|  |
| --- |
|  |
| ***Ping Pong Game***  ***NUMAN ASGHAR*** |
| |  |  |  | | --- | --- | --- | | [Author name] | [Date] | [Course title] | |

**Ping Pong Game Report**

**Introduction**

The **Ping Pong game** is one of the most iconic video games ever created. It simulates a simple tennis-like game where two players use paddles to bounce a ball back and forth. This project aims to implement a basic Pong game in assembly language, designed to run on a DOS-like environment using direct memory manipulation, video output to the screen, and keyboard input. The game involves player-controlled paddles that interact with a ball, scoring points as the ball passes the opponent’s paddle.

This assembly-based Pong game offers a unique learning experience on low-level programming, hardware interaction, and the intricacies of video memory manipulation, as well as providing an engaging gameplay experience.

**Implementation Details**

The implementation of the Pong game involves several key components:

1. **Video Memory Management**:
   * The game uses **B800h** segment for video memory to manipulate the screen. Each character cell in video memory has two bytes: one for the character and one for its attribute (e.g., color).
   * The game's screen is cleared and redrawn every time the ball or paddles move, ensuring smooth animation.
2. **Game Entities**:
   * **Ball**: Represented by a specific character (0x0C6F), the ball’s movement is controlled by changing its position in the video memory based on velocity components (vx for horizontal, vy for vertical).
   * **Paddles**: Each player’s paddle is also represented by specific characters (paddle1: 0x0DB2 and paddle2: 0x0EB2), and their positions are manipulated based on player input.
3. **User Input**:
   * Keyboard input is read using BIOS interrupts, where players control their paddles with the **WASD keys** for player 1 and the **Arrow keys** for player 2. The input handling uses int 16h BIOS services.
4. **Collision Detection**:
   * The ball's movement is checked for collisions with the paddles, walls, or other game boundaries. When a collision is detected, the ball’s direction is reversed.
   * If the ball moves past a paddle (i.e., out of bounds), the opponent scores a point.
5. **Score Tracking and Display**:
   * Scores are maintained for both players and displayed at the top of the screen. When a player scores, the display is updated.
6. **Game Flow**:
   * The game checks for player input, updates the positions of paddles and the ball, detects collisions, and redraws the game state. It also checks for a win condition when a player's score reaches 13.
7. **Endgame Conditions**:
   * If either player reaches 13 points, a win message is displayed, and the game waits for a key press before restarting.

**Challenges**

1. **Memory Management**:
   * Manipulating video memory directly in assembly can be tricky, especially when working with custom characters or handling multiple graphical elements at once. Proper alignment and memory addressing are critical to avoid overwriting unintended sections of memory.
2. **Input Handling**:
   * Detecting and interpreting user input efficiently with the BIOS interrupts can sometimes lead to difficulties in distinguishing between keys or handling simultaneous key presses (e.g., diagonal movement).
3. **Collision Detection**:
   * The game’s collision detection logic requires precise checking to ensure the ball interacts correctly with the paddles and walls. Special care had to be taken to handle boundary conditions (e.g., ensuring the ball doesn’t move beyond the screen edges).
4. **Game Synchronization**:
   * Since the game is running in real-time, ensuring smooth and consistent updates for the ball and paddle movement while preventing lag and glitches can be difficult in assembly language. The int 16h interrupt for input handling and constant screen updates require careful synchronization.
5. **Handling Win Conditions**:
   * Detecting and displaying the win condition when a player reaches 13 points involves updating the game state and ensuring the game halts gracefully before restarting. Handling multiple state transitions was one of the more complex parts of the implementation.

**Testing**

During testing, several aspects of the game were examined for correctness and performance:

1. **Functional Testing**:
   * Verified that the paddles respond correctly to player input, including moving up, down, left, and right.
   * Checked ball movement and ensured that it bounces off paddles and walls as expected.
   * Ensured that points are correctly awarded when the ball passes the opponent’s paddle.
2. **Boundary Testing**:
   * Tested the ball's behavior at the edges of the screen and ensured that collisions with the walls were handled properly.
   * Ensured that paddle movement stops at the screen boundaries to prevent paddles from moving off-screen.
3. **Win Condition**:
   * Validated that the game correctly recognizes when a player reaches 13 points and displays the appropriate win message.
4. **Performance Testing**:
   * The game was tested for responsiveness, ensuring smooth gameplay without noticeable lag. The ball’s speed was also tested at different intervals to ensure consistency.

**Guide**

To play the Pong game:

1. **Start the Game**:
   * Run the program in a DOS-like environment. You will be greeted with a welcome screen, and the game will start after pressing any key.
2. **Controls**:
   * **Player 1 (left side)**: Use **W** (Up), **A** (Left), **S** (Down), and **D** (Right) to move the paddle.
   * **Player 2 (right side)**: Use the **Arrow keys** (Up, Down, Left, Right) to control the second paddle.
3. **Scoring**:
   * Each player earns a point when the ball passes the opponent's paddle.
   * The score is displayed at the top of the screen for both players.
4. **Win Condition**:
   * The game ends when one player reaches 13 points. A message will display the winner's name along with the message "has won!".
   * The game pauses to await a key press before restarting.
5. **Pausing the Game**:
   * Press the **P** key to pause the game and any other key to resume.
6. **Ending the Game**:
   * Press **Esc** to quit the game at any time.

**Conclusion**

This assembly Pong game represents a fun and educational project that showcases the power and complexity of low-level programming. By directly manipulating video memory, handling keyboard inputs, and managing real-time game updates, the game offers an insightful look into the fundamentals of game development at the hardware level. Despite the challenges encountered, such as memory management and collision detection, the game provides an enjoyable gameplay experience and a deep understanding of assembly language programming.

**Game Code**

[org 100h]

jmp Start

; direction vector of velocity

vy: dw -1 ; y component of velocity vector

vx: dw -2 ; x component of velocity vector

;design

ball: dw 0x0C6F ; character used to represent the ball

paddle1: dw 0x0DB2 ; character used to build paddle1

paddle2: dw 0x0EB2 ; character used to build paddle2

;coordinates

paddle1Start: dw 1606 ; location of paddle1

paddle2Start: dw 1752 ; location of paddle2

;scores

p1Score: dw 0 ; score of paddle1

p2Score: dw 0 ; score of paddle2

;names

P1Name: db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

P1NameLength: dw 0

P2Name: db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

P2NameLength: dw 0

speed: dw 3;

CLS:

pushAD

push es

mov ax, 0xB800

push ax

pop es

mov di, 0

mov cx, 2000

mov ax, 0x0720

cld

rep STOSW

pop es

popAD

ret

WelcomeScreen:

pushAD ; Save all registers

push es ; Save ES register

mov ax, 0xB800 ; Access video memory

mov es, ax

; Clear the screen

call CLS

; Center the welcome text

mov di, 1976 ; Position near the center of the screen (row 12, column 40)

mov si, WelcomeMsg ; Load the message string

mov cx, WelcomeMsgL ; Load the message length

mov ah, 0x0D ; Bright Yellow attribute

WelcomeLoop:

lodsb ; Load the next character

stosw ; Store character and attribute

loop WelcomeLoop ; Repeat until end of message

; Wait for user input to continue

call WaitForKeyPress

pop es ; Restore ES register

popAD ; Restore all registers

ret

WaitForKeyPress:

mov ah, 00h ; BIOS interrupt for keyboard input

int 16h ; Wait for a key press

ret

WelcomeMsg db 'WELCOME TO PING PONG GAME', 0

WelcomeMsgL equ $-WelcomeMsg

printboard:

push bp

mov bp, sp

pushAD

push ds

push es

mov ax, 0xB800

push ax

pop es

;names

mov di, 40

mov si, P1Name

mov cx, [P1NameLength]

mov ax, 0x0900

LoopName1:

lodsb

stosw

loop LoopName1

mov di, 80

mov ah, 0x09

mov al, '|'

stosw

mov di, 100

mov si, P2Name

mov cx, [P2NameLength]

LoopName2:

lodsb

stosw

loop LoopName2

;board

mov di, 160

mov ax, 0x07C9

STOSW

mov cx, 78

mov ax, 0x07CD

cld

rep STOSW

mov ax, 0x07BB

STOSW

mov dx, 21

L1:

mov ax, 0x07BA

STOSW

add di, 156

mov ax, 0x07BA

STOSW

dec dx

cmp dx, 0

jne L1

mov ax, 0x07C8

STOSW

mov cx, 78

mov ax, 0x07CD

cld

rep STOSW

mov ax, 0x07BC

STOSW

pop es

pop ds

popAD

pop bp

ret

scoreDisplay:

pushAD

push es

mov ax, 0xB800

mov es, ax

mov di, 0

mov ax, [p1Score]

mov bx, 10

mov cx, 0

LDigit:

mov dx, 0

div bx

push dx

inc cx

cmp ax, 0

jne LDigit

nextDigit:

pop ax

mov ah, 0x0A

add al, 0x30

stosw

loop nextDigit

mov di, 158

mov si, [p2Score]

std

LDigitR:

mov dx, 0

mov ax, si

div bx

mov si, ax

mov ax, dx

add ax, 0x0A30

stosw

cmp si, 0

jne LDigitR

cld

cmp WORD [p1Score], 13

jge callWin1

cmp WORD [p2Score], 13

jge callWin2

pop es

popAD

ret

messageWin: db ' has won!'

messageWinL: dw 9

callWin1:

call CLS

call printboard

mov di, 1820

mov si, P1Name

mov cx, [P1NameLength]

mov ax, 0x0C00

LoopNameP1:

lodsb

stosw

loop LoopNameP1

mov si, messageWin

mov cx, [messageWinL]

mov ax, 0x0C00

LoopWinMessage1:

lodsb

stosw

loop LoopWinMessage1

pop es

popAD

pop bp

mov ax, 0

int 16h

jmp END

callWin2:

call CLS

call printboard

mov di, 1820

mov si, P2Name

mov cx, [P2NameLength]

mov ax, 0x0C00

LoopNameP2:

lodsb

stosw

loop LoopNameP2

mov si, messageWin

mov cx, [messageWinL]

mov ax, 0x0C00

LoopWinMessage2:

lodsb

stosw

loop LoopWinMessage2

pop es

popAD

pop bp

mov ax, 0

int 16h

jmp END

spawnBall:

pushAD

push es

mov ax, 0xB800

mov es, ax

mov di, 1838

mov ax, [ball]

STOSW

pop es

popAD

ret

deSpawnBall:

pushAD

push es

mov ax, 0xB800

push ax

pop es

mov di, 0

mov cx, 2000

mov ax, 0

loopFindBall:

mov ax, [es:di]

cmp ax, [ball]

je foundToDeSpawn

add di, 2

loop loopFindBall

foundToDeSpawn:

mov WORD [es:di], 0x0720

pop es

popAD

ret

spawnPaddle:

push bp

mov bp, sp

pushAD

push ds

push es

mov ax, 0xB800

mov es, ax

mov di, [paddle1Start]

mov ax, [paddle1]

mov cx, 5

sub di, 158

printPaddleLoop1:

add di, 158

STOSW

loop printPaddleLoop1

mov di, [paddle2Start]

mov ax, [paddle2]

mov cx, 5

sub di, 158

printPaddleLoop2:

add di, 158

STOSW

loop printPaddleLoop2

pop es

pop ds

popAD

pop bp

ret

movePaddle1Up:

push bp

mov bp, sp

pushAD

push es

mov ax, 0xB800

push ax

pop es

mov di, [paddle1Start]

cmp WORD [es:di-160], 0x0720

jne NoMovP1Up

sub WORD [paddle1Start], 160

mov di, [paddle1Start]

NoMovP1Up:

mov ax, [paddle1]

mov cx, 5

printPaddleLoopP1UP:

STOSW

add di, 158

loop printPaddleLoopP1UP

mov ax, 0x0720

STOSW

pop es

popAD

pop bp

ret

movePaddle1Down:

push bp

mov bp, sp

pushAD

push es

mov ax, 0xB800

push ax

pop es

mov di, [paddle1Start]

add di, 800

cmp WORD [es:di], 0x0720

jne NoMovP1Down

mov di, [paddle1Start]

mov ax, 0x0720

STOSW

add WORD [paddle1Start], 160

NoMovP1Down:

mov di, [paddle1Start]

mov ax, [paddle1]

mov cx, 5

printPaddleLoopP1Down:

STOSW

add di, 158

loop printPaddleLoopP1Down

pop es

popAD

pop bp

ret

movePaddle1Left:

push bp

mov bp, sp

pushAD

push es

mov ax, 0xB800

push ax

pop es

mov di, [paddle1Start]

cmp WORD [es:di-2], 0x0720

jne NoMovP1Left

mov di, [paddle1Start]

sub WORD [paddle1Start], 2

mov ax, 0x0720

mov cx, 5

printPaddleLoopP1Left1:

STOSW

add di, 158

loop printPaddleLoopP1Left1

NoMovP1Left:

mov di, [paddle1Start]

mov ax, [paddle1]

mov cx, 5

printPaddleLoopP1Left2:

STOSW

add di, 158

loop printPaddleLoopP1Left2

pop es

popAD

pop bp

ret

movePaddle1Right:

push bp

mov bp, sp

pushAD

push es

mov ax, 0xB800

push ax

pop es

mov ax, [paddle1Start]

mov bx, 160

mov dx, 0

div bx

cmp dx, 70

jge NoMovP1Right

mov di, [paddle1Start]

cmp WORD [es:di+2], 0x0720

jne NoMovP1Right

mov di, [paddle1Start]

add WORD [paddle1Start], 2

mov ax, 0x0720

mov cx, 5

printPaddleLoopP1Right1:

STOSW

add di, 158

loop printPaddleLoopP1Right1

NoMovP1Right:

mov di, [paddle1Start]

mov ax, [paddle1]

mov cx, 5

printPaddleLoopP1Right2:

STOSW

add di, 158

loop printPaddleLoopP1Right2

pop es

popAD

pop bp

ret

movePaddle2Up:

push bp

mov bp, sp

pushAD

push es

mov ax, 0xB800

push ax

pop es

mov di, [paddle2Start]

cmp WORD [es:di-160], 0x0720

jne NoMovP2Up

sub WORD [paddle2Start], 160

mov di, [paddle2Start]

NoMovP2Up:

mov ax, [paddle2]

mov cx, 5

printPaddleLoopP2UP:

STOSW

add di, 158

loop printPaddleLoopP2UP

mov ax, 0x0720

STOSW

pop es

popAD

pop bp

ret

movePaddle2Down:

push bp

mov bp, sp

pushAD

push es

mov ax, 0xB800

push ax

pop es

mov di, [paddle2Start]

add di, 800

cmp WORD [es:di], 0x0720

jne NoMovP2Down

mov di, [paddle2Start]

mov ax, 0x0720

STOSW

add WORD [paddle2Start], 160

NoMovP2Down:

mov di, [paddle2Start]

mov ax, [paddle2]

mov cx, 5

printPaddleLoopP2Down:

STOSW

add di, 158

loop printPaddleLoopP2Down

pop es

popAD

pop bp

ret

movePaddle2Left:

push bp

mov bp, sp

pushAD

push es

mov ax, 0xB800

push ax

pop es

mov ax, [paddle2Start]

mov bx, 160

mov dx, 0

div bx

cmp dx, 90

jle NoMovP2Left

mov di, [paddle2Start]

cmp WORD [es:di-2], 0x0720

jne NoMovP2Left

mov di, [paddle2Start]

sub WORD [paddle2Start], 2

mov ax, 0x0720

mov cx, 5

printPaddleLoopP2Left1:

STOSW

add di, 158

loop printPaddleLoopP2Left1

NoMovP2Left:

mov di, [paddle2Start]

mov ax, [paddle2]

mov cx, 5

printPaddleLoopP2Left2:

STOSW

add di, 158

loop printPaddleLoopP2Left2

pop es

popAD

pop bp

ret

movePaddle2Right:

push bp

mov bp, sp

pushAD

push es

mov ax, 0xB800

push ax

pop es

mov di, [paddle2Start]

cmp WORD [es:di+2], 0x0720

jne NoMovP2Right

mov di, [paddle2Start]

add WORD [paddle2Start], 2

mov ax, 0x0720

mov cx, 5

printPaddleLoopP2Right1:

STOSW

add di, 158

loop printPaddleLoopP2Right1

NoMovP2Right:

mov di, [paddle2Start]

mov ax, [paddle2]

mov cx, 5

printPaddleLoopP2Right2:

STOSW

add di, 158

loop printPaddleLoopP2Right2

pop es

popAD

pop bp

ret

checkInput:

push bp

mov bp, sp

pushAD

push es

mov ax, 100h

int 16h

jz noInput

mov ax, 0

int 16h

cmp al, 'w'

je callUp1

cmp al, 'a'

je callLeft1

cmp al, 's'

je callDown1

cmp al, 'd'

je callRight1

cmp al, 'W'

je callUp1

cmp al, 'A'

je callLeft1

cmp al, 'S'

je callDown1

cmp al, 'D'

je callRight1

cmp ah, 48h

je callUp2

cmp ah, 4Bh

je callLeft2

cmp ah, 50h

je callDown2

cmp ah, 4Dh

je callRight2

cmp al, 'p'

je callPause

cmp al, 'P'

je callPause

cmp al, 1Bh

je callEnd

noInput:

pop es

popAD

pop bp

ret

callUp1:

call movePaddle1Up

jmp noInput

callLeft1:

call movePaddle1Left

jmp noInput

callDown1:

call movePaddle1Down

jmp noInput

callRight1:

call movePaddle1Right

jmp noInput

callUp2:

call movePaddle2Up

jmp noInput

callLeft2:

call movePaddle2Left

jmp noInput

callDown2:

call movePaddle2Down

jmp noInput

callRight2:

call movePaddle2Right

jmp noInput

callPause:

mov ax, 0

int 16h

cmp al, 'p'

je noInput

cmp al, 'P'

je noInput

jmp callPause

callEnd:

pop es

popAD

pop bp

pop bp

jmp END

checkCollision:

push bp

mov bp, sp

sub sp, 2

pushAD

push es

mov ax, 0xB800

push ax

pop es

mov WORD [bp-2], 0

mov di, 0

mov cx, 2000

mov ax, 0

L2:

mov ax, [es:di]

add di, 2

cmp ax, [ball]

je found

loop L2

call spawnBall

found:

cmp WORD [vx], 0

jg xpositive

sub di, 4

xpositive:

cmp WORD [vy], 0

je checkLocation

cmp WORD [vy], 0

jg ypositive

add di, 160

jmp checkLocation

ypositive:

sub di, 160

checkLocation:

mov ax, 0x0720

cmp ax, [es:di]

je noColission

mov WORD [bp-2], 1

noColission:

pop es

popAD

pop ax

pop bp

ret

flag: db 0

moveBall:

push bp

mov bp, sp

pushAD

push es

mov ax, 0xB800

push ax

pop es

NOT BYTE [flag]

cmp BYTE [flag], 0

jne DontMove

mov di, 0

mov cx, 2000

mov ax, 0

L3:

mov ax, [es:di]

add di, 2

cmp ax, [ball]

je found1

loop L3

found1:

sub di, 2

mov ax, 0x0720

stosw

cmp WORD [vx], 0

jg xpositive1

sub di, 4

xpositive1:

cmp WORD [vy], 0

je movIT

cmp WORD [vy], 0

jg ypositive1

add di, 160

jmp movIT

ypositive1:

sub di, 160

movIT:

mov ax, [ball]

stosw

DontMove:

pop es

popAD

pop bp

ret

collisionHandler:

push bp

mov bp, sp

sub sp, 2

pushAD

push es

mov ax, 0xB800

push ax

pop es

mov WORD [bp-2], 0

mov di, 0

mov cx, 2000

mov ax, 0

L4:

mov ax, [es:di]

add di, 2

cmp ax, [ball]

je found2

loop L4

found2:

cmp WORD [vx], 0

jg xpositive2

sub di, 4

xpositive2:

cmp WORD [es:di], 0x0720

je yesForward

NOT WORD [vx]

inc WORD [vx]

cmp WORD [es:di], 0x07BA

je countScore

jmp noColission2

countScore:

call deSpawnBall

cmp WORD [vx], 0

jg player2score

inc WORD [p1Score]

jmp noColission

player2score:

inc WORD [p2Score]

jmp noColission2

yesForward:

cmp WORD [vy], 0

je noColission2

cmp WORD [vy], 0

jg ypositive2

add di, 160

cmp WORD [es:di], 0x0720

je noColission2

NOT WORD [vy]

inc WORD [vy]

jmp noColission2

ypositive2:

sub di, 160

cmp WORD [es:di], 0x0720

je noColission2

NOT WORD [vy]

inc WORD [vy]

noColission2:

pop es

popAD

pop ax

pop bp

ret

Start:

call WelcomeScreen

call CLS

call getNameInput

call CLS

call printboard

call spawnBall

call spawnPaddle

game:

call scoreDisplay

call checkInput

mov cx, [speed]

delayGiver:

call Delay

loop delayGiver

call checkCollision

cmp ax, 1

je callCollisionHandler

call moveBall

callCollisionHandler:

call collisionHandler

jmp game

END:

mov ax, 4c00h

int 21h

Delay:

pushAD

mov cx, 0x4000

delayLoop:

add ax, ax

loop delayLoop

popAD

ret

message1 : db 'Enter Name of Player 1 : '

message2 : db 'Enter Name of Player 2 : '

messageL : dw 25

getNameInput:

pushAD

push es

mov ax, 0xB800

mov es, ax

mov di, 160

mov si, message1

mov ax, 0x0700

mov cx, [messageL]

LoopPrintMessage1:

lodsb

stosw

loop LoopPrintMessage1

mov si, P1Name

mov cx, 20

loopInput1:

mov ax, 0

int 16h

cmp al, 0x0D

je endLoopInput1

mov ah, 0x07

stosw

mov [si], al

inc si

inc WORD [P1NameLength]

loop loopInput1

endLoopInput1:

call CLS

mov di, 160

mov si, message2

mov ax, 0x0700

mov cx, [messageL]

LoopPrintMessage2:

lodsb

stosw

loop LoopPrintMessage2

mov si, P2Name

mov cx, 20

loopInput2:

mov ax, 0

int 16h

cmp al, 0x0D

je endLoopInput2

mov ah, 0x07

stosw

mov [si], al

inc si

inc WORD [P2NameLength]

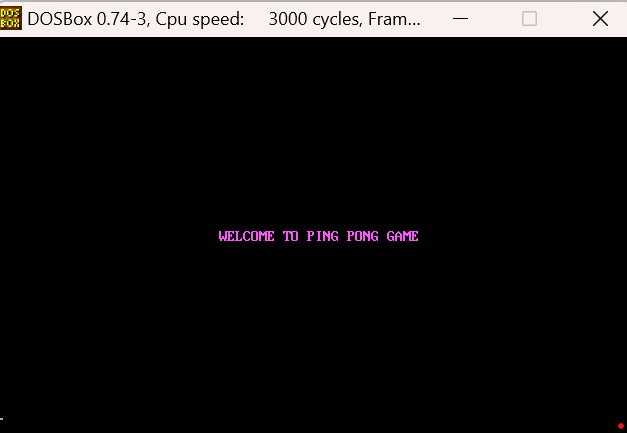
loop loopInput2

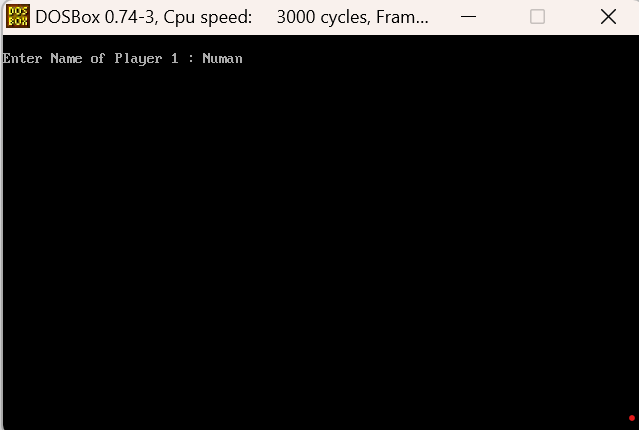
endLoopInput2:

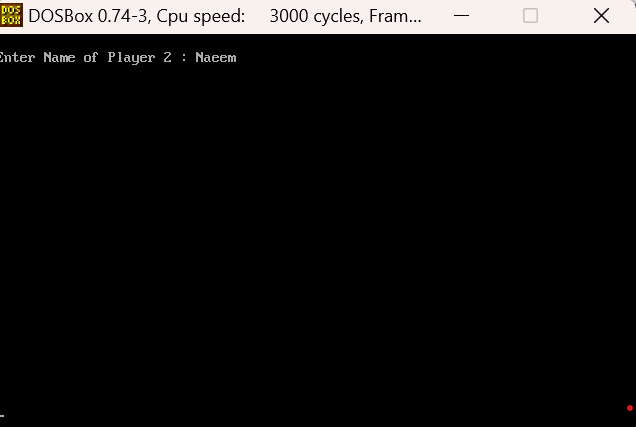
pop es

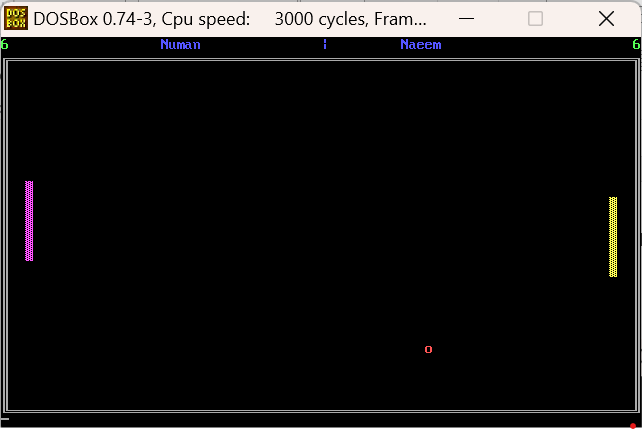
popAD

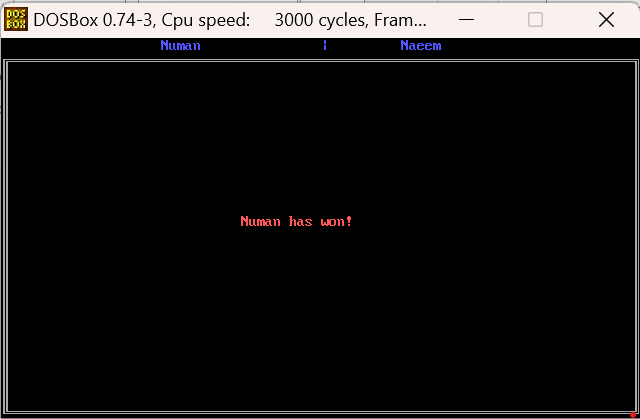
ret











Working video of it

