RIPHAH INTERNATIONAL UNIVERSITY

PROJECT REPORT

Snake Game (Console Based)

Course: Computer Organization and Assembly Language

Instructor: Mueed Ahmed

Student Name: Muhammad Abdullah Nadeem

Sap ID: 66156

Date: 08 December 2025

Contents:

1. Introduction

The program implements a classic "Snake Game" simulation in 8086 Assembly language. It demonstrates the mastery of array manipulation, direct hardware interaction, and non-blocking keyboard input handling[6]. The goal is to simulate a snake that moves continuously across the screen, allowing the user to control its direction while implementing "screen wrapping" logic (where the snake reappears on the opposite side if it hits a wall).

2. Problem Statement & Objectives

Problem Statement:

Design an assembly program that simulates a movement-based Snake game on a DOS system7. The system must render a moving snake composed of multiple segments, update its position in real-time based on user input (Arrow Keys), and handle boundary conditions by wrapping the coordinates around the screen edges without pausing execution8.

Objectives:

1. **Real-Time Rendering:** Render a moving snake using ASCII characters (specifically *) and handle the erasure of the tail to simulate movement[9].
2. **Array Management:** Manage the snake's body coordinates using a memory array (snake dw s_size dup(0)) to track the head and tail positions.
3. **Screen Wrapping:** Implement logic to detect when the snake reaches the screen edge (Right, Left, Top, Bottom) and transport it to the opposing side.
4. **Timing Control:** Use the System Timer (INT 1Ah) to control the speed of the game loop.
5. **Input Handling:** Implement non-blocking input to allow the user to change direction (Up, Down, Left, Right) dynamically[10].

3. Tools & Environment

- **Assembler:** MASM or TASM (Microsoft/Borland Assembler) [11]
- **Execution Environment:** Emu8086 Emulator [12]
- **Programming Language:** 8086 Assembly [13]
- **Interrupts Used:** BIOS Interrupt 10h (Video), BIOS Interrupt 16h (Keyboard), DOS Interrupt 21h (System), BIOS Interrupt 1Ah (Time)[14].
- **Editor:** Emu8086 Emulator[15].

4. Algorithm & Logic Flow

1. **Initialization:** Set up the Data Segment, define the snake array, and print the welcome/rules message[16].
2. **Setup Phase:** Clear the video screen and hide the blinking text cursor.
3. **Game Loop Start:** Begin the infinite loop that drives the game[17].
4. **Draw Phase:**
    - Draw the new Head (*) at the current coordinates.

    - Erase the old Tail (replace with) to create the illusion of movement[18].
5. **Timing Phase:**
    - Read the system clock (INT 1Ah).
    - Wait until the current clock tick matches the wait_time to control game speed.
6. **Input Phase:**
    - Check the keyboard buffer (Non-blocking)[19].
    - If an Arrow Key is pressed, update the cur_dir (Current Direction) variable.
    - If 'ESC' is pressed, exit the game.
7. **Update Phase (Move Snake):**
    - Shift the array values: Move each body part to the position of the part in front of it.
    - Update the Head coordinates based on cur_dir.
    - **Boundary Checking:** If the head coordinate exceeds screen limits (Row or Column), reset it to the opposite side (0 or Max)[20].
8. **Repeat:** Jump back to Step 3 (Game Loop).

5. Technical Implementation

5.1 Memory Variables

Variable	Type	Purpose
snake	DW Array	Stores the coordinates (row/col) of the snake's body parts.
s_size	EQU	Constant defining the length of the snake (set to 7).
cur_dir	DB	Stores the current movement direction (Left, Right, Up, Down).
wait_time	DW	Used for the timing loop to control game speed.
tail	DW	Temporarily stores the coordinates of the tail for erasure.
msg1 / msg2	String	Messages displayed at the start (Rules and Welcome).

5.2 Logic & Movement

Unlike the Pong game which used single variables for position, this project uses **Array Shifting**.

- **Body Movement:** To move the snake, we iterate through the snake array. Starting from the tail, every segment adopts the coordinates of the segment preceding it.
- **Wrapping Logic:**
    - If Column < 0 $\rightarrow$ Set Column = Max_Width.
    - If Column > Max_Width $\rightarrow$ Set Column = 0.
    - (Similar logic applies to Rows).

5.3 Interrupts Used

Interrupt	Register Settings	Functionality
INT 10h	AH=00h	Set Video Mode / Clear Screen.
INT 10h	AH=02h	Set Cursor Position (used before printing *). [21]
INT 10h	AH=09h	Write Character at cursor with attribute (Color). [22]
INT 10h	AH=01h	Set Cursor Type (Used to hide the blinking cursor).
INT 16h	AH=01h	Check keyboard status (Zero Flag) without stopping. [23]
INT 16h	AH=00h	Read key from buffer (scancode retrieval). [24]
INT 1Ah	AH=00h	Read System Time (Tick count) for game speed delay.
INT 21h	AH=09h	Display string terminated with $. [25]

6. Code Analysis

- **MAIN PROC:** The entry point. It displays the welcome rules (msg1), waits for a key press, clears the screen, and hides the cursor[26].
- **Game_Loop:** The central engine. It manages the timing (using INT 1Ah) and orchestrates the Drawing and Updating phases[27].
- **Check_for_key:** Uses INT 16h (AH=01h) to "peek" at the keyboard buffer. If a key is waiting, it reads it; otherwise, it jumps to the timing logic. This ensures the snake keeps moving even if the user isn't typing[28].
- **Move_Snake:** This is the most complex procedure.
    1. **Array Loop:** It uses a loop to shift the snake's body coordinates (snake[di] = snake[di-2]).
    2. **Direction Handling:** Compares cur_dir with constants (left, right, up, down).
    3. **Wrapping:** Checks ES segment data (es:[4ah] and es:[84h]) to determine screen boundaries and reset coordinates if necessary.

7. User Guide & Output

How to Run:

1. **Assemble:** Open snake.asm in Emu8086.
2. **Emulate:** Click the "Emulate" button.
3. **Run:** Click "Run" on the emulation window[29].

Controls:

- **Start:** Press any key at the welcome screen.
- **Movement:**

    - **Up Arrow:** Move Up
    - **Down Arrow:** Move Down
    - **Left Arrow:** Move Left
    - **Right Arrow:** Move Right
- **Exit:** Press **ESC** to quit the game[30].

Sample Output:

The screen starts with a "Rules" page. Upon starting, a snake (chain of asterisks *) appears. As the user presses arrow keys, the snake changes direction. When the snake hits a wall, it reappears on the opposite side of the screen.

8. Conclusion

The Snake Game project successfully demonstrates the capability of 8086 assembly to handle array-based data structures and real-time simulation[31]. By managing memory arrays for the snake's body, handling coordinate-based logic for screen wrapping, and utilizing BIOS interrupts for timing and rendering, the program achieves a fluid, interactive experience. This project provided deep insight into how early video games managed object states and hardware timing efficiently[32].