TIC-TAC-TOE GAME DEVELOPMENT USING MUTUAL EXCLUSION

PROJECT REPORT

Submitted by

SATVIK SHARMA (RA2211029010003)

PREM LOHIA (RA2211029010007)

ANITEJ MISHRA (RA2211029010023)

Under the guidance of

Dr P. Mahalakshmi

Assistant Professor, Department of Networking and Communications

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

with specialization in Computer Networking

DEPARTMENT OF NETWORKING AND COMMUNICATIONS

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR-603 203

OCTOBER 2023

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR-603 203

BONAFIDE CERTIFICATE

Certified that this Project Report titled "**TIC-TAC-TOE GAME DEVELOPMENT USING MUTUAL EXCLUSION**" is the bonafide work done by:

SATVIK SHARMA (RA2211029010003)

PREM LOHIA (RA2211029010007)

ANITEJ MISHRA (RA2211029010023)

who completed the project under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

SIGNATURE

Dr P. Mahalakshmi

OS-Course Faculty

Assistant Professor

Department of Networking and Communications

SRMIST

SIGNATURE

Dr Annapurani Panaiyappan

Head of the Department

Department of Networking and Communications

SRMIST

Department of Networking and Communications

SRM Institute of Science and Technology

Own Work Declaration Form

Degree/Course: B. Tech in Computer Science and Engineering with specialization in Computer Networking

Names of the Students: Satvik Sharma, Prem Lohia and Anitej Mishra

Registration Numbers: RA2211029010003, RA2211029010007, RA2211029010023

Title of Work: Tic-Tac-Toe Game Development using Mutual Exclusion

We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc.)
- Given the sources of all pictures, data etc. that are not our own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (eg. fellow students, technicians, statisticians, external sources)
- Complied with any other plagiarism criteria specified in the course handbook/University website

We understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my/our own work, except where indicated by referring, and that I have followed the good academic practices noted above.
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

ACKNOWLEDGMENT

We express our humble gratitude to **Dr C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr T. V. Gopal**, for his invaluable support.

We wish to thank **Dr Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department, **Dr Annapurani Panaiyappan,** Professor, Department of Networking and Communications, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We register our immeasurable thanks to our Faculty Advisor, **Dr S. Ushasukhanya**, Assistant Professor, Department of Networking and Communications, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Dr P. Mahalakshmi**, Associate Professor, Department of Networking and Communications, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under his mentorship. She provided us with the freedom and support to explore the research topics of our interest. Her passion for solving problems and helping us solve our problems has always been inspiring.

We sincerely thank **the staff, faculty and the students of the Department of Networking and Communications**, SRM Institute of Science and Technology, for their help during our project.

Finally, we would like to thank **our parents, family members, and friends** for their unconditional love, constant support, and encouragement.

TABLE OF CONTENTS

ABSTRACT

Mutual exclusion is a fundamental concept in computer science and concurrent programming, ensuring that only one process or thread can access a critical section of code at any given time.

In the context of game development, mutual exclusion plays a crucial role in maintaining game integrity and preventing conflicts between players. This abstract explores the application of mutual exclusion in the classic game of Tic-Tac-Toe and its implications for creating a fair and engaging gaming experience.

Tic-Tac-Toe is a two-player, turn-based game where each player strives to create a winning pattern of their symbols on a 3x3 grid. To implement mutual exclusion in this game, developers must address several key challenges.

First, they need to ensure that only one player can make a move at a time to prevent simultaneous moves, which would disrupt the game's logic. Second, developers must handle the synchronization of player inputs and game state updates to maintain consistency and fairness. Finally, mutual exclusion mechanisms must prevent unauthorized access to game data to prevent cheating and maintain a level playing field.

Chapter 1

INTRODUCTION

In the realm of game development, creating engaging and interactive experiences for players is a complex art. It involves a multitude of factors, including graphics, sound, user interface design, and most importantly, gameplay mechanics. One fundamental aspect of gameplay mechanics is ensuring that players interact with the game world in a consistent and conflict-free manner. This is where the concept of mutual exclusion comes into play, and what better way to explore this concept than by examining it through the lens of the classic game of Tic-Tac-Toe?

Tic-Tac-Toe is a two-player game that dates back centuries. Despite its simplicity, it provides an excellent platform for understanding the importance of mutual exclusion in game development. In this game, two players take turns placing their symbols on a 3x3 grid, aiming to form a row, column, or diagonal of their symbols while preventing their opponent from doing the same. The key challenge here is to ensure that both players cannot occupy the same grid cell simultaneously, illustrating the concept of mutual exclusion.

This exploration into mutual exclusion within the context of Tic-Tac-Toe will delve into several crucial aspects of game development:

Game Logic: Understanding the logic behind Tic-Tac-Toe, which involves ensuring that players take turns and enforcing rules to prevent multiple symbols from occupying the same space simultaneously.

User Interface: Designing an intuitive and responsive user interface that allows players to interact with the game grid seamlessly while maintaining mutual exclusion.

Multiplayer Experience: Exploring how mutual exclusion is maintained in both local and online multiplayer versions of Tic-Tac-Toe, ensuring a fair and enjoyable experience for all players.

Game State Management: Analyzing how game states are managed to keep track of the progress of the game, including detecting when a player has won or when the game ends in a draw.

Error Handling: Handling exceptional cases and errors to prevent unintended consequences, such as handling illegal moves or disconnects in online multiplayer scenarios.

Chapter 2

LITERATURE SURVEY

1. **“A Review of various Mutual Exclusion Algorithms in Distributed Environment”**
   Authors: Nisha Yadav, Sudha Yadav, Sonam Mandiratta
   Publication Year: 2015
   We referred to this research paper to understand the concepts of mutual exclusion and implement it in our project for developing a fair, simple game of tic tac toe using C.
2. **“Implementation of Tic-Tac-Toe Game in LabVIEW”**
   Authors: Lalitha Saroja Thota, Naseema Shaik, Rawan Ali and Others.
   Publication Year: 2014
   We referred to this research paper to understand the concepts of mutual exclusion and implement it in our project for developing a fair, simple game of tic tac toe using C.
3. **“Operating System Concepts”**
   Authors: Abraham Silberschatz, Peter Baer Galvin, Greg Gagne
   Publication Year: 2018
   Publisher: Wiley
   We referred to this book to understand what is Mutual Exclusion, Critical Section and how to use these concepts to write a code for a game of Tic Tac Toe.
   Link for the references
   https://drive.google.com/drive/folders/1MZzV61He-lyMkmOM0GXJP-khfBFx7pzQ

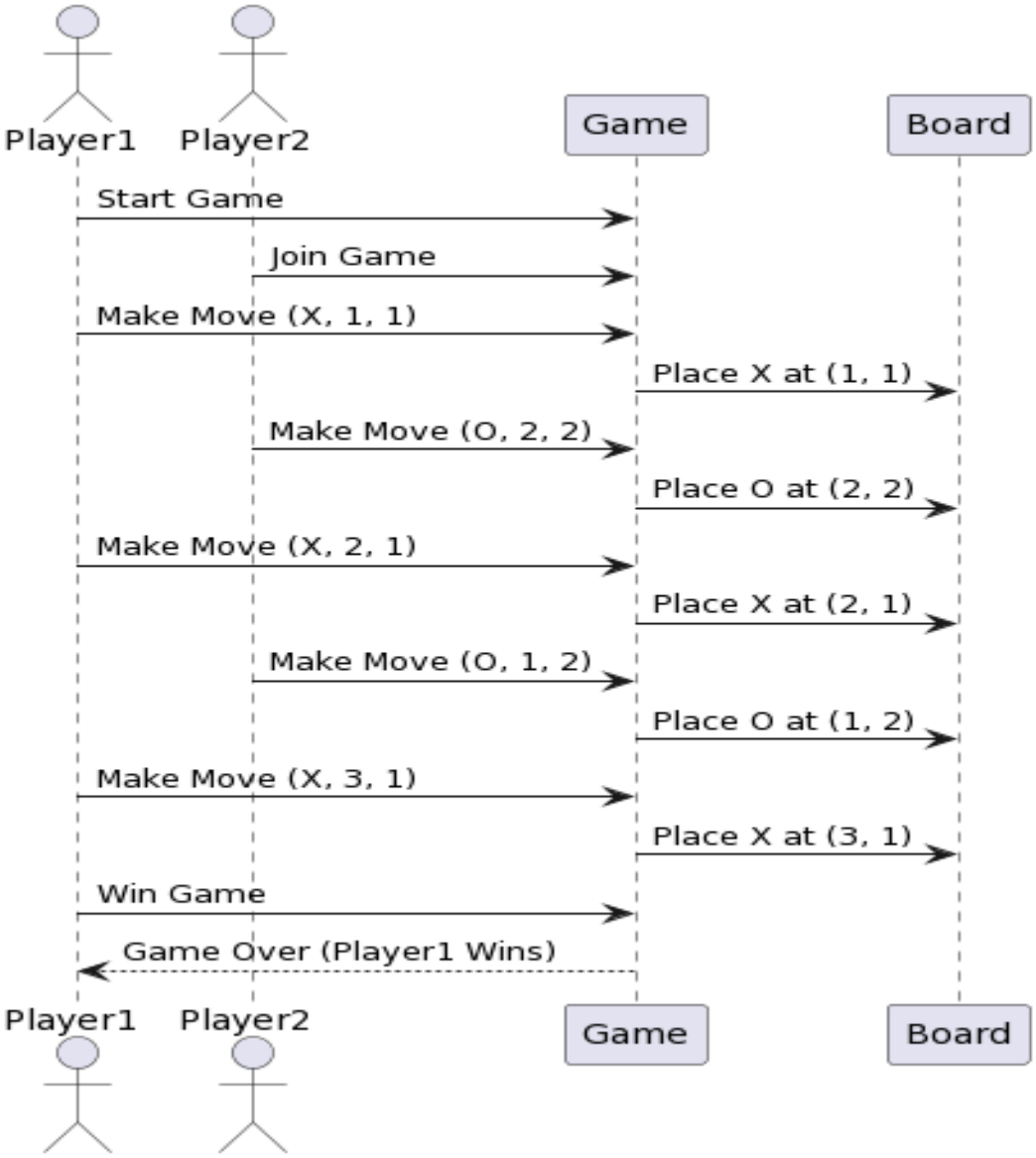
QR Code

Chapter 3

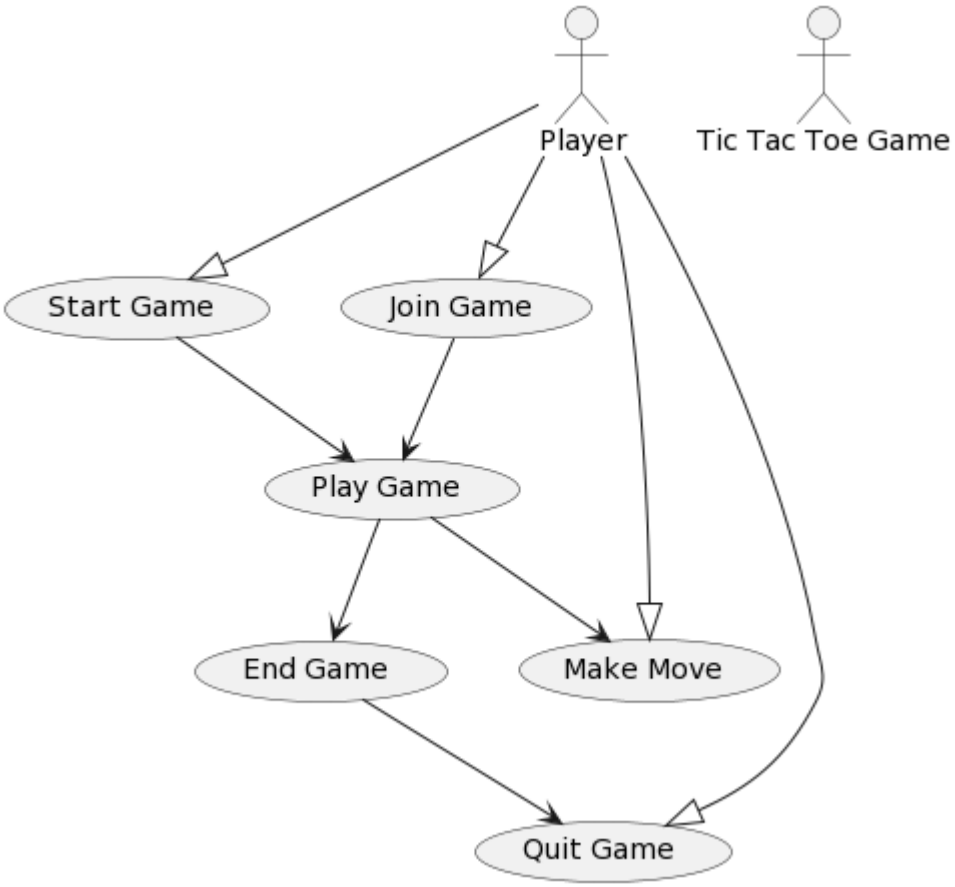
SYSTEM AND DESIGN

Part I: UML Diagrams

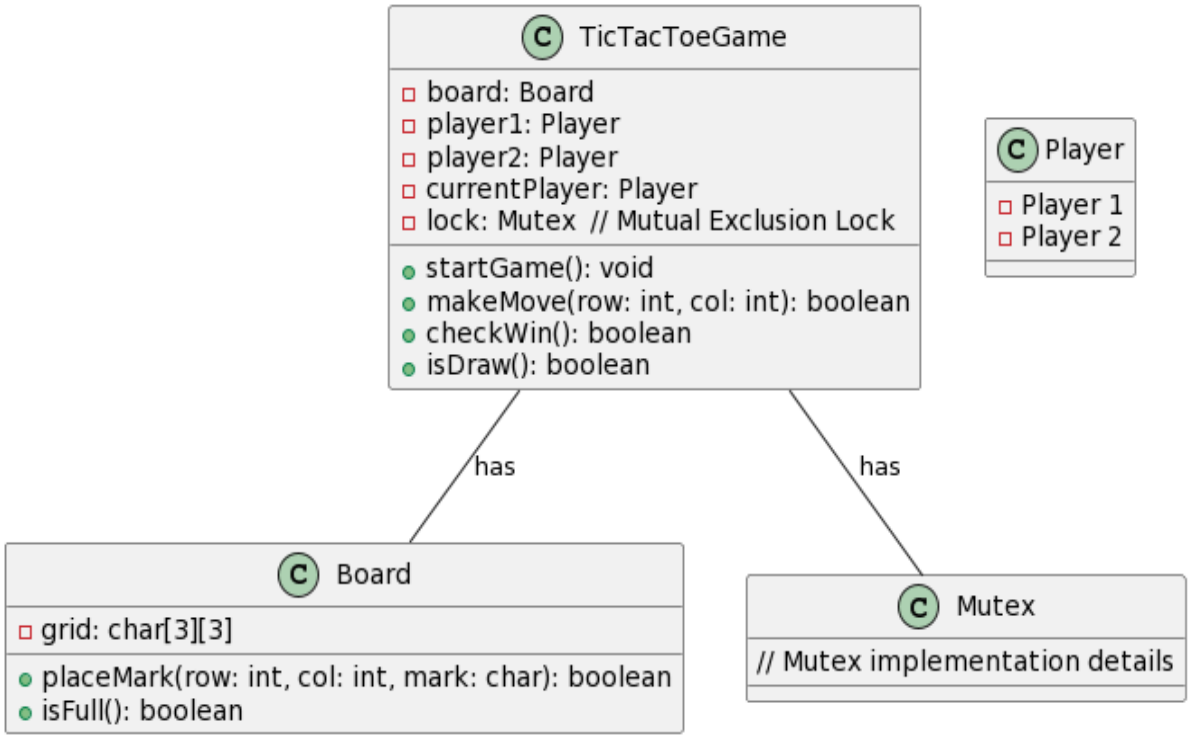
1. Sequence Diagram



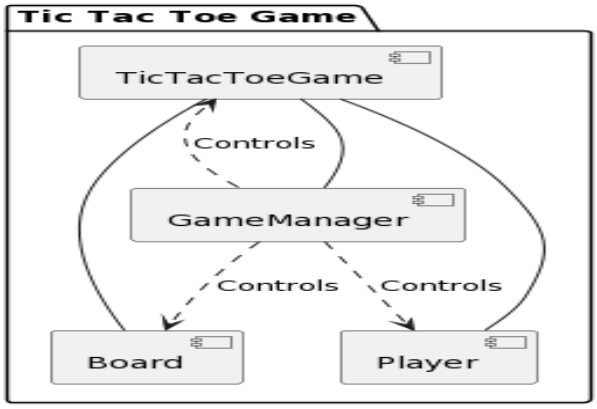
2. Use Case Diagram



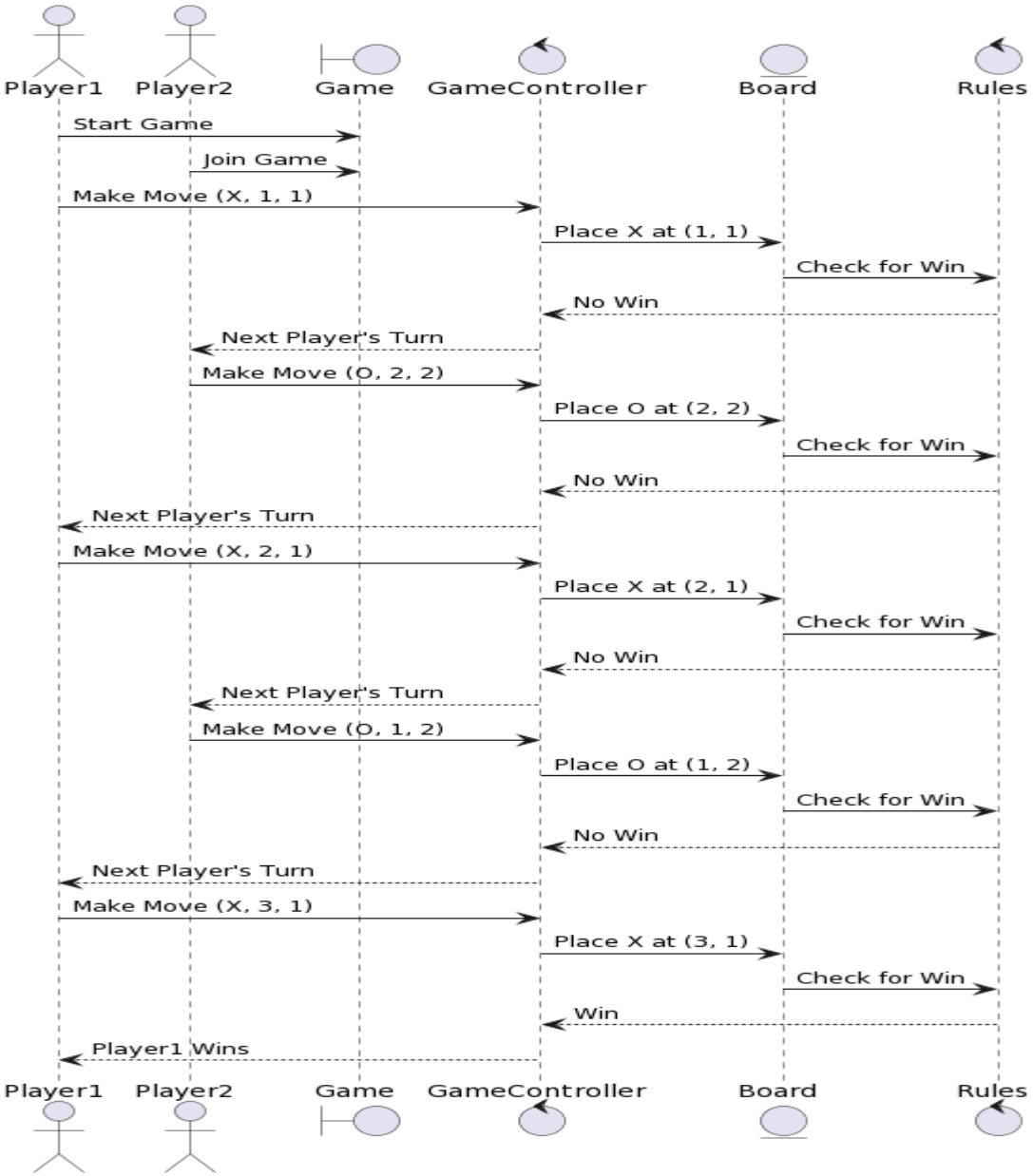
3. Class Diagram



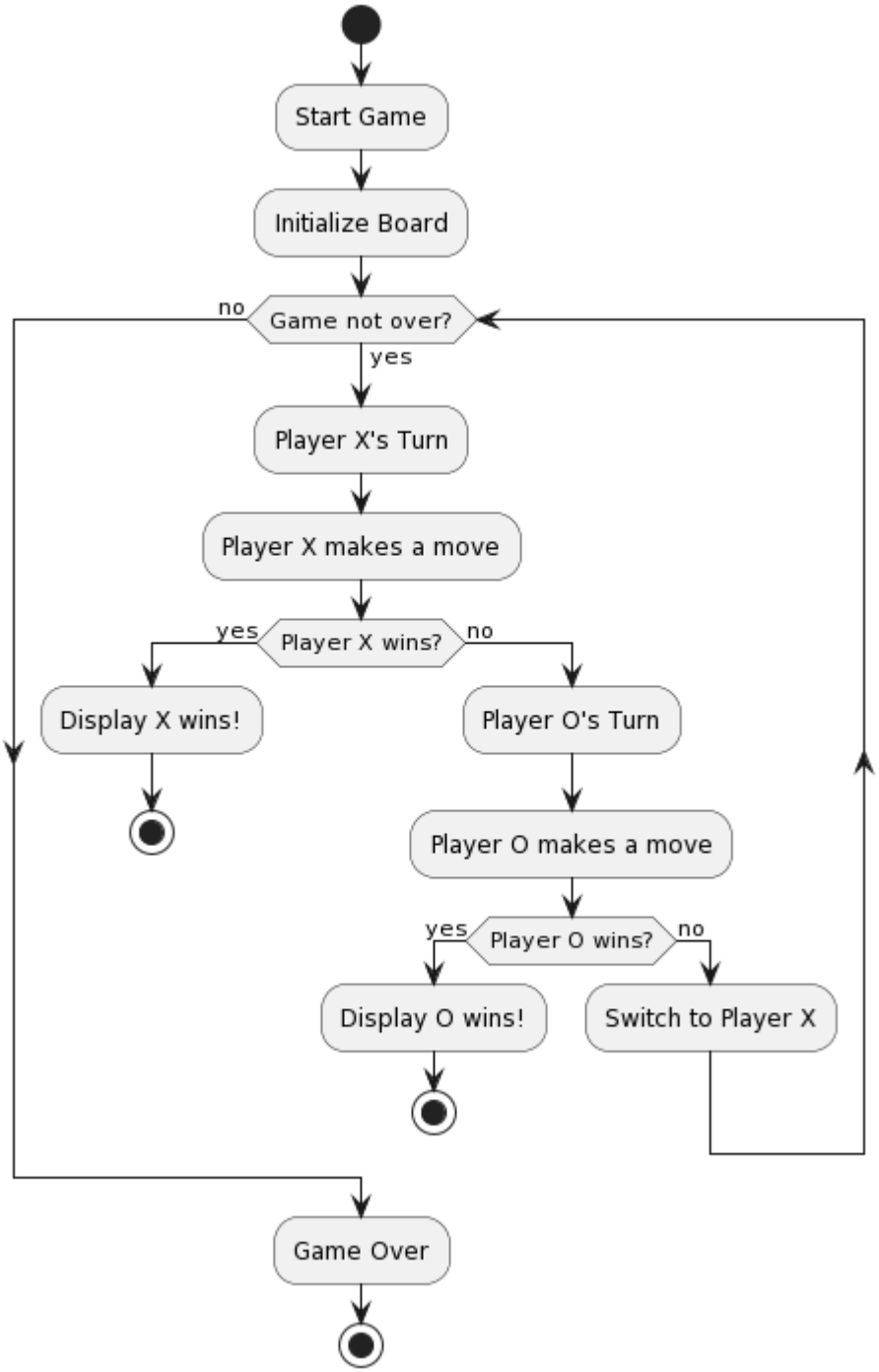
4. Component Diagram



5. Collaboration Diagram



6. Activity Diagram



Part II: Design and Algorithm Used

1. **initialize_board() function:**
   Initializes the game board by setting all elements to spaces (' ') to represent empty cells.

2. **display_board() function:**
   Displays the current state of the game board, including the positions of 'X' and 'O' as well as empty cells.

3. **make_move(int row, int col) function:**
   I. Checks if a move is valid by verifying that the specified row and column are within the board boundaries and the cell is empty.
   II. If the move is valid, it updates the board with the current player's symbol ('X' or 'O') and increments the moves counter.
   III. Returns true for a valid move and false for an invalid move.

4. **check_winner() function:**
   I. Checks for a winning condition for the current player by examining rows, columns, and diagonals on the game board.
   II. If a winning condition is met, it returns true. Otherwise, it returns false.

5. **player_thread(void* player_name) function:**
   I. This function represents the behavior of a player thread.
   II. It runs in a loop as long as there are available moves on the board (moves < 9 for a 3x3 board).
   III. It uses a mutex (mutex) to ensure that only one player can make a move at a time to prevent race conditions.
   IV. Displays the current state of the board and prompts the current player for their move (row and column).
   V. Calls make_move() to make the move if it's valid. If the move results in a win or a draw, it exits the thread.
   VI. Switches the current player for the next iteration.

6. **main() function:**
    I. Initializes the game board using initialize_board().
    II. Creates two player threads (player_X_thread and player_O_thread) representing players 'X' and 'O'.
    III. Each thread runs the player_thread function and passes the player's name ('X' or 'O') as an argument.
    IV. Waits for both player threads to finish using pthread_join().
    V. The game concludes when one of the players wins or there's a draw.

Chapter 4

SYSTEM REQUIREMENTS

1. **OPERATING SYSTEM**
   Tic Tac Toe game development can be done on various operating systems, including Windows, macOS, and Linux. We can choose the one that we are most comfortable with.

2. **DEVELOPMENT ENVIRONMENT**
   We can use a variety of programming languages and frameworks for developing Tic Tac Toe, such as Python, JavaScript (for web-based games), or even game development platforms like Unity or Godot. For our project we have chosen C in an Ubuntu terminal through a Virtual Machine.

3. **HARDWARE**
   We don't need a high-end computer for this type of game development. A basic desktop or laptop with at least 4GB of RAM and a modern multi-core processor should suffice.

4. **GRAPHICS**
   Tic Tac Toe is a simple 2D game, so we don't need a powerful graphics card. Integrated graphics on most modern computers will be more than enough.

5. **STORAGE**
   We don't need much storage space for code and assets. A few gigabytes should be sufficient.

6. **MUTUAL EXCLUSION MECHANISM**
   To implement mutual exclusion for multiplayer or multi-threaded gameplay, we'll need a basic understanding of concurrency control mechanisms in our chosen programming language (in our case, C). This might involve using mutexes, semaphores, or other synchronization techniques.

7. **TESTING**
   Testing the game will require a computer or devices to simulate multiple players. This can be done on a single machine with multiple instances of the game running simultaneously.

Chapter 5

IMPLEMENTATION

Implementing mutual exclusion in a Tic Tac Toe game is important to ensure that only one player can make a move at a time, preventing conflicts and ensuring fair gameplay. In a game development context, mutual exclusion is typically achieved through locking mechanisms or synchronization techniques. Here's an overview of how we can implement mutual exclusion in a Tic Tac Toe game:

- **GAME STATE REPRESENTATION**
  Represent the Tic Tac Toe game state using a data structure, such as a 2D array, to track the positions of X's and O's on the board.
- **PLAYER TURN**
  Maintain a variable to keep track of whose turn it is, whether it's the X player or the O player.
- **MUTUAL EXCLUSION**
  Use a mutex (short for mutual exclusion) or a similar synchronization mechanism to ensure that only one thread or player can access and modify the game state at a time.
- **PLAYER INPUT HANDLING**
  When a player wants to make a move, they should first acquire the mutex or lock to gain exclusive access to the game state.
- **MOVE VALIDATION**
  Check if the move requested by the player is valid (e.g., the selected cell is empty). If the move is invalid, release the mutex and ask the player for another move.
- **UPDATING GAME STATE**
  If the move is valid, update the game state with the player's symbol (X or O) in the selected cell.
- **CHECK FOR A WIN OR DRAW**
  After each move, check if the game has been won by one of the players or if it has resulted in a draw.
- **RELEASE MUTEX**
  After updating the game state and determining the game's outcome, release the mutex to allow the other player to make their move.

CODE

```
#include <stdio.h>
#include <stdbool.h>
#include <pthread.h>
#define BOARD_SIZE 3
char board[BOARD_SIZE][BOARD_SIZE];
char currentPlayer = 'X';
int moves = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
bool game_over = false;
void initialize_board()
{
   for (int i = 0; i < BOARD_SIZE; i++)
   {
      for (int j = 0; j < BOARD_SIZE; j++)
      board[i][j] = ' ';
   }
}
void display_board()
{
   for (int i = 0; i < BOARD_SIZE; i++)
   {
      for (int j = 0; j < BOARD_SIZE; j++)
      printf("| %c ", board[i][j]);
      printf("|\n");
   }
}
```

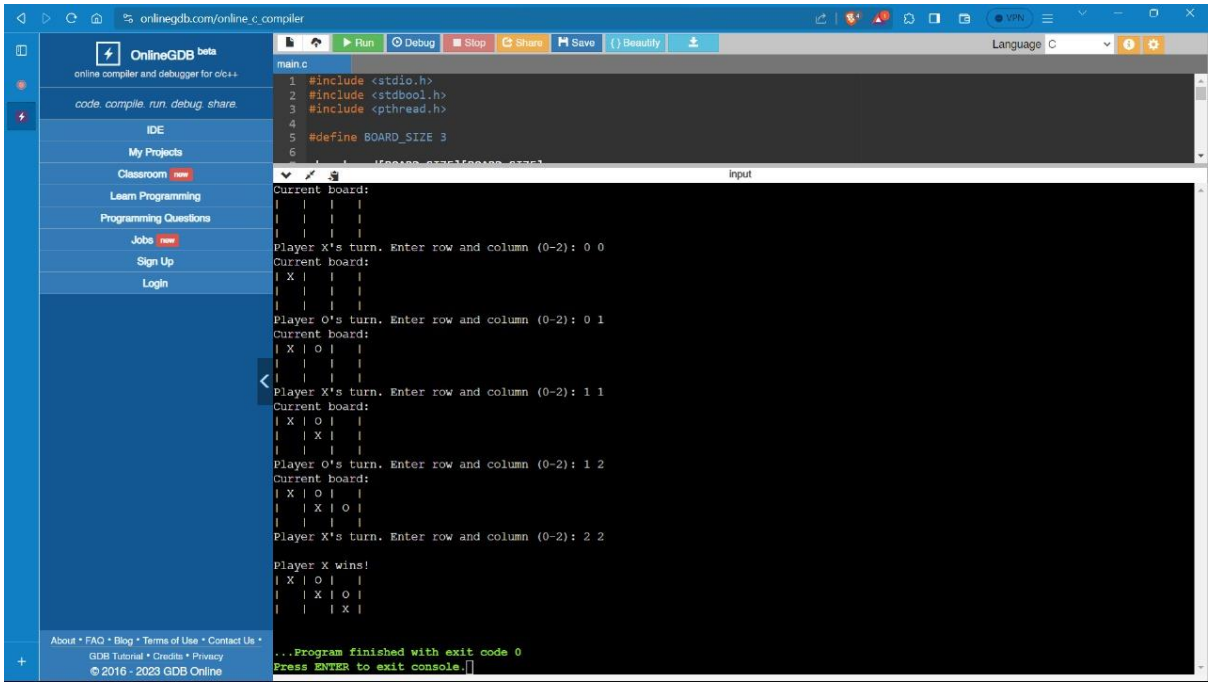
```
bool make_move(int row, int col)
{
    if (row < 0 || row >= BOARD_SIZE || col < 0 || col >= BOARD_SIZE ||
board[row][col] != ' ')
    return false;
    board[row][col] = currentPlayer;
    moves++;
    return true;
}
bool check_winner()
{
    for (int i = 0; i < BOARD_SIZE; i++)
    {
        if (board[i][0] == currentPlayer && board[i][1] == currentPlayer &&
board[i][2] == currentPlayer)
        return true;
        if (board[0][i] == currentPlayer && board[1][i] == currentPlayer &&
board[2][i] == currentPlayer)
        return true;
    }
    if (board[0][0] == currentPlayer && board[1][1] == currentPlayer &&
board[2][2] == currentPlayer)
    return true;
    if (board[0][2] == currentPlayer && board[1][1] == currentPlayer &&
board[2][0] == currentPlayer)
    return true;
    return false;
}
void* player_thread(void* player_name)
```

```
{
    while (moves < BOARD_SIZE * BOARD_SIZE)
    {
        pthread_mutex_lock(&mutex);
        if (game_over)
        {
            pthread_mutex_unlock(&mutex);
            break;
        }
        printf("Current board:\n");
        display_board();
        printf("Player %c's turn. Enter row and column (0-2): ", currentPlayer);
        int row, col;
        scanf("%d %d", &row, &col);
        if (make_move(row, col))
        {
            if (check_winner())
            {
                printf("\nPlayer %c wins!\n", currentPlayer);
                display_board();
                game_over = true;
            }
            else if (moves == BOARD_SIZE * BOARD_SIZE)
            {
                printf("\nIt's a draw!\n");
                display_board();
                game_over = true;
```

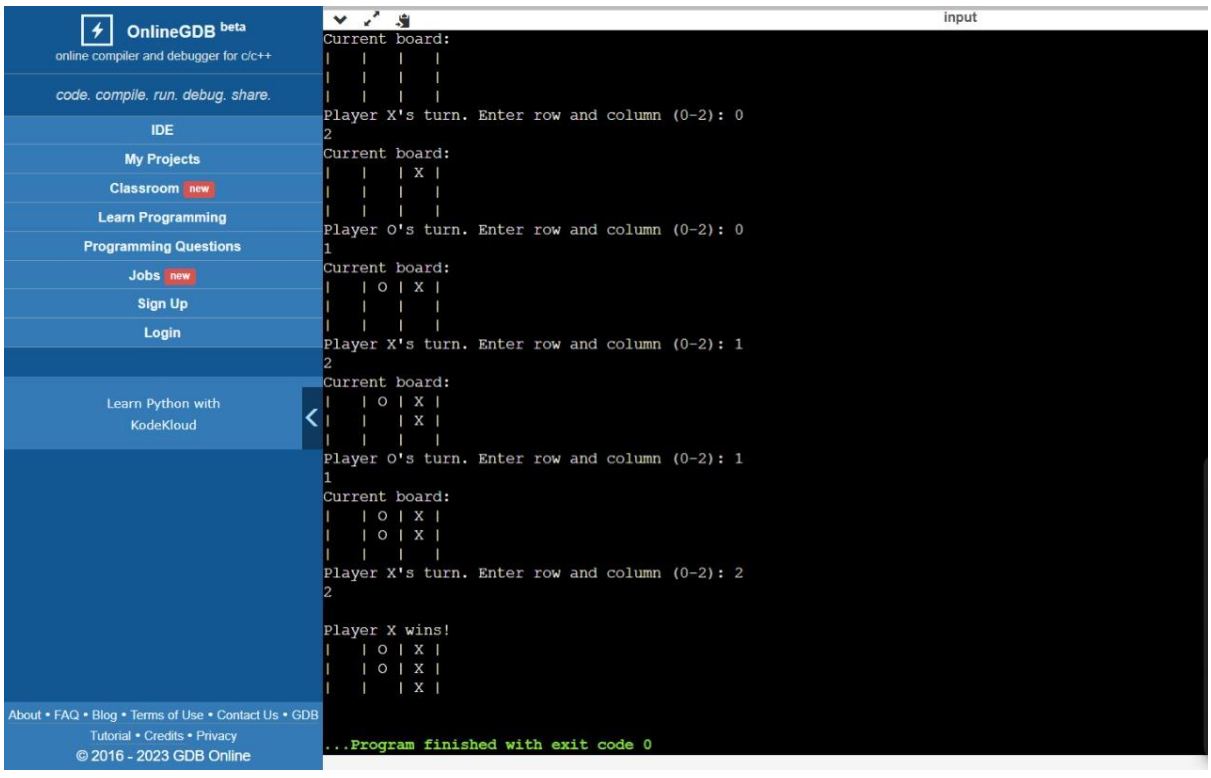
```
            }
            currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
        }
        else
        printf("Invalid move. Try again.\n");
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
int main()
{
    initialize_board();
    pthread_t player_X_thread, player_O_thread;
    char player_X_name = 'X';
    char player_O_name = 'O';
    pthread_create(&player_X_thread, NULL, player_thread, &player_X_name);
    pthread_create(&player_O_thread, NULL, player_thread, &player_O_name);
    pthread_join(player_X_thread, NULL);
    pthread_join(player_O_thread, NULL);
    return 0;
}
```

RESULTS

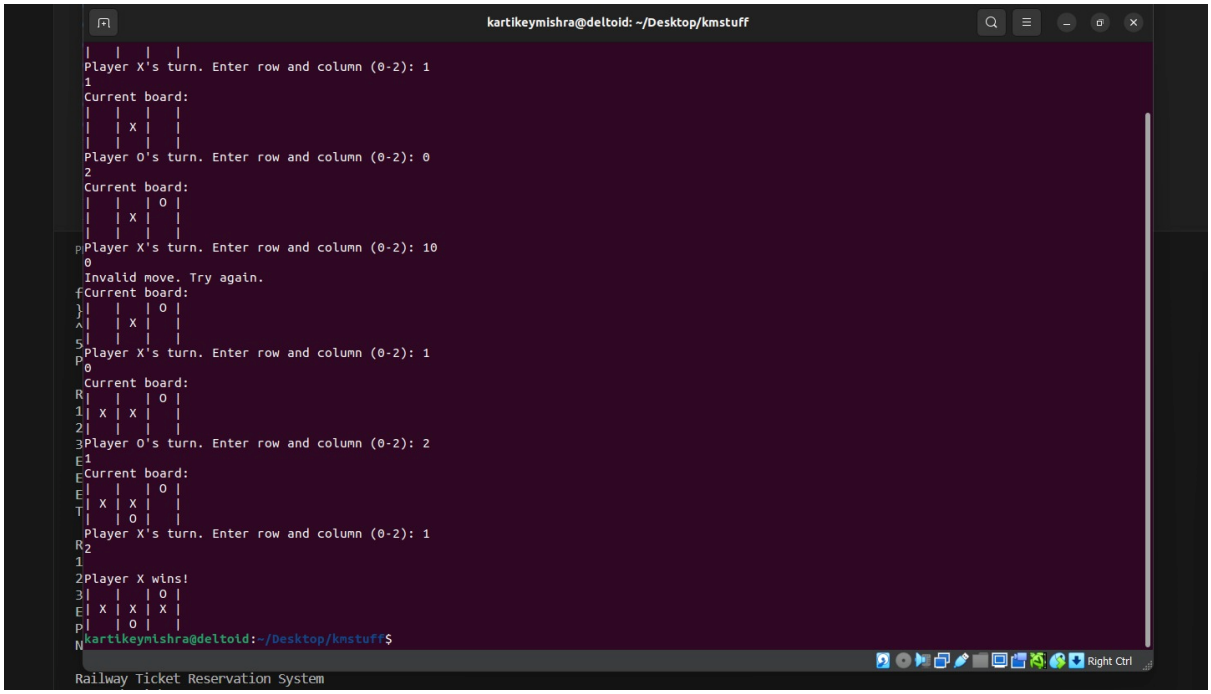
Case 1



Case 2



Case 3


kartikeymishra@deltoid: ~/Desktop/kmstuff
| | | |
Player X's turn. Enter row and column (0-2): 1
1
Current board:
	X	
Player O's turn. Enter row and column (0-2): 0
2
Current board:
		O
	X	
Player X's turn. Enter row and column (0-2): 10
0
Invalid move. Try again.
Current board:
		O
	X	
Player X's turn. Enter row and column (0-2): 1
0
Current board:
		O
X	X	
Player O's turn. Enter row and column (0-2): 2
1
Current board:
		O
X	X	
	O	
Player X's turn. Enter row and column (0-2): 1
2
Player X wins!
		O
X	X	X
	O	
kartikeymishra@deltoid:~/Desktop/kmstuff$
Right Ctrl
Railway Ticket Reservation System

CONCLUSION

In conclusion, incorporating mutual exclusion mechanisms into the development of a game like Tic-Tac-Toe can significantly enhance the overall gaming experience.

By implementing techniques such as mutex locks or semaphores, we can ensure that multiple players cannot simultaneously modify the game state, preventing potential conflicts and ensuring fair gameplay. This not only adds a layer of realism and fairness to the game but also helps maintain its integrity and reliability.

Moreover, it opens opportunities for creating multiplayer versions of the game, allowing players to engage with others in real-time, further enriching the gaming experience.

In essence, mutual exclusion is a crucial aspect of game development, particularly in multiplayer scenarios, and it plays a pivotal role in ensuring a smooth and enjoyable gaming experience for all participants in games like Tic-Tac-Toe.

REFERENCES

1. **GeeksForGeeks**
   https://www.geeksforgeeks.org/implementation-of-tic-tac-toe-game/

2. **Institution of Engineering and Technology-Online Library**
   https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/ccs.2020.0018

3. **Evolution of No-loss Strategies for the Game of Tic-Tac-Toe**
   By Anurag Bhatt, Pratul Varshney and Kalyanmoy Deb
   Indian Institute of Technology, Kanpur
   KanGAL Report Number 2007002

4. **Dartmouth Undergraduate Journal of Science**
   "Analyzing Mutual Exclusion with a Game"
   https://sites.dartmouth.edu/dujs/2015/02/17/analyzing-mutual-exclusion-with-a-game-siam-talk-prasad-jayanti/
   By Jayanti Prasad

DEVELOPMENT ENVIRONMENTS

- VirtualBox
- Ubuntu OS
- OnlineGDB Compiler for C
    - https://www.onlinegdb.com/online_c_compiler