



ASSIGNMENT 3

PROJECT REPORT

Outline

This report has been prepared to be a delicate explanation of assignment 3.
In this assignment, I was requested to create a board game.

İbrahim Özgürçan ÖZTAŞ
oztas.ozgurcan@gmail.com

Context:

- I. Project Description
- II. Project Solution
- III. Implementation
- IV. Output Illustration
- V. Conclusion

I. Problem Description:

In this assignment, we were asked to create a program that creates a game board and play with its rules. The game is simply a game similar to 'XOX' and there are 1 player and 1 computer playing against each other. But, different from 'XOX', there are 16 different pieces designed by 4 set of feature namely black or white, tall or short, round or square, and hollow or solid. Each piece can be played once. From the very beginning, computer picks a piece for player and gives it to player. Then, player puts the piece wherever he/she wants to play. Furthermore, player picks a piece for computer and gives it to computer. And computer puts the piece random available position. Then the game continues. If at a row, coloumn or diagonal; one of each character of 4 piece is same respectively, one who puts the last piece wins.

There is an option set that asks whether a player wants to start with a new game or continue from the previous game. Then at the end of each turn, the game is saved to a file namely "input.txt". And the game continues as written above.

II. Problem Solution:

This program uses several methods to reach out a solution.

a. `isNew(Scanner console)` method includes a introduction and asks user whether user wants to start a new game or not. By checking what user've typed, the method returns boolean value to its caller method.

b. `boardCreation()` is created to solve the issue when there is no `input.txt` file in the same directory of the program. This method creates a new file and new board and returns the board to its caller method.

c. `board()` is the board creator method. It returns a `String[][]` array matrix to its caller method.

d. `emptyBoard(String[][] gameTable)` fills the parameter matrix with a `String "EEEE"`.

e. `readBoard(String[][] gameTable, Scanner input)` reads the board from the "input.txt" file and sync each cell to the next token in the file.

f. `printBoard(String[][] gameTable)` prints the board to the Screen.

g. `writeBoard(String[][] gameTable)` gathers the data from the board and writes the matrix to the "input.txt".

h. `userTurn(String[][] gameTable, Scanner console, Random rand)` is the combination of all necessary methods for the user's turn. It summons `aiPiece(Random rand)`, `isOnBoard(String piece, String[][] gameTable)`, `printBoard(String[][] gameTable)`, `userCoord(Scanner console)`, `writeBoard(String[][] gameTable)` methods and executes its assignments.

i. `aiTurn(String[][] gameTable, Scanner console, Random rand)` is the combination method of all needed methods. It summons `printBoard(String[][] gameTable)`, `piecePrinter(String[][] gameTable)`, `userPick(Scanner console)`, `isOnBoard(String aiPiece, String[][] gameTable)`, `aiCoord(String[][] gameTable, Random rand)`, `isFull(String[][] gameTable, aiCoordinate)`, `writeBoard(String[][] gameTable)` methods and complete their tasks.

j. `aiPiece(Random rand)` randomly decides the piece for player's turn and returns the piece to its calling method.

k. `aiCoord(String[][] gameTable, Random rand)` randomly creates coordinates for computer's turn and returns its values to the caller method.

l. `isWin(String[][] gameTable)` checks if there is a winning condition for the board.

m. `isDrawStart(String[][] gameTable)` checks if there is no place to play and returns it to its caller method.

n. `whoWins(String[][] gameTable)` checks who wins and returns true or false to its caller method.

o. `userCoord(Scanner console)` gathers coordinates from user and returns an array as a coordinate.

p. `userPick(Scanner console)` gathers the piece for computer to put it on the board. It returns a string to its caller method.

r. `isFull(String[][] gameTable, int[] coordinate)` checks if the coordinates are empty or not and returns a boolean value to its caller method.

s. `isOnBoard(String piece, String[][] gameTable)` checks if the piece is on the board and returns a boolean value to its caller method.

t. `piecePrinter(String[][] gameTable)` prints the available pieces for the user.

u. `whoseTurn(String[][] gameTable)` decides whose turn is next turn and returns a boolean value for its calling method. If it returns true, this means user played his/her turn and waits for computer to play. Else is the vice versa.

III. Implementation:

```
import java.io.*;

import java.util.*;

public class 002016400198 {

    public static void main(String[] args) throws FileNotFoundException,
    InterruptedException, IOException{
        Scanner console = new Scanner(System.in);

        Random rand = new Random();
        String[] arr = new String[16];
        arr[0] = "BTSH"; arr[1] = "BTSS"; arr[2] = "BTRH"; arr[3] = "BTRS";
arr[4] = "BSSH"; arr[5] = "BSSS"; arr[6] = "BSRH"; arr[7] = "BSRS";
        arr[8] = "WTSH"; arr[9] = "WTSS"; arr[10] = "WTRH"; arr[11] = "WTRS";
arr[12] = "WSSH"; arr[13] = "WSSS"; arr[14] = "WSRH"; arr[15] = "WSRS";
        String[][] gameTable = board();
        if(isNew(console)){
            emptyBoard(gameTable);
            Thread.sleep(2500);
            do{
                userTurn(gameTable,console,rand);
                if(isWin(gameTable)) break;
                aiTurn(gameTable,console,rand);
            }while(!isWin(gameTable) && !isDrawStart(gameTable));
        }
        else{
            File f = new File("input.txt");
            boolean fvar = f.createNewFile();
            if(fvar){
                Thread.sleep(2500);
                System.out.println("There was no previous game.");
                System.out.println("A new file has already been created
for you.");

                Thread.sleep(2500);
                emptyBoard(gameTable);
                writeBoard(gameTable);
            }
            else{
                Thread.sleep(2500);
            }
        }
    }
}
```

```

        System.out.println("Playing from last saved one...");
    }

    Scanner input = new Scanner(new File("input.txt"));
    readBoard(gameTable, input);
    while(!isWin(gameTable) && !isDrawStart(gameTable)){
        if(whoseTurn(gameTable)){
            aiTurn(gameTable, console, rand);
            if(isWin(gameTable)) break;
            else if(isDrawStart(gameTable)){
                System.out.println("There was a draw. No
one Wins.");
                return;
            }
            userTurn(gameTable, console, rand);
        }
        else if(!whoseTurn(gameTable)){
            userTurn(gameTable, console, rand);
            if(isWin(gameTable)) break;
            else if(isDrawStart(gameTable)){
                System.out.println("There was a draw. No
one Wins.");
            }
            aiTurn(gameTable, console, rand);
        }
    }
    }
    printBoard(gameTable);
    String q = whoWins(gameTable);
    System.out.println("The winner is: ");
    System.out.println(q);

}

// this method asks whether or not a new game is requested.

public static boolean isNew(Scanner console){
    System.out.println("Welcome to the F.A.T.E.");
    System.out.println("aka (Fantasy Algorithm Training Exercise)");
    System.out.println("To start with a new game, type new.");
    System.out.println("To continue from a previous save, type
previous.");
    String a = console.nextLine();
    while(!a.equalsIgnoreCase("new") && !a.equalsIgnoreCase("previous")){
        System.out.println("Invalid input. Enter a valid one.");
        a = console.nextLine();
    }
    if(a.equalsIgnoreCase("new")){
        return true;
    }
    else{
        return false;
    }
}
}

```

```
// this method is used to execute a board in a file. It was used only one
condition which is
// the file does not exists and a new file is created.
```

```
public static String[][] boardCreation() throws FileNotFoundException{
    PrintStream Ps = new PrintStream("input.txt");
    String[][] newBoard = new String[4][4];
    emptyBoard(newBoard);
    for(int i=0;i<newBoard.length;i++){
        for(int j = 0;j<newBoard.length;j++){
            Ps.print(newBoard[i][j] + "\t");
        }
        Ps.println();
    }
    Ps.close();
    return newBoard;
}
```

```
// this method creates the 4 x 4 matrix as a gameBoard.
```

```
public static String[][] board(){
    String[][] board = new String[4][4];
    return board;
}
```

```
// this method fills the board with empty strings.
```

```
public static void emptyBoard(String[][] board){
    String[][] boardprocess = board;
    for(int i=0;i<4;i++){
        for(int j=0;j<4;j++){
            boardprocess[i][j] = "EEEE";
        }
    }
}
```

```
// this method reads the board from the file namely "input.txt"
```

```
public static void readBoard(String[][] gameTable, Scanner input){
    for(int i=0;i<gameTable.length;i++){
        for(int j=0;j<gameTable[i].length;j++){
            if(input.hasNext())
                gameTable[i][j] = input.next();
        }
    }
}
```

```
// this method prints the board to the console.
```

```
public static void printBoard (String[][] board){
    System.out.println("\t0\t1\t2\t3");
    for(int i=0;i<board.length;i++){
        System.out.print(i + "\t");
        for(int j=0;j<board[i].length;j++){
            System.out.print(board[i][j] + "\t");
        }
        System.out.println();
    }
}
```

```

    }

    // this method writes the board to the input.txt file.

    public static void writeBoard(String[][] board) throws
FileNotFoundException{
        PrintStream ps = new PrintStream("input.txt");
        for(int i=0;i<board.length;i++){
            for(int j=0;j<board.length;j++){
                ps.print(board[i][j] + "\t");
            }
            ps.println();
        }
        ps.close();
    }

    // this method is the uniter of the methods for user.

    public static void userTurn(String[][] gameTable,Scanner console,Random
rand) throws InterruptedException,FileNotFoundException{
        String piece = aiPiece(rand);
        while(isOnBoard(piece, gameTable)){
            piece = aiPiece(rand);
        }
        printBoard(gameTable);
        System.out.println("Your piece is " + piece + ". Put it anywhere on
board available.");
        System.out.println("Be careful of the coordinates.");
        int[] coordinate = userCoord(console);
        while(isFull(gameTable, coordinate)){
            System.out.println("This location is not available.");
            System.out.println("Enter a valid location.");
            coordinate = userCoord(console);
        }
        gameTable[coordinate[0]][coordinate[1]] = piece;
        writeBoard(gameTable);
        System.out.println("Next turn. Please wait...\n");
        int constant = rand.nextInt(3)+1;
        int sleeptime = constant * 1000;
        Thread.sleep(sleeptime);
    }

    // this method is the uniter of the methods for aI.

    public static void aiTurn(String[][] gameTable,Scanner console,Random rand)
throws InterruptedException,FileNotFoundException{
        int constant = rand.nextInt(3)+1;
        int sleeptime = constant * 1000;
        printBoard(gameTable);
        System.out.println("Select a piece for AI to play.");
        piecePrinter(gameTable);
        String aiPiece = userPick(console);
        while(isOnBoard(aiPiece, gameTable)){
            System.out.println("This piece is already placed on board.");
            System.out.println("Enter a nonplayed one.");
            aiPiece = userPick(console);
        }
    }

```



```

        int[] aiCoordinate = aiCoord(gameTable,rand);
        while(isFull(gameTable,aiCoordinate)){
            aiCoordinate = aiCoord(gameTable,rand);
        }
        gameTable[aiCoordinate[0]][aiCoordinate[1]] = aiPiece;
        writeBoard(gameTable);
        System.out.println("Next turn. Please wait...\n");
        constant = rand.nextInt(4)+1;
        Thread.sleep(sleeptime);
    }
    // this method picks a piece for user.

    public static String aiPiece(Random rand){
        String[] arr = new String[16];
        arr[0] = "BTSH"; arr[1] = "BTSS"; arr[2] = "BTRH"; arr[3] = "BTRS";
arr[4] = "BSSH"; arr[5] = "BSSS"; arr[6] = "BSRH"; arr[7] = "BSRS";
        arr[8] = "WTSH"; arr[9] = "WTSS"; arr[10] = "WTRH"; arr[11] = "WTRS";
arr[12] = "WSSH"; arr[13] = "WSSS"; arr[14] = "WSRH"; arr[15] = "WSRS";
        return arr[rand.nextInt(16)];
    }

    // this method picks a coordinate binary for AI.

    public static int[] aiCoord(String[][] board,Random rand){

        int[] coord = new int[2];
        do{
            coord[0]=rand.nextInt(4);
            coord[1]=rand.nextInt(4);
        }
        while(!board[coord[0]][coord[1]].equals("EEEE"));
        return coord;
    }

    // this method returns the actual winning condition.

    public static boolean isWin(String[][] board){
        boolean flagx = true;
        boolean flagy = true;
        boolean flagd1 = true;
        boolean flagd2 = true;
        for(int index = 0;index<4;index++){
            flagd1 = true;
            flagd2 = true;
            for(int i=0;i<board.length;i++){
                flagx = true;
                flagy = true;
                for(int j=0;j<board.length;j++){

                    if(board[i][j].charAt(index)!=board[i][0].charAt(index) ||
board[i][j].charAt(index)=='E'){

                        flagx = false;
                    }

                    if(board[j][i].charAt(index)!=board[0][i].charAt(index) ||
board[j][i].charAt(index)=='E'){

```

```

        flagy = false;
    }
}
if(flagx==true || flagy == true) return flagx||flagy;
if(board[i][i].charAt(index)!=board[0][0].charAt(index)
|| board[i][i].charAt(index)=='E'){
    flagd1 = false;
}
if(board[board.length-1-
i][i].charAt(index)!=board[3][0].charAt(index) || board[board.length-1-
i][i].charAt(index)=='E'){
    flagd2 = false;
}
}
if(flagd1 == true || flagd2 == true) return flagd1||flagd2;
}
return false;
}

```

// this method checks whether there is a draw condition.

```

public static boolean isDrawStart(String[][] board){
    int eCount = 16;
    for(int i=0;i<board.length;i++){
        for(int j=0;j<board.length;j++){
            if(!board[i][j].equals("EEEE")){
                eCount--;
            }
        }
    }
    if(eCount==0){
        return true;
    }
    else{
        return false;
    }
}

```

// this method decides who is winner.

```

public static String whoWins(String[][] board){

    int countPiece = 0;
    for(int i=0;i<board.length;i++){
        for(int j=0;j<board[i].length;j++){
            if(!board[i][j].equals("EEEE")) countPiece++;
        }
    }

    if(countPiece%2==1) return "User";
    else return "AI";
}

```

// this method gathers the coordinates of user from Scanner console.

```

public static int[] userCoord(Scanner console){
    System.out.println("Enter the coordinates.");

    String coord = console.nextLine();
}

```

```

        while(coord.length()!=3 || coord.charAt(1)!=' ' ||
(coord.charAt(0)!='0'
        && coord.charAt(0)!='1' && coord.charAt(0)!='2' &&
coord.charAt(0)!='3')
        || (coord.charAt(2)!='0' && coord.charAt(2)!='1' &&
coord.charAt(2)!='2'
        && coord.charAt(2)!='3')){
            System.out.println("Invalid input. Type again.");
            coord = console.nextLine();
        }
        String x = coord.substring(0, 1);
        int i = Integer.parseInt(x);
        String y = coord.substring(2);
        int j = Integer.parseInt(y);
        int[] coord = {i,j};
        return coord;
    }

    // this method gathers the piece for AI.

    public static String userPick(Scanner console){

        System.out.println("Enter the piece.");
        String piece = console.nextLine().toUpperCase();
        String[] arrU = new String[16];
        arrU[0] = "BTSH"; arrU[1] = "BTSS"; arrU[2] = "BTRH"; arrU[3] =
"BTRS"; arrU[4] = "BSSH"; arrU[5] = "BSSS"; arrU[6] = "BSRH"; arrU[7] = "BSRS";
        arrU[8] = "WTSH"; arrU[9] = "WTSS"; arrU[10] = "WTRH"; arrU[11] =
"WTRS"; arrU[12] = "WSSH"; arrU[13] = "WSSS"; arrU[14] = "WSRH"; arrU[15] =
"WSRS";

        boolean flag = false;
        while(!flag){
            for(int i=0;i<arrU.length;i++){
                if(piece.equalsIgnoreCase(arrU[i])){
                    flag = true;
                }
            }
            if(!flag){
                System.out.println("Invalid input. Enter a new one.");
                piece = console.nextLine().toUpperCase();
            }
        }
        return piece;
    }

    // this method checks whether the place is full.

    public static boolean isFull(String[][] board, int[] coordinate){
        int x = coordinate[0];
        int y = coordinate[1];
        if(!board[x][y].equals("EEEE")){
            return true;
        }
        else{
            return false;
        }
    }
}

```

```

// this method checks whether or not piece is on board.

public static boolean isOnBoard(String piece, String[][] board){
    for(int i=0;i<board.length;i++){
        for(int j=0;j<board[i].length;j++){
            if(board[i][j].equals(piece)){
                return true;
            }
        }
    }
    return false;
}

// this method prints the available pieces of the game.

public static void piecePrinter(String[][] board){
    String[] arr = new String[16];
    arr[0] = "BTSH"; arr[1] = "BTSS"; arr[2] = "BTRH"; arr[3] = "BTRS";
arr[4] = "BSSH"; arr[5] = "BSSS"; arr[6] = "BSRH"; arr[7] = "BSRS";
    arr[8] = "WTSH"; arr[9] = "WTSS"; arr[10] = "WTRH"; arr[11] = "WTRS";
arr[12] = "WSSH"; arr[13] = "WSSS"; arr[14] = "WSRH"; arr[15] = "WSRS";
    for(int i=0;i<board.length;i++){
        for(int j=0;j<board[i].length;j++){
            for(int q = 0;q<16;q++){
                if(board[i][j].equals(arr[q])){
                    arr[q] = "EEEE";
                }
            }
        }
    }
    System.out.println("Available pieces : ");
    for(int k = 0;k<16;k++){
        if(!arr[k].equals("EEEE")){
            System.out.print(arr[k] + " ");
        }
    }
    System.out.println();
}

// this method decides whose turn is.

public static boolean whoseTurn(String[][] board){
    int countTurn = 0;
    for(int i=0;i<board.length;i++){
        for(int j=0;j<board[i].length;j++){
            if(!board[i][j].equals("EEEE")){
                countTurn++;
            }
        }
    }
    if(countTurn%2==1){
        return true;
    }
    else{
        return false;
    }
}
}

```

IV. Output of the program:

```
Eclipse Workspace - Java - assignment 3/src/OO2016400198/java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

OO2016400198 (1) [Java Application] C:\Program Files\Java\jre1.8.0_112\bin\javaw.exe (15 Ara 2017 02:45:46)

Welcome to the F.A.T.E.
aka (Fantasy Algorithm Training Exercise)
To start with a new game, type new.
To continue from a previous save, type previous.
new
  0      1      2      3
0  EEEE EEEE EEEE EEEE
1  EEEE EEEE EEEE EEEE
2  EEEE EEEE EEEE EEEE
3  EEEE EEEE EEEE EEEE
Your piece is WSSS. Put it anywhere on board available.
Be careful of the coordinates.
Enter the coordinates.
0 0
Next turn. Please wait...
  0      1      2      3
0  WSSS EEEE EEEE EEEE
1  EEEE EEEE EEEE EEEE
2  EEEE EEEE EEEE EEEE
3  EEEE EEEE EEEE EEEE
Select a piece for AI to play.
Available pieces :
BTSH BTSS BTRH BTRS BSSH BSSS BSRH BSRS WSSH WSRH WTRS
Enter the piece.
btsh
Next turn. Please wait...
  0      1      2      3
0  WSSS EEEE EEEE EEEE
1  EEEE EEEE EEEE EEEE
2  EEEE EEEE EEEE EEEE
3  EEEE EEEE BTSH EEEE
Your piece is WSRH. Put it anywhere on board available.
Be careful of the coordinates.
Enter the coordinates.
1 1
Next turn. Please wait...
  0      1      2      3
0  WSSS EEEE EEEE EEEE
1  EEEE WSRH EEEE EEEE
2  EEEE EEEE EEEE EEEE
3  EEEE EEEE EEEE EEEE
```

```
Eclipse Workspace - Java - assignment 3/src/OO2016400198/java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

terminated - OO2016400198 (1) [Java Application] C:\Program Files\Java\jre1.8.0_112\bin\javaw.exe (15 Ara 2017 02:45:46)

  0      1      2      3
0  WSSS EEEE BTRS EEEE
1  EEEE WSRH EEEE EEEE
2  EEEE EEEE WSRH EEEE
3  EEEE EEEE BTSH EEEE
Select a piece for AI to play.
Available pieces :
BTSS BTRH BSSH BSSS BSRH BSRS WSSH WSRH WTRS WTSR
Enter the piece.
btss
Next turn. Please wait...
  0      1      2      3
0  WSSS EEEE BTRS EEEE
1  EEEE WSRH EEEE EEEE
2  EEEE EEEE WSRH EEEE
3  EEEE BTSS BTSH EEEE
Your piece is BSRS. Put it anywhere on board available.
Be careful of the coordinates.
Enter the coordinates.
2 1
Next turn. Please wait...
  0      1      2      3
0  WSSS EEEE BTRS EEEE
1  EEEE WSRH EEEE EEEE
2  EEEE BSRS WSRH EEEE
3  EEEE BTSS BTSH EEEE
Select a piece for AI to play.
Available pieces :
BTRH BSSH BSSS BSRH WSSH WTSR WTRS WTSR
Enter the piece.
bssr
Next turn. Please wait...
  0      1      2      3
0  WSSS EEEE BTRS EEEE
1  EEEE WSRH EEEE EEEE
2  EEEE BSRS WSRH EEEE
3  EEEE BTSS BTSH BSSS
The winner is:
AI
```

V. Conclusion

My program to this problem solves the problem. At the process, I encountered with a hardship which is the algorithm of isWin() method is hard to configure. But with a deep look into the logic of 'XOX', I've finally devised the method.