Abderahman Haouit

1.Overview of my game:

Tic-tac-toe game or X and o's as some people will call it. The player who succeeds in placing three of their marks in a horizontal, vertical, or a diagonal row wins the game or in some cases the game can end up as a draw. Turns are taken marking the spaces in a 3x3 grid. This is a two-player game and in this game, there is going to be one player versus a second player only. An appropriate message should be printed out stating which player has won or else a draw. Putting a move in a space that's already been taking before should result in a message stating that position x has been taken already and at the same time prompting the user to enter an alternative position. If a number that is entered isn't between 1-9 inclusive an error message should be printed out as well and prompting the user to enter a correct number in the appropriate range. A gameboard will be updated to help the 2 players visual the game each time a move has been made. As soon as there is a win the game automatically stops taking any sort of input and prints out the final result gameboard followed by a message stating the outcome of the game.

2.How to play:

The two players playing have to decide who's going to start first, this can be done with a quick game of rock paper scissors.

To play this game you will be prompted to enter a number 1-9 which will then fill an empty space/box corresponding to the position of that empty space/box (horizontally in numerical order) and then player2 is going to counter your move, a visualised gameboard will be printed each time after a number is inputted from the user (which is the position the user wants his/her mark to be placed at) and after the second players move as well of course. This is done to help both of the players keep track of their movements in order to aid both user's strategy their future moves in order to win the game.

3.The Code structure:

Starting from top to bottom, my code consists of 4 methods including the main method where the gameBoard is created using a 2-D char array which is going to hold the letters X&O. In the main I've called each of the other methods at least once as my main method is going to be the brain of this whole program where everything is going to be controlled. I create the game loop where the cpu gets input from user1 using scanner on a position from 1-9, we store it and check if we have a winner depending on the circumstances and rules that we implemented.

The second method (printGameBoard) is simply created so that whenever the method is called in the main an image of the gameBoard is printed to the screen again to help the users keep track of their progress. The method also takes in a parameter list which is the list of the variables the method takes in and, in this case, the 2-D array. In this method we have a foreach loop with 2-D array which goes through the gameboard by row and column and prints the contents.

The third method (placePiece) takes in 3 different parameters the gameBoard contents, the input from user which is the position and the user himself as we are dealing with 2 players. We assign player1 to 'X' and player2 to 'O' then we use the switch statement where we create 9 cases because we are only interested in the 9 available boxes that are empty and when the number (input from user) is inputted it'll match with the case statement and will execute that statement which is basically filling the symbol from the players into the empty spaces. The placePiece method is only called once for each player (so twice altogether in the main) and each will continue to execute until the while loop condition isn't met.

Last but not least we have the (checkWinner) method which returns a boolean and contains no parameters. Here we check if we have a winner and return a boolean followed by a message to be printed out to the screen. Theirs 3 different ways the game can end, either with player 1 or player 2 winning or a draw. Also taking into consideration the different possible ways you can win as well, horizontally, vertically and diagonally with a consist of the same letter .9 lists were created holding

the positions which marked the different ways a player can win for example (1,2,3) so at position 1,2 and 3 if it contains a consist of the same letter, the player wins. This is how all the win conditions are checked corresponding to the stored positions that were inputted. We then create arrayList and connect it the 9 conditions using the 2 arrayList we made at the start in the class scope.

I also imported 4 different imports the scanner to read user input, arrayList to add elements to the arrayList, the Arrays import was used to sort fill and add even search elements in the array and the List import to hold the information of the positions and different condition wins.

4.Data structures and algorithms used:

Using ArrayList allows you to have the benefits of an array while offering flexibility in size, running out of space isn't an issue because the capacity grows as you insert elements this makes it is easy to access elements and modify them quickly. I also used LinkedList to help insert elements very quickly and efficiently which is done in constant time O(1). My game doesn't offer the user a choice to delete a move but would ask again to insert a move. As the game is played arrayList helped change the size array at runtime because as we play the game that's what we want or else the game wouldn't work as it involves user interaction. The complexity ArrayList bring adding and inserting elements are both O(1) as well like we seen with the LinkedList. This makes the game very fast as the game is played. The list data structure has two very distinctive implementations arrayList & linkedList that's why in my code all of these 3 different data structures are linked in a way to work together to reach efficiency which provides the best performance our code can run on.

5.Learning outcomes:

Designing and analysing simple algorithms is very important because it can give you an understanding of what to expect solving problems and also implementing the solution in the most efficient way as possible, iteration, calculating the running time of iterative algorithms in my game the (foreach loop). Using the big 'O' notation to express algorithmic running time is extremely important because you don't want your game to be slow for the users you want it to be as fast as possible which gives the user an easy & untroubled experience. My strong understanding of algorithm analysis and data structures I feel has drastically improved my ability to solve problems with powerful and efficient programs which in this game Links ,arrayList even simple while loops etc has helped me make this 2 player game work in the most simple and efficient way taking into account the different scenarios the game can interpret and being able to adapt my code to these different scenarios and making the game work has definitely improved the way I think and come across questions or problems computationally.

6.Appendix with all the code, commented appropriately:

```
/* Tic-tac-Toe game
 * 2 Players only!.
 */

import java.util.Scanner;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
public class Game {
     //two import statements for player1 and player2.
      static ArrayList<Integer>player1Positions=new
ArrayList<Integer>();
```

```
     static ArrayList<Integer>player2Positions=new
ArrayList<Integer>();

     public static void main(String[]args) {
          //Creating the game board and printing the game
board
          char [][] gameBoard= {{' ','|', ' ', '|',' '},
                    {'-','+', '-', '+','-'},
                    {' ','|', ' ', '|',' '},
                    {'-','+', '-', '+','-'},
                    {' ','|', ' ', '|',' '}};
          printGameBoard(gameBoard);
          /*Create the game loop , get input from user1 using
scanner on a position from 1-9,
           * we store it and then check if their's a winner.
Next user2 moves we store it and check for a winner,
           * with help from printGameBoard method,placePiece
method and the checkWinner method.
           */

          while(checkWinner()==false) {
               Scanner sc = new Scanner(System.in);
               System.out.println("Player1(X): enter your
placement number from 1-9!");
               int playerPos1 = sc.nextInt();
               /*Here we make sure that user1 input is between
1-9 otherwise keep
                 prompting user to enter the correct input
range.
               */
               while(playerPos1>=10||playerPos1<=0) {
                    System.out.println("Player1 please enter a
number between 1-9!!");
                    playerPos1 = sc.nextInt();
               }
               //Check if the inputed position entered already
exists or not for player 1.

     while(player2Positions.contains(playerPos1)||player1Posit
ions.contains(playerPos1)){
                    System.out.println("Position
"+playerPos1+" is Taken!");
                    playerPos1=sc.nextInt();
                    }
               placePiece(gameBoard,playerPos1,"player1");
               printGameBoard(gameBoard);
               //if we find winner stop the game and print
result ,else continue.
               if(checkWinner()==false)
```

```
                    System.out.println("Player2(O): enter your
placement number from 1-9!");
                    int playerPos2 = sc.nextInt();
                    /*Here we make sure that user2 input is
between 1-9 otherwise keep
                      prompting user to enter the correct
input range.
                    */
                    while(playerPos2>9||playerPos2<1) {
                         System.out.println("Player2 please
enter a number between 1-9!!");
                         playerPos2 = sc.nextInt();
                         }
                    //Check if the inputed position entered
already exists or not for player 2.

     while(player1Positions.contains(playerPos2)||player2Posit
ions.contains(playerPos2)){
                         System.out.println("Position
"+playerPos2+" is Taken!");
                         playerPos2=sc.nextInt();
                    }

     placePiece(gameBoard,playerPos2,"player2");
                    printGameBoard(gameBoard);
                //sc.close();
          }

     }
     public static void printGameBoard(char[][]gameBoard){
          /*Print the board each time a move is made and keeps
updating the new board
           *until the final move or result is reached.
          */
          for(char[] row:gameBoard) {
               for(char col:row) {
                    System.out.print(col);
               }
               System.out.println();
          }
     }
     public static void placePiece(char[][]gameBoard,int
pos,String user) {
          /* when user1 or user2 input (position) equals to
the case number execute the case
           * command.
           */
          char symbol = ' ';
          if(user.equals("player1")) {
```

```
                symbol='X';
        //assign player 1 to the symbol X.
                player1Positions.add(pos);
            }else if(user.equals("player2")) {
                symbol='O';
        //assign player 2 to the symbol O.
                player2Positions.add(pos);
            }
            //These are the 9 empty spaces that are going to be
matched and filled.
            switch(pos) {
            case 1:
                gameBoard[0][0]=symbol;
                break;
            case 2:
                gameBoard[0][2]=symbol;
                break;
            case 3:
                gameBoard[0][4]=symbol;
                break;
            case 4:
                gameBoard[2][0]=symbol;
                break;
            case 5:
                gameBoard[2][2]=symbol;
                break;
            case 6:
                gameBoard[2][4]=symbol;
                break;
            case 7:
                gameBoard[4][0]=symbol;
                break;
            case 8:
                gameBoard[4][2]=symbol;
                break;
            case 9:
                gameBoard[4][4]=symbol;
                break;
            default:
                break;
            }
        }
        public static boolean checkWinner() {
            /*As we check for a winner we stored all the win
conditions and we check if,
             * any of the stored positions has it and we keep
checking and placing till
             * we get a winner and we print.
             */
```

```
            //The 9 different win conditions with their position
            List topRow=Arrays.asList(1, 2, 3);
            List midRow=Arrays.asList(4, 5, 6);
            List botRow=Arrays.asList(7, 8, 9);
            List leftCol=Arrays.asList(1, 4, 7);
            List midCol=Arrays.asList(2, 5, 8);
            List rightCol=Arrays.asList(3, 6, 9);
            List cross1=Arrays.asList(1, 5, 9);
            List cross2=Arrays.asList(7, 5, 3);

            //The link that links to arrayList above^
            List<List>winning=new ArrayList<List>();
            winning.add(topRow);
            winning.add(midRow);
            winning.add(botRow);
            winning.add(leftCol);
            winning.add(midCol);
            winning.add(rightCol);
            winning.add(cross1);
            winning.add(cross2);

            //3 different ways the game can end, either player 1
or 2 wins or a draw.
            for(List l:winning) {
                 if(player1Positions.containsAll(l)) {
                      System.out.println("Player1 Wins! :)");
                      return true;
                 }else if(player2Positions.containsAll(l)) {
                      System.out.println("Player2 Wins! :)");
                      return true;
                 }else
if(player1Positions.size()+player2Positions.size()==9) {
                      System.out.println("It's a Draw! ;/");
                      return true;
                 }
            }
            return false;
       }
}
```