

**PROJECT REPORT**

**Project Title**

**“TIC TAC TOE”**

*Group Members*

|  |  |
| --- | --- |
| **Name** | **Enrollment** |
| 1. Haris Bashir | **02-134221-041** |
| 2. Emad Tariq | **02-134221-068** |
| 3.Ammarah Ali | **02-134221-077** |

**Submitted to: MISS Ambreen**

**Table of Contents**

* **Introduction: 3**
* **Project Objectives 3**
* **Problem Statement 3**
* **Methodology 3**
* **Data storage segment 3**
* **Scope 3**
* **Explanation 3**
* **Conclusion 3**
* **FlowChart 4**
* **Code 4**
* **Output 9**

**INTRODUCTION:-**

This is an assembly code for a Tic-Tac-Toe game, also known as Noughts and Crosses. The game is a simple strategy game that can be played by two players. The goal of the game is to get three of the same symbols in a row, either horizontally, vertically, or diagonally. The symbols used in the game are "x" and "o", where one player uses "x" and the other player uses "o".

**OBJECTIVE:-**

The goal of the game is for players to position their marks so that they make a continuous line of three cells vertically, horizontally, or diagonally. An opponent can prevent a win by blocking the completion of the opponent's line.

**PROBLEM STATEMENT:-**

The game algorithm should be such that the circuit cannot be beaten (in the hardest level in case of multiple levels), and in the worst case the game ends in a draw.

**METHODOLOGY:-**

Tic Tac Toe is a game played on a 3x3 grid. Two players take turns, one as X and the other as O. They place their marks in empty cells on the grid. Moves must be validated to ensure they are valid. After each move, the game checks for a win by looking for three marks in a row horizontally, vertically, or diagonally. If a win is found, the player who achieved it is declared the winner. Players continue alternating turns until a win occurs. After the game ends, players can choose to play again or conclude the game.

**DATA STORAGE SEGMENTATION IN CODE:-**

The code uses various segments for data storage. The "data" segment is used to store the game board, player information, and messages that are displayed during the game. The "stack" segment is used to store temporary data used during subroutine calls. The "extra" segment is not used in this code.

**SCOPE:-**

The Tic-Tac-Toe game is changed into a 2 player game on a computer- one plays with X's and the other as O's. Both these players play on the client's side. The server program evaluates the game and declares the winners or if the game is a draw This game is implemented in client server manner 2 players will be playing at the client side-one playing as 'X' and the other playing as "O". If it is win or lose server declares game over.

**EXPLANATION:-**

The code utilizes various subroutines to handle different tasks. The "set\_game\_pointer" subroutine is used to initialize the game board. The "clear\_screen" subroutine is used to clear the screen before displaying the game board. The "print" subroutine is used to display messages on the screen. The "read\_keyboard" subroutine is used to read player input. The "update\_draw" subroutine is used to update the game board with the player's symbol. The "check" subroutine is used to check for a win. The "change\_player" subroutine is used to switch between players. The "check\_line" subroutine is used to check lines on the game board for a win.

**CONCLUSION:-**

The Tic Tac Toe game is most familiar among all the age groups. Intelligence can be a property of any purpose-driven decision maker. The basic idea has been suggested many times. An algorithm of playing Tic Tac Toe has been presented and tested that works in efficient way. Overall the system works without any bugs.

**FLOWCHART:-**



**CODE:-**

data segment

new\_line db 13, 10, "$"

game\_draw db "1|2|3", 13, 10

db "4|5|6", 13, 10

db "7|8|9", 13, 10, "$"

game\_pointer db 9 DUP(?)

win\_flag db 0

player db "0$"

game\_over\_message db "GAME OVER", 13, 10, "$"

game\_start\_message db " (TIC TAC TOE) by 068 077 041", 13, 10, "$"

player\_message db "PLAYER $"

win\_message db " WIN!$"

type\_message db "TYPE A POSITION: $"

ends

stack segment

dw 128 dup(?)

ends

extra segment

ends

code segment

start:

; set segment registers

mov ax, data

mov ds, ax

mov ax, extra

mov es, ax

; game start

call set\_game\_pointer

main\_loop:

call clear\_screen

lea dx, game\_start\_message

call print

lea dx, new\_line

call print

lea dx, player\_message

call print

lea dx, player

call print

lea dx, new\_line

call print

lea dx, game\_draw

call print

lea dx, new\_line

call print

lea dx, type\_message

call print

; read draw position

call read\_keyboard

; calculate draw position

sub al, 49

mov bh, 0

mov bl, al

call update\_draw

call check

; check if game ends

cmp win\_flag, 1

je game\_over

call change\_player

jmp main\_loop

change\_player:

lea si, player

xor ds:[si], 1

ret

update\_draw:

mov bl, game\_pointer[bx]

mov bh, 0

lea si, player

cmp ds:[si], "0"

je draw\_x

cmp ds:[si], "1"

je draw\_o

draw\_x:

mov cl, "x"

jmp update

draw\_o:

mov cl, "o"

jmp update

update:

mov ds:[bx], cl

ret

check:

call check\_line

ret

check\_line:

mov cx, 0

check\_line\_loop:

cmp cx, 0

je first\_line

cmp cx, 1

je second\_line

cmp cx, 2

je third\_line

call check\_column

ret

first\_line:

mov si, 0

jmp do\_check\_line

second\_line:

mov si, 3

jmp do\_check\_line

third\_line:

mov si, 6

jmp do\_check\_line

do\_check\_line:

inc cx

mov bh, 0

mov bl, game\_pointer[si]

mov al, ds:[bx]

cmp al, "\_"

je check\_line\_loop

inc si

mov bl, game\_pointer[si]

cmp al, ds:[bx]

jne check\_line\_loop

inc si

mov bl, game\_pointer[si]

cmp al, ds:[bx]

jne check\_line\_loop

mov win\_flag, 1

ret

check\_column:

mov cx, 0

check\_column\_loop:

cmp cx, 0

je first\_column

cmp cx, 1

je second\_column

cmp cx, 2

je third\_column

call check\_diagonal

ret

first\_column:

mov si, 0

jmp do\_check\_column

second\_column:

mov si, 1

jmp do\_check\_column

third\_column:

mov si, 2

jmp do\_check\_column

do\_check\_column:

inc cx

mov bh, 0

mov bl, game\_pointer[si]

mov al, ds:[bx]

cmp al, "\_"

je check\_column\_loop

add si, 3

mov bl, game\_pointer[si]

cmp al, ds:[bx]

jne check\_column\_loop

add si, 3

mov bl, game\_pointer[si]

cmp al, ds:[bx]

jne check\_column\_loop

mov win\_flag, 1

ret

check\_diagonal:

mov cx, 0

check\_diagonal\_loop:

cmp cx, 0

je first\_diagonal

cmp cx, 1

je second\_diagonal

ret

first\_diagonal:

mov si, 0

mov dx, 4 ;tamanho do pulo

jmp do\_check\_diagonal

second\_diagonal:

mov si, 2

mov dx, 2

jmp do\_check\_diagonal

do\_check\_diagonal:

inc cx

mov bh, 0

mov bl, game\_pointer[si]

mov al, ds:[bx]

cmp al, "\_"

je check\_diagonal\_loop

add si, dx

mov bl, game\_pointer[si]

cmp al, ds:[bx]

jne check\_diagonal\_loop

add si, dx

mov bl, game\_pointer[si]

cmp al, ds:[bx]

jne check\_diagonal\_loop

mov win\_flag, 1

ret

game\_over:

call clear\_screen

lea dx, game\_start\_message

call print

lea dx, new\_line

call print

lea dx, game\_draw

call print

lea dx, new\_line

call print

lea dx, game\_over\_message

call print

lea dx, player\_message

call print

lea dx, player

call print

lea dx, win\_message

call print

jmp fim

set\_game\_pointer:

lea si, game\_draw

lea bx, game\_pointer

mov cx, 9

loop\_1:

cmp cx, 6

je add\_1

cmp cx, 3

je add\_1

jmp add\_2

add\_1:

add si, 1

jmp add\_2

add\_2:

mov ds:[bx], si

add si, 2

inc bx

loop loop\_1

ret

print: ; print dx content

mov ah, 9

int 21h

ret

clear\_screen: ; get and set video mode

mov ah, 0fh

int 10h

mov ah, 0

int 10h

ret

read\_keyboard: ; read keybord and return content in ah

mov ah, 1

int 21h

ret

fim:

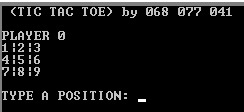
jmp fim

code ends

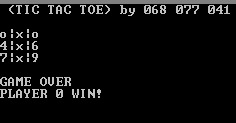
end start

**OUTPUT:-**

1)Main Screen of the game one user choose ‘x’, and the other chooses ‘o’.



2)X win’s the match



3)O wins the Match

