Ömer Faruk Özdemir 2016400048

1)Problem Description:

We need to develop a two-player board game. The board contains 16 pieces that are organized in a 4\*4 square. Each piece can be either black or white, tall or short, square or round, and hollow or solid. The game starts with an empty board. Players take turns in playing. At each round, a player picks a piece and gives it to the other player, who then places it on the board. A player wins the game after a move if there are four pieces that share one property horizontally, vertically, or diagonally. We can name each piece with the four properties such that “BTSH” means a black, tall, square, and hollow.

2)Problem Solution:

In total 28 for loops has been used.

BotsPlay(): Created an extra. Bots play with each other 1000 matches.

PlayerPick(): Player picks a suitable piece according to regulations. Method returns it.

PlayerPut(String): Player puts the given piece to table according to regulations.

BotPut(String): Computer puts the given piece to table according to regulations. Looks to all empty spaces. İf there is a winning move plays it. Else puts to a random location.

BotPick():Bot picks a random piece with respect to rules. İf piece has a winning move picks another random piece. Tries this 10000 times. İf there is no piece like that it returns a random piece.

AnyWinningMove(String): Puts the piece to each empty location, by copying table to TestTable, checks if the game is won or not. Makes Coordinate[0]=-1 if there is no winning move, if there is a winning move saves coordinates to Coordinate[]

RandomPiece(): Returns a random piece. By RandomNumber%2 decides each property.

CoordinateEmpty(array, int , int ): Returns true if CoordinateEmpty. Checks by that coordinate's first letter equals "E" or not.

PieceNotAvailable(String): Checks if the given String is one of game pieces. Returns true if it is not.

PieceExistsInTable(String): Returns true if given piece equals one of pieces in the table.

UpdateTestTable(): Equals TestTable[] to values of table[].

CheckWin(Array): Returns true if given array satisfies the winning condition.

RestartTable(): Equals table to empty one.

ReadTable(): Asks user to load or not. İf yes load from input.txt. Else Restarts table.

DrawTable(): Draws table to console.

SaveGame(): Writes game to input.txt. First prints turn number then table.

3)İmplementation:

**import** java.util.\*;  
**import** java.io.\*;  
  
**public class** OFO2016400048 {  
 **static** Scanner *console* = **new** Scanner(System.***in***);*//Scanner in whole code* **static** String[][][] *table* = **new** String[4][4][4];*//Game table* **static** String[][][] *TestTable* = **new** String[4][4][4];*//Test table where bot tries possibilities* **static** String[] *Piece* = **new** String[4];*//Used for taking letters from it* **static int** *TurnNumber* = 0;*//How many pieces have been put* **static** Random *rand* = **new** Random();*//Generates a number randomly* **static** String[] *letter* = **new** String[4];*//For converting random number to letter* **static int**[] *Coordinate* = **new int**[2];*//For returning coordinate* **public static void** main(String[] args) **throws** FileNotFoundException {  
 *BotsPlays*(); *//BONUS 2 bots plays 1000 match.  
 TurnNumber*=0;  
 *ReadTable*();  
 String CurrentPiece;  
 **while** (*TurnNumber* < 16) {  
 **if** (*TurnNumber* % 2 == 0) { *//Checks if it is computer's turn or not by TurnNumber  
 DrawTable*();  
 CurrentPiece = *BotPick*();  
 *AnyWinningMove*(CurrentPiece);  
 *PlayerPut*(CurrentPiece);  
 *TurnNumber*++;  
 }  
 *DrawTable*();  
 **if** (*CheckWin*(*table*)) {  
 System.***out***.println(**"You won"**);  
 **break**;  
 }  
 *SaveGame*();  
 CurrentPiece = *PlayerPick*();  
 *BotPut*(CurrentPiece);  
 *DrawTable*();  
 *TurnNumber*++;  
 **if** (*CheckWin*(*table*)) {  
 System.***out***.println(**"You lost"**);  
 **break**;  
 }  
 *SaveGame*();  
 }  
 }  
  
 *//BONUS:: 2 bots play with each other* **public static void** BotsPlays() {  
 **int** Matches = 0;  
 **int** Bot1Win = 0;  
 **int** Bot2Win = 0;  
 **while** (Matches < 1000) {  
 *RestartTable*();  
  
 *TurnNumber* = 0;  
 String CurrentPiece;  
 **while** (*TurnNumber* < 16) {  
 **if** (*TurnNumber* % 2 == 0) { *//Checks if it is computer's turn or not by TurnNumber* CurrentPiece = *BotPick*();  
 *BotPut*(CurrentPiece);  
 *TurnNumber*++;  
 }  
 **if** (*CheckWin*(*table*)) {  
 *DrawTable*();  
 Bot1Win++;  
 **break**;  
 }  
 CurrentPiece = *BotPick*();  
 *BotPut*(CurrentPiece);  
 *TurnNumber*++;  
 **if** (*CheckWin*(*table*)) {  
 *DrawTable*();  
 Bot2Win++;  
 **break**;  
  
 }  
 }  
 Matches++;  
 }  
 System.***out***.println(**"Bot1: "** + Bot1Win + **"\nBot2: "** + Bot2Win);  
 }  
  
 *//Returns player's piece pick if it is pickable.* **public static** String PlayerPick() {  
 System.***out***.println(**"Please pick a piece for your opponent to put"**);  
 String piece = *console*.next();  
 **while** ((*PieceExistsInTable*(piece) || *PieceNotAvailable*(piece) || piece.length() > 4)) {  
 System.***out***.println(**"Wrong İnput!\nPlease enter another input"**);  
 piece = *console*.next();  
 }  
 **return** piece;  
 }  
  
 *//Player puts his piece according to regulations.* **public static void** PlayerPut(String piece) {  
 System.***out***.println(**"Your piece is "** + piece + **" please pick a location"**);  
 **int** row, column;  
 **while** (**true**) {  
 System.***out***.println(**"Please enter 2 integer [1,4] for coordinates(row column)ex: 1 1"**);  
 row = *console*.nextInt();  
 column = *console*.nextInt();  
 **while** (row > 4 || column > 4 || row < 1 || column < 1) {  
 System.***out***.println(**"Please enter 2 integer [1,4] for coordinates(row column)ex: 1 1"**);  
 row = *console*.nextInt();  
 column = *console*.nextInt();  
 }  
 **if** (*CoordinateEmpty*(*table*, row - 1, column - 1)) {  
 **break**;  
 }  
 }  
 **for** (**int** i = 0; i < 4; i++) {  
 *Piece*[i] = (piece.substring(i, i + 1)).toUpperCase();  
 }  
 **for** (**int** i = 0; i < 4; i++) {  
 *table*[row - 1][column - 1][i] = *Piece*[i];  
 }  
 }  
  
 *//Checks if there is a winning location, if not puts randomly* **public static void** BotPut(String piece) {  
 *AnyWinningMove*(piece);  
 **if** (*Coordinate*[0] != -1) {  
 **for** (**int** i = 0; i < 4; i++) {  
 *table*[*Coordinate*[0]][*Coordinate*[1]][i] = *Piece*[i];  
 }  
 }  
 *//Picks a empty random location and puts piece to there* **else if** (*Coordinate*[0] == -1) {  
 **int** row = *rand*.nextInt(10000) % 4;  
 **int** column = *rand*.nextInt(10000) % 4;  
 **while** (!*CoordinateEmpty*(*table*, row, column)) {  
 row = *rand*.nextInt(10000) % 4;  
 column = *rand*.nextInt(10000) % 4;  
 }  
 **for** (**int** i = 0; i < 4; i++) {  
 *table*[row][column][i] = *Piece*[i];  
 }  
 }  
 }  
  
 *//Randoms a piece if it is put before or it has a winning move it sends it to end of the array then it does not include that element to randoming space* **public static** String BotPick() {  
 String[] Pieces = {**"BTSH"**, **"BTSS"**, **"BTRH"**, **"BTRS"**, **"BSSH"**, **"BSSS"**, **"BSRH"**, **"BSRS"**, **"WTSH"**, **"WTSS"**, **"WTRH"**, **"WTRS"**, **"WSSH"**, **"WSSS"**, **"WSRH"**, **"WSRS"**};  
 **int** number = 0;  
 String memory = **""**;  
 String piece = **""**;  
 **int** c = 0;*//Flag for 16 try* **while** (c < 16) {  
 number = (*rand*.nextInt(16000) % (16 - c));  
 piece = Pieces[number];  
 **if** (*PieceExistsInTable*(piece)) {  
 memory = Pieces[15 - c]; *//Changes location with end element* Pieces[15 - c] = Pieces[number]; *//Then does not take it into randoming space* Pieces[number] = memory;  
 number = (*rand*.nextInt((16 - c) \* 1000) % (16 - c));  
 piece = Pieces[number];  
 c++;  
 **continue**;  
 }  
 *AnyWinningMove*(piece);  
 **if** (*Coordinate*[0] == -1) {  
 **return** piece;  
 } **else** {  
 memory = Pieces[15 - c];  
 Pieces[15 - c] = Pieces[number];  
 Pieces[number] = memory;  
 c++;  
 }  
 }  
 System.***out***.println(**"Unfortunately i dont have any move that i won't lose, you won congratulations"**);  
 piece = *RandomPiece*();  
 **while** (*PieceExistsInTable*(piece)) {  
 piece = *RandomPiece*();  
 }  
 **return** piece;  
 }  
  
 *//Makes Coordinate[0]=-1 if there is no winning move, if there is a winning move saves coordinates to Coordinate[]* **public static void** AnyWinningMove(String piece) {  
 **int** found = 0;*//flag for finding move* **for** (**int** i = 0; i < 4; i++) {  
 *Piece*[i] = (piece.substring(i, i + 1)).toUpperCase();  
 }  
 *//Checks if there is a winning location* **for** (**int** i = 0; i < 4; i++) {  
 **for** (**int** k = 0; k < 4; k++) {  
 **if** (*CoordinateEmpty*(*table*, i, k)) {  
 *UpdateTestTable*();  
 **for** (**int** j = 0; j < 4; j++) {  
 *TestTable*[i][k][j] = *Piece*[j];  
 }  
 **if** (*CheckWin*(*TestTable*)) {  
 found = 1;  
 *Coordinate*[0] = i;  
 *Coordinate*[1] = k;  
 **break**;  
 }  
 }  
 }  
 **if** (found == 1)  
 **break**;  
 }  
 **if** (found == 0)  
 *Coordinate*[0] = -1; *//Means doesnt exist* }  
  
 *//Returns a random piece* **public static** String RandomPiece() {  
 **int** number = *rand*.nextInt(100);  
  
 **if** (number % 2 == 0)  
 *letter*[0] = **"B"**;  
 **else** *letter*[0] = **"W"**;  
 number = *rand*.nextInt(100);  
 **if** (number % 2 == 0)  
 *letter*[1] = **"T"**;  
 **else** *letter*[1] = **"S"**;  
 number = *rand*.nextInt(100);  
 **if** (number % 2 == 0)  
 *letter*[2] = **"S"**;  
 **else** *letter*[2] = **"R"**;  
 number = *rand*.nextInt(100);  
 **if** (number % 2 == 0)  
 *letter*[3] = **"H"**;  
 **else** *letter*[3] = **"S"**;  
 **return** *letter*[0] + *letter*[1] + *letter*[2] + *letter*[3];  
  
 }  
  
 *//Returns true if coordinate is empty* **public static boolean** CoordinateEmpty(String[][][] arr, **int** row, **int** column) {  
 **return** (arr[row][column][0].equals(**"E"**));  
 }  
  
 *//Returns true if piece is not available in the game* **public static boolean** PieceNotAvailable(String piece) {  
 **for** (**int** i = 0; i < 4; i++) {  
 *Piece*[i] = (piece.substring(i, i + 1)).toUpperCase();  
 }  
 **return** (!(((*Piece*[0].equals(**"B"**) || *Piece*[0].equals(**"W"**)) && ((*Piece*[1].equals(**"T"**) || *Piece*[1].equals(**"S"**)) && ((*Piece*[2].equals(**"S"**) || *Piece*[2].equals(**"R"**)) && ((*Piece*[3].equals(**"H"**) || *Piece*[3].equals(**"S"**))))))));  
 }  
  
 *//Returns true if piece exists in the table* **public static boolean** PieceExistsInTable(String piece) {  
 **for** (**int** i = 0; i < 4; i++) {  
 *Piece*[i] = (piece.substring(i, i + 1)).toUpperCase();  
 }  
 **for** (**int** i = 0; i < 4; i++) {  
 **for** (**int** k = 0; k < 4; k++) {  
 **if** (*Piece*[0].equals(*table*[i][k][0]) && *Piece*[1].equals(*table*[i][k][1]) && *Piece*[2].equals(*table*[i][k][2]) && *Piece*[3].equals(*table*[i][k][3]))  
 **return true**;  
 }  
 }  
 **return false**;  
 }  
  
 *//Updates TestTable to current state* **public static void** UpdateTestTable() {  
 **for** (**int** i = 0; i < 4; i++) {  
 **for** (**int** k = 0; k < 4; k++) {  
 **for** (**int** j = 0; j < 4; j++) {  
 *TestTable*[i][k][j] = *table*[i][k][j];  
 }  
 }  
 }  
 }  
  
 *//Return true if winning condition satisfied for an array* **public static boolean** CheckWin(String[][][] arr) {  
 **for** (**int** i = 0; i < 4; i++) {  
 **for** (**int** k = 0; k < 4; k++) {  
 **if** ((!*CoordinateEmpty*(arr, i, 0)) && arr[i][0][k].equals(arr[i][1][k]) && arr[i][0][k].equals(arr[i][2][k]) && arr[i][0][k].equals(arr[i][3][k]))  
 **return true**;*//Row win* **if** ((!(*CoordinateEmpty*(arr, 0, i))) && arr[0][i][k].equals(arr[1][i][k]) && arr[0][i][k].equals(arr[2][i][k]) && arr[0][i][k].equals(arr[3][i][k]))  
 **return true**; *//Column win* **if** ((!(*CoordinateEmpty*(arr, 0, 0)) && arr[0][0][k].equals(arr[1][1][k]) && arr[0][0][k].equals(arr[2][2][k]) && arr[0][0][k].equals(arr[3][3][k])))  
 **return true**; *//Diagonal win* **if** ((!(*CoordinateEmpty*(arr, 0, 3)) && arr[0][3][k].equals(arr[1][2][k]) && arr[0][3][k].equals(arr[2][1][k]) && arr[0][3][k].equals(arr[3][0][k])))  
 **return true**; *//Diagonal win* }  
 }  
 **return false**;  
 }  
  
 *//Creates table from 0* **public static void** RestartTable() {  
 **for** (**int** i = 0; i < 4; i++) {  
 **for** (**int** j = 0; j < 4; j++) {  
 *table*[i][j][0] = **"E"**;  
 *table*[i][j][1] = **" "**;  
 *table*[i][j][2] = **" "**;  
 *table*[i][j][3] = **" "**;  
 }  
 }  
 }  
  
 *//Asks to load the table or not then reads it from input.txt or it creates from 0.* **public static void** ReadTable() **throws** FileNotFoundException {  
 System.***out***.println(**"Do you want to continue to old game?(Yes/No)"**);  
 String answer = *console*.next();  
 **while** (!(answer.equalsIgnoreCase(**"yes"**) || answer.equalsIgnoreCase(**"no"**))) {  
 System.***out***.println(**"Wrong input"**);  
 answer = *console*.next();  
 }  
 **if** (answer.equalsIgnoreCase(**"yes"**)) {  
 Scanner input = **new** Scanner(**new** File(**"input.txt"**));  
 *TurnNumber* = input.nextInt();  
 **for** (**int** i = 0; i < 4; i++) {  
 **for** (**int** j = 0; j < 4; j++) {  
 String memory = input.next();  
 **if** (memory.equals(**"E"**)) {  
 *table*[i][j][0] = **"E"**;  
 *table*[i][j][1] = **" "**;  
 *table*[i][j][2] = **" "**;  
 *table*[i][j][3] = **" "**;  
 } **else** {  
 **for** (**int** k = 0; k < 4; k++)  
 *table*[i][j][k] = memory.substring(k, k + 1);  
 }  
 }  
 }  
 input.close();  
 } **else** {  
 *RestartTable*();  
 }  
 }  
  
 *//Draws current table to console* **public static void** DrawTable() {  
 System.***out***.println(**" 1\t2\t3\t4"**);  
 System.***out***.println(**"------------------"**);  
 **for** (**int** i = 0; i < 4; i++) {  
 System.***out***.print((i + 1) + **"| "**);  
 **for** (**int** j = 0; j < 4; j++) {  
 **for** (**int** k = 0; k < 4; k++) {  
 System.***out***.print(*table*[i][j][k]);  
 }  
 System.***out***.print(**" "**);  
 }  
 System.***out***.println();  
 }  
 }  
  
 *//Saves game to input.txt* **public static void** SaveGame() **throws** FileNotFoundException {  
 PrintStream writer = **new** PrintStream(**"input.txt"**);*//Writer stream* writer.println(*TurnNumber*);  
 **for** (**int** i = 0; i < 4; i++) {  
 **for** (**int** j = 0; j < 4; j++) {  
 **for** (**int** k = 0; k < 4; k++) {  
 writer.print(*table*[i][j][k]);  
 }  
 writer.print(**" "**);  
 }  
 writer.println();  
 }  
 }  
}

4)Output Of The Program:



5)Conclusion: I have solved problem correctly. And added an extra where 2 bots play with each other before the assingment.