

1.INTRODUCTION

Tic Tac Toe, also known as Noughts and Crosses, is a classic two-player strategy game played on a 3×3 grid. The game is simple in structure but provides a foundation for exploring decision-making and game logic. Each player takes turns placing their symbol (X or O) on an empty square of the grid. The player who first aligns three of their symbols in a horizontal, vertical, or diagonal row wins. If all nine squares are filled without a winner, the game ends in a draw.

Implementing Tic Tac Toe in **Prolog**, a logic programming language, presents a unique and educational experience. Unlike imperative languages where the programmer specifies *how* a task is performed, Prolog focuses on *what* the solution should look like. This makes it particularly suitable for rule-based problems and symbolic reasoning—both of which are essential in game logic.

In the case of Tic-Tac-Toe, the main tasks are:

1. **Board Representation:** The board is usually represented as a list of nine elements (since the Tic-Tac-Toe board is 3x3). Each element in the list can either be x, o, or empty, reflecting the state of each square on the board.
2. **Making Moves:** A move consists of placing either an x or an o in one of the empty positions on the board. This requires checking if a position is empty and then updating the board accordingly.
3. **Checking for a Winner:** To determine if someone has won, Prolog can check if any player has aligned three marks in a row—either horizontally, vertically, or diagonally.
4. **Game Logic:** The game can be controlled by alternating turns between the two players (X and O), and checking after each move whether the game has ended (either through a win or a draw).

2. LITERATURE SURVEY

The development of board games such as Tic Tac Toe in logic programming languages like Prolog has been the subject of interest in both artificial intelligence (AI) research and educational contexts. This literature survey reviews key concepts, past work, and approaches used in implementing decision-based games using Prolog.

1. Game Theory and Decision-Making

Tic Tac Toe is often studied as a foundational model for understanding basic concepts in game theory, such as zero-sum games, deterministic outcomes, and perfect information. Early works in AI, such as those by **Alan Turing** and **Claude Shannon**, examined simple games as testbeds for simulating decision-making processes in machines.

2. Prolog in AI Education

Prolog is commonly used in AI courses to teach symbolic reasoning, pattern matching, and logical inference. According to Bratko (1990) in *Prolog Programming for Artificial Intelligence*, games like Tic Tac Toe offer an ideal entry point for demonstrating rule-based problem solving. Prolog's ability to backtrack and match patterns naturally supports the development of game rules and recursive move generation.

3. Game Representation in Prolog

Multiple studies and tutorials have focused on representing game boards as lists in Prolog. This flat structure simplifies access to positions and allows for concise rules for checking winning combinations. Researchers have shown that this method supports efficient checking of board states through predicates like `nth0/3` and unification techniques.

3. ANALYSIS AND DESIGN

The purpose of this section is to outline the structural and logical framework of the Tic Tac Toe game implementation using Prolog. This includes identifying the functional requirements, understanding the game's logic, and designing the system using appropriate data representations and Prolog predicates.

3.1. Problem Analysis

Tic Tac Toe is a turn-based, two-player game played on a 3×3 grid. Each player takes turns marking an empty cell with their respective symbol (× or ○). The game ends when one player places three of their marks in a straight line (horizontally, vertically, or diagonally), or when all positions are filled without any player winning (a draw).

In Prolog, the problem translates to managing board states, validating moves, detecting wins or draws, and alternating turns between players.

3.2. Functional Requirements

- Represent the game board.
- Allow players to make legal moves.
- Check for winning conditions after each move.
- Check for a draw when the board is full.
- Alternate turns between × and ○.
- End the game when a win or draw is detected.
- Optionally, allow the computer to play intelligently (AI).

3.3. System Design

Board Representation

The game board is represented as a flat list of 9 elements:

Prolog syntax:

[Pos0, Pos1, Pos2,
Pos3, Pos4, Pos5,
Pos6, Pos7, Pos8]

Each element can be:

- x – Player X's move
- o – Player O's move
- _ – An empty space

Core Predicates

Predicate	Purpose
display_board/1	Displays the current board in a human-readable format
make_move/4	Places a move on the board at a specific index
valid_move/2	Checks if a move is valid (i.e., the cell is empty)
win/2	Checks if a given player has won
draw/1	Checks if the board is full with no winner
switch_player/2	Alternates between x and o
play/2	Manages the turn-based logic of the game

Winning Conditions

A player wins if they occupy any of the following index combinations:

- Rows: [0, 1, 2], [3, 4, 5], [6, 7, 8]
- Columns: [0, 3, 6], [1, 4, 7], [2, 5, 8]
- Diagonals: [0, 4, 8], [2, 4, 6]

A predicate `win(Board, Player)` is used to check these conditions.

Move Logic

The `make_move/4` predicate attempts to place a player's symbol on the board. If the position is already occupied, the move is rejected.

Example logic:

`make_move(Board, Index, Player, NewBoard) :-`

`nth0(Index, Board, _, Rest),`

`nth0(Index, NewBoard, Player, Rest).`

Game Flow

1. Start with an empty board.
2. Prompt player \times to move.
3. Validate the move.
4. Update the board.
5. Check for win or draw.
6. Switch player and repeat.

3.4. USE-CASE Diagram

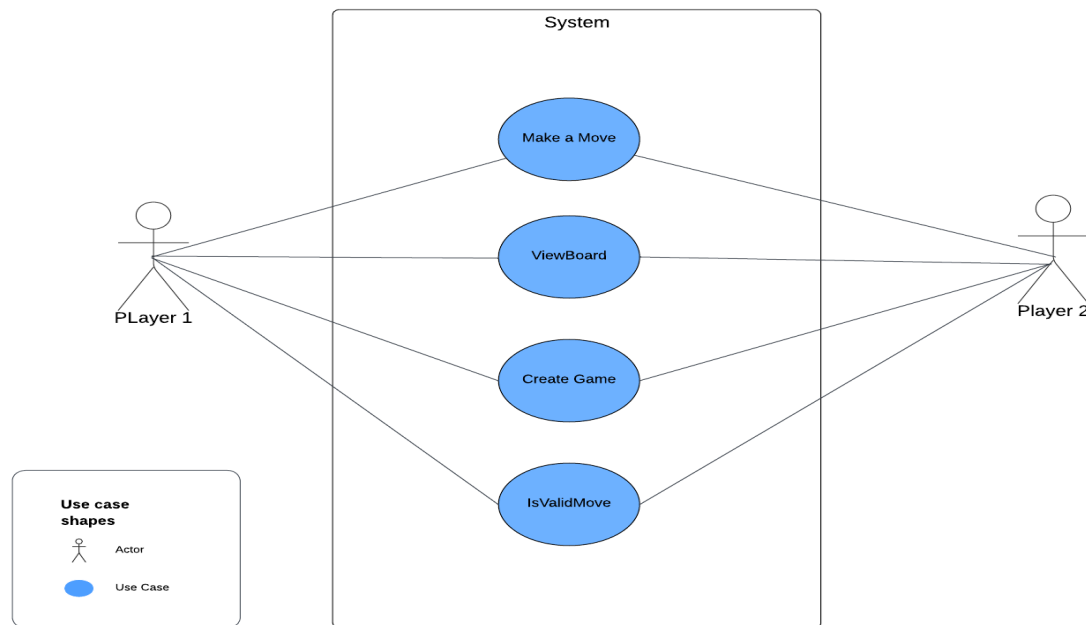


Fig1: Class Diagram for Tic Tac Toe Game

Actors:

- Player1 and Player2:- These are the two primary external entities (users) that interact with the Tic Tac Toe system. Both players share similar functionalities within the game and are treated equally in terms of permissions and interactions.

Use Cases:

Each oval represents a distinct use case — a functionality or service the system provides to its users.

1. Create Game

- This is the initial action where a new game is set up.
- The system initializes the game board and sets the starting player.
- Both players are involved in this use case.

2. Make a Move

- Once the game is started, players take turns making moves.
- This use case represents the action of placing an “X” or “O” on the board.

- The system checks if it's a valid turn and updates the board accordingly.

3. View Board

- At any point, both players can view the current state of the game board.
- This use case ensures that players have a clear understanding of the game status after each move.

4. IsValidMove

- Before a move is accepted, the system checks if the intended cell is empty and the move adheres to game rules.
- This use case helps prevent overwriting previous moves and keeps the game state valid.

3.5. CLASS Diagram

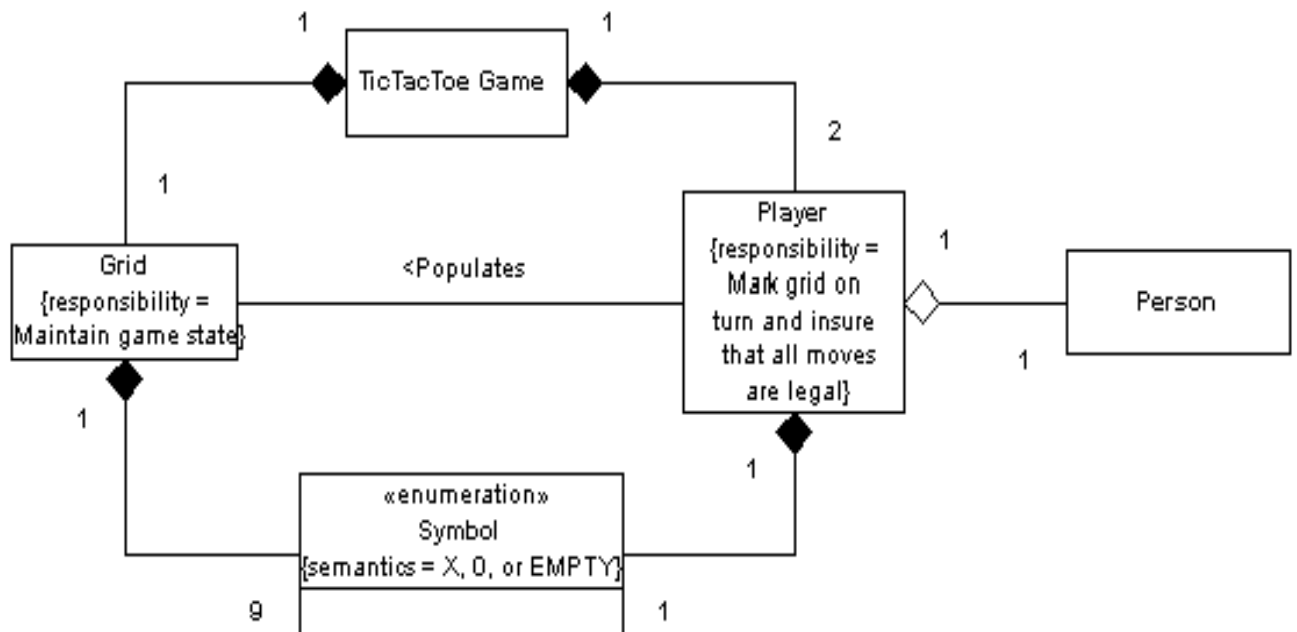


Fig2: Class Diagram for Tic Tac Toe Game

TicTacToe Game:

- Central controller of the game.

- Connected to 2 Players and 1 Grid.

Player:

- Makes moves and ensures legality.
- Associated with a Person (for general player details).

Grid:

- Maintains the game state (3x3 board).
- Contains 9 Symbols.

Symbol (Enum):

- Values: X, O, EMPTY.

Relationships:

- Composition (◆): Strong ownership (e.g., Game \rightarrow Grid).
- Aggregation (◇): Loose link (e.g., Player \rightarrow Person).
- Multiplicities define exact counts (e.g., 9 Symbols in Grid).

3.5. SEQUENCE Diagram

Actors/Lifelines:

- Player 1
- Game Controller
- Grid
- Player 2

Flow:

1. Player 1 → Game: makeMove(position)
2. Game → Grid: isValidMove(position)
3. Grid → Game: true/false
4. Game → Grid: updateCell(position, X)
5. Game → Game: checkWinCondition()
6. Game → Player 2: yourTurn()
7. Repeat for Player 2
8. Game → All: announceResult(win/draw)

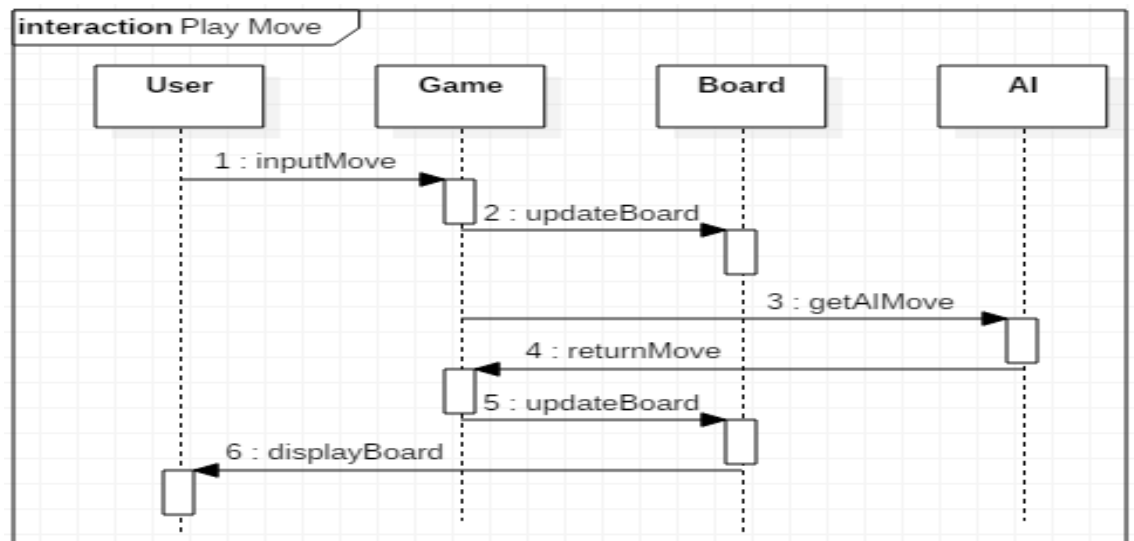


Fig3: Class Diagram for Tic Tac Toe Game

4. IMPLEMENTATION

% Define the winning conditions

win(Player, Board) :-

Board = [Player, Player, Player, _, _, _, _, _, _];

Board = [_, _, _, Player, Player, Player, _, _, _];

Board = [_, _, _, _, _, _, Player, Player, Player];

Board = [Player, _, _, Player, _, _, Player, _, _];

Board = [_, Player, _, _, Player, _, _, Player, _];

Board = [_, _, Player, _, _, Player, _, _, Player];

Board = [Player, _, _, _, Player, _, _, _, Player];

Board = [_, _, Player, _, Player, _, Player, _, _].

% Check for a draw (no empty spaces left)

draw(Board) :- \+ member('_', Board).

% Display the board

display_board([A, B, C, D, E, F, G, H, I]) :-

format("~w | ~w | ~w\n", [A, B, C]),

format("--+---+--\n"),

format("~w | ~w | ~w\n", [D, E, F]),

```

format("---+---+---\n"),

format("~w | ~w | ~w\n", [G, H, I]), nl.

% Make a move

move(Player, Board, Pos, NewBoard) :-

    nth0(Pos, Board, '-', TempBoard), % Ensure position is empty

    replace(Board, Pos, Player, NewBoard).

% Replace an element in a list

replace([_|T], 0, X, [X|T]).

replace([H|T], I, X, [H|R]) :- I > 0, I1 is I - 1, replace(T, I1, X, R).

% AI makes a move (chooses the first available empty space)

ai_move(Board, NewBoard) :-

    nth0(Pos, Board, '-', _), % Find first empty space

    move(o, Board, Pos, NewBoard), !.

% Game loop

game(Board, Player) :-

    display_board(Board),

```

```
(win(x, Board) -> writeln('Player X wins!');  
  
win(o, Board) -> writeln('Player O wins!');  
  
draw(Board) -> writeln('It\'s a draw!');  
  
(Player = x -> write('Enter position (0-8): '), read(Pos),  
    move(x, Board, Pos, NewBoard), game(NewBoard, o);  
    ai_move(Board, NewBoard), game(NewBoard, x))).  
  
% Start the game  
  
start :-  
    writeln('Welcome to Tic-Tac-Toe!'),  
    Board = ['-', '-', '-', '-', '-', '-', '-', '-', '-'],  
    game(Board, x).
```

5. TESTING AND DEBUGGING


Enter position (0-8):


o		o		x
-+-----+--				
-		x		x
-+-----+--				
-		-		-

Enter position (0-8):

o		o		x
-+-----+--				
o		x		x
-+-----+--				
-		-		x

Player X wins!

 start.

 Singleton variables: [TempBoard]

Welcome to Tic-Tac-Toe!

-		-		-
-+-----+--				
-		-		-
-+-----+--				
-		-		-

Enter position (0-8):

Please enter a Prolog term

?- start.

6. ADVANTAGES

1. Natural Fit for AI & Logic Problems

- Prolog is built for declarative programming, which is ideal for representing rules, facts, and game logic.
- It allows a clear and human-readable way to express game strategies and conditions.

2. Built-in Backtracking

- Prolog can automatically explore different possible moves and backtrack when necessary.
- This is powerful for implementing AI strategies like Minimax and decision trees.

3. Concise and Elegant Code

- Complex logic (like checking winning combinations or validating moves) can be expressed in very few lines.
- No need for loops or traditional conditionals – rules handle everything.

4. Perfect for Learning AI Concepts

- Helps you understand foundational AI topics such as:
 - Game trees
 - Rule-based decision making
 - Recursive reasoning

5. Pattern Matching is Super Easy

- Matching board positions, checking for wins, or finding valid moves is very simple and intuitive using Prolog's pattern matching capabilities.

7. LIMITATIONS

1. Not Beginner-Friendly for All

- Prolog has a different programming paradigm (declarative vs. procedural), which can feel strange or confusing for people used to C, Python, or Java.
- Concepts like backtracking and unification are tricky at first.

2. Performance Constraints

- While great for logic problems, Prolog isn't optimized for speed in larger or more complex games.
- Evaluating many possibilities in bigger games can slow things down.

3. Limited GUI Support

- Prolog is not designed for graphical user interfaces (GUI).
- Making a Prolog-based Tic Tac Toe with a nice visual layout requires integrating with other tools or languages (like Java or Python).

4. Debugging Can Be Hard

- It's often difficult to trace errors because Prolog uses automatic backtracking and rule-based flow.
- The code may silently fail or take unexpected paths if logic isn't tight.

5. Not Widely Used in Industry

- Prolog is mostly used in academia, AI research, and specific fields like natural language processing or expert systems.
 - You won't find it in mainstream software development or game engines.

8. CONCLUSION

The implementation of Tic Tac Toe in Prolog demonstrates the power and elegance of logic programming for solving rule-based problems. Unlike imperative languages, Prolog allows for expressing game rules, winning conditions, and turn-based logic in a highly declarative and human-readable manner. This project not only simulates a complete Tic Tac Toe game but also showcases how Prolog handles state management, pattern matching, and decision-making through logical inference.

By representing the game board as a list and using recursive predicates to control the game flow, the project effectively models the structure and behavior of a turn-based game. Key features such as move validation, win and draw detection, and turn switching were implemented in a concise and modular way.

This project also lays a strong foundation for future enhancements, such as incorporating an AI opponent using the minimax algorithm, adding a user interface, or extending the game to larger board sizes. Overall, the project serves as both a practical demonstration of Prolog's capabilities and a valuable learning experience in logic-based game development.

REFERENCES

1. **Book:** Prolog Programming for Artificial Intelligence by Ivan Bratko
<https://www.pearson.com/en-us/subject-catalog/p/prolog-programming-for-artificial-intelligence/P2000000003878>
2. **Book:** *Programming in Prolog* by W.F. Clocksin & C.S. Mellish
<https://link.springer.com/book/10.1007/978-3-662-12316-0>
3. **Online Tutorial:** Learn Prolog Now!
<http://www.learnprolognow.org>
4. **Documentation:** SWI-Prolog Official Documentation:-<https://www.swi-prolog.org/pldoc/>
5. **Diagram Tool:** Lucidchart – Online UML & Flowchart Tool
 <https://www.lucidchart.com>