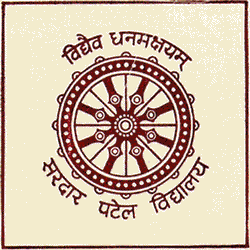
**Project Report**

**on**

**Intelligent Tic-Tac-Toe**







**Submitted By**

Abhimanyu Bhatnagar

Class: XII D

**Under the Guidance of**

Ms. Angel Panesar

Department of Computer Science

Sardar Patel Vidyalaya

Lodhi Estate, New Delhi 110003

**CERTIFICATE**

This is to certify that Abhimanyu Bhatnagar of Class XII D has prepared the report on the Project entitled “Intelligent Tic-Tac-Toe”. The report is the result of his efforts & endeavors. The report is found worthy of acceptance as final project report for the subject Computer Science of Class XII. He has prepared the report under my guidance.

Ms. Angel Panesar

Department of Computer Science

Sardar Patel Vidyalaya

Lodhi Estate, New Delhi 110003

**DECLARATION**

I hereby declare that the project work entitled “Intelligent Tic-Tac-Toe”, submitted to Department of Computer Science, Sardar Patel Vidyalaya, Lodhi Estate, New Delhi 110003 is prepared by me. The project work is result of my personal efforts.

Abhimanyu Bhatnagar

Class: XII D

**ACKNOWLEDGEMENT**

I would like to express a deep sense of thanks & gratitude to my project guide Ms. Angel Panesar for guiding me immensely through the course of the project. She always evinced keen interest in my work. Her constructive advice & constant motivation has been responsible for the successful completion of this project.

Abhimanyu Bhatnagar

Class: XII D

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| S. No. | Topic | Page No. |
| 1. | Header files used | 5 |
| 2. | Classes implemented | 6 |
| 2. | Working Description | 7 |
| 3. | Project Code | 11 |
| 4. | Output Screens | 41 |
| 5. | Conclusion | 49 |
| 6. | Bibliography | 50 |

**HEADER FILES USED**

The following header files have been used:

* iostream.h
* conio.h
* stdlib.h
* process.h
* fstream.h
* stdio.h
* string.h
* iomanip.h
* dos.h
* time.h

**CLASSES IMPLEMENTED**

The program implements the following classes:

* Class Matrix: The class Matrix contains the data grid of the game and the methods required to manipulate the data grid and provide intelligence to compute the computer's moves
* Class User: This class User is used to manage the data and historical record of a user.
* Class History: This class is used to manage the historical data of all users. It contains file handling methods for storing user data in a file and manipulating it.

**WORKING DESCRIPTION**

This project implements a game of Tic-Tac-Toe in which the computer plays against a human player. An attempt has been made to make the computer play intelligently so that it cannot be easily beaten. However, the program is not designed to be unbeatable and the human player can win in some rare scenarios.

The project implements Reactive Artificial Intelligence, a Rules-based Artificial Intelligence, in which the software is built to contain rules that provide it the intelligence to react to current inputs. This is different from Machine Learning based AI where the software progressively learns as it plays or gains experience.

In this project, a Reactive AI software has been developed by providing the rules that allow it to compute potential moves for itself and its opponent and decide the most optimal move that it can play in a given situation. In this sense, it is similar to IBM’s Deep Blue, the Reactive AI system that had beaten chess GM & former world champion Gary Kasparov in 1997.

The class Matrix implements the game logic, both for the basic rules as well as the intelligence to play against a human player. The 9 square grid of the Tic-Tac-Toe game is represented as a 2-dimentional 3X3 array, which is henceforth referred as Game Status Matrix. Squares which have been marked by the human player are internally represented by 1. Squares which have been marked by the computer are represented internally by -1. Squares which are not yet marked by either player are represented by 0. At the start of the game, the Game Status Matrix is initialized to all zeroes.

The method main() drives the game. It first randomly chooses whether the computer or human player will have the first move. It then instantiates an object of class Matrix which in turn creates the Game Status Matrix and initializes it to all zeroes. Next, main() draws the game grid on screen. It then runs a loop from 0 to 8, representing the 9 possible moves that can be made by both players together. Turns of the computer & the human player are allocated alternately, starting with the player who was randomly chosen. If it is the human player’s move, the computer asks the player to enter the row & column number of the grid that he/she chooses to mark. If it is the computer’s move, the program computes the best possible move depending on the Game Status Matrix. Thereafter, the game grid is re-drawn on the screen, showing ‘X’ for the human player’s moves and ‘O’ for the computer’s moves. Thereafter, each row, column and diagonal is checked to determine if it contains three marks by the same player i.e. if the sum of all values in it is +3 (for human player) or -3 (for computer). If such a row, column or diagonal is found, the corresponding player is declared a winner and the game ends. Otherwise the loop is repeated for the next move. If all 9 moves are completed without either player winning, the game is declared a draw.

The method drawfig() in class Matrix is used to draw the current status of the game on screen. It first clears the screen to remove the previously drawn grid and then prints out the grid represented by the Game Status Matrix. It prints ‘X’ for the human player’s moves (at positions where the Game Status Matrix contains 1), ‘O’ for the computer’s moves (at positions where the Game Status Matrix contains -1) and blanks where no move has yet been made (0s in Game Status Matrix).

The method conv() in class Matrix takes an integer input. This input can have values 1, -1 or 0, which are the internal representations of the human player’s move, computer’s move or no move respectively. The method returns the character ‘X’, ‘O’ or blank (‘ `) which are the respective external representations of the move.

The method rowsum() in class Matrix takes as input an integer row\_num. It computes and returns the sum of all values in row number row\_num in the Game Status Matrix.

The method columnsum() in class Matrix takes as an integer column\_num. It computes and returns the sum of all values in column number column\_num in the Game Status Matrix.

The method diag1() in class Matrix computes and returns the sum of all values in the diagonal from top left to bottom right in the Game Status Matrix.

The method diag2() in class Matrix computes and returns the sum of all values in the diagonal from bottom left to top right in the Game Status Matrix.

The method **compute\_move()** in class Matrix provides the intelligence in computing the computer’s next move. It takes as input an integer, move number, and computes the next move for the computer for the prevailing status of the Game Status Matrix. It applies the following steps:

* Basic attack: It checks if there is any row, column or diagonal with a sum -2. This means that there are 2 marks by the computer in that row, column or diagonal. If so, the 3rd position in the row, column, or diagonal is chosen as the computer’s next move and the corresponding index is returned.
* Basic defense: It checks if there is any row, column or diagonal with a sum +2. This means that there are 2 marks by the human player in that row, column or diagonal. If so, the 3rd position in the row, column, or diagonal is chosen as the computer’s next move to thwart the opponent from winning on the next move. The index of the chosen position is returned.
* Diagonal defense: If the opponent occupies two ends of a diagonal after 3 moves in the game, a non-diagonal position is chosen & returned. This is to beat some special situations not covered in look forward attack/defense.
* Look forward attack: The look forward method is called to simulate every move possible in the current game situation and to check if the simulated move leads to 2 rows, columns or diagonals having a sum of -2, i.e. having 2 marks each by the computer. If such a move is found, it will lead to the computer’s win after 3 moves. This position is hence chosen as the computer’s next move and its index is returned.
* Look forward defense: The look forward method is called to simulate every move possible in the current game situation and to check if the simulated move leads to 2 rows, columns or diagonals having a sum of +2, i.e. having 2 marks each by the human player after his/her next move. If such a move is found, it will lead to the human player’s win after 4 moves. So this position is hence chosen as the computer’s next move to thwart the possibility and its index is returned.
* If the move is either the 1st or 2nd move, a random vacant corner or central position is chosen and returned. This is done since a corner position or central is advantageous in initial moves.
* If none of the above steps leads to choice of a move, a vacant position is chosen at random as the next move & returned.

The method **look\_forward()** in class Matrix is used to simulate the game status for every potential next move that either the computer or the human player can next make. It takes as input an integer which is -1 for looking forward the computer’s potential next moves and 1 for the human player's moves. It checks all possible moves that can be made by the player. As explained above, if any potential move will result in a situation where the player gets 2 marks in 2 rows, columns or diagonals, the function returns the index for the move. Otherwise it returns -1.

The class User is used to represent the data for a user and the methods to initialize, update and access this data.

The class History is used to manage the historical record of all users. It contains methods for file-handling that read the historical game records of all users from a data file, display the historical data on screen, update the current player’s data on his/her request and update the player’s historical game record with the result of the current game.

**PROJECT CODE**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Title: Intelligent Tic-Tac-Toe

Description: Program to implement the game Tic-Tac-Toe using Reactive Artificial Intelligence so that the computer can play the game intelligently against a human player.

Author: Abhimanyu Bhatnagar

Class: XII-D

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream.h>

#include <conio.h>

#include <stdlib.h>

#include <process.h>

#include <fstream.h>

#include <stdio.h>

#include <string.h>

#include <iomanip.h>

#include <dos.h>

#include <time.h>

// The class Matrix contains the data grid and

// the methods required to manipulate it and compute

// the computer's moves

class Matrix

{

int grid[3][3];

char conv(int y);

int look\_forward(int player\_val);

public:

Matrix();

void setGridPoint(int row, int col, int value);

int getGridPoint(int row, int col);

int diag2();

int diag1();

int columnsum(int column\_num);

int rowsum(int row\_num);

void drawfig();

int compute\_move(int move\_count);

};

// The class User is used to store historical

// results of a user

class User

{

int serial\_num;

char name[100];

int num\_games;

int num\_wins;

int num\_draws;

char time\_last\_played[100];

public:

User();

User(int,char\*,int,int,int,char\*);

int get\_serial\_num();

char\* get\_name();

void set\_name(char\*);

int get\_num\_games();

void set\_num\_games(int);

int get\_num\_wins();

void set\_num\_wins(int);

int get\_num\_draws();

void set\_num\_draws(int);

char\* get\_time\_last\_played();

void set\_time\_last\_played(char\*);

void display();

};

// The class History is used to manage the historical

// dataset of all users. It contains methods to handle

// files for storing user data and manipulating them.

class History

{

User current\_user;

int new\_user;

void display\_header();

char\* History::currentDateTime();

void offer\_update();

public:

int find\_current\_user();

int update\_current\_user\_result(int);

int display();

};

// Basic constuctor

User::User()

{

serial\_num = 0;

num\_games = 0;

num\_wins = 0;

num\_draws = 0;

strcpy(name,"");

strcpy(time\_last\_played,"");

}

// Constructor with initialization values of all

// data members

User::User(int s, char\* n, int ng, int nw, int nd, char\* lp)

{

serial\_num = s;

num\_games = ng;

num\_wins = nw;

num\_draws = nd;

strcpy(name,n);

strcpy(time\_last\_played,lp);

}

// Accessor method to retrieve serial number

int User::get\_serial\_num()

{

return serial\_num;

}

// Accessor method to retrieve name

char\* User::get\_name()

{

return name;

}

// Mutator method to update name

void User::set\_name(char \*n)

{

strcpy(name,n);

}

// Accessor method to retrieve number of games played

int User::get\_num\_games()

{

return num\_games;

}

// Mutator method to update the number of games played

void User::set\_num\_games(int ng)

{

num\_games = ng;

}

// Accessor method to retrieve number of wins

int User::get\_num\_wins()

{

return num\_wins;

}

// Mutator method to update the number of wins

void User::set\_num\_wins(int nw)

{

num\_wins = nw;

}

// Accessor method to retrieve number of games drawn

int User::get\_num\_draws()

{

return num\_draws;

}

// Mutator method to update the number of games drawn

void User::set\_num\_draws(int nd)

{

num\_draws = nd;

}

// Accessor method to retrieve time\_last\_played

char\* User::get\_time\_last\_played()

{

return time\_last\_played;

}

// Mutator method to update time\_last\_played

void User::set\_time\_last\_played(char \*lp)

{

strcpy(time\_last\_played,lp);

}

// Method to display a user

void User::display()

{

cout.setf(ios::left);

cout << setw(5) << serial\_num;

cout << setw(26) << name;

cout << setw(7) << num\_games;

cout << setw(5) << num\_wins;

cout << setw(8) << num\_games-num\_wins-num\_draws;

cout << setw(7) << num\_draws;

cout << setw(20) << time\_last\_played << endl;

cout.setf(ios::right);

}

// Method used to identify the current player. The

// player is shown the history of all players and

// asked for his/her serial number if present in

// the history list. If the player provides a serial

// number, the that user is chosen. Otherwise the

// player is asked to provide his/her name and a

// new user is created

int History::find\_current\_user()

{

clrscr();

int max\_serial\_num = 0, current\_serial\_num, user\_count = 0;

User hist\_user;

fstream fhist;

fhist.open("tic\_hist.dat",ios::in|ios::binary);

if (!fhist)

{

// Create the data file if it does not exist

fhist.open("tic\_hist.dat",ios::out|ios::in|ios::binary);

}

// display\_header();

// Display the list of all players

while (!fhist.eof())

{

fhist.read((char \*) &hist\_user, sizeof(User));

if (fhist.eof())

{

// The latest read encountered eof. Hence break.

break;

}

if (user\_count % 5 == 0)

{

if (user\_count!=0) {

cout << endl << "Press any key to continue" << endl;

getch();

}

clrscr();

display\_header();

}

user\_count++;

hist\_user.display();

if (hist\_user.get\_serial\_num() > max\_serial\_num)

{

max\_serial\_num = hist\_user.get\_serial\_num();

}

}

if (user\_count)

{

cout << "\nIf your name is on the list, please enter your sequence number." << endl;

cout << "Otherwise enter 0" << endl;

cin >> current\_serial\_num;

}

else

{

cout << "Empty list" << endl << endl;

current\_serial\_num = 0;

}

if (current\_serial\_num > 0 && current\_serial\_num <= max\_serial\_num)

{

fhist.clear();

fhist.seekg(0);

while (!fhist.eof())

{

fhist.read((char \*) &hist\_user, sizeof(User));

if (fhist.eof())

{

break;

}

if (hist\_user.get\_serial\_num() == current\_serial\_num)

{

// Found a matching user. Set that to current user/

current\_user = hist\_user;

offer\_update();

new\_user = 0;

fhist.close();

return 0;

}

}

fhist.close();

}

// Create a new user

char name[256];

current\_serial\_num = max\_serial\_num + 1;

cout << "Please enter your name" << endl;

gets(name);

name[25] = '\0'; // Truncate the name to 25 characters

current\_user = User(current\_serial\_num, name, 0, 0, 0, currentDateTime());

new\_user = 1;

return 0;

}

// Method to allow use to update his/her name or reset

// his/her record

void History::offer\_update()

{

char update\_choice = '1';

while ('1' == update\_choice || '2' == update\_choice)

{

cout << "If you want to correct your name, enter 1" << endl;

cout << "If you want to reset your record, enter 2" << endl;

cout << "If you want to play, enter any other key" << endl;

cin >> update\_choice;

if ('1' == update\_choice)

{

char new\_name[256];

cout << "Please enter your name" << endl;

gets(new\_name);

new\_name[25] = '\0'; // Truncate the name to 25 characters

current\_user.set\_name(new\_name);

cout << "Your name has been updated" << endl;

} else if ('2' == update\_choice)

{

current\_user.set\_num\_games(0);

current\_user.set\_num\_wins(0);

current\_user.set\_num\_draws(0);

current\_user.set\_time\_last\_played("");

cout << "Your record has been reset" << endl;

}

}

}

// Update records of the current user with the game

// result and then update the history data file.

int History::update\_current\_user\_result(int result)

{

fstream fhist, ftemp;

User hist\_user;

current\_user.set\_time\_last\_played(currentDateTime());

current\_user.set\_num\_games(current\_user.get\_num\_games()+1);

if (1==result)

{

// User won

current\_user.set\_num\_wins(current\_user.get\_num\_wins()+1);

}

if (2==result)

{

// Game was drawn

current\_user.set\_num\_draws(current\_user.get\_num\_draws()+1);

}

fhist.open("tic\_hist.dat",ios::in|ios::binary); // Open the hsitory data file

ftemp.open("tic\_hist.tmp",ios::out|ios::binary); // Open a temporary file

if (!fhist || !ftemp)

{

cout << "Unable to open history file" << endl;

return 1;

}

while (!fhist.eof())

{

fhist.read((char \*) &hist\_user, sizeof(User));

if (fhist.eof())

{

// The latest read encountered eof. Hence break.

break;

}

if (hist\_user.get\_serial\_num() == current\_user.get\_serial\_num())

{

// Store the updated record of the current

// user instead of the historical record

ftemp.write((char \*) &current\_user, sizeof(User));

}

else

{

// Store the historical record unchanged

// since it doesn't belong to current user

ftemp.write((char \*) &hist\_user, sizeof(User));

}

}

if (new\_user)

{

// This is a new user. So write his/her

// record after all historical records

// have been written

ftemp.write((char \*) &current\_user, sizeof(User));

}

fhist.close();

ftemp.close();

rename("tic\_hist.dat","tic\_hist.del"); // Rename historical data file as .del (deleted) file

rename("tic\_hist.tmp","tic\_hist.dat"); // Rename temp file as historical data file

remove("tic\_hist.del"); // Now actually delete del file. This way we won't use historical file even if program crashes midway

return 0;

}

// Method to return currentDate & time

// Uses time(), localtime() and strftime() from time.h

char\* History::currentDateTime()

{

time\_t now = time(0);

struct tm tstruct;

char buf[100];

tstruct = \*localtime(&now);

strftime(buf,sizeof(buf),"%Y-%m-%d %X", &tstruct);

return buf;

}

// Display header of historical table

void History::display\_header()

{

cout << setw(44) << "History of Players" << endl;

for (int i=0;i<77;i++)

{

cout << "-";

}

cout << endl;

cout.setf(ios::left);

cout << setw(5) << "Seq";

cout << setw(26) << "Name";

cout << setw(7) << "Num";

cout << setw(5) << "Num";

cout << setw(8) << "Num";

cout << setw(7) << "Num";

cout << setw(20) << "Last played" << endl;

cout << setw(5) << "no.";

cout << setw(26) << "";

cout << setw(7) << "games";

cout << setw(5) << "wins";

cout << setw(8) << "losses";

cout << setw(7) << "draws" << endl;

for (i=0;i<77;i++)

{

cout << "-";

}

cout << endl;

cout.setf(ios::right);

}

// Display the historical data of all players present

// in historical data file

int History::display()

{

User hist\_user;

fstream fhist;

int count = 0;

clrscr();

fhist.open("tic\_hist.dat",ios::in|ios::binary);

if (!fhist)

{

cout << "Unable to open history file" << endl;

return 1;

}

display\_header();

while (!fhist.eof())

{

fhist.read((char \*) &hist\_user, sizeof(User));

if (fhist.eof())

{

// The latest read encountered eof. Hence break.

break;

}

if (count % 5 == 0)

{

if (count!=0) {

cout << endl << "Press any key to continue" << endl;

getch();

}

clrscr();

display\_header();

}

count++;

hist\_user.display();

}

fhist.close();

return 0;

}

// Constructor for Matrix. It initializes the

// game grid by storing 0s in all positions

Matrix::Matrix()

{

for(int i=0;i<3;i++)

{

for(int j=0;j<3;j++)

{

grid[i][j]=0;

}

}

}

// Mutator method to update a point on the grid.

void Matrix::setGridPoint(int row, int col, int value)

{

grid[row][col] = value;

}

// Accessor method to retrieve a point on the grid

int Matrix::getGridPoint(int row, int col)

{

return (grid[row][col]);

}

// Draw the grid representing the current status of the

// game on the screen

void Matrix::drawfig()

{

clrscr();

int n=179;

char z=n;

cout<<"\n\n\n\t\t\t\t"<<z<<"\t"<<z<<endl;

cout<<"\t\t\t "<<conv(grid[0][0])<<" "<<z<<" "<<conv(grid[0][1])<<" "<<z<<" "<<conv(grid[0][2])<<" " <<endl;

cout<<"\t\t\t \_\_\_\_\_\_"<<z<<"\_\_\_\_\_\_\_"<<z<<"\_\_\_\_\_\_"<<endl;

cout<<"\t\t\t\t"<<z<<"\t"<<z<<endl;

cout<<"\t\t\t "<<conv(grid[1][0])<<" "<<z<<" "<<conv(grid[1][1])<<" "<<z<<" "<<conv(grid[1][2])<<" " <<endl;

cout<<"\t\t\t \_\_\_\_\_\_"<<z<<"\_\_\_\_\_\_\_"<<z<<"\_\_\_\_\_\_"<<endl;

cout<<"\t\t\t\t"<<z<<"\t"<<z<<endl;

cout<<"\t\t\t "<<conv(grid[2][0])<<" "<<z<<" "<<conv(grid[2][1])<<" "<<z<<" "<<conv(grid[2][2])<<" " <<endl;

cout<<"\t\t\t\t"<<z<<"\t"<<z<<endl;

return ;

}

// Convert the internal representation of game status

// to the external representation of 'X', 'O' or blank

char Matrix::conv(int y)

{

char z;

if(0==y)

{

z=' ';

}

if(1==y)

{

z='X';

}

if(-1==y)

{

z='O';

}

return z;

}

// Compute the sum of all values in the row row\_num

int Matrix::rowsum(int row\_num)

{

int sum;

sum=0;

for(int i=0;i<3;i++)

{

sum=grid[row\_num][i]+sum;

}

return sum;

}

// Compute the sum of all values in the

// column column\_num

int Matrix::columnsum(int column\_num)

{

int sum;

sum=0;

for(int i=0;i<3;i++)

{

sum=grid[i][column\_num]+sum;

}

return sum;

}

// Compute the sum of all values in the diagonal

// from top left to bottom right

int Matrix::diag1()

{

int sum=0;

for(int i=0;i<3;i++)

{

sum += grid[i][i];

}

return sum;

}

// Compute the sum of all values in the diagonal

// from bottom left to top right

int Matrix::diag2()

{

int sum=0;

for(int i=0;i<3;i++)

{

int j=2-i;

sum += grid[i][j];

}

return sum;

}

// Compute the computer's next move

int Matrix::compute\_move(int move\_count)

{

int row, column, move\_index, look\_forward\_val;

if(diag1()==-2)

{

for(int i=0;i<3;i++)

{

if(grid[i][i]==0)

{

row=i;

column=i;

cout << "Attacking " << row << ", " << column << endl;

return row\*3+column;

}

}

}

if(diag2()==-2)

{

for(int i=0;i<3;i++)

{

int j=2-i;

if(grid[i][j]==0)

{

row=i;

column=j;

cout << "Attacking " << row << ", " << column << endl;

return row\*3+column;

}

}

}

for(int t=0;t<3;t++)

{

// Attack loop

if(-2==rowsum(t))

{

for(int i=0;i<3;i++)

{

if(grid[t][i]==0)

{

row=t;

column=i;

cout << "Attacking " << row << ", " << column << endl;

return row\*3+column;

}

}

}

if(-2==columnsum(t))

{

for(int i=0;i<3;i++)

{

if(grid[i][t]==0)

{

column=t;

row=i;

cout << "Attacking " << row << ", " << column << endl;

return row\*3+column;

}

}

}

}

if(diag1()==2)

{

for(int i=0;i<3;i++)

{

if(grid[i][i]==0)

{

row=i;

column=i;

cout << "Defending " << row << ", " << column << endl;

return row\*3+column;

}

}

}

if(diag2()==2)

{

for(int i=0;i<3;i++)

{

int j=2-i;

if(grid[i][j]==0)

{

row=i;

column=j;

cout << "Defending " << row << ", " << column << endl;

return row\*3+column;

}

}

}

for(t=0;t<3;t++)

{

// defence loop

if(2==rowsum(t))

{

for(int i=0;i<3;i++)

{

if(grid[t][i]==0)

{

row=t;

column=i;

cout << "Defending " << row << ", " << column << endl;

return row\*3+column;

}

}

}

if(2==columnsum(t))

{

for(int i=0;i<3;i++)

{

if(grid[i][t]==0)

{

column=t;

row=i;

cout << "Defending " << row << ", " << column << endl;

return row\*3+column;

}

}

}

}

// Special case of opponent occupying opposite corners

if(move\_count==3 && ((grid[0][0]==1 && grid[2][2]==1)||(grid[2][0]==1 && grid[0][2]==1)))

{

for(int j=1;j<8;j+=2)

{

row=j/3;

column=j%3;

if(grid[row][column]==0)

{

cout << "Defending against diagonal" << endl;

return j;

}

}

}

look\_forward\_val = look\_forward(-1);

if (look\_forward\_val > -1)

{

cout << "Look forward attacking" << endl;

return look\_forward\_val;

}

look\_forward\_val = look\_forward(1);

if (look\_forward\_val > -1)

{

cout << "Look forward defending" << endl;

return look\_forward\_val;

}

if (move\_count <= 2)

{

// In initial moves, choose a position on diagonal

if (0==grid[1][1]) {

// If center is free, choose that

cout << "Choosing center" << endl;

return 4;

}

do

{

row = random (3);

if (random(2)==0)

{

column = row;

} else {

column = 2-row;

}

}

while (grid[row][column]!=0);

cout << "Choosing diagonal position" << endl;

return row\*3+column;

}

do

{

row=random(3);

column=random(3);

} while ( grid[row][column] != 0);

cout << "Random choice " << row << ", " << column << endl;

move\_index = row\*3+column;

return move\_index;

}

// This function takes as input the game matrix and

// integer player\_val which is -1 if it is the

// computer's move and 1 for the human player's move.

// It checks checks all possible moves that can be made

// by the player. If any potential move will results in

// a situation where the player gets 2 marks in 2 rows,

// columns or diagonals, it returns the index for the

// move. Otherwise it returns -1.

int Matrix::look\_forward(int player\_val)

{

int row, column, double\_count;

for (row = 0; row < 3; row++)

{

for (column = 0; column < 3; column++)

{

if (grid[row][column]==0)

{

// Simulate a move on the square if it is unmarked

grid[row][column] = player\_val;

double\_count = 0;

double\_count+= (rowsum(0)==2\*player\_val)?1:0;

double\_count+= (rowsum(1)==2\*player\_val)?1:0;

double\_count+= (rowsum(2)==2\*player\_val)?1:0;

double\_count+= (columnsum(0)==2\*player\_val)?1:0;

double\_count+= (columnsum(1)==2\*player\_val)?1:0;

double\_count+= (columnsum(2)==2\*player\_val)?1:0;

double\_count+= (diag1()==2\*player\_val)?1:0;

double\_count+= (diag2()==2\*player\_val)?1:0;

grid[row][column] = 0; // Reverse the simulated move

if (double\_count >= 2)

{

return 3\*row+column;

}

}

}

}

return -1;

}

// Method to display the game banner

void display\_banner()

{

// Flash the banner four times

for (int k=0;k<4;k++)

{

clrscr();

cout << endl << endl;

cout << " ###" << endl;

cout << " # # # ##### ###### # # # #### ###### # # #####" << endl;

cout << " # ## # # # # # # # # # ## # #" << endl;

cout << " # # # # # ##### # # # # ##### # # # #" << endl;

cout << " # # # # # # # # # # ### # # # # #" << endl;

cout << " # # ## # # # # # # # # # ## #" << endl;

cout << " ### # # # ###### ###### ###### # #### ###### # # #" << endl << endl;

cout << "####### ####### #######" << endl;

cout << " # # #### # ## #### # #### ######" << endl;

cout << " # # # # # # # # # # # # #" << endl;

cout << " # # # #### # # # # #### # # # ####" << endl;

cout << " # # # # ###### # # # # #" << endl;

cout << " # # # # # # # # # # # # #" << endl;

cout << " # # #### # # # #### # #### ######" << endl;

delay(1000); // Wait for 1000 ms = 1 sec before clearing the screen

clrscr();

delay(300); // Wait for 300 ms with blank screen before drawing the banner again, thus giving flashing effect

}

}

int main()

{

unsigned int seedval;

time\_t t;

seedval = (unsigned) time(&t);

srand(seedval);

int i, j, row, column, move\_count, count;

int first\_mover = rand()%2; // Randomly choose who has the 1st move

int game\_end = 0, game\_result = 0;

// Display the game banner

display\_banner();

// Display player history & identify current player

History hist;

clrscr();

hist.find\_current\_user();

clrscr();

// Instantiate an object of class Matrix

// The squares on which neither player has marked a

// move are represented by 0. The squares marked by

// the human player are represented internally

// as 1. The squares marked by the computer are

// represented internally as -1

Matrix gameMatrix;

gameMatrix.drawfig(); // Draw the initial grid on screen

for(move\_count=0;move\_count<9;move\_count++)

{

if (move\_count%2 == first\_mover)

{

// Human player's move. Collect input.

count = 0;

do

{

if (count >0)

{

cout <<"Invalid move. Please enter again" << endl;

}

count += 1;

cout<<"Enter the row number \n";

cin>>row;

if(row == 15)

{

// Hack to exit midway. Not published.

exit(0);

}

row -= 1;

cout<<"Enter the column number \n";

cin>>column;

column -= 1;

} while ( row>=3 || \

column>=3 || \

row<0 || \

column<0 || \

gameMatrix.getGridPoint(row,column)!=0);

gameMatrix.setGridPoint(row,column,1); // Mark the move location with +1

}

else

{

// Computer's move. Calculate the move.

int move\_index = gameMatrix.compute\_move(move\_count);

row = move\_index/3;

column = move\_index%3;

gameMatrix.setGridPoint(row,column,-1); // Mark the move location with -1

}

// getch();

gameMatrix.drawfig();

// Check if either player has won the game. If yes announce the result

for(int t=0;t<3;t++)

{

if(3==gameMatrix.rowsum(t))

{

cout<<"You win"<<endl;

game\_end=1;

game\_result = 1;

break;

}

if(-3==gameMatrix.rowsum(t))

{

cout<<"Computer wins"<<endl;

game\_end=1;

break;

}

if(3==gameMatrix.columnsum(t))

{

cout<<"You win"<<endl;

game\_end=1;

game\_result = 1;

break;

}

if(-3==gameMatrix.columnsum(t))

{

cout<<"Computer wins"<<endl;

game\_end=1;

break;

}

}

if(3==gameMatrix.diag1())

{

cout<<"You win"<<endl;

game\_end=1;

game\_result = 1;

break;

}

if(-3==gameMatrix.diag1())

{

cout<<"Computer wins"<<endl;

game\_end=1;

break;

}

if(3==gameMatrix.diag2())

{

cout<<"You win"<<endl;

game\_end=1;

game\_result = 1;

break;

}

if(-3==gameMatrix.diag2())

{

cout<<"Computer wins"<<endl;

game\_end=1;

break;

}

if(game\_end==1)

{

break;

}

}

// If all moves are over without either player

// winning, the game is a draw

if(0==game\_end)

{

cout<<"The game is a draw"<<endl;

game\_result = 2;

}

// Update the historical data file with the result

// of the current game

hist.update\_current\_user\_result(game\_result);

cout << endl << "Press any key to continue" << endl;

getch();

// Display updated historical results

hist.display();

cout << endl << "Press any key to continue" << endl;

getch();

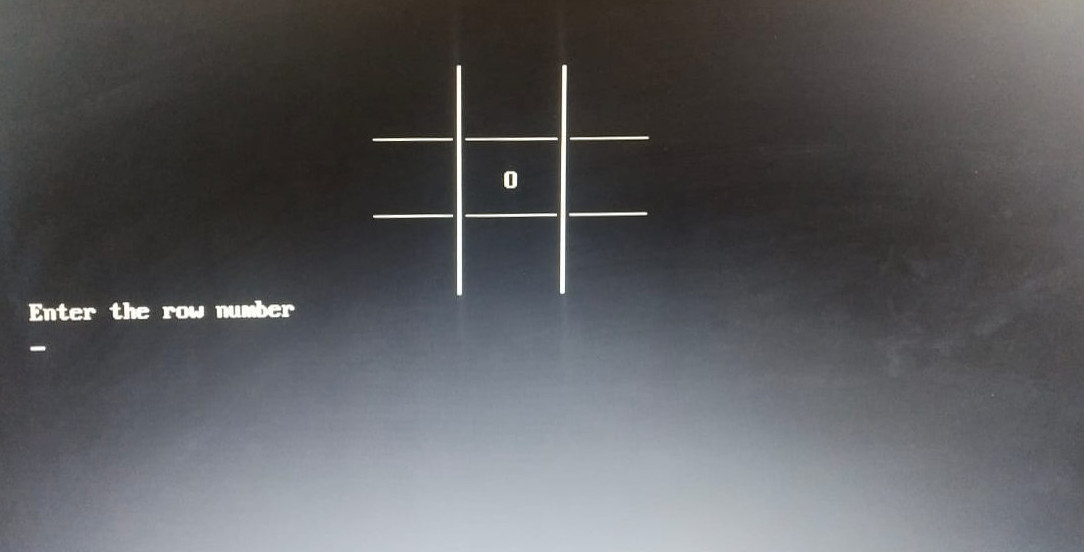
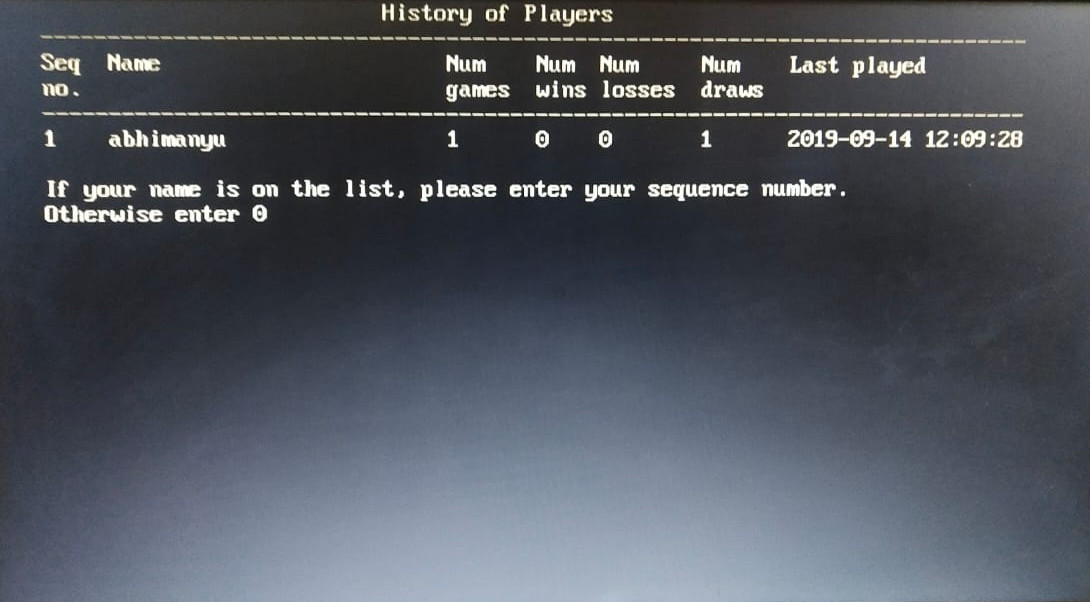
return 0;

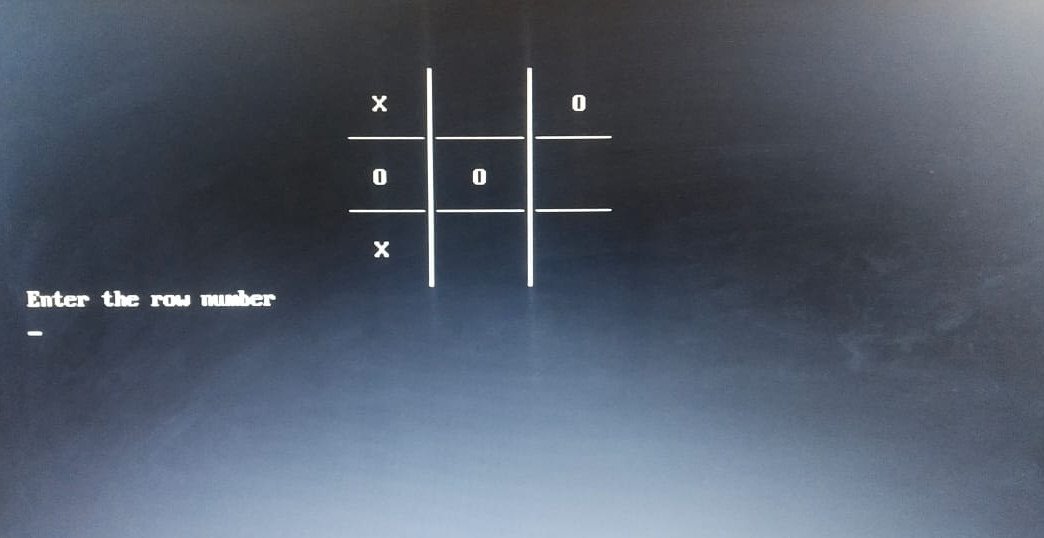
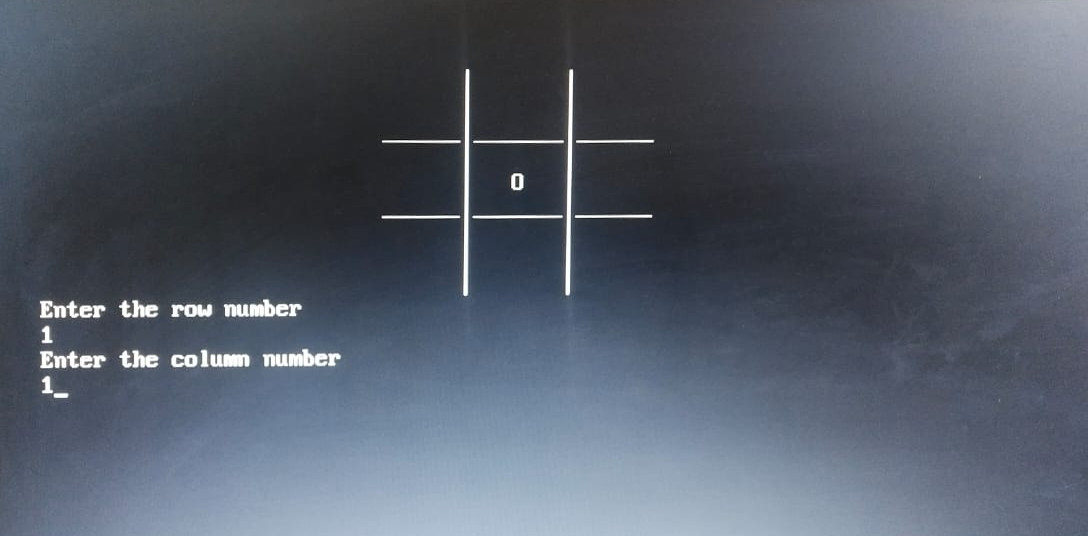
}

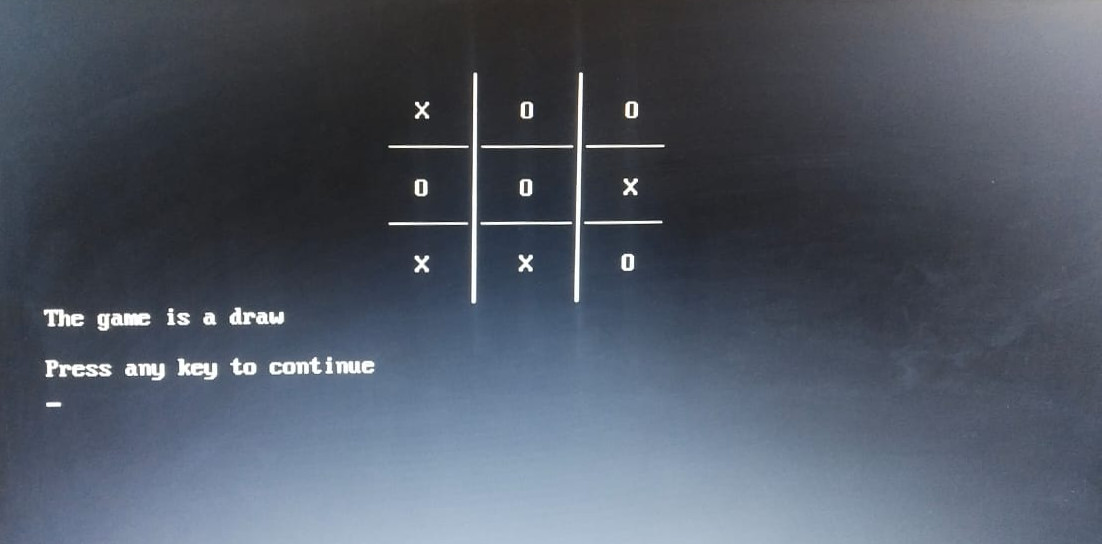
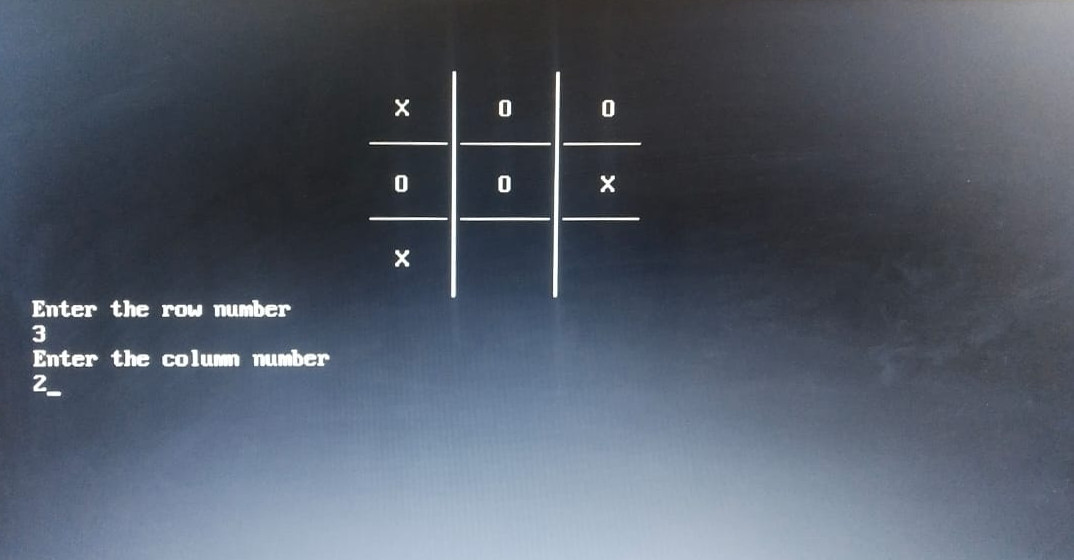
**OUTPUT SCREENS**

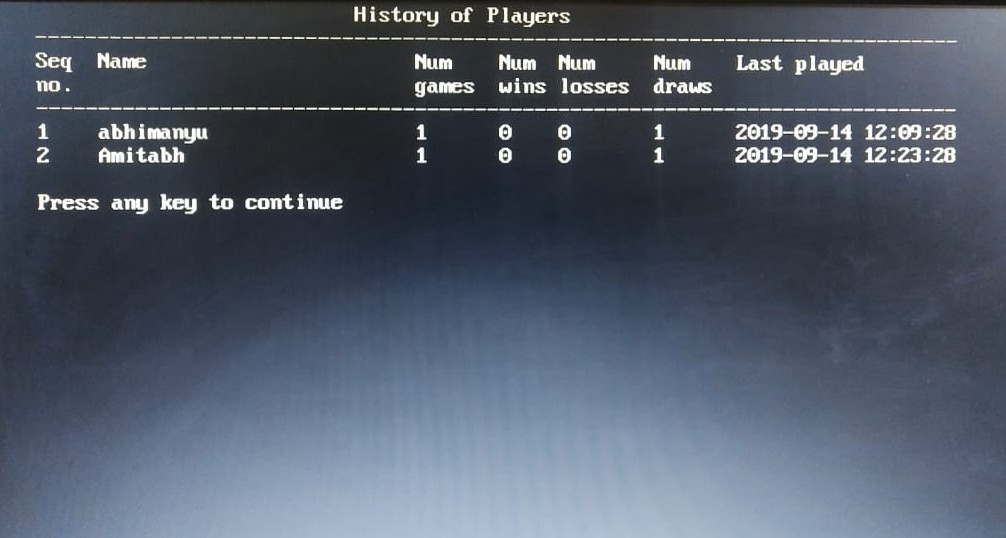
The following screen shots show one execution of the game.



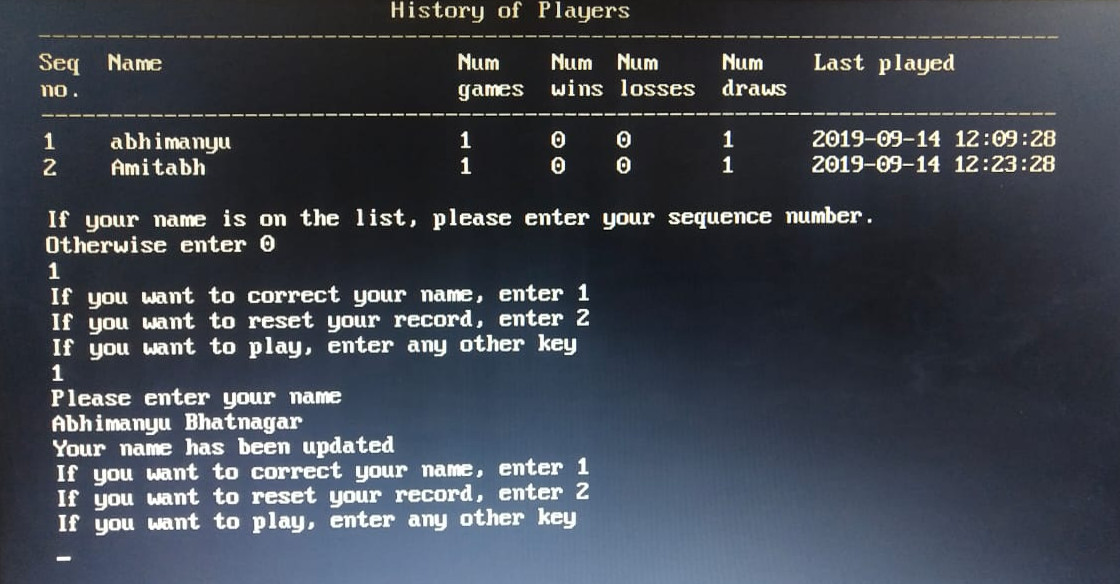


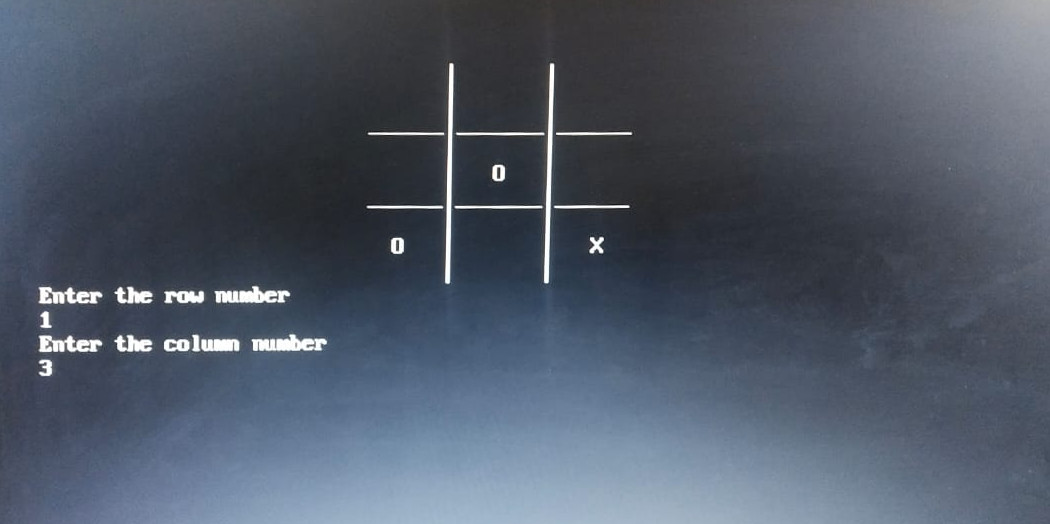
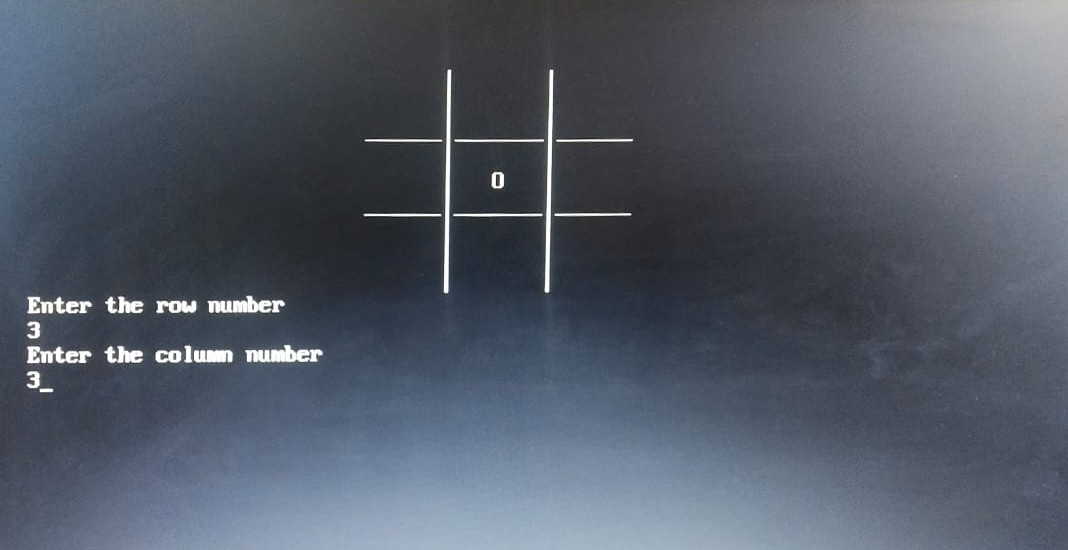


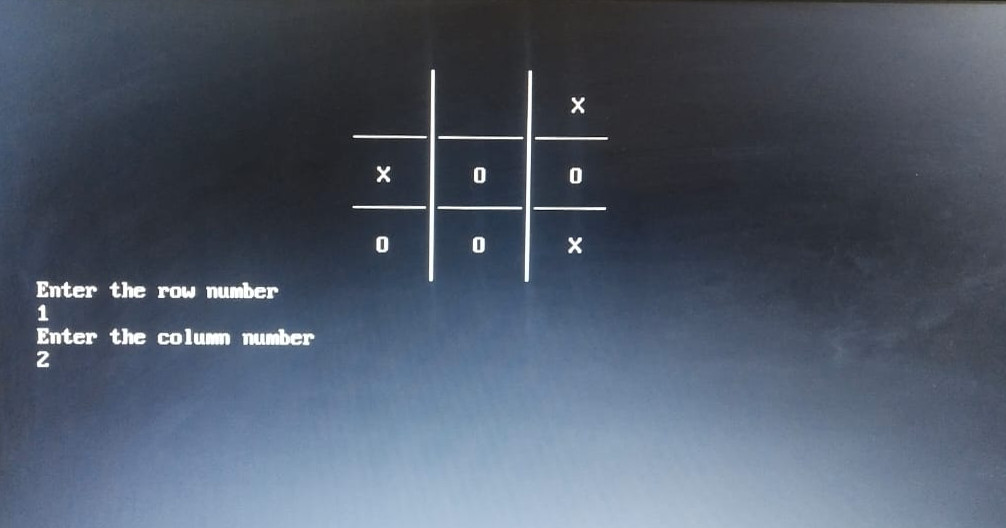
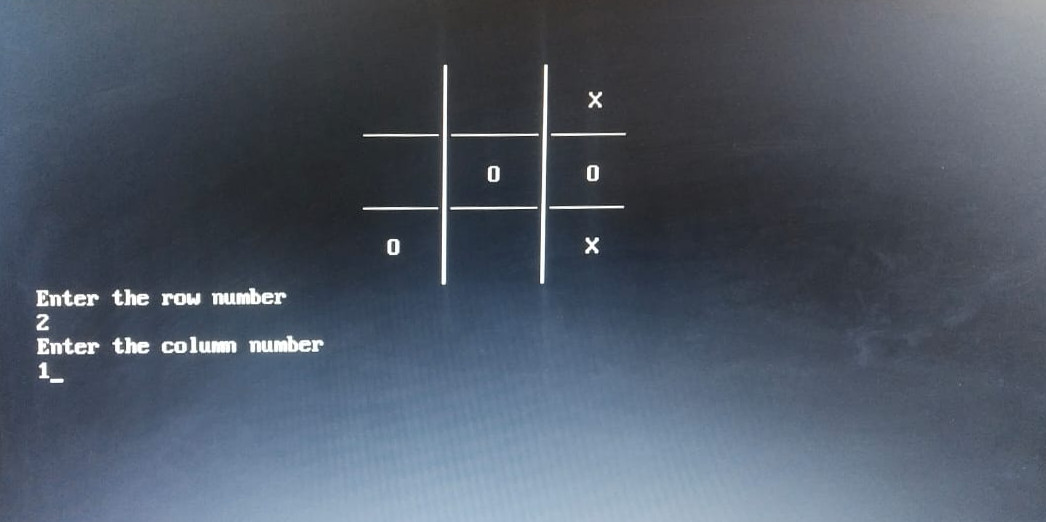


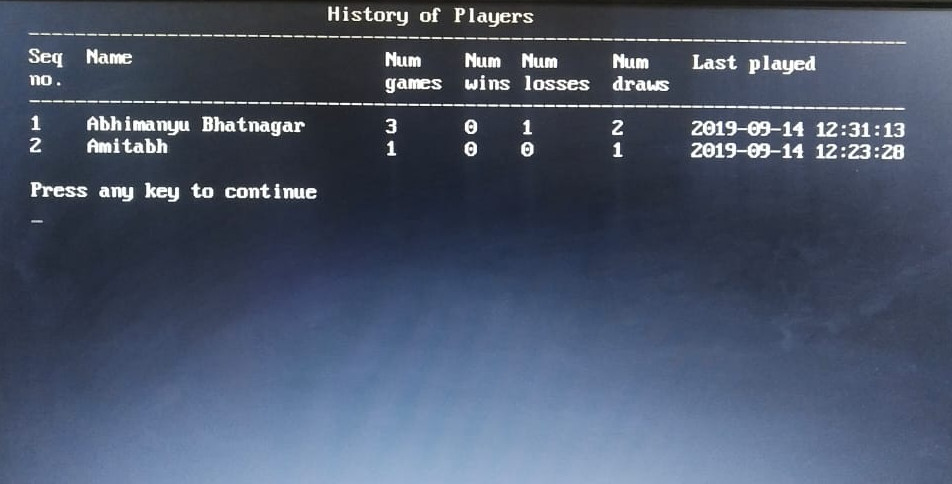
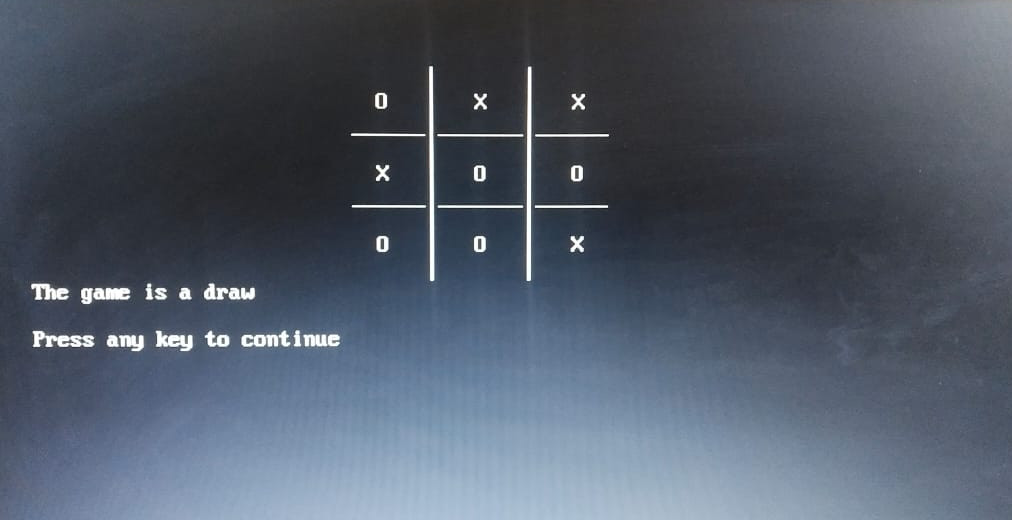


The following screenshots show another execution of the game.









**CONCLUSION**

After running the program dozens of times when playing with different players, I found that the program plays Tic-Tac-Toe quite intelligently against human players and is quite difficult to beat. In most cases, the computer either wins or draws the game. Only in certain specific scenarios, a human player is able to beat the computer. Hence I conclude that my self-developed design and implementation of Tic-Tac-Toe is successful.

**BIBLIOGRAPHY**

1. Sumita Arora, *Computer Science with C++* , 2018
2. Understanding the Four Types of Artificial Intelligence, <https://www.govtech.com/computing/Understanding-the-Four-Types-of-Artificial-Intelligence.html>
3. 7 Types Of Artificial Intelligence, <https://www.forbes.com/sites/cognitiveworld/2019/06/19/7-types-of-artificial-intelligence/#688393ba233e>
4. Types of AI: Reactive to self-aware being, <http://www.fintechbd.com/types-of-ai-reactive-to-self-aware-beings/>
5. IBM Icons of Progress: Deep Blue, <https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/>