**Lab 10**

Q1. Write a code of tic-tac-toe program in CLISP using AI machine learning concepts.

|  |
| --- |
| (defconstant empty 0 "An empty square") |
|  | (defconstant cross -1 "A cross") |
|  | (defconstant naught 1 "A naught") |
|  |  |
|  | (defconstant winning-positions |
|  | '( |
|  | ;; Horizontal positions |
|  | (0 1 2) |
|  | (3 4 5) |
|  | (6 7 8) |
|  |  |
|  | ;; Vertical |
|  | (0 3 6) |
|  | (1 4 7) |
|  | (2 5 8) |
|  |  |
|  | ;; Diagonal |
|  | (0 4 8) |
|  | (6 4 2))) |
|  |  |
|  | (deftype piece () `(integer ,empty ,naught)) |
|  | (defun name-of-piece (piece) |
|  | (cond |
|  | ((eql piece cross) "X") |
|  | ((eql piece naught) "O") |
|  | ((eql piece empty) ".") |
|  | (t nil))) |
|  |  |
|  | (defun opponent (player) |
|  | (if (eql player cross) |
|  | naught |
|  | cross)) |
|  |  |
|  | (deftype board () '(simple-array piece (9))) |
|  |  |
|  | (defun bref (board square) |
|  | (aref board square)) |
|  |  |
|  | (defsetf bref (board square) (val) |
|  | `(setf (aref ,board ,square) ,val)) |
|  |  |
|  | (defun empty-cells (board) |
|  | "Given a board, return a list of indices corresponding to empty |
|  | cells" |
|  | (loop for i from 0 upto 8 when (eql (aref board i) empty) collect i)) |
|  |  |
|  | (defun initial-board () |
|  | "Initialise a board to 9 empty squares" |
|  | (let ((board (make-array 9 :initial-element empty))) |
|  | board)) |
|  |  |
|  | (defun copy-board (board) |
|  | (copy-seq board)) |
|  |  |
|  | (defun valid-move? (move board) |
|  | (numberp (position move (empty-cells board)))) |
|  |  |
|  | (defun get-move (strategy player board) |
|  | (let ((move (funcall strategy player (copy-board board)))) |
|  | (cond |
|  | ((valid-move? move board) |
|  | (format t "~&~c moves to ~d." (name-of-piece player) move) |
|  | (make-move move player board)) |
|  | (t (warn "Illegal move: ~d" move) |
|  | (get-move strategy player board))))) |
|  |  |
|  | (defun make-move (move player board) |
|  | (setf (bref board move) player) |
|  | board) |
|  |  |
|  | (defun minimax (player board) |
|  | ;; Loop for moves in the empty cells of the board |
|  | ;; Try playing that move |
|  | ;; Get the score - minimax of the new board for the opponent |
|  | ;; If that score is better, use that as the score with the position |
|  | (if (not (null (outcome board))) |
|  | (progn |
|  | (values (\* player (outcome board)) |
|  | 0)) |
|  | (let ((best-val -2) |
|  | (best-move nil) |
|  | (moves (empty-cells board))) |
|  | (dolist (move moves) |
|  | (let\* ((board2 (make-move move player (copy-board board))) |
|  | (val (- (minimax (opponent player) board2)))) |
|  | (when (or (eql best-val -2) |
|  | (> val best-val)) |
|  | (setf best-val val) |
|  | (setf best-move move)))) |
|  | (values best-val best-move)))) |
|  |  |
|  | (defun minimax-strategy (player board) |
|  | (print "In minimax strategy") |
|  | (multiple-value-bind (value move) |
|  | (minimax player board) |
|  | (declare (ignore value)) |
|  | move)) |
|  |  |
|  | (defun winning-position-met (board) |
|  | (loop for winning-position in winning-positions |
|  | when |
|  | (and (not (eql (bref board (first winning-position)) 0)) |
|  | (eql (bref board (first winning-position)) |
|  | (bref board (second winning-position))) |
|  | (eql (bref board (first winning-position)) |
|  | (bref board (third winning-position)))) |
|  | return (bref board (first winning-position)))) |
|  |  |
|  | (defun outcome (board) |
|  | (let ((winning-position (winning-position-met board))) |
|  | (cond |
|  | ((numberp winning-position) winning-position) |
|  | ((eql (length (empty-cells board)) 0) 0) |
|  | (t nil)))) |
|  |  |
|  | (defun tictactoe (c-strategy n-strategy) |
|  | (let ((board (initial-board))) |
|  | (loop for player = cross |
|  | then (opponent player) |
|  | for strategy = (if (eql player cross) |
|  | c-strategy |
|  | n-strategy) |
|  | until (not (null (outcome board))) |
|  | do |
|  | (get-move strategy player board) |
|  | (print-board board)) |
|  | (printable-result (outcome board)))) |
|  |  |
|  | (defun printable-result (outcome) |
|  | (cond |
|  | ((eql outcome cross) (format t "~&Crosses won")) |
|  | ((eql outcome naught) (format t "~&Naughts won")) |
|  | ((eql outcome 0) (format t "~&The game is a tie")))) |
|  |  |
|  | (defun human (player board) |
|  | "A human player for tictactoe" |
|  | (declare (ignore board)) |
|  | (format t "~&~c to move: " (name-of-piece player)) |
|  | (read)) |
|  |  |
|  | (defun random-strategy (player board) |
|  | "Make any legal move" |
|  | (declare (ignore player)) |
|  | (random-elt (empty-cells board))) |
|  |  |
|  | (defun random-elt (choices) |
|  | (elt choices (random (length choices)))) |
|  |  |
|  | (defun print-board (board) |
|  | (format t "~2& 1 2 3") |
|  | (loop for row from 0 to 2 do |
|  | (format t "~& ~d " (\* 10 (+ row 1))) |
|  | (loop for col from 0 to 2 |
|  | for piece = (bref board (+ col (\* row 3))) |
|  | do (format t "~c " (name-of-piece piece)))) |
|  | (format t "~2&")) |

**Q2. Write a report on your program titled Machine Learning in Tic-Tac-Toe game. Describe features of your program in the report.**

1. **INTRODUCTION**

Tic-Tac-Toe is a famous basic amusement for two players everywhere throughout the world. Customarily one individual plays for "X" while another plays for "O". The diversion is played on the square leading body of 4x4 TicTac-Toe or significantly more size. The point of this amusement for player is to win by loading with "X" or "O" any line, section or corner to corner of the principle diversion board [1, 2]. The diversion sets players figuring since players ought to mirror a tad on methodology and foresee a few moves of the rival. So it instructs individuals to settle on choices rapidly. Python is computer programming for specialized figuring from the programming Works . Utilized for wide assortment of logical and building estimations, the point of the report is to depict how Tic-Tac-Toe diversion reenactment was made with the assistance of python. The report portrays how program peruses the code and methodology for winning in the amusement. The objective of this undertaking is to give a Python program that plays a session of 4x4 Tic -Tac Toe. The Tic-tac-toes amusement rationale does not make presumptions about the structure of the board. This enables us to effectively build Tic Toe diversions with various board arrangements. The board is spoken to as a twodimensional cluster. A wide range of assessment capacities are given. Additionally included is a capacity for checking if the diversion has been won. We were actualizing twodimensional exhibit Tic Tac 4x4 amusement. This everwell known amusement can be played against two individuals and at last the diversion will end telling which individual is the victor and which individual has lost. The multi-dimensional cluster is utilized to store the X's and O's that the players enter . The single dimensional cluster is utilized to store the circumstances that either X or O has won. The other thing that it stores number of feline's amusement that has been played, to do this, we began by characterizing both clusters. After this we started to request that the two players enter the qualities for their coveted move. Each time esteem is entered by a client the program test to ensure that spot hasn't been already picked just to ensure that no bamboozling goes on. No test for a 4 out of a line is performed inside the initial four moves since it isn't conceivable to win in the initial four moves regardless of how indiscreet an adversary might be. After the initial four moves another test is added to the system. This new test is to test if the qualities entered by the client are connected together, four out of a column, corner to corner, on a level plane, or vertically.

**Main Objective**

The aim of this project is to develop a Tic-Tac-Toe 4 game. The game is supposed to consist of two parts, one a single player game ‘X’ and the other a player game ‘O’ (both are the group members or anyone who is willing to play a game) playing against each other. The goal of the diversion is to put four ‘X’s’ or ‘O’s’ pieces in succession.