**PRACTICAL-3**

**AIM:** Write a program to implement Tic-Tac-Toe game problem.

**Source Code:**

% Predicates that define the winning conditions:

win(Board, Player) :- rowwin(Board, Player).

win(Board, Player) :- colwin(Board, Player).

win(Board, Player) :- diagwin(Board, Player).

rowwin(Board, Player) :- Board = [Player,Player,Player,\_,\_,\_,\_,\_,\_].

rowwin(Board, Player) :- Board = [\_,\_,\_,Player,Player,Player,\_,\_,\_].

rowwin(Board, Player) :- Board = [\_,\_,\_,\_,\_,\_,Player,Player,Player].

colwin(Board, Player) :- Board = [Player,\_,\_,Player,\_,\_,Player,\_,\_].

colwin(Board, Player) :- Board = [\_,Player,\_,\_,Player,\_,\_,Player,\_].

colwin(Board, Player) :- Board = [\_,\_,Player,\_,\_,Player,\_,\_,Player].

diagwin(Board, Player) :- Board = [Player,\_,\_,\_,Player,\_,\_,\_,Player].

diagwin(Board, Player) :- Board = [\_,\_,Player,\_,Player,\_,Player,\_,\_].

% Helping predicate for alternating play in a "self" game:

other(x,o).

other(o,x).

game(Board, Player) :- win(Board, Player), !, write([player, Player, wins]).

game(Board, Player) :-

other(Player,Otherplayer),

move(Board,Player,Newboard),

!,

display(Newboard),

game(Newboard,Otherplayer).

move([b,B,C,D,E,F,G,H,I], Player, [Player,B,C,D,E,F,G,H,I]).

move([A,b,C,D,E,F,G,H,I], Player, [A,Player,C,D,E,F,G,H,I]).

move([A,B,b,D,E,F,G,H,I], Player, [A,B,Player,D,E,F,G,H,I]).

move([A,B,C,b,E,F,G,H,I], Player, [A,B,C,Player,E,F,G,H,I]).

move([A,B,C,D,b,F,G,H,I], Player, [A,B,C,D,Player,F,G,H,I]).

move([A,B,C,D,E,b,G,H,I], Player, [A,B,C,D,E,Player,G,H,I]).

move([A,B,C,D,E,F,b,H,I], Player, [A,B,C,D,E,F,Player,H,I]).

move([A,B,C,D,E,F,G,b,I], Player, [A,B,C,D,E,F,G,Player,I]).

move([A,B,C,D,E,F,G,H,b], Player, [A,B,C,D,E,F,G,H,Player]).

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

write([G,H,I]),nl,nl.

selfgame :- game([b,b,b,b,b,b,b,b,b],x).

% Predicates to support playing a game with the user:

x\_can\_win\_in\_one(Board) :- move(Board, x, Newboard), win(Newboard, x).

% The predicate orespond generates the computer's (playing o) reponse

% from the current Board.

orespond(Board,Newboard) :-

move(Board, o, Newboard),

win(Newboard, o),

!.

orespond(Board,Newboard) :-

move(Board, o, Newboard),

not(x\_can\_win\_in\_one(Newboard)).

orespond(Board,Newboard) :-

move(Board, o, Newboard).

orespond(Board,Newboard) :-

not(member(b,Board)),

!,

write('Cats game!'), nl,

Newboard = Board.

% The following translates from an integer description

% of x's move to a board transformation.

xmove([b,B,C,D,E,F,G,H,I], 1, [x,B,C,D,E,F,G,H,I]).

xmove([A,b,C,D,E,F,G,H,I], 2, [A,x,C,D,E,F,G,H,I]).

xmove([A,B,b,D,E,F,G,H,I], 3, [A,B,x,D,E,F,G,H,I]).

xmove([A,B,C,b,E,F,G,H,I], 4, [A,B,C,x,E,F,G,H,I]).

xmove([A,B,C,D,b,F,G,H,I], 5, [A,B,C,D,x,F,G,H,I]).

xmove([A,B,C,D,E,b,G,H,I], 6, [A,B,C,D,E,x,G,H,I]).

xmove([A,B,C,D,E,F,b,H,I], 7, [A,B,C,D,E,F,x,H,I]).

xmove([A,B,C,D,E,F,G,b,I], 8, [A,B,C,D,E,F,G,x,I]).

xmove([A,B,C,D,E,F,G,H,b], 9, [A,B,C,D,E,F,G,H,x]).

xmove(Board, \_, Board) :- write('Illegal move.'), nl.

% The 0-place predicate playo starts a game with the user.

playo :- explain, playfrom([b,b,b,b,b,b,b,b,b]).

explain :-

write('You play X by entering integer positions followed by a period.'),

nl,

display([1,2,3,4,5,6,7,8,9]).

playfrom(Board) :- win(Board, x), write('You win!').

playfrom(Board) :- win(Board, o), write('I win!').

playfrom(Board) :- read(N),

xmove(Board, N, Newboard),

display(Newboard),

orespond(Newboard, Newnewboard),

display(Newnewboard),

playfrom(Newnewboard).

**Output:**

