A logo with a bird and a triangle

AI-generated content may be incorrect.

Mamoona .

Project Report

**A screenshot of a computer

AI-generated content may be incorrect.**

**TIC TAC TOE**

**(Noughts and Crosses)** Another name

I have pushed it to github <https://github.com/Mamoonalatif/TicTacToe-Asm>

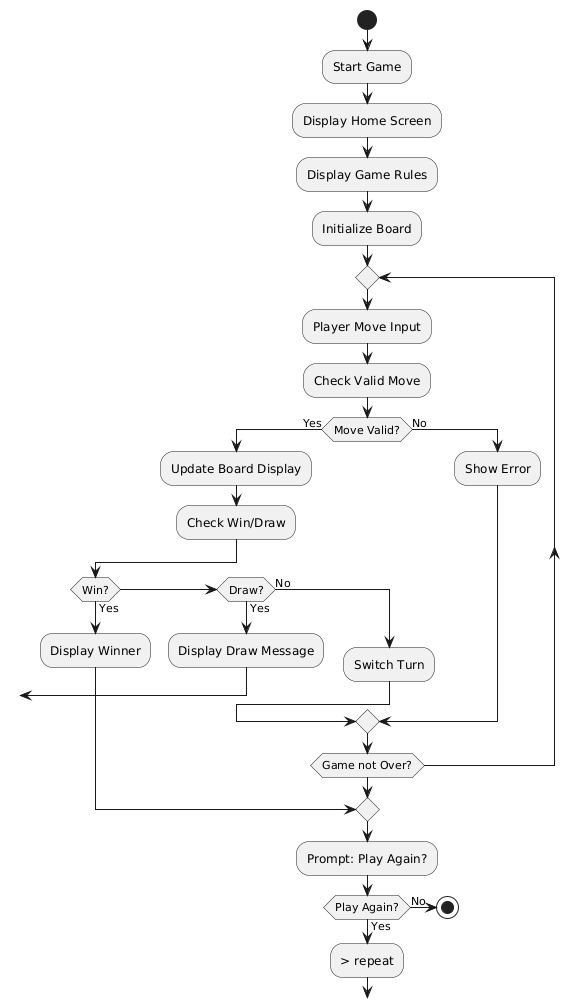
* **Introduction**

This project presents the implementation of the **Tic-Tac-Toe game** using **x86 16-bit Assembly Language**, designed to run under the **MS-DOS** environment. The goal of this project is to understand low-level hardware programming and system interrupts used in legacy systems.Assembly language offers direct access to system resources and hardware. This project utilizes BIOS and DOS interrupts to manage display, input, and logic, thereby showcasing how simple games can be developed at the hardware-near level.

* **Objectives**
* To design a two-player **Tic-Tac-Toe** game using x86 assembly.
* To understand the use of BIOS interrupts (INT 10h) for display control.
* To implement game logic, state management, and user input using low-level programming.
* To simulate decision-making, turn-taking, win condition checking, and replay functionality.
* **Tools & Technologies Used**

|  |  |
| --- | --- |
| Tool | Description |
| EMU8086 | Emulator for x86 assembly |
| DOSBox | DOS environment emulator |
| TASM/MASM | Turbo/Microsoft Assembler |
| Windows/Linux | Host OS for emulators |

* **System flow diagram**

****

* **Program Logic & Working**

**Game Initialization**

* Clears the screen.
* Displays home screen with ASCII-art style headers using BIOS interrupt 10h.
* Shows rules and instructions using string outputs via DOS interrupt 21h.

**Game Board Representation**

* 1D array of 9 bytes used as the game board.
* Indexed from 1 to 9 corresponding to cell positions.

**User Input**

* Uses INT 21h to get keyboard input.
* Validates whether the input is a digit between 1-9 and the position is not already taken.

**Win Condition Check**

* 8 combinations are checked:
  + 3 rows
  + 3 columns
  + 2 diagonals
* Checks if same player's mark ('X' or 'O') occupies all three.

**Turn Switching**

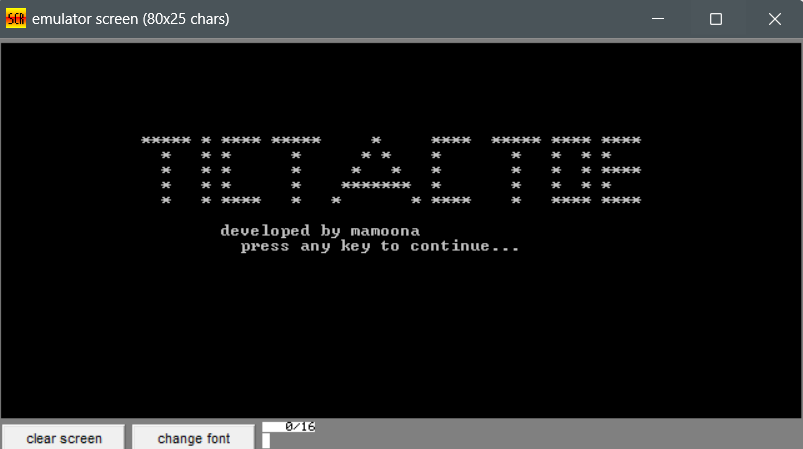
* Toggles between Player 1 and Player 2 by switching ASCII values '1' and '2'.

**Replay Option**

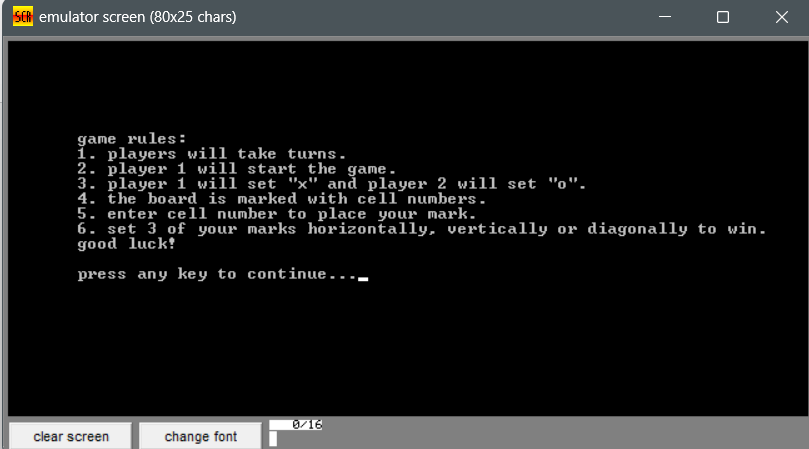
* After win or draw, user is prompted to press 'y' or 'n'.
* Based on input, game either restarts or terminates.

**Output (TEXT FILE OF CODE IS ATTACHED)**

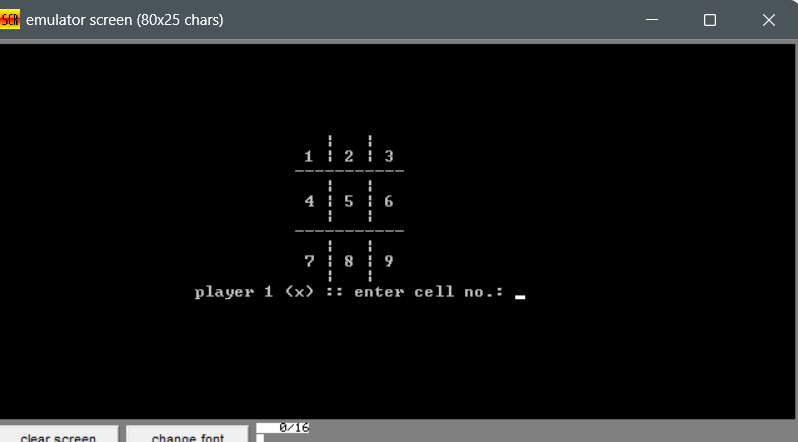
HomePage



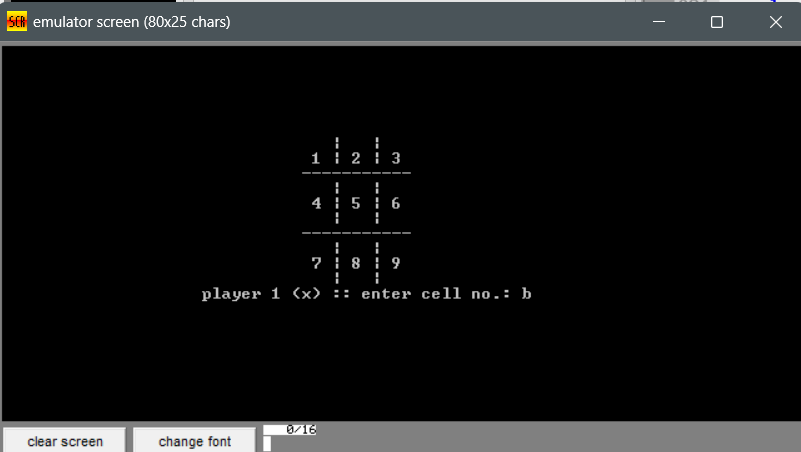
RulesPage



Gameboard

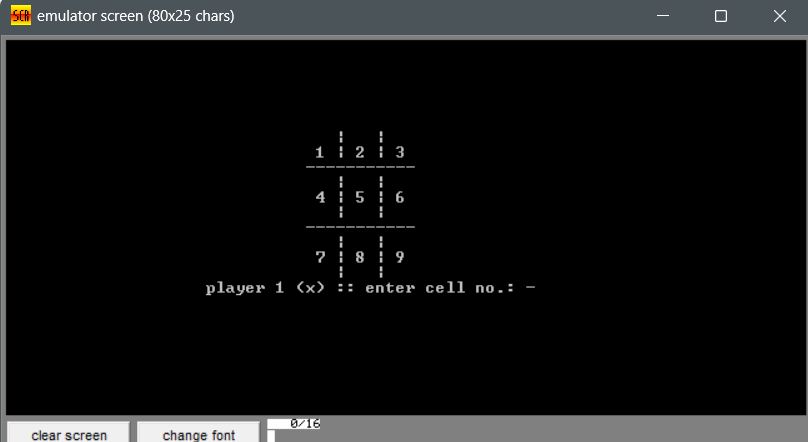


Typing invalid input (alphabets)



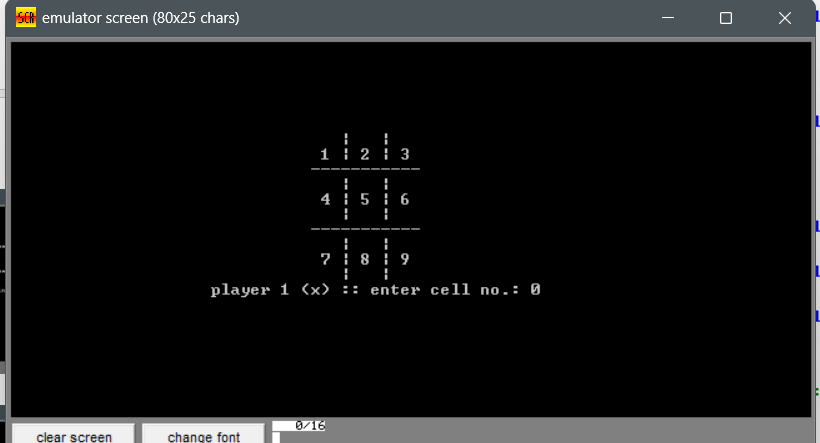


Typing invalid input (specialcharacter)





Typing invalid input (0)



A screenshot of a computer

AI-generated content may be incorrect.

When one user wins



When game draws

A screenshot of a computer

AI-generated content may be incorrect.

Play again

A screenshot of a computer

AI-generated content may be incorrect.

**Conclusion**

This project effectively demonstrates how a classic game like **Tic-Tac-Toe** can be implemented at a low-level using **Assembly language**, utilizing:

* Memory management
* Screen cursor control
* Character I/O using interrupts

It improves our understanding of the architecture, data handling, and programming near the hardware level. The logic is optimized, and user interaction is intuitive for a console-based game.

# Refrences

Pearson. (2010). *Assembly Language for x86 Processors.* K.R.