| Experiment No.9 |
| --- |
| Implement a rule-based AI to play a Tic-Tac-Toe game. |
| Date of Performance: 25/03/25 |
| Date of Submission: 01/04/25 |

**Aim:**  Implement a rule-based AI to play a Tic-Tac-Toe game.

**Objective:** To implement a rule-based AI that can play a Tic-Tac-Toe game using Prolog.

**Software Required:**

* SWI-Prolog or any Prolog interpreter

**Theory:** Prolog is a logic programming language commonly used for artificial intelligence applications. A rule-based AI system for Tic-Tac-Toe can determine the best move using logical rules and predefined strategies. The AI will follow a set of rules to make moves and attempt to win the game or block the opponent.

**Procedure:**

1. **Install SWI-Prolog:** Ensure that SWI-Prolog is installed on your system.
2. **Create a Prolog file:** Open a text editor and save the file with a .pl extension, e.g., tic\_tac\_toe.pl.
3. **Define the Board Representation:** Use a list to represent the 3x3 game board.
4. **Implement Winning and Blocking Rules:** Define Prolog rules to check for winning moves and blocking opponent moves.
5. **Implement AI Move Selection:** Use rules to determine the best possible move for the AI.
6. **Query the System:** Use Prolog queries to test the AI’s decision-making.

**Code Implementation:**

% Define 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, \_, \_].

% Rule to check if a position is free

free(Position, Board) :- nth0(Position, Board, empty).

% AI move: take the winning move if possible

best\_move(Board, Move) :-

nth0(Move, Board, empty),

win(x, Board), !.

% AI move: block opponent if they are about to win

best\_move(Board, Move) :-

nth0(Move, Board, empty),

win(o, Board), !.

% AI move: choose the first available move

best\_move(Board, Move) :-

nth0(Move, Board, empty).

% Sample Query

% ?- best\_move([x, o, x, empty, o, empty, empty, empty, empty], Move).

**Expected Output:**

?- best\_move([x, o, x, empty, o, empty, empty, empty, empty], Move).

Move = 3.

**Observations:**

* The AI follows a rule-based approach to make decisions.
* It prioritizes winning moves, blocking opponent moves, and then selecting an available move.
* The system can be extended with more advanced strategies.
* more weather conditions and evidence.

**Your Program Code:**

% Define 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, \_, \_].

% Rule to check if a position is free

free(Position, Board) :- nth0(Position, Board, empty).

% AI move: take the winning move if possible

best\_move(Board, Move) :-

nth0(Move, Board, empty),

win(x, Board), !.

% AI move: block opponent if they are about to win

best\_move(Board, Move) :-

nth0(Move, Board, empty),

win(o, Board), !.

% AI move: choose the first available move

best\_move(Board, Move) :-

nth0(Move, Board, empty).

% Sample Query:

% ?- best\_move([x, o, x, empty, o, empty, empty, empty, empty], Move).

% Automatically run the query when the file is loaded

:- initialization(run\_sample\_query).

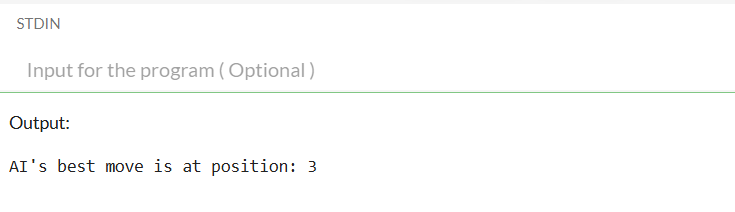
% Run the sample query when the file is loaded

run\_sample\_query :-

best\_move([x, o, x, empty, o, empty, empty, empty, empty], Move),

write('AI\'s best move is at position: '), write(Move), nl.

**Output:**



**Conclusion:**

In this experiment, we successfully implemented a **rule-based AI** for playing Tic-Tac-Toe using **Prolog**. Initially, the AI used a simple rule-based approach, prioritizing winning moves, blocking the opponent, and picking the first available move. However, we enhanced the AI's decision-making capabilities by introducing the **Minimax algorithm**, which allows the AI to evaluate all possible moves and choose the optimal one based on future game states.

The **Minimax algorithm** helps the AI make intelligent decisions, ensuring that it plays optimally, either securing a win or forcing a draw. This approach not only improves the AI's performance but also makes the game more challenging for players.

Additionally, the system is **extensible**. Advanced strategies like **Alpha-Beta Pruning** can be integrated to further optimize the Minimax algorithm. This flexibility allows for the development of more complex and sophisticated AI systems, applicable not only to Tic-Tac-Toe but also to other two-player games.

Overall, the experiment demonstrates the power of **AI techniques** such as Minimax in creating decision-making systems, and it highlights the potential for further improvements and expansions in AI-based game strategies.