Flappy Bird Game in Assembly Language on x86 Architecture

Any instances of copying and/or plagiarism within the project will be addressed with strict consequences.

Project Overview

The goal of this project is to design and implement the classic mobile game "Flappy Bird" using computer architecture principles and assembly language programming. This project will deepen your understanding of low-level programming and provide practical experience in developing a simple yet engaging game.

Project Description

In this project, you will create a version of Flappy Bird using assembly language. The game should feature a bird that moves vertically on the screen, controlled by user input, and navigates through a series of pipes. The primary focus is on implementing game mechanics, collision detection, and rendering in assembly language.

Key Components

1. **Game Initialization:**
   Set up data structures for game variables and parameters.

   Initialize the game window and necessary resources.

2. **User Input:**
   Capture user input to control the bird's movement (e.g., tapping or key presses).

   Only the space key is used as input.

3. **Game Logic:**
   Implement logic for bird movement, gravity, and collision detection with pipes.

   Define the scoring mechanism based on the distance traveled.

4. **Graphics and Rendering with Sound:**
   Develop routines for rendering game elements, including the bird, pipes, and background.

   Implement efficient graphics handling for smooth gameplay with suitable sounds. See the Appendix for a sample graphic representation.

5. **Game Over and Restart:**
   Implement a game over screen with a score display.

   Allow users to restart the game after a collision.

Requirements

1. Implement the game in assembly language, targeting a specific x86 architecture.
2. Utilize low-level graphics routines for efficient rendering.
3. Demonstrate effective use of registers and memory management.
4. Create a user-friendly interface for input and feedback.
5. Optimize the code for performance, considering the limitations of the chosen architecture.

Deliverables

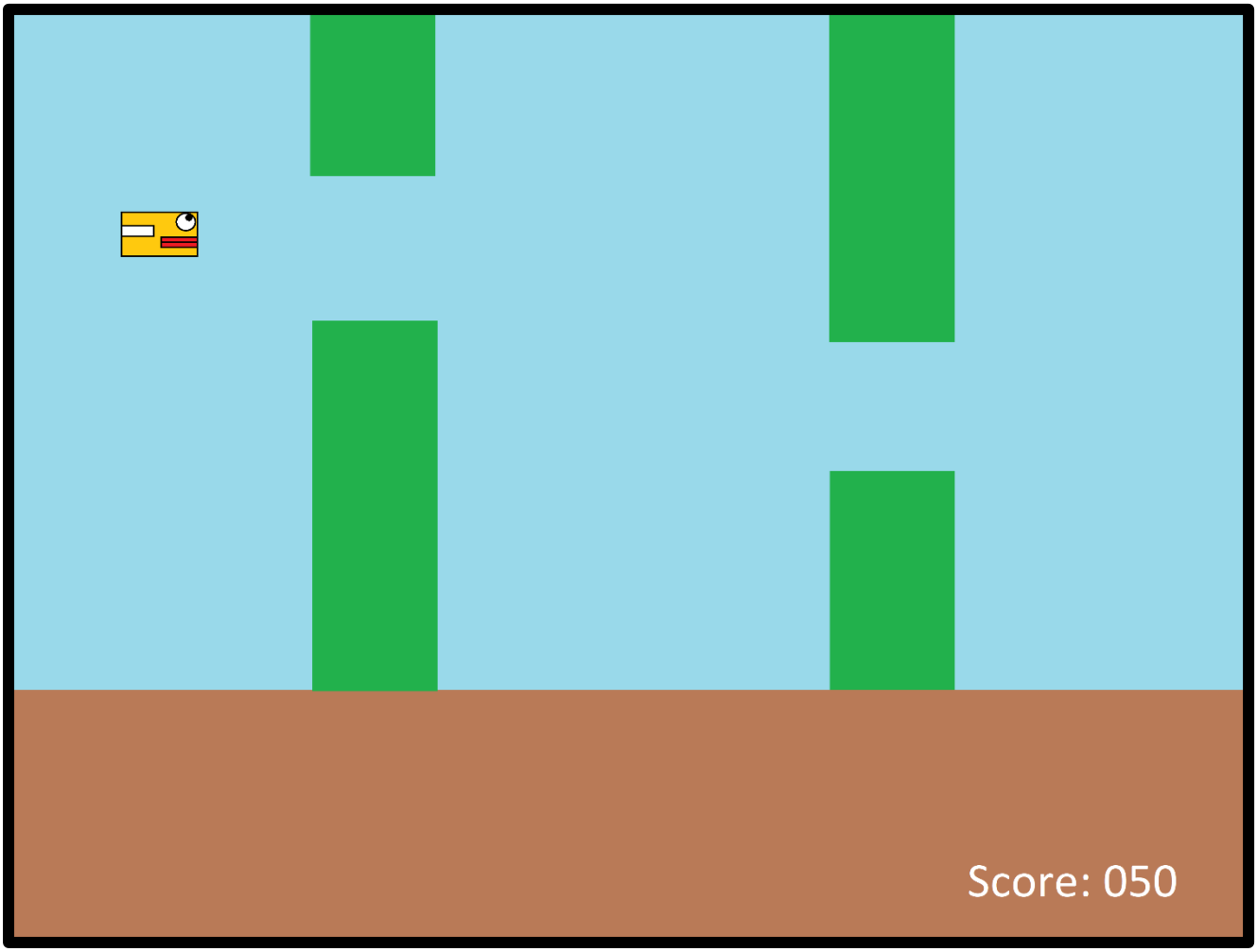
- Complete assembly language source code for the Flappy Bird game.
- A demonstration and viva of the working game, showcasing the implemented features.

Evaluation Criteria

- Correctness and functionality of the Flappy Bird game.
- Efficient use of assembly language features and optimizations.
- Clarity and organization of the source code.
- Demonstration of the project during evaluation.

Appendix

Sample screen:




```
;See which scan codes are take as input from key board
[org 0x0100]
jmp start

clrscr:
        push es
```

```
        push ax
        push cx
        push di
        mov ax, 0xb800
        mov es, ax ; point es to video base
        xor di, di ; point di to top left column
        mov ax, 0x0720 ; space char in normal attribute
        mov cx, 2000 ; number of screen locations
        cld ; auto increment mode
        rep stosw ; clear the whole screen
        pop di
        pop cx
        pop ax
        pop es
        ret

printnum: push bp
 mov bp, sp
 push es
 push ax
 push bx
 push cx
 push dx
 push di
 mov ax, 0xb800
 mov es, ax ; point es to video base
 mov ax, [bp+4] ; load number in ax
```

```
 mov bx, 10 ; use base 10 for division
 mov cx, 0 ; initialize count of digits
nextdigit: mov dx, 0 ; zero upper half of dividend
 div bx ; divide by 10
 add dl, 0x30 ; convert digit into ascii value
 push dx ; save ascii value on stack
 inc cx ; increment count of values
 cmp ax, 0 ; is the quotient zero
 jnz nextdigit ; if no divide it again
 mov di, 0 ; point di to top left column
 nextpos: pop dx ; remove a digit from the stack
 mov dh, 0x07 ; use normal attribute
 mov [es:di], dx ; print char on screen
 add di, 2 ; move to next screen location
 loop nextpos ; repeat for all digits on stack
 pop di
 pop dx
 pop cx
 pop bx
 pop ax
 pop es
 pop bp
 ret 2

; keyboard interrupt service routine
kbisr:
 push ax
 push es
 in al, 0x60 ; read a char from keyboard port
 mov ah, 00h
```

```
call clrscr
push ax ; place number on stack
call printnum
;mov word [es:0], ax ; yes, print L at top left
mov al, 0x20
out 0x20, al ; send EOI to PIC
pop es
pop ax
iret

start:
xor ax, ax
mov es, ax ; point es to IVT base
cli ; disable interrupts
mov word [es:9*4], kbisr ; store offset at n*4
mov [es:9*4+2], cs ; store segment at n*4+2
sti ; enable interrupts
l1: jmp l1 ; infinite loop
```