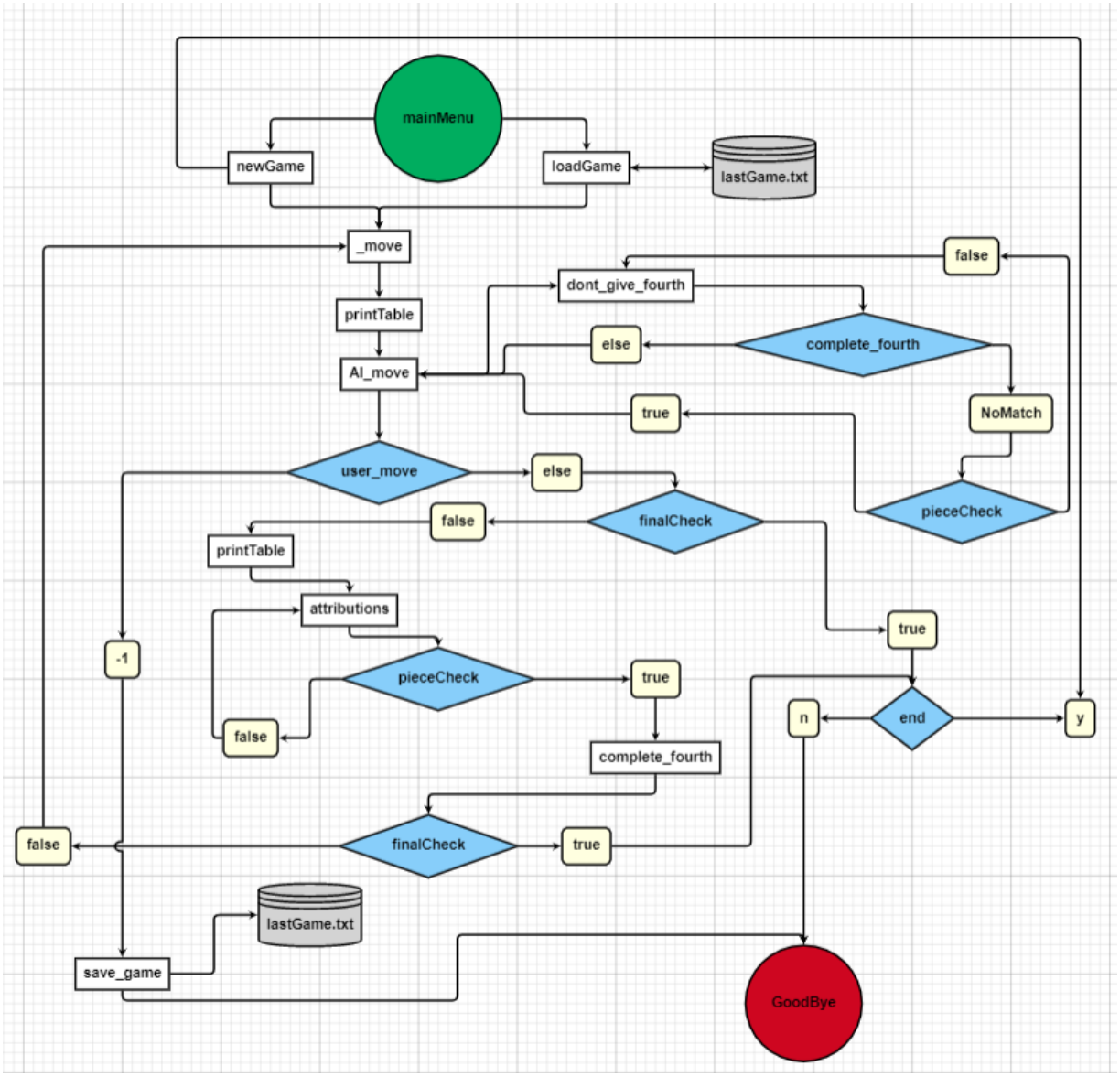
CMPE150 ASSİGNMENT 3 (EMRE GİRGİN)

1-) Problem Description:

In this assignment, we have to write a Java program which is a board game. Game consist of a 4x4 board and 16 different pieces. Each piece could be “black”(B) or “white”(W) and “tall”(T) or “short”(S) and “square”(S) or “round”(R) and “hollow”(H) or “solid”(S). 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. The game must be saved and quit if user wants to.

2-) Problem Solution:



1. Program starts with mainMenu method. If user wants to start a new game newGame method runs or if s/he wants to load last save, loadGame method runs.

2. Then _move method prints the board with printTable method and program gives the first piece to the user. While doing this thanks to dont_give_fourth method, program does not give the game-ending piece and checks the piece with complete_fourth method.Then checks piece again with pieceCheck method. If piece is already on the board program generates another piece and checks it again.
3. At user's turn user can save&quit by typing "-1" and save_game method runs. If save_game method is invoked, program saves the current board to the "lastGame.txt" and terminates. On the other hand, if user plays the piece to any coordinates. Then program check whether the piece is already on the board and the coordinates that user just determined empty or marked before. After user's move program checks whether the game is over or not. If it is over returns "end" method.
4. If game is still on, program wants user to type the attributions (properties) of the piece which is going to be given to the program to play. If piece is already on the board, program wants user to type different attributions.
5. Then, program takes the piece and checks it with complete_fourth method. complete_fourth method is going to look whether there is a coordinate it can win the game by playing to there. Since complete_fourth method is not a boolean method, it is going to continue anyway. I mean there is a winning condition or not. And makes it's move.
6. Then, program checks whether the game is over or not. If it is over returns end method. If it is not returns _move method again.
7. "end" method asks user whether s/he wants to play another game or not. If user wants so, program invokes newGame method. If s/he does not program ends.

3-)Implementation:

```
import java.util.*;

import java.io.*;


public class EG2016400099 {


	public static String mainMenu() throws FileNotFoundException {

		/*

		 * This method is kind of game's main menu.

		 * It asks user whether s/he wants to start a new game of load the last game.

		 * If user types an invalid expressions, it wants her/him to retype his answer.

		 */

		Scanner sc = new Scanner(System.in);

		System.out.println("Welcome the board game!");

		System.out.println("Do you want to start a new game or continue to the last
game?");
```

```
		System.out.println("Type \"N\" for new game or type \"L\" to load last saved game.");
		String gameType = sc.next();
		while (!(gameType.equalsIgnoreCase("n") || gameType.equalsIgnoreCase("l"))) {
			System.out.println("You typed an unvalid expression.\nPlease make sure that you have typed \"N\" or \"L\"");
			gameType = sc.next();
		}

		if (gameType.equalsIgnoreCase("n"))
			return newGame(sc);
		else//If user types "l".
			return loadGame(sc);

	}

	public static String newGame(Scanner sc) throws FileNotFoundException {
		/*
		 * This methods creates a new three dimensional char array, which is going to be our board.
		 * And fills it with 'E', which represents empty slots.
		 * Rest of the program, integer "i" is going to represent number of rows, ...
		 * ..."j" columns and "q" attributes such as white or black.
		 */
		char[][][] table = new char[4][4][4]; // Rest of the program, I have named game board as "table".
		for (int i = 0; i < 4; i++) {
			for (int j = 0; j < 4; j++) {
				for (int q = 0; q < 4; q++) {
					table[i][j][q] = 'E';
				}
			}
```

```
		}
		return _move(sc, table);
	}

	public static String loadGame(Scanner sc) throws FileNotFoundException {
		/*
		 *This method creates a File which reads save text.
		 *And fills its values to new three dimensional char array.
		 *If save text has not been created before at "else" part it creates a text and fills it with full of 'E's.
		 *More detailed features were explained in the method.
		 */
		File fileReader = new File("lastGame.txt");
		char[][][] table = new char[4][4][4];
		if (fileReader.exists()) {
			Scanner readFile = new Scanner(fileReader);//In order to keep using the same Scanner, ...
			//which is created in "mainMenu" method and PrintStream, program creates new Scanner and closes it before PrintStream.
			int eCount = 0;//Counts the number of 'E's in the file.
			for (int i = 0; i < 4; i++) {
				for (int j = 0; j < 4; j++) {
					String piece = readFile.next();//Reads features of pieces as String.
					for (int q = 0; q < 4; q++) {
						table[i][j][q] = piece.charAt(q);//And places the attributes of the piece.
						if (table[i][j][q] == 'E')
							eCount++;//Counts the number of 'E's.
					}
				}
			}
			readFile.close();//New Scanner is closed.
```

```
                        if (eCount == 64)// If text is full of 'E', that means save text has been created
before and reseted because last game is completed.
                                System.out.println("Last game was completed. New game is
started.");
                } else {//If save text has not been created before program creates its own save text
and fills it.
                        PrintStream firstCreation = new PrintStream("lastGame.txt");//Creates
"lastGame.txt".
                        System.out.println("It is your first time. There is not any game to load.\nNew
game is started.");
                        for (int i = 0; i < 4; i++) {
                                for (int j = 0; j < 4; j++) {
                                        for (int q = 0; q < 4; q++) {
                                                table[i][j][q] = 'E';
                                        }
                                }
                        }
                        firstCreation.close();//PrintStream closed.
                }
                return _move(sc, table);
        }

        public static String _move(Scanner sc, char[][][] table) throws FileNotFoundException {
                /*
                 * This method is the main step of program.
                 * It collects vary data from other methods.
                 * More details are going to explain below.
                 */
                Random rand = new Random();
                int flag = 0;//This integer determines whose turn it is.
                while (true) {//This step continues as long as it is broken, which means end of the
game.
                        System.out.println("Table's current configuration.");
```

printTable(table);//Simply the method printing the board to the console. It is explained in its block.

String features_AI_determined = AI_move(table);// AI_move returns a String which contains the attributes determined by the program.

char[] AI_offer = new char[4];//Stores the attributes of piece.

for (int i = 0; i < 4; i++)// In this for the values of string are stored in a char.

AI_offer[i] = features_AI_determined.charAt(i);

System.out.println("AI gave you " + features_AI_determined + ". Now it is your turn.");

String user_coordinates = user_move(table, sc);//user_move returns a String containing the coordinates user choosed.

flag = 1;//When flag is 1 this means it's time for user to play his piece.

int row_User = user_coordinates.charAt(0) - 48;// In ASCII table 48 represents the "0", so -48 turns value to it's integer value.

int column_User = user_coordinates.charAt(1) - 48;// I made this process because program gets coordinates as String but need to use them as integers.

if (row_User == 35 || column_User == 35)// If user types "-1", this means save and quit and -1 equals 35 in ASCII.

break;// If user wants to save and quit, program lefts the while loop.

for (int i = 0; i < 4; i++)//In this for the attributes determined by program are assigned to coordinates determined by user.

table[row_User][column_User][i] = AI_offer[i];

if (finalCheck(table, flag))//This is the check part for end of the game to determine the is over.

return end(table);//If it is over, returns the "end" method.

printTable(table);//If game is not over, prints the table's final configuration.

String features_user_determined = attribution(table, sc);// "attribution" returns a String containing the attributes -determined by user- of piece.

System.out.println(features_user_determined);

flag = 0;//It's program's turn.

char[] user_offer = new char[4];

for (int i = 0; i < 4; i++)//Stores the attributions of piece to a char array.

```
                user_offer[i] = features_user_determined.charAt(i);


            int row_AI = 0;//Declaration of variables which are going to store the
coordinates determined by program.
            int column_AI = 0;


            if (!complete_fourth(table, user_offer).equals("NoMatch")) {
                /*
                 * This block takes coordinates from "complete_fourth" method, ...
                 * which increases the possibility of winning of program.
                 * In this block the method is invoked three times but it does not
create a problem.
                 * Because every time we invoked the method it is going to produce
same result for same board.
                 * More detailed explanations is written is that method.
                 */
                row_AI = complete_fourth(table, user_offer).charAt(0) - 48;// To turn
the String value to it's real integer value.
                column_AI = complete_fourth(table, user_offer).charAt(1) - 48;
            } else {//This block creates coordinates randomly as long as that coordinates
are not empty.
                row_AI = rand.nextInt(4);
                column_AI = rand.nextInt(4);
                while (table[row_AI][column_AI][0] != 'E') {
                    row_AI = rand.nextInt(4);
                    column_AI = rand.nextInt(4);
                }
            }


            System.out.println("AI choosed R:" + row_AI + " C:" + column_AI);


            for (int i = 0; i < 4; i++)//In this for the attributes determined by user are
assigned to coordinates determined by program.
```

```
                        table[row_AI][column_AI][i] = user_offer[i];
                if (finalCheck(table, flag))
                        return end(table);
            }
            return "End of the Game!";
        }

        public static String AI_move(char[][][] table) {
            /*
             * In this method program determines attributes of the piece which will be given to
user.
             * If the piece is already in the board. Program creates a new piece and checks it
again.
             */
            char[] AI_choosed = new char[4];//Stores attributes in an array.
            Random rand = new Random();
                if (rand.nextInt(2) == 0)
                        AI_choosed[0] = 'B';
                else
                        AI_choosed[0] = 'W';
                if (rand.nextInt(2) == 0)
                        AI_choosed[1] = 'T';
                else
                        AI_choosed[1] = 'S';
                if (rand.nextInt(2) == 0)
                        AI_choosed[2] = 'S';
                else
                        AI_choosed[2] = 'R';
                if (rand.nextInt(2) == 0)
                        AI_choosed[3] = 'H';
                else// (rand.nextInt(2)==1)
                        AI_choosed[3] = 'S';
```

```
                while (!pieceCheck(table, AI_choosed)) {
                        /*
                         * "pieceCheck" is a boolean method which returns false if the piece is on the board already.
                         * If it is not it returns true.
                         */
                        if (rand.nextInt(2) == 0)
                                AI_choosed[0] = 'B';
                        else
                                AI_choosed[0] = 'W';
                        if (rand.nextInt(2) == 0)
                                AI_choosed[1] = 'T';
                        else
                                AI_choosed[1] = 'S';
                        if (rand.nextInt(2) == 0)
                                AI_choosed[2] = 'S';
                        else
                                AI_choosed[2] = 'R';
                        if (rand.nextInt(2) == 0)
                                AI_choosed[3] = 'H';
                        else// (rand.nextInt(2)==1)
                                AI_choosed[3] = 'S';
                }
                dont_give_fourth(table,AI_choosed);
                return AI_choosed[0] + "" + AI_choosed[1] + "" + AI_choosed[2] + "" + AI_choosed[3]; //Stores attributes in a String.
        }

        public static String complete_fourth(char[][][] table, char[] features) {
                /*
                 * In this method, if user gives a piece which program can win the game with it, program places it to the right coordinates.
```

```
		* Firstly, program determines is there any row which have 3 pieces and 1 empty slot.
		* If it is, it assigns the coordinates of empty slot to two variables.
		* If end of the sequence (row,column, and diagonals) there is only one empty slots, program continues.
		* Then, program stores the attributions of a piece of that sequence to an "sameFeatures" array.
		* Then, compare the attributions of pieces on that sequence.
		* If three of the pieces have the same attribution and it matches with the attribution of piece gave by user, ...
		* ... it returns the coordinates of the empty slot as a String.
		* If there is not any sequence, method returns "NoMatch".
		* More detailed explanations are made below.
		*/
		int countE = 0, countFirst = 0, countSecond = 0;
		int emptyRow = 0, emptyColumn = 0;
		for (int i = 0; i < 4; i++) {//Firstly, program checks the rows.
			countE = 0;
			emptyRow = 0;
			emptyColumn = 0;
			for (int q = 0; q < 4; q++) {
				for (int j = 0; j < 4; j++) {
					if (table[i][j][q] == 'E') {//If the first attribution of a piece is 'E', program keeps it's coordinates and count one increases.
						countE++;
						emptyRow = i;
						emptyColumn = j;
					}
				}
				if (countE == 1) {//If count is one which means there is only one empty slot in a row, that means there is something worth to examine.
					for (int f = 0; f < 4; f++) {// "f" represents the attributions.
						char[] sameFeatures = new char[4];//For each attribution the array ,storing the values of fth attribution, is reseted.
```

```
                                        for (int c = 0; c < 4; c++) {// Since there is nothing to
compare in empty slot for does not work for "emptyColumn".
                                                sameFeatures[c] =
table[emptyRow][c][f];//Attributions are stored to an array.
                                        }
                                        countFirst=0;
                                        countSecond=0;
                                        for (int k = 0; k < 4; k++) {//Now, program counts how
many times the same attribution is repeated.
                                                if (f == 0 && sameFeatures[k] == 'B') //If first
attribution of piece on the row is 'B'.
                                                        countFirst++;//First represents the
first choice of each attributions. For example Black, Tall, Square, Hollow.
                                                else if (f == 0 && sameFeatures[k] ==
'W')//f=0 which means Zeroth attribution.(B or W)
                                                        countSecond++;
                                                else if (f == 1 && sameFeatures[k] ==
'T')//First attribution.(T or S)
                                                        countFirst++;
                                                else if (f == 1 && sameFeatures[k] == 'S')
                                                        countSecond++;
                                                else if (f == 2 && sameFeatures[k] ==
'S')//Second attribution.(S or R)
                                                        countFirst++;
                                                else if (f == 2 && sameFeatures[k] == 'R')
                                                        countSecond++;
                                                else if (f == 3 && sameFeatures[k] ==
'H')//Third attribution.(H or S)
                                                        countFirst++;
                                                else if (f == 3 && sameFeatures[k] == 'S')
                                                        countSecond++;
                                        }
                                        if (f == 0 && countFirst == 3 && features[f] == 'B') //If
first attribution of piece given by user 'B' and 0th attributions of pieces on the board are the same
and 'B'.
```

```
                                                return emptyRow + "" +
emptyColumn;//Program is going to put that piece on the empty slot on that row.
                                        else if (f == 0 && countSecond == 3 && features[f] ==
'W')
                                                return emptyRow + "" + emptyColumn;
                                        else if (f == 1 && countFirst == 3 && features[f] == 'T')
                                                return emptyRow + "" + emptyColumn;
                                        else if (f == 1 && countSecond == 3 && features[f] ==
'S')
                                                return emptyRow + "" + emptyColumn;
                                        else if (f == 2 && countFirst == 3 && features[f] == 'S')
                                                return emptyRow + "" + emptyColumn;
                                        else if (f == 2 && countSecond == 3 && features[f] ==
'R')
                                                return emptyRow + "" + emptyColumn;
                                        else if (f == 3 && countFirst == 3 && features[f] ==
'H')
                                                return emptyRow + "" + emptyColumn;
                                        else if (f == 3 && countSecond == 3 && features[f] ==
'S')
                                                return emptyRow + "" + emptyColumn;
                                }
                        } else
                                break;
                }
        }

        for (int j = 0; j < 4; j++) {//This part checks for columns.
                countE = 0;
                emptyRow = 0;
                emptyColumn = 0;
                for (int q = 0; q < 4; q++) {
                        for (int i = 0; i < 4; i++) {
```

```
                if (table[i][j][q] == 'E') {
                        countE++;
                        emptyRow = i;
                        emptyColumn = j;
                }
            }
            if (countE == 1) {
                for (int f = 0; f < 4; f++) {
                        char[] sameFeatures = new char[4];
                        for (int r = 0; r < 4; r++) {// Storing fth attributions to
an array.
                                sameFeatures[r] = table[r][emptyColumn][f];
                        }
                        countFirst=0;
                        countSecond=0;
                        for (int k = 0; k < 4; k++) {
                                if (f == 0 && sameFeatures[k] == 'B')
                                        countFirst++;
                                else if (f == 0 && sameFeatures[k] == 'W')
                                        countSecond++;
                                else if (f == 1 && sameFeatures[k] == 'T')
                                        countFirst++;
                                else if (f == 1 && sameFeatures[k] == 'S')
                                        countSecond++;
                                else if (f == 2 && sameFeatures[k] == 'S')
                                        countFirst++;
                                else if (f == 2 && sameFeatures[k] == 'R')
                                        countSecond++;
                                else if (f == 3 && sameFeatures[k] == 'H')
                                        countFirst++;
                                else if (f == 3 && sameFeatures[k] == 'S')
```

```
                countSecond++;
            }
            if (f == 0 && countFirst == 3 && features[f] == 'B')
                return emptyRow + "" + emptyColumn;
            else if (f == 0 && countSecond == 3 && features[f] ==
'W')
                return emptyRow + "" + emptyColumn;
            else if (f == 1 && countFirst == 3 && features[f] == 'T')
                return emptyRow + "" + emptyColumn;
            else if (f == 1 && countSecond == 3 && features[f] ==
'S')
                return emptyRow + "" + emptyColumn;
            else if (f == 2 && countFirst == 3 && features[f] == 'S')
                return emptyRow + "" + emptyColumn;
            else if (f == 2 && countSecond == 3 && features[f] ==
'R')
                return emptyRow + "" + emptyColumn;
            else if (f == 3 && countFirst == 3 && features[f] ==
'H')
                return emptyRow + "" + emptyColumn;
            else if (f == 3 && countSecond == 3 && features[f] ==
'S')
                return emptyRow + "" + emptyColumn;
        }
    } else
        break;
    }
}

int emptySlot=0;
for (int i = 0; i < 4; i++) {//This part checks for from left top to right bottom diagonal.
    countE = 0;
    emptySlot=0;
```

```
for (int q = 0; q < 4; q++) {
        for(int j=0;j<4;j++) {
            if (table[j][j][q] == 'E') {
                countE++;
                emptySlot=j;
            }
        }
        if (countE == 1) {
            for (int f = 0; f < 4; f++) {
                char[] sameFeatures = new char[4];
                for (int r = 0; r < 4; r++) {
                    sameFeatures[r] = table[r][r][f];
                }
                countFirst=0;
                countSecond=0;
                for (int k = 0; k < 4; k++) {
                    if (f == 0 && sameFeatures[k] == 'B')
                        countFirst++;
                    else if (f == 0 && sameFeatures[k] == 'W')
                        countSecond++;
                    else if (f == 1 && sameFeatures[k] == 'T')
                        countFirst++;
                    else if (f == 1 && sameFeatures[k] == 'S')
                        countSecond++;
                    else if (f == 2 && sameFeatures[k] == 'S')
                        countFirst++;
                    else if (f == 2 && sameFeatures[k] == 'R')
                        countSecond++;
                    else if (f == 3 && sameFeatures[k] == 'H')
                        countFirst++;
                    else if (f == 3 && sameFeatures[k] == 'S')
```

```
                                countSecond++;
                            }
                            if (f == 0 && countFirst == 3 && features[f] == 'B')
                                return emptySlot + "" + emptySlot;
                            else if (f == 0 && countSecond == 3 && features[f] ==
'W')
                                return emptySlot + "" + emptySlot;
                            else if (f == 1 && countFirst == 3 && features[f] == 'T')
                                return emptySlot + "" + emptySlot;
                            else if (f == 1 && countSecond == 3 && features[f] ==
'S')
                                return emptySlot + "" + emptySlot;
                            else if (f == 2 && countFirst == 3 && features[f] == 'S')
                                return emptySlot + "" + emptySlot;
                            else if (f == 2 && countSecond == 3 && features[f] ==
'R')
                                return emptySlot + "" + emptySlot;
                            else if (f == 3 && countFirst == 3 && features[f] ==
'H')
                                return emptySlot + "" + emptySlot;
                            else if (f == 3 && countSecond == 3 && features[f] ==
'S')
                                return emptySlot + "" + emptySlot;
                        }
                    } else
                        break;
                }
            }

            for (int i = 0; i < 4; i++) {//This part checks for from right top to left bottom diagonal.
                countE = 0;
                emptySlot=0;
                for (int q = 0; q < 4; q++) {
```

```
for(int j=0;j<4;j++) {
		if (table[j][3-j][q] == 'E') {
			countE++;
			emptySlot=j;
		}
}
if (countE == 1) {
		for (int f = 0; f < 4; f++) {
			char[] sameFeatures = new char[4];
			for (int r = 0; r < 4; r++) {
				sameFeatures[r] = table[r][3-r][f];
			}
			countFirst=0;
			countSecond=0;
			for (int k = 0; k < 4; k++) {
				if (f == 0 && sameFeatures[k] == 'B')
					countFirst++;
				else if (f == 0 && sameFeatures[k] == 'W')
					countSecond++;
				else if (f == 1 && sameFeatures[k] == 'T')
					countFirst++;
				else if (f == 1 && sameFeatures[k] == 'S')
					countSecond++;
				else if (f == 2 && sameFeatures[k] == 'S')
					countFirst++;
				else if (f == 2 && sameFeatures[k] == 'R')
					countSecond++;
				else if (f == 3 && sameFeatures[k] == 'H')
					countFirst++;
				else if (f == 3 && sameFeatures[k] == 'S')
					countSecond++;
```

```
                                        }
                                        if (f == 0 && countFirst == 3 && features[f] == 'B')
                                                return emptySlot + "" + emptySlot;
                                        else if (f == 0 && countSecond == 3 && features[f] ==
'W')
                                                return emptySlot + "" + emptySlot;
                                        else if (f == 1 && countFirst == 3 && features[f] == 'T')
                                                return emptySlot + "" + emptySlot;
                                        else if (f == 1 && countSecond == 3 && features[f] ==
'S')
                                                return emptySlot + "" + emptySlot;
                                        else if (f == 2 && countFirst == 3 && features[f] == 'S')
                                                return emptySlot + "" + emptySlot;
                                        else if (f == 2 && countSecond == 3 && features[f] ==
'R')
                                                return emptySlot + "" + emptySlot;
                                        else if (f == 3 && countFirst == 3 && features[f] ==
'H')
                                                return emptySlot + "" + emptySlot;
                                        else if (f == 3 && countSecond == 3 && features[f] ==
'S')
                                                return emptySlot + "" + emptySlot;
                                }
                        } else
                                break;
                }
        }
        return "NoMatch";//If there is not any sequence method returns "NoMatch".
    }

    public static void dont_give_fourth(char[][][] table, char[] AI_choosed) {
        /*
         * This method prevents program to give to user a piece which can be end the game.
```

```
		* Program creates a new piece until that piece is not on the board, ...
		* ... and user can not win the game with this piece.
		* However, there is an exception condition:
		* If all the pieces ,which are not played yet, allow user to win the game, while loop has to be broken.
		* To do this there is a count variable. Every time the piece is checked it increases by one.
		* Finally, if count reaches the value determined by me, it breaks the while and gives a random piece to user.
		* Therefore, this method does not guarantee that program is going to win.
		* It just increases the possibility of the winning of the program.
		*/
		int count=0;
		while(!complete_fourth(table,AI_choosed).equals("NoMatch")&&count<999999) {
			/*
			 * I determined the upper limit as 999999, which is big enough to try all possibilities.
			 * By doing this program is going to try all pieces for current board.
			 */
			Random rand = new Random();
			if (rand.nextInt(2) == 0)
				AI_choosed[0] = 'B';
			else
				AI_choosed[0] = 'W';
			if (rand.nextInt(2) == 0)
				AI_choosed[1] = 'T';
			else
				AI_choosed[1] = 'S';
			if (rand.nextInt(2) == 0)
				AI_choosed[2] = 'S';
			else
				AI_choosed[2] = 'R';
```

```
                        if (rand.nextInt(2) == 0)
                                AI_choosed[3] = 'H';
                        else// (rand.nextInt(2)==1)
                                AI_choosed[3] = 'S';
                        while (!pieceCheck(table, AI_choosed)) {
                                /*
                                 * "pieceCheck" is a boolean method which returns false if the piece
is on the board already.
                                 * If it is not it returns true.
                                 */
                                if (rand.nextInt(2) == 0)
                                        AI_choosed[0] = 'B';
                                else
                                        AI_choosed[0] = 'W';
                                if (rand.nextInt(2) == 0)
                                        AI_choosed[1] = 'T';
                                else
                                        AI_choosed[1] = 'S';
                                if (rand.nextInt(2) == 0)
                                        AI_choosed[2] = 'S';
                                else
                                        AI_choosed[2] = 'R';
                                if (rand.nextInt(2) == 0)
                                        AI_choosed[3] = 'H';
                                else// (rand.nextInt(2)==1)
                                        AI_choosed[3] = 'S';
                        }
                        count++;
                }
        }
```

```
public static String user_move(char[][][] table, Scanner sc) throws FileNotFoundException {
		/*
		 * In this method user determines the coordinates of his move.
		 * Since it is user's turn, at this method user can save and quit.
		 */
		int row_User, column_User;
		System.out.println("Since it's your turn, you can \"Save&Quit\" by typing \"-1\".");
		System.out.println("Please enter a row.");
		while(!sc.hasNextInt()) {//Prevents user from typing different than integer.
			System.out.println("Please type an integer for row.");
			sc.next();//Consumes non-integer input.
		}
		row_User = sc.nextInt();
		if (row_User == -1)//If user types "-1" returns save method.
			return save_game(table);
		System.out.println("Please enter a column.");
		while(!sc.hasNextInt()) {
			System.out.println("Please type an integer for column.");
			sc.next();
		}
		column_User = sc.nextInt();
		if (column_User == -1)
			return save_game(table);
		while (-1 > row_User || row_User > 3 || -1 > column_User || column_User > 3
				|| table[row_User][column_User][0] != 'E') {
			/*
			 * If user types an integer but not a valid value, program wants her/him to rewrite its coordinates.
			 */
			if (-1 > row_User || row_User > 3 || -1 > column_User || column_User > 3)//Makes correct warning to user.
				System.out.println("Please enter a valid coordination.(4X4)");
```

```
                else if (table[row_User][column_User][0] != 'E')
                        System.out.println("This point has been already marked.");
                System.out.println("Please enter a row.");
                while(!sc.hasNextInt()) {//Type control still continues.
                        System.out.println("Please type an integer for row.");
                        sc.next();
                }
                row_User = sc.nextInt();
                if (row_User == -1)//User can still save and quit.
                        return save_game(table);
                System.out.println("Please enter a column.");
                while(!sc.hasNextInt()) {
                        System.out.println("Please type an integer for column.");
                        sc.next();
                }
                column_User = sc.nextInt();
                if (column_User == -1)
                        return save_game(table);

        }

        return row_User + "" + column_User;//Stores value of coordinates in terms of String.
    }

    public static String attribution(char[][][] table, Scanner sc) throws FileNotFoundException{
        /*
         * In this method user determines the attributions of piece which will be given to the
program.
         * Program warns user if s/he types an invalid value with while loops.
         * If user determines attributions which is the same with a piece which is already on
the board, ...
         * ... program wants her/him to type another attributions.
```

```
                */
                char[] user_choosed = new char[4];
                System.out.println("Enter the attributions of opponent's move.");
                System.out.println("Black or white?(B\\W)");
                String s1 = sc.next();
                while (!s1.equalsIgnoreCase("b") && !s1.equalsIgnoreCase("w")) {
                        System.out.println("You have typed an invalid expression.Please type \"B\"
or \"W\".");
                        s1 = sc.next();
                }
                if (s1.equalsIgnoreCase("b"))
                        user_choosed[0] = 'B';
                if (s1.equalsIgnoreCase("w"))
                        user_choosed[0] = 'W';
                System.out.println("Tall or short?(T\\S)");
                String s2 = sc.next();
                while (!s2.equalsIgnoreCase("t") && !s2.equalsIgnoreCase("s")) {
                        System.out.println("You have typed an invalid expression.Please type \"T\" or
\"S\".");
                        s2 = sc.next();
                }
                if (s2.equalsIgnoreCase("t"))
                        user_choosed[1] = 'T';
                if (s2.equalsIgnoreCase("s"))
                        user_choosed[1] = 'S';
                System.out.println("Square or round?(S\\R)");
                String s3 = sc.next();
                while (!s3.equalsIgnoreCase("s") && !s3.equalsIgnoreCase("r")) {
                        System.out.println("You have typed an invalid expression.Please type \"S\" or
\"R\".");
                        s3 = sc.next();
                }
```

```
		if (s3.equalsIgnoreCase("s"))
			user_choosed[2] = 'S';
		if (s3.equalsIgnoreCase("r"))
			user_choosed[2] = 'R';

		System.out.println("Hollow or Solid?(H\\S)");
		String s4 = sc.next();
		while (!s4.equalsIgnoreCase("h") && !s4.equalsIgnoreCase("s")) {
			System.out.println("You have typed an invalid expression.Please type \"H\" or \"S\".");
			s4 = sc.next();
		}
		if (s4.equalsIgnoreCase("h"))
			user_choosed[3] = 'H';
		if (s4.equalsIgnoreCase("s"))
			user_choosed[3] = 'S';
		while (!pieceCheck(table, user_choosed)) {// Checks whether the piece is on the board or not.
			System.out.println("That piece is already on the board. Please pick another piece.");
			System.out.println("Black or white?(B\\W)");
			s1 = sc.next();
			while (!s1.equalsIgnoreCase("b") && !s1.equalsIgnoreCase("w")) {
				System.out.println("You have typed an invalid expression.Please type \"B\" or \"W\".");
				s1 = sc.next();
			}
			if (s1.equalsIgnoreCase("b"))
				user_choosed[0] = 'B';
			if (s1.equalsIgnoreCase("w"))
				user_choosed[0] = 'W';
			System.out.println("Tall or short?(T\\S)");
```

```
s2 = sc.next();
while (!s2.equalsIgnoreCase("t") && !s2.equalsIgnoreCase("s")) {
	System.out.println("You have typed an invalid expression.Please type \"T\" or \"S\".");
	s2 = sc.next();
}
if (s2.equalsIgnoreCase("t"))
	user_choosed[1] = 'T';
if (s2.equalsIgnoreCase("s"))
	user_choosed[1] = 'S';
System.out.println("Square or round?(S\\R)");
s3 = sc.next();
while (!s3.equalsIgnoreCase("s") && !s3.equalsIgnoreCase("r")) {
	System.out.println("You have typed an invalid expression.Please type \"S\" or \"R\".");
	s3 = sc.next();
}
if (s3.equalsIgnoreCase("s"))
	user_choosed[2] = 'S';
if (s3.equalsIgnoreCase("r"))
	user_choosed[2] = 'R';

System.out.println("Hollow or Solid?(H\\S)");
s4 = sc.next();
while (!s4.equalsIgnoreCase("h") && !s4.equalsIgnoreCase("s")) {
	System.out.println("You have typed an invalid expression.Please type \"H\" or \"S\".");
	s4 = sc.next();
}
if (s4.equalsIgnoreCase("h"))
	user_choosed[3] = 'H';
if (s4.equalsIgnoreCase("s"))
```

```
                        user_choosed[3] = 'S';
                }
                return user_choosed[0] + "" + user_choosed[1] + "" + user_choosed[2] + "" +
user_choosed[3];//Stores it as a String.
        }

        public static boolean pieceCheck(char[][][] table, char[] choosed) {
                /*
                 * In this method program checks whether the piece is already on the board or not.
                 */
                int count = 0;//This variable stores the similarities between new piece and pieces
which are already on the board.
                for (int i = 0; i < 4; i++) {
                        for (int j = 0; j < 4; j++) {
                                count = 0;//For each piece count is reseted.
                                for (int q = 0; q < 4; q++) {
                                        if (table[i][j][q] == choosed[q])
                                                count++;//Counts the similar attributes
                                }
                                if (count == 4)// If all atributes of the new piece matches with a piece
which is already on the board, count becomes 4.
                                        return false;// If count is 4 that means all four attributes are
match, which means the pieces are same.

                        }
                }
                return true;// Count has never been four, so new piece is unique.
        }

        public static void printTable(char[][][] table) {
                /*
                 * Prints the board's current configuration.
```

```
		 * If a coordinate is empty, prints "____".
		 */
		for (int i = 0; i < 4; i++) {
			for (int j = 0; j < 4; j++) {
				for (int q = 0; q < 4; q++) {
					if (table[i][j][q] == 'E') {
						System.out.print("____");
						break;//If first attribute of a piece is 'E', it is meaningless to continue to for loop.
					}
					System.out.print(table[i][j][q]);
				}
				System.out.print(" ");
			}
			System.out.println();
		}
	}

	public static String save_game(char[][][] table) throws FileNotFoundException {
		/*
		 * In this method the current configuration of board is written to a text.
		 * In every save program resets text file.
		 */
		PrintStream stream = new PrintStream("lastGame.txt");
		for (int i = 0; i < 4; i++) {
			for (int j = 0; j < 4; j++) {
				for (int q = 0; q < 4; q++) {
					stream.print(table[i][j][q]);
				}
				stream.print(" ");//Creates spaces between pieces so that Scanner can read them piece by piece.
			}
```

```
                stream.println();//Moves to new row.
            }
            stream.close();
            System.out.println("Game has been saved successfully.\nSee you later! :)");
            return "Save end.";//It is not important the value that method returns but it has to return a String
            //...because it is used in other methods which returns String.
        }

        public static boolean finalCheck(char[][][] table, int flag) {
            /*
             * This method check whether the game is over or not.
             * Returns "true" if game is over or returns "false" if the game is going on.
             * That means if 4 of the sequential pieces have at least one the same attribution, game is over.
             * Firstly, program checks is there any two pieces having the same attribution and they are not empty slots in a  row.
             * If it is, variable "count" increase by one.
             * End of the row if count is 3, that means there are four pieces having at least one the same attribution.
             * In addition, in order to determine who is the winner, program checks "flag" variable which is created in "move" method and passed as a parameter to this method.
             * And this steps are the same for column check and diagonals with a little bit different.
             * In the bottom of the method program checks whether there is a tie or not.
             */
            int count = 0;

            for (int i = 0; i < 4; i++) {//Checks for row end.
                for (int q = 0; q < 4; q++) {
                    count = 0;//For each attribution count is reseted.
                    for (int j = 0; j < 3; j++) {// Variable "j" represents columns a usual and it is up to 3, because in order to prevent arrayindexoutofboundexception.
```

```
                                        if (table[i][j][q] == table[i][j + 1][q] && table[i][j][q] != 'E')
{//ie. The first attribution of 0:0 and 0:1 is the same and they are not 'E'.
                                                count++;
                                        }
                                }
                                if (count == 3) {
                                        printTable(table);
                                        System.out.print("There is a row end.");
                                        if (flag == 1)//1 represents user.
                                                System.out.println("You won!");
                                        if (flag == 0)//0 represents program.
                                                System.out.println("AI won!");
                                        return true;
                                }
                        }
                }
                for (int j = 0; j < 4; j++) {//Checks for column end.
                        for (int q = 0; q < 4; q++) {
                                count = 0;
                                for (int i = 0; i < 3; i++) {
                                        if (table[i][j][q] == table[i + 1][j][q] && table[i][j][q] != 'E') {
                                                count++;
                                        }
                                }
                                if (count == 3) {
                                        printTable(table);
                                        System.out.print("There is a column end.");
                                        if (flag == 1)
                                                System.out.println("You won!");
                                        if (flag == 0)
                                                System.out.println("AI won!");
```

```
                    return true;
                }
            }
        }

        for (int q = 0; q < 4; q++) {// Checks the diagonal which is from left top to the right bottom.
            count = 0;
            for (int i = 0; i < 3; i++) {//Since in this diagonal row and column depend on each other there are only two for loops.
                if (table[i][i][q] == table[i + 1][i + 1][q] && table[i][i][q] != 'E') {
                    count++;
                }
            }
            if (count == 3) {
                printTable(table);
                System.out.print("There is a diagonal(from left to right) end.");
                if (flag == 1)
                    System.out.println("You won!");
                if (flag == 0)
                    System.out.println("AI won!");
                return true;
            }
        }

        for (int q = 0; q < 4; q++) {// Checks the diagonal which is form right top to the left bottom.
            count = 0;
            for (int i = 0; i < 3; i++) {
                if (table[i][3 - i][q] == table[i + 1][2 - i][q] && table[i][3 - i][q] != 'E') {
                    count++;
                }
```

```
                }
                if (count == 3) {
                        printTable(table);
                        System.out.print("There is a diagonal(from right to left) end.");
                        if (flag == 1)
                                System.out.println("You won!");
                        if (flag == 0)
                                System.out.println("AI won!");
                        return true;
                }
        }
        int tieCount = 0;
        for (int i = 0; i < 4; i++) {// If there is not any end and none of the coordinates is not empty, that means there is tie.
                for (int j = 0; j < 4; j++) {
                        if (table[i][j][0] != 'E')
                                tieCount++;
                }
        }
        if (tieCount == 16) {
                System.out.println("There is a tie. No winner!");
                return true;
        }

        return false;
    }

    public static String end(char[][][] table) throws FileNotFoundException {
        /*
         * This method is run if the game is over.
         * Firstly, this method resets the board of save file.
```

```
		 * Then asks user whether s/he wants to play another game or not.
		 * If user wants, returns "newGame" method.
		 * If user does not want, program ends.
		 */
		PrintStream stream = new PrintStream("lastGame.txt");//Resets the save file.
		for (int i = 0; i < 4; i++) {//And fills it with full of 'E'.
			for (int j = 0; j < 4; j++) {
				for (int q = 0; q < 4; q++) {
					stream.print('E');
				}
				stream.print(" ");
			}
			stream.println();
		}
		stream.close();
		Scanner sc = new Scanner(System.in);
		System.out.println("Game is over. Do you want to play another game?(Y\\N)");
		String answer = sc.next();
		while (!answer.equalsIgnoreCase("y") && !answer.equalsIgnoreCase("n")) {//In case there is any mistyped input.
			System.out.println("You have typed invalid expression.\n Please type \"Y\" or \"N\".");
			answer=sc.next();
		}
		if (answer.equalsIgnoreCase("y"))
			return newGame(sc);
		else
			System.out.println("Thanks for playing!");
		sc.close();
		return "End with one of the players won!";//Program ends.
	}
```

```
    public static void main(String[] args) throws FileNotFoundException {
        mainMenu();
    }
```

}**4-)Outputs:**

Scenario:

Since it is first game there is not save file. User wants to load game then program warns her/him and opens a new board, not crashes. At the end of the game any pieces that user is going to choose will lead to program to win. So AI wins! After game is over, program asks user whether s/he wants to play another game or not.

```
Welcome the board game!
Do you want to start a new game or continue to the last game?
Type "N" for new game or type "L" to load last saved game.
1
It is your first time. There is not any game to load.
New game is started.
Table's current configuration.
____ ____ ____ ____
____ ____ ____ ____
____ ____ ____ ____
____ ____ ____ ____
AI gave you WTRH. Now it is your turn.
Since it's your turn, you can "Save&Quit" by typing "-1".
Please enter a row.
2
Please enter a column.
2
____ ____ ____ ____
____ ____ ____ ____
____ ____ WTRH ____
____ ____ ____ ____
Enter the attributions of opponent's move.
Since it's your turn you can "Save&Quit" by typing "-1".
However, notice that you can only save right now, not at later attributions.
Black or white?(B\W)
b
Tall or short?(T\S)
t
Square or round?(S\R)
s
Hollow or Solid?(H\S)
h
BTSH
AI choosed R:1 C:3
Table's current configuration.
____ ____ ____ ____
____ ____ ____ BTSH
____ ____ WTRH ____
____ ____ ____ ____
```

```
AI gave you WTSH. Now it is your turn.
Since it's your turn, you can "Save&Quit" by typing "-1".
Please enter a row.
0
Please enter a column.
2
____ ____ WTSH ____
____ ____ ____ BTSH
____ ____ WTRH ____
____ ____ ____ ____
Enter the attributions of opponent's move.
Since it's your turn you can "Save&Quit" by typing "-1".
However, notice that you can only save right now, not at later attributions.
Black or white?(B\W)
w
Tall or short?(T\S)
t
Square or round?(S\R)
s
Hollow or Solid?(H\S)
s
WTSS
AI choosed R:0 C:0
Table's current configuration.
WTSS ____ WTSH ____
____ ____ ____ BTSH
____ ____ WTRH ____
____ ____ ____ ____
AI gave you WSRH. Now it is your turn.
Since it's your turn, you can "Save&Quit" by typing "-1".
Please enter a row.
3
Please enter a column.
2
WTSS ____ WTSH ____
____ ____ ____ BTSH
____ ____ WTRH ____
____ ____ WSRH ____
Enter the attributions of opponent's move.
Since it's your turn you can "Save&Quit" by typing "-1".
However, notice that you can only save right now, not at later attributions.
Black or white?(B\W)
b
Tall or short?(T\S)
s
Square or round?(S\R)
r
Hollow or Solid?(H\S)
s
BSRS
AI choosed R:3 C:1
Table's current configuration.
WTSS ____ WTSH ____
____ ____ ____ BTSH
____ ____ WTRH ____
____ BSRS WSRH ____
AI gave you BTSS. Now it is your turn.
Since it's your turn, you can "Save&Quit" by typing "-1".
Please enter a row.
1
Please enter a column.
0
WTSS ____ WTSH ____
BTSS ____ ____ BTSH
____ ____ WTRH ____
____ BSRS WSRH ____
```

```
Enter the attributions of opponent's move.
Since it's your turn you can "Save&Quit" by typing "-1".
However, notice that you can only save right now, not at later attributions.
Black or white?(B\W)
b
Tall or short?(T\S)
s
Square or round?(S\R)
s
Hollow or Solid?(H\S)
s
BSSS
AI choosed R:2 C:1
Table's current configuration.
WTSS ____ WTSH ____
BTSS ____ ____ BTSH
____ BSSS WTRH ____
____ BSRS WSRH ____
AI gave you BTRS. Now it is your turn.
Since it's your turn, you can "Save&Quit" by typing "-1".
Please enter a row.
3
Please enter a column.
0
WTSS ____ WTSH ____
BTSS ____ ____ BTSH
____ BSSS WTRH ____
BTRS BSRS WSRH ____
```

```
Enter the attributions of opponent's move.
Since it's your turn you can "Save&Quit" by typing "-1".
However, notice that you can only save right now, not at later attributions.
Black or white?(B\W)
b
Tall or short?(T\S)
s
Square or round?(S\R)
s
Hollow or Solid?(H\S)
s
That piece is already on the board. Please pick another piece.
Black or white?(B\W)
b
Tall or short?(T\S)
t
Square or round?(S\R)
s
Hollow or Solid?(H\S)
h
That piece is already on the board. Please pick another piece.
Black or white?(B\W)
b
Tall or short?(T\S)
t
Square or round?(S\R)
r
Hollow or Solid?(H\S)
h
BTRH
AI choosed R:3 C:3

WTSS ____ WTSH ____
BTSS ____ ____ BTSH
____ BSSS WTRH ____
BTRS BSRS WSRH BTRH
There is a row end.AI won!
Game is over. Do you want to play another game?(Y\N)
n
Thanks for playing!
```

Scenario:

User loads a game from lastGame.txt and AI wins with diagonal end.

```
Welcome the board game!
Do you want to start a new game or continue to the last game?
Type "N" for new game or type "L" to load last saved game.
l
Table's current configuration.
WSSS ____ ____ BTRH
____ BTSH ____ ____
____ WTRH ____ WSRS
____ ____ ____ BSSS
AI gave you BSRH. Now it is your turn.
Since it's your turn, you can "Save&Quit" by typing "-1".
Please enter a row.
1
Please enter a column.
3
WSSS ____ ____ BTRH
____ BTSH ____ BSRH
____ WTRH ____ WSRS
____ ____ ____ BSSS
Enter the attributions of opponent's move.
Black or white?(B\W)
w
Tall or short?(T\S)
s
Square or round?(S\R)
r
Hollow or Solid?(H\S)
h
WSRH
AI choosed R:1 C:2
Table's current configuration.
WSSS ____ ____ BTRH
____ BTSH WSRH BSRH
____ WTRH ____ WSRS
____ ____ ____ BSSS
```

```
AI gave you WTRS. Now it is your turn.
Since it's your turn, you can "Save&Quit" by typing "-1".
Please enter a row.
2
Please enter a column.
0
WSSS ____ ____ BTRH
____ BTSH WSRH BSRH
WTRS WTRH ____ WSRS
____ ____ ____ BSSS
Enter the attributions of opponent's move.
Black or white?(B\W)
b
Tall or short?(T\S)
t
Square or round?(S\R)
s
Hollow or Solid?(H\S)
s
BTSS
AI choosed R:2 C:2
WSSS ____ ____ BTRH
____ BTSH WSRH BSRH
WTRS WTRH BTSS WSRS
____ ____ ____ BSSS
There is a diagonal(from left to right) end.AI won!
Game is over. Do you want to play another game?(Y\N)
y
Table's current configuration.
____ ____ ____ ____
____ ____ ____ ____
____ ____ ____ ____
____ ____ ____ ____
AI gave you BSSS. Now it is your turn.
Since it's your turn, you can "Save&Quit" by typing "-1".
Please enter a row.
```

Scenario:

User starts a new game and saves and quits two steps later.

```
Welcome the board game!
Do you want to start a new game or continue to the last game?
Type "N" for new game or type "L" to load last saved game.
n
Table's current configuration.
____ ____ ____ ____
____ ____ ____ ____
____ ____ ____ ____
____ ____ ____ ____
AI gave you WSRS. Now it is your turn.
Since it's your turn, you can "Save&Quit" by typing "-1".
Please enter a row.
2
Please enter a column.
2
____ ____ ____ ____
____ ____ ____ ____
____ ____ WSRS ____
____ ____ ____ ____
Enter the attributions of opponent's move.
Black or white?(B\W)
b
Tall or short?(T\S)
t
Square or round?(S\R)
r
Hollow or Solid?(H\S)
h
BTRH
AI choosed R:2 C:3
Table's current configuration.
____ ____ ____ ____
____ ____ ____ ____
____ ____ WSRS BTRH
____ ____ ____ ____
AI gave you BTSS. Now it is your turn.
Since it's your turn, you can "Save&Quit" by typing "-1".
Please enter a row.
-1
Game has been saved successfully.
See you later! :)
```

5-) Conclusion:

My program works exactly how it has to do. In addition,

- If it is the first time the game is played, and user tries to open save file which is not exist yet, my program does not crash. My program warns user and creates a save file.
- End of the game program asks user whether s/he wants to play another game or not.
- My program has a kind of primitive Artificial Intelligence:
    1. It wins the game if the game-ending piece is given by user.
    2. It does not give the game-ending piece to the user.