PAMANTASAN NG LUNGSOD NG MAYNILA

(University of the City of Manila)

In Partial Fulfillment of the Requirements for the

CSC 0222-2 | Architecture and Organization,

HariBird's Tiny Wing Escape

An Assembly Language Project Proposal

College of Information Systems and Technology Management

Pamantasan ng Lungsod ng Maynila

Submitted to:

Engr. Elsa Pascual

Submitted by:

BSCS 2-1 | Group 3

Group Leader:

Goyena, Shawn Kieffer E.

Group Members:

Cordova, Venelyn Mae C.

De Mesa, Rita Angeli M.

Musni, Lorelie Joy A.

Navarro, Sofia Alexi P.

Santiago, Christian Andrei V.

Segovia, Aliyah Loise C.

Verdida, Maverick Isaiah A.

Date:

May 17, 2025

I. Members and their tasks

Members	**Tasks**
Cordova, Venelyn Mae C.	• Booth Design • Fullstack Developer
De Mesa, Rita Angeli M.	• UI Design • Booth Design
Goyena, Shawn Kieffer E.	• Fullstack Developer • Booth Design
Musni, Lorelie Joy A.	• UI Design • Fullstack Developer
Navarro, Sofia Alexi P.	• UI Design • Booth Design
Santiago, Christian Andrei V.	• Fullstack Developer • Booth Design
Segovia, Aliyah Loise C.	• Fullstack Developer • Documentation
Verdida, Maverick Isaiah A.	• Fullstack Developer • Documentation

II. Introduction

Flappy Bird is a mobile game created by Vietnamese programmer Dong Nguyen in 2013. Despite its simplicity, the game quickly gained worldwide popularity because of its highly addictive mechanics and challenging gameplay. Players guide a small bird that flaps its wings every time the screen is tapped, navigating through a series of green pipes without colliding with them. The game's minimalist design, combined with its difficulty and reliance on precise timing, aided its widespread popularity. It demonstrated that even a game with a simple concept and simple mechanics could achieve enormous commercial success and cultural impact. Furthermore, Flappy Bird had a significant impact on the indie game development community by demonstrating the viability of small-scale, independently developed games in the mobile marketplace.

Inspired by Flappy Bird's success and mechanics, a separate development team created an arcade adaptation called Haribird. Unlike the original, which was written in high-level programming languages, Haribird was written in assembly language, a low-level language that communicates directly with computer hardware through symbolic machine instructions. The developers chose assembly language to demonstrate its technical quality while also demonstrating that classic gameplay experiences could be recreated and optimized using such a low-level approach. Their goals were threefold: to keep the original game's addictive and fast-paced nature, to emphasize the complexity and precision required in low-level programming, and to provide a nostalgic arcade experience through optimized performance and minimalist graphics. Haribird thus serves as a tribute to both retro gaming culture and the technological possibilities provided by assembly language.

A new team of developers is currently working to improve Haribird, building on its existing framework and transforming it into a more refined version called HariBird's Tiny Wing Escape. This improved version will keep the core gameplay mechanics that defined the original while significantly improving functionality and performance. The current development team intends to enhance the game's features by adding more responsive controls, improved collision detection, additional visual and audio elements, and optimized execution to ensure a smoother and more immersive gaming experience. These improvements are intended not only to increase player engagement but also to show how low-level programming can be adapted to meet modern game design standards. HariBird's Tiny Wing Escape aims to combine the nostalgic appeal of classic arcade games with modern standards of functionality and interactivity. This project therefore continues and advances the original developers' vision by emphasizing both technical proficiency and creative innovation in low-level game development.

III. Enhancements

HariBird's Tiny Wing Escape includes several new features that improve gameplay, increase user control, and enhance overall experience:

1. Pause / Restart / Quit Options
   - Players can now pause the game mid-session to take a break.
   - A restart option lets players quickly begin a new game without returning to the main menu.
   - The quit function allows users to exit to the main menu at any time during gameplay.
2. Power-Ups
   - Extra Life: When the player crashes, they are given a second chance to continue from where they left off. This can only be used once per game session.
   - Invincibility (Press "I"): Grants temporary immunity to obstacles. During invincibility, the bird can fly through pipes without dying, making it easier to survive tough sections.
3. Audio Feedback
   - Background Music: A looping soundtrack plays during gameplay to set the mood and add excitement.
   - Flap Sound: A sound effect plays every time the bird flaps its wings.
   - Collision Sound: When the bird crashes into a pipe, a distinct sound plays to signal game over or extra life activation.
   - Score Sound: A satisfying audio cue plays every time the player earns a point by passing through pipes.

IV. Definition of Terms

Terms and Keywords	Definition
Addictive Mechanics	Game design elements that strongly encourage repeated play due to their engaging or challenging nature.
Arcade Adaptation	A version of a game designed to mimic or run on arcade-style machines, often with fast-paced and skill-based gameplay.
Collision Detection	A programming technique used to determine when two objects in a game interact or come into contact.
Core Gameplay Mechanics	The essential actions and rules that define how a game operates and how players interact with it.

High-Level Programming Language	A programming language like Python, Java, or C++ that is closer to human language and abstracts away hardware details.
Assembly Language	A low-level programming language that provides direct control over a computer's hardware using symbolic representations of machine code.
Low-Level Programming	Programming that interacts closely with hardware, typically using assembly or machine code. It allows for high performance but is more complex and error prone.
Optimized Execution	Code that is written or refined to run more efficiently, using less memory or processing power.
Framework	A base structure used by developers to build and maintain applications, often providing pre-written code and libraries.
Technical Proficiency	A high level of skill or expertise in programming or system design, often involving detailed knowledge of computer systems.
User Engagement	The degree to which players are involved and interested in a game, often influenced by design, mechanics, and feedback systems.
Creative Innovation	The act of introducing new ideas or methods in design and development to create unique or enhanced user experiences.
Indie Game Development	The process of creating games by individuals or small teams without financial support from large publishers.
NASM	An open-source assembler for the x86 architecture that translates assembly language code into machine code or binary executable files. It is widely used for programming in low-level languages on Intel and AMD processors.

V. How to Play

Getting Started

- Launch HariBird’s Tiny Wing Escape using DOSBox. Make sure DOSBox is installed and properly set up on your computer.

Main Menu Navigation

After the game starts, you’ll see the main menu. Here, you can:

- Choose a difficulty level (Easy, Medium, Hard)
- How to play
- Quit the game

Difficulty Levels

- Easy – Slower game speed, ideal for beginners.
- Medium – Moderate game speed for a balanced experience.
- Hard – Fast game speed for a more intense challenge.

Gameplay Controls

- Press the Spacebar flap and lift the bird upward.
- Tap repeatedly to keep flying and avoid crashing into pipes. Timing is key.

Power-Ups

- Extra Life: Grants one respawn if you crash. It brings you back to where you left off, but can only be used once per game session.
- Invincibility (Press "I"): Temporarily makes the bird immune to collisions. Use it to fly through pipes safely without dying.

In-Game Options

- Pause/Resume – Pause the game anytime and continue later.
- Restart – Start the current game over from the beginning.
- Quit – Exit the game and return to the main menu.

Scoring

- You earn points for pipes you pass through.
- Try to survive as long as possible and beat your high score.

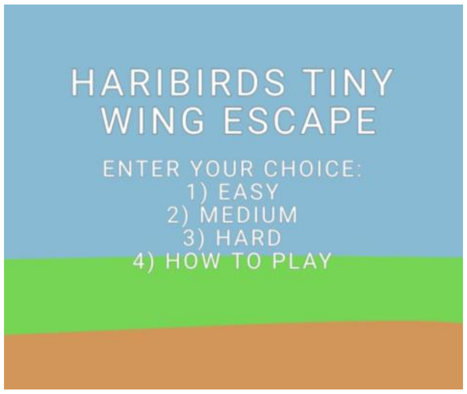
Tips

- Keep a steady rhythm when tapping to maintain control.
- Save your power-ups for tricky moments, they can make the difference.

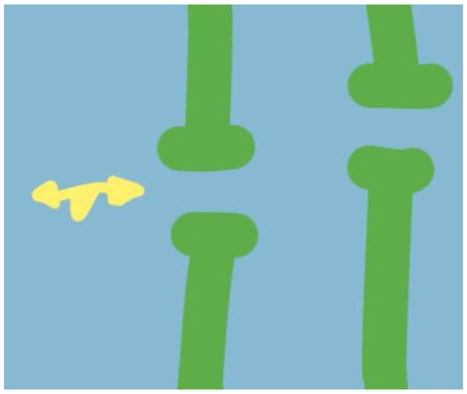
HariBird's Tiny Wing Escape blends the charm of old-school games with new twists, making each session fun and fast-paced.

VI. Storyboard / UI Design

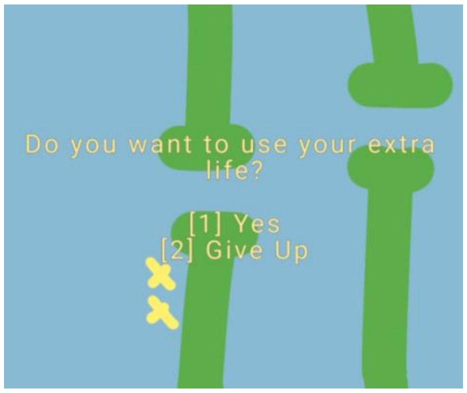
Storyboard



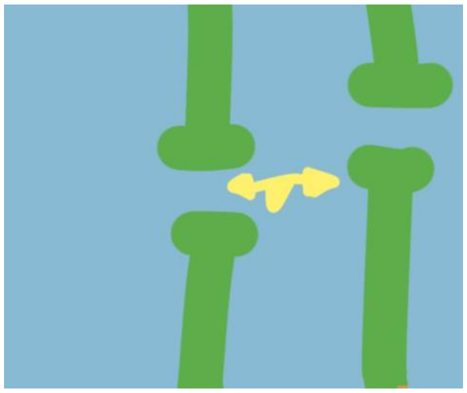
Game Menu



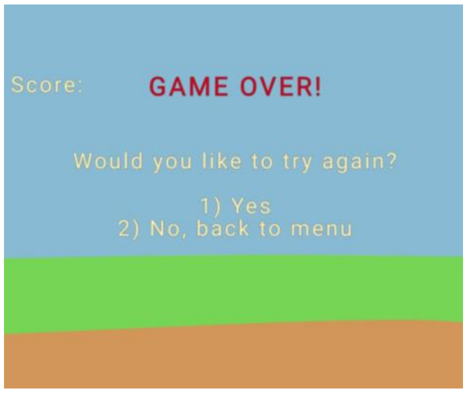
Game Screen



Extra Life



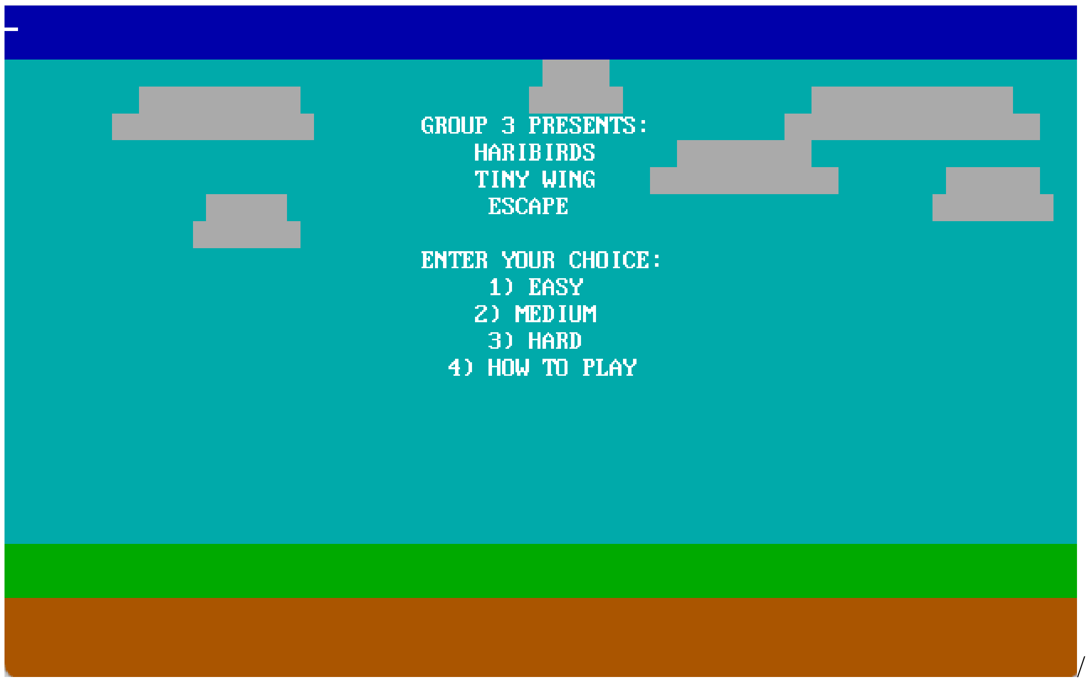
If yes, respawn



If no, End Screen

UI Design

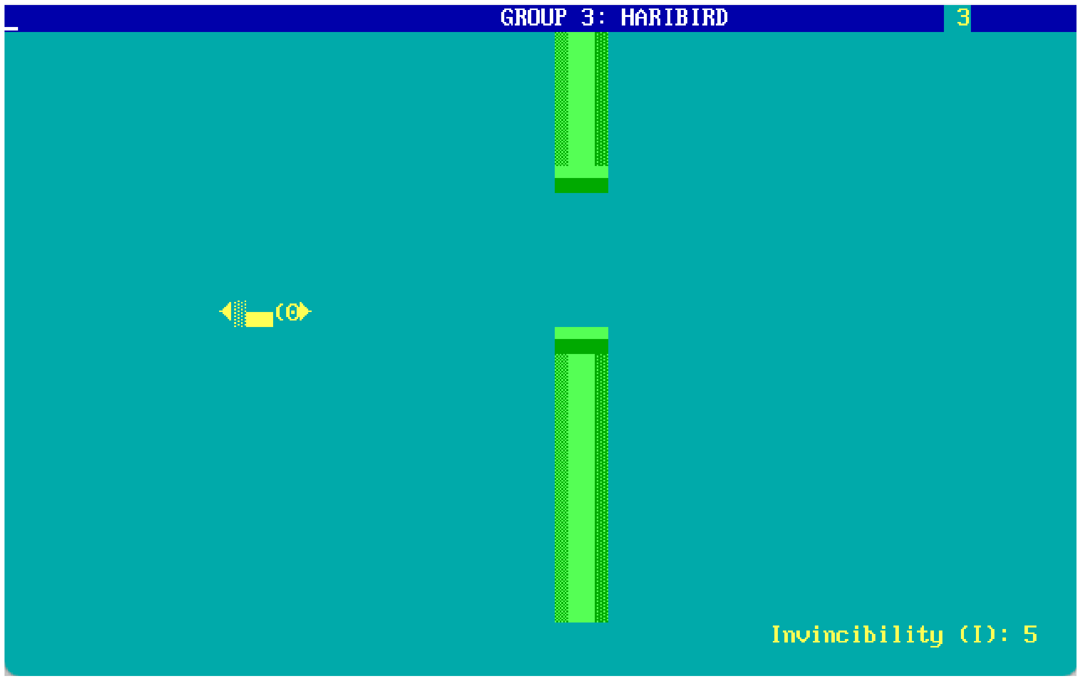
Starting Screen



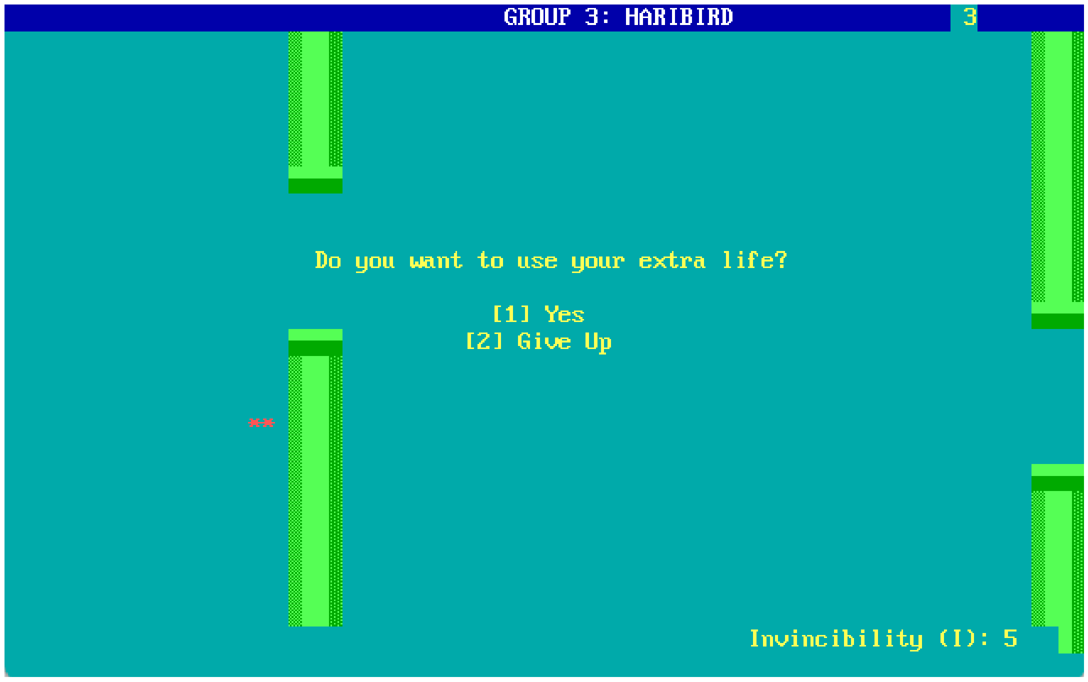
GROUP 3 PRESENTS:
HARIBIRDS
TINY WING
ESCAPE

ENTER YOUR CHOICE:
1) EASY
2) MEDIUM
3) HARD
4) HOW TO PLAY

Start of the Game



Extra Life prompt


GROUP 3: HARIBIRD

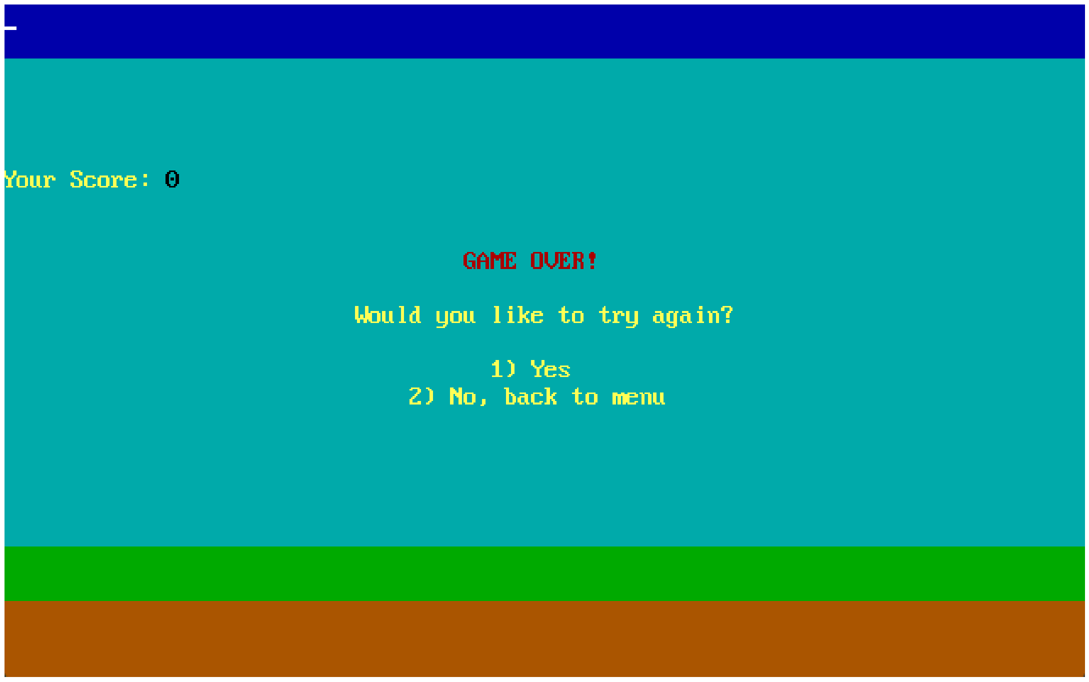
3

Do you want to use your extra life?

[1] Yes
[2] Give Up

Invincibility (I): 5

‘Game Over’ Screen

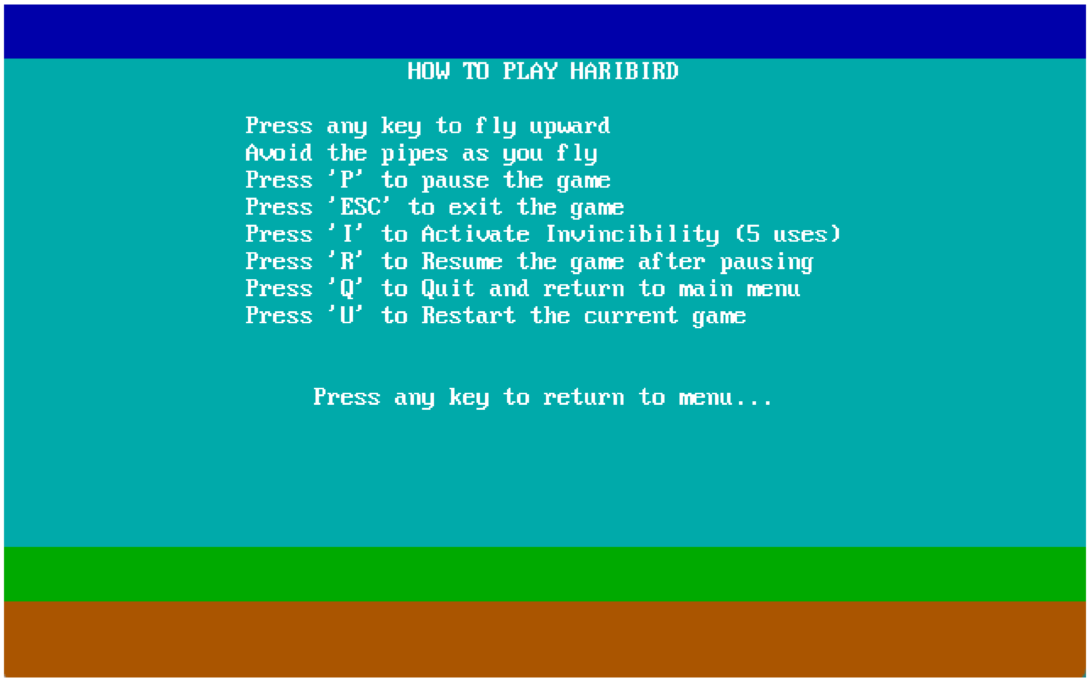

Your Score: 0

GAME OVER!

Would you like to try again?

1) Yes
2) No, back to menu

‘How to Play’ Screen



```
HOW TO PLAY HARIBIRD

Press any key to fly upward
Avoid the pipes as you fly
Press 'P' to pause the game
Press 'ESC' to exit the game
Press 'I' to Activate Invincibility (5 uses)
Press 'R' to Resume the game after pausing
Press 'Q' to Quit and return to main menu
Press 'U' to Restart the current game


Press any key to return to menu...
```

VII. Updated Code

```
use16

org 0x100



    mov ax,0x0002   ; Set 80x25 text mode

    int 0x10        ; Call BIOS

    cld             ; Reset direction flag (so stosw increments registers)

    mov ax,0xb800   ; Point to video segment

    mov ds,ax       ; Both the source (common access)

    mov es,ax       ; and target segments

    mov word [high_score], 0 ; Initialize high score to 0 only once at program start
    ; Display menu

    ;

menu:

    call clear_screen

    mov byte [extra_life_available], 1 ; Reset extra life when returning to main menu

    mov byte [paused], 0    ; Initialize paused state to 0 (not paused)

    ; --- START: Initialize Invincibility in Menu ---

    mov byte [invincibility_uses_left], 5 ; Reset invincibility uses to 5

    mov byte [invincibility_active], 0   ; Ensure invincibility is not active

    mov word [invincibility_timer], 0    ; Reset invincibility timer

    ; --- END: Initialize Invincibility in Menu ---




    ; --- Draw cloud backgrounds with white background ---

    ; First cloud moved 3 rows down (now at row 3-4)

    call draw_cloud_background_row3_col10


    ; Removed middle cloud (was at row 1-2, col 30)
; Third cloud moved 3 rows down (now at row 5-6)

call draw_cloud_background_row5_col50



; Fourth cloud moved 3 rows down (now at row 3-4)

call draw_cloud_background_row3_col60



; Additional small clouds

call draw_small_cloud_row7_col15

call draw_small_cloud_row2_col40

call draw_small_cloud_row6_col70



; --- Display: GROUP 3 PRESENTS ---

mov di, (4 * 80 + 31) * 2     ; Row 4, Column 31

mov ah, 0x3f                  ; White on Cyan

mov al, 'G'                   ; "GROUP 3 PRESENTS:"

stosw

mov al, 'R'

stosw

mov al, 'O'

stosw

mov al, 'U'
```

```
stosw

mov al, 'P'

stosw

mov al, ' '

stosw

mov al, '3'

stosw

mov al, ' '

stosw

mov al, 'P'

stosw

mov al, 'R'

stosw

mov al, 'E'

stosw

mov al, 'S'

stosw

mov al, 'E'

stosw

mov al, 'N'

stosw
```

```
mov al, 'T'

stosw

mov al, 'S'

stosw

mov al, ':'

stosw


; --- HARIBIRD ---

mov di, (5 * 80 + 35) * 2     ; Row 5, Column 35

mov ah, 0x3f

mov al, 'H'

stosw

mov al, 'A'

stosw

mov al, 'R'

stosw

mov al, 'I'

stosw

mov al, 'B'

stosw

mov al, 'I'
```

```
stosw

mov al, 'R'

stosw

mov al, 'D'

stosw

mov al, 'S'

stosw



mov di, (6 * 80 + 35) * 2      ; Row 6, Column 35

mov ah, 0x3f

mov al, 'T'

stosw

mov al, 'I'

stosw

mov al, 'N'

stosw

mov al, 'Y'

stosw

mov al, ' '

stosw
```

```
mov al, 'W'

stosw

mov al, 'I'

stosw

mov al, 'N'

stosw

mov al, 'G'

stosw



mov di, (7 * 80 + 36) * 2      ; Row 7, Column 36

mov ah, 0x3f

mov al, 'E'

stosw

mov al, 'S'

stosw

mov al, 'C'

stosw

mov al, 'A'

stosw

mov al, 'P'

stosw
```

```
mov al, 'E'

stosw



; --- ENTER YOUR CHOICE: ---

mov di, (9 * 80 + 31) * 2     ; Row 9, Column 31

mov ah, 0x3f                  ; White on Cyan

mov al, 'E'

stosw

mov al, 'N'

stosw

mov al, 'T'

stosw

mov al, 'E'

stosw

mov al, 'R'

stosw

mov al, ' '

stosw

mov al, 'Y'

stosw

mov al, 'O'
```

```
stosw

mov al, 'U'

stosw

mov al, 'R'

stosw

mov al, ' '

stosw

mov al, 'C'

stosw

mov al, 'H'

stosw

mov al, 'O'

stosw

mov al, 'I'

stosw

mov al, 'C'

stosw

mov al, 'E'

stosw

mov al, ':'

stosw
```

```
mov al, ' '

stosw

add di, 2

; Move cursor after ": "



; --- 1) EASY ---

mov di, (10 * 80 + 36) * 2     ; Row 10, Column 36

mov ah, 0x3f

mov al, '1'

stosw

mov al, ')'

stosw

mov al, ' '

stosw

mov al, 'E'

stosw

mov al, 'A'

stosw

mov al, 'S'

stosw

mov al, 'Y'
```

```
stosw

; --- 2) MEDIUM ---
mov di, (11 * 80 + 35) * 2    ; Row 11, Column 35
mov ah, 0x3f
mov al, '2'
stosw
mov al, ')'
stosw
mov al, ' '
stosw
mov al, 'M'
stosw
mov al, 'E'
stosw
mov al, 'D'
stosw
mov al, 'I'
stosw
mov al, 'U'
stosw
```

```
mov al, 'M'

stosw



; --- 3) HARD ---

mov di, (12 * 80 + 36) * 2    ; Row 12, Column 36

mov ah, 0x3f

mov al, '3'

stosw

mov al, ')'

stosw

mov al, ' '

stosw

mov al, 'H'

stosw

mov al, 'A'

stosw

mov al, 'R'

stosw

mov al, 'D'

stosw
```

```
; --- 4) HOW TO PLAY ---

mov di, (13 * 80 + 33) * 2    ; Row 13, Column 33

mov ah, 0x3f

mov al, '4'

stosw

mov al, ')'

stosw

mov al, ' '

stosw

mov al, 'H'

stosw

mov al, 'O'

stosw

mov al, 'W'

stosw

mov al, ' '

stosw

mov al, 'T'

stosw

mov al, 'O'

stosw
```

```
    mov al, ' '
    stosw
    mov al, 'P'
    stosw
    mov al, 'L'
    stosw
    mov al, 'A'
    stosw
    mov al, 'Y'
    stosw

    ; Wait for menu selection
    call read_input ;call read input
    ret

; Updated cloud drawing procedures (moved 3 rows down)
draw_cloud_background_row3_col10:
    ; Draw cloud shape at row 3, column 10 (was row 0)
    mov di, (3 * 80 + 10) * 2     ; Start position
    mov cx, 12                    ; Cloud width
    mov ah, 0xF0                  ; White background, black foreground
```

```
        mov al, ' '                  ; Space character
    .loop_row0:
        stosw
        loop .loop_row0

        ; Second row of the cloud (slightly wider)
        mov di, (4 * 80 + 8) * 2      ; Row 4, Column 8 (was row 1)
        mov cx, 15                   ; Cloud width
    .loop_row1:
        stosw
        loop .loop_row1
        ret

; Removed draw_cloud_background_row1_col30 (middle cloud)

draw_cloud_background_row5_col50:
        ; Draw cloud shape at row 5, column 50 (was row 2)
        mov di, (5 * 80 + 50) * 2     ; Start position
        mov cx, 10                   ; Cloud width
        mov ah, 0xF0                 ; White background, black foreground
        mov al, ' '                  ; Space character
```

```
  .loop_row0:
    stosw
    loop .loop_row0

    ; Second row of the cloud (slightly wider)
    mov di, (6 * 80 + 48) * 2     ; Row 6, Column 48 (was row 3)
    mov cx, 14                    ; Cloud width
  .loop_row1:
    stosw
    loop .loop_row1
    ret

draw_cloud_background_row3_col60:
    ; Draw cloud shape at row 3, column 60 (was row 0)
    mov di, (3 * 80 + 60) * 2     ; Start position
    mov cx, 15                    ; Cloud width
    mov ah, 0xF0                  ; White background, black foreground
    mov al, ' '                   ; Space character
  .loop_row0:
    stosw
    loop .loop_row0
    ; Second row of the cloud (slightly wider)
    mov di, (4 * 80 + 58) * 2     ; Row 4, Column 58 (was row 1)
    mov cx, 19                    ; Cloud width
  .loop_row1:
    stosw
    loop .loop_row1
    ret


; New small cloud procedures
draw_small_cloud_row7_col15:
    ; Draw small cloud at row 7, column 15
    mov di, (7 * 80 + 15) * 2     ; Start position
    mov cx, 6                     ; Small cloud width
    mov ah, 0xF0                  ; White background, black foreground
    mov al, ' '                   ; Space character
  .loop_row0:
    stosw
    loop .loop_row0


    ; Second row of the small cloud
    mov di, (8 * 80 + 14) * 2     ; Row 8, Column 14
    mov cx, 8                     ; Slightly wider
  .loop_row1:
    stosw
    loop .loop_row1
    ret


draw_small_cloud_row2_col40:
    ; Draw small cloud at row 2, column 40
    mov di, (2 * 80 + 40) * 2     ; Start position
    mov cx, 5                     ; Small cloud width
    mov ah, 0xF0                  ; White background, black foreground
    mov al, ' '                   ; Space character
  .loop_row0:
    stosw
    loop .loop_row0


    ; Second row of the small cloud
    mov di, (3 * 80 + 39) * 2     ; Row 3, Column 39
    mov cx, 7                     ; Slightly wider
  .loop_row1:
    stosw
    loop .loop_row1
    ret

draw_small_cloud_row6_col70:
    ; Draw small cloud at row 6, column 70
    mov di, (6 * 80 + 70) * 2     ; Start position
    mov cx, 7                     ; Small cloud width
    mov ah, 0xF0                  ; White background, black foreground
    mov al, ' '                   ; Space character
  .loop_row0:
    stosw
    loop .loop_row0

    ; Second row of the small cloud
    mov di, (7 * 80 + 69) * 2     ; Row 7, Column 69
    mov cx, 9                     ; Slightly wider
  .loop_row1:
    stosw
    loop .loop_row1
    ret
```

```
read_input:

        mov ah, 00h     ; Function 0 of int 16h: read keyboard input

        int 0x16        ; BIOS interrupt call

cmp al, 0x1b    ; Is the pressed key Escape (ASCII 0x1b)?

        je near exit_to_dos  ; If yes, jump to our common exit label

        cmp al, '1'     ; Check if the key pressed is '1'

        je easy

        cmp al, '2'     ; Check if the key pressed is '2'

        je medium

        cmp al, '3'     ; Check if the key pressed is '3'

        je hard

        cmp al, '4'     ; Check if the key pressed is '5'

        je how_to_play  ; Jump to how_to_play instructions

        jmp read_input  ; Invalid key, ask again


easy:

        mov byte [user_choice], 1

        jmp fb21


medium:

        mov byte [user_choice], 2
        jmp fb21


hard:
        mov byte [user_choice], 3
        jmp fb21


how_to_play:
        call clear_screen


        ; --- Title: "HOW TO PLAY HARIBIRD" ---
        ; Length: 19. Target Row: 3 (0-indexed: 2)
        ; Start Column: (80 - 19) / 2 = 30
        mov di, (2 * 80 + 30) * 2     ; DI for Row 2, Col 30
        mov ah, 0x3F                  ; Attribute: Bright White on Cyan
        mov al, 'H'
        stosw
        mov al, 'O'
        stosw
        mov al, 'W'
        stosw
        mov al, ' '
```

```
stosw

mov al, 'T'

stosw

mov al, 'O'

stosw

mov al, ' '

stosw

mov al, 'P'

stosw

mov al, 'L'

stosw

mov al, 'A'

stosw

mov al, 'Y'

stosw

mov al, ' '

stosw

mov al, 'H'

stosw

mov al, 'A'

stosw
```

```
mov al, 'R'

stosw

mov al, 'I'

stosw

mov al, 'B'

stosw

mov al, 'I'

stosw

mov al, 'R'

stosw

mov al, 'D'

stosw


; --- Line 1: " Press any key to fly upward" ---

; Target Row: 5 (0-indexed: 4), Col 17

mov di, (4 * 80 + 17) * 2

mov ah, 0x3F            ; Attribute: Bright White on Cyan

mov al, ' '             ; Leading space for alignment

stosw

mov al, 'P'

stosw
```

```
mov al, 'r'

stosw

mov al, 'e'

stosw

mov al, 's'

stosw

mov al, 's'

stosw

mov al, ' '

stosw

mov al, 'a'

stosw

mov al, 'n'

stosw

mov al, 'y'

stosw

mov al, ' '

stosw

mov al, 'k'

stosw

mov al, 'e'
```

```
stosw

mov al, 'y'

stosw

mov al, ' '

stosw

mov al, 't'

stosw

mov al, 'o'

stosw

mov al, ' '

stosw

mov al, 'f'

stosw

mov al, 'l'

stosw

mov al, 'y'

stosw

mov al, ' '

stosw

mov al, 'u'

stosw
```

```
mov al, 'p'

stosw

mov al, 'w'

stosw

mov al, 'a'

stosw

mov al, 'r'

stosw

mov al, 'd'

stosw


; --- Line 2: " Avoid the pipes as you fly" ---

; Target Row: 6 (0-indexed: 5), Col 17

mov di, (5 * 80 + 17) * 2

mov ah, 0x3F            ; Attribute: Bright White on Cyan

mov al, ' '             ; Leading space for alignment

stosw

mov al, 'A'

stosw

mov al, 'v'

stosw
```

```
mov al, 'o'

stosw

mov al, 'i'

stosw

mov al, 'd'

stosw

mov al, ' '

stosw

mov al, 't'

stosw

mov al, 'h'

stosw

mov al, 'e'

stosw

mov al, ' '

stosw

mov al, 'p'

stosw

mov al, 'i'

stosw

mov al, 'p'
```

```
stosw

mov al, 'e'

stosw

mov al, 's'

stosw

mov al, ' '

stosw

mov al, 'a'

stosw

mov al, 's'

stosw

mov al, ' '

stosw

mov al, 'y'

stosw

mov al, 'o'

stosw

mov al, 'u'

stosw

mov al, ' '

stosw
```

```
mov al, 'f'

stosw

mov al, 'l'

stosw

mov al, 'y'

stosw



; --- Line 3: " Press 'P' to pause the game" ---

; Target Row: 7 (0-indexed: 6), Col 17

mov di, (6 * 80 + 17) * 2

mov ah, 0x3F            ; Attribute: Bright White on Cyan

mov al, ' '             ; Leading space for alignment

stosw

mov al, 'P'

stosw

mov al, 'r'

stosw

mov al, 'e'

stosw

mov al, 's'

stosw
```

```
mov al, 's'

stosw

mov al, ' '

stosw

mov al, 0x27                    ; "'

stosw

mov al, 'P'

stosw

mov al, 0x27                    ; "'

stosw

mov al, ' '

stosw

mov al, 't'

stosw

mov al, 'o'

stosw

mov al, ' '

stosw

mov al, 'p'

stosw

mov al, 'a'
```

```
stosw

mov al, 'u'

stosw

mov al, 's'

stosw

mov al, 'e'

stosw

mov al, ' '

stosw

mov al, 't'

stosw

mov al, 'h'

stosw

mov al, 'e'

stosw

mov al, ' '

stosw

mov al, 'g'

stosw

mov al, 'a'

stosw
```

```
mov al, 'm'

stosw

mov al, 'e'

stosw



; --- Line 4: " Press 'ESC' to exit the game" ---

; Target Row: 8 (0-indexed: 7), Col 17

mov di, (7 * 80 + 17) * 2

mov ah, 0x3F            ; Attribute: Bright White on Cyan

mov al, ' '             ; Leading space for alignment

stosw

mov al, 'P'

stosw

mov al, 'r'

stosw

mov al, 'e'

stosw

mov al, 's'

stosw

mov al, 's'

stosw
```

```
mov al, ' '

stosw

mov al, 0x27                    ; "'

stosw

mov al, 'E'

stosw

mov al, 'S'

stosw

mov al, 'C'

stosw

mov al, 0x27                    ; "'

stosw

mov al, ' '

stosw

mov al, 't'

stosw

mov al, 'o'

stosw

mov al, ' '

stosw

mov al, 'e'
```

```
stosw
mov al, 'x'
stosw
mov al, 'i'
stosw
mov al, 't'
stosw
mov al, ' '
stosw
mov al, 't'
stosw
mov al, 'h'
stosw
mov al, 'e'
stosw
mov al, ' '
stosw
mov al, 'g'
stosw
mov al, 'a'
stosw
```

```
mov al, 'm'

stosw

mov al, 'e'

stosw



; --- Line 5: " Press 'I' to Activate Invincibility (5 uses)" ---

; Target Row: 9 (0-indexed: 8), Col 17

mov di, (8 * 80 + 17) * 2

mov ah, 0x3F                ; Attribute: Bright White on Cyan

mov al, ' '                 ; Leading space for alignment

stosw

mov al, 'P' ; Press

stosw

mov al, 'r'

stosw

mov al, 'e'

stosw

mov al, 's'

stosw

mov al, 's'

stosw
```

```
mov al, ' '

stosw

mov al, 0x27 ; '

stosw

mov al, 'I'  ; I

stosw

mov al, 0x27 ; '

stosw

mov al, ' '

stosw

mov al, 't' ; to

stosw

mov al, 'o'

stosw

mov al, ' '

stosw

mov al, 'A' ; Activate Invincibility (5 uses)

stosw

mov al, 'c'

stosw

mov al, 't'
```

```
stosw

mov al, 'i'

stosw

mov al, 'v'

stosw

mov al, 'a'

stosw

mov al, 't'

stosw

mov al, 'e'

stosw

mov al, ' '

stosw

mov al, 'I'

stosw

mov al, 'n'

stosw

mov al, 'v'

stosw

mov al, 'i'

stosw
```

```
mov al, 'n'

stosw

mov al, 'c'

stosw

mov al, 'i'

stosw

mov al, 'b'

stosw

mov al, 'i'

stosw

mov al, 'l'

stosw

mov al, 'i'

stosw

mov al, 't'

stosw

mov al, 'y'

stosw

mov al, ' '

stosw

mov al, '('
stosw

mov al, '5'

stosw

mov al, ' '

stosw

mov al, 'u'

stosw

mov al, 's'

stosw

mov al, 'e'

stosw

mov al, 's'

stosw

mov al, ')'

stosw


; --- Line 6: " Press 'R' to Resume the game after pausing" ---

; Target Row: 10 (0-indexed: 9), Col 17

mov di, (9 * 80 + 17) * 2

mov ah, 0x3F                ; Attribute: Bright White on Cyan

mov al, ' '                 ; Leading space for alignment
```

```
stosw

mov al, 'P' ; Press

stosw

mov al, 'r'

stosw

mov al, 'e'

stosw

mov al, 's'

stosw

mov al, 's'

stosw

mov al, ' '

stosw

mov al, 0x27 ; '

stosw

mov al, 'R'  ; R

stosw

mov al, 0x27 ; '

stosw

mov al, ' '

stosw
```

```
mov al, 't' ; to

stosw

mov al, 'o'

stosw

mov al, ' '

stosw

mov al, 'R' ; Resume the game after pausing

stosw

mov al, 'e'

stosw

mov al, 's'

stosw

mov al, 'u'

stosw

mov al, 'm'

stosw

mov al, 'e'

stosw

mov al, ' '

stosw

mov al, 't'
```

```
stosw

mov al, 'h'

stosw

mov al, 'e'

stosw

mov al, ' '

stosw

mov al, 'g'

stosw

mov al, 'a'

stosw

mov al, 'm'

stosw

mov al, 'e'

stosw

mov al, ' '

stosw

mov al, 'a'

stosw

mov al, 'f'

stosw
```

```
mov al, 't'

stosw

mov al, 'e'

stosw

mov al, 'r'

stosw

mov al, ' '

stosw

mov al, 'p'

stosw

mov al, 'a'

stosw

mov al, 'u'

stosw

mov al, 's'

stosw

mov al, 'i'

stosw

mov al, 'n'

stosw

mov al, 'g'
```

```
stosw


; --- Line 7: " Press 'Q' to Quit and return to main menu" ---

; Target Row: 11 (0-indexed: 10), Col 17

mov di, (10 * 80 + 17) * 2

mov ah, 0x3F                 ; Attribute: Bright White on Cyan

mov al, ' '                  ; Leading space for alignment

stosw

mov al, 'P' ; Press

stosw

mov al, 'r'

stosw

mov al, 'e'

stosw

mov al, 's'

stosw

mov al, 's'

stosw

mov al, ' '

stosw

mov al, 0x27 ; '
```

```
stosw

mov al, 'Q'  ; Q

stosw

mov al, 0x27 ; '

stosw

mov al, ' '

stosw

mov al, 't' ; to

stosw

mov al, 'o'

stosw

mov al, ' '

stosw

mov al, 'Q' ; Quit and return to main menu

stosw

mov al, 'u'

stosw

mov al, 'i'

stosw

mov al, 't'

stosw
```

```
mov al, ' '

stosw

mov al, 'a'

stosw

mov al, 'n'

stosw

mov al, 'd'

stosw

mov al, ' '

stosw

mov al, 'r'

stosw

mov al, 'e'

stosw

mov al, 't'

stosw

mov al, 'u'

stosw

mov al, 'r'

stosw

mov al, 'n'
```

```
stosw

mov al, ' '

stosw

mov al, 't'

stosw

mov al, 'o'

stosw

mov al, ' '

stosw

mov al, 'm'

stosw

mov al, 'a'

stosw

mov al, 'i'

stosw

mov al, 'n'

stosw

mov al, ' '

stosw

mov al, 'm'

stosw
```

```
mov al, 'e'

stosw

mov al, 'n'

stosw

mov al, 'u'

stosw



; --- Line 8: " Press 'U' to Restart the current game" ---

; Target Row: 12 (0-indexed: 11), Col 17

mov di, (11 * 80 + 17) * 2

mov ah, 0x3F                    ; Attribute: Bright White on Cyan

mov al, ' '                     ; Leading space for alignment

stosw

mov al, 'P' ; Press

stosw

mov al, 'r'

stosw

mov al, 'e'

stosw

mov al, 's'

stosw
```

```
mov al, 's'

stosw

mov al, ' '

stosw

mov al, 0x27 ; '

stosw

mov al, 'U'  ; U

stosw

mov al, 0x27 ; '

stosw

mov al, ' '

stosw

mov al, 't' ; to

stosw

mov al, 'o'

stosw

mov al, ' '

stosw

mov al, 'R' ; Restart the current game

stosw

mov al, 'e'
```

```
stosw

mov al, 's'

stosw

mov al, 't'

stosw

mov al, 'a'

stosw

mov al, 'r'

stosw

mov al, 't'

stosw

mov al, ' '

stosw

mov al, 't'

stosw

mov al, 'h'

stosw

mov al, 'e'

stosw

mov al, ' '

stosw
```

```
mov al, 'c'

stosw

mov al, 'u'

stosw

mov al, 'r'

stosw

mov al, 'r'

stosw

mov al, 'e'

stosw

mov al, 'n'

stosw

mov al, 't'

stosw

mov al, ' '

stosw

mov al, 'g'

stosw

mov al, 'a'

stosw

mov al, 'm'
```

```
stosw

mov al, 'e'

stosw


; --- Prompt: "Press any key to return to menu..." ---

; Length: 34. Target Row: 15 (0-indexed: 14), Col 23 (centered)

mov di, (14 * 80 + 23) * 2

mov ah, 0x3F                 ; Attribute: Bright White on Cyan

mov al, 'P'

stosw

mov al, 'r'

stosw

mov al, 'e'

stosw

mov al, 's'

stosw

mov al, 's'

stosw

mov al, ' '

stosw

mov al, 'a'
```

```
stosw

mov al, 'n'

stosw

mov al, 'y'

stosw

mov al, ' '

stosw

mov al, 'k'

stosw

mov al, 'e'

stosw

mov al, 'y'

stosw

mov al, ' '

stosw

mov al, 't'

stosw

mov al, 'o'

stosw

mov al, ' '

stosw
```

```
mov al, 'r'

stosw

mov al, 'e'

stosw

mov al, 't'

stosw

mov al, 'u'

stosw

mov al, 'r'

stosw

mov al, 'n'

stosw

mov al, ' '

stosw

mov al, 't'

stosw

mov al, 'o'

stosw

mov al, ' '

stosw

mov al, 'm'
```

```
    stosw

    mov al, 'e'

    stosw

    mov al, 'n'

    stosw

    mov al, 'u'

    stosw

    mov al, '.'

    stosw

    mov al, '.'

    stosw

    mov al, '.'

    stosw



wait_for_key:

    mov ah, 00h          ; Read keyboard input

    int 0x16             ; Wait for key

    jmp menu             ; Return to menu


fb21:   ; STARTING POINT OF THE GAME
```

```
mov di,pipe     ; Init variables in video segment (saves big bytes)

xor ax,ax

stosw           ; pipe

stosw           ; score

stosw           ; grav

mov al,0xa0

stosw           ; next

mov al,0x60

stosw           ; bird


; Initialize extra life for this game session

mov byte [extra_life_available], 1

; --- START: Initialize Invincibility at Game Start ---

mov byte [invincibility_uses_left], 5 ; Max 5 uses per game

mov byte [invincibility_active], 0   ; Not active initially

mov word [invincibility_timer], 0    ; Timer starts at 0

; --- END: Initialize Invincibility at Game Start ---


mov di,0x004a   ; Game title position


mov ax, 0x1f47 ; 'G' in white on BLUE
```

```
stosw

mov ax, 0x1f52 ; 'R' in white on BLUE

stosw

mov ax, 0x1f4f ; 'O' in white on BLUE

stosw

mov ax, 0x1f55 ; 'U' in white on BLUE

stosw

mov ax, 0x1f50 ; 'P' in white on BLUE

stosw

mov ax, 0x1f20 ; Space in white on BLUE

stosw

mov ax, 0x1f33 ; '3' in white on BLUE

stosw

mov ax, 0x1f3a ; ':' in white on BLUE

stosw

mov ax, 0x1f20 ; Space in white on BLUE

stosw

mov ax, 0x1f48 ; 'H' in white on BLUE

stosw

mov ax, 0x1f41 ; 'A' in white on BLUE

stosw
```

```
        mov ax, 0x1f52 ; 'R' in white on BLUE

        stosw

        mov ax, 0x1f49 ; 'I' in white on BLUE

        stosw

        mov ax, 0x1f42 ; 'B' in white on BLUE

        stosw

        mov ax, 0x1f49 ; 'I' in white on BLUE

        stosw

        mov ax, 0x1f52 ; 'R' in white on BLUE

        stosw

        mov ax, 0x1f44 ; 'D' in white on BLUE

        stosw


        mov cx,80       ; Introduce 80 columns of scenery

fb1:    push cx

        call scroll_scenery

        pop cx

        loop fb1


fb23:   mov ah,0x01     ; Check if key pressed

        int 0x16
        pushf
        xor ax,ax       ; Wait for a key
        int 0x16
        popf
        jnz fb23        ; Jump if key was accumulated, if not already waited for key

        ;
        ; Main loop
        ;
fb12:   ; Calculate bird's new vertical position and screen offset
        mov al,[bird]
        add al,[grav]
        mov [bird],al
        and al,0xf8     ; Mask fraction for row calculation
        mov ah,0x14     ; Multiply by 20 (80 columns * 2 bytes/char / 8 = 20)
        mul ah          ; AX = screen row offset
        add ax,$0020    ; Add fixed column offset (column 16)
        xchg ax,di      ; DI holds the screen memory offset for the bird's tail (◄)

        ; --- START OF NEW BIRD DRAWING, COLLISION, AND ERASING LOGIC ---
        ; Bird design: Wings Up: ▲ / ◄░█(•► , Wings Down: ◄░▀(•►
        ; Bird is 7 characters long, from di to di+12.


        ; Check for invincibility first

        cmp byte [invincibility_active], 1

        je near .draw_bird_and_continue ; If invincible, skip collision check and draw directly


        ; New Collision Detection (bird is ◄░█(•► or ◄░▀(•► from di to di+12)

        mov al, [es:di]     ; Tail ◄ at di

        cmp al, 0x20

        jne .collision_detected_new_impl

        mov al, [es:di+2]   ; Body ░ at di+2

        cmp al, 0x20

        jne .collision_detected_new_impl

        mov al, [es:di+4]   ; Body █/▀ (first part) at di+4

        cmp al, 0x20

        jne .collision_detected_new_impl

        mov al, [es:di+6]   ; Body █/▀ (second part) at di+6

        cmp al, 0x20

        jne .collision_detected_new_impl

        mov al, [es:di+8]   ; Body ( at di+8

        cmp al, 0x20
```

```
    jne .collision_detected_new_impl

    mov al, [es:di+10]  ; Head 0 (was •) at di+10

    cmp al, 0x20

    jne .collision_detected_new_impl

    mov al, [es:di+12]  ; Beak ► at di+12

    cmp al, 0x20

    jne .collision_detected_new_impl



    mov bl, [frame]

    and bl, 4           ; Check 3rd bit of frame counter for flap state

    jz .skip_upper_wing_collision_check_new ; If zero, it's flap_down (◄░▀▀(•►), so skip upper wing
check

    ; Collision check for upper wing ▲ (for ◄░██(•► design)

    mov al, [es:di-160+4] ; Position of upper wing ▲ (above the first █ of body)

    cmp al, 0x20

    jne .collision_detected_new_impl

.skip_upper_wing_collision_check_new:

    jmp .draw_bird_and_continue



.collision_detected_new_impl:

    mov [saved_crash_di], di
```

```
 ;——— PC-Speaker beep @ ~1 kHz ———

 ; 1) Tell PIT channel 2 we want mode 3 (square wave)

 mov    al, 0xB6        ; 1011 0110b: Ch2, lo/hi, mode3, binary

 out    0x43, al



 ; 2) Send divisor = 1193 → ~1 000 Hz

 mov    bx, 1193

 mov    al, bl          ; low byte

 out    0x42, al

 mov    al, bh          ; high byte

 out    0x42, al



 ; 3) Turn speaker ON (port 0x61 bits 0+1)

 in     al, 0x61

 or     al, 00000011b

 out    0x61, al



 ; 4) Simple delay (adjust CX for length)

 mov    cx, 0x4000

.tone_delay:

 loop   .tone_delay
```

```
   ; 5) Turn speaker OFF

   in     al, 0x61

   and    al, 11111100b

   out    0x61, al

   ;————————————————————————————————————————————————————————————

      ; Draw crash symbols '**' over the bird's body (on the █/▀ parts)

      mov word [es:di+4], 0x3C2A   ; Attribute 0x3C (Bright Red/Cyan), Char 0x2A ('*')

      mov word [es:di+6], 0x3C2A   ; Attribute 0x3C (Bright Red/Cyan), Char 0x2A ('*')

      jmp game_over_check_extra_life


.draw_bird_and_continue:

      mov al, [frame]

      and al, 4

      jz .flap_up_new_impl ; If bit is 0, flap up (◄░█(•► with ▲)


.flap_down_new_impl: ; Bird state: ◄░▀(•►

   mov word [es:di-160+4], 0x3020    ; Erase potential upper wing ▲ (Space on Cyan: Attr 0x30, Char
0x20)

   mov word [es:di],     0x3E11     ; ◄ tail (Char 0x11) (Yellow on Cyan: Attr 0x3E)

   mov word [es:di+2],   0x3EB0     ; ░ body (Char 0xB0)
```

```
    mov word [es:di+4],   0x3EDC     ; ▀ wing/body (Char 0xDC - Lower half block)

    mov word [es:di+6],   0x3EDC     ; ▀ wing/body (Char 0xDC - Lower half block)

    mov word [es:di+8],   0x3E28     ; ( body (Char 0x28)

    mov word [es:di+10],  0x3E30     ; 0 head/eye (Char 0x30 - digit zero)

    mov word [es:di+12],  0x3E10     ; ► beak (Char 0x10)

    jmp .after_draw_new_impl


.flap_up_new_impl: ; Bird state: ▲ / ◄░██(•►

    mov word [es:di-160+4], 0x3E1E   ; ▲ upper wing (Char 0x1E) (Yellow on Cyan: Attr 0x3E)

    mov word [es:di],     0x3E11     ; ◄ tail (Char 0x11)

    mov word [es:di+2],   0x3EB0     ; ░ body (Char 0xB0)

    mov word [es:di+4],   0x3EDB     ; █ body (Char 0xDB - Full block)

    mov word [es:di+6],   0x3EDB     ; █ body (Char 0xDB - Full block)

    mov word [es:di+8],   0x3E28     ; ( body (Char 0x28)

    mov word [es:di+10],  0x3E30     ; 0 head/eye (Char 0x30 - digit zero)

    mov word [es:di+12],  0x3E10     ; ► beak (Char 0x10)


.after_draw_new_impl:

    call display_status_messages

    call wait_frame
    ; Erase bird (7 characters + 1 optional upper wing)
    mov ax, 0x3020       ; Space on Cyan background (Attribute 0x30, Char 0x20)
    mov [es:di-160+4], ax  ; Erase upper wing location
    mov [es:di], ax        ; Erase ◄
    mov [es:di+2], ax      ; Erase ░
    mov [es:di+4], ax      ; Erase ▌/▀
    mov [es:di+6], ax      ; Erase ▌/▀
    mov [es:di+8], ax      ; Erase (
    mov [es:di+10], ax     ; Erase 0 (was •)
    mov [es:di+12], ax     ; Erase ►

    mov al,[frame]
    and al,7
    jnz .skip_gravity_inc_for_new_bird
    inc word [grav]
.skip_gravity_inc_for_new_bird:
    ; --- END OF NEW BIRD LOGIC ---

    jmp perform_difficulty_scrolling ; Jump to pipe scrolling section

perform_difficulty_scrolling:
    xor al, al

    mov al, [user_choice]

    cmp al, 1

    je .scroll_easy

    cmp al, 2

    je .scroll_medium

    cmp al, 3

    je .scroll_hard



    ; Default scroll (should not be reached if user_choice is always 1, 2, or 3)

    call scroll_scenery

    jmp after_scrolling_logic



.scroll_easy:

    call scroll_scenery

    call scroll_scenery

    jmp after_scrolling_logic



.scroll_medium:

    call scroll_scenery

    call scroll_scenery
    call scroll_scenery

    jmp after_scrolling_logic


.scroll_hard:

    call scroll_scenery

    call scroll_scenery

    call scroll_scenery

    call scroll_scenery

    ; Fall through to after_scrolling_logic


after_scrolling_logic:

    ; This is where the old fb_end logic (score checking, input) begins

    cmp byte [0x00a0], 0xb0     ; Check if first char of pipe line is a pipe segment

    jz fb27

    cmp byte [0x00a2], 0xb0     ; Check second char


fb27:   jnz fb24                    ; If not pipe, don't increment score

        inc word [score]            ; Increment score

 mov al, 0xB6            ; Select PIT channel 2, mode 3 (square wave)

    out (0x43), al
```

```
; Turn speaker on FIRST

in  al, (0x61)          ; Read port 0x61

or  al, 0x03            ; enable speaker

out (0x61), al          ; Write back to enable speaker sound



; First, higher pitch (briefly)

mov ax, 0x0180          ; divisor = 384 -> frequency ≈ 3107 Hz (high pitch)

out (0x42), al          ; Send low byte of divisor

mov al, ah              ; high byte

out (0x42), al          ; Send high byte of divisor

; No explicit delay here, we want it to change quickly

; For a very short blip of this first tone, you might add a few NOPs or a tiny delay loop

; For example:

; mov cx, 5000

; .short_delay_zap:

;   dec cx

;   jnz .short_delay_zap



; Second, slightly lower pitch (this one will be held for wait_frame)

mov ax, 0x0280          ; divisor = 640 -> frequency ≈ 1864 Hz (mid-high pitch)

out (0x42), al          ; Send low byte of divisor
```

```
    mov al, ah              ; high byte
    out (0x42), al          ; Send high byte of divisor


    call wait_frame         ; wait a bit so the second tone is audible


    ; Turn speaker off
    in  al, (0x61)          ; read port 0x61 again
    and al, 0xFC            ; disable speaker
    out (0x61), al          ; Write back to disable speaker sound


        mov ax,[score]
        mov di,0x008e                ; Position for score display (top right)
fb25:   xor dx,dx                    ; Clear DX for division
        mov bx,10                    ; Divisor for BCD conversion
        div bx                       ; AX = AX / 10, DX = remainder
        add dx,0x3E30                ; Convert remainder to ASCII char, Bright Red on CYAN (Attr 0x3C, '0' is
0x30)
        xchg ax,dx                   ; Use DX (char) for stosw, save quotient in AX
        std                          ; Set direction flag for stosw to decrement DI (right to left display)
        stosw                        ; Store char and attribute
        mov word [es:di],0x3720      ; Clean next char position with space, Light Gray on CYAN
        cld                     ; Clear direction flag
        xchg ax,dx              ; Restore quotient to AX
        or ax,ax                ; Check if quotient is zero
        jnz fb25                ; Loop if not zero
fb24:   mov ah,0x01             ; Check for key press (non-blocking)
        int 0x16
        jz near fb26            ; If no key pressed, jump to loop start

        mov ah,0x00             ; Key pressed, get it (blocking)
        int 0x16                ; AL = ASCII code, AH = scan code

        cmp al,0x1b             ; ESC key?
        je exit_game_to_dos     ; Exit to DOS

        cmp al,'p'              ; 'p' for pause?
        je handle_game_pause_jmp     ; Jump to pause handler
        cmp al,'P'              ; 'P' for pause?
        je handle_game_pause_jmp     ; Jump to pause handler

        cmp al, 'i'             ; 'i' for invincibility?
        je process_invincibility_key_attempt
```

```
cmp al, 'I'                 ; 'I' for invincibility?

je process_invincibility_key_attempt



jmp near fb4                ; If other key (space assumed for flap), jump to flap logic



exit_game_to_dos:

int 0x20                    ; Terminate program



handle_game_pause_jmp:

jmp handle_pause            ; Jump to pause handling routine



process_invincibility_key_attempt:

cmp byte [invincibility_active], 1 ; Already active?

jne try_activate_invincibility_check_uses ; If not, try to activate

jmp near fb26                       ; If active, ignore key, continue game loop



try_activate_invincibility_check_uses:

cmp byte [invincibility_uses_left], 0 ; Any uses left?

jne can_activate_this_invincibility   ; If yes, activate

jmp near fb26                     ; If no uses left, ignore key, continue
```

```
    can_activate_this_invincibility:

    mov byte [invincibility_active], 1    ; Activate invincibility

    dec byte [invincibility_uses_left]    ; Decrement uses left

    mov word [invincibility_timer], 145   ; Set timer (approx 8 seconds at ~18.2 ticks/sec)

    jmp near fb26                         ; Continue game loop


handle_pause:

    mov byte [paused], 1        ; Set paused flag

    call display_pause_menu     ; Show pause menu

pause_input_loop:

    mov ah, 00h                 ; Wait for key press

    int 0x16


    cmp al, 'r'                 ; 'r' to resume?

    je resume_game

    cmp al, 'R'                 ; 'R' to resume?

    je resume_game


    cmp al, 'q'                 ; 'q' to quit to menu?

    je quit_to_menu

    cmp al, 'Q'                 ; 'Q' to quit to menu?
    je quit_to_menu



cmp al, 'u'             ; 'u' to restart

    je near restart_from_pause      ; Jump to our new restart label

    cmp al, 'U'             ; 'U' to restart (uppercase)

    je near restart_from_pause      ; Jump to our new restart label

    cmp al, 0x1b            ; ESC to quit to menu?

    je near exit_to_dos         ; <<< CHANGED LINE



    jmp pause_input_loop        ; Loop for valid pause menu input


resume_game:

    mov byte [paused], 0        ; Clear paused flag

    call clear_pause_menu       ; Erase pause menu from screen

    jmp near fb26           ; Continue game loop (effectively re-enters main loop)


quit_to_menu:

    call clear_screen           ; Clear the game screen

    jmp menu                ; Go back to the main menu
```

```
display_pause_menu:

    mov di, 0x05E4 ; Row 11, Col 34

    mov ax, 0x3F47  ; 'G' in white on CYAN

    stosw

    mov al, 0x41    ; 'A'

    stosw

    mov al, 0x4D    ; 'M'

    stosw

    mov al, 0x45    ; 'E'

    stosw

    mov al, 0x20    ; ' ' (space)

    stosw

    mov al, 0x50    ; 'P'

    stosw

    mov al, 0x41    ; 'A'

    stosw

    mov al, 0x55    ; 'U'

    stosw

    mov al, 0x53    ; 'S'

    stosw
```

```
mov al, 0x45    ; 'E'

stosw

mov al, 0x44    ; 'D'

stosw


mov di, 0x0722    ;

mov ax, 0x3E72  ; 'r' in yellow on CYAN

stosw

mov al, 0x29    ; ')'

stosw

mov al, 0x20    ; ' ' (space)

stosw

mov al, 0x52    ; 'R'

stosw

mov al, 0x65    ; 'e'

stosw

mov al, 0x73    ; 's'

stosw

mov al, 0x75    ; 'u'

stosw

mov al, 0x6D    ; 'm'
```

```
    stosw

    mov al, 0x65    ; 'e'

    stosw

    mov al, 0x20    ; ' ' (space)

    stosw

    mov al, 0x47    ; 'G'

    stosw

    mov al, 0x61    ; 'a'

    stosw

    mov al, 0x6D    ; 'm'

    stosw

    mov al, 0x65    ; 'e'

    stosw




; “u) Restart Game” (yellow on cyan)

 mov di, 0x07C0    ;



 mov ax, 0x3E75    ; 'u' in yellow on cyan

 stosw

 mov al, 0x29      ; ')'
```

```
stosw
mov al, 0x20     ; ' '
stosw
mov al, 0x52     ; 'R'
stosw
mov al, 0x65     ; 'e'
stosw
mov al, 0x73     ; 's'
stosw
mov al, 0x74     ; 't'
stosw
mov al, 0x61     ; 'a'
stosw
mov al, 0x72     ; 'r'
stosw
mov al, 0x74     ; 't'
stosw
mov al, 0x20     ; ' '
stosw
mov al, 0x47     ; 'G'
stosw
```

```
mov al, 0x61      ; 'a'

stosw

mov al, 0x6D      ; 'm'

stosw

mov al, 0x65      ; 'e'

stosw

    mov di, 0x0860 ; Row 17, Col 16

    mov ax, 0x3E71  ; 'q' in yellow on CYAN

    stosw

    mov al, 0x29    ; ')'

    stosw

    mov al, 0x20    ; ' ' (space)

    stosw

    mov al, 0x51    ; 'Q'

    stosw

    mov al, 0x75    ; 'u'

    stosw

    mov al, 0x69    ; 'i'

    stosw

    mov al, 0x74    ; 't'

    stosw
```

```
        mov al, 0x20    ; ' ' (space)

        stosw

        mov al, 0x74    ; 't'

        stosw

        mov al, 0x6F    ; 'o'

        stosw

        mov al, 0x20    ; ' ' (space)

        stosw

        mov al, 0x4D    ; 'M'

        stosw

        mov al, 0x65    ; 'e'

        stosw

        mov al, 0x6E    ; 'n'

        stosw

        mov al, 0x75    ; 'u'

        stosw


        ret


clear_pause_menu:

        mov cx, 80      ; Number of characters (words) to clear per line
```

```
    mov ax, 0x3020  ; Space character with light cyan background (0x30)



   ; Clear row 10 (GAME PAUSED was at 0x05E4)

mov di, 0x05A0  ; Start clearing from the beginning of the row

push cx

rep stosw       ; Clear row 10

pop cx



; Clear row 12 (r) Resume Game was at 0x0720)

mov di, 0x06E0  ; Start clearing from the beginning of the row

push cx

rep stosw       ; Clear row 12

pop cx



; --- ADD THIS LINE TO CLEAR THE NEW RESTART OPTION ON ROW 13 ---

; Clear row 13 (u) Restart Game is at 0x0800)

mov di, 0x0780  ; Start clearing from the beginning of the row

push cx

rep stosw       ; Clear row 13

pop cx
```

```
    ; --- END ADDITION ---


    ; Clear row 15 (q) Quit to Menu is now at 0x08FC)

    mov di, 0x0860; Start clearing from the beginning of the row

    mov cx, 80      ; Make sure CX is 80 for the last rep stosw

    rep stosw       ; Clear row 15


    ret

restart_from_pause:

    ; Clear the pause menu text from the screen first

    call clear_pause_menu


    ; Ensure the paused flag is reset (important before restarting)

    mov byte [paused], 0


    ; Jump back to the main game initialization point (where scores, pipes, bird are reset)

    jmp fb21

fb4:    ; Bird flap logic (when space or other non-handled key is pressed)

        mov ax,[bird]

        sub ax,0x10                 ; Move bird up (decrease row value by 2, as 0x08 is one row)

        cmp ax,0x08                  ; Check if bird hits top (row 1)
        jb fb18                 ; If below top, don't let it go higher

        mov [bird],ax               ; Update bird's new higher position

fb18:   mov byte [grav],0  ; Reset gravity



        ; Super dynamic Flappy Bird-style wing flapping sound! (Cooler Version - Attempt 2)

        mov al,0xb6        ; Command byte: Select channel 2, mode 3 (square wave)

        out (0x43),al      ; Write to PIT command register



        ; Turn on the speaker

        in al,(0x61)       ; Read current value of port 0x61

        or al,0x03         ; Set bits 0 and 1 to enable speaker

        out (0x61),al      ; Write back



        ; *** PHASE 1: VERY HIGH, VERY SHORT "TICK" or start of a "ZAP" (~4000 Hz) ***

        mov al,0xb6        ; Command byte: Select channel 2, mode 3 (square wave)

        out (0x43),al      ; Write to PIT command register

        ; 1193180 / 4000 = 298 (0x012A)

        mov ax,0x012A      ; Frequency divisor for ~4000 Hz (Very high!)

        out (0x42),al      ; Send low byte

        mov al,ah          ; Move high byte to AL

        out (0x42),al      ; Send high byte
```

```
    ; Extremely brief for a sharp attack

    mov cx,1

fb18_whoosh:

    push cx

    mov cx,5         ; EXTREMELY short "tick" (Original: 20)

fb18_inner_whoosh:

    loop fb18_inner_whoosh

    pop cx

    loop fb18_whoosh



    ; *** PHASE 2: RAPID DROP to a MID-frequency (~1200 Hz) ***

    mov al,0xb6

    out (0x43),al

    ; 1193180 / 1200 = 994 (0x03E2)

    mov ax,0x03E2    ; Frequency divisor for ~1200 Hz (Quick drop)

    out (0x42),al

    mov al,ah

    out (0x42),al



    ; Very short to make the frequency drop feel fast
    mov cx,1
fb18_swoop:
    push cx
    mov cx,8        ; Very short for quick transition (Original: 15)
fb18_inner_swoop:
    loop fb18_inner_swoop
    pop cx
    loop fb18_swoop

    ; *** PHASE 3: Main "FLAP" BODY tone, mid-low (~700 Hz) ***
    mov al,0xb6
    out (0x43),al
    ; 1193180 / 700 = 1704 (0x06A8)
    mov ax,0x06A8   ; Frequency divisor for ~700 Hz (The main "body" of the flap)
    out (0x42),al
    mov al,ah
    out (0x42),al

    ; This is the main audible part of the "flap"
    mov cx,1
fb18_mid:
    push cx
    mov cx,35          ; Main flap duration (Original: 25)
fb18_inner_mid:
    loop fb18_inner_mid
    pop cx
    loop fb18_mid

    ; *** PHASE 4: Distinct LOW "THUD" or impact (~300 Hz) ***
    mov al,0xb6
    out (0x43),al
    ; 1193180 / 300 = 3977 (0x0F89)
    mov ax,0x0F89      ; Frequency divisor for ~300 Hz (Low thud)
    out (0x42),al
    mov al,ah
    out (0x42),al

    ; Noticeable thud, but not too long
    mov cx,1           ; Outer loop count for this phase (Original was cx,2)
fb18_pop:
    push cx
    mov cx,25          ; Thud duration (Original: 55)
```

```
fb18_inner_pop:

        loop fb18_inner_pop

        pop cx

        loop fb18_pop



        ; *** PHASE 5: VERY SHORT, LOW "FADE" or end (~150 Hz) ***

        mov al,0xb6

        out (0x43),al

        ; 1193180 / 150 = 7954 (0x1F12)

        mov ax,0x1F12      ; Frequency divisor for ~150 Hz (Very low, quick end)

        out (0x42),al

        mov al,ah

        out (0x42),al



        ; Extremely brief final sound

        mov cx,1

fb18_fade:

        push cx

        mov cx,7           ; Super short fade (Original: 30)

fb18_inner_fade:

        loop fb18_inner_fade
        pop cx
        loop fb18_fade


        ; Turn off the speaker
        in al,(0x61)       ; Read port again
        and al,0x fc       ; Clear bits 0 and 1 to disable speaker
        out (0x61),al      ; Turn off sound


        ; Continue to main loop
fb26:   jmp near fb12               ; Jump back to the start of the main game loop (as per your original)


game_over_check_extra_life:
        cmp byte [extra_life_available], 1
        jne display_oops_and_real_game_over


        mov byte [extra_life_available], 0
        call display_extra_life_prompt_sub


read_extra_life_input_loop:
        mov ah, 00h
        int 0x16
```

```
    cmp al, '1'

    je handle_use_extra_life

    cmp al, '2'

    je handle_give_up_extra_life

    jmp read_extra_life_input_loop


handle_use_extra_life:

    call clear_extra_life_prompt_sub ; Clear the prompt text from screen



    mov di, [saved_crash_di]    ; Get DI where bird crashed

    mov byte [es:di], ' '       ; Erase '*' crash marker (char part)

    mov byte [es:di+2], ' '     ; Erase second '*' crash marker (char part)

                                ; Attributes at [di+1], [di+3] remain (background color)



    mov byte [bird], 0x60       ; Reset bird's logical Y position to a safe mid-value

    mov word [grav], 0          ; Reset gravity, so bird doesn't instantly plummet

    jmp fb12                    ; Resume main game loop


handle_give_up_extra_life:

    call clear_extra_life_prompt_sub ; Clear the prompt text

    ; Fall through to display_oops_and_real_game_over
    ; The '*' crash markers are already on screen from before the prompt.


display_oops_and_real_game_over:
    ; Display "OOPS!" message
    mov di,0x37CA          ; Position for "OOPS!"
    mov di, 0x0724         ; Centered-ish for "OOPS!"
    mov ax,0x3c4f          ; 'O' in Red (attribute 0x0C)
    stosw
    mov al,0x4f            ; 'O' (AH still 0x0C)
    stosw
    mov al,0x50            ; 'P'
    stosw
    mov al,0x53            ; 'S'
    stosw
    mov al,0x21            ; '!'
    stosw


    mov cx,80              ; Wait up to 80 frames
oops_delay_loop:
    ; Check for key press (non-blocking)
    mov ah, 01h
    int 16h
    jnz skip_oops_delay     ; If a key was pressed, skip the delay


    ; Otherwise, wait a frame
    push cx
    call wait_frame
    pop cx
    loop oops_delay_loop
skip_oops_delay:


    call clear_screen       ; Clear screen




    mov ax, [score]         ; Get current score
    cmp ax, [high_score]    ; Compare with high score
    jle .skip_high_score_update ; If score <= high_score, skip update


    mov [high_score], ax    ; If score > high_score, update high_score
    .skip_high_score_update:
    ; Now proceed to game over menu
    jmp game_over_menu      ; Go to "Try again? Yes/No" menu
```

```
game_over_menu:

    mov di, 0x05E4 ; Row 11, Col 34

    mov ax, 0x3447  ; 'G' in Red on CYAN

    stosw

    mov al, 0x41    ; 'A'

    stosw

    mov al, 0x4d    ; 'M'

    stosw

    mov al, 0x45    ; 'E'

    stosw

    mov al, 0x20    ; ' ' (space)

    stosw

    mov al, 0x4f    ; 'O'

    stosw

    mov al, 0x56    ; 'V'

    stosw

    mov al, 0x45    ; 'E'

    stosw

    mov al, 0x52    ; 'R'

    stosw
    mov al, 0x21    ; '!'

    stosw

;; <<< START: ADD CODE TO DISPLAY YOUR SCORE LABEL >>>

    mov di, 0x07C2          ; Set DI to Row 13 (index 12), Col 34 (index 33)

    mov ah, 0x3e            ; Set attribute to Yellow on Cyan (0x3E)

    mov al, 'Y'             ; Write "Your Score: "

    stosw

    mov al, 'o'

    stosw

    mov al, 'u'

    stosw

    mov al, 'r'

    stosw

    mov al, ' '

    stosw

    mov al, 'S'

    stosw

    mov al, 'c'

    stosw

    mov al, 'o'

    stosw
```

```
        mov al, 'r'

        stosw

        mov al, 'e'

        stosw

        mov al, ':'

        stosw

        mov al, ' '          ; Space before the number

        stosw                ; After this, DI points where the *first character after the space* would go.

                           ; This is the position for the *rightmost* digit if we print right-to-left.

; ... (code for displaying "Your Score: " label) ...

; DI is now pointing where the *first character after the space* would go.

; This is the position for the *leftmost* digit.



    mov ax, [score]        ; Load the player's score value into AX.

    mov bx, 10             ; Set BX to 10 for decimal conversion.

    xor cx, cx             ; Clear CX (digit counter).



.yourscore_push_digits:

    xor dx, dx             ; Clear DX for DIV.

    div bx                 ; Divide AX by BX. Quotient in AX, Remainder in DX.

    push dx                ; Push the remainder (digit) onto the stack.
    inc cx                  ; Increment digit counter.

    or ax, ax               ; Check if quotient (AX) is zero.

    jnz .yourscore_push_digits ; Loop if AX is not zero.



; --- Handle the case where the score is 0 ---

    cmp cx, 0

    jnz .yourscore_pop_and_display ; If CX is not 0, we have digits.

; If CX *is* 0:

    mov cx, 1               ; Set CX to 1 to display one '0'.

    xor dx, dx              ; DX = 0.

    push dx                 ; Push 0 onto the stack.



; --- Pop Digits from Stack and Display ---

.yourscore_pop_and_display:

    pop ax                  ; Pop the digit into AX (AL).

    add al, '0'             ; Convert numerical digit to ASCII character.

    mov ah, 0x3e            ; Attribute: Yellow foreground on Cyan background.

                            ; REMOVED: std - We want DI to increment.

    stosw                   ; Store AX (character and attribute) at ES:DI.

                            ; DI will now increment (assuming DF is clear globally, which it should be for other parts
to work).
```

```
                    ; REMOVED: cld - Not needed as std was removed.

   loop .yourscore_pop_and_display ; Decrement CX and loop if CX is not zero.



; Display "High Score:"

    ; Target DI = 0x0544 (Row 9, Col 35)

    mov di, 0x0544

    mov ah, 0x3e        ; Yellow on Cyan attribute

    mov al, 'H'         ; "High Score: " (Length 12)

    stosw

    mov al, 'i'

    stosw

    mov al, 'g'

    stosw

    mov al, 'h'

    stosw

    mov al, ' '

    stosw

    mov al, 'S'

    stosw

    mov al, 'c'

    stosw
```

```
mov al, 'o'

stosw

mov al, 'r'

stosw

mov al, 'e'

stosw

mov al, ':'

stosw

mov al, ' '

stosw



; Convert and display the high score number (adapt from show_current_score)

mov ax, [high_score]

mov bx, 10      ; Divisor

xor cx, cx      ; Digit counter



.highscore_push_digits:

    xor dx, dx      ; Clear high word for division

    div bx          ; AX = AX/10, DX = remainder

    push dx         ; Save digit

    inc cx          ; Count digits
    or ax, ax       ; Check if quotient is 0
    jnz .highscore_push_digits ; If not, continue

; Handle case where high score is 0 (display '0')
cmp cx, 0
jnz .highscore_pop_and_display
mov cx, 1 ; If count is 0, push 0 to display '0'
xor dx, dx
push dx

.highscore_pop_and_display:
    pop ax          ; Get digit (0-9)
    add al, '0'     ; Convert to ASCII ('0'-'9')
    mov ah, 0x3e    ; Yellow attribute (match label)
    stosw           ; Write digit character and attribute
    loop .highscore_pop_and_display

mov di, 0x08D4 ; Row 15, Col 10  ;; <<< CHANGE THIS LINE >>>

mov ax, 0x3e57  ; 'W' in Yellow on CYAN
stosw
```

```
mov al, 0x6f    ; 'o'

stosw

mov al, 0x75    ; 'u'

stosw

mov al, 0x6c    ; 'l'

stosw

mov al, 0x64    ; 'd'

stosw

mov al, 0x20    ; ' '

stosw

mov al, 0x79    ; 'y'

stosw

mov al, 0x6f    ; 'o'

stosw

mov al, 0x75    ; 'u'

stosw

mov al, 0x20    ; ' '

stosw

mov al, 0x6c    ; 'l'

stosw

mov al, 0x69    ; 'i'
```

```
stosw

mov al, 0x6b    ; 'k'

stosw

mov al, 0x65    ; 'e'

stosw

mov al, 0x20    ; ' '

stosw

mov al, 0x74    ; 't'

stosw

mov al, 0x6f    ; 'o'

stosw

mov al, 0x20    ; ' '

stosw

mov al, 0x74    ; 't'

stosw

mov al, 0x72    ; 'r'

stosw

mov al, 0x79    ; 'y'

stosw

mov al, 0x20    ; ' '

stosw
```

```
mov al, 0x61    ; 'a'

stosw

mov al, 0x67    ; 'g'

stosw

mov al, 0x61    ; 'a'

stosw

mov al, 0x69    ; 'i'

stosw

mov al, 0x6e    ; 'n'

stosw

mov al, 0x3f    ; '?'

stosw



mov di, 0x0AC8 ; Row 18, Col 20  ;; <<< CHANGE THIS LINE >>>



mov ax, 0x3e31  ; '1' in Yellow on CYAN

stosw

mov al, 0x29    ; ')'

stosw

mov al, 0x20    ; ' '

stosw
```

```
mov al, 0x59    ; 'Y'

stosw

mov al, 0x65    ; 'e'

stosw

mov al, 0x73    ; 's'

stosw



 mov di, 0x0B68 ; Row 19, Col 20  ;; <<< CHANGE THIS LINE >>>

mov ax, 0x3e32  ; '2' in Yellow on CYAN

stosw

mov al, 0x29    ; ')'

stosw

mov al, 0x20    ; ' '

stosw

mov al, 0x4e    ; 'N'

stosw

mov al, 0x6f    ; 'o'

stosw

mov al, 0x2c    ; ','

stosw

mov al, 0x20    ; ' '
```

```
stosw

mov al, 0x62    ; 'b'

stosw

mov al, 0x61    ; 'a'

stosw

mov al, 0x63    ; 'c'

stosw

mov al, 0x6b    ; 'k'

stosw

mov al, 0x20    ; ' '

stosw

mov al, 0x74    ; 't'

stosw

mov al, 0x6f    ; 'o'

stosw

mov al, 0x20    ; ' '

stosw

mov al, 0x6d    ; 'm'

stosw

mov al, 0x65    ; 'e'

stosw
```

```
        mov al, 0x6e    ; 'n'

        stosw

        mov al, 0x75    ; 'u'

        stosw


game_over_read_input:

        xor al, al

        mov ah, 00h     ; Wait for key press

        int 0x16

cmp al, 0x1b    ; Is the pressed key Escape?

        je near exit_to_dos  ; If yes, jump to our common exit label

        cmp al, '1'     ; '1' to play again?

        je fb21         ; Jump to game start


        cmp al, '2'     ; '2' to go to menu?

        je menu         ; Jump to main menu


        jmp game_over_read_input ; Loop for valid input




display_extra_life_prompt_sub:
```

```
mov di, 0x05CE ; Row 11, Col 23

mov ah, 0x3E ; Yellow text on CYAN

mov al, 'D'; stosw

stosw

mov al, 'o'; stosw

stosw

mov al, ' '; stosw

stosw

mov al, 'y'; stosw

stosw

mov al, 'o'; stosw

stosw

mov al, 'u'; stosw

stosw

mov al, ' '; stosw

stosw

mov al, 'w'; stosw

stosw

mov al, 'a'; stosw

stosw

mov al, 'n'; stosw
```

```
stosw

mov al, 't'; stosw

stosw

mov al, ' '; stosw

stosw

mov al, 't'; stosw

stosw

mov al, 'o'; stosw

stosw

mov al, ' '; stosw

stosw

mov al, 'u'; stosw

stosw

mov al, 's'; stosw

stosw

mov al, 'e'; stosw

stosw

mov al, ' '; stosw

stosw

mov al, 'y'; stosw

stosw
```

```
mov al, 'o'; stosw

stosw

mov al, 'u'; stosw

stosw

mov al, 'r'; stosw

stosw

mov al, ' '; stosw

stosw

mov al, 'e'; stosw

stosw

mov al, 'x'; stosw

stosw

mov al, 't'; stosw

stosw

mov al, 'r'; stosw

stosw

mov al, 'a'; stosw

stosw

mov al, ' '; stosw

stosw

mov al, 'l'; stosw
```

```
stosw

mov al, 'i'; stosw

stosw

mov al, 'f'; stosw

stosw

mov al, 'e'; stosw

stosw

mov al, '?'; stosw

stosw


mov di, 0x0728 ; Row 14, Col 20

mov ah, 0x3E ; Yellow text on CYAN

mov al, '['; stosw

stosw

mov al, '1'; stosw

stosw

mov al, ']'; stosw

stosw

mov al, ' '; stosw

stosw

mov al, 'Y'; stosw
```

```
stosw

mov al, 'e'; stosw

stosw

mov al, 's'; stosw

stosw


mov di, 0x07C4 ; Row 15, Col 50 (approx)

mov ah, 0x3E ; Yellow text on CYAN

mov al, '['; stosw

stosw

mov al, '2'; stosw

stosw

mov al, ']'; stosw

stosw

mov al, ' '; stosw

stosw

mov al, 'G'; stosw

stosw

mov al, 'i'; stosw

stosw

mov al, 'v'; stosw
```

```
    stosw
    mov al, 'e'; stosw
    stosw
    mov al, ' '; stosw
    stosw
    mov al, 'U'; stosw
    stosw
    mov al, 'p'; stosw
    stosw
    ret

clear_extra_life_prompt_sub:
    mov cx, 80      ; Clear 3 full rows
    mov ax, 0x3F20  ; Space char, White on CYAN attribute (match play area)

    mov di, 0x05A0  ; Start of row 11
    push cx
    rep stosw
    pop cx

    mov di, 0x06E0  ; Start of row 14
```

```
        push cx

        rep stosw

        pop cx



        mov di, 0x0780  ; Start of row 15

        mov cx, 80

        rep stosw



        ret



scroll_scenery:

        mov si,0x00a2

        mov di,0x00a0

fb2_scroll:

        mov cx,79

        repz

        movsw



        mov ax,0x3F20   ; Clean last character (Space, White on CYAN - to match play area)

        stosw
        lodsw


        cmp si,0x0fa2

        jnz fb2_scroll


fb5:    dec word [next]

        mov bx,[next]

        cmp bx,0x03

        ja fb6

        jne fb8


        in al,(0x40)

        and ax,0x0007

        add al,0x04

        mov [tall],ax

fb8:

        mov cx,[tall]

        or bx,bx

        mov dl,0xb0

        jz near fb7
```

```
    mov dl,0xdb

    cmp bx,0x03

    jb fb7

    mov dl,0xb1

fb7:

    mov di,0x013e

    mov ah,0x2a     ; Bright Green attribute for pipes on Dark Green (was 0x3a for Cyan BG)

    mov al,dl

fb9:   stosw

    add di,0x009e

    loop fb9



    mov al, 0xdf

    stosw



    add di, (0x009e * 6) + 10



    mov al,0xdf

    stosw

    add di,0x009e
```

```
fb10:

        mov al,dl

        stosw

        add di,0x009e

        cmp di,0x0f00

        jb fb10



        or bx,bx

        jnz fb6



        mov ax,[pipe]

        inc ax

        mov [pipe],ax



        mov cl,3

        shr ax,cl

        mov ah,0x37

        sub ah,al

        cmp ah,0x10

        ja fb11

        mov ah,0x10
```

```
fb11:   mov [next],ah

fb6:    ret




wait_frame:

        mov ah,0x00     ; Get current tick count (INT 1Ah, AH=00h)

        int 0x1a        ; DX:AX = tick count

fb14:   push dx         ; Save current tick count (low part in DX)

        mov ah,0x00

        int 0x1a        ; Get new tick count

        pop bx          ; Restore previous tick count to BX

        cmp bx,dx       ; Compare previous DX with current DX

        jz near fb14         ; Loop if tick count hasn't changed



        inc word [frame] ; Increment global frame counter



        cmp byte [invincibility_active], 1 ; Is invincibility active?

        jne invincibility_timer_done       ; If not, skip timer logic



        cmp word [invincibility_timer], 0  ; Timer reached zero?

        je deactivate_invincibility_now    ; If yes, deactivate
```

```
    dec word [invincibility_timer]     ; Decrement timer

    jmp invincibility_timer_done       ; Continue



    deactivate_invincibility_now:

    mov byte [invincibility_active], 0 ; Deactivate invincibility



    invincibility_timer_done:



    in al,(0x61)    ; Get speaker port status

    and al,0xfc     ; Turn speaker off (clear bits 0 and 1)

    out (0x61),al   ; Send to port

    ret



    db "OTG"        ; Signature



    db 0x55,0xaa    ; Boot signature (though not a boot sector)


clear_screen:

    mov ax, 0xb800

    mov es, ax      ; ES must point to video memory for stosw
```

```
xor di, di      ; Start at offset 0000h in video memory


; Top: 2 rows of Blue background

mov cx, 2 * 80  ; 2 rows * 80 columns/row

mov ax, 0x1F20  ; Attribute: White FG (F) on Blue BG (1), Char: Space (0x20)

rep stosw


; Middle: 18 rows of Cyan background

mov cx, 18 * 80 ; 18 rows

mov ax, 0x3F20  ; Attribute: White FG (F) on Cyan BG (3), Char: Space

rep stosw


; Thin line: 2 rows of Green background

mov cx, 2 * 80  ; 2 rows

mov ax, 0x2F20  ; Attribute: White FG (F) on Green BG (2), Char: Space

rep stosw


; Bottom: 3 rows of Brown background

mov cx, 3 * 80  ; 3 rows

mov ax, 0x6F20  ; Attribute: White FG (F) on Brown BG (6), Char: Space

rep stosw
```

```
        mov dx, 0x0000  ; Set cursor to top-left (row 0, col 0)

        mov bh, 0x00    ; Page 0

        mov ah, 0x02    ; Function to set cursor position

        int 0x10        ; BIOS video interrupt

        ret


display_status_messages:

    pusha               ; Save all general registers

    push es

    push ds


    mov ax, 0xb800          ; Video segment

    mov es, ax

    mov ds, ax


    mov di, (23*80*2) + (45*2)

    mov cx, 35

    mov ax, 0x3F20              ; Space char ' ' with White on CYAN

    cld

    rep stosw
```

```
cmp byte [invincibility_active], 1

je .display_active_duration_info



mov di, (23*80*2) + (57*2)

mov ah, 0x3E                ; Yellow text on CYAN attribute



mov al, 'I'; stosw

stosw

mov al, 'n'; stosw

stosw

mov al, 'v'; stosw

stosw

mov al, 'i'; stosw

stosw

mov al, 'n'; stosw

stosw

mov al, 'c'; stosw

stosw

mov al, 'i'; stosw

stosw
```

```
mov al, 'b'; stosw

stosw

mov al, 'i'; stosw

stosw

mov al, 'l'; stosw

stosw

mov al, 'i'; stosw

stosw

mov al, 't'; stosw

stosw

mov al, 'y'; stosw

stosw

mov al, ' '; stosw

stosw

mov al, '('; stosw

stosw

mov al, 'I'; stosw

stosw

mov al, ')'; stosw

stosw

mov al, ':'; stosw
```

```
    stosw

    mov al, ' '; stosw

    stosw



    mov ah, 0x3E

    mov al, [invincibility_uses_left]

    add al, '0'

    stosw

    jmp .status_display_done_final



.display_active_duration_info:

    mov di, (23*80*2) + (65*2)

    mov ah, 0x3A                ; Bright Green on CYAN attribute



    mov al, 'I'; stosw

    stosw

    mov al, 'n'; stosw

    stosw

    mov al, 'v'; stosw

    stosw

    mov al, 'i'; stosw
```

```
stosw

mov al, 'n'; stosw

stosw

mov al, 'c'; stosw

stosw

mov al, 'i'; stosw

stosw

mov al, 'b'; stosw

stosw

mov al, 'l'; stosw

stosw

mov al, 'e'; stosw

stosw

mov al, ':'; stosw

stosw

mov al, ' '; stosw

stosw


mov ax, [invincibility_timer]

xor dx, dx

mov bx, 18
```

```
    cmp ax, 0
    je .display_zero_seconds_now


    div bx


    cmp ax, 0
    jne .display_the_calculated_seconds
    cmp word [invincibility_timer], 0
    je .display_the_calculated_seconds
    mov al, 1
    jmp .convert_seconds_to_ascii


.display_zero_seconds_now:
    mov al, 0


.display_the_calculated_seconds:
.convert_seconds_to_ascii:
    add al, '0'
    mov ah, 0x3A
    stosw
    mov al, 's'
    mov ah, 0x3A
    stosw

.status_display_done_final:
    pop ds
    pop es
    popa
    ret



; --- Score Display Subroutine ---


; Display the player's score
show_current_score:
        ; Show current score
        mov di, 0x03C0  ; Row 8, centered
        mov ax, 0x3e59  ; 'Y' in yellow
        stosw
        mov al, 0x6f    ; 'o'
        stosw
```

```
mov al, 0x75    ; 'u'

stosw

mov al, 0x72    ; 'r'

stosw

mov al, 0x20    ; ' '

stosw

mov al, 0x53    ; 'S'

stosw

mov al, 0x63    ; 'c'

stosw

mov al, 0x6f    ; 'o'

stosw

mov al, 0x72    ; 'r'

stosw

mov al, 0x65    ; 'e'

stosw

mov al, 0x3a    ; ':'

stosw

mov al, 0x20    ; ' '

stosw
```

```
    ; Convert and display the score
    mov ax, [score]
    mov bx, 10      ; Divisor
    xor cx, cx      ; Digit counter

score_push_digits:
    xor dx, dx      ; Clear high word for division
    div bx          ; AX = AX/10, DX = remainder
    push dx         ; Save digit
    inc cx          ; Count digits
    or ax, ax       ; Check if quotient is 0
    jnz score_push_digits ; If not, continue

score_pop_and_display:
    pop ax          ; Get digit
    add al, '0'     ; Convert to ASCII
    mov ah, 0x30    ; White color
    stosw
    loop score_pop_and_display

    ret
```

```
; --- End of Score Display Subroutine ---


pipe:   equ 0x0fa0

score:  equ 0x0fa2

grav:   equ 0x0fa4

next:   equ 0x0fa6

bird:   equ 0x0fa8

tall:   equ 0x0faa

frame:  equ 0x0fac

user_choice: equu 0x0fae


extra_life_available: equ 0x0faf

saved_crash_di:     equ 0x0fb0

paused:  equ 0x0fb2


invincibility_uses_left: equ 0x0fb3

invincibility_active:    equ 0x0fb4

invincibility_timer:     equ 0x0fb5 ; word (ends 0x0fb6)

high_score:      dw 0      ; Variable to store the highest score achieved (2 bytes)
```

```
exit_to_dos:

    int 0x20    ; Call DOS terminate program function
```

VIII. How to Install

1. Download and Install NASM (Netwide Assembler)

Make sure NASM is installed on your system. You need it to convert the source code into a runnable file.

Download the .zip here: bit.ly/4kpwUa4

2. Get the Haribird's Tiny Wing Escape Source Code

Download or clone the Haribird source code from the official source.

3. Move the Folder to Drive C

For easier access, place the Haribird folder in Drive C.

4. Assemble the Source Code

Once everything is ready:

- Open the command prompt or terminal where NASM is available.
- Then, compile the source code using this command:

```
nasm fbird.asm -o fbird.com -l fbird.lst
```

This will generate the fbird.com file, which is the runnable version of the game.

IX. References / Credits

Upadhye, G. D., Shaikh, A. H. H., Jadhav, A., Matkar, A., & Bonde, A. (2023, June). Chess Game Using Assembly Language Programming. In *International Conference on Machine Learning, Deep Learning and Computational Intelligence for Wireless Communication* (pp. 593-604). Cham: Springer Nature Switzerland.

Kawash, J., & Collier, R. (2013, October). Using video game development to engage undergraduate students of assembly language programming. In *Proceedings of the 14th annual ACM SIGITE conference on Information technology education* (pp. 71-76).

Hyde, R. (2010). *The art of assembly language*. No Starch Press.

Goulding, T. (2010). An encryption system in assembly language: a game-like project for novice programmers. *ACM SIGCSE Bulletin*, *41*(4), 40-44.

Genrica. (n.d.). *Assembler (NASM) Installation & Usage Guide*. Genrica. https://genrica.com/html5CSS3/Assembler_NASM_Installation_Usage_Guide.aspx

X. Pictures during development