CMPE 150 Introduction to Computing **CMPE150.06**

Fall 2018 – Assignment 2 **08.12.2018**

Project Report

Problem Description

The project requires us to write a 4x4 sized tic-tac-toe game, which imports a board from a default file or a file specified by the player. The java program must check if the imported file is valid before importing it as a board. The java program must contain at least two static methods. The program must check every action of the player, see if they are valid.

Problem Solution

I created nine methods. In the main method the program asks the user if they want to start a new game of import a custom board. If they choose to import, my first method *importBoard* is triggered. This method asks the user which file they want to import, then it calls another method inside it: *isLegitBoard*. This method further investigates the file chosen.

isLegitBoard checks if the typed file exists. If it does, it goes on to check if the file contains any character other than "E", "O", "X", "|" or space, at the same time it counts how many lines the file has and checks if they are equal to four. After that, the file is read to a *String board* by another method: *readBoard*. This method simply turns the given file into a set of letters for easier use, eliminating spaces and "|". isLegitBoard counts how many "E"s, "X"s and "O"s there are in *board* and checks if they are in the logical range. Lastly, it checks if the game is already over in the given board, by checking the number of "E"s (empty cells), and by calling another method: *isWon*.

isWon is a boolean method that takes the current board and the symbol of the player or the computer, and checks all the conditions when a player has won, and if one of the conditions are met, returns true. If the imported board is such a case, *isLegitBoard* returns false.

When all the necessary conditions are met and the chosen board is valid, *importBoard* returns the chosen file. In the main method, the chosen file (or if user selected to create a new board, the initial board) is turned into a *String* by *readBoard* and held in a *String currentBoard* for further use.

After the board to be used is ready, the program asks the user symbol they want to use, X or O. In return, the opposite sign is chosen for the computer.

Then the game begins. If the game is started from a created board, the program determines who starts randomly. If a board is imported, it determines who starts by counting the number of “O” and “X” symbols there are in the *currentBoard*. In each turn, *playerTurn* or *computerTurn* is called, which of them is called is determined by how many turns passed. They both take *currentBoard* and the symbol of the player or the computer as parameters.

computerTurn chooses a spot on the board randomly, by selecting two integers randomly and after checking if the randomized coordinates are suitable, it returns a new *String currentBoard* with the changed coordinate.

playerTurn reads two integers from the player, checks if they are suitable for input, if they are, it returns a new *String currentBoard* with the changed coordinate.

When the game is over, the program checks if there is a winner. If there is, it announces who won. The program counts who won, and how many times they won the game. If there is no winner but the game is over (there is no empty cells left) the game announces that there is a tie.

The program asks the user if they want to play again. If they do, the game starts from a new board, skipping asking the user if they want to import a board. If the player doesn’t want to play again, the program prints how many times the player and the computer won, then the program is terminated.

■ Implementation

```
import java.util.*;

import java.io.*;

public class ENA2018400312 {

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

        // rounds is the number of rounds played, turns is the number of turns played in
        // the current round
        // userWon is the number of times the player won the game, computerWon is the
        // number of times the computer won the game
        // computerSymbol and userSymbol are the symbols that the player selects in the
        // beginning of the first game
        // userS and compS are to hold and not lose the values of the computerSymbol and
        // userSymbol Strings
        int rounds = 0, turns = 0, userWon = 0, computerWon = 0;
        String computerSymbol, userSymbol, userS = "", compS = "";
        Random r = new Random();
        Scanner console = new Scanner(System.in);

        do { // This while loop is to start again and again after every game is over until
                    // the player quits
            String currentBoard = "";// the board which the program will be using in the game, now empty but will be
                                                                            // filled further in the code

            // If it is the first round, the game will ask if the user wants to import a
            // board or start from a new game
            if (rounds == 0) {
                System.out.println("Welcome to the XOX Game.");
                System.out.print("Would you like to load the board from file or create a new game? (L or C) ");
                String answer = console.next().toUpperCase();
                // This loop makes sure the player entered a valid answer
                while (!answer.equals("L") && !answer.equals("C")) {
                    System.out.print("Please type L or C: ");
                    answer = console.next().toUpperCase();
                }
                // If the player wants to load, importBoard method is called, which returns a
                // file
                if (answer.equals("L")) {
                    Scanner board = new Scanner(importBoard());
                    currentBoard = readBoard(board).toUpperCase();
                    System.out.println("Load successful.");
                } else { // If the player wants to start a new game, the game starts from the empty board
                    currentBoard = " EEEEEEEEEEEEEEEE";
                }
            } else {
                // This statement will never be used, but required for the java program to work,
                // (it is required for the game to know currentBoard is initialized in every
                // condition) it means if the answer is nor L neither C, the game will start
                // from an empty board anyway
        currentBoard = " EEEEEEEEEEEEEEEE";
}
// If it is the first time the game is played, the game will ask the player to
// choose a symbol, until it gets a valid answer
if (rounds == 0) {
        System.out.print("Enter your symbol: (X or O) ");
        userSymbol = console.next().toUpperCase();
        while (!userSymbol.equals("X") && !userSymbol.equals("O")) {
                System.out.print("Incorrect input. Please type X or O. ");
                userSymbol = console.next().toUpperCase();
        }
        if (userSymbol.equals("X")) {
                computerSymbol = "O";
        } else {
                computerSymbol = "X";

        }
}
// If it's not the first time the game is played, the symbols are the same as
// the previous game's symbols, again this statement is here so that the java
// program works correctly, it knows that userSymbol and computerSymbol are
// initialized in every condition
else {
        userSymbol = userS;
        computerSymbol = compS;
}

// if the board is empty, not loaded from a file, a random side starts
if (currentBoard.equals(" EEEEEEEEEEEEEEEE"))
        turns += r.nextInt(2);
else {
        // If the board is imported, the program will count how many "X"s and "O"s there
        // are in the board and decide who should start first
        int x = 0, o = 0;
        for (int i = 1; i < currentBoard.length(); i++) {
                if (currentBoard.charAt(i) == 'O')
                        o++;
                if (currentBoard.charAt(i) == 'X')
                        x++;
        }
        if (x > o && userSymbol.equals("X")) {
                turns += 1;
        } else if (o > x && userSymbol.equals("O")) {
                turns += 1;
        } else if (o == x) {
                turns += r.nextInt(2);
        }

}

////////////////////////////////////////////

while (!(isWon(currentBoard, "X") || isWon(currentBoard, "O") || !(currentBoard.contains("E")))) {
        // checks if the game is over, by checking if there is a winner or the board has
        // empty cells (if not, there is a tie) every move
```

```
                        if (turns % 2 == 0) { // turns decides which side will play each turn
                                if (turns == 0) {// if it's the first turn, the game will tell who starts first
                                        System.out.println("You will start:");
                                        printCurrentBoard(currentBoard);
                                        turns += 2;
                                }
types                           currentBoard = playerTurn(currentBoard, userSymbol);// playerTurn method is called and player
                // coordinates
                        } else {
                                if (turns == 1)// if it's the first turn, the game will tell who starts first
                                        System.out.println("Computer will start:");
                                currentBoard = computerTurn(currentBoard, computerSymbol);// computer plays randomly
                                printCurrentBoard(currentBoard);
                        }
                        turns++;// after the computer or the player makes a move, turns increase

                }
                turns = 0;// after the game is over, turns is 0 again

                // Here, the winner announced. If there is no winner, the game will announce
                // that there is a tie. It will count the number of times the computer or the
                // player has won
                if (isWon(currentBoard, userSymbol)) {
                        printCurrentBoard(currentBoard);
                        System.out.print("You win! ");
                        userWon++;
                } else if (isWon(currentBoard, computerSymbol)) {
                        System.out.print("Computer wins! ");
                        computerWon++;
                } else {
                        System.out.println("It's a tie!");
                }

                rounds++;
                userS = userSymbol;
                compS = computerSymbol;
        } while (isStartAgain());// isStartAgain method is called, which asks the player if they want to start
                                                                                    // again

        // after all the rounds are over, the game shows how many times the player and
        // the computer has won
        System.out.println("You: " + userWon + " Computer: " + computerWon);
}

// This method takes a valid board file and turns it into a String, that way I
// have a easily-editable board.
// It eliminates all the spaces and "|"s
public static String readBoard(Scanner file) {
        String board = " "; // at first, the String the method will return is empty, is will be filled in
                                                                // the while loop

        // This loop reads the file until it has no lines left
        while (file.hasNextLine()) {
                String line = file.nextLine(); // line is the current line the Scanner reads
                Scanner lineReader = new Scanner(line); // A new scanner that reads the line at hand
                lineReader.next();
                // This loop reads the symbols in each cells, eliminating any other characters,
                // like "|"s
                for (int i = 0; i < 4; i++) {
                        board += lineReader.next();
                        lineReader.next();
                }
        }
        return board.toUpperCase(); // I make sure the board at hand is in uppercase
}

// If the user wants to import a board, this method is called. It asks for the
// file name, and deletes the spaces around it, if there are spaces. The method
// calls for isLegitBoard, which asks for a valid file name until it gets one.
// Then this method returns the correct file.
public static File importBoard() throws FileNotFoundException {
        Scanner console = new Scanner(System.in);
        System.out.print("Please enter the file name: ");

        String customBoard = console.nextLine();
        // This loop deletes the spaces that comes before the file name
        while (customBoard.startsWith(" ")) {
                customBoard = customBoard.substring(1);
        }
        // This loop deletes the spaces that comes after the file name
        while (customBoard.endsWith(" ")) {
                customBoard = customBoard.substring(0, customBoard.length() - 1);
        }
        // This loop asks for a legit file name until it gets one
        while (!isLegitBoard(customBoard)) {
                System.out.print("Please enter a different file name: ");
                customBoard = console.nextLine();
        }
        return new File(customBoard);
}

// This method checks if the chosen file is valid.
public static boolean isLegitBoard(String input) throws FileNotFoundException {

        // This loop checks if the chosen file name points to a valid file. If no such
        // file exists, the
        // method goes no further and returns false.
        if (!(new File(input)).exists()) {
                System.out.print("There is no such file. ");
                return false;
        }
        // This Scanner scans the file entered for further inspection
        Scanner inputFile = new Scanner(new File(input));

        int lineCount = 0; // lineCount is four counting how many lines the file has

        // This loop checks if every line in the file has a length of 17 characters and
        // every character in the imported board is one of the
        // following characters: E,X,O,| or space
        // If not, the method goes no further and returns false.
        while (inputFile.hasNextLine()) {
                String line = inputFile.nextLine().toUpperCase(); // line is the current line the Scanner is reading
                for (int i = 0; i < line.length(); i++) {
                        if (line.length() != 17 || !((line.charAt(i) != 'E' || line.charAt(i) != 'O' || line.charAt(i) != '|'
                                        || line.charAt(i) != ' '))) {
                                System.out.print("Bad board. ");
                                return false;
                        }
                }
                lineCount++;
        }
        // If the file has more than 4 lines, it is not recognized as a valid board and
        // the method returns false.
        if (lineCount != 4) {
                System.out.println("No board recognized. ");
                return false;
        }
        // If all until now is true, the file is turned into a String, eliminating
        // spaces and "|"s
        String board = readBoard(new Scanner(new File(input)));

        // These variables are there to count the number of every symbol in the board
        int numOfO = 0, numOfX = 0, numOfE = 0;

        // This loop counts how many of each symbol there are in the board and checks
        // again if there is anything other than X or E or O
        for (int i = 1; i < board.length(); i++) {
                if (board.toUpperCase().charAt(i) == 'E')
                        numOfE++;
                else if (board.toUpperCase().charAt(i) == 'O')
                        numOfO++;
                else if (board.toUpperCase().charAt(i) == 'X')
                        numOfX++;
                else {
                        System.out.print("Bad board. ");
                        return false;
                }

        }
        // This statements checks if the board has any empty cells, if not it decides
        // the game is over
        if (numOfE == 0) {
                System.out.print("Board is finished. ");
                return false;
        }
        // This statement checks if the total number of symbols are equal to 16, which
        // is the board size.
        // It also checks if the difference of X and O symbols are more than 1, if so,
        // it decides the board is invalid
        if (numOfO + numOfX + numOfE != 16 || (Math.abs(numOfX - numOfO) != 1 && Math.abs(numOfX - numOfO) != 0)) {
                System.out.print("Bad board. ");
                return false;
        }
        // This statement checks if the imported board already has a winner, then
```

```
        // rejects the board
        if (isWon(board, "X") || isWon(board, "O")) {
                System.out.println("Game already over. ");
                return false;
        }
        // If none of the conditions apply, then the board is valid and it is accepted
        // by returning true
        return true;
}

// This method takes the currentBoard string that holds the current board's data
// and prints it to the console.
public static void printCurrentBoard(String board) {
        for (int i = 1; i < board.length(); i = i + 4) {
                System.out.println("| " + board.charAt(i) + " | " + board.charAt(i + 1) + " | " + board.charAt(i + 2)
                                + " | " + board.charAt(i + 3) + " |");
        }
}

// This method is where the player changes the board.
public static String playerTurn(String board, String side) {
        Scanner input = new Scanner(System.in);

        board = board.toUpperCase(); // I make sure that the board at hand is in uppercase
        System.out.print("Enter coordinates: ");

        // This loop prompts the user to not type anything other than an integer
        while (!input.hasNextInt()) {
                input.next();
                System.out.print("Wrong input! Try again: ");
        }
        int y = input.nextInt();// y is the vertical coordinate of the board
        // This loop prompts the user to not type anything other than an integer
        while (!input.hasNextInt()) {
                input.next();
                System.out.print("Wrong input! Try again: ");
        }
        int x = input.nextInt();// x is the horizontal coordinate of the board

        // This loop checks if the given coordinates are valid. If the given coordinates
        // are outside of the board, or the cell is not empty, it prompts for another
        // input
        while (x < 1 || y < 1 || x > 4 || y > 4 || board.charAt((y - 1) * 4 + x) != 'E') {
                System.out.print("Wrong input! Try again: ");
                while (!input.hasNextInt()) {
                        input.next();
                        System.out.print("Wrong input! Try again: ");
                }
                y = input.nextInt();
                while (!input.hasNextInt()) {
                        input.next();
                        System.out.print("Wrong input! Try again: ");
                }
                x = input.nextInt();
        }
        // board is changed accordingly to the turn
        board = board.substring(0, (y - 1) * 4 + x) + side + board.substring((y - 1) * 4 + x + 1);
        return board;
}

// This method is where the computer changes the board.
public static String computerTurn(String board, String side) {
        // I created a random function to randomize the moves of the computer
        Random r = new Random();

        int y = r.nextInt(4) + 1;// y is the vertical coordinates of the board, between 1 and 4 inclusive
        int x = r.nextInt(4) + 1;// x is the horizontal coordinates of the board, between 1 and 4 inclusive

        // this loop checks if the randomized coordinates are occupied by a symbol
        while (board.charAt((y - 1) * 4 + x) != 'E') {
                y = r.nextInt(4) + 1;
                x = r.nextInt(4) + 1;
        }
        // The board is cahnged accordingly to the randomized coordinates
        board = board.substring(0, (y - 1) * 4 + x) + side + board.substring((y - 1) * 4 + x + 1);
        return board;
}

// This method asks if the user wants to play again
public static boolean isStartAgain() {
        Scanner console = new Scanner(System.in);
        System.out.print("Do you want to play again? (Y or N) ");
        String answer = console.next().toUpperCase();
        // This loop checks if the given answer is valid
        while (!answer.equals("Y") && !answer.equals("N")) {
                System.out.print("Please type Y or N: ");
                answer = console.next().toUpperCase();
        }
        // If the user types Y, the method returns true, if they type N, it returns
        // false
        if (answer.equals("Y"))
                return true;
        return false;

}

// This method checks if the game has a winner. I typed all the conditions that
// make the game over without a tie.
// The method returns if the given board is won by given player symbol.
public static boolean isWon(String board, String side) {
        board = board.toUpperCase(); // Again, I make sure the board at hand is in uppercase

        if (board.charAt(1) == side.charAt(0) && board.charAt(2) == side.charAt(0)
                        && board.charAt(3) == side.charAt(0)) {
                return true;
        }
        if (board.charAt(2) == side.charAt(0) && board.charAt(3) == side.charAt(0)
                        && board.charAt(4) == side.charAt(0)) {
                return true;
        }
```

```
if (board.charAt(5) == side.charAt(0) && board.charAt(6) == side.charAt(0)
                    && board.charAt(7) == side.charAt(0)) {
          return true;
}
if (board.charAt(6) == side.charAt(0) && board.charAt(7) == side.charAt(0)
                    && board.charAt(8) == side.charAt(0)) {
          return true;
}
if (board.charAt(9) == side.charAt(0) && board.charAt(10) == side.charAt(0)
                    && board.charAt(11) == side.charAt(0)) {
          return true;
}
if (board.charAt(10) == side.charAt(0) && board.charAt(11) == side.charAt(0)
                    && board.charAt(12) == side.charAt(0)) {
          return true;
}
if (board.charAt(13) == side.charAt(0) && board.charAt(14) == side.charAt(0)
                    && board.charAt(15) == side.charAt(0)) {
          return true;
}
if (board.charAt(14) == side.charAt(0) && board.charAt(15) == side.charAt(0)
                    && board.charAt(16) == side.charAt(0)) {
          return true;
}

if (board.charAt(1) == side.charAt(0) && board.charAt(5) == side.charAt(0)
                    && board.charAt(9) == side.charAt(0)) {
          return true;
}
if (board.charAt(2) == side.charAt(0) && board.charAt(6) == side.charAt(0)
                    && board.charAt(10) == side.charAt(0)) {
          return true;
}
if (board.charAt(4) == side.charAt(0) && board.charAt(8) == side.charAt(0)
                    && board.charAt(12) == side.charAt(0)) {
          return true;
}
if (board.charAt(8) == side.charAt(0) && board.charAt(12) == side.charAt(0)
                    && board.charAt(16) == side.charAt(0)) {
          return true;
}
if (board.charAt(1) == side.charAt(0) && board.charAt(6) == side.charAt(0)
                    && board.charAt(11) == side.charAt(0)) {
          return true;
}
if (board.charAt(6) == side.charAt(0) && board.charAt(11) == side.charAt(0)
                    && board.charAt(16) == side.charAt(0)) {
          return true;
}
if (board.charAt(2) == side.charAt(0) && board.charAt(7) == side.charAt(0)
                    && board.charAt(11) == side.charAt(0)) {
          return true;
}
if (board.charAt(5) == side.charAt(0) && board.charAt(10) == side.charAt(0)
                    && board.charAt(15) == side.charAt(0)) {
```

```
                return true;
        }

        if (board.charAt(4) == side.charAt(0) && board.charAt(7) == side.charAt(0)
                        && board.charAt(10) == side.charAt(0)) {
                return true;
        }
        if (board.charAt(7) == side.charAt(0) && board.charAt(10) == side.charAt(0)
                        && board.charAt(13) == side.charAt(0)) {
                return true;
        }
        if (board.charAt(3) == side.charAt(0) && board.charAt(6) == side.charAt(0)
                        && board.charAt(9) == side.charAt(0)) {
                return true;
        }
        if (board.charAt(8) == side.charAt(0) && board.charAt(11) == side.charAt(0)
                        && board.charAt(14) == side.charAt(0)) {
                return true;
        }
        return false;

    }

}
```

Output of the Program

Example Run 1:

```
Welcome to the XOX Game.
Would you like to load the board from
file or create a new game? (L or C) c
Enter your symbol: (X or O) x
Computer will start:
| E | E | E | E |
| E | E | E | E |
| E | E | O | E |
| E | E | E | E |
Enter coordinates: 1 1
| X | E | E | E |
| E | O | E | E |
| E | E | O | E |
| E | E | E | E |
Enter coordinates: 1 2
| X | X | E | E |
| E | O | O | E |
| E | E | O | E |
| E | E | E | E |
Enter coordinates: 1 3
| X | X | X | E |
| E | O | O | E |
| E | E | O | E |
| E | E | E | E |
You win! Do you want to play again? (Y
or N) y
Computer will start:
| E | E | E | E |
| E | E | E | E |
| E | E | E | E |
| O | E | E | E |
Enter coordinates: e
Wrong input! Try again: 5 6
Wrong input! Try again: 4 2
| E | E | E | E |
| E | E | E | E |
| E | E | E | O |
| O | X | E | E |
Enter coordinates: 4 3
| E | E | O | E |
| E | E | E | E |
| E | E | E | O |
| O | X | X | E |
Enter coordinates: 1 1
| X | E | O | E |
| E | E | E | E |
| O | E | E | O |
| O | X | X | E |
Enter coordinates: 1 2
| X | X | O | E |
| E | O | E | E |
| O | E | E | O |
| O | X | X | E |
Computer wins! Do you want to play
again? (Y or N) n
You: 1 Computer: 1
```

Example Run 2:

```
Welcome to the XOX Game.
Would you like to load the board from
file or create a new game? (L or C) l
Please enter the file name:
customBoard.txt
Load successful.
Enter your symbol: (X or O) o
You will start:
| O | X | O | X |
| O | X | O | X |
| X | O | X | O |
| X | O | X | E |
Enter coordinates: 4 4
It's a tie!
Do you want to play again? (Y or N) n
You: 0 Computer: 0
```

■ Conclusion

As seen from the output, the game works as intended, and handles the exceptions well. One major flaw of the program is that the moves of the computer are randomized. The game could be improved by implementing a better *computerTurn* method, which examines the board and finds the best move.

Other than the problem mentioned above, the program works correctly in every possible situation.