COMSATS University Islamabad (CUI)

Project Proposal

For

Snake Game

Kulsoom Khurshid CIIT/SP20-BCS-044/ISB

Course

Microprocessor and Assembly Language

Professor

Taimur Shahzad

Bachelor of Science in Computer Science (2020-2024)

Table of Contents

1. Introduction

This project is about a game that we all have been playing in our childhood, ‘the snake game’. It is not the best game to develop, but you can have an idea of how to code a simple game in assembly language using the Irvine library. This game has no special name but people call it snake game as the player maneuvers a line that grows in length if the player eats the fruit which is represented by a dot. If that line touches the boundary walls or it’s body the snake dies indicating the end of the game. The concept for this game was initiated by Arcade Game Blockade in 1976. There are many developers in the world who have developed this game leading to many versions of this game on different platforms. In 1988, it’s variant was released on Nokia Mobiles that attracted a large number of audience towards it.

The aim of this project is to have simplicity and fun. Before starting the game it asks the user for the speed and once the game is started the user moves the line i.e. snake left and right, up and down.

To successfully complete the project, the user need to have a strong understanding of the device on which it is being deployed, the platform on which it is created and to have the strong concepts of the assembly language. An algorithm is to be designed for handling various task such as asking the user for the speed, handling the line or snake according to that speed, for calculating the total score, time, etc.

2. General Overview

The snake game programmed in assembly language is a program that eats a fruit that leads to the growth of snake body and increases players score. The program is controlled by the microprocessor that executes the program written in the assembly language to carry out the various task such as snake movement, eating the fruit and eventually increasing the players score. The hardware components that are used are the keyboard keys such as W, A, S, D, display screen and the game mechanism that is controlled by microprocessor and assembly language program that allows the user to interact with the hardware and the program.

The assembly language program includes the instruction for the handling various task such as moving the snake, score count, time count, etc. The program also handle the situations when the snake hits the wall or touches it’s body itself.

The program is designed in a way that every time the snake touches the dot the score gets increased by 1.

3. Background

Assembly language is a low-level programming language that is used to write programs that are executed directly by the processor. It is considered a "low-level" language because it is closer to the machine language that the processor can execute directly and is more difficult to read and understand than higher level languages like C or Python. Assembly language is often used in systems where speed and memory efficiency are important, such as in embedded systems or real-time systems. Snake game is an example of such program that might benefit from using assembly language because they often have limited resources and require fast response times to Handle flow of the program.

The snake game is one of the game that was introduced in mid 90’s and played by almost all of us. The game was first introduced in Nokia phones which helped them to gain many more customers. The game looks quite simple but it is not that simple to develop many efficient algorithms were used for it’s working. Hence the use of assembly language to program this game increases its efficiency and reliability which is important for the seamless user experience.

4. Description

The snake game by its name it is obvious that its about a snake that moves in the boundary and looks for its food that is displayed in the form of a dot. As the snake hits this dot, its length increases and the score get incremented by 1. More hit means the difficulty level increase as the snake body itself is the biggest obstacle in the game.

The snake movement is controlled by the keyboard key. W is used to move the snake forward, S for moving the snake downwards, A to turn left and D to turn right. These are constants and you can program according to will. For instance, you can use the keyboard arrow keys for the movement as well but that totally depends on the structure of the program.

The aim of the game is to collect as many dots as possible without hitting the boundary walls or the snake body itself as the leads to the end of the game. As the snake collects the dots the body increases along with the score more chances of crashing to its body itself. After a certain time you have collected the enough food the game moves to next level which is harder than the previous as the snake is long and the speed is also increase and the amount of food to collect to progress through the level gets larger.

Player gets score on the amount of food or dots it took. The speed depends on the user as the user decides in the beginning whether he wants fast, medium or slow. In my program, 1 is for fast, 2 is for medium and 3 is for slow. There is no option for retrieve if once the game is end. As you hit the obstacle the game ends the score is displayed to the user and ask the user if he wants to play again if yes press 1 else 0 to finish the program execution. Playing again option doesn't means the game begins from that position it starts a totally new game and the score is reset to 0. Make sure the capslock is off as it doesn't work for those keys.

Since this game is developed in the assembly language, I have used procedures for implementing the task. Following are some of the procedures that are used in the program.

DrawWall	It is the part of the code that is called when the program is moved to the line call DrawWall is executed it automatically moves the flow to this label where the boundary wall for the game is created.
DrawSnake	It is responsible for drawing the snake. In this procedure another procedure called UpdatePlayer is called that is responsible for updating the snake body.
CreateRandomCoin	This procedure is responsible for initializing the coins at random positions for the snake in the board.
CheckSnake	This procedure is responsible to evaluate if the snake head collides with the wall or its body. If it collides the other procedure is called.
EatingCoin	If the user eats the dot/coin this procedure is called that increments the body of snake and the score is incremented by 1.
YouDied	This procedure is called as the head collides with body or wall.
ReinitializeGame	This procedure initialize the game from beginning where the score is reset, the snake body turns to original length.

Apart from these procedure many built-in Irvine functions are used. The most important one is call function, flag registers are used to jump to specific part of code. Overall the goal of this program is to develop an efficient game for the users. This requires a careful planning and attention to detail, as well as a strong understanding of assembly language and the hardware components of the snake game. In this project, I have tried to apply the knowledge of assembly language that we learnt in the subject computer organization and architecture to complete all the tasks required. The main task of this project is to develop a simulation program of a snake game.

5. Related Work

There are many versions of this game on different platforms. Even in the assembly language we can find many versions of it but with different libraries. I have used the Irvine32 library for this project.

- Snake.io
- Worms Zone.io
- Snake '97:retro phone classic
- Snake Rival
- Snake game
- Slither.io
- Hungry snake

The names differ but all these games are same, its purpose is same as well as working. The only thing that differs is the language in which it is written and the platform on which it is being deployed.

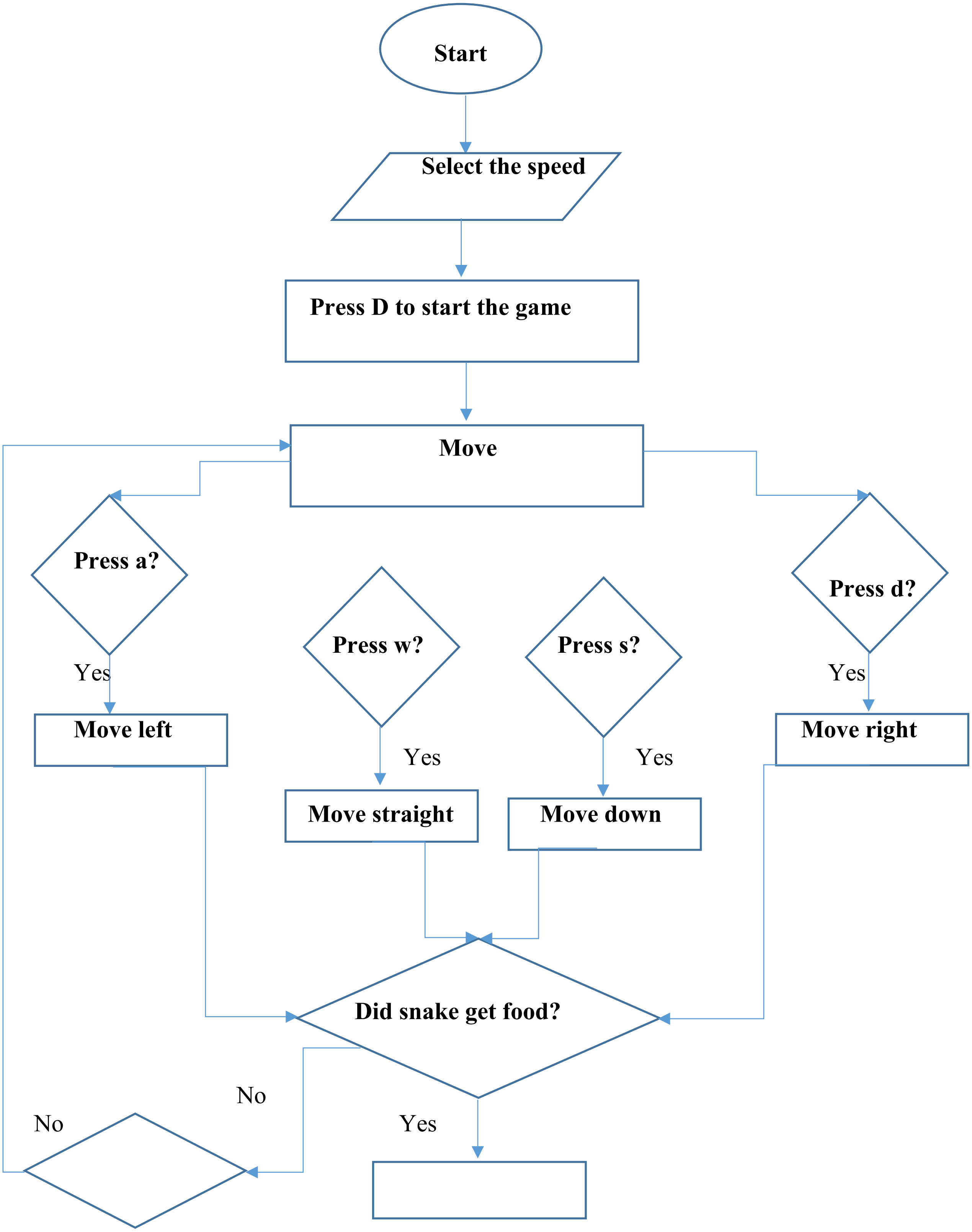
6. Flow Chart

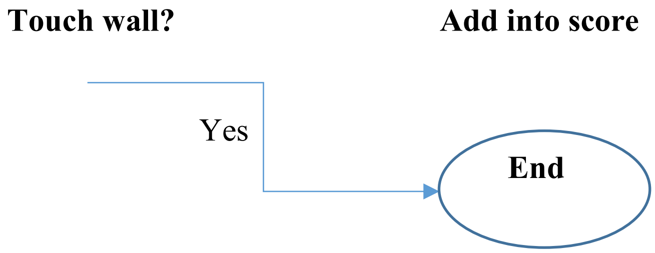
1. **Initialize**
   This initialize the game by drawing the board and asking the user for the desired speed.
2. **Draw snake**
   After the speed, the snake is drawn on the board
3. **Move Snake and the availability of coin**
   The snake moves within the board and looks for the coin as the head collides with the coin the score gets incremented and the length of the snake body is also increased.
4. **Colliding with the wall**

If the snake collide its head with the wall or with itself the game ends at that spot and users total score is displayed.

5. **Reinitialize**

   If the user opt for new game the game is reset to its original setting and the process continues.



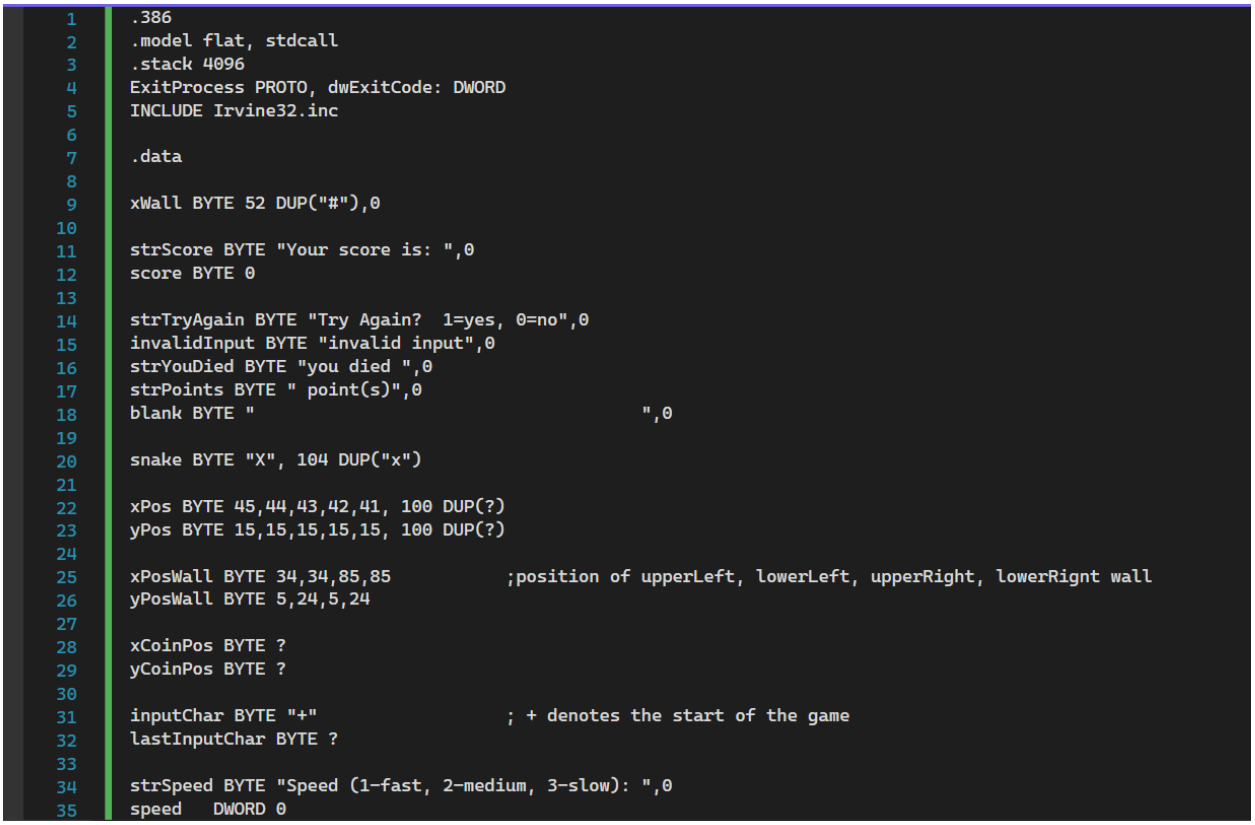

Touch wall?
Add into score
Yes
End

7. Advantage

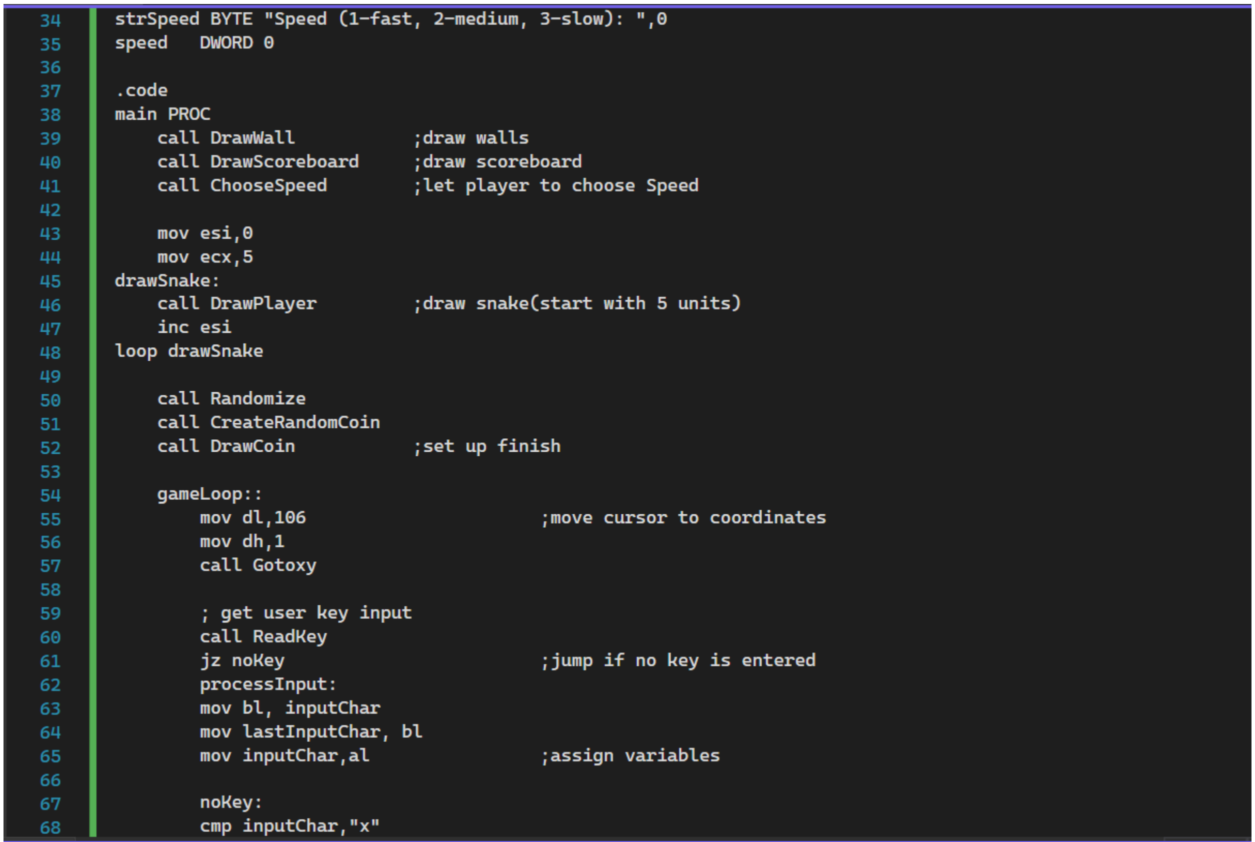
Following are some advantages of this project.

- It is an exercise of brain and improvement to problem solving.
- Relax and relief from anxiety and stress.
- Improve the skills.
- Mind training.
- Important for children for learning purposes.
- Strong memory.
- Power to decide.
- Powerful tool for children to have certain life skills.
- Memory efficiency as the assembly program requires less memory than high level languages which can be very beneficial for the systems with the limited memory resources,
- Control over the program. As the assembly language has much more control over the hardware and the code can be optimize for specific features.
- Debugging assembly languages are way easier than high level languages.

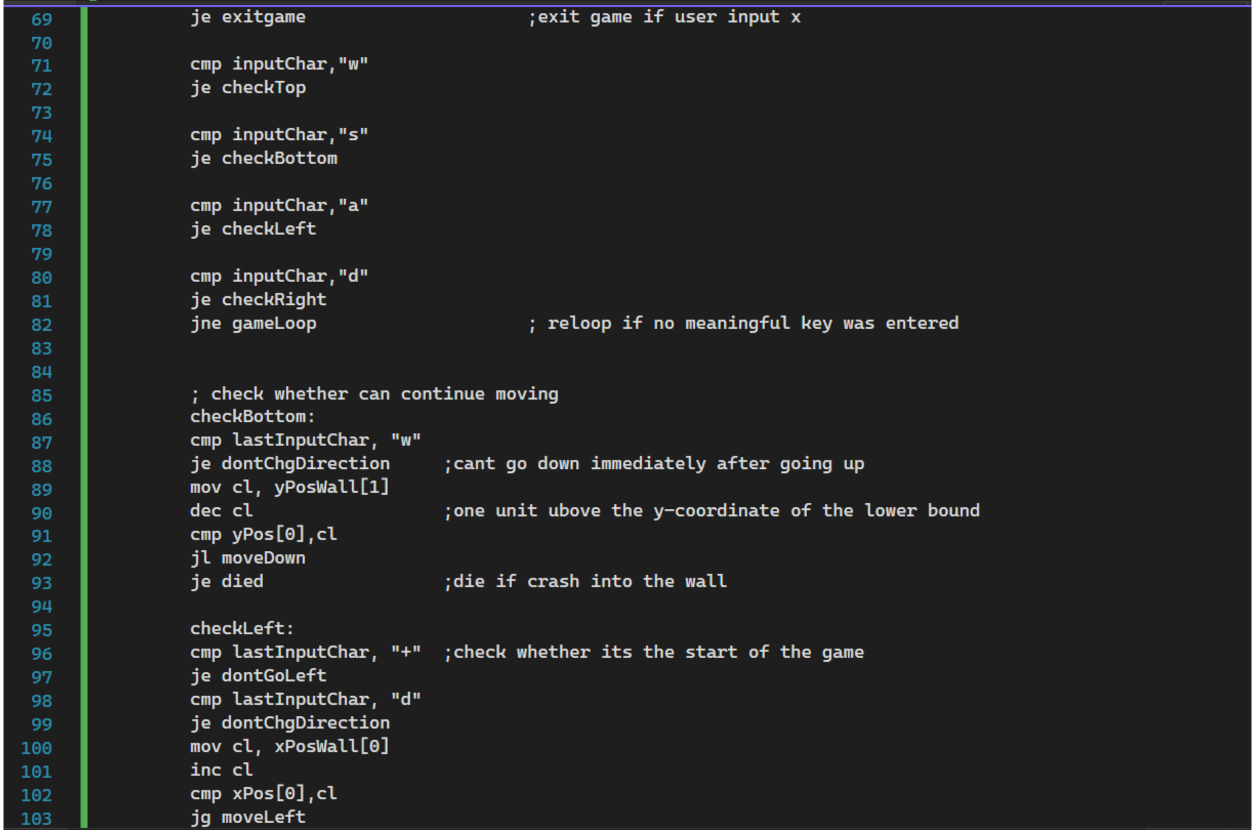
8. Code



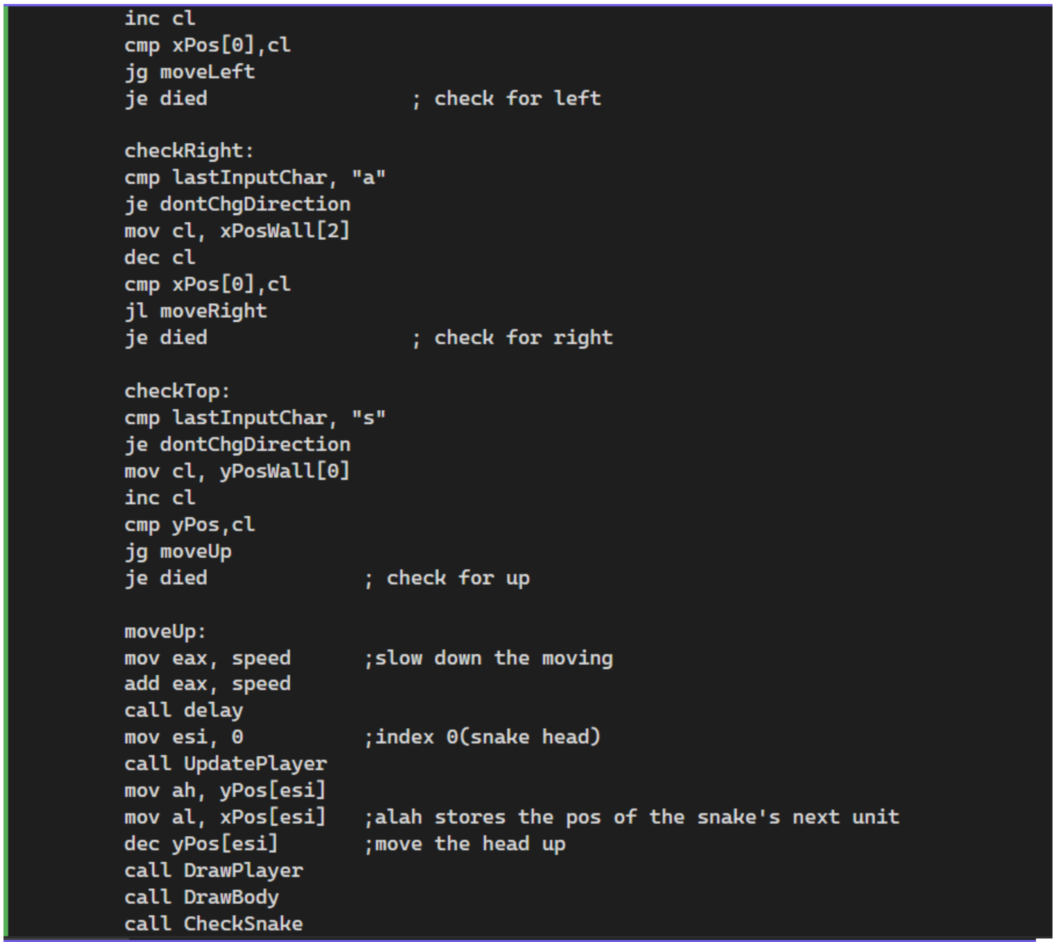
```
.386
.model flat, stdcall
.stack 4096
ExitProcess PROTO, dwExitCode: DWORD
INCLUDE Irvine32.inc

.data

xWall BYTE 52 DUP("#"),0

strScore BYTE "Your score is: ",0
score BYTE 0

strTryAgain BYTE "Try Again?  1=yes, 0=no",0
invalidInput BYTE "invalid input",0
strYouDied BYTE "you died ",0
strPoints BYTE " point(s)",0
blank BYTE "                                     ",0

snake BYTE "X", 104 DUP("x")

xPos BYTE 45,44,43,42,41, 100 DUP(?)
yPos BYTE 15,15,15,15,15, 100 DUP(?)

xPosWall BYTE 34,34,85,85           ;position of upperLeft, lowerLeft, upperRight, lowerRignt wall
yPosWall BYTE 5,24,5,24

xCoinPos BYTE ?
yCoinPos BYTE ?

inputChar BYTE "+"                  ; + denotes the start of the game
lastInputChar BYTE ?

strSpeed BYTE "Speed (1-fast, 2-medium, 3-slow): ",0
speed   DWORD 0
```



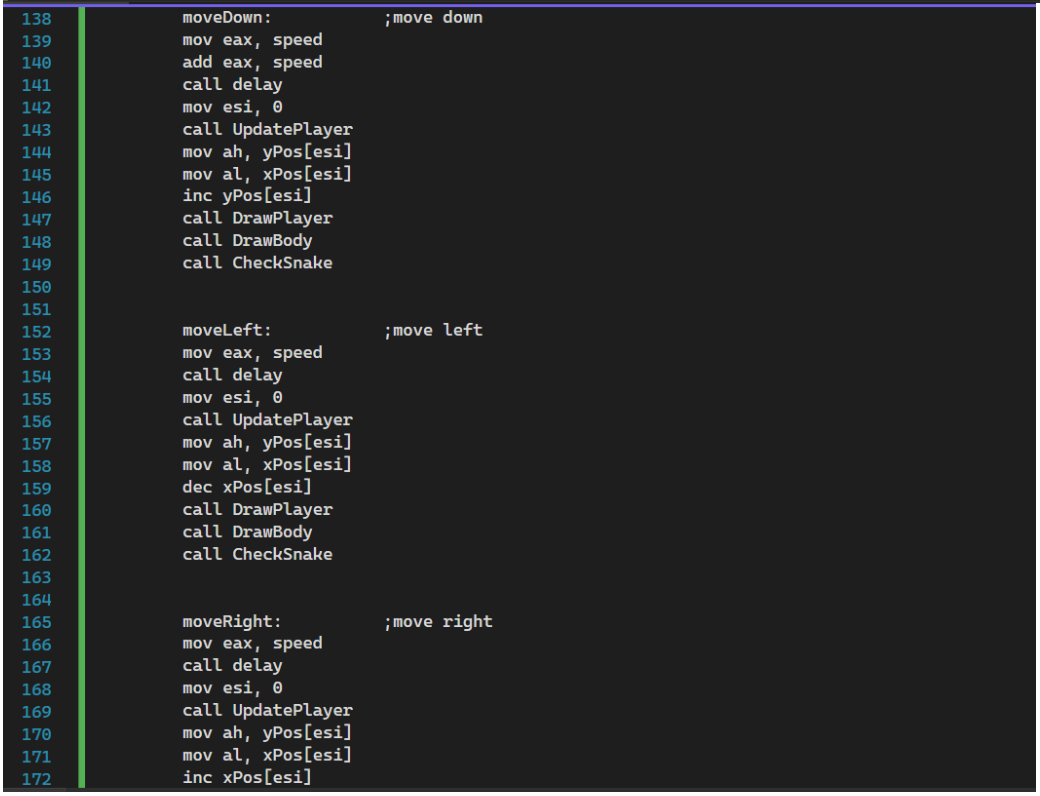
```
strSpeed BYTE "Speed (1-fast, 2-medium, 3-slow): ",0
speed   DWORD 0

.code
main PROC
    call DrawWall           ;draw walls
    call DrawScoreboard     ;draw scoreboard
    call ChooseSpeed        ;let player to choose Speed

    mov esi,0
    mov ecx,5
drawSnake:
    call DrawPlayer         ;draw snake(start with 5 units)
    inc esi
loop drawSnake

    call Randomize
    call CreateRandomCoin
    call DrawCoin           ;set up finish

    gameLoop::
        mov dl,106                      ;move cursor to coordinates
        mov dh,1
        call Gotoxy

        ; get user key input
        call ReadKey
        jz noKey                        ;jump if no key is entered
        processInput:
        mov bl, inputChar
        mov lastInputChar, bl
        mov inputChar,al                ;assign variables

        noKey:
        cmp inputChar,"x"
```



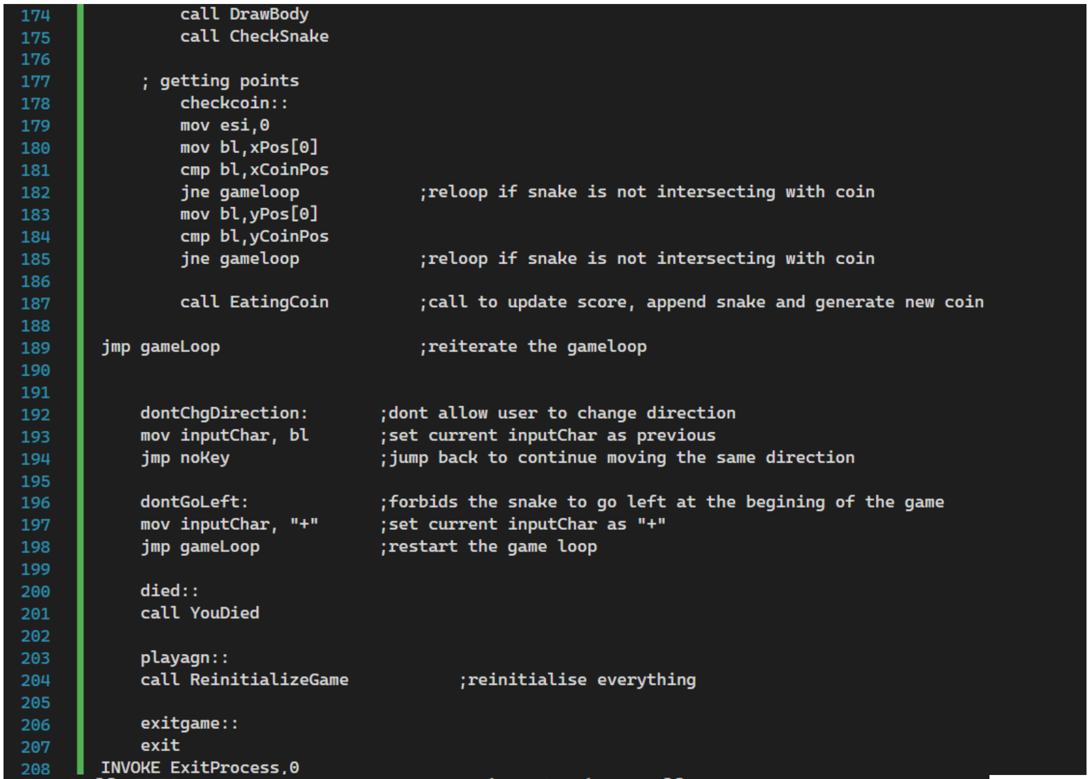
```
        je exitgame                     ;exit game if user input x

        cmp inputChar,"w"
        je checkTop

        cmp inputChar,"s"
        je checkBottom

        cmp inputChar,"a"
        je checkLeft

        cmp inputChar,"d"
        je checkRight
        jne gameLoop                    ; reloop if no meaningful key was entered


        ; check whether can continue moving
        checkBottom:
        cmp lastInputChar, "w"
        je dontChgDirection     ;cant go down immediately after going up
        mov cl, yPosWall[1]
        dec cl                  ;one unit ubove the y-coordinate of the lower bound
        cmp yPos[0],cl
        jl moveDown
        je died                 ;die if crash into the wall

        checkLeft:
        cmp lastInputChar, "+"  ;check whether its the start of the game
        je dontGoLeft
        cmp lastInputChar, "d"
        je dontChgDirection
        mov cl, xPosWall[0]
        inc cl
        cmp xPos[0],cl
        jg moveLeft
```



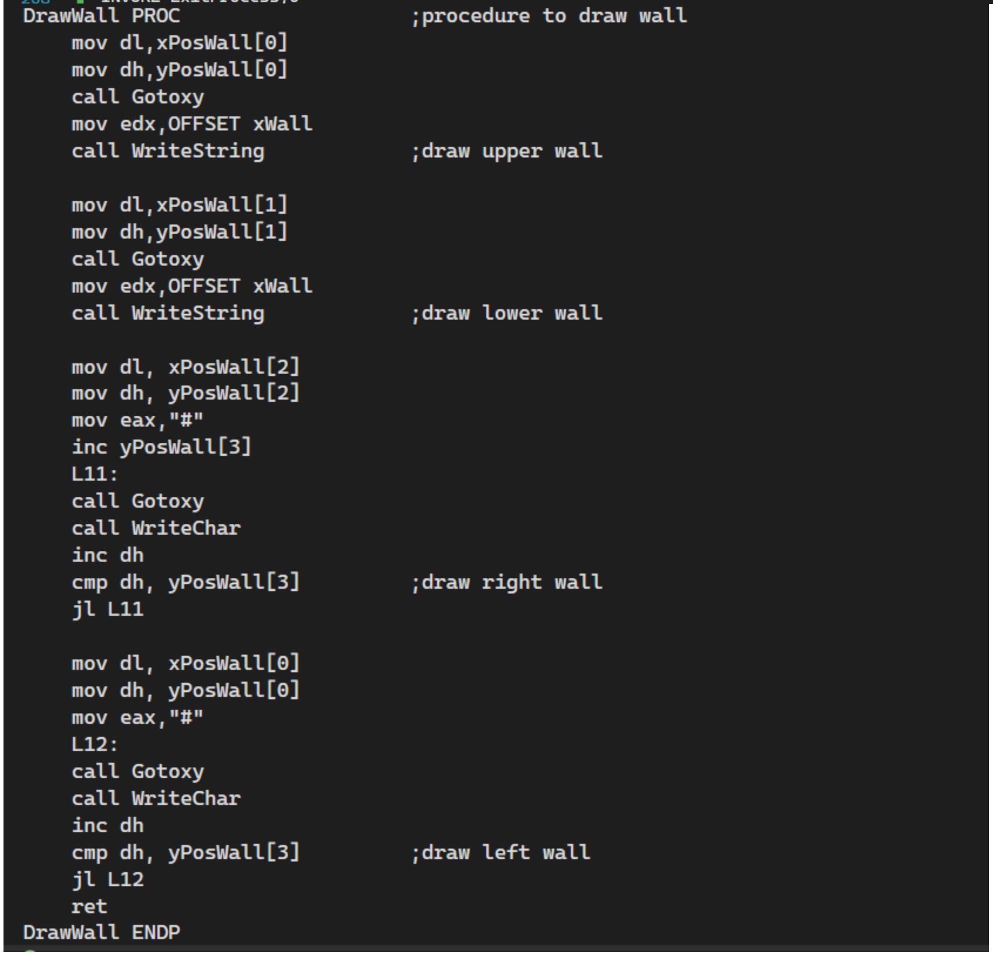
```
inc cl
cmp xPos[0],cl
jg moveLeft
je died                 ; check for left

checkRight:
cmp lastInputChar, "a"
je dontChgDirection
mov cl, xPosWall[2]
dec cl
cmp xPos[0],cl
jl moveRight
je died                 ; check for right

checkTop:
cmp lastInputChar, "s"
je dontChgDirection
mov cl, yPosWall[0]
inc cl
cmp yPos,cl
jg moveUp
je died             ; check for up

moveUp:
mov eax, speed      ;slow down the moving
add eax, speed
call delay
mov esi, 0          ;index 0(snake head)
call UpdatePlayer
mov ah, yPos[esi]
mov al, xPos[esi]   ;alah stores the pos of the snake's next unit
dec yPos[esi]       ;move the head up
call DrawPlayer
call DrawBody
call CheckSnake
```



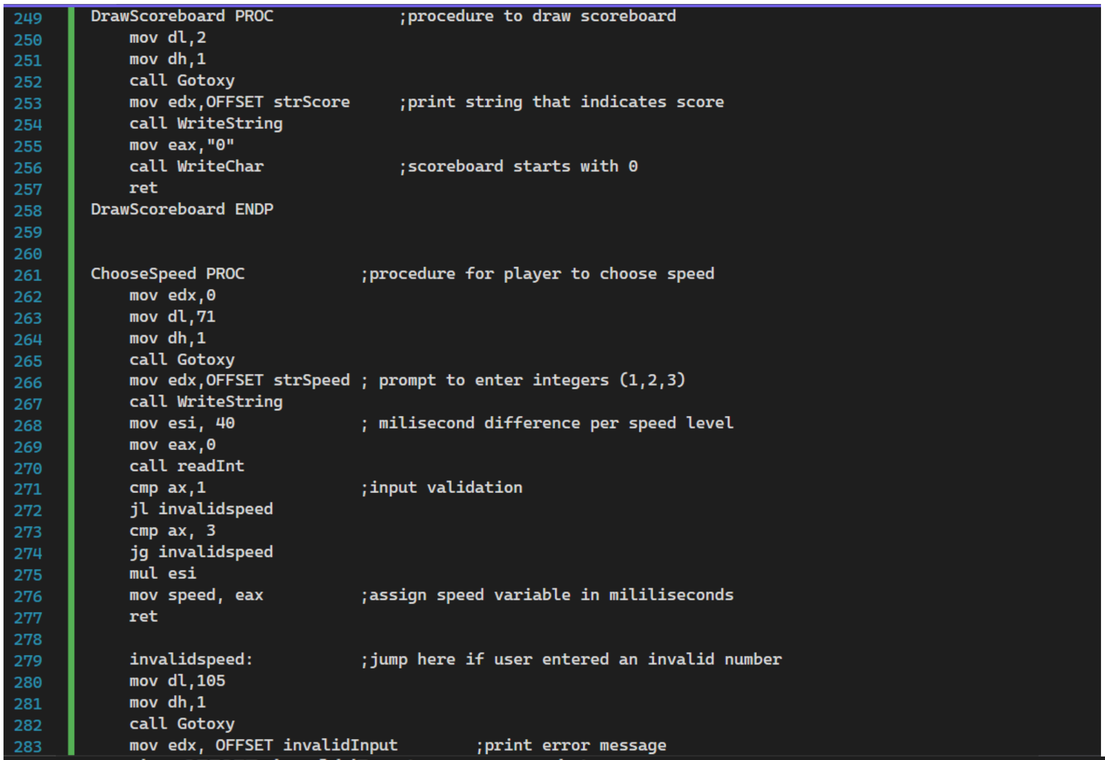
```
moveDown:           ;move down
mov eax, speed
add eax, speed
call delay
mov esi, 0
call UpdatePlayer
mov ah, yPos[esi]
mov al, xPos[esi]
inc yPos[esi]
call DrawPlayer
call DrawBody
call CheckSnake


moveLeft:           ;move left
mov eax, speed
call delay
mov esi, 0
call UpdatePlayer
mov ah, yPos[esi]
mov al, xPos[esi]
dec xPos[esi]
call DrawPlayer
call DrawBody
call CheckSnake


moveRight:          ;move right
mov eax, speed
call delay
mov esi, 0
call UpdatePlayer
mov ah, yPos[esi]
mov al, xPos[esi]
inc xPos[esi]
```



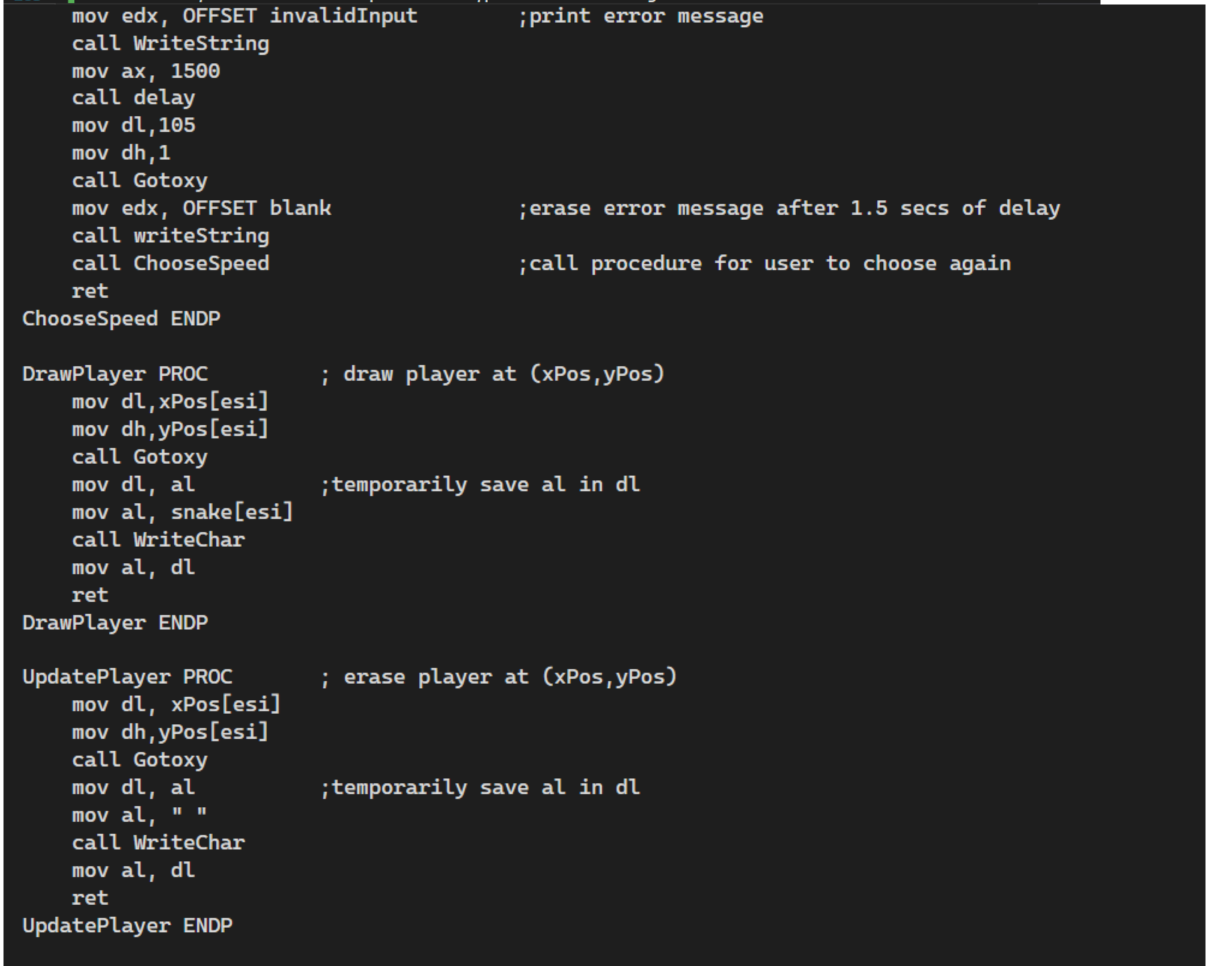
```
        call DrawBody
        call CheckSnake

    ; getting points
        checkcoin::
        mov esi,0
        mov bl,xPos[0]
        cmp bl,xCoinPos
        jne gameloop            ;reloop if snake is not intersecting with coin
        mov bl,yPos[0]
        cmp bl,yCoinPos
        jne gameloop            ;reloop if snake is not intersecting with coin

        call EatingCoin         ;call to update score, append snake and generate new coin

jmp gameLoop                    ;reiterate the gameloop


    dontChgDirection:       ;dont allow user to change direction
    mov inputChar, bl       ;set current inputChar as previous
    jmp noKey               ;jump back to continue moving the same direction

    dontGoLeft:             ;forbids the snake to go left at the begining of the game
    mov inputChar, "+"      ;set current inputChar as "+"
    jmp gameLoop            ;restart the game loop

    died::
    call YouDied

    playagn::
    call ReinitializeGame           ;reinitialise everything

    exitgame::
    exit
INVOKE ExitProcess,0
```



```
DrawWall PROC                   ;procedure to draw wall
    mov dl,xPosWall[0]
    mov dh,yPosWall[0]
    call Gotoxy
    mov edx,OFFSET xWall
    call WriteString            ;draw upper wall

    mov dl,xPosWall[1]
    mov dh,yPosWall[1]
    call Gotoxy
    mov edx,OFFSET xWall
    call WriteString            ;draw lower wall

    mov dl, xPosWall[2]
    mov dh, yPosWall[2]
    mov eax,"#"
    inc yPosWall[3]
    L11:
    call Gotoxy
    call WriteChar
    inc dh
    cmp dh, yPosWall[3]         ;draw right wall
    jl L11

    mov dl, xPosWall[0]
    mov dh, yPosWall[0]
    mov eax,"#"
    L12:
    call Gotoxy
    call WriteChar
    inc dh
    cmp dh, yPosWall[3]         ;draw left wall
    jl L12
    ret
DrawWall ENDP
```

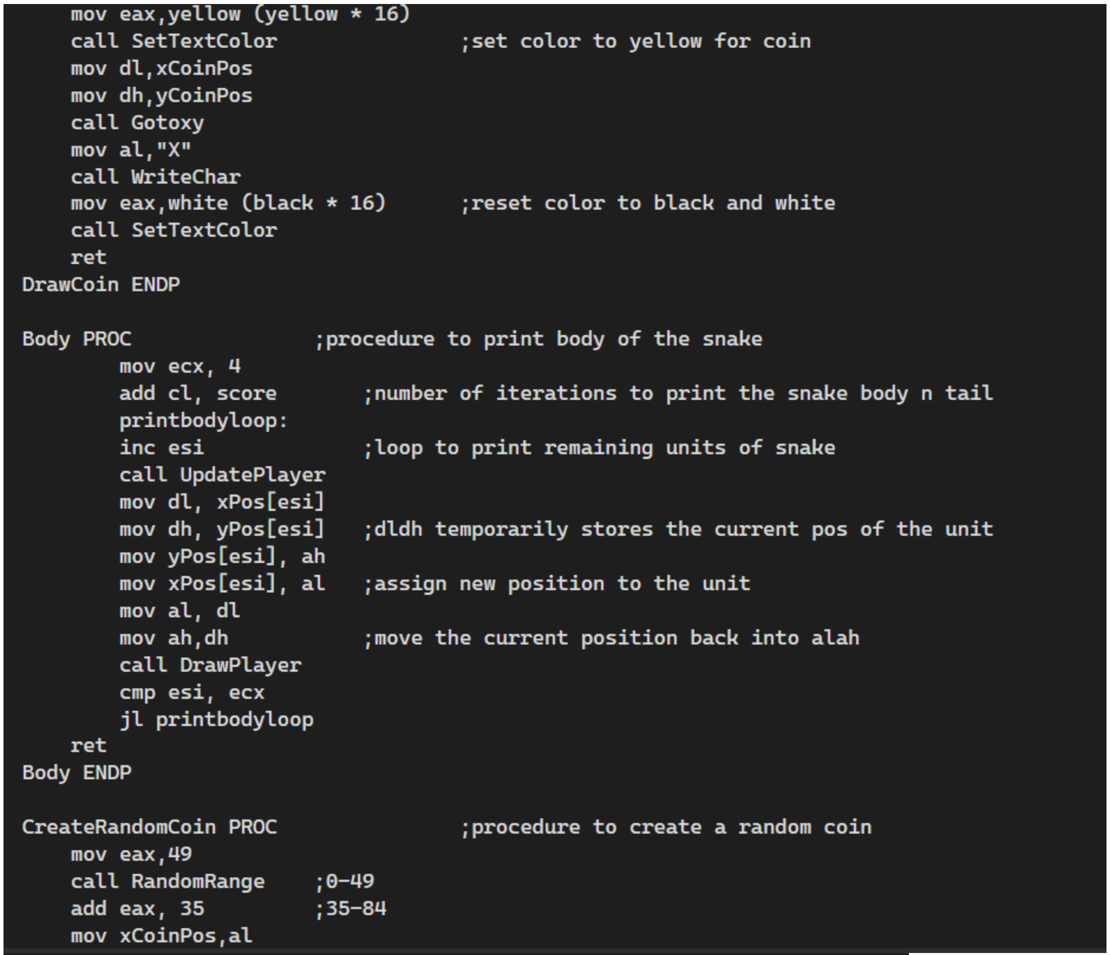


```
DrawScoreboard PROC             ;procedure to draw scoreboard
    mov dl,2
    mov dh,1
    call Gotoxy
    mov edx,OFFSET strScore     ;print string that indicates score
    call WriteString
    mov eax,"0"
    call WriteChar              ;scoreboard starts with 0
    ret
DrawScoreboard ENDP


ChooseSpeed PROC            ;procedure for player to choose speed
    mov edx,0
    mov dl,71
    mov dh,1
    call Gotoxy
    mov edx,OFFSET strSpeed ; prompt to enter integers (1,2,3)
    call WriteString
    mov esi, 40             ; milisecond difference per speed level
    mov eax,0
    call readInt
    cmp ax,1                ;input validation
    jl invalidspeed
    cmp ax, 3
    jg invalidspeed
    mul esi
    mov speed, eax          ;assign speed variable in mililiseconds
    ret

    invalidspeed:           ;jump here if user entered an invalid number
    mov dl,105
    mov dh,1
    call Gotoxy
    mov edx, OFFSET invalidInput        ;print error message
```

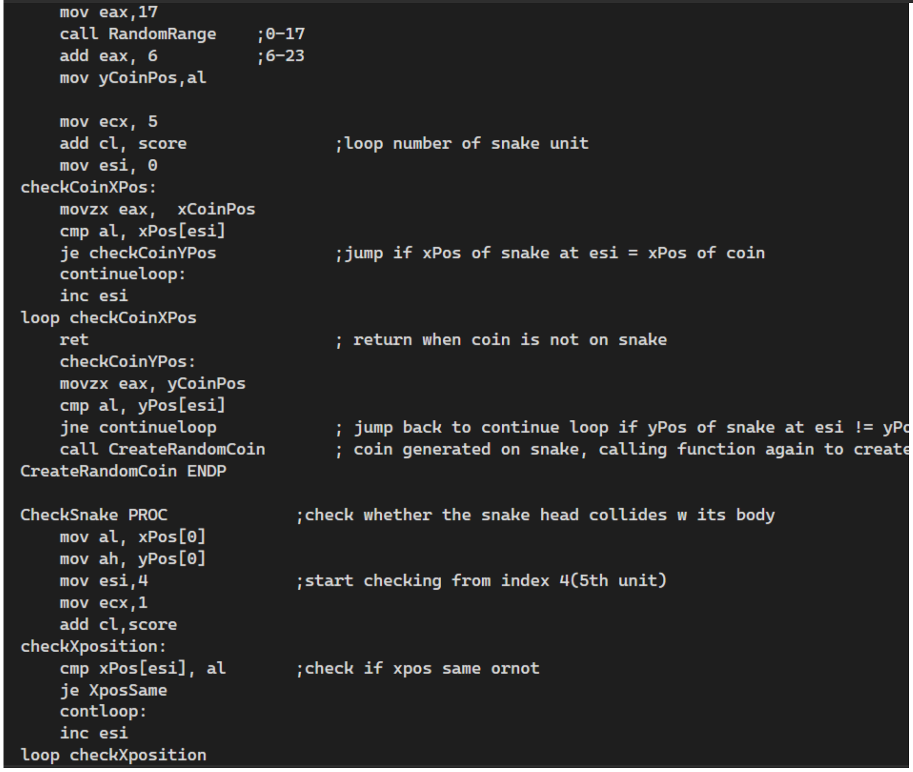


```
    mov edx, OFFSET invalidInput        ;print error message
```

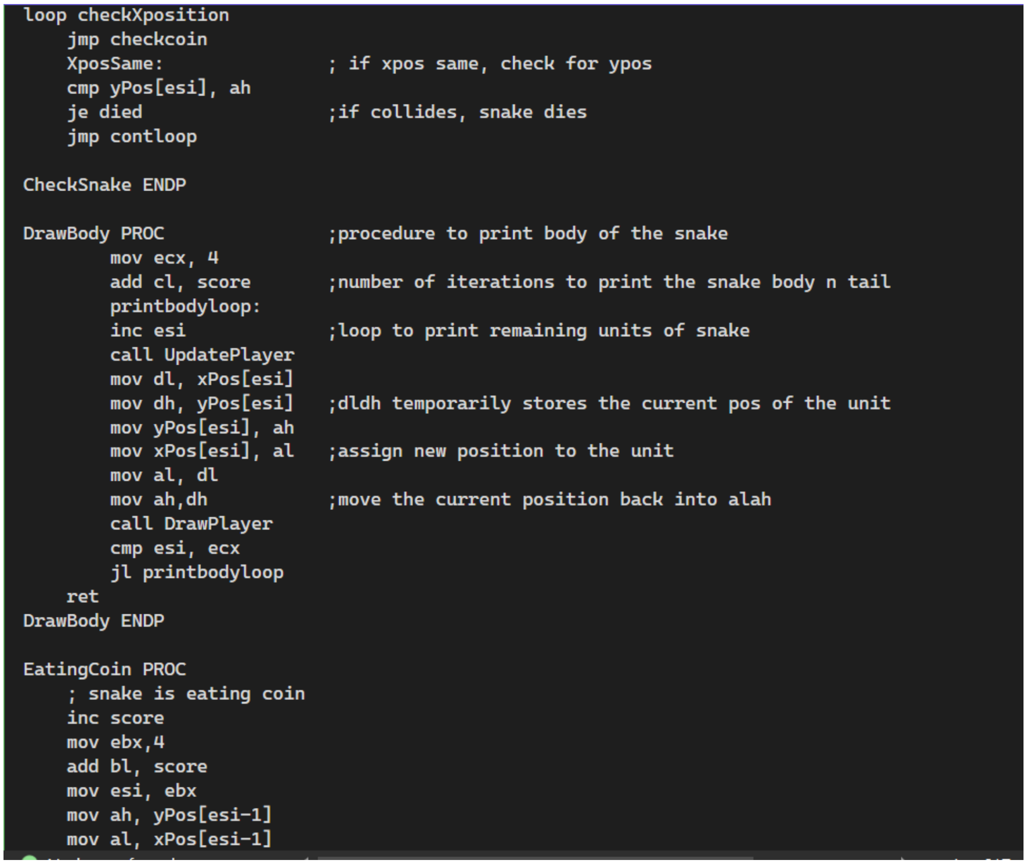
```
    call WriteString
    mov ax, 1500
    call delay
    mov dl,105
    mov dh,1
    call Gotoxy
    mov edx, OFFSET blank               ;erase error message after 1.5 secs of delay
    call writeString
    call ChooseSpeed                    ;call procedure for user to choose again
    ret
ChooseSpeed ENDP

DrawPlayer PROC         ; draw player at (xPos,yPos)
    mov dl,xPos[esi]
    mov dh,yPos[esi]
    call Gotoxy
    mov dl, al          ;temporarily save al in dl
    mov al, snake[esi]
    call WriteChar
    mov al, dl
    ret
DrawPlayer ENDP

UpdatePlayer PROC       ; erase player at (xPos,yPos)
    mov dl, xPos[esi]
    mov dh,yPos[esi]
    call Gotoxy
    mov dl, al          ;temporarily save al in dl
    mov al, " "
    call WriteChar
    mov al, dl
    ret
UpdatePlayer ENDP
```



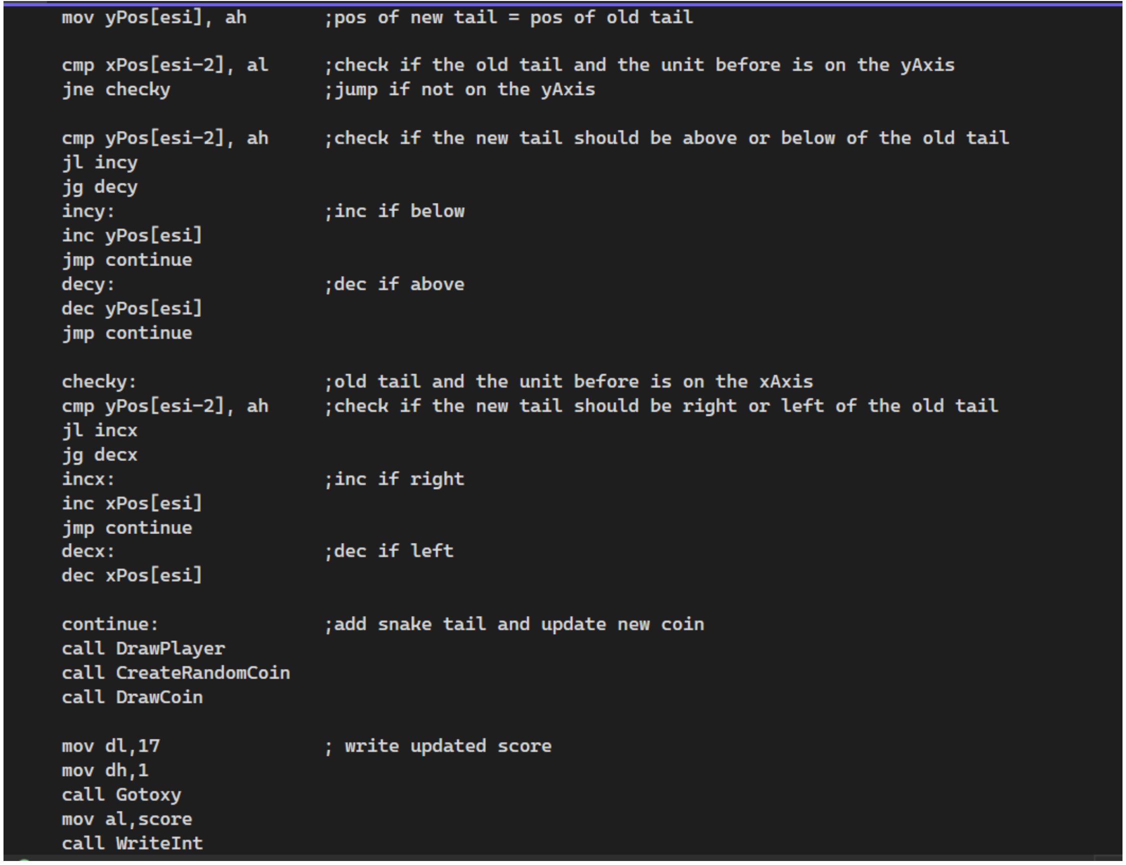
```
    mov eax,yellow (yellow * 16)
    call SetTextColor               ;set color to yellow for coin
    mov dl,xCoinPos
    mov dh,yCoinPos
    call Gotoxy
    mov al,"X"
    call WriteChar
    mov eax,white (black * 16)      ;reset color to black and white
    call SetTextColor
    ret
DrawCoin ENDP

Body PROC               ;procedure to print body of the snake
        mov ecx, 4
        add cl, score       ;number of iterations to print the snake body n tail
        printbodyloop:
        inc esi             ;loop to print remaining units of snake
        call UpdatePlayer
        mov dl, xPos[esi]
        mov dh, yPos[esi]   ;dldh temporarily stores the current pos of the unit
        mov yPos[esi], ah
        mov xPos[esi], al   ;assign new position to the unit
        mov al, dl
        mov ah,dh           ;move the current position back into alah
        call DrawPlayer
        cmp esi, ecx
        jl printbodyloop
    ret
Body ENDP

CreateRandomCoin PROC               ;procedure to create a random coin
    mov eax,49
    call RandomRange    ;0-49
    add eax, 35         ;35-84
    mov xCoinPos,al
```



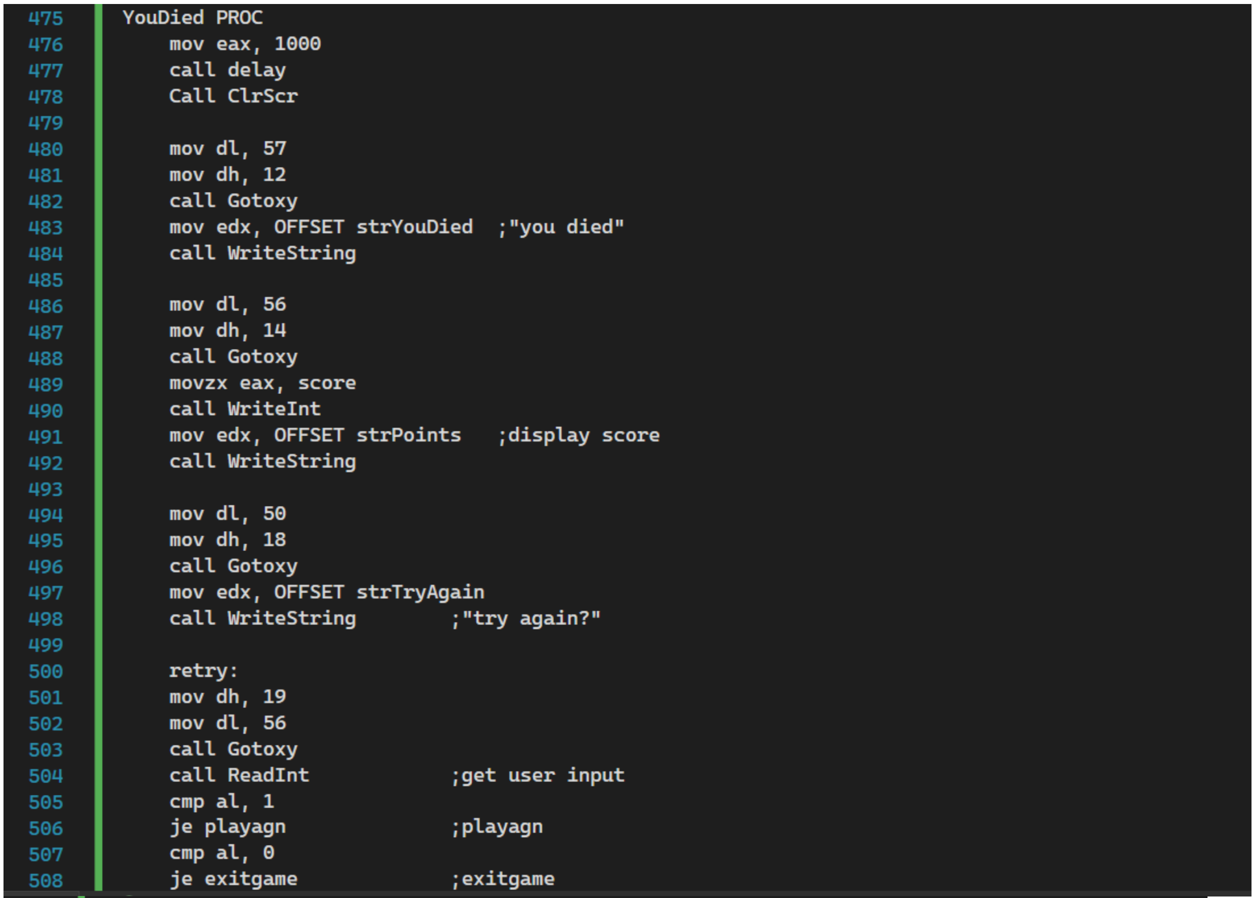
```
    mov eax,17
    call RandomRange    ;0-17
    add eax, 6          ;6-23
    mov yCoinPos,al

    mov ecx, 5
    add cl, score               ;loop number of snake unit
    mov esi, 0
checkCoinXPos:
    movzx eax,  xCoinPos
    cmp al, xPos[esi]
    je checkCoinYPos            ;jump if xPos of snake at esi = xPos of coin
    continueloop:
    inc esi
loop checkCoinXPos
    ret                         ; return when coin is not on snake
    checkCoinYPos:
    movzx eax, yCoinPos
    cmp al, yPos[esi]
    jne continueloop            ; jump back to continue loop if yPos of snake at esi != yPo
    call CreateRandomCoin       ; coin generated on snake, calling function again to create
CreateRandomCoin ENDP

CheckSnake PROC             ;check whether the snake head collides w its body
    mov al, xPos[0]
    mov ah, yPos[0]
    mov esi,4               ;start checking from index 4(5th unit)
    mov ecx,1
    add cl,score
checkXposition:
    cmp xPos[esi], al       ;check if xpos same ornot
    je XposSame
    contloop:
    inc esi
loop checkXposition
```



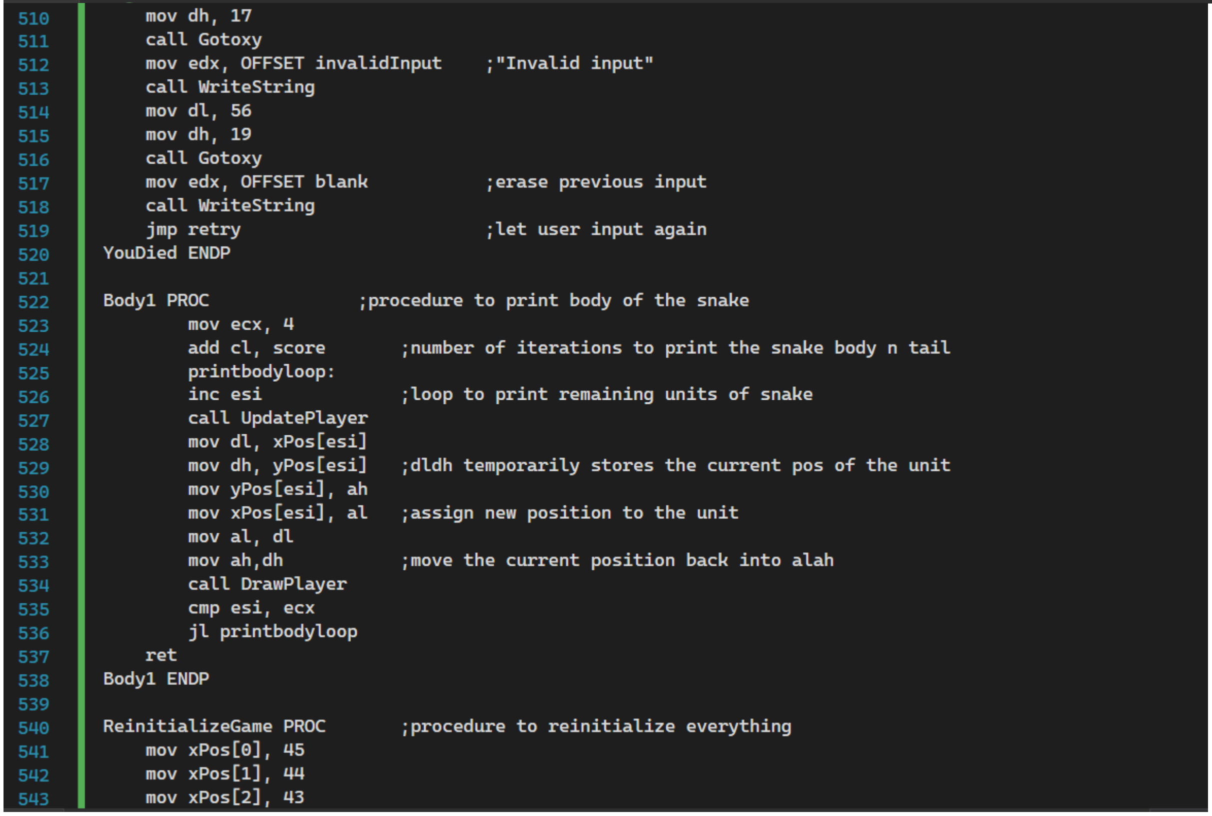
```
loop checkXposition
    jmp checkcoin
    XposSame:               ; if xpos same, check for ypos
    cmp yPos[esi], ah
    je died                 ;if collides, snake dies
    jmp contloop

CheckSnake ENDP

DrawBody PROC               ;procedure to print body of the snake
        mov ecx, 4
        add cl, score       ;number of iterations to print the snake body n tail
        printbodyloop:
        inc esi             ;loop to print remaining units of snake
        call UpdatePlayer
        mov dl, xPos[esi]
        mov dh, yPos[esi]   ;dldh temporarily stores the current pos of the unit
        mov yPos[esi], ah
        mov xPos[esi], al   ;assign new position to the unit
        mov al, dl
        mov ah,dh           ;move the current position back into alah
        call DrawPlayer
        cmp esi, ecx
        jl printbodyloop
    ret
DrawBody ENDP

EatingCoin PROC
    ; snake is eating coin
    inc score
    mov ebx,4
    add bl, score
    mov esi, ebx
    mov ah, yPos[esi-1]
    mov al, xPos[esi-1]
```



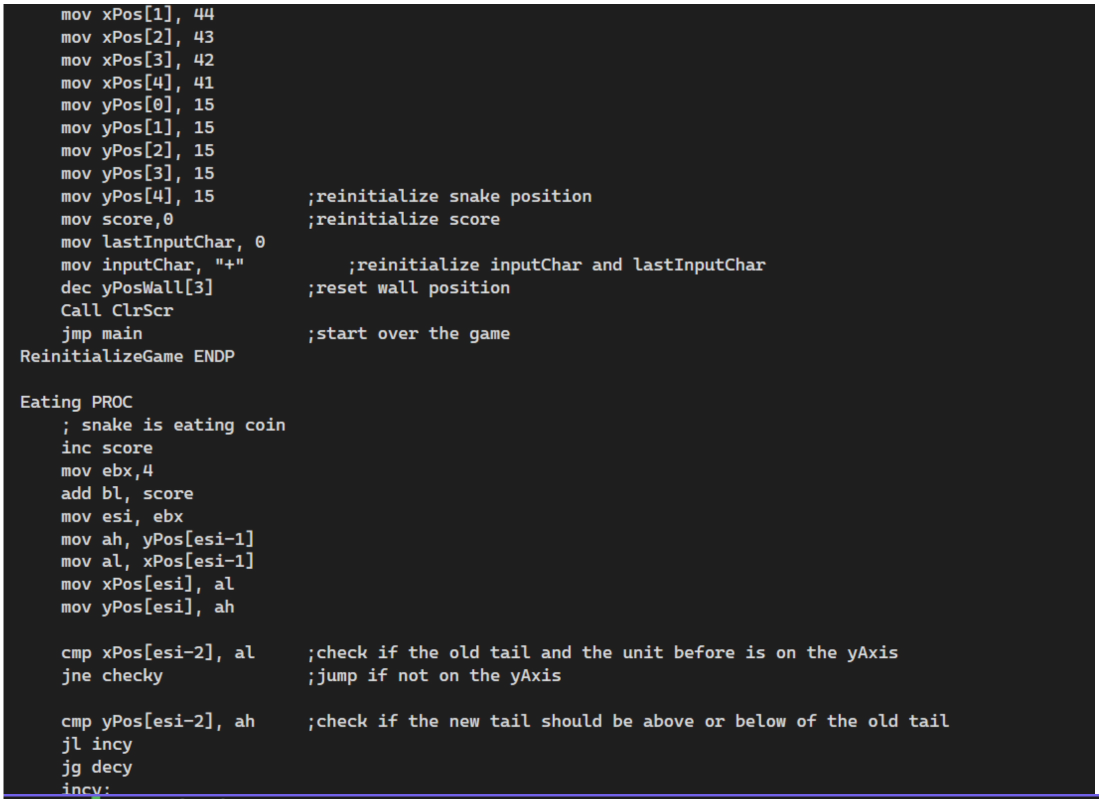
```
    mov yPos[esi], ah       ;pos of new tail = pos of old tail

    cmp xPos[esi-2], al     ;check if the old tail and the unit before is on the yAxis
    jne checky              ;jump if not on the yAxis

    cmp yPos[esi-2], ah     ;check if the new tail should be above or below of the old tail
    jl incy
    jg decy
    incy:                   ;inc if below
    inc yPos[esi]
    jmp continue
    decy:                   ;dec if above
    dec yPos[esi]
    jmp continue

    checky:                 ;old tail and the unit before is on the xAxis
    cmp yPos[esi-2], ah     ;check if the new tail should be right or left of the old tail
    jl incx
    jg decx
    incx:                   ;inc if right
    inc xPos[esi]
    jmp continue
    decx:                   ;dec if left
    dec xPos[esi]

    continue:               ;add snake tail and update new coin
    call DrawPlayer
    call CreateRandomCoin
    call DrawCoin

    mov dl,17               ; write updated score
    mov dh,1
    call Gotoxy
    mov al,score
    call WriteInt
```



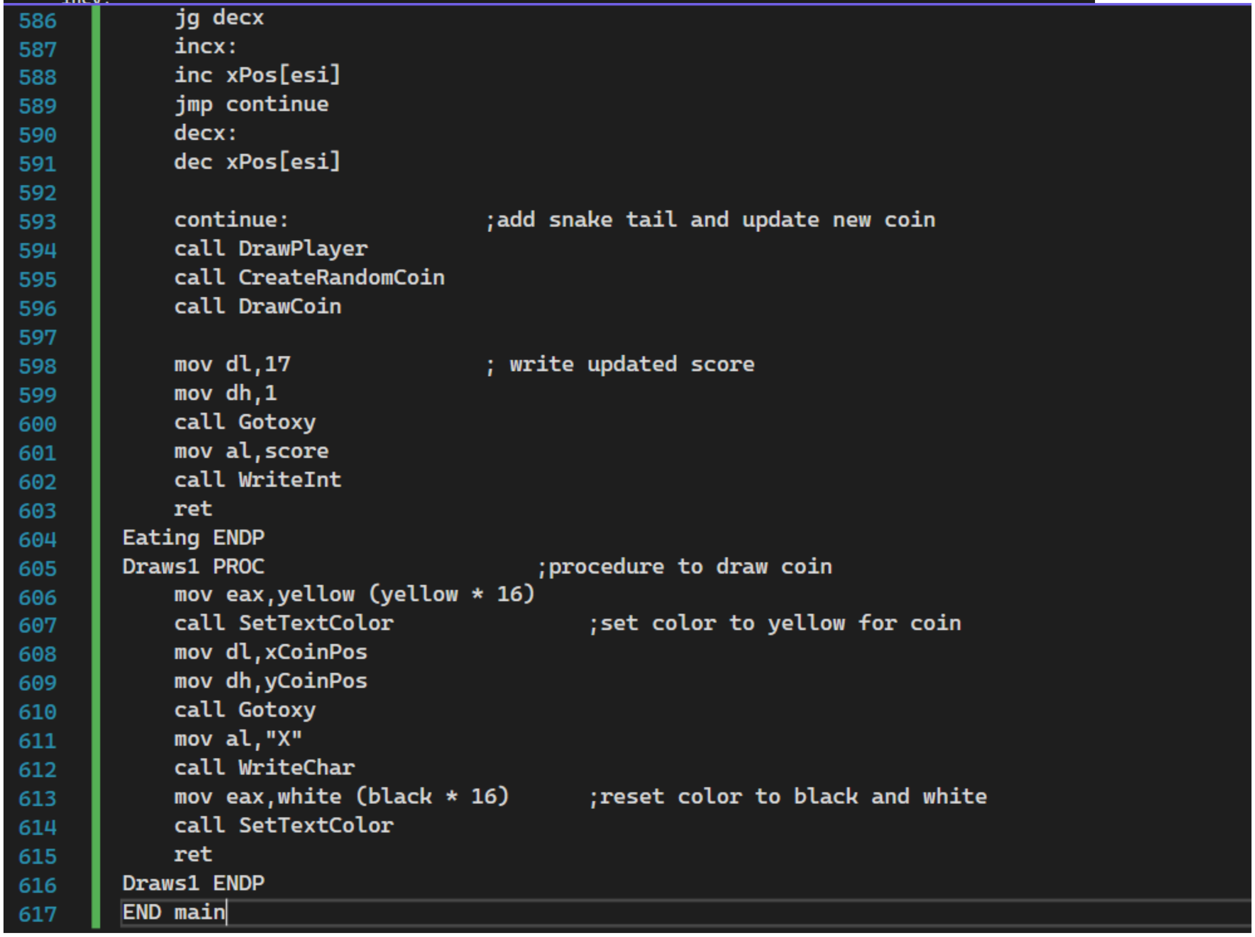
```
YouDied PROC
    mov eax, 1000
    call delay
    Call ClrScr

    mov dl, 57
    mov dh, 12
    call Gotoxy
    mov edx, OFFSET strYouDied  ;"you died"
    call WriteString

    mov dl, 56
    mov dh, 14
    call Gotoxy
    movzx eax, score
    call WriteInt
    mov edx, OFFSET strPoints   ;display score
    call WriteString

    mov dl, 50
    mov dh, 18
    call Gotoxy
    mov edx, OFFSET strTryAgain
    call WriteString        ;"try again?"

    retry:
    mov dh, 19
    mov dl, 56
    call Gotoxy
    call ReadInt            ;get user input
    cmp al, 1
    je playagn              ;playagn
    cmp al, 0
    je exitgame             ;exitgame
```



```
    mov dh, 17
    call Gotoxy
    mov edx, OFFSET invalidInput    ;"Invalid input"
    call WriteString
    mov dl, 56
    mov dh, 19
    call Gotoxy
    mov edx, OFFSET blank           ;erase previous input
    call WriteString
    jmp retry                       ;let user input again
YouDied ENDP

Body1 PROC              ;procedure to print body of the snake
        mov ecx, 4
        add cl, score       ;number of iterations to print the snake body n tail
        printbodyloop:
        inc esi             ;loop to print remaining units of snake
        call UpdatePlayer
        mov dl, xPos[esi]
        mov dh, yPos[esi]   ;dldh temporarily stores the current pos of the unit
        mov yPos[esi], ah
        mov xPos[esi], al   ;assign new position to the unit
        mov al, dl
        mov ah,dh           ;move the current position back into alah
        call DrawPlayer
        cmp esi, ecx
        jl printbodyloop
    ret
Body1 ENDP

ReinitializeGame PROC       ;procedure to reinitialize everything
    mov xPos[0], 45
    mov xPos[1], 44
    mov xPos[2], 43
```

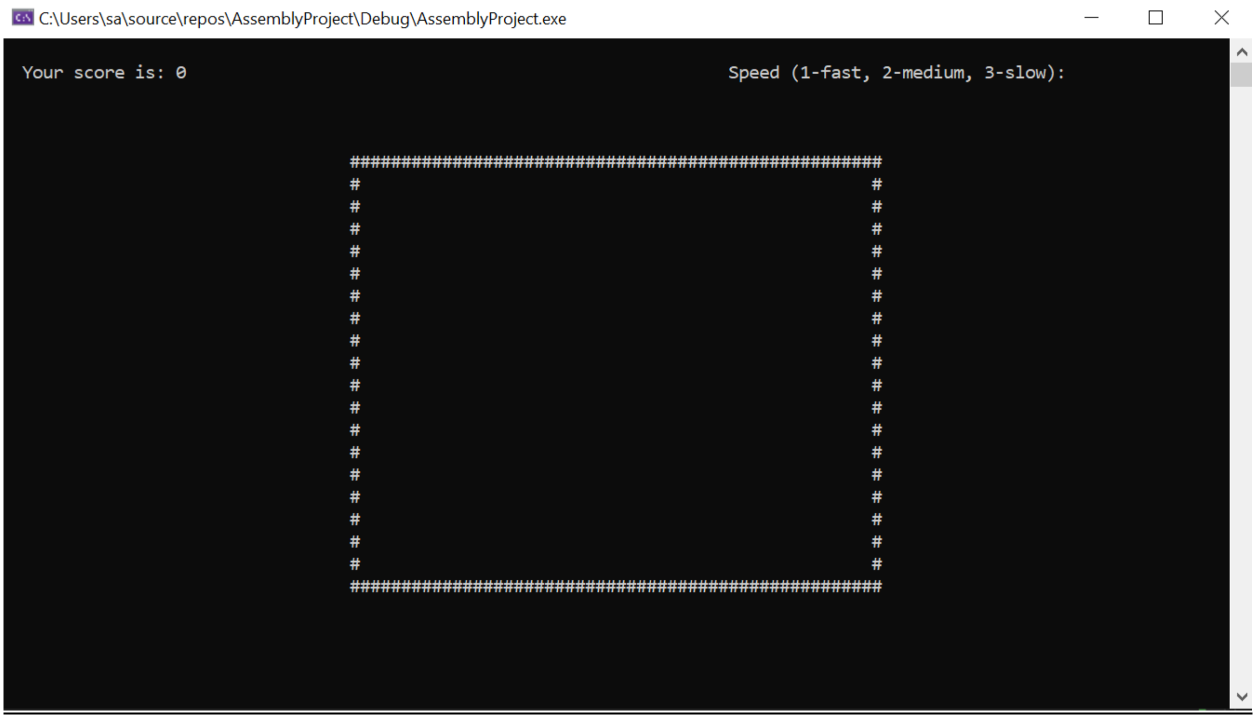


```
    mov xPos[1], 44
    mov xPos[2], 43
    mov xPos[3], 42
    mov xPos[4], 41
    mov yPos[0], 15
    mov yPos[1], 15
    mov yPos[2], 15
    mov yPos[3], 15
    mov yPos[4], 15         ;reinitialize snake position
    mov score,0             ;reinitialize score
    mov lastInputChar, 0
    mov inputChar, "+"          ;reinitialize inputChar and lastInputChar
    dec yPosWall[3]         ;reset wall position
    Call ClrScr
    jmp main                ;start over the game
ReinitializeGame ENDP

Eating PROC
    ; snake is eating coin
    inc score
    mov ebx,4
    add bl, score
    mov esi, ebx
    mov ah, yPos[esi-1]
    mov al, xPos[esi-1]
    mov xPos[esi], al
    mov yPos[esi], ah

    cmp xPos[esi-2], al     ;check if the old tail and the unit before is on the yAxis
    jne checky              ;jump if not on the yAxis

    cmp yPos[esi-2], ah     ;check if the new tail should be above or below of the old tail
    jl incy
    jg decy
    incy:
```



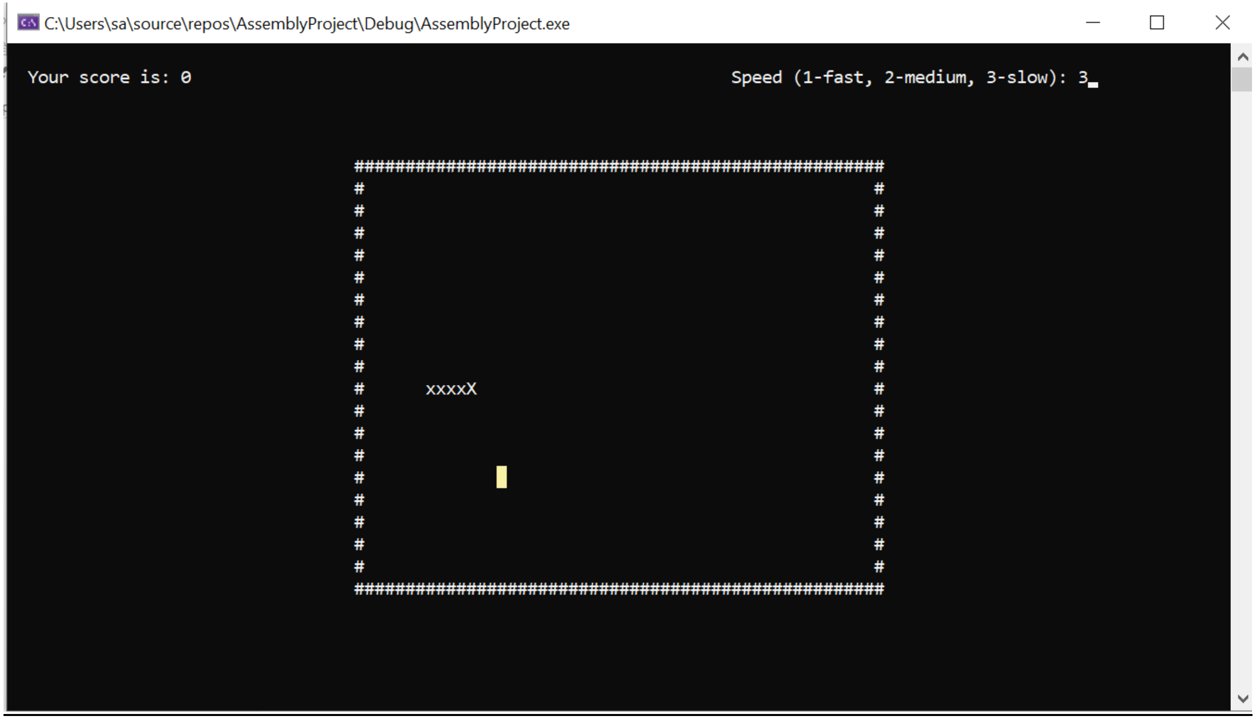
```
    jg decx
    incx:
    inc xPos[esi]
    jmp continue
    decx:
    dec xPos[esi]

    continue:               ;add snake tail and update new coin
    call DrawPlayer
    call CreateRandomCoin
    call DrawCoin

    mov dl,17               ; write updated score
    mov dh,1
    call Gotoxy
    mov al,score
    call WriteInt
    ret
Eating ENDP
Draws1 PROC                     ;procedure to draw coin
    mov eax,yellow (yellow * 16)
    call SetTextColor               ;set color to yellow for coin
    mov dl,xCoinPos
    mov dh,yCoinPos
    call Gotoxy
    mov al,"X"
    call WriteChar
    mov eax,white (black * 16)      ;reset color to black and white
    call SetTextColor
    ret
Draws1 ENDP
END main
```

Output



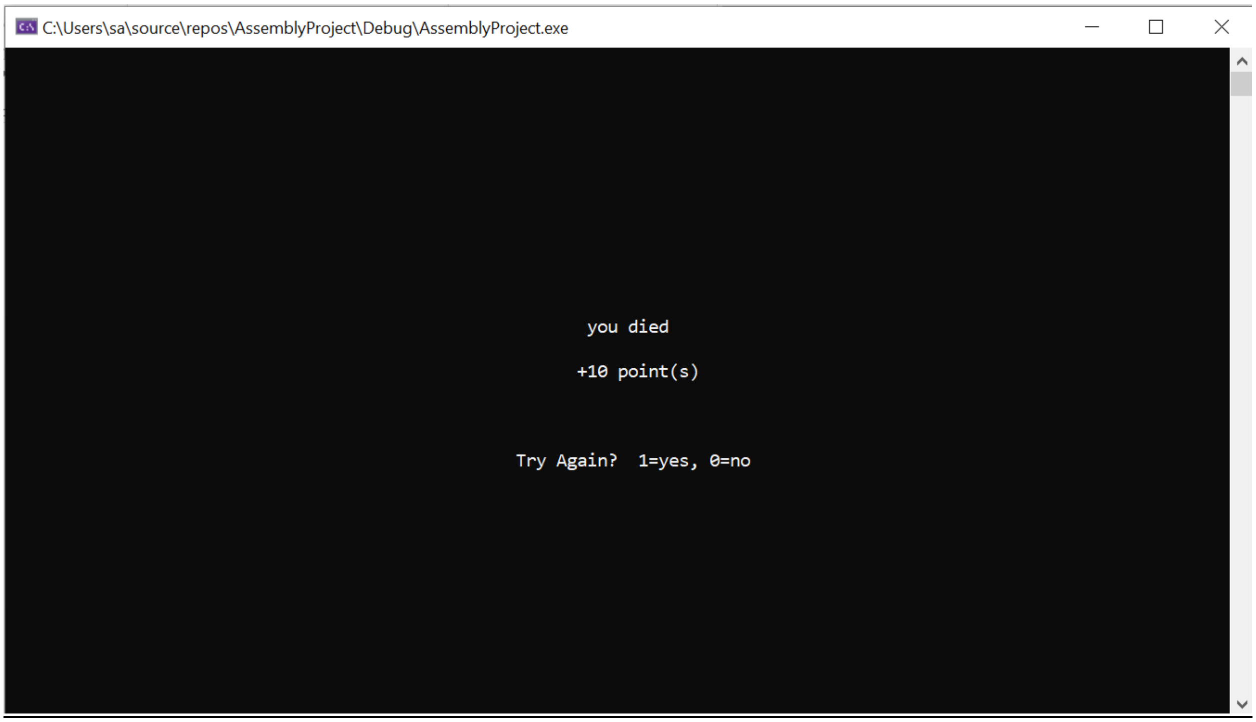
C:\Users\sa\source\repos\AssemblyProject\Debug\AssemblyProject.exe

```
Your score is: 0                                                     Speed (1-fast, 2-medium, 3-slow):


                                ####################################################
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                ####################################################
```



C:\Users\sa\source\repos\AssemblyProject\Debug\AssemblyProject.exe

```
Your score is: 0                                                     Speed (1-fast, 2-medium, 3-slow): 3


                                ####################################################
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #      xxxxX                                       #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #             █                                    #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                #                                                  #
                                ####################################################
```



C:\Users\sa\source\repos\AssemblyProject\Debug\AssemblyProject.exe

```
you died

+10 point(s)


Try Again?  1=yes, 0=no
```

9. Conclusion

This project was about the snake game on assembly language. Though the graphics of the game is not that great but still the logic and algorithm that we have used works perfectly sine as any other snake game on any other platform. Procedures are used for each task be it initializing the board for the game, taking the speed on which the game will be proceeded, initializing the snake for the game, randomly assigning the coins in the board for the snake, the snake movement, the score count, the died procedure that executes when the snake hit the wall or its body, the final score display and the reinitializing function. This project is completed on the Irvine library.