****



**COURSE PROJECT REPORT**

**Fall 2024**

**TIC TAC TOE**

**SUBMITTED BY:**

**SANI-E-ZEHRA ABBAS F24CSC030**

**RIDA SALEEM F24CSC007**

**MAHEEN HABIB F24CSC024**

**FACULTY NAME**

**SIR BILAL AHMED**

**A project report submitted in partial fulfillment of the Requirements for the award of the degree of Bachelor of Computer Science**

**COMPUTER NETWORKS**

**DEPARTMENT OF COMPUTER SCIENCE**

**FACULTY OF INFORMATION TECHNOLOGY**

**SALIM HABIB UNIVERSITY, KARACHI**

**26/05/2025**

DECLARATION

We hereby declare that this project report is based on our original work except for citations and quotations which have been duly acknowledged. We also declare that it has not been previously and concurrently submitted for any other degree or award at Salim Habib University or other institutions.

|  |  |  |
| --- | --- | --- |
| Name: **Sani-e-Zehra Abbas** | Name: **Rida Saleem** | Name: **Maheen Habib** |
| Reg No.: F24CSC030 | Reg No.: F24CSC007 | Reg No.: F24CSC024 |
| Date: 26/05/2025 | Date: 26/05/2025 | Date: 26/05/2025 |
| Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

APPROVAL FOR SUBMISSION

We certify that this project report entitled **“TIC TAC TOE”** was prepared by **Sani-e-Zehra ,Maheen Habib & Rida Saleem** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Computer **Science** at Salim Habib University.

Approved by,

|  |  |
| --- | --- |
| Supervisor :  Date : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Signature : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Co-Supervisor :  Date : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Signature : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

The copyright of this report belongs to Salim Habib University according to the Intellectual Property Policy of Salim Habib University amended on July 2nd 2021. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2024 Salim Habib University. All rights reserved.

ACKNOWLEDGEMENTS

We would like to thank everyone who had contributed to the successful completion of this project. We would like to express my gratitude to my research supervisor, Dr/Mr/Miss/Madam XXXXX for his/her invaluable advice, guidance and his/her enormous patience throughout the development of the research.

In addition, We would also like to express my gratitude to our loving parent and friends who had helped and given me encouragement.

Yours Sincerely,

Sani-e-Zehra

Maheen Habib   
Rida Saleem

ABSTRACT

This project presents the development of a console-based Tic Tac Toe game using Java. The game features two human players who take turns marking spaces on a 3x3 grid. It follows standard Tic Tac Toe rules and incorporates object-oriented programming principles for modular design.

The application uses an abstract **Player** class and a concrete **HumanPlayer** class to manage player behavior. Game logic is controlled via a central **Tictactoe** class that maintains the board state, handles user input, checks for win conditions, and resets the board for replay.

To enhance functionality, the game stores moves using an **ArrayList** for history tracking, a **Queue** for managing turn order, and a **Stack** to support undo features. Error handling ensures that users cannot choose invalid or occupied cells.

The project was implemented in Eclipse IDE and was tested for multiple complete game cycles. Its design reflects real-world use of Java's core libraries and data structures in game development. This project not only reinforces OOP skills but also introduces practical usage of stack and queue concepts.

**Keywords**

 Java

 Tic Tac Toe

 Object-Oriented Programming

 Stack

 Queue

 Game Logic

 Console Application

 Exception Handling

 Inheritance

 ArrayList

TABLE OF CONTENTS

[DECLARATION ii](#_Toc131678506)

[APPROVAL FOR SUBMISSION iii](#_Toc131678507)

[ACKNOWLEDGEMENTS v](#_Toc131678508)

[ABSTRACT vi](#_Toc131678509)

[TABLE OF CONTENTS vii](#_Toc131678510)

[LIST OF FIGURES x](#_Toc131678512)

[LIST OF SYMBOLS / ABBREVIATIONS xi](#_Toc131678513)

[LIST OF APPENDICES xii](#_Toc131678514)

**CHAPTER**

[1. INTRODUCTION 1](#_Toc183439473)

[1.1 Background 1](#_Toc183439474)

[1.2 Problem Statements 1](#_Toc183439475)

[1.3 Aims and Objectives 2](#_Toc183439476)

[1.4 Scope of Project 2](#_Toc183439477)

[2. LITERATURE REVIEW 3](#_Toc183439478)

[2.1 Introduction to game development and oop 3](#_Toc183439479)

[2.2 Java data structures and their application 3](#_Toc183439480)

[2.3 Related work and Implementation 4](#_Toc183439481)

[2.3.1 Console Based Implementation 4](#_Toc183439482)

[2.3.2 Use of OOP 4](#_Toc183439483)

[2.4 Enhancment through Data Structures 4](#_Toc183439484)

[3. PROJECT METHODOLOGY 6](#_Toc183439485)

[3.1 Requirement Gathering and planning 6](#_Toc183439486)

[3.2 System Design and Development 6](#_Toc183439487)

[3.3 Testing and Iterative improvements 7](#_Toc183439488)

[4. SOFTWARE REQUIREMENT SPECIFICATIONS 8](#_Toc183439490)

[4.1 Functional Requirements 8](#_Toc183439491)

[4.2 Non functional Requirements 8](#_Toc183439492)

[4.3 System Design Requirements 9](#_Toc183439493)

[5. SOFTWARE DESIGN SPECIFICATION 10](#_Toc183439495)

[5.1 System Architecture 10](#_Toc183439496)

[5.2 Data Structures and flow Control 10](#_Toc183439497)

[5.3 User Interface and Interaction 11](#_Toc183439498)

[6. DISCUSSION AND CONCLUSION 12](#_Toc183439500)

[6.1 Conclusion 12](#_Toc183439501)

[7. REFERENCES 13](#_Toc183439502)

[APPENDICES 14](#_Toc183439503)

LIST OF FIGURES

**FIGURE TITLE PAGE**

[Figure 2.1: Data Sturctures 5](#_Toc131678118)

[Figure 3.1: Testing Cycle 7](#_Toc131678119)

[Figure 4.1: System Design Requirements 9](#_Toc131678120)

[Figure 5.1: Expected UI 11](#_Toc131678121)

LIST OF SYMBOLS / ABBREVIATIONS

|  |  |
| --- | --- |
| OOP | Object-Oriented Programming |

|  |  |
| --- | --- |
| JDK | Java Development Kit |

|  |  |
| --- | --- |
| JVM | Java Virtual Machine |

|  |  |
| --- | --- |
| IDE | Integrated Development Environment |

|  |  |
| --- | --- |
| CLI | Command Line Interface |

|  |  |
| --- | --- |
| Player | Abstract class for general player structure |

|  |  |
| --- | --- |
| HumanPlayer | Subclass representing a real user |

|  |  |
| --- | --- |
| Queue<Player> | Java queue to manage player turns |

|  |  |
| --- | --- |
| Stack<String> | Java stack for undo functionality |

|  |  |  |
| --- | --- | --- |
| ArrayList<String> |  | Stores history of all moves made |

|  |  |
| --- | --- |
| LinkedList<String> | Maintains queue of move strings |

|  |  |
| --- | --- |
| System.out.print | Prints output to console |

|  |  |
| --- | --- |
| Scanner | Reads user input from command line |

|  |  |
| --- | --- |
| SIZE = 3 | Constant for board dimensions (3x3) |

LIST OF APPENDICES

**APPENDIX TITLE PAGE**

[APPENDIX A: Project Schedule 14](#_Toc131678098)

# 

## INTRODUCTION

### Background

Tic Tac Toe is a simple two-player game that serves as a foundational exercise in programming and game logic. It is widely used in computer science education to introduce core concepts like conditional logic, loops, and arrays. This project extends the classic game by implementing it in Java using object-oriented programming principles. The goal is to demonstrate practical use of abstraction, inheritance, and data structures such as queues, stacks, and lists. The console-based interface provides a clean, text-driven experience for two human players.

### Problem Statements

The game of Tic Tac Toe operates on a finite 3x3 grid, which can be mathematically modeled as a matrix **B[3][3**], where each cell can take one of three values:

* **'X', 'O'**, or a digit **'0'–'8**' representing an empty cell.

Let **P =** **{X, O}** be the set of players, and let **M = {0,1,2,...,8}** represent possible move indices.  
A valid move m ∈ M must satisfy the condition:  
**B[i][j] ∉ P**, where **i = m / 3 and j = m % 3.**

The win condition W(p) for a player p ∈ P is defined as:  
**∃ i (B[i][0] = p ∧ B[i][1] = p ∧ B[i][2] = p)** ∨  
**∃ j (B[0][j] = p ∧ B[1][j] = p ∧ B[2][j] = p)** ∨  
**(B[0][0] = p ∧ B[1][1] = p ∧ B[2][2] = p)** ∨  
**(B[0][2] = p ∧ B[1][1] = p ∧ B[2][0] = p)**

The project aims to ensure all player inputs comply with these logical constraints, detect win/draw conditions efficiently, and simulate fair gameplay with alternating turns using a FIFO queue

### Aims and Objectives

The primary aim of this project is to develop a functional, console-based Tic Tac Toe game using Java that reinforces core object-oriented programming concepts and the use of basic data structures. The project serves as both a learning tool and a demonstration of applying theory to practical coding.

**The specific objectives are as follows:**

* To design and implement a two-player Tic Tac Toe game in Java.
* To apply object-oriented principles such as abstraction, inheritance, and polymorphism.
* To utilize core data structures like arrays, stacks, queues, and array lists within the game logic.
* To implement input validation and exception handling for robust gameplay.
* To maintain move history and provide an undo feature using a stack.
* To manage alternating turns using a queue structure.
* To test the application for different gameplay scenarios and edge cases.

### Scope of Project

This project implements a two-player Tic Tac Toe game in Java with a console-based interface. It showcases object-oriented programming and core data structures, supporting features like move validation, undo, and game reset. The scope is limited to terminal interaction, excluding GUI and AI, but the design allows for future enhancements.

## LITERATURE REVIEW

### Introduction to Game Development and Object-Oriented Programming

Game development in programming serves as a hands-on method to understand core computational concepts and coding practices. It combines logic, user interaction, and structured flow, making it an effective learning tool for beginners. Tic Tac Toe, a classical turn-based game, is widely adopted in educational settings to introduce students to decision-making logic, control structures, and data representation. Using object-oriented programming (OOP) principles to build such games enhances modularity, code reuse, and maintainability. Java, with its strong support for OOP, exception handling, and built-in data structures, provides an ideal environment for developing simple yet interactive console-based games.

### Java Data Structures and Their Application in Games

The Java programming language offers a comprehensive collection of data structures, including arrays, stacks, queues, and lists, which play a crucial role in game mechanics. Arrays are used to represent the game board as a fixed-size grid, allowing efficient access and update of cell values. Stacks are typically implemented to support undo functionality by storing the history of moves in reverse order. Queues enable alternating player turns, maintaining a fair sequence of gameplay. ArrayLists and LinkedLists allow dynamic storage of move histories and future enhancements like replay or analysis. These data structures not only simplify game logic but also highlight real-world use cases in software

### Related Work and Implementation Approaches

This section discusses existing approaches, variations, and techniques used in similar Tic Tac Toe implementations, and how the current project aligns with or diverges from them.

#### 2.3.1 Console-Based Implementations

Many existing Tic Tac Toe projects emphasize simplicity through terminal interaction, targeting beginners. These implementations focus on basic gameplay without introducing complexity such as GUIs or AI. The current project aligns with this approach but extends it by incorporating structured OOP design, move history tracking, and undo functionality—features often missing in basic models.

#### 2.3.2 Use of Object-Oriented Principles

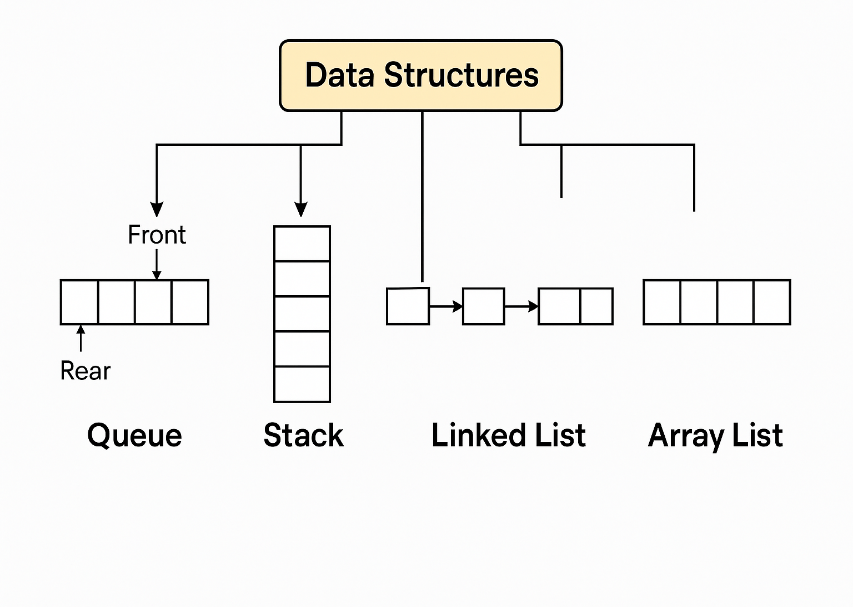
Previous implementations often bundle logic into a single class or use procedural programming. In contrast, the present project uses OOP features such as abstraction (**Player class**), inheritance (**HumanPlayer class**), and encapsulation (**Tictactoe class**). This modular design improves code readability, supports code maintenance, and opens pathways for future extensions like AI players or graphical interfaces.

### Enhancement Through Data Structures

While many basic projects use only arrays, this project demonstrates the use of multiple data structures:

* A **Queue** to manage player turns in a FIFO manner,
* A **Stack** to enable undoing moves,
* A **LinkedList** to queue moves, and
* An **ArrayList** to store and display move history.

Such enhancements add layers of functionality, mimicking real-world scenarios where tracking state and history is vital.



*Figure 2.1: Data Structures*

The current implementation limits itself to the terminal, its clean and modular design allows seamless future upgrades. Developers can plug in GUI components or AI logic without altering core gameplay, showcasing good software engineering practices.

## PROJECT METHODOLOGY

### Requirements Gathering and Planning

In the initial phase, the objective was to identify the core functionalities and components necessary to build a console-based Tic Tac Toe game in Java. The primary requirements included implementing a 3x3 game board, two-player turn-based gameplay, move validation, winner and draw detection, and maintaining a history of moves. Additionally, features like replay functionality and undo tracking were planned to enhance user interaction. Java collections like **ArrayList, Queue,** and **Stack** were selected for storing game states efficiently. This planning laid the foundation for an object-oriented and modular design.

### System Design and Development

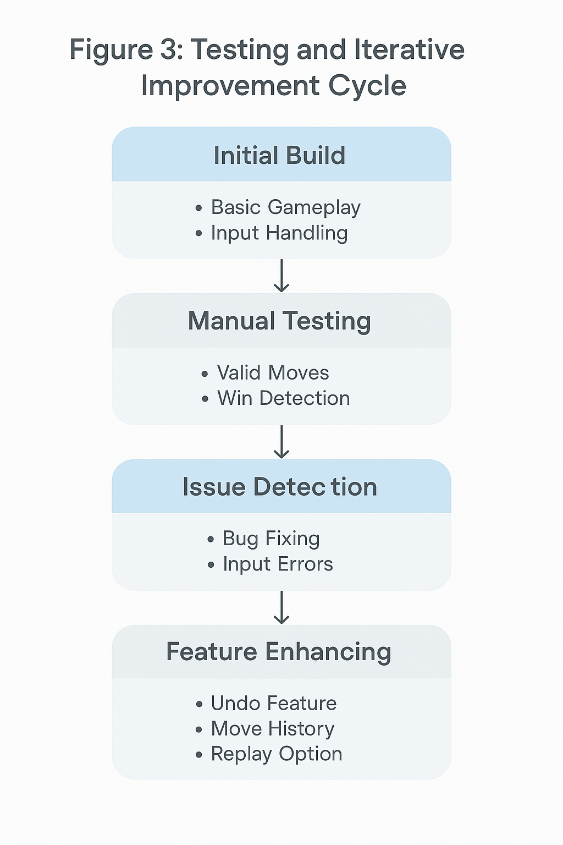
The second phase involved translating the requirements into a structured design using object-oriented programming principles. The application was divided into multiple classes for clarity and separation of concerns:

* The **Tictactoe** class handled the game board, game loop, and logic.
* The **Player** abstract class and its subclass **HumanPlayer** encapsulated user interaction and move handling.
* The **Main** class was responsible for initializing and starting the game.

A **Queue** was used to alternate turns between players, an **ArrayList** recorded the move history, a **LinkedList** stored the move queue, and a Stack tracked the undo history. Error handling was implemented to ensure users provided valid inputs, and the game board was consistently updated and printed after each move.

### Testing and Iterative Improvement

After the initial implementation, the game was tested with various input scenarios to ensure stability and correctness. Tests included win conditions for rows, columns, and diagonals, detection of draw situations, and prevention of invalid or repeated moves. The system was iteratively refined based on these test results. Additional features such as displaying the last move and offering a replay option were included to enhance usability. The final version met all the functional requirements and provided a smooth user experience in a console-based environment.



*Figure 3.1: Testing cycle*

## SOFTWARE REQUIREMENT SPECIFICATIONS

### Functional Requirements

This section outlines the core functionalities that the system must provide:

* **Two-Player Gameplay**: The system shall allow two human players to play the game in alternating turns.
* **Board Display**: The system shall display a 3x3 board showing the current game state after each move.
* **Input Handling**: The system shall accept user input for selecting an unoccupied cell using box numbers (0 to 8).
* **Move Validation**: The system shall reject invalid inputs such as non-numeric entries, out-of-range values, or already-occupied cells.
* **Win/Draw Detection**: The system shall automatically detect and declare the winner or a draw at the end of the game.
* **Move History**: The system shall maintain and display a history of all moves made during the game.
* **Undo Functionality**: The system shall allow tracking the last move for potential undo capabilities (though not explicitly triggered in this version).
* **Game Reset**: The system shall reset the board and game state when players opt to replay after one round ends.

### Non-Functional Requirements

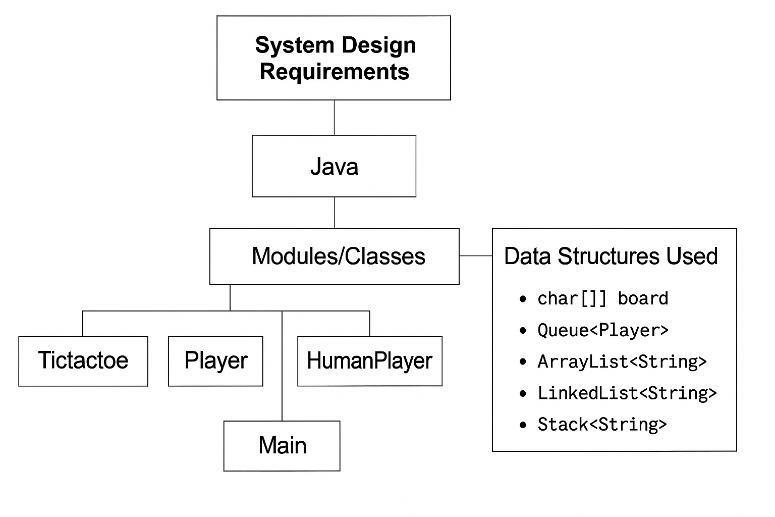
This section defines the quality attributes and constraints of the software:

* **Platform**: The game shall be a terminal-based Java application, compatible with standard Java runtime environments.
* **Usability**: The interface shall be simple and intuitive, displaying clear prompts and board visuals using console output.
* **Reliability**: The system shall handle unexpected input gracefully using exception handling and provide appropriate error messages.
* **Performance**: The game shall respond to user actions in real-time, without noticeable delays.
* **Extensibility**: The code shall follow modular object-oriented principles to support future enhancements, such as GUI or AI integration.
* **Maintainability**: Classes and methods shall be well-structured and commented to ease debugging and extension.

### System Design Requirements

This section describes the structural and data requirements that support the implementation:

* **Programming Language**: Java (JDK 8 or above).
* **Modules/Classes**:
  + **Tictactoe:** Manages the game loop, board state, and flow control.
  + **Player:** Abstract class representing a player with a symbol and move interface.
  + **HumanPlayer**: Subclass of Player implementing console-based move input.
  + **Main**: Initializes and launches the game.
* **Data Structures Used**:
  + **char[][] board** – 2D array for representing the 3x3 game board.
  + **Queue<Player>** – To alternate turns between players using FIFO.
  + **ArrayList<String>** – For storing move history in order.
  + **LinkedList<String>** – For queuing moves, useful for potential replays.
  + **Stack<String>** – For storing moves to support undo capability.
* **External Dependencies**: No third-party libraries are used; relies solely on Java standard library.
* **User Interface**: Text-based interface via the command line.



*Figure 4.1: System Design Requirements*

## SOFTWARE DESIGN SPECIFICATION

### System Architecture

The application follows a **modular, object-oriented architecture** using Java. It is organized into the following core components:

* **Main Class (Main)**: Serves as the entry point to initialize and start the game loop.
* **Game Controller (Tictactoe)**: Handles game state, board operations, player turns, and win/draw logic. Manages data structures for move history (ArrayList), undo tracking (Stack), and player turn management (Queue).
* **Abstract Player Class (Player)**: Defines the basic structure for any player (e.g., symbol and move method).
* **Human Player (HumanPlayer)**: Implements the makeMove logic to allow human input via the console.

This design enables flexibility and easy extension (e.g., adding AI players or GUI), while keeping concerns separated and manageable.

### Data Structures and Flow Control

To ensure efficient operation and easy tracking of gameplay, the following Java data structures are used:

* **char[][] board**: Represents the 3x3 grid.
* **Queue<Player>:** Alternates player turns using a FIFO queue.
* **ArrayList<String> moveHistory**: Stores a log of all player moves.
* **LinkedList<String> moveQueue**: Maintains sequential order of moves.
* **Stack<String> undoStack**: Tracks the most recent moves for potential undo operations.

**Game Loop Flow**:

1. Print board state.
2. Fetch current player from queue.
3. Request valid input via **makeMove().**
4. Place symbol, update logs.
5. Check for win/draw.
6. Continue or prompt for replay.

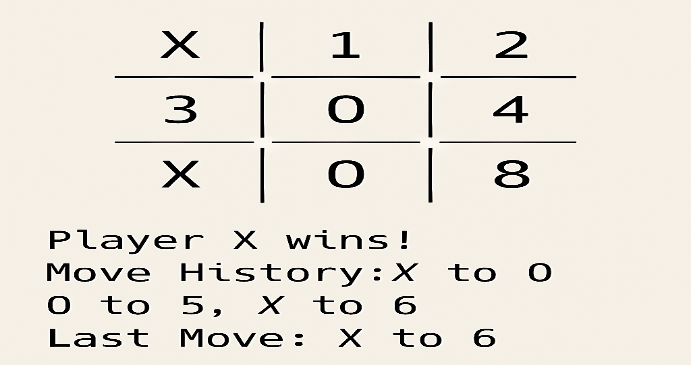
This control flow ensures clear execution and handles all edge cases such as invalid inputs, repeated moves, and board completion.

### User Interface and Interaction Design

The game uses a **console-based UI** for simplicity and clarity. All interactions are text-based:

* Users are prompted with clear instructions to choose a cell (0–8).
* After each move, the updated board is printed with appropriate formatting.
* If a move is invalid (occupied or out of bounds), informative error messages guide the user to retry.
* At game end, the winner is announced or a draw is declared.
* The user is then asked if they want to play again, supporting multiple rounds in one session.
* Move history and the last move are shown to give feedback and transparency.

This interaction design ensures a user-friendly experience while demonstrating core software engineering concepts like input validation, state feedback, and replayability.



*Figure 5.1: Expected UI*

## DISCUSSION AND CONCLUSION

### CONCLUSION

The development of the Tic Tac Toe game using Java provided a comprehensive understanding of object-oriented programming concepts, data structure implementation, and interactive console-based design. By structuring the application into modular classes such as **Tictactoe, Player**, and **HumanPlayer**, the project demonstrated effective use of abstraction, encapsulation, and inheritance.

Additionally, integrating collections like **Queue, ArrayList**, and **Stack** allowed efficient tracking of player turns, move history, and undo functionality. The program not only fulfilled its core objective—delivering a playable 2-player game—but also emphasized user-friendly interaction through input validation, clear instructions, and replay support.

Overall, this project successfully showcased practical problem-solving, clean code design, and a solid foundation for future enhancements such as AI players or a graphical interface. It stands as a strong example of applying theoretical knowledge to build a functional and engaging software application.

## REFERENCES

* YouTubeTutorial:  
  *"JavaTicTaCToeGame"*  
  Available at: <https://youtu.be/WRr2h_R0AFM?si=RZfJ5Oo46MqU-8nG>

## APPENDICES

**APPENDIX A: Project Schedule**

1. Gantt Chart

