

# **X TIC TAC TOE GAME O**

## **UML PROJECT REPORT**

Academic Year 2022-23

ODD SEMESTER

(2018 Regulation)

Department with Specialization: Computer Science Engineering with  
Specialization in Artificial Intelligence & Machine Learning.

Semester: III.

Course Code: 18CSC202J.

Course Title: Object Oriented Design and Programming.

Submitted By:

**Nilay Kale (Reg No: RA2111026010287)**

**Anandkrishna V. (Reg No. RA2111026010285)**

Under the Guidance of:

Dr. M. Ferni Ukrit,

Associate Professor.



DEPARTMENT OF COMPUTATIONAL INTELLIGENCE COLLEGE OF  
ENGINEERING AND TECHNOLOGY SRM INSTITUTE OF SCIENCE AND  
TECHNOLOGY KATTANKULATHUR- 603 203

NOVEMBER 2022

## **BONAFIDE**

This is to certify that **18CSC202J - OBJECT ORIENTED DESIGN AND PROGRAMMING LABORATORY Project Report** titled  
**“TIC TAC TOE GAME”** is the Bonafide work of

Nilay Kale (RA2111026010287)

Anandkrishna V. (RA2111026010285)

who undertook the task of completing the project within the allotted time.

**Signature of the Guide**

Dr. M. Ferni Ukrit

**Associate Professor**

Department of CINTEL,

SRM Institute of Science and Technology

**Signature of the II Year Academic Advisor**

Dr. N. Arivazhagan

**Professor and Head**

Department of CINTEL

SRM Institute of Science and Technology

**About the course: -**

18CSC202J/ 8AIC203J - Object Oriented Design and Programming are 4 credit courses with **L T P C as 3-0-2-4** (Tutorial modified as Practical from 2018 Curriculum onwards)

**Objectives:**

The student should be made to:

- Learn the basics of OOP concepts in C++
- Learn the basics of OOP analysis and design skills.
- Be exposed to the UML design diagrams.
- Be familiar with the various testing techniques

**Course Learning Rationale (CLR): The purpose of learning this course is to:**

1. Utilize class and build domain model for real-time programs
2. Utilize method overloading and operator overloading for real-time application development programs
3. Utilize inline, friend and virtual functions and create application development programs
4. Utilize exceptional handling and collections for real-time object-oriented programming applications
5. Construct UML component diagram and deployment diagram for design of applications
6. Create programs using object-oriented approach and design methodologies for real-time application development

**Course Learning Outcomes (CLO): At the end of this course, learners will be able to:**

1. Identify the class and build domain model
2. Construct programs using method overloading and operator overloading
3. Create programs using inline, friend and virtual functions, construct programs using standard templates
4. Construct programs using exceptional handling and collections
5. Create UML component diagram and deployment diagram
6. Create programs using object-oriented approach and design methodologies

**Table 1: Rubrics for Laboratory Exercises**  
(Internal Mark Splitup:- As per Curriculum)

<b>CLAP-1</b>	5=(2(E-lab Completion) + 2(Simple Exercises)( from CodeZinger, and any other coding platform) + 1(HackerRank/Code chef/LeetCode Weekend Challenge)	Elab test
<b>CLAP-2</b>	7.5=(2.0(E-lab Completion)+ 2.0 (Simple Exercises)( from CodeZinger, and any other coding platform) + 3.5 (HackerRank/Code chef/LeetCode Weekend Challenge)	Elab test
<b>CLAP-3</b>	7.5=(2.0(E-lab Completion(80 Pgms)+ 2.0 (Simple Exercises)( from CodeZinger, and any other coding platform) + 3.5 (HackerRank/Code chef/LeetCode Weekend Challenge)	<b>2 Mark - E-lab Completion 80 Program</b> Completion from 10 Session (Each session min 8 program) <b>2 Mark - Code to UML</b> conversion GCR Exercises <b>3.5 Mark - Hacker Rank</b> Coding challenge completion
<b>CLAP-4</b>	5= 3 ( Model Practical) + 2( Oral Viva)	<ul style="list-style-type: none"> <li>• <b>3 Mark</b> – Model Test</li> <li>• <b>2 Mark</b> – Oral Viva</li> </ul>
<b>Total</b>	25	

### COURSE ASSESSMENT PLAN FOR OODP LAB

S.No	List of Experiments	Course Learning Outcomes (CLO)	Blooms Level	PI	No of Programs in each session
1.	Implementation of I/O Operations in C++	CLO-1	Understand	2.8.1	10
2.	Implementation of Classes and Objects in C++	CLO-1	Apply	2.6.1	10
3.	To develop a problem statement. 1. From the problem statement, Identify Use Cases and develop the Use Case model. 2. From the problem statement, Identify the conceptual classes and develop a domain model with a UML Class diagram.	CLO-1	Analysis	4.6.1	Mini Project Given
4.	Implementation of Constructor Overloading and Method Overloading in C++	CLO-2	Apply	2.6.1	10
5.	Implementation of Operator Overloading in C++	CLO-2	Apply	2.6.1	10
6.	Using the identified scenarios, find the interaction between objects and represent them using UML Sequence diagrams and Collaboration diagrams	CLO-2	Analysis	4.6.1	Mini Project Given
7.	Implementation of Inheritance concepts in C++	CLO-3	Apply	2.6.1	10
8.	Implementation of Virtual function & interface concepts in C++	CLO-3	Apply	2.6.1	10
9.	Using the identified scenarios in your project, draw relevant state charts and activity diagrams.	CLO-3	Analysis	4.6.1	Mini Project Given
10.	Implementation of Templates in C++	CLO-3	Apply	2.6.1	10
11.	Implementation of Exception of Handling in C++	CLO-4	Apply	2.6.1	10
12.	Identify the User Interface, Domain objects, and Technical Services. Draw the partial layered, logical architecture diagram with UML package diagram notation such as ComponentDiagram, Deployment Diagram.	CLO-5	Analysis	4.6.1	Mini Project Given
13.	Implementation of STL Containers in C++	CLO-6	Apply	2.6.1	10
14.	Implementation of STL associate containers and algorithms in C++	CLO-6	Apply	2.6.1	10
15.	Implementation of Streams and File Handling in C++	CLO-6	Apply	2.6.1	10

## LIST OF EXPERIMNENTS FOR UML DESIGN AND MODELLING:

**To develop a mini project by following the exerciseslisted below.**

1. To develop a problem statement.
2. Identify Use Cases and develop the Use Case model.
3. Identify the conceptual classes and develop a domainmodel with UML Class diagram.
4. Using the identified scenarios, find the interactionbetween objects and represent them  
using UML Sequence diagrams.
5. Draw relevant state charts and activity diagrams.
6. Identify the User Interface, Domain objects, and technical services.  
Draw the partial layered, logical architecture diagram with UML  
package diagram notation.

### Suggested Software Tools for UML:

StarUML, Rational Suite, Argo UML (or) equivalent,Eclipse IDE and Junit.

## Table of Contents

Abstract .....	
Class Diagram.....	
Relationship Among Classes .....	
Interaction Diagram.....	
Sequence Diagram.....	
Collaboration Diagram .....	
Behavioral Diagram .....	
State Chart Diagram .....	
Activity Diagram.....	
Package Diagram.....	
Component Diagram .....	
Deployment Diagram .....	
Conclusion .....	
References .....	



## Abstract:

Tic-Tac-Toe is a game in which two players take turns in drawing either an 'O' or an 'X' in one square of a grid consisting of nine squares. The winner is the first player to get three inline of the same symbols in a row.

In this project we've tried to show the basic working functionality of our Tic-tac-toe game. There will be two players competing against one another and the computer (the game board) will be validating the move after verifying it with the server.

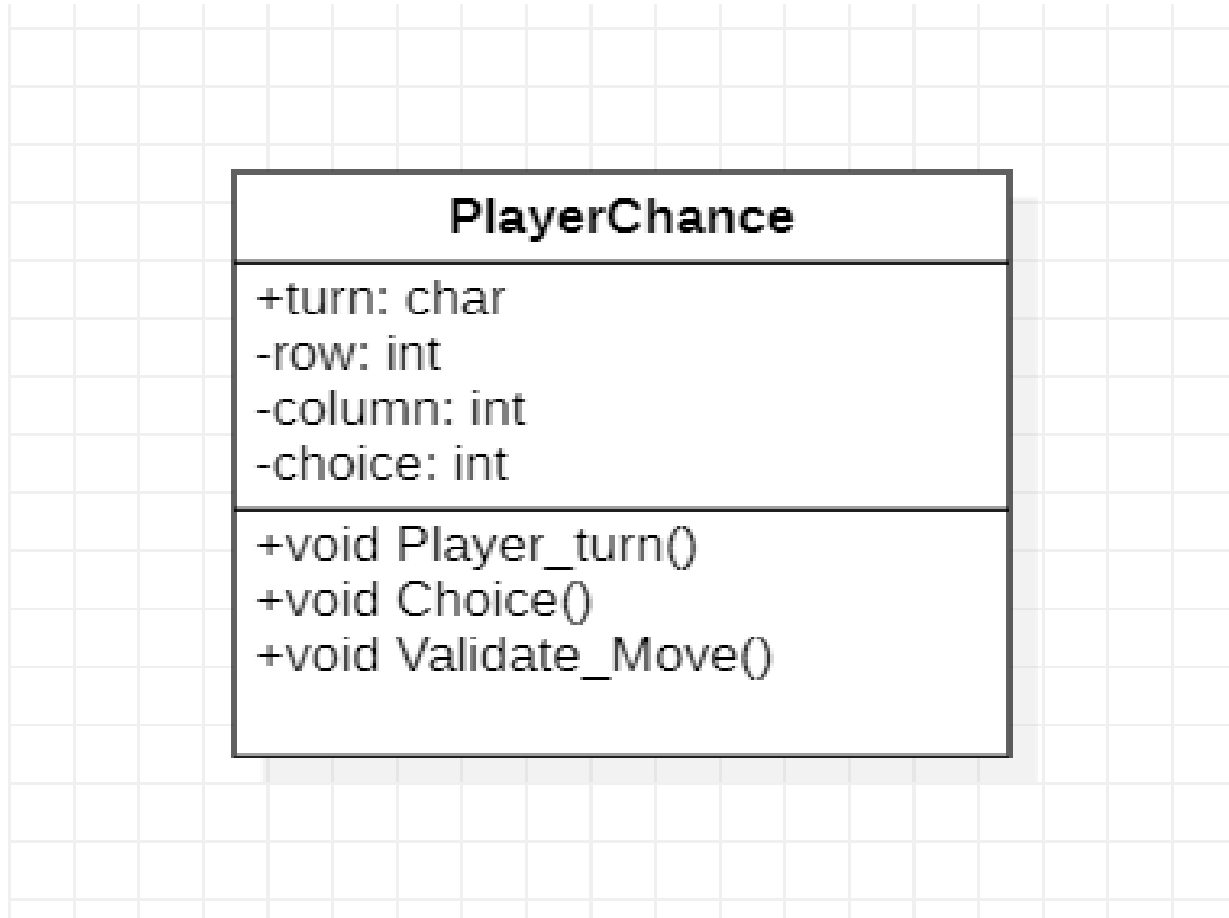
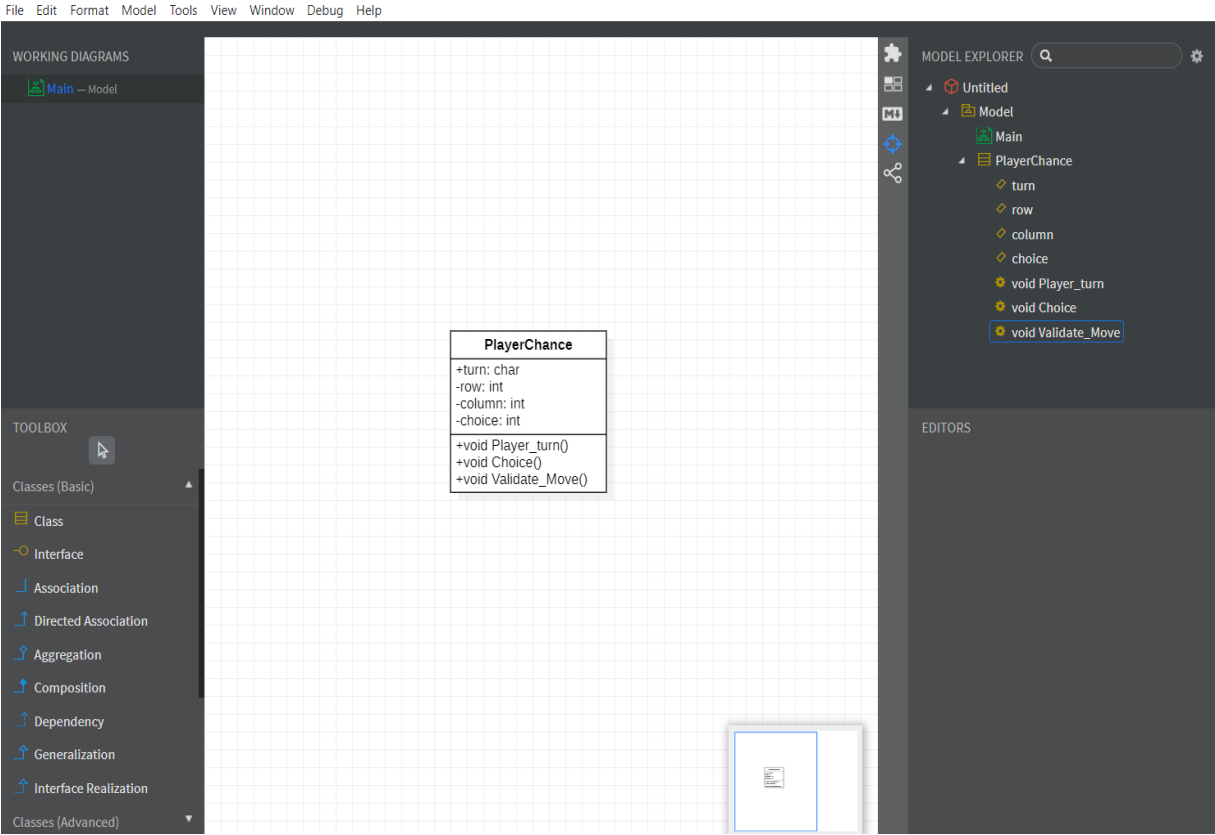
At first the player will be prompted to play a move on the game board. After this move, the second player plays the move. The game board verifies with the server if the move is valid or the space was already occupied. This process continues simultaneously until the game board has found three inline either of 'X' or of 'O'. If found, it will declare the respective player as the winner and if not, it declares the game as a draw.

We've tried to simplify and depicts these scenarios in real time using UML diagrams. Various diagrams show the dynamic and static modelling of various instances in the game in diagrammatic and sequential flow. These above features will make it easy for the user to understand the game and also have a fluid experience while playing it.



# **MODULE DESCRIPTION**

# Class Diagram:





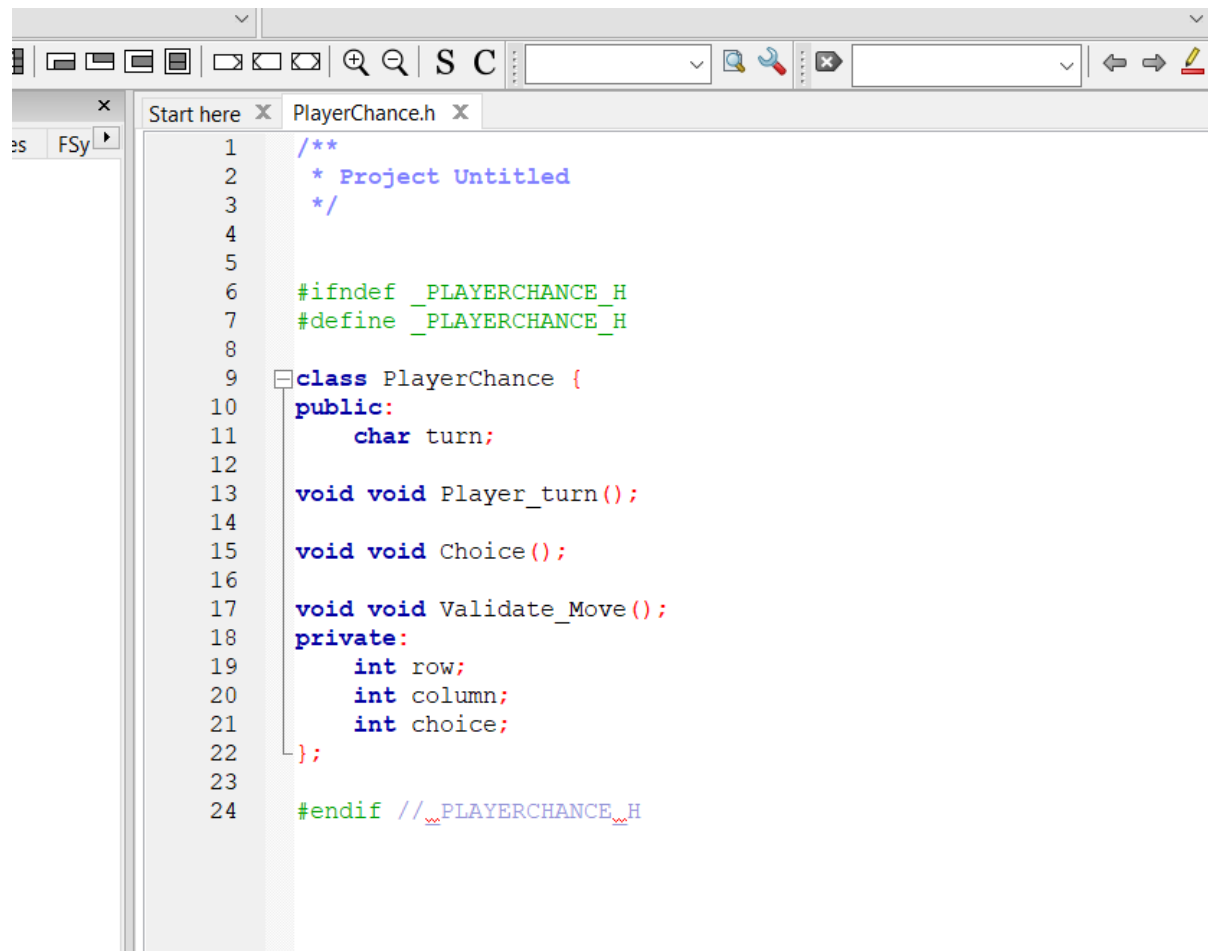
### **Relationship Among Classes :**

The following Tic Tac Toe Game has a number of different classes according to the various different actions to be performed. It includes classes for printing rules, game board, taking information input, giving result output, player chance and validating the move, which are all interrelated.

The above Class Diagram with Code Generation is for taking player chance and validating the move.

## Code Generation:

Given below is the code generated for the drawn classdiagram. It gives us the basic structure of the program:



```
1  /**
2   * Project Untitled
3   */
4
5
6  #ifndef _PLAYERCHANCE_H
7  #define _PLAYERCHANCE_H
8
9  class PlayerChance {
10 public:
11     char turn;
12
13     void void Player_turn();
14
15     void void Choice();
16
17     void void Validate_Move();
18 private:
19     int row;
20     int column;
21     int choice;
22 };
23
24 #endif // _PLAYERCHANCE_H
```

## Code:

Given below is the code written with the necessary changes and added logic to get our desired output for this particular class:

```
/**
 * Project Untitled
 */
#ifndef
_PLAYERCHANCE_H
#define
_PLAYERCHANCE_H
#include <iostream>
using namespace std;
class PlayerChance
{
private:
    int row;
    int column;
    int choice;
public:
    char turn;
    char board[3][3] = {{'1','2','3'},{'4','5','6'},{'7','8','9'}};
void Player_turn()
{
    if(turn == 'X')
    {
        cout<<"\n\tPlayer - 1 [X]'s turn : ";
    }
}
```

```
else if(turn == 'O')
{
    cout<<"\n\tPlayer - 2 [O]'s turn : ";
}
}
```

```
void Choice()
{
    cin>> choice;
    switch(choice)
    {
        case 1: row=0; column=0; break;
        case 2: row=0; column=1; break;
        case 3: row=0; column=2; break;
        case 4: row=1; column=0; break;
        case 5: row=1; column=1; break;
        case 6: row=1; column=2; break;
        case 7: row=2; column=0; break;
        case 8: row=2; column=1; break;
        case 9: row=2; column=2; break;
        default:
            cout<<"Invalid Move\n";
    }
}
```

```

void Validate_Move()
{
    if(turn == 'X' && board[row][column] != 'X' &&
board[row][column] != 'O')
    {
        board[row][column] = 'X';
        turn = 'O';
    }
    else if(turn == 'O' && board[row][column] != 'X' &&
board[row][column] != 'O')
    {
        board[row][column] = 'O';
        turn = 'X';
    }
    else
    {
        cout<<"Box is already filled!\nPlease choose another
one!!\n\n";
        Player_turn();
    }
}
};

int main()
{

```

```

PlayerChance
c1,c2,c3;
c1.Player_turn
();
c2.Choice();
c3.Validate_Move(); return
0;
}
#endif //_PLAYERCHANCE_H

```

### Sample Input & Output:

This is a sample output for the given input to this taking player chance and validating the move class:



The screenshot shows a C++ IDE with a code editor and a console window. The code editor displays the following code:

```

66 };
67 int main()
68 {

```

The console window shows the following output:

```

6
Box is already filled!
Please choose another one!!

...Program finished with exit code 0
Press ENTER to exit console.

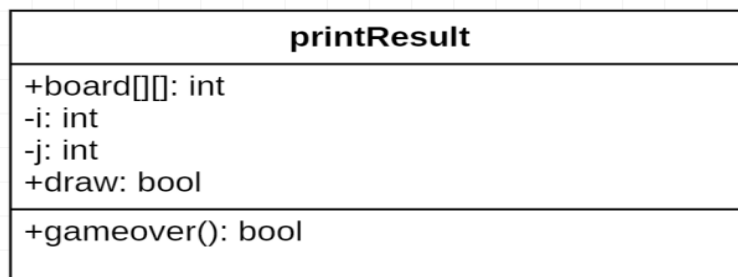
```



# Class Diagram

Window   Debug   Help

---



## Code:

```
/**
 * Project Untitled
 */

#include <iostream>
using namespace std;
class printResult {
public:
    int board[100][100];
    int i;
    int j;
    bool draw;
```

```

bool gameover(){

    for(int i=0; i<3; i++)
        if(board[i][0] == board[i][1] && board[i][0] == board[i][2]
        || board[0][i] == board[1][i] && board[0][i] == board[2][i])
            return false;

    if(board[0][0] == board[1][1] && board[0][0] ==
board[2][2] || board[0][2] == board[1][1] && board[0][2]
== board[2][0])
        return false;
    for(int i=0; i<3; i++)
        for(int j=0; j<3; j++)
            if(board[i][j] != 'X' && board[i][j] != 'O')
                return true;

    draw = true;
    return false;
}
};
int main(){
    return 0;
}

```

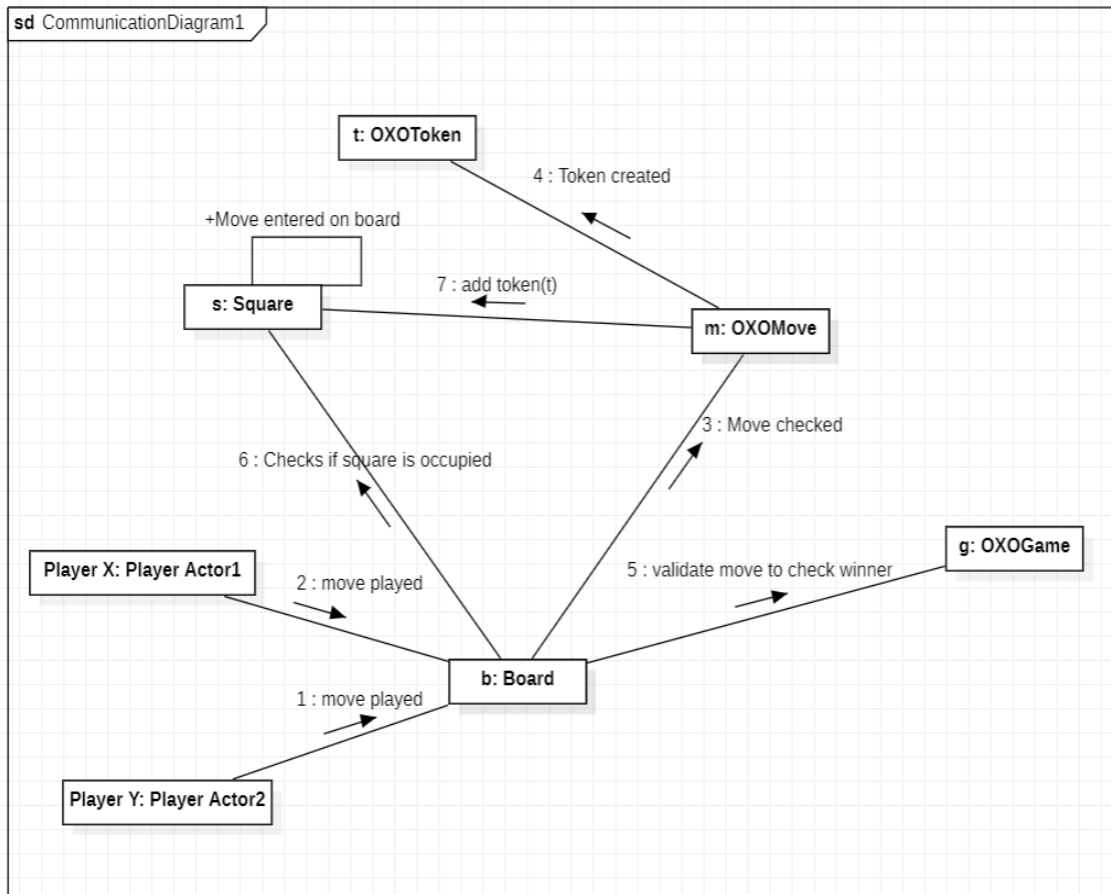
```

#endif //_PRINTRESULT_H

```

# UML: Interaction Diagrams.

## Collaboration Diagram in UML:



## Explanation:

The human player interacts with a screen representation of the Tic Tac Toe board, controlled by the Board. The player clicks somewhere on the board, which a message to be sent to the Board,

with information about where the click happened/move was played. The Board, with help from its squares, works out from the co-ordinates of which Square of the Board has been clicked in. The Board knows that it is Player X's or Y's turn, so the meaning of the click is that the player wishes to place a token on the square. It creates a new OXOMove object which knows which square and which Player are concerned, and passes the OXOMove to OXOGame for validation. OXOGame checks that there is not yet any OXOToken on the square. If there isn't, then it is a valid move. OXOGame must then check whether this move finishes the game. If it does not, OXOGame's reply to the validate message says that the move is OK and that O is now to move. Board asks the OXOMove to update the position it displays to the user, so the OXOMove creates a new X OXOToken on the square that it refers to, forgets the

OXOMove object which is no longer needed (so it will be garbage collected), records that the next move is to be made by O, and waits for the next click, which will represent O's move.

We have decided that it is not an OXOToken's job to know where it is; rather, a square knows about an OXOToken which is on it, and a square knows its own position.

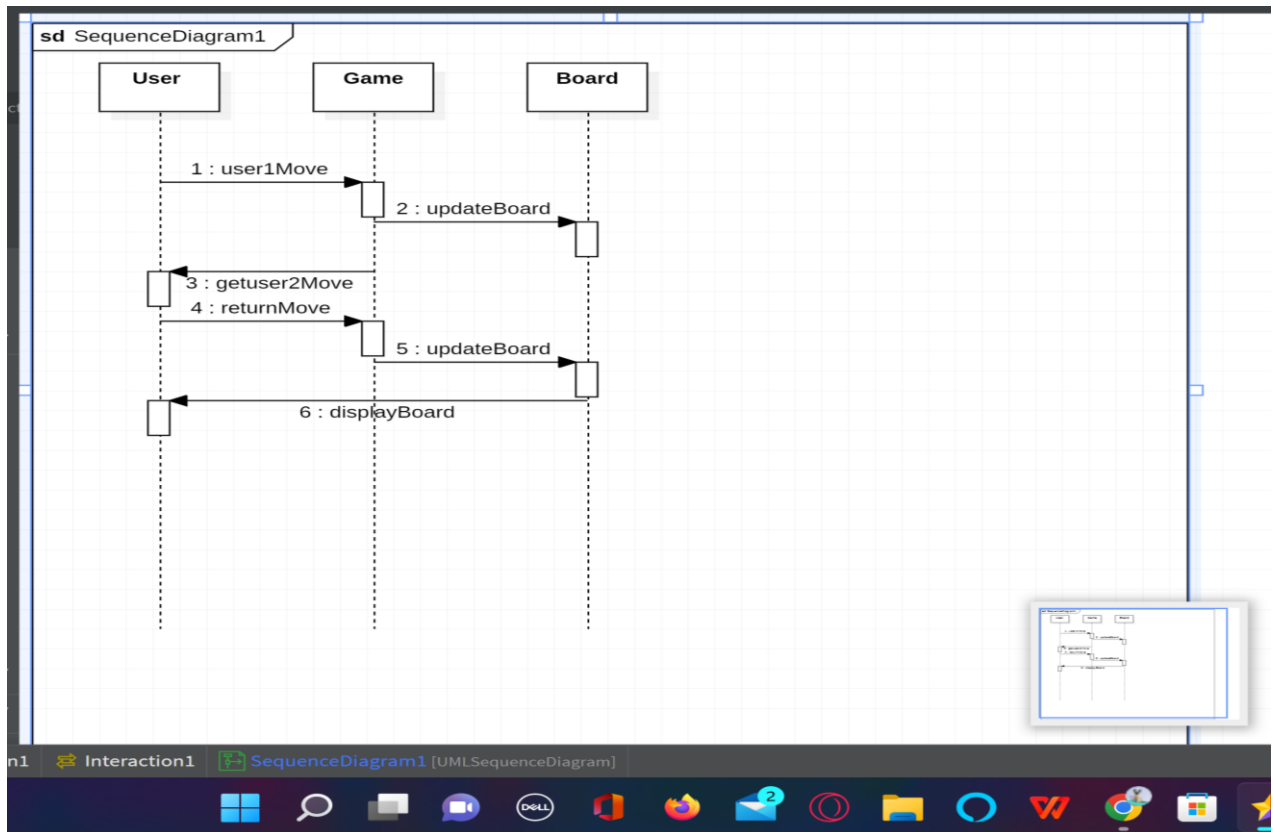
Crystallizing this into messages passing, we get this following figure as the collaboration diagram for a correct Tic Tac Toe (Noughts and Crosses) move by the player.

Thus, this collaboration diagram accounts for all the things we need:

- Accepting the player's move.
- Checking the move.
- Checking whether the square was already occupied or not.
- Creating token to process OXO moves.

- Validate Move to check the winner.

## Sequence Diagram



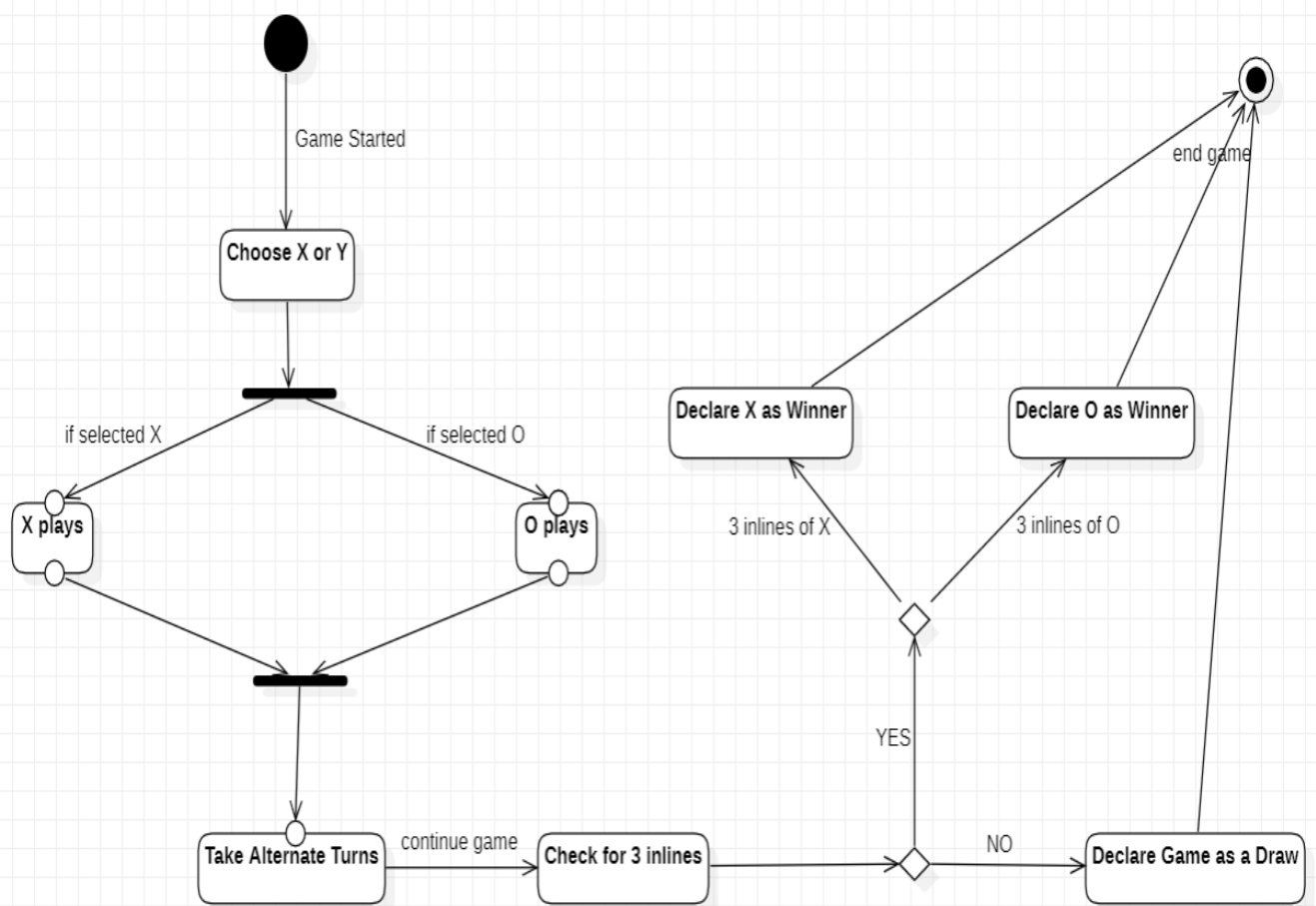
## Explanation:

The program accepts input from user1. The program then updates the changes made by the user1 input on the board. It then requests the user2 for input and after accepting the input

again updates the board according to the respective input. Then it displays the updated board on the computer display. This process continues until one of the said user wins the Tic Tac Toe game.

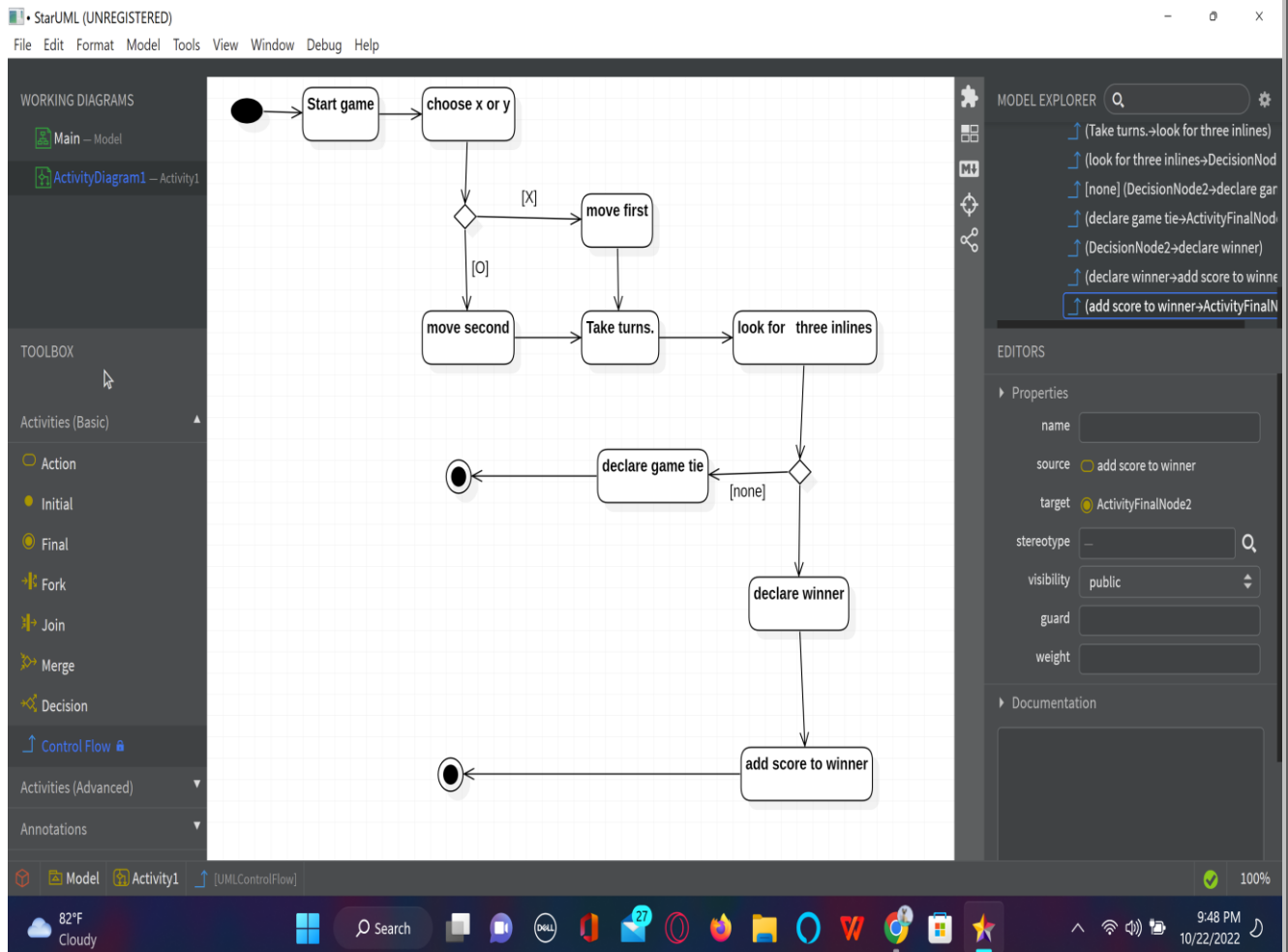
# UML: State Chart and Activity Diagrams.

## State Chart Diagram:





# Activity Diagram:



## Report:

A state diagram is used to represent the condition of the system or a part of the system at finite instances of time. It's a behavioural diagram and it represents

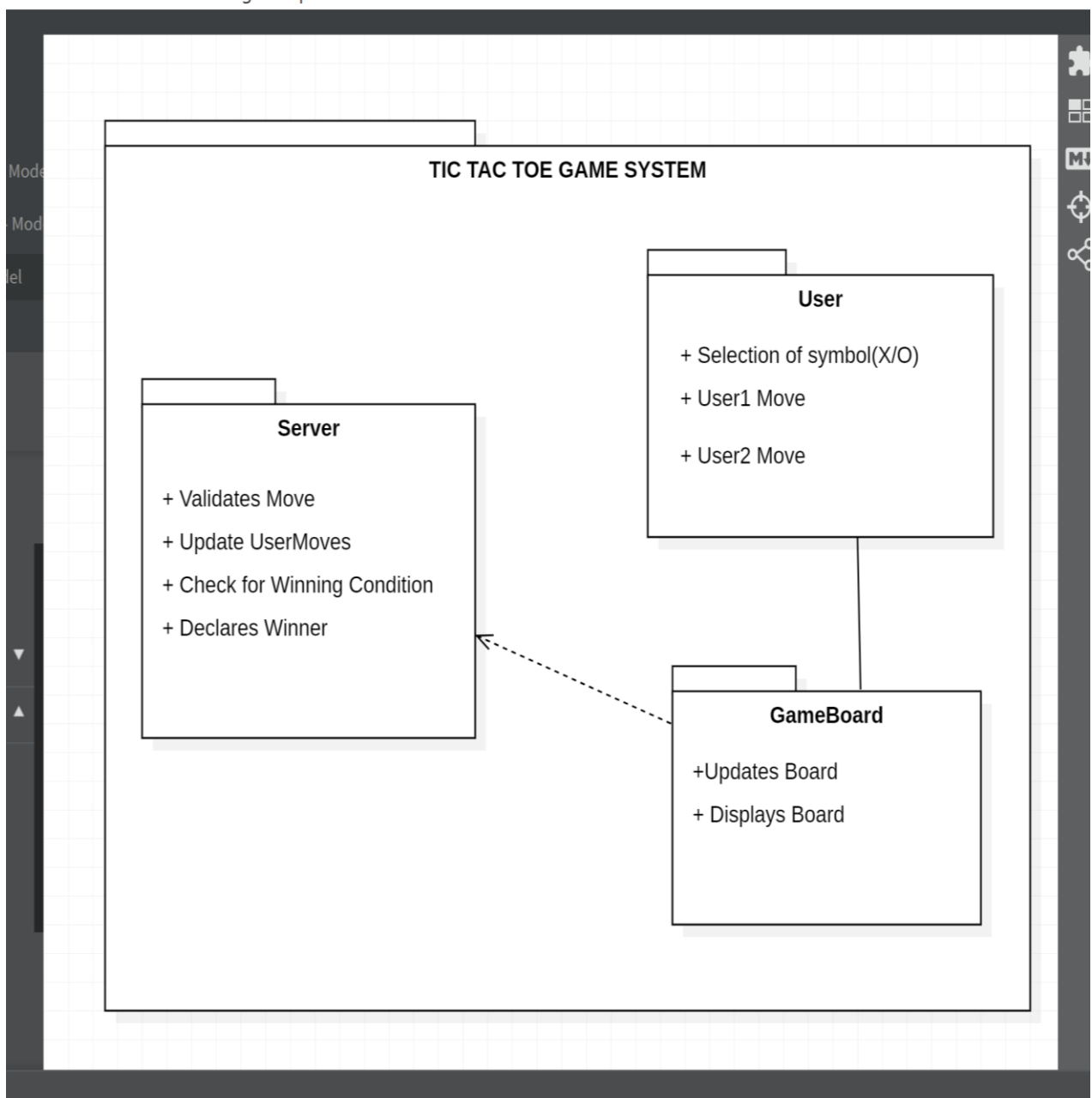
the behaviour using finite state transitions.

Here in this Diagram the various stages are depicted in step wise order. First the game starts and either “X” or “O” is selected. Accordingly, each take turns chance by chance and play their move until the game reaches an end. The computer then checks for 3 inlines in a row, either of X or O. If found, it accordingly will declare that player as the winner. If not found, the game will be declared as a draw and it ends.

**UML: partial layered, logical architecture diagram with UML package diagram notation such as Package Diagram, Component Diagram, Deployment Diagram.**

**Package Diagram:**

Tools View Window Debug Help



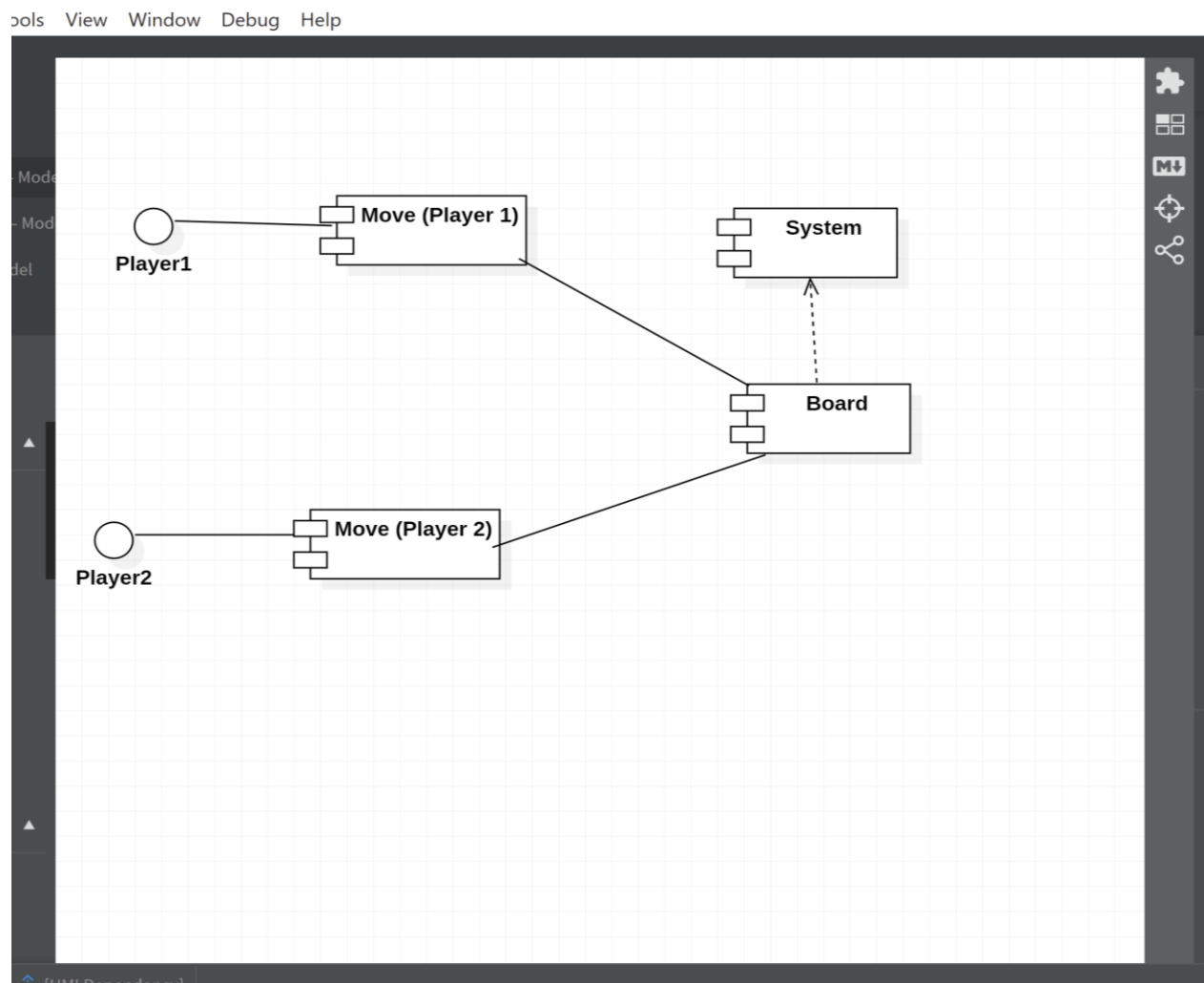
## Explanation:

Package diagram provides static models of modules, their parts and their relationships and presents the architectural modelling of the system.

Here it groups the UML elements, to specify the logical distribution of classes. This diagram emphasizes the logical distribution of classes which is inferred from the logical architecture of the system.

First we have the main Tic-tac-toe game package with other packages inside it such as User for selecting your symbol and for each player to alternately play their move. Gameboard package for updating and displaying the current status of the game and the Server package which validates all moves and after verifying the winning condition, declares a final winner.

# Component Diagram:

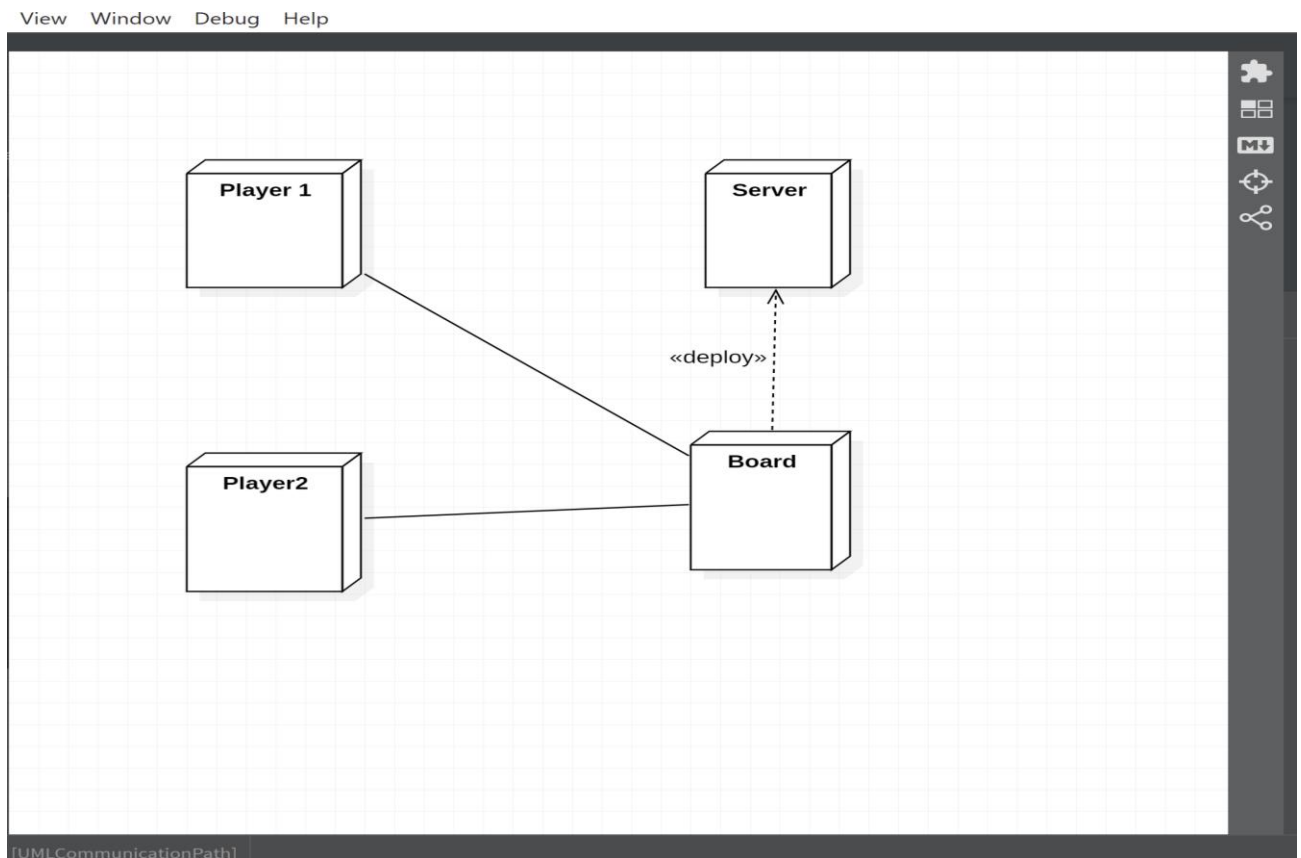


## Explanation:

A component diagram shows the physical view of the system. A component is an autonomous unit within a system. We combine packages or individual entities to form components.

It helps to specify a physical database. It enables to model the high level software components and the interfaces to those components. The program accepts Player1 input which then is passed to the server. The server validates this input and requests for input from Player2. Then the server accepts the player 2 input and validates it. The server finally updates the Player inputs. The updated information is then passed to the Game board which then updates the game board and displays it.

## Deployment Diagram:



## Explanation:

A deployment diagram shows the physical placement of components in nodes over a network. A deployment diagram can be drawn by identifying nodes and components. A deployment diagram usually describes the resources required for processing and the installation of software components in those resources.

## Conclusion:

Here were the UML Diagrams that composes a Tic tac toe game. Each of the UML Diagrams has a major role in achieving a well-developed and provide a fluid game experience.

The UML Diagrams works together to achieve the most desired functions of the Tic-Tac-Toe Project. All of these were designed to guide programmers and beginners about the behavior and structure of various UML diagrams used in this game.

By completing all the given Diagrams, the game development would be much easier and attainable. So those UML diagrams were given to teach and guide one through their project development journey. You can use all the given UML diagrams as a reference or have them for a project development. The ideas presented in UML Diagrams were all based on the game requirements.

## References:

<https://techbyexample.com/system-design-tic-tac-toe-game/>

<https://medium.com/@pelensky/java-tic-tac-toe-uml-diagram-329803b97bb2>

[https://www.tutorialspoint.com/uml/uml\\_standard\\_diagrams.htm](https://www.tutorialspoint.com/uml/uml_standard_diagrams.htm)

<https://www.javatpoint.com/uml-diagrams>