CMPE 150 ASSIGNMENT 2

ZUHAL DIDEM AYTAC 20184000045

I PROBLEM DESCRIPTION

We are asked to write an XOX game. The program should be user-interactive. It makes the user choose between creating a new file or loading an existing file. So many exceptions must be checked in the load section for a right output and to avoid errors.

One player is user and the other is the computer. They should play one-by-one until somebody wins or until the game ends in tie. When a game ends, we ask the user if he/she wants to play again. If so, the program should start a new game with an empty board.

After each game end, we record who has won. And when the user does not want to play any more games, the program should print out the final results.

II PROBLEM SOLUTION

playTheGame method is the one that makes the moves. It plays one move for the player in turn. The condition for the moves is that game should be continuing, with no wins and no full board. The board configuration is printed out before every move of the user. After each play, the turn passes to the other player. This method still runs if the game has ended. Firstly, tie condition is checked, which means nobody has won and no empty spaces are left. No player gains points in that situation. If somebody has won, the total points of that player is incremented here, and the user won that game is printed put. (Not the final results yet.) After that, the endOfGame method is called.

The method named **endOfTheGame** applies the rules after the game has ended. It is called in playTheGame method because these steps should occur after the end of every game. It has 3 options. If player types “Y”, it assigns a new current board and a random player. It assigns integer again to value 1, so that the while loop in main can continue and the playTheGame method can be called. But if player types “N”, the program assigns again to 0, so that the while loop in main is not entered again. Final results are printed in that part. If the user does something instead of these, the program asks him/her to type again.

CheckIfFileIsProper method returns boolean, and checks for the rules for the loaded file to be assigned as a board. The file should be at least 16 characters long. It should not contain any other characters than X, O and E. The difference between Xs and Os can be 1 at maximum. If so, the player whose symbol has lower frequency may start. If they are equal, starter may be chosen randomly.

The method named **whereToStart** either loads a board from a file and returns it to a string if it is proper, or creates a new empty board as a string. If a file is loaded, it decides who starts.

ChooseSymbol asks the user which symbol he/she wants to use. The other symbol is assigned as computer's symbol.

PrintTheBoard method accepts a string of 16 characters length and prints it to a 4x4 board, with spaces and | symbols between them. It has 2 for loops inside, one to create 4 rows and the other to fill each row with 4 cols.

WhoStarts method is called when the starter is to be chosen randomly. The random has 2 opportunities, each representing a starter. This method also prints out the prompt that announces the starter.

ContinueGame is the method that returns true while somebody can still make a move. Otherwise it returns false, indicating the game has ended. The continuing conditions are the game not having a winner and empty spaces are being left.

CrossCheck, verticalCheck and horizontalCheck methods are the ones that checks for winning condition. They are all gathered together in the **checkForWin** method that returns true when either one of the 3 check methods return true.

ComputersPlay method decides a move for the computer. It chooses an index between 0-16, randomly. Then checks if that index is empty, if not, it assigns a new random until an empty index is found. When so, it first substrings the current board till that index, adds the computer's symbol and then adds the rest of the current board; that new string created is assigned as current board and returned.

PlayersPlay method applies user's move choice. It asks the user to give coordinates, checks if proper. The user, different from the computer, enters coordinates, not an index. So, both x and y should be between 1 and 4. If that holds, the coordinates are returned to index and stored in an integer. Than the method checks if that index is empty, if so, it substrings and changes the E symbol to the player's symbol. If not, it asks the user to enter proper coordinates.

All of these methods are gathered together in the **main** method, we can say. A scanner is created and interacts with the user. First method to be called is whereToStarts as it assigns the starting board as a string. After that, playTheGame method is called in the main, the first game is played. In the playTheGame method, endOfTheGame method is called. Integer named again has a value of –1, currently. If the user wants to play again, it is changed to 1 and the while loop is entered. Inside that loop, playTheGame (and endOfTheGame method inside it) methods are called. This while loop runs until the user types N at the end of one game. At that situation again's value is changed to 0 and we get out of the while loop. The final results are printed, and the program has ended.

III *IMPLEMENTATION*

```
import java.io.*;

import java.util.*;


public class ZDA2018400045 {

    // Outside the main method, i declared some important variables for them to be visible in
all methods.

    public static char playerSymbol;        // This char represents computer's symbol

    public static char computerSymbol;        // This char represents computer's symbol

    public static String startingBoard;        // This String holds the starting board.

    public static String currentBoard;        // This String holds the current board and updated in
each step

    public static String currentPlayer;        // This String holds the current player as
ComputersTurn or PlayersTurn

    public static int countx;        // These integer is to count X's if a file is uploaded.

    public static int counto;        // These integer is to count O's if a file is uploaded.

    public static int counte;        // These integer is to count E's if a file is uploaded.

    public static int countPlayersWins;     // These integer counts the wins of the user in one
play.

    public static int countComputersWins;    // These integer counts the wins of the computer
in one play.

    public static int again;               // This integer shows if the user wants to play again (1 if
yes, 0 if no)
```

```
   // This is my main method. I tried to shorten it by splitting it into methods.

   public static void main(String [] args) throws FileNotFoundException{


       Scanner console= new Scanner(System.in); // I read some inputs from the user so i
declare a new scanner.

       System.out.println("Welcome to the XOX Game.");

       whereToStart();                 // This method is called for the user to choose to load or
create a file.

       playTheGame();                   // This method runs until someone wins or there is tie.

       while ( again == 1) {            // again being equal to 1 means user wants to play again,

          playTheGame();                // when so, we call the playTheGame method.

       }


   }
```

```
// This method is run after the game ends.

// We ask the user if he/she wants to play again

public static void endOfTheGame() throws FileNotFoundException{

   System.out.print("Do you want to play again? (Y or N) ");

   Scanner console= new Scanner(System.in);

   String answer= console.next();

   while(true) {

      // I reset again to -1 here.

      again=-1;

      if(answer.charAt(0)=='Y')     {

         currentBoard = "EEEEEEEEEEEEEEEE";    //Games other than the first will always start
with an empty board.

         currentPlayer= whoStarts();           //Symbols are same but the starting user will
again be random.

         again=1;                        // again being equal to 1 means a new game will begin.

         break;

 }

      else if(answer.charAt(0)=='N') {    //If the user types N, we print out the statistics.

         System.out.println("You: " + countPlayersWins+ " Computer: "+ countComputersWins);

         System.out.println("Thanks for playing!");

         again=0;                      // again being equal to 0 means

         break;

      }

      else {                           //If the user types something other than Y and N, we prompt
to type again.

         System.out.println("Wrong input!");

         System.out.print("Do you want to play again? (Y or N) ");

         answer= console.next();        //We go back to the beginning of the while loop.

         continue;

      }

   }

}
```

```
//This is the most important method that makes the moves.

public static void playTheGame() throws FileNotFoundException{

Scanner console= new Scanner(System.in);

while(continueGame(currentBoard)) { //This loop runs if there is no winner and there is at
least one empty place.


          if(currentPlayer.equals("computersTurn")) {        //If it is computers turn,

             currentBoard = computersPlay(currentBoard);       //currentBoard is assigned to
the return value of the computersPlay method.

             System.out.println("");

             currentPlayer = "playersTurn";                   //The turn passes to the player.

          }


          else if (currentPlayer.equals("playersTurn")){             //If it is players turn,

             printTheBoard(currentBoard);

             currentBoard = playersPlay(currentBoard,playerSymbol); //currentBoard is
assigned to the return value of the playersPlay method.

             currentPlayer = "computersTurn";                    //The turn passes to the
computer.



          }





      }


// The game no longer continues after here.

if (currentBoard.indexOf('E')==-1 && !checkForWin(currentBoard)) {

   System.out.println("There was a tie!");        // First condition to check if there is a tie

}


else if (currentPlayer.equals("playersTurn")) {        // If turn has passed to player and the
player can't make a move,
            countComputersWins++;                  // That means computer has won. So win
number of computer is incremented.

            printTheBoard(currentBoard);

            System.out.print("Computer wins! ");

        }

else if (currentPlayer.equals("computersTurn")) {       // If turn has passed to player and
the computer can't make a move,

            countPlayersWins++;                     // That means computer has won. So win
number of player is incremented.

            printTheBoard(currentBoard);

            System.out.print("You win! ");

        }

// after checking win-lose-tie conditions, we can call the end method to ask the user if
he/she wants a new game.

endOfTheGame();



    }
```

```
  public static boolean checkIfFileIsProper(String inside) throws FileNotFoundException{
        // This method checks if the first 16 char of the string is proper to be a board.
        // If even one character is not equal to E,X or O or if there is a winner; method returns
false
        if(inside.length()<16) { //If the file is shorter than 16 characters, it can't be used as a
board.
            System.out.println("This is not a proper file!");
            return false;
        }
    // The numbers of X,O and E's are reseted to 0 in every existing new file.
    countx=0;
    counto=0;
    counte=0;
        currentBoard = inside.substring(0, 16);
        for(int i=0; i<16; i++) {
            if(inside.charAt(i)=='O')
                counto++;
            if(inside.charAt(i)=='X')
                countx++;
            if(inside.charAt(i)=='E')
                counte++;
            }
            if(!(counto+counte+countx==16) || checkForWin(currentBoard) || Math.abs(countx-
counto)>1) {
                // sum of the three integers must equal 16, the length of the string.
                // if not, that means there is some other character inside; so method returns false,
                // method returns false also when an ended game is loaded.
                // As players play one-by-one this method returns false when the difference between Xs
and Os is greater than one
            System.out.println("This is not a proper file!");
            return false;
        }  return true;}
```

```
// This method interacts with the user and returns the starting board as a String.
    public static String whereToStart() throws FileNotFoundException{


        System.out.print("Would you like to load the board from file or create a new one? (L or C)
");
        Scanner console= new Scanner(System.in);
        while(console.hasNext()) {
        String where= console.next();
        if (where.charAt(0)=='L') {
            while(true) {


                //If the user wants to load from a file we ask him/her to type a file name.
                // We take the file name as a string with scanner and declare a file.
                System.out.print("Please enter file name: ");
                String fileName= console.next();
                File f= new File(fileName);


            if (f.exists() && f.canRead() && f.isFile()) {
                //If such a file exists we create a new scanner to scan the file.
                // We take the first line of the file as a string with scanner.
                Scanner file= new Scanner(new File(fileName));
                String inside= file.nextLine();
                    if(checkIfFileIsProper(inside)) {
                        System.out.println("Load successful.");
                        chooseSymbol();
                        if((playerSymbol=='X' && countx>counto)||(playerSymbol=='O' &&
counto>countx)) {
                            // the symbol with lower frequency may start.
                            currentPlayer="computersTurn";
                            System.out.println("Computer will start:");
                        }                       else if(countx==counto)
                            // starter may be selected randomly if frequencies are equal
                            currentPlayer= whoStarts();
                else {
                    currentPlayer="playersTurn";
                    System.out.println("You will start:");
                }
                // if the first 16 characters are proper we substring and return as the current
board.
                startingBoard=currentBoard;
                return currentBoard;
            }
        }
        // If such a file does not exist, we prompt and continue the while loop from the
beginning.
        else {
            System.out.println(fileName + " does not exist. ");
            continue;
        }
    } // While loop ends here
    }
    //If the user does not load a board, an empty board is assigned.
    else if (where.charAt(0)=='C') {
        currentBoard = "EEEEEEEEEEEEEEEE";
        chooseSymbol();
        currentPlayer= whoStarts();
        startingBoard= currentBoard;
        return currentBoard;
    }
    else {
        System.out.println("Wrong input!" );
        System.out.print("Would you like to load the board from file or create a new one? (L or
C) ");
        continue;
    } }
    return currentBoard;}
```

```
 // This method asks the user the choose his/her symbol.
  public static void chooseSymbol() {
      Scanner console= new Scanner(System.in);
      System.out.print("Enter your symbol: (X or O) ");
      playerSymbol= console.next().charAt(0);


      //I start an indefinite for loop
     // If the user types X or O, that character is assigned as player's symbol,
     // the other character is assigned as computer's symbol.
     // We get out of the loop in this two situations.
      while(true) {
      if (playerSymbol== 'X') {
          computerSymbol = 'O';
          System.out.println("You are player "+ playerSymbol + " and the computer is player "+
computerSymbol + ".");
          break;
      }
      else if (playerSymbol== 'O') {
          computerSymbol='X';
          System.out.println("You are player "+ playerSymbol + " and the computer is player "+
computerSymbol + ".");
          break;
      }
      else {
          // If the user types something other than X or O,
        // We urge him/her to type something proper, the loop continues until a proper char is typed.
          System.out.println("Wrong input! Please try again.");
          System.out.print("Enter your symbol: (X or O) ");
          playerSymbol= console.next().charAt(0);
          continue;}
      }
  }
```

```
}



    // This method prints the string board in a proper way, it sets the structure.
    public static void printTheBoard(String Board) {
        // The first for loop is for lines.
        for(int i=0; i<4; i++) {
            for(int j=0; j<4; j++) {
                //The second for loop is for rows.
                System.out.print("| ");
                System.out.print(Board.charAt(i*4+j));
                System.out.print(" ");
            }
            System.out.println("|");
        }
    }



    // this method assigns a random integer between 0-2(not included. It basically has two
options: 0 or 1.
    // As the possibility of 0 and 1 are equal, i said if i equals 0, let it be computers turn,
    // and if i equals 1, it is players turn.
    public static String whoStarts() {
        Random r= new Random();
        int i = r.nextInt(2);
        if (i==0) {
            System.out.print("Computer will start:");
            return "computersTurn";
        }
        System.out.println("You will start:");
        return "playersTurn";
    }
    // this method checks if there are still empty spaces on the board and if nobody has
won.

    // returns true if so, indicating game can continue.

    public static boolean continueGame(String currentBoard) throws
FileNotFoundException {

    if (!(currentBoard.indexOf('E')==-1) && !checkForWin(currentBoard)) {

        return true;

    }

    return false;


    }




    // this method gathers together the three options of a player to win.

    // If one of them is true, the method returns true, indicating there is a winner.

    public static boolean checkForWin(String currentBoard) throws FileNotFoundException {

        if (verticalCheck(currentBoard) || horizontalCheck(currentBoard) ||
crossCheck(currentBoard)) {

            return true;

        }

    return false;


    }
```

```
// This method checks if somebody won, diagonally.

// I based the method on indexes.

// if characters at indexes (2,5,8),(3,6,9),(6,9,12),(7,10,13),(0,5,10),(1,6,11),(4,9,14) or
(5,10,15) are same,

// and are not empty, this method returns true.

public static boolean crossCheck(String currentBoard) {


    for(int i=2; i<=3; i++) {    // checks (2,5,8),(3,6,9)

    if (currentBoard.charAt(i)==currentBoard.charAt(i+3) &&
currentBoard.charAt(i+3)==currentBoard.charAt(i+6) && !(currentBoard.charAt(i)=='E')) {

        return true;

        }

    }


    for(int i=6; i<=7; i++) {    // checks (6,9,12),(7,10,13)

        if (currentBoard.charAt(i)==currentBoard.charAt(i+3) &&
currentBoard.charAt(i+3)==currentBoard.charAt(i+6)&& !(currentBoard.charAt(i)=='E')) {

            return true;

        }

    }

    for(int i=0; i<=5; i++) {    // checks (0,5,10),(1,6,11),(4,9,14) or (5,10,15)

    while ((i%4==0) || (i%4==1)) {

        if (currentBoard.charAt(i)==currentBoard.charAt(i+5) &&
currentBoard.charAt(i+5)==currentBoard.charAt(i+10)&& !(currentBoard.charAt(i)=='E')) {

        return true;

        }

    break;

    }

  }

return false;

}
```

```
}
```

```
//This method checks if somebody won, vertically.

//I based the method on indexes.

//if characters at indexes (0,4,8), (1,5,9), (2,6,10), (3,7,11), (4,8,12), (5,9,13), (6,10,14) or
(7,11,15) are same,

//and are not empty, this method returns true.

public static boolean verticalCheck(String currentBoard) {

   for (int i=0; i<=7; i++) {

      if (currentBoard.charAt(i)==currentBoard.charAt(i+4) &&
currentBoard.charAt(i+4)==currentBoard.charAt(i+8) && !(currentBoard.charAt(i)=='E')) {

         return true;

      }

   }

   return false;

}
```

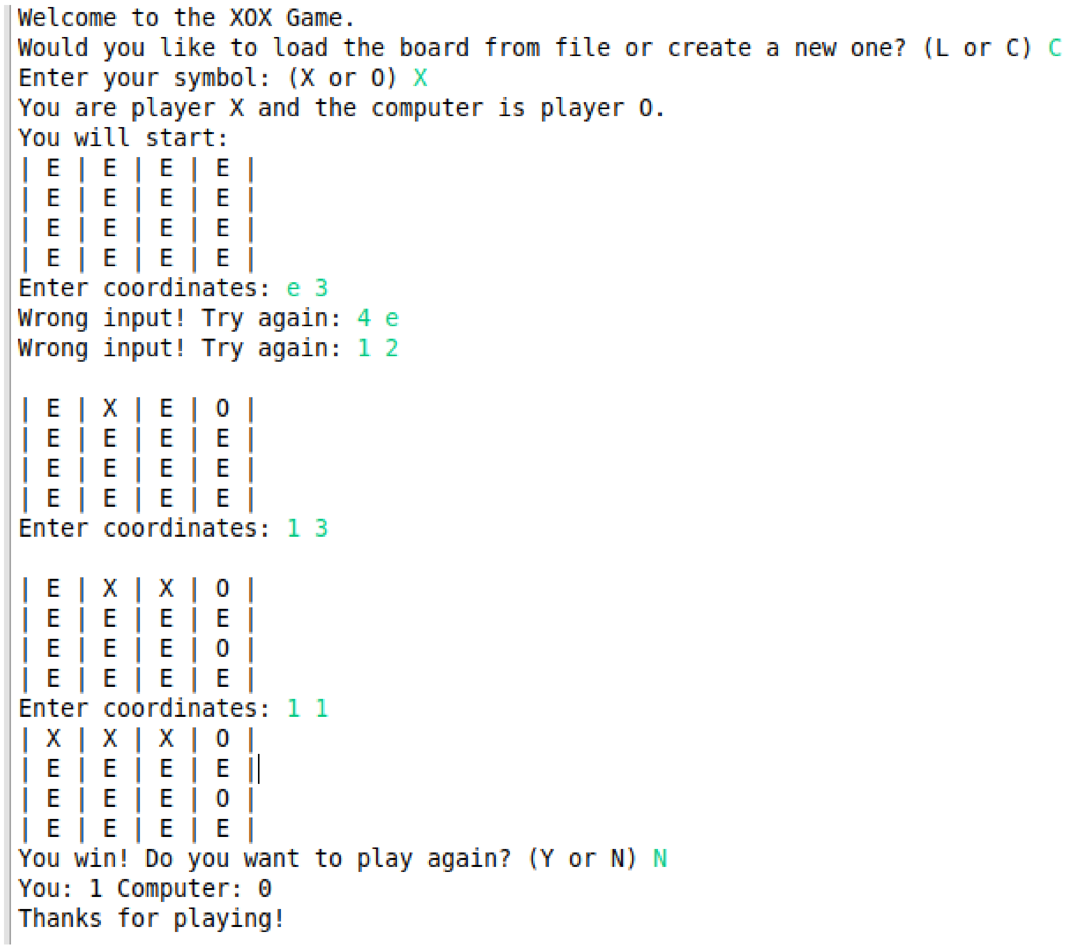
```
// This method checks if somebody won, horizontally.

// I based the method on indexes.

// if characters at indexes (0,1,2), (1,2,3), (4,5,6), (5,6,7), (8,9,10), (9,10,11), (12,13,14)
or (13,14,15) are same,

// and are not empty, this method returns true.

public static boolean horizontalCheck(String currentBoard) {

   for (int i=0; i<=13; i++) {

      if(i%4==0 || i%4==1) {

      if (currentBoard.charAt(i)==currentBoard.charAt(i+1) &&
currentBoard.charAt(i+1)==currentBoard.charAt(i+2) && !(currentBoard.charAt(i)=='E')) {

         return true;

      }}

   }

   return false;

}
```

```
// This method plays one move for the computer.
// This methods takes current board as parameter and returns the new board.
public static String computersPlay(String currentBoard) throws FileNotFoundException {


        Random r= new Random();
        int index= r.nextInt(16);
//as the board's length is 16, we assign a random integer between 0 and 15(inclusive)


        // if character at that index is E, which means that place is empty, we change the
current board.
        // if not, we assign a new index until we find an empty place.
        while     (!(currentBoard.charAt(index)=='E')) {
            index= r.nextInt(16);
        }


        // We do that by first substringing current board until that index, then adding the
computer's symbol
        // and than adding the rest part, starting from index+1, until the end.
        currentBoard= currentBoard.substring(0,index) + computerSymbol+
currentBoard.substring(index+1,16) ;


        return currentBoard;
    }
```

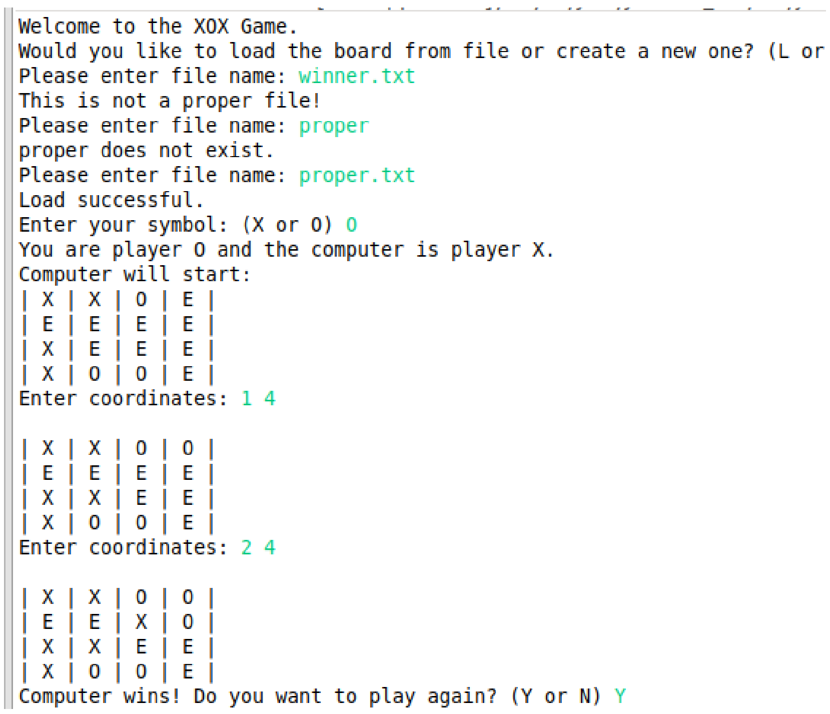
```
   // This method plays one move for the player.

   // This methods takes player's symbol and current board as parameter and returns the
new board.


   public static String playersPlay(String currentBoard, char playerSymbol) throws
FileNotFoundException {

       int x,y;

       x=-1;y=-1;

       Scanner console=new Scanner(System.in);

       System.out.print("Enter coordinates: ");

       // We ask the player to write to integers which can be 1,2,3,4.


       while(true) {                          //I started an indefinite while loop here.

          String line= console.nextLine();    //Firstly, i read the next line as a whole.

          Scanner read= new Scanner(line);    //I declare another scanner to read that line

          if(read.hasNextInt()) {           //If there is an int at the beginning,that is assigned to x

             x=read.nextInt();

             if (read.hasNextInt()) {        //Then we check if there is still an int

                y=read.nextInt();            //If so it is assigned to y

                break;                       //We break out.

             }

             else {                          //If there is not a second int, that input is wrong.


                System.out.print ("Wrong input! Try again: ");

                continue;                    // The while loop starts again.


             }


             }    System.out.print ("Wrong input! Try again: ");

                continue;                    //If there is not even a first int, the while loop starts again.

          }
```

```
      console=new Scanner(System.in);


      // If the player types integers other than these or if the character at the index of
that coordinates is not E

      // which means that place is not empty, we ask the user to type again until he/she
types a proper value.
    while( x<1 || x>4 || y<1 || y>4 || !(currentBoard.charAt((x-1)*4 + (y-1))=='E')) {
        System.out.print ("Wrong input! Try again: ");
        x=-1; y=-1;
        while(true) {        //This while loop is exactly the same with the upper one and has
the same purpose.
            String line= console.nextLine();
            Scanner read= new Scanner(line);
            if(read.hasNextInt()) {
                x=read.nextInt();
                if (read.hasNextInt()) {
                    y=read.nextInt();
                    break;
                }
                else {

                    System.out.print ("Wrong input! Try again: ");
                    continue;

                }

            }   System.out.print ("Wrong input! Try again: ");
                    continue;
            }
    }


    int index=(x-1)*4 + (y-1);
    // I turn the coordinates to index form by formula: (x-1)*4 + (y-1)

    // if character at that index is E, which means that place is empty, we change the
current board.

    // We do that by first substringing current board until that index, then adding the
player's symbol

    // and than adding the rest part, starting from index+1, until the end.


    currentBoard= currentBoard.substring(0,index) + playerSymbol +
currentBoard.substring(index+1,16);


    return currentBoard;


    }


}
```

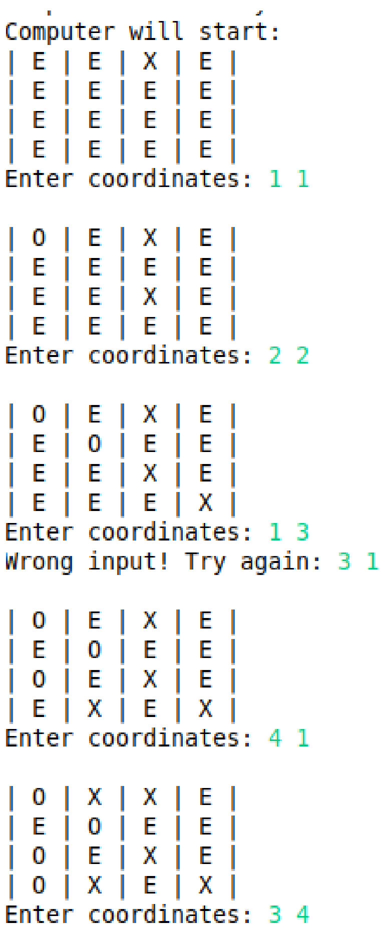
IV *OUTPUT OF THE PROGRAM*



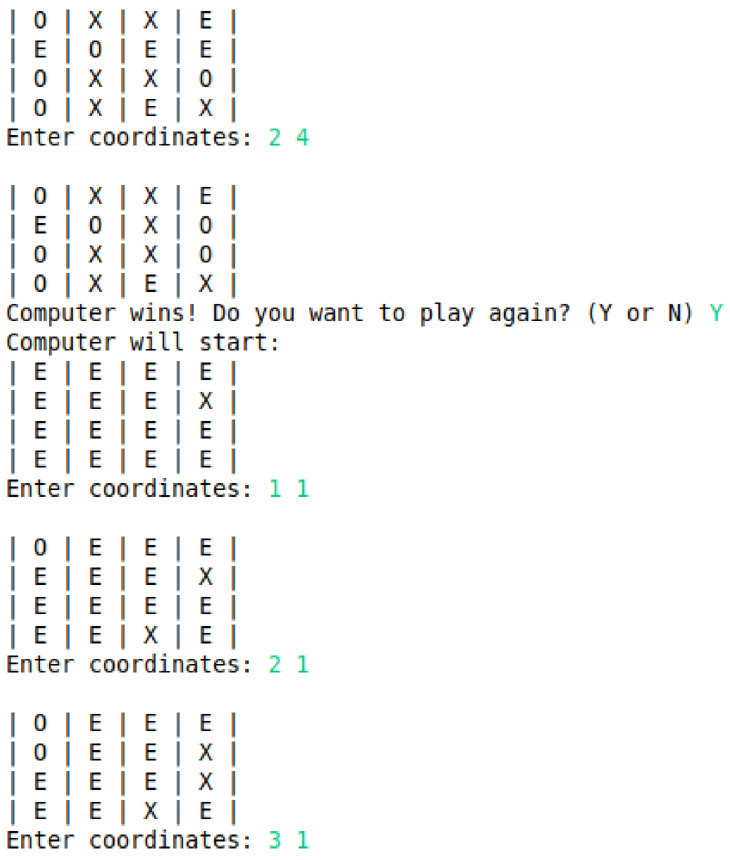
```
Welcome to the XOX Game.
Would you like to load the board from file or create a new one? (L or C) C
Enter your symbol: (X or O) X
You are player X and the computer is player O.
You will start:
| E | E | E | E |
| E | E | E | E |
| E | E | E | E |
| E | E | E | E |
Enter coordinates: e 3
Wrong input! Try again: 4 e
Wrong input! Try again: 1 2

| E | X | E | O |
| E | E | E | E |
| E | E | E | E |
| E | E | E | E |
Enter coordinates: 1 3

| E | X | X | O |
| E | E | E | E |
| E | E | E | O |
| E | E | E | E |
Enter coordinates: 1 1
| X | X | X | O |
| E | E | E | E |
| E | E | E | O |
| E | E | E | E |
You win! Do you want to play again? (Y or N) N
You: 1 Computer: 0
Thanks for playing!
```



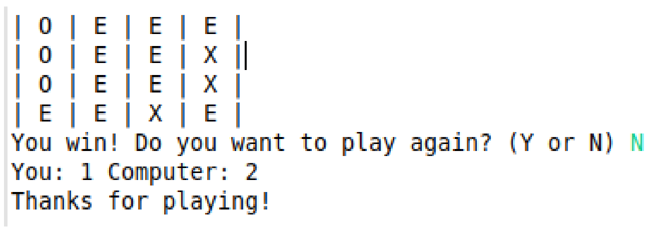
```
Welcome to the XOX Game.
Would you like to load the board from file or create a new one? (L or
Please enter file name: winner.txt
This is not a proper file!
Please enter file name: proper
proper does not exist.
Please enter file name: proper.txt
Load successful.
Enter your symbol: (X or O) O
You are player O and the computer is player X.
Computer will start:
| X | X | O | E |
| E | E | E | E |
| X | E | E | E |
| X | O | O | E |
Enter coordinates: 1 4

| X | X | O | O |
| E | E | E | E |
| X | X | E | E |
| X | O | O | E |
Enter coordinates: 2 4

| X | X | O | O |
| E | E | X | O |
| X | X | E | E |
| X | O | O | E |
Computer wins! Do you want to play again? (Y or N) Y
```



```
Computer will start:
| E | E | X | E |
| E | E | E | E |
| E | E | E | E |
| E | E | E | E |
Enter coordinates: 1 1

| O | E | X | E |
| E | E | E | E |
| E | E | X | E |
| E | E | E | E |
Enter coordinates: 2 2

| O | E | X | E |
| E | O | E | E |
| E | E | X | E |
| E | E | E | X |
Enter coordinates: 1 3
Wrong input! Try again: 3 1

| O | E | X | E |
| E | O | E | E |
| O | E | X | E |
| E | X | E | X |
Enter coordinates: 4 1

| O | X | X | E |
| E | O | E | E |
| O | E | X | E |
| O | X | E | X |
Enter coordinates: 3 4
```



```
| O | X | X | E |
| E | O | E | E |
| O | X | X | O |
| O | X | E | X |
Enter coordinates: 2 4

| O | X | X | E |
| E | O | X | O |
| O | X | X | O |
| O | X | E | X |
Computer wins! Do you want to play again? (Y or N) Y
Computer will start:
| E | E | E | E |
| E | E | E | X |
| E | E | E | E |
| E | E | E | E |
Enter coordinates: 1 1

| O | E | E | E |
| E | E | E | X |
| E | E | E | E |
| E | E | X | E |
Enter coordinates: 2 1

| O | E | E | E |
| O | E | E | X |
| E | E | E | X |
| E | E | X | E |
Enter coordinates: 3 1
```



```
| O | E | E | E |
| O | E | E | X |
| O | E | E | X |
| E | E | X | E |
You win! Do you want to play again? (Y or N) N
You: 1 Computer: 2
Thanks for playing!
```

V ***CONCLUSION***

I accomplished to solve the task properly. I believe writing many methods helped me a lot. The program runs properly, throwing error possibilities. The structure helps us to see the bigger picture clearly.