Introduction

*TIC-TAC-TOE is the most popular and easiest game. It is a two players (X and O) game, where each player taking turns marks the spaces in a 3×3 grid. I have developed this game in TurboC graphics with interfacing keyboard event and have also shown some interactive animation. This tip will be helpful for anyone who wants to develop this type of game using TurboC graphics. So let’s get started…*

Requirements

To run and develop TurboC graphics code, you need the following:

1. TurboC compiler to execute and run the code.
2. Graphic mode functions require a graphics monitor and adapter card such as CGA, EGA and VGA.
3. To start graphics programming, you need two files which are *GRAPHICS.H* and *GRAPHICS.LIB*. Especially, these files are provided as part of TurboC compile.

## Start Game

Graphics Environment Starts with the graphics initializes function initgraph() that loads a graphics driver from disk, then putting the system into graphics mode. It accepts three arguments, one is

graphics driver(gd), the second is graphics mode(gm) and the third is path for bgi directory that specifies the directory path where initgraph looks for graphics drivers (\*.BGI). Here, I have set path c:\\turboc3\\bgi.

Next, I have played an animation text that is write a string “TIC TAC TOE” where each character animatedly appears one by one. To animatedly write each letter, a function is called corresponding to the letter. After completing the write “TIC TAC TOE” string two animations are sparking on two sides of the string.

void start(){

int gdrive=DETECT,gmode;

initgraph(&gdrive,&gmode,"c:\\turboc3\\bgi");

delay(2000);

T(50,50); delay(100);

I(); delay(100);

C(160,50);delay(100);

T(300,50); delay(100);

A();delay(100);

C(455,50);delay(100);

T(160,200);delay(100);

O();delay(100);

C(250,200);delay(100);

E();delay(200);

for(int q=0;q<5;q++){

SCR(80,235);

SCR(470,235);

CIR(235);

}

closegraph();

}

To write animated letter T a function T() is calling that draws a horizontal line and then a vertical line by the circle() function, where circle() function draws a circle in x,y coordinate with radius r. If we increase the coordinate value of x and y is constant of parameter of circle() function, then it will create a horizontal line. Same thing can be applied to draw a vertical line.

void T(int t1,int t2){

for(int i=0;i<70;i++){

setcolor(2);

circle(t1+i,t2,3);

delay(10);

}

for(i=0;i<70;i++){

setcolor(2);

circle(t1+35,t2+i,3);

delay(10);

}

}

## Menu Selection

Now you need to select your options that is 1 for Start the game and 2 for Quit the game. By entering your choice by the standard input/output method, the game will go for the further process. This input is taken by the simple standard I/O function scanf(). According to the input value, the corresponding operation will be performed by the switch-case statement.

printf("\n1.Start The Game");

printf("\n2.Quit The Game");

printf("\nEnter your choice(1-2) : ");

scanf("%d",&ch);

switch(ch)

{

case 1:

...

Case 2:

...

}

## Enter Player

Enter the player name corresponding to the player X and Player O. Player name will be captured as string by the gets() function. Where gets() function takes the string input by the standard I/O method.

printf("\nEnter the name of the player playing for \'X\': ");

fflush(stdin);

gets(name\_X);

printf("\nEnter the name of the player playing for \'O\': ");

fflush(stdin);

gets(name\_O)

Display Board

In this game, two players alternate placing stones of their own color on an m×n board. Board is drawn by the simple printf() function.

void Board()

{

int i,j;

clrscr();

printf("\n\t\t\t\tTIC TAC TOE BOARD");

printf("\n\t\t\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\n\n\n");

printf("\n\t\t\t 1\t 2\t 3");

for(i=1;i<=3;i++)

{

printf("\n \t\t\t \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

printf("\n \t\t\tº\t º\t º\t º");

printf("\n\t\t%d\t",i);

for(j=1;j<=3;j++)

{

if(pos\_for\_X[i][j]==1)

{

printf(" X");

printf(" ");

}

else if(pos\_for\_O[i][j]==1)

{

printf(" O");

printf(" ");

}

else

{

printf(" ");

continue;

}

}

printf("\n\t\t\tº\t º\t º\t º");

}

printf("\n\t\t\t------------------------------");

Player\_win();

}

Player X Turn

The first player who shall be designated "X" needs to enter the row and column number. If column or row number is out of range of [1,3] like row<1 or row > 3 or column<1 or column>3, then it would be a wrong position or else grid position array pos\_marked[row][col] will be assigned 1 according to the row and column number.

Hide   Copy Code

void PlayerX()

{

int row,col;

if(win==1)

return;

printf("\nEnter the row no. : ");

fflush(stdin);

scanf("%d",&row);

printf("Enter the column no. : ");

fflush(stdin);

scanf("%d",&col);

if(pos\_marked[row][col]==1 || row<1 || row>3 || col<1 || col>3)

{

printf("\nWRONG POSITION!! Press any key.....");

wrong\_X=1;

getch();

Board();

}

else

{

pos\_for\_X[row][col]=1;

pos\_marked[row][col]=1;

Board();

}

}

Player O Turn

The second player who shall be designated "O" needs to enter the row and column number. If PlayerO enters row or column position that is not valid, that means value is out of [1,3] like row<1 or row>3 or column<1 or column>3 then it will be warning through an message “WRONG POSITION!” or else it will be marked the grid position array pos\_marked[row][col] assign 1 according to the value of row and column value.

Hide   Copy Code

void PlayerO()

{

int row,col;

if(win==1)

return;

printf("\nEnter the row no. : ");

scanf("%d",&row);

printf("Enter the column no. : ");

scanf("%d",&col);

if(pos\_marked[row][col]==1 || row<1 || row>3 || col<1 || col>3)

{

printf("\nWRONG POSITION!! Press any key....");

wrong\_O=1;

getch();

Board();

}

else

{

pos\_for\_O[row][col]=1;

pos\_marked[row][col]=1;

Board();

}

}

Wrong Turn

Wrong turn can be made up of two ways:

1. If player gives an entry that cell is already fill-up. That means pos\_marked[row][col]=1 then position will be wrong.
2. If player gives the row or column number that is out of 3x3 grid. To check if row<1 or row>3 or col<1 or col>1 then wrong position will happen.

Hide   Copy Code

if(pos\_marked[row][col]==1 || row<1 || row>3 || col<1 || col>3)

{

printf("\nWRONG POSITION!! Press any key.....");

wrong\_X=1;

getch();

Board();

}

Result Check

After each turn of a player, check a calculation that the pos\_marked[i][j]=1 where I and j is increment value from 1 to 3 and the value of check(chk) is increased or else it would be continue.

void check()

{

int i,j;

for(i=1;i<=3;i++)

{

for(j=1;j<=3;j++)

{

if(pos\_marked[i][j]==1)

chk++;

else

continue;

}

}

}

Win Calculation

Now, it's time to check the result. The player who succeeds in placing three respective marks in a horizontal, vertical, or diagonal row wins the game.

To check the horizontal line for the player x if the pos\_for\_X[i][1]=1 and pos\_for\_X[i][2]=1 and pos\_for\_X[i][3]=1 then playerX will win where I increment value from 1 to 3. In the same way, I have checked the vertical line and diagonal line. And the code segment is as follows:

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

void Player\_win()

{

int i;

for(i=1;i<=3;i++)

{

if(pos\_for\_X[i][1]==1 && pos\_for\_X[i][2]==1 && pos\_for\_X[i][3]==1)

{

win=1;

printf("\n\nRESULT: %s wins!!",name\_X);

printf("\nPress any key............");

return;

}

}

for(i=1;i<=3;i++)

{

if(pos\_for\_X[1][i]==1 && pos\_for\_X[2][i]==1 && pos\_for\_X[3][i]==1)

{

win=1;

printf("\n\nRESULT: %s wins!!",name\_X);

printf("\nPress any key............");

return;

}

}

if(pos\_for\_X[1][1]==1 && pos\_for\_X[2][2]==1 && pos\_for\_X[3][3]==1)

{

win=1;

printf("\n\nRESULTL: %s wins!!",name\_X);

printf("\nPress any key......");

return;

}

else if(pos\_for\_X[1][3]==1 && pos\_for\_X[2][2]==1 && pos\_for\_X[3][1]==1)

{

win=1;

printf("\n\nRESULT: %s wins!!",name\_X);

printf("\nPress any key.....");

return;

}

for(i=1;i<=3;i++)

{

if(pos\_for\_O[i][1]==1 && pos\_for\_O[i][2]==1 && pos\_for\_O[i][3]==1)

{

win=1;

printf("\n\nRESULT: %s wins!!",name\_O);

printf("\nPress any key.....");

return;

}

}

for(i=1;i<=3;i++)

{

if(pos\_for\_O[1][i]==1 && pos\_for\_O[2][i]==1 && pos\_for\_O[3][i]==1)

{

win=1;

printf("\n\nRESULT: %s wins!!",name\_O);

printf("\nPress any key.....");

return;

}

}

if(pos\_for\_O[1][1]==1 && pos\_for\_O[2][2]==1 && pos\_for\_O[3][3]==1)

{

win=1;

printf("\n\nRESULT: %s wins!!",name\_O);

printf("\nPress any key.....");

return;

}

else if(pos\_for\_O[1][3]==1 && pos\_for\_O[2][2]==1 && pos\_for\_O[3][1]==1)

{

win=1;

printf("\n\nRESULT: %s wins!!",name\_O);

printf("\nPress any key.....");

return;

}

}

Match Draws

Game will be draws by the following condition:

1. If no result is returned from the board against the player X or O, that means if chk variable value is 9.
2. If no result is returned for the win variable, that means if win variable is not equal to 1.

Hide   Copy Code

if(chk==9)

{

printf("\n\t\t\tMATCH DRAWS");

printf("\nPress any key....");

break;

}

…

if(win!=1)

{

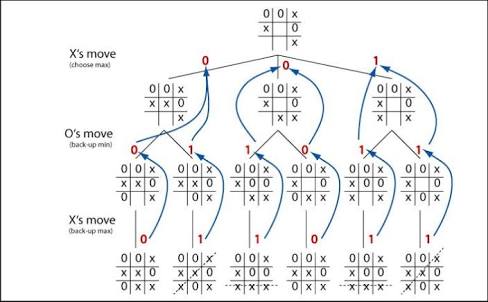
printf("\n\t\t\tMATCH DRAWS!!");

printf("\nPress any key.......");

}

## Quit Game

After finishing the game, the following good bye message will be shown. This simple text is present by the printf() function.



#### Minimax :

To check whether or not the current move is better than the best move we take the help of **minimax()**function which will consider all the possible ways the game can go and returns the best value for that move, assuming the opponent also plays optimally  
The code for the maximizer and minimizer in the **minimax()** function is similar to **findBestMove()** , the only difference is, instead of returning a move, it will return a value. Here is the pseudocode

**function** minimax(board, depth, isMaximizingPlayer):

**if** current board state is a terminal state :

**return** value of the board

**if** isMaximizingPlayer :

bestVal = -INFINITY

**for each** move in board :

value = minimax(board, depth+1, false)

bestVal = max( bestVal, value)

**return** bestVal

**else** :

bestVal = +INFINITY

**for each** move in board :

value = minimax(board, depth+1, true)

bestVal = min( bestVal, value)

**return** bestVal