Practical 1: Programming Warm-Up Through the Tic-Tac-Toe Game

Aayush Rana Magar
21070188
Department of Computer Engineeing
Everest Engineering College
12/5/2024

***Summary*—This lab is an introductory course to warm up the skills and knowledge of java and become comfortable to use it. To achieve that we are simulating a tic tac toe game in console. This game has two version, classical and variant version. Classical version is a standard game of tic tac toe of 3 by 3 grid and player to get a straight-line win. Variant version, on turn basis players are requested to put marks until all the boxes are filled and we calculated all possible straight lines and player with a greater number of lines wins.**

Keywords—Tic-tac-toe, Java, Two-Player Game, X & O, Variant.

I. Problem Description

In this practical we are going to write a java program to simulate a tic-tac-toe game in the console. It should simulate two type of tic-tac-toe game. They are:

1. Classical game where 9 boxes are arranged in 3 by 3 grid.
2. Variant game where n*n boxes are arranged in n by n grid.

II. Background

A. Tic-Tac-Toe

It is one of the oldest games that kids have been playing since their early childhood and introduction to strategical games. It is a 2-player game and due to its limited choices, this game is also solved.

So, a classical game of tic-tac-toe consists of 9 boxes arranged in 3 by 3 grid. Initially, all the boxes are empty. Two players one after another put their marks (X or O) in one of the empty boxes (can't re-write marks) until all the boxes are filled or a win state is reached. A win state is when 3 mark makes a straight line (horizontal, vertical or diagonal) in the grid. If all boxes are filled and win state is not reached then it's a draw.

But, a variant game is a bit different. it has n*n boxes arranged in n by n grid. players put their marks one after another until all the boxes are filled then we count all the straight lines made by X mark and O mark. Whoever has the higher number of straight lines is the winner and if the number of straight lines is same then it's a draw.

III. Program Description

When the program is stared it will allow us to choose option 1(classical game) or option 2(variant game). Initially the game is setup for a classical game. if choice is 1 classical game is played with initial setup and if choice is 2 then game ask you to enter size of maze greater than 3 and game resets board size and then variant game is played. After game is played the conclusion of the game is drawn i.e. (X wins or O wins or a draw).

For a classical game, a loop created with the conditions move is less than 9 and winner is not determined. X's turn is set to be 1st to move and then X is asked to enter a box id to put its mark and a mark is kept in that box if that box is empty. then similar with O's turn. It won't check for winner until move 5 as before it no winner is possible. After move 5 it check for horizontal, vertical, diagonal and then anti-diagonal straight line in that order. If a winner is found then loop ends and winner is displayed else after move 9 loops is terminated and a draw is declared.

For a variant game, a loop created with the conditions move is less than n*n and winner is not determined. X's turn is set to be 1st to move and then X is asked to enter a box id to put its mark and a mark is kept in that box if that box is empty. then similar with O's turn. It will only check for winner after all the boxes are filled. After the boxes a filled it checks for horizontal straight lines and count the number of lines made by X and O. for other Straight lines also, if it was encountered then that was added to their respective count

Area of search for lines for:

- Horizontal lines: (x, y), (x+1, y), (x+2, y) ranging from [0,0] to [row, column-2]
- Vertical lines: (x, y), (x, y+1), (x, y+2) ranging from [0,0] to [row-2, column]
- Diagonal lines: (x, y), (x+1, y+1), (x+2, y+2) ranging from [0,0] to [row-2, column-2]
- Anti-diagonal lines: (x, y), (x-1, y+1), (x-2, y+2) ranging from [0,2] to [row-2, column]

After that the number of lines is compared and whichever is the greatest is declared as winner else it's a draw.

IV. Implementation

A. TicTacToe Class

This class is used to prepare the grid of tic-tac-toe and consist of basic function to run the game.

1) TicTacToe() : This is a constructor of this calss that setup the gride(3 by 3) for a classical game. This function gets instally called when the object is created

2) domainExpansion(): After the choice is made to play the varient game this functions is called. It asks user to input size of grid greater than 3 and reszies the board to n by n.

3) getWinner(int moveNumber): This method is called after the mark is sucessfully kept in the box. For a classical game it checks for the winner after move number 5 and return either winner or game contines. For a varient game it will only check for winners after all the boxes are filled then if for each straight line it finds of a mark it increaments the count for that mark and this is continued for all possible straight line. Then it compairs the count and retures the winners as the one with highest count else it return darw.

B. TTTGame Class

This class is used to run and monitor the flow of the game including choosing to play classical or variant game to displaying the conclusion of the game. It in itself don't have any game logic but game control logic.

1) startGame(): This method is used at the beginning of program run. When class is declared the calssical game is preperared, and then this method is called and it asks uesrs to play a classical game or a variant game. On choossing variant game it calls on domainExpansion() which prepars the game for the varient. Then it displayes the borad and users start playing game (playGame())and then upon ending of game it also displays messgae by calling method printMessgae().

2) playGame() : Upon calling on this method it starts by defining the current move number and max possible move number. Then a loop is created which keeps on looping while winner is not declared and current move number is less than or equal to the max possible move number. In that loop it determines who's turn is it, ask a valid box to put mark in , display the updated board and check if winner is determined.

3) printMessgae():.It displays the conclusion of the game. If the winner variable is either 'X' or 'O' then the respective mark is the winner else it's a draw.

V. Tests and Results

For classical game there are total 8 straight lines (3 horizontal + 3 vertical + 1 diagonal + 1 anti-diagonal) that user can win. So, all possible 8 lines were inserted and they all test cases came positive.

For variant game it is impossible to test all test cases. So, the next best option was to do white box test and black box test from my colleges and they all also came positive.

Currently no bugs are found.

VI. Lessons Learnt

This lab was an attempt to warm up my java programming. This exercise helped me how classes and objects are used in java and how it gives its independency from machines. It also taught me how to handle arrays in java and how array properties are used to manipulate arrays. I also learned that characters can be compared by using logical operators.

The part where we struggled the most was in determining the winner of the game. There were 2 major problems I faced while completing the labs. 1st of all, the variant game does not have any official rules, this means that there is some grey area in the logic like what to do if one make is in intersection of two line, what to do then? For some reason I was adamant on one mark can be part of only one line and due to this reason, I started to over engineer the game which ultimately made me realize that majority of the game will be a draw. So, I decided to accept all possible lines that could be possibly made. Which comes to my 2nd dilemma on how to check all possible lines that could be made. At, first I thought I will go thought every point and check for horizontal, vertical, diagonal and anti-diagonal lines. But there were cases of array out of bounds errors (trying to access data outside the bounds of array) like in a game of n x n game, suppose we reach to (0, n) position so then for vertical we check (0, n), (1, n) and (2, n) but for horizontal we will be accessing (0, n), (0, n+1) and (0, n+2) but (0, n+1) and (0, n+2) doesn't exist. So, to solve this we simply search for all horizontal lines then vertical lines and so on, one at a time.

Till now this is being run in console and it works all fine but java already supports Jframe. So, we can move forward in development by integrating GUI elements in the program. In this program X always plays the 1st move, so we can develop it such a way that it be random. This program runs one time and then terminates but we can make it so we can play it multiple times and keep scoring until program is terminated.

VII. Program Listing

A. TicTacToe.java

```
    public void domainExpansion(){
        this.isStandard = false;
        Scanner sc = new Scanner(System.in);
        int size ;
        //to make sure the size of board is
more than 3

        do {
            System.out.println("enter size
greater than 3");
            size = sc.nextInt();
        } while (size <= 3);

        this.size = size;
        //this is ok since redefining will
update board clear
        board = new char[size][size];
        for (int i = 0; i<size ;i++) {
            for (int j = 0; j<size ;j++){
                board[i][j]=' ';
            }
        }
    }

    public  void displayBoard() { // this
displays the board in console
        int row = size;
```

```
        int coulmn = size;

        System.out.println(turn+"s turn to
put the mark");

        for (int j = 0; j < coulmn; j++) {
            System.out.print("--------");
        }
        System.out.println();
        /*
                with each box we check if
there exists X or O in box.
                if yes we print it and if no
we print it's respective number to give it's
input
         */
        int index = 0;
        String numOrXO;

        for (int i = 0; i < row; i++) {
            for (int j = 0; j < coulmn; j++)
{
                if(board[i][j]!= ' '){
                     numOrXO =
String.valueOf(board[i][j]);
                }else {
                     numOrXO =
String.valueOf(index);
                }

System.out.print("|\t"+numOrXO+"\t|");
                index++;
            }
            System.out.println();
            for (int j = 0; j < coulmn; j++)
{
                System.out.print("--------
");
            }
            System.out.println();
        }
    }



    public char getWinner(int moveNumber){

        /*
        here there are 2 types of winners
        1: standard game winner that can be
called out before finishing all the moves
        2: variation  where all the boxes
are
            filled and then winner is
decided by counting the number of line made
         */
        if(isStandard){
            //since it is impossible to wun
before move 5
            if (moveNumber <5){ return ' ';}
            //for horizontal lines
            for(int x=0; x< size ; x++){
                if(board[x][0] ==
board[x][1] && board[x][1] == board[x][2] &&
board[x][0] != ' ' ){
                    return board[x][0];
                }
            }
            //for vertical lines
            for(int x=0; x< size ; x++){
                if(board[0][x] ==
board[1][x] && board[1][x] == board[2][x] &&
board[0][x] != ' ' ){
                    return board[0][x];
                }
            }

            //for diagonal line
            if(board[0][0] == board[2][2] &&
board[2][2] == board[1][1] && board[1][1] !=
' '){
                return board[0][0];
            }

            //for anti-diagonal line
            if(board[0][2] == board[1][1] &&
board[1][1] == board[2][0] && board[1][1] !=
' '){
                return board[1][1];
            }
            return ' ';
        }
        else {
            /*
                for this variation we don't
need to calculate winnings for
                move number less than max
moves. So we just return ' ' for efficiency
purposes
             */
            if (moveNumber <
maxNumberOfTurn()){return ' ';}
        /*
            for grid of n*m
            we can make this efficient by
selecting (x,y)box and concatenation 3
consecutive chars
            horizontal((x,y),(x,y+1),(x,y+2)
) where 0<=x<n-2 and 0<=y<m,
            vertical((x,y),(x+1,y),(x+2,y) )
where 0<=x<n and 0<=y<m-2,

diagonal((x,y),(x+1,y+1),(x+2,y+2)) where
0<=x<n-2 and 0<=y<m-2 and
             anti diagonal((n-x,m-y),(n-
(x+1),m-(y+1)),(n-(x+2),m-(y+2)) where
0<=x<n-2 and 2<=y<m

             for multiple grid grater than 3
it counts the number of lines made and one
mark CAN be
             used in marking another line.
```

```
             eg: if there are X's
(1,1)(2,2) and (3,3) creating line 1 and
another set of
             X's in (3,1) (2,2) and (1,3)
creating line 2.
             then both line 1 or line 2
exists even if there is overlapping element
(2,2)
         */

            int row=size;
            int coulmn= size;
            int countXWins= 0;
            int countOWins = 0;


            // checking horizontal lines are
drawn if any
            for(int x= 0 ; x< row ; x++){
                for(int y =0 ; y < coulmn-
2;y++){
                    if(board[x][y] ==
board[x][y+1] && board[x][y+1] ==
board[x][y+2] && board[x][y+2] != ' ' ){
                        //someone has made a
line
                        if(board[x][y] ==
PLAYER_1){
                            countXWins++;
                        }else {
                            countOWins++;
                        }
                    }
                }
            }


            // checking vertical lines are
made or not
            for(int x= 0 ; x< row-2 ; x++){
                for(int y =0 ; y <
coulmn;y++){
                    if(board[x][y] ==
board[x+1][y] && board[x+1][y] ==
board[x+2][y] && board[x+2][y] != ' '  ){
                        //someone has made a
line
                        if(board[x][y] ==
PLAYER_1){
                            countXWins++;
                        }else {
                            countOWins++;
                        }
                    }

                }
            }

            // checking diagonal lines are
made or not
            for(int x= 0 ; x< row-2 ; x++){
                for(int y =0 ; y < coulmn -
2;y++){
                    if(board[x][y] ==
board[x+1][y+1] && board[x+1][y+1] ==
board[x+2][y+2] && board[x+2][y+2] != ' ' ){
                        //someone has made a
line
                        if(board[x][y] ==
PLAYER_1){
                            countXWins++;
                        }else {
                            countOWins++;
                        }
                    }

                }
            }

            // checking anti-diagonal lines
are made or not
            for(int x=0  ; x<row-2 ; x++){
                for(int y =2 ; y <coulmn
;y++){
                    if(board[x][y] ==
board[x+1][y-1] && board[x+1][y-1] ==
board[x+2][y-2] && board[x+2][y-2] != ' ' ){
                        //someone has made a
line
                        if(board[x][y] ==
PLAYER_1){
                            countXWins++;
                        }else {
                            countOWins++;
                        }
                    }

                }
            }

            System.out.println("wins of X =
"+countXWins+ " \n" +
                    "wins of O = "
+countOWins);

            if(countXWins>countOWins){
                return PLAYER_1;
            } else if
(countOWins>countXWins) {
                return PLAYER_2;
            }else {
                return ' ';
            }
        }

    }
```

B. TTTGame.java

```
       public void startGame(){
           Scanner       sn       =       new
Scanner(System.in);
           System.out.println("""
                   Choose one:
                   1: Play a Classical game (3
x 3)
                   2: Play a n X n game
                   """);
           //to  make  sure  correct  input  in
entered
           int choice;
           while (true){
               choice = sn.nextInt();
               if(choice == 1 || choice ==2 ){
                   break;
               }
               else {
                   System.out.println("Wrong
choice: enter 1 or 2");
               }
           }

           if(choice == 2){
               game.domainExpansion();
           }

           game.displayBoard();
           playGame();
           printMessage();
       }

       public void playGame(){
           Scanner       in       =       new
Scanner(System.in);
           int moveNumber =1;
           int           maxMoves           =
game.maxNumberOfTurn();
           char turn;
           int boxId;
           /*
           while no one has won or drawn a
game yet
           and needs to mark all the boxes and
then calculate winner
            */
           while ( winner == ' ' && moveNumber
<= maxMoves){
               turn = game.whoseTurn();
               System.out.println(turn+"'s
turn . enter the number you want to put your
mark");
               do{
                   boxId = in.nextInt();
               }while (game.getMark(boxId) !=
' ');
               /*
               is the cell empty?
               if yes we choose another one;
                */

               game.putMark(boxId);
               game.displayBoard();
               winner                          =
game.getWinner(moveNumber);

               moveNumber++;
           }
           in.close();
       }
```