Assignment 1 (Marks 10+2):

Instructions:

- Posting Date 8th Aug 2019;
- Due date of submission 21st Aug 2019;
- Demo: TA will announce demo Schedule during Lab timings and out of lab timings and must finish demos by 23rd Aug
- Assignment is to be done in group of two students. Form a group and send the information to me and TAs. Names and email of TAs are .
    - Anukriti Tyagi - at492@snu.edu.in
    - Surbhi Jain - surbhi.jain@snu.edu.in
- Well documented programs will be given 2 additional marks

Develop and implement the following in C/C++/Java/python:

- (2 marks) Write a program to generate magic square of dimension 3 x 3.
- (5 marks) Write program for 2-D tic tac toe using magic square concept. Winning situation is making collinear line.
- (3 marks) Display the board position after each turn along with list of contents for both the players.

CODE

```
package ai_assignment1;


import static java.lang.System.exit;

import java.util.ArrayList;

import java.util.Scanner;


public class CSD311_AI_1710110146_1710110263 {


    static int board[][] = new int[4][4]; //initialising a board of 4x4 with first row marking the indices of row number and first column marking the indices of column number

    static int tempMagic[][] = new int[3][3]; //temporary magic square generating array

    static  ArrayList<Integer> magicSquare = new ArrayList<Integer>(); //ArrayList containing the magic square values in order

    static ArrayList<Integer> playerh = new ArrayList<Integer>(); //ArrayList maintaing the moves of human player

    static ArrayList<Integer> playerc = new ArrayList<Integer>(); //ArrayList maintaing the moves of system

    static int move = 1; //variable to track the number of moves taken at any point of time. Move<=9 for a 3x3 TIC TAC TOE board.


    private static void createMagicSquare()  //Function to generate a Magic Square and then insert values into the ArrayList

    {

        int n = 3;

        int r = 0;
```

```
    int c = n/2;

    for (int i = 1; i <= n*n; i++) {

        tempMagic[r][c] = i;

        if (i % n == 0)
            r++;

        else
        {
            if (r == 0)
                r = n - 1;
            else
                r--;

            if (c == (n - 1))
                c = 0;
            else
                c++;
        }
    }

//Inserting the values of the magic square into the arraylist
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
```

```
                magicSquare.add(tempMagic[i][j]);

            }

        }

    }

    public static void main(String[] args) {

        Scanner s = new Scanner(System.in);

        int ch = 0;

        createMagicSquare();

        System.out.println("----------------------------------
Prepared by Illisha Singh and R Deepa--------------------------------
-------------------");

        System.out.println("Hello! Welcome to the game Tic Tac
Toe!\nEnter 1 if you wish to be player 1, Enter 2 if you wish to be
player 2");

        ch = s.nextInt();

        System.out.println("The game has the following rules:");

        System.out.println("1. You are allowed to make a move when
prompted");

        System.out.println("2. After each move, the grid will be
shown with 'X' marking the move of the first player"

                + "and 'O' marking the move of the second player");

        System.out.println("3. Enter -1 at any point to exit the
game");

        System.out.println("4. Initial state of the board is empty as
shown below, with the number of each row and column being
displayed");



          //initialising the first row of the board with index values

        for (int c=1;c<4;c++)

            board[0][c]=c;
```

```
          //initialising the first column of the board with index
values

        for (int r=1;r<4;r++)

            board[r][0]=r;


       //initialising the rest of the board with integer values of
the index as ordered linearly

        for(int r=1,i=1;r<4;r++)

            for(int c=1;c<4;c++)

                board[r][c]=i++;


        //shows the initial state of the board before the game begins

        showBoard();


        switch (ch) {

            case 1:  //when human choses to be the first player

                humanBegin();  //function called if human starts
playing first, subsequent calls to systemBegin() are made thereafter

                break;

            case 2: //when human choses to be the second player

                systemBegin();  //function called if system starts
playing first, subsequent calls to humanBegin() are made thereafter

                break;

            case -1:

                System.out.println("Exiting...");

                exit(0);

            default:
```

```
                System.out.println("You entered an invalid choice");
//prompt the user if the choice of player is incorrect i.e. neither 1
nor 2.

        }

    }



    private static void humanBegin()  //function that marks the
begining of the human moves on the board

    {

        while (true)

        {

            System.out.println("Your turn:");

            System.out.println("Enter the row and column of your move
"); //prompt the user to enter the move index according the the board
markings

            Scanner s = new Scanner(System.in);

            System.out.print("Row: ");

            int r = s.nextInt();

            if(r==-1)

            {

                System.out.println("Exiting...");

                exit(0);

            }

            System.out.print("Column: ");

            int c = s.nextInt();

            if(c==-1)

            {

                System.out.println("Exiting...");
```

```
                exit(0);

            }


            int m=board[r][c];  //variable m that stores the linear
value from 1 to 9(inclusive) of the move number from the board


            if (m >= 1 && m <= 9) {

                playerh.add(m);  //if the move is valid, add it to
the arraylist of the human moves

                if(move%2 == 0)

                    board[r][c] = 200;  //store the value on the
board as 200 if human is the second player

                else

                    board[r][c] = 100;  //store the value on the
board as 100 if human is the first player

                break;

            }

            else {

                System.out.println("Invalid Move. Enter again.");
//prompt the human if incase the entered move is not valid, i.e. does
not belong to the set [1,9].

            }

        }

        showBoard(); //display the state of the board to the human
after the move has been made

        checkIfWon('h'); //check if the current board state is the
one where human won

        move++; //increment move count
```

```
        if(move<=9)  //if there are still moves left go ahead and
call the system for a move

        {

            systemBegin();

        }

        else   //if system still has not won and moves are over, the
game is a tie

            System.out.println("The match is a tie");

    }



    private static void systemBegin() //function that marks the
begining of the system moves on the board

    {



        System.out.println("System's Turn: ");



        if(move == 1)  //if system is allowed to be player 1, always
choose 5

        {

            go(5);

        }

        else if(move == 2) //if system is player 2, go for center
position if available else pick any available position

        {

            if(!playerh.contains(5))  //check if the center position
is occupied by human

            {

                go(5);

            }
```

```
            else

            {

                goAnywhere(-1);

            }

        }

        else if(move == 3)  //if system has the third move, any
available position is equally valid

        {

            int d=15-(magicSquare.get(playerh.get(0)-1) +
magicSquare.get(playerc.get(0)-1));

            goAnywhere(magicSquare.indexOf(d)+1); //goAnywhere EXCEPT
d

        }

        else if(move == 5 && playerh.contains(1) &&
playerh.contains(7))

        {

            go(4);

        }

        else

        {

            int play=posWinC();  //choose a play position based on
the possibility of system winning or human NOT winning



            if(play > 0) //if there exists a move that allows system
to win or to block possible human win, play there

            {

                go(play);

            }
```

```
            else //take up a random position if none of the two
players are on the verge of winning

            {

                goAnywhere(-1);  //-1 because no need for any
exceptions, so the system can play anywhere

            }

        }



        showBoard(); //display the state of the board to the human
after the move of the system has been made

        checkIfWon('c'); //check if the current board state is the
one where system won

        move++; //increment move count



        if(move<=9) //if positions on the board are still vacant,
prompt the user to make a move

        {

            humanBegin();

        }

        else  //if system did not win and neither did the human and
the moves are over, it is a tie. Prompt accordingly.

            System.out.println("The match is a tie");

    }



    private static void showBoard() //funtion to display the state of
the board

    {



        for (int r = 0; r < 4; r++)
```

```
        {
            for(int c = 0; c < 4; c++)
            {
                if(r==0 && c==0)
                    System.out.print("    ");  //empty space for all
the available moves
                else if(r==0 && c!=0)
                    System.out.print(c+"   ");  //show the column
index and make partition on the board
                else if(r!=0 && c==0)
                    System.out.print(r+" | ");  //show the row index
and make partition on the board
                else
                {
                    if(board[r][c]==100)
                        System.out.print("X | ");  //if
board[r][c]=100, player 1 made the move there, mark as X
                    else if(board[r][c]==200)
                        System.out.print("O | ");  //if
board[r][c]=200, player 2 made the move there, mark as O
                    else
                        System.out.print("  | ");  //no moves made on
this position yet, display it to be empty.
                }
            }

            System.out.println();
            System.out.println("  -------------");
        }
```

```
    }


    static void go(int n) //function to make the system play or make
its move at board position passed in the parameter

    {

        playerc.add(n); //add the position chosen by the system to
its arraylist



        int r=0,c=0;

        //get the indices of the move number n from the board

        for(int i=1;i<4;i++)

            for(int j=1;j<4;j++)

                if(board[i][j]==n)

                {

                    r=i;

                    c=j;

                    break;

                }

      //insert the value of the move. 200 or 100, depending on the
player that made the move

        board[r][c] = (int) (100 * Math.pow(2, (move+1)%2));

    }



    static void goAnywhere(int e)   //go anywhere except the index
number e

    {

        for(int k=1;k<=9;k++)  //iterates through all possible
positions on the board to look for a vacant one
```

```
        {

            if(k!=e && !playerc.contains(k) && !playerh.contains(k))
//check if none of the players' arrayLists contains the position
value

            {

                go(k); //call go() and make the system take up this
vacant position as its next move

                return;

            }



        }

    }



    /* Strategy for posWinH() and posWinC() functions:

        The array lists playerh and playerc maintain the list of each
player's blocks in which he has played.

        Consider each pair of blocks that player owns.

        Compute difference D between 15 and the sum of the two
blocks.

        If D <= 0 or D > 9

        then these two blocks are not collinear and so can be ignored

        otherwise if the block representing difference is blank
(i.e., not in either list) then a move in that block will produce a
win.

    */



    static int posWinH() {

        for(int i=0;i<playerh.size();i++)

        {
```

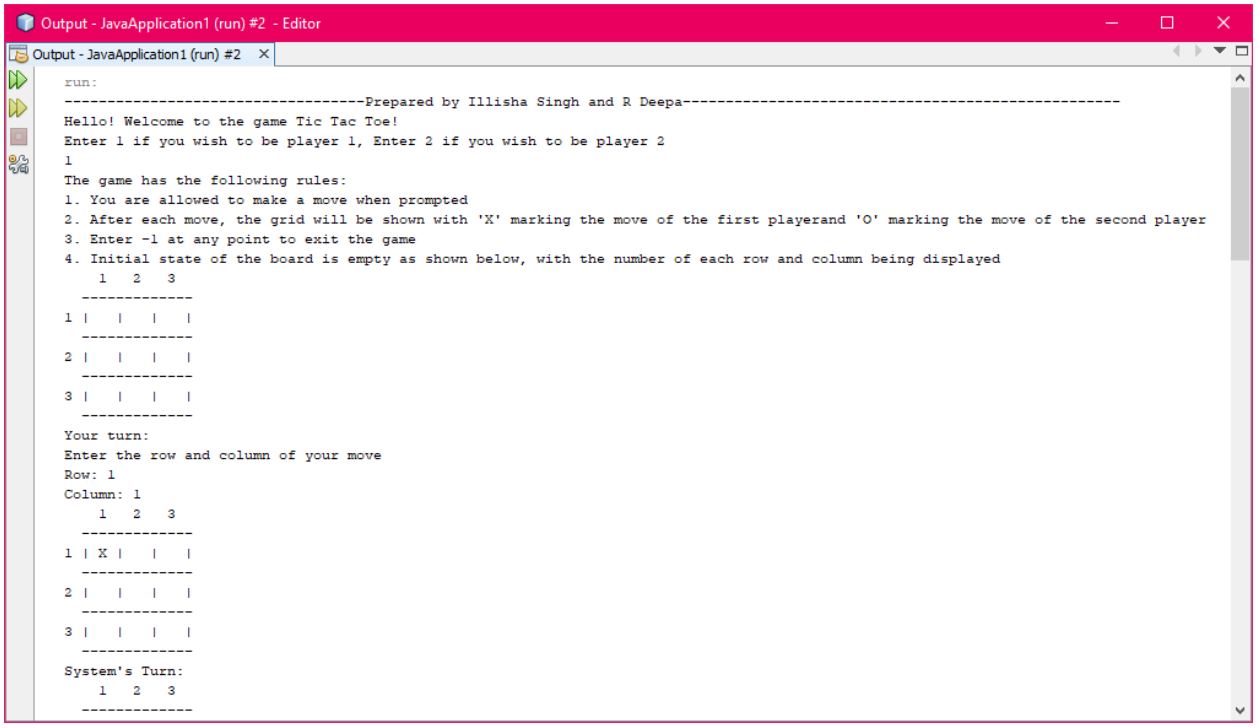
```
            for(int j=i+1;j<playerh.size();j++)

            {

                int d=15 - (magicSquare.get(playerh.get(i)-1) +
magicSquare.get(playerh.get(j)-1));

                //checking if there is any such pair of indices in
the list of player human which are a part of the same row or same
column or same diagonal

                if(d>=1 && d<=9)

                {

                    if(!playerh.contains(magicSquare.indexOf(d)+1) &&
!playerc.contains(magicSquare.indexOf(d)+1))

                    {

                        return magicSquare.indexOf(d)+1;

                    }

                }

            }

        }


        return -1;

    }


    static int posWinC() {


        for(int i=0;i<playerc.size();i++)

        {

            for(int j=i+1;j<playerc.size();j++)

            {
```

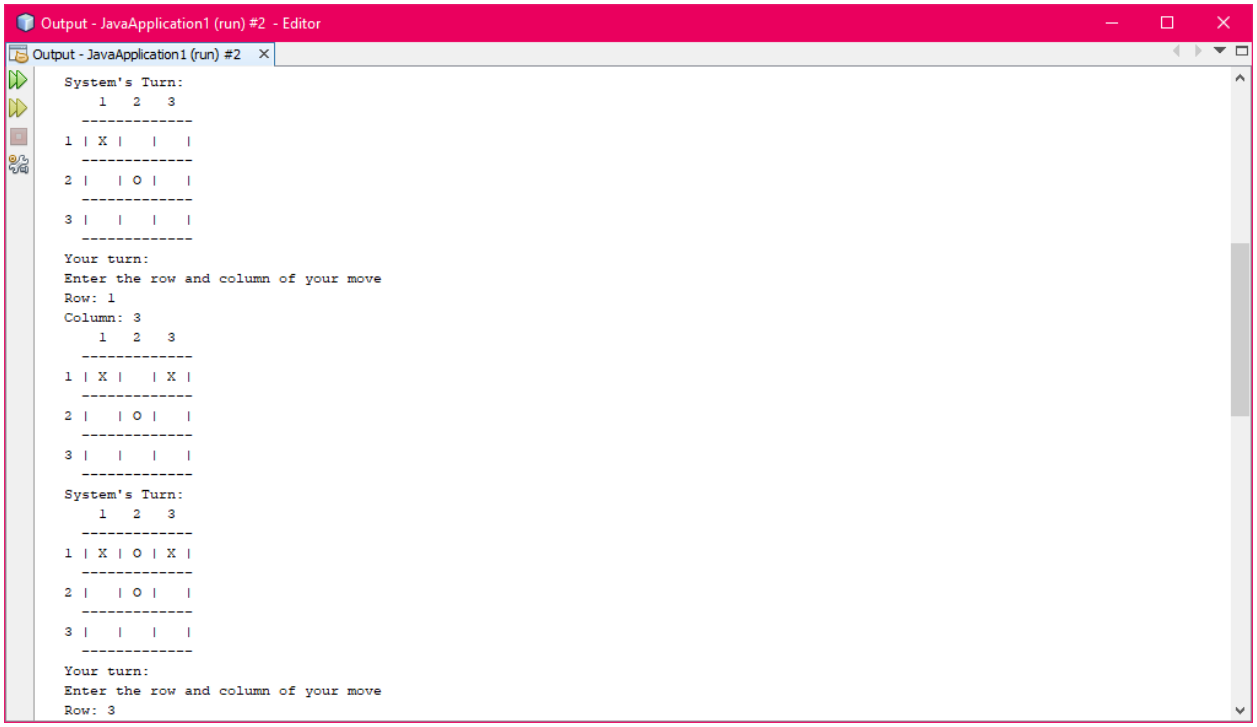
```
                int d=15 - (magicSquare.get(playerc.get(i)-1) +
magicSquare.get(playerc.get(j)-1));

                //checking if there is any such pair of indices in
the list of player computer which are a part of the same row or same
column or same diagonal

                if(d>=1 && d<=9)

                {

                    if(!playerh.contains(magicSquare.indexOf(d)+1) &&
!playerc.contains(magicSquare.indexOf(d)+1))

                    {

                        return magicSquare.indexOf(d)+1;

                    }

                    else //if there is no such pair, then check the
winning position of player human and block it

                    {

                        return posWinH();

                    }

                }

            }

        return posWinH(); //in any case if there is no possible win
of the computer, then find the winning position of human player and
block it



    }



    static void checkIfWon(char ch) //function to check if the player
ch has won

    {

        ArrayList<Integer> player = new ArrayList<Integer>();
```

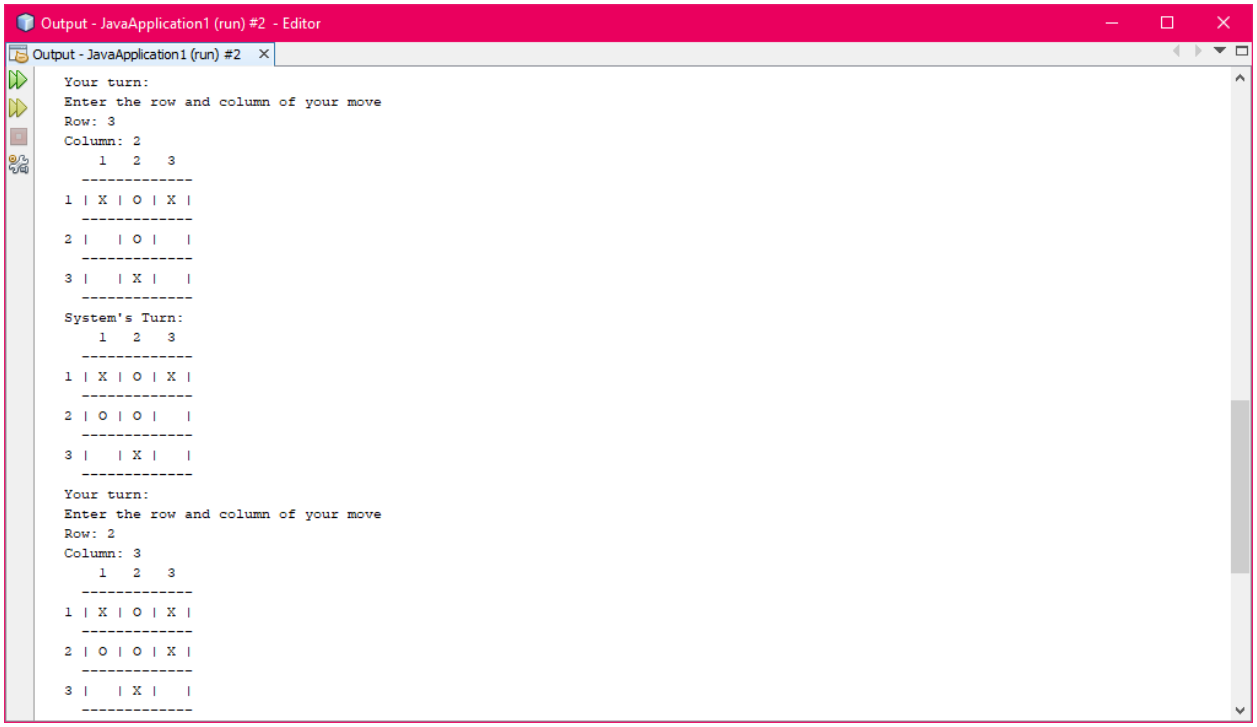
```
            if(ch=='c')
                player=playerc;
            else
                player=playerh;

            for(int i=0;i<player.size();i++)
            {
                for(int j=i+1;j<player.size();j++)
                {
                    //if there are any two elements in the list of the
    player ch that are collinear and if the player ch contains the third
    element required to make a sum of 15, then the player has won
                    int d=15 - (magicSquare.get(player.get(i)-1) +
    magicSquare.get(player.get(j)-1));
                    if(d !=(magicSquare.get(player.get(i)-1)) &&
    d!=(magicSquare.get(player.get(j)-1)) && d>=1 && d<=9)
                    {
                        if(player.contains(magicSquare.indexOf(d)+1))
                        {
                            System.out.println("Player " + (int)
    Math.pow(2, (move+1)%2) + " has won");
                            exit(0);
                        }
                    }
                }
            }
        }
}
```

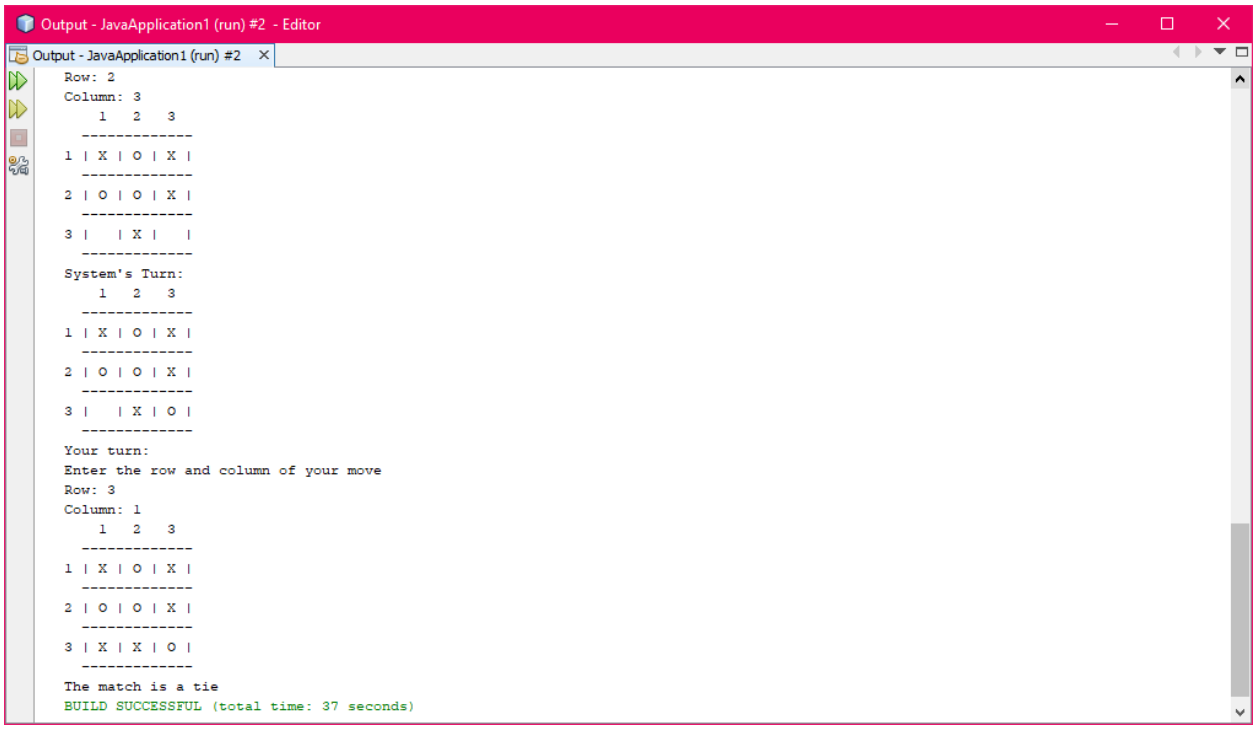
SCREENSHOTS

As player 1:

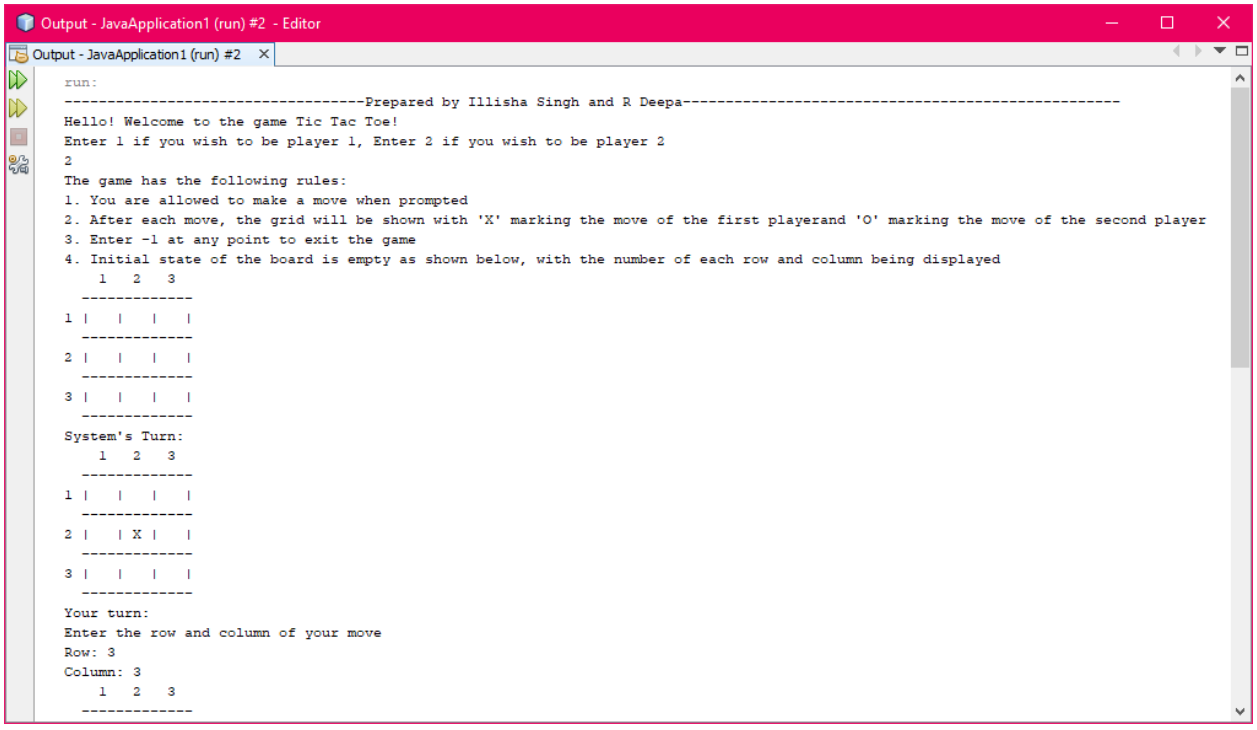


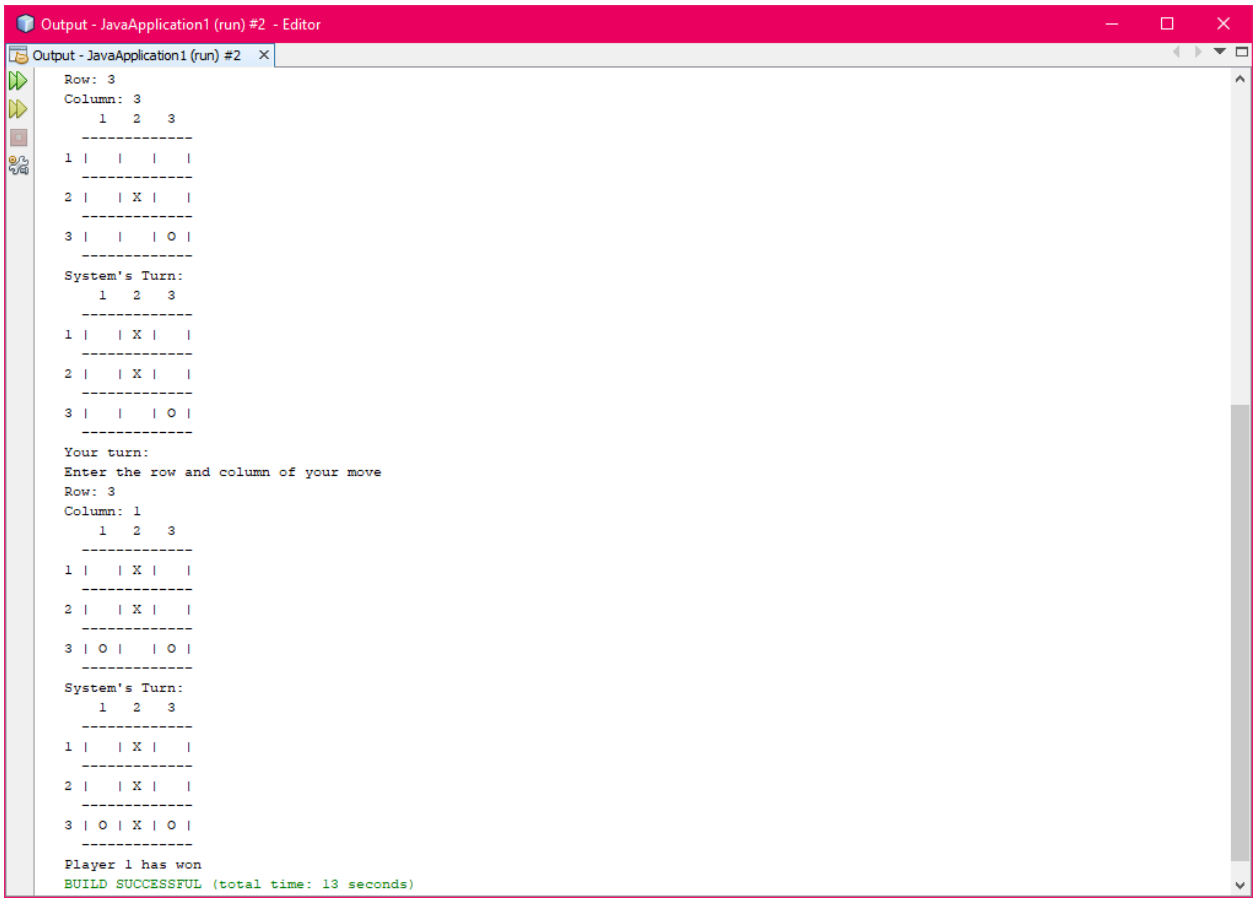



Output - JavaApplication1 (run) #2 - Editor
Output - JavaApplication1 (run) #2
Your turn:
Enter the row and column of your move
Row: 3
Column: 2
1 2 3
1 | X | O | X |
2 | | O | |
3 | | X | |
System's Turn:
1 2 3
1 | X | O | X |
2 | O | O | |
3 | | X | |
Your turn:
Enter the row and column of your move
Row: 2
Column: 3
1 2 3
1 | X | O | X |
2 | O | O | X |
3 | | X | |


Output - JavaApplication1 (run) #2 - Editor
Output - JavaApplication1 (run) #2
Row: 2
Column: 3
1 2 3
1 | X | O | X |
2 | O | O | X |
3 | | X | |
System's Turn:
1 2 3
1 | X | O | X |
2 | O | O | X |
3 | | X | O |
Your turn:
Enter the row and column of your move
Row: 3
Column: 1
1 2 3
1 | X | O | X |
2 | O | O | X |
3 | X | X | O |
The match is a tie
BUILD SUCCESSFUL (total time: 37 seconds)

As player 2:


Output - JavaApplication1 (run) #2 - Editor
Output - JavaApplication1 (run) #2
run:
----------Prepared by Illisha Singh and R Deepa----------
Hello! Welcome to the game Tic Tac Toe!
Enter 1 if you wish to be player 1, Enter 2 if you wish to be player 2
2
The game has the following rules:
1. You are allowed to make a move when prompted
2. After each move, the grid will be shown with 'X' marking the move of the first playerand 'O' marking the move of the second player
3. Enter -1 at any point to exit the game
4. Initial state of the board is empty as shown below, with the number of each row and column being displayed
1 2 3
1 | | | |
2 | | | |
3 | | | |
System's Turn:
1 2 3
1 | | | |
2 | | X | |
3 | | | |
Your turn:
Enter the row and column of your move
Row: 3
Column: 3
1 2 3


Output - JavaApplication1 (run) #2 - Editor
Output - JavaApplication1 (run) #2
Row: 3
Column: 3
1 2 3
1 | | | |
2 | | X | |
3 | | | O |
System's Turn:
1 2 3
1 | | X | |
2 | | X | |
3 | | | O |
Your turn:
Enter the row and column of your move
Row: 3
Column: 1
1 2 3
1 | | X | |
2 | | X | |
3 | O | | O |
System's Turn:
1 2 3
1 | | X | |
2 | | X | |
3 | O | X | O |
Player 1 has won
BUILD SUCCESSFUL (total time: 13 seconds)