammerLightLoop4:

drawBlockSize si, di, 4, 2, MetalLighter

sub si, 2

add di, 2

loop HammerLightLoop4

popa

add si, 2

add di, 2

loop HammerLightLoop3

mov si, [bx]

mov di, [bx+2]

add si, 21

add di, 17

drawBlockSize si, di, 12, 12, 7Ch

add si, 3

add di, 3

drawBlockSize si, di, 6, 6, SmashEffect

ret

drawHammerLight endp

drawHammerOutline proc near

mov si, [bx]

mov di, [bx+2]

add si, 15

drawBlockSize si, di, 12, 2, MetalOutline

mov cx, 6

HammerOutlineLoop1:

drawBlockSize si, di, 4, 2, MetalOutline

add di, 2

sub si, 2

loop HammerOutlineLoop1

add si, 2

drawBlockSize si, di, 2, 12, MetalOutline

add di, 10

mov cx, 10

HammerOutlineLoop2:

drawBlockSize si, di, 2, 2, MetalOutline

add si, 2

add di, 2

loop HammerOutlineLoop2

drawBlockSize si, di, 12, 2, MetalOutline

add si, 10

mov cx, 6

HammerOutlineLoop3:

drawBlockSize si, di, 4, 2, MetalOutline

add si, 2

sub di, 2

loop HammerOutlineLoop3

sub di, 8

drawBlockSize si, di, 2, 12, MetalOutline

mov cx, 12

HammerOutlineLoop4:

drawBlockSize si, di, 2, 2, MetalOutline

sub si, 2

sub di, 2

loop HammerOutlineLoop4

ret

drawHammerOutline endp

drawMole proc near

pusha

; MOLES

mov si, [bx]

mov di, [bx+2]

; 1 Start

add si, 2

drawBlockSize si, di, 3, 37, MoleOutline

add si, 3

drawBlockSize si, di, 50, 37, MoleBody

; Nose Start

pusha

add si, 20

add di, 3

drawBlockSize si, di, 8, 8, MoleNose

popa

; Nose End

; Mouth Start

pusha

add si, 5

add di, 8

drawBlockSize si, di, 5, 5, MoleOutline

add si, 35

drawBlockSize si, di, 5, 5, MoleOutline

sub si, 30

add di, 3

drawBlockSize si, di, 5, 5, MoleOutline

add si, 25

drawBlockSize si, di, 5, 5, MoleOutline

sub si, 20

add di, 3

drawBlockSize si, di, 20, 5, MoleOutline

; Teeth Start

add di, 3

add si, 5

drawBlockSize si, di, 4, 6, White

add si, 6

drawBlockSize si, di, 4, 6, White

; Teeth End

popa

; Mouth End

add si, 50

drawBlockSize si, di, 3, 37, MoleOutline

; 1 End

; 2 Start

mov si, [bx]

mov di, [bx+2]

add si, 7

sub di, 10

drawBlockSize si, di, 3, 10, MoleOutline

add si, 3

drawBlockSize si, di, 40, 10, MoleBody

;Eyes Start

add si, 5

drawBlockSize si, di, 10, 10, Black

add si, 2

add di, 2

drawBlockSize si, di, 3, 3, White

add si, 18

sub di, 2

drawBlockSize si, di, 10, 10, BLack

add si, 2

add di, 2

drawBlockSize si, di, 3, 3, White

; Eyes End

sub di, 2

add si, 13

drawBlockSIze si, di, 3, 10, MoleOutline

; 2 End

; 3 Start

mov si, [bx]

mov di, [bx+2]

add si, 12

sub di, 15

drawBlockSize si, di, 3, 5, MoleOutline

add si, 3

drawBlockSize si, di, 30, 5, MoleBody

add si, 30

drawBlockSize si, di, 3, 5, MoleOutline

; 3 End

; 4 Start

mov si, [bx]

mov di, [bx+2]

add si, 17

sub di, 20

drawBlockSize si, di, 3, 5, MoleOutline

add si, 3

drawBlockSize si, di, 20, 5, MoleBody

add si, 20

drawBlockSize si, di, 3, 5, MoleOutline

; 4 End

; 5 Start

mov si, [bx]

mov di, [bx+2]

add si, 20

sub di, 22

drawBlockSize si, di, 20, 3, MoleOutline

; 5 End

popa

ret

drawMole endp

drawTopPanel proc near

drawBlockSize 0, 0, 800, 150, SkyColor

ret

drawTopPanel endp

drawWoodScore proc near

pusha

; Paa ng scoreboard

mov si, 330

mov di, 5

drawBlockSize si, di, 10, 145, WoodColor

add si, 140

drawBlockSize si, di, 10, 145, WoodColor

; ung scoreboard

mov si, 285

mov di, 20

call drawWood

mov si, 275

mov di, 50

call drawWood

mov si, 285

mov di, 80

call drawWood

popa

ret

drawWoodScore endp

drawWood proc near

drawBlockSize si, di, 240, 35, WoodColor

drawBlockSize si, di, 240, 5, WoodOutline

add di, 30

drawBlockSize si, di, 240, 5, WoodOutline

sub di, 30

drawBlockSize si, di, 5, 35, WoodOutline

add si, 235

drawBlockSize si, di, 5, 35, WoodOutline

ret

drawWood endp

drawSun proc near

mov si, 150

mov di, 35

; Main Sun

drawBlockSize si, di, 40, 40, SunColor

; Outer Sun

drawBlockSize si, di, 40, 2, SunOutline

add di, 40

drawBlockSize si, di, 42, 2, SunOutline

sub di, 40

drawBlockSize si, di, 2, 40, SunOutline

add si, 40

drawBlockSize si, di, 2, 40, SunOutline

;Inner Sun

sub si, 27

add di, 12

drawBlockSize si, di, 15, 2, SunOutline

add di, 15

drawBlockSize si, di, 17, 2, SunOutline

sub di, 15

drawBlockSize si, di, 2, 15, SunOutline

add si, 15

drawBlockSize si, di, 2, 15, SunOutline

; Sun Rays

mov si, 160

mov di, 10

drawBlockSize si, di, 3, 20, SunColor

add si, 20

drawBlockSize si, di, 3, 20, SunColor

add di, 70

drawBlockSize si, di, 3, 20, SunColor

sub si, 20

drawBlockSize si, di, 3, 20, SunColor

mov si, 125

mov di, 45

drawBlockSize si, di, 20, 3, SunColor

add di, 20

drawBlockSize si, di, 20, 3, SunColor

add si, 75

drawBlockSize si, di, 20, 3, SunColor

sub di, 20

drawBlockSize si, di, 20, 3, SunColor

ret

drawSun endp

drawRightPanel proc near

pusha

;RIGHT PANEL

; Center

drawBlockSize 673, 173, 104, 404, 79h

; Outer Horizontal

drawBlockSize 650, 150, 142, 15, 32h

drawBlockSize 650, 577, 142, 15, 32h

;Outer Vertical

drawBlockSize 650, 165, 15, 412, 32h

drawBlockSize 777, 165, 15, 412, 32h

;Shadow

drawBlockSize 650, 592, 150, 8, 0D8h

drawBlockSize 792, 150, 8, 442, 0D8h

drawBlockSize 665, 165, 112, 8, 0D8h

drawBlockSize 665, 173, 8, 404, 0D8h

popa

ret

drawRightpanel endp

drawAMoleVertical proc near

; A MOLE Shadow

mov FontColor, DarkLightBlue

PrintAlphaNum 715, 200, \_A

PrintAlphaNum 710, 295, \_M

PrintAlphaNum 715, 365, \_O

PrintAlphaNum 715, 435, \_L

PrintAlphaNum 715, 505, \_E

; A MOLE

mov FontColor, LightBlue

PrintAlphaNum 710, 195, \_A

PrintAlphaNum 705, 290, \_M

PrintAlphaNum 710, 360, \_O

PrintAlphaNum 710, 430, \_L

PrintAlphaNum 710, 500, \_E

ret

drawAMoleVertical endp

drawLeftPanel proc near

pusha

; LEFT PANEL

; Center

drawBlockSize 23, 165, 112, 412, 79h

; Outer Horizontal

drawBlockSize 8, 150, 142, 15, 32h

drawBlockSize 8, 577, 142, 15, 32h

; Outer Vertical

drawBlockSize 8, 165, 15, 412, 32h

drawBlockSize 135, 165, 15, 412, 32h

; Shadow

drawBlockSize 0, 150, 8, 450, 0D8h

drawBlockSize 8, 592, 142, 8, 0D8h

drawBlockSize 23, 165, 112, 8, 0D8h

drawBlockSize 127, 173, 8, 404, 0D8h

popa

ret

drawLeftPanel endp

drawWhackVertical proc near

; WHACK Shadow

mov FontColor, DarkLightBlue

PrintAlphaNum 55, 210, \_W

PrintAlphaNum 68, 280, \_H

PrintAlphaNum 68, 350, \_A

PrintAlphaNum 68, 420, \_C

PrintAlphaNum 68, 490, \_K

; WHACK

mov FontColor, LightBlue

PrintAlphaNum 50, 205, \_W

PrintAlphaNum 63, 275, \_H

PrintAlphaNum 63, 345, \_A

PrintAlphaNum 63, 415, \_C

PrintAlphaNum 63, 485, \_K

ret

drawWhackVertical endp

drawMiddlePanel proc near

pusha

; Center

drawBlockSize 180, 175, 440, 405, 02h

; Horizontal Inner

drawBlockSize 157, 150, 486, 15, 30h

drawBlockSize 157, 578, 486, 15, 30h

; Vertical Inner

drawBlockSize 157, 165, 15, 413, 30h

drawBlockSize 628, 165, 15, 413, 30h

; Shadow

; Horizontal

drawBlockSize 172, 165, 456, 10, 0C1h

drawBlockSize 150, 592, 500, 8, 0C1h

; Vertical Inner

drawBlockSize 172, 175, 8, 403, 0C1h

drawBlockSize 620, 175, 8, 403, 0C1h

; Vertical Outer

drawBlockSize 150, 150, 8, 442, 0C1h

drawBlockSize 642, 150, 8, 442, 0C1h

popa

ret

drawMiddlePanel endp

drawGrassDesign proc near

pusha

mov di, 188

mov cx, 6

GrassDesign:

push cx

call GrassRow1

add di, 15

call GrassRow2

add di, 15

call GrassRow3

add di, 15

call GrassRow4

add di, 15

pop cx

loop GrassDesign

call GrassRow1

add di, 15

call GrassRow2

popa

ret

drawGrassDesign endp

GrassRow1 proc near

mov si, 200

mov cx, 7

DarkGrass\_1\_Out:

push cx

drawBlockSize si, di, GrassSize, GrassSize, DarkGrass

add si, 20

mov cx, 2

LightGrass\_2\_In:

drawBlockSize si, di, GrassSize, GrassSize, LightGrass

add si, 20

loop LightGrass\_2\_In

pop cx

loop DarkGrass\_1\_Out

ret

GrassRow1 endp

GrassRow2 proc near

mov si, 190

mov cx, 7

LightGrass\_1\_In:

push cx

mov cx, 2

DarkGrass\_2\_Out:

drawBlockSize si, di, GrassSize, GrassSize, DarkGrass

add si, 20

loop DarkGrass\_2\_Out

pop cx

drawBlockSize si, di, GrassSize, GrassSize, LightGrass

add si, 20

loop LightGrass\_1\_In

ret

GrassRow2 endp

GrassRow3 proc near

mov si, 190

mov cx, 7

DarkGrass\_1\_In:

push cx

mov cx, 2

LightGrass\_2\_Out:

drawBlockSize si, di, GrassSize, GrassSize, LightGrass

add si, 20

loop LightGrass\_2\_Out

pop cx

drawBlockSize si, di, GrassSize, GrassSize, DarkGrass

add si, 20

loop DarkGrass\_1\_In

ret

GrassRow3 endp

GrassRow4 proc near

mov si, 200

mov cx, 7

LightGrass\_1\_Out:

push cx

drawBlockSize si, di, GrassSize, GrassSize, LightGrass

add si, 20

mov cx, 2

DarkGrass\_2\_In:

drawBlockSize si, di, GrassSize, GrassSize, DarkGrass

add si, 20

loop DarkGrass\_2\_In

pop cx

loop LightGrass\_1\_Out

ret

GrassRow4 endp

draw proc near

; Write Graphic Pixel

mov ah, 0ch

; Position to Start Drawing

mov cx, xPos

mov dx, yPos

drawLoop:

int 10h

inc cx

cmp cx, xEnd

jne drawLoop

mov cx, xPos

inc dx

cmp dx, yEnd

jne drawLoop

ret

draw endp

drawAllHoles proc near

lea bx, Hole1

mov cx, 9

drawHoleLoop:

call drawFullHole

add bx, 4

loop drawHoleLoop

ret

drawAllHoles endp

drawFullHole proc near

pusha

;Base

call drawHoleBase

;Shadow

mov HoleColor, ShadowColor

mov HoleBlockSize, 11

call drawHole

;Outer

mov HoleColor, OuterHoleColor

mov HoleBlockSize, 7

call drawHole

popa

ret

drawFullHole endp

drawHoleBase proc near

pusha

; DRAW Base for the holes

mov si, [bx]

add si, 15

mov cx, 19

drawHoleBase\_out:

mov di, [bx+2]

sub di, 20

push cx

mov cx, 14

drawHoleBase\_in:

drawBlockSize si, di, 4, 4, BaseColor

add di, 4

loop drawHoleBase\_in

add si, 4

pop cx

loop drawHoleBase\_out

popa

ret

drawHoleBase endp

drawHole proc near

pusha

; SI = X position

; DI = Y position

; Hole-A Horizontal

mov di, [bx+2] ; Hole y Center

sub di, 25 ; y Start Adjusted

mov cx, 2 ; 2 vertical bars

draw\_A\_out:

mov si, [bx] ; Hole x Center

add si, 25 ; x Start Adjusted

push cx

mov cx, 10 ; 10 blocks to the right

draw\_A\_in:

drawBlock si, di, HoleColor

add si, 5 ; x Offset adjust

loop draw\_A\_in

add di, 60 ; y space in between

pop cx

loop draw\_A\_out

; Hole-B\_Left Horizontal

mov di, [bx+2]

sub di, 20

mov cx, 2

draw\_Bleft\_out:

mov si, [bx]

add si, 15

push cx

mov cx, 3

draw\_Bleft\_in:

drawBlock si, di, HoleColor

add si, 5

loop draw\_Bleft\_in

add di, 50

pop cx

loop draw\_Bleft\_out

; Hole-B\_Right Horizontal

mov di, [bx+2]

sub di, 20

mov cx, 2

draw\_Bright\_out:

mov si, [bx]

add si, 70

push cx

mov cx, 3

draw\_Bright\_in:

drawBlock si, di, HoleColor

add si, 5

loop draw\_Bright\_in

add di, 50

pop cx

loop draw\_Bright\_out

; Hole-C\_Left Horizontal

mov di, [bx+2]

sub di, 15

mov cx, 2

draw\_Cleft\_out:

mov si, [bx]

add si, 10

push cx

mov cx, 2

draw\_Cleft\_in:

drawBlock si, di, HoleColor

add si, 5

loop draw\_Cleft\_in

add di, 40

pop cx

loop draw\_Cleft\_out

; Hole-C\_Right Horizontal

mov di, [bx+2]

sub di, 15

mov cx, 2

draw\_Cright\_out:

mov si, [bx]

add si, 80

push cx

mov cx, 2

draw\_Cright\_in:

drawBlock si, di, HoleColor

add si, 5

loop draw\_Cright\_in

add di, 40

pop cx

loop draw\_Cright\_out

; Hole-D Vertical

mov si, [bx]

add si, 5

mov cx, 2

draw\_D\_out:

mov di, [bx+2]

sub di, 10

push cx

mov cx, 7

draw\_D\_in:

drawBlock si, di, HoleColor

add di, 5

loop draw\_D\_in

add si, 85

pop cx

loop draw\_D\_out

popa

ret

drawHole endp

end main

;font.asm

drawFontSize macro sizeX, sizeY

pusha

mov xPos, si

mov yPos, di

mov xSize, sizeX

mov ySize, sizeY

mov al, FontColor

mov bh, 0

mov xEnd, si

add xEnd, sizeX

mov yEnd, di

add yEnd, sizeY

call draw

popa

endm

PrintAlphaNum macro x, y, AlphaNum

pusha

mov si, x

mov di, y

call GetAlphaNum[AlphaNum]

popa

endm

.data

; AlphaNumeric Equivalence

\_0 equ 0

\_1 equ 2

\_2 equ 4

\_3 equ 6

\_4 equ 8

\_5 equ 10

\_6 equ 12

\_7 equ 14

\_8 equ 16

\_9 equ 18

\_A equ 20

\_B equ 22

\_C equ 24

\_D equ 26

\_E equ 28

\_F equ 30

\_G equ 32

\_H equ 34

\_I equ 36

\_J equ 38

\_K equ 40

\_L equ 42

\_M equ 44

\_N equ 46

\_O equ 48

\_P equ 50

\_Q equ 52

\_R equ 54

\_S equ 56

\_T equ 58

\_U equ 60

\_V equ 62

\_W equ 64

\_X equ 66

\_Y equ 68

\_Z equ 70

.code

putFontsToArray proc near

lea di, GetAlphaNum

xor bx, bx

lea si, draw0

mov [di+bx], si

add bx, 2

lea si, draw1

mov [di+bx], si

add bx, 2

lea si, draw2

mov [di+bx], si

add bx, 2

lea si, draw3

mov [di+bx], si

add bx, 2

lea si, draw4

mov [di+bx], si

add bx, 2

lea si, draw5

mov [di+bx], si

add bx, 2

lea si, draw6

mov [di+bx], si

add bx, 2

lea si, draw7

mov [di+bx], si

add bx, 2

lea si, draw8

mov [di+bx], si

add bx, 2

lea si, draw9

mov [di+bx], si

add bx, 2

lea si, drawA

mov [di+bx], si

add bx, 2

lea si, drawB

mov [di+bx], si

add bx, 2

lea si, drawC

mov [di+bx], si

add bx, 2

lea si, drawD

mov [di+bx], si

add bx, 2

lea si, drawE

mov [di+bx], si

add bx, 2

lea si, drawF

mov [di+bx], si

add bx, 2

lea si, drawG

mov [di+bx], si

add bx, 2

lea si, drawH

mov [di+bx], si

add bx, 2

lea si, drawI

mov [di+bx], si

add bx, 2

lea si, drawJ

mov [di+bx], si

add bx, 2

lea si, drawK

mov [di+bx], si

add bx, 2

lea si, drawL

mov [di+bx], si

add bx, 2

lea si, drawM

mov [di+bx], si

add bx, 2

lea si, drawN

mov [di+bx], si

add bx, 2

lea si, drawO

mov [di+bx], si

add bx, 2

lea si, drawP

mov [di+bx], si

add bx, 2

lea si, drawQ

mov [di+bx], si

add bx, 2

lea si, drawR

mov [di+bx], si

add bx, 2

lea si, drawS

mov [di+bx], si

add bx, 2

lea si, drawT

mov [di+bx], si

add bx, 2

lea si, drawU

mov [di+bx], si

add bx, 2

lea si, drawV

mov [di+bx], si

add bx, 2

lea si, drawW

mov [di+bx], si

add bx, 2

lea si, drawX

mov [di+bx], si

add bx, 2

lea si, drawY

mov [di+bx], si

add bx, 2

lea si, drawZ

mov [di+bx], si

add bx, 2

ret

putFontsToArray endp

; FONTS

drawA proc near

push di

add di, 5

drawFontSize 8, 45

add si, 22

drawFontSize 8, 45

pop di

sub si, 17

drawFontSize 20, 8

add di, 21

drawFontSize 20, 8

ret

drawA endp

drawB proc near

push si

drawFontSize 8, 50

add si, 22

add di, 5

drawFontSize 8, 40

pop si

add si, 8

sub di, 5

drawFontSize 14, 8

add di, 21

drawFontSize 14, 8

add di, 21

drawFontSize 14, 8

ret

drawB endp

drawC proc near

drawFontSize 30, 8

drawFontSize 8, 50

add di, 42

drawFontSize 30, 8

ret

drawC endp

drawD proc near

push si

drawFontSize 8, 50

add si, 22

add di, 5

drawFontSize 8, 40

pop si

add si, 8

sub di, 5

drawFontSize 14, 8

add di, 42

drawFontSize 14, 8

ret

drawD endp

drawE proc near

push di

drawFontSize 30, 8

add di, 21

drawFontSize 30, 8

add di, 21

drawFontSize 30, 8

pop di

add di, 8

drawFontSize 8, 34

ret

drawE endp

drawF proc near

push di

drawFontSize 30, 8

add di, 21

drawFontSize 30, 8

pop di

add di, 8

drawFontSize 8, 42

ret

drawF endp

drawG proc near

push di

drawFontSize 30, 8

add di, 21

add si, 15

drawFontSize 15, 8

sub si, 15

add di, 21

drawFontSize 30, 8

pop di

add di, 8

drawFontSize 8, 34

add si, 22

add di, 20

drawFontSize 8, 14

ret

drawG endp

drawH proc near

drawFontSize 8, 50

add si, 22

drawFontSize 8, 50

sub si, 14

add di, 21

drawFontSize 14, 8

ret

drawH endp

drawI proc near

drawFontSize 30, 8

add di, 42

drawFontSize 30, 8

sub di, 34

add si, 11

drawFontSize 8, 34

ret

drawI endp

drawJ proc near

add si, 22

drawFontSize 8, 50

sub si, 22

add di, 25

drawFontSize 8, 22

add di, 17

drawFontSize 22, 8

ret

drawJ endp

drawK proc near

drawFontSize 8, 50

add si, 23

drawFontSize 7, 8

sub si, 7

add di, 8

drawFontSize 7, 8

sub si, 8

add di, 8

drawFontSize 8, 8

add si, 8

add di, 8

drawFontSize 7, 8

add si, 7

add di, 8

drawFontSize 7, 16

ret

drawK endp

drawL proc near

drawFontSize 8, 50

add di, 42

drawFontSize 30, 8

ret

drawL endp

drawM proc near

drawFontSize 8, 50

add si, 8

add di, 8

drawFontSize 8, 8

add si, 8

add di, 8

drawFontSize 8, 8

add si, 8

sub di, 8

drawFontSize 8, 8

add si, 8

sub di, 8

drawFontSize 8, 50

ret

drawM endp

drawN proc near

drawFontSize 8, 50

add si, 8

add di, 8

drawFontSize 8, 8

add si, 8

add di, 8

drawFontSize 8, 8

add si, 8

add di, 8

drawFontSize 8, 16

sub di, 24

add si, 8

drawFontSize 8, 50

ret

drawN endp

drawO proc near

drawFontSize 8, 50

drawFontSize 22, 8

add di, 42

drawFontSize 22, 8

sub di, 42

add si, 22

drawFontSize 8, 50

ret

drawO endp

drawP proc near

drawFontSize 8, 50

drawFontSize 30, 8

add di, 20

drawFontSize 30, 8

sub di, 12

add si, 22

drawFontSize 8, 12

ret

drawP endp

drawQ proc near

drawFontSize 8, 50

drawFontSize 30, 8

add si, 30

drawFontSize 8, 34

sub si, 22

add di, 42

drawFontSize 14, 8

sub di, 16

add si, 6

drawFontSize 8, 8

add si, 8

add di, 8

drawFontSize 8, 8

add si, 8

add di, 8

drawFontSize 8, 8

ret

drawQ endp

drawR proc near

drawFontSize 8, 50

drawFontSize 25, 8

add di, 21

drawFontSize 25, 8

sub di, 13

add si, 25

drawFontSize 8, 13

add di, 21

drawFontSize 8, 21

ret

drawR endp

drawS proc near

call draw5

ret

drawS endp

drawT proc near

drawFontSize 34, 8

add si, 13

add di, 8

drawFontSize 8, 42

ret

drawT endp

drawU proc near

drawFontSize 8, 50

add si, 22

drawFontSize 8, 50

sub si, 14

add di, 42

drawFontSize 14, 8

ret

drawU endp

drawV proc near

drawFontSize 8, 26

add si, 32

drawFontSize 8, 26

add di, 26

sub si, 24

drawFontSize 8, 15

add si, 16

drawFontSize 8, 15

add di, 15

sub si, 8

drawFontSize 8, 8

ret

drawV endp

drawW proc near

drawFontSize 8, 26

add si, 48

drawFontSize 8, 26

sub si, 40

add di, 26

drawFontSize 8, 15

add si, 16

drawFontSize 8, 15

add si, 16

drawFontSize 8, 15

sub si, 24

add di, 15

drawFontSize 8, 8

add si, 16

drawFontSize 8, 8

ret

drawW endp

drawX proc near

drawFontSize 8, 20

add si, 22

drawFontSize 8, 20

sub si, 14

add di, 20

drawFontSize 14, 8

sub si, 8

add di, 8

drawFontSize 8, 22

add si, 22

drawFontSize 8, 22

ret

drawX endp

drawY proc near

drawFontSize 8, 20

add si, 30

drawFontSize 8, 20

sub si, 22

add di, 20

drawFontSize 8,8

add si, 14

drawFontSize 8,8

sub si, 6

add di, 8

drawFontSize 8, 22

ret

drawY endp

drawZ proc near

drawFontSize 32, 8

add si, 24

add di, 8

drawFontSize 8, 8

sub si, 8

add di, 8

drawFontSize 8, 8

sub si, 8

add di, 8

drawFontSize 8, 8

sub si, 8

add di, 8

drawFontSize 8, 8

add di, 8

drawFontSize 32, 8

ret

drawZ endp

draw0 proc near

call drawO

ret

draw0 endp

draw1 proc near

add si, 11

drawFontSize 8, 50

add si, 11

ret

draw1 endp

draw2 proc near

push di

drawFontSize 30, 8

add di, 21

drawFontSize 30, 8

add di, 21

drawFontSize 30, 8

pop di

push di

push si

add si, 22

add di, 8

drawFontSize 8, 14

pop si

pop di

add di, 28

drawFontSize 8, 14

ret

draw2 endp

draw3 proc near

push di

drawFontSize 30, 8

add di, 21

drawFontSize 22, 8

add di, 21

drawFontSize 30, 8

pop di

add si, 22

add di, 8

drawFontSize 8, 34

ret

draw3 endp

draw4 proc near

push di

drawFontSize 8, 21

add di, 21

drawFontSize 22, 8

pop di

add si, 22

drawFontSize 8, 50

ret

draw4 endp

draw5 proc near

push di

drawFontSize 30, 8

add di, 21

drawFontSize 30, 8

add di, 21

drawFontSize 30, 8

pop di

add di, 8

drawFontSize 8, 14

add si, 22

add di, 20

drawFontSize 8, 14

ret

draw5 endp

draw6 proc near

push di

drawFontSize 30, 8

add di, 21

drawFontSize 30, 8

add di, 21

drawFontSize 30, 8

pop di

add di, 8

drawFontSize 8, 34

add si, 22

add di, 20

drawFontSize 8, 14

ret

draw6 endp

draw7 proc near

drawFontSize 30, 8

add si, 22

drawFontSize 8, 50

ret

draw7 endp

draw8 proc near

push si

drawFontSize 8, 50

add si, 22

drawFontSize 8, 50

pop si

add si, 8

drawFontSize 14, 8

add di, 21

drawFontSize 14, 8

add di, 21

drawFontSize 14, 8

ret

draw8 endp

draw9 proc near

push si

drawFontSize 8, 29

add si, 22

drawFontSize 8, 50

pop si

add si, 8

drawFontSize 14, 8

add di, 21

drawFontSize 14, 8

ret

draw9 endp

# Sample Output



Fig. 1 Above illustration is the Main Menu which includes three selectable options: Play, Help and Exit.



Fig.2 In game graphics



Fig. 3 Game Over

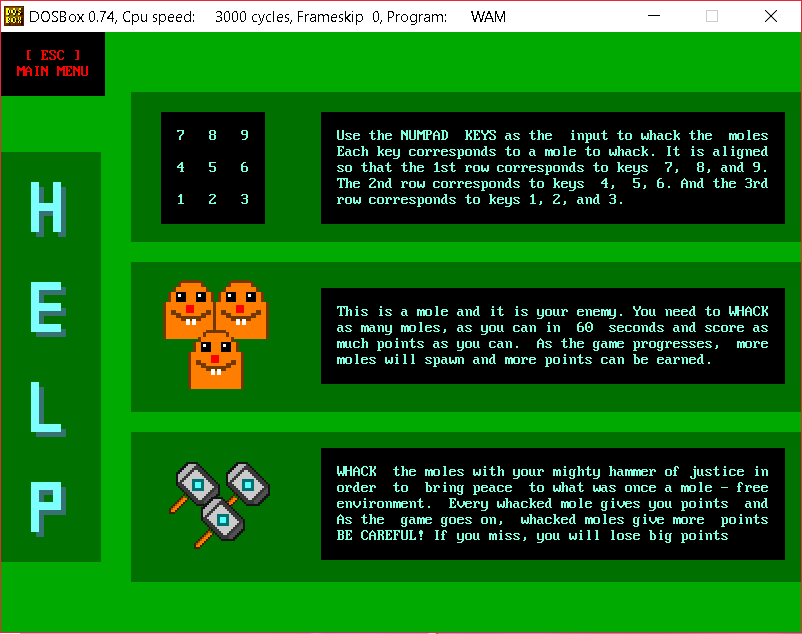


Fig. 4 Help Menu displays the isntructions or mechanics of the game.

# Recommendations

The software project Whack a Mole has met the necessary objectives and expected outcomes. Despite of its successful implementation and result, the machine program still contains minimal bugs and the program itself can also be further improved. In order to reduce the impact of the given issue, the program needs to conduct a proper structure of codes to avoid the said bug where in the mallet is left in the hole when the user was not able to whack a mole and after some time it will eventually disappear. Whereas the mallet should be disappeared right away even if the user was not able to whack a mole. This kind of bug arises due to the fact that the program is running too many codes and cannot run all of them simultaneously at the same time which then results to the said problem that these lines of code have to wait to be implemented before the mallet itself disappears in the hole. In addition, it should also implement mouse operations as a tool in selecting options in the main menu and whacking a mole in the gameplay mode. The program should implement INT 33H functions in assembly language programming to use the said operation.

Regarding the scoring algorithm of the game, it may seem unnoticeable but as the player keeps repeatedly hitting a spawned mole, it continues to add to the score for as long as the mole is active. The incrementation of the score should only be done once per hit and thus there should be only one allowable hit for every spawn of each mole.

When the player keeps on giving inputs, (hitting the moles) repeatedly fast, it may overrun the execution of other procedures. Instead of spawning new set of moles in the next second, it will continue in the current cycle. Because the jumps to the labels and calling of procedures were out jumped or passed out. Too many processes are executed in a very short span of time, that it doesn't have enough time to execute all of it.