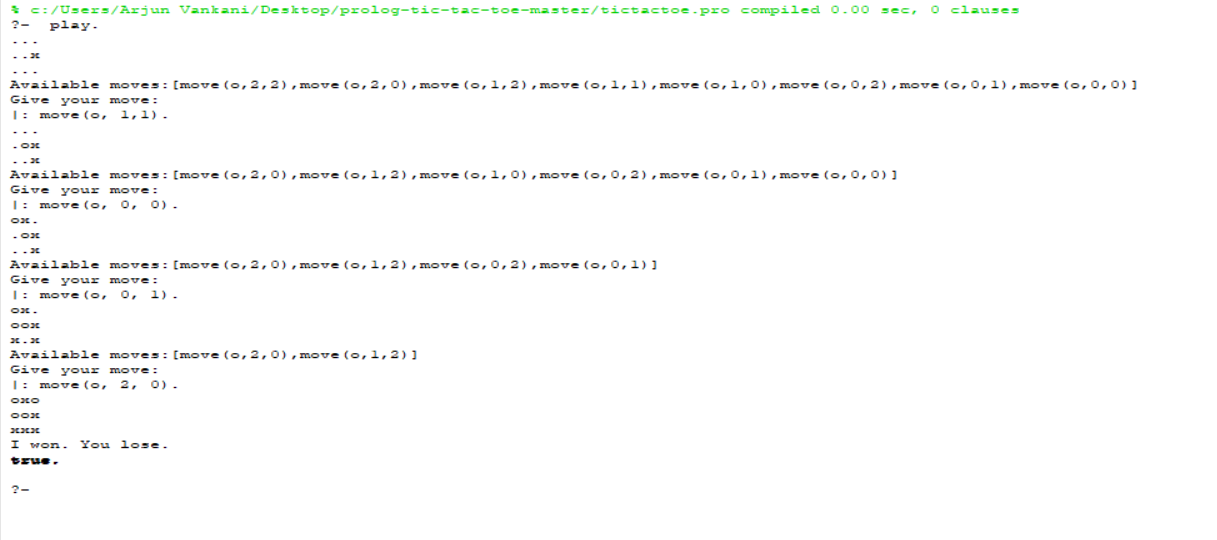
**Artificial Intelligence**

* **Practical-1: Write a program to implement Tic-Tac-Toe game problem.**
* **Code in Prolog:**

|  |
| --- |
| play :- my\_turn([]). |
|  |  |
|  | my\_turn(Game) :- |
|  | valid\_moves(ValidMoves, Game, x), |
|  | any\_valid\_moves(ValidMoves, Game). |
|  |  |
|  | any\_valid\_moves([], \_) :- |
|  | write('It is a tie'), nl. |
|  | any\_valid\_moves([\_|\_], Game) :- |
|  | findall(NextMove, game\_analysis(x, Game, NextMove), MyMoves), |
|  | do\_a\_decision(MyMoves, Game). |
|  |  |
|  | % This can only fail in the beginning. |
|  | do\_a\_decision(MyMoves, Game) :- |
|  | not(MyMoves = []), |
|  | length(MyMoves, MaxMove), |
|  | random(0, MaxMove, ChosenMove), |
|  | nth0(ChosenMove, MyMoves, X), |
|  | NextGame = [X | Game], |
|  | print\_game(NextGame), |
|  | (victory\_condition(x, NextGame) -> |
|  | (write('I won. You lose.'), nl); |
|  | your\_turn(NextGame), !). |
|  |  |
|  | your\_turn(Game) :- |
|  | valid\_moves(ValidMoves, Game, o), |
|  | (ValidMoves = [] -> (write('It is a tie'), nl); |
|  | (write('Available moves:'), write(ValidMoves), nl, |
|  | ask\_move(Y, ValidMoves), |
|  | NextGame = [Y | Game], |
|  | (victory\_condition(o, NextGame) -> |
|  | (write('I lose. You win.'), nl); |
|  | my\_turn(NextGame), !))). |
|  |  |
|  | ask\_move(Move, ValidMoves) :- |
|  | write('Give your move:'), nl, |
|  | read(Move), member(Move, ValidMoves), !. |
|  |  |
|  | ask\_move(Y, ValidMoves) :- |
|  | write('not a move'), nl, |
|  | ask\_move(Y, ValidMoves). |
|  |  |
|  | movement\_prompt(X, Y, ValidMoves) :- |
|  | write('Give your X:'), nl, read(X), member(move(o, X, Y), ValidMoves), !, |
|  | write('Give your Y:'), nl, read(Y), member(move(o, X, Y), ValidMoves). |
|  |  |
|  | % A routine for printing games.. Well you can use it. |
|  | print\_game(Game) :- |
|  | plot\_row(0, Game), plot\_row(1, Game), plot\_row(2, Game). |
|  |  |
|  | plot\_row(Y, Game) :- |
|  | plot(Game, 0, Y), plot(Game, 1, Y), plot(Game, 2, Y), nl. |
|  |  |
|  | plot(Game, X, Y) :- |
|  | (member(move(P, X, Y), Game), ground(P)) -> write(P) ; write('.'). |
|  |  |
|  | % This system determines whether there's a perfect play available. |
|  | game\_analysis(\_, Game, \_) :- |
|  | victory\_condition(Winner, Game), |
|  | Winner = x. % We do not want to lose. |
|  | % Winner = o. % We do not want to win. (egostroking mode). |
|  | % true. % If you remove this constraint entirely, it may let you win. |
|  | game\_analysis(Turn, Game, NextMove) :- |
|  | not(victory\_condition(\_, Game)), |
|  | game\_analysis\_continue(Turn, Game, NextMove). |
|  |  |
|  | game\_analysis\_continue(Turn, Game, NextMove) :- |
|  | valid\_moves(Moves, Game, Turn), |
|  | game\_analysis\_search(Moves, Turn, Game, NextMove). |
|  |  |
|  | % Comment these away and the system refuses to play, |
|  | % because there are no ways to play this without a possibility of tie. |
|  | game\_analysis\_search([], o, \_, \_). % Tie on opponent's turn. |
|  | game\_analysis\_search([], x, \_, \_). % Tie on our turn. |
|  |  |
|  | game\_analysis\_search([X|Z], o, Game, NextMove) :- % Whatever opponent does, |
|  | NextGame = [X | Game], % we desire not to lose. |
|  | game\_analysis\_search(Z, o, Game, NextMove), |
|  | game\_analysis(x, NextGame, \_), !. |
|  |  |
|  | game\_analysis\_search(Moves, x, Game, NextMove) :- |
|  | game\_analysis\_search\_x(Moves, Game, NextMove). |
|  |  |
|  | game\_analysis\_search\_x([X|\_], Game, X) :- |
|  | NextGame = [X | Game], |
|  | game\_analysis(o, NextGame, \_). |
|  | game\_analysis\_search\_x([\_|Z], Game, NextMove) :- |
|  | game\_analysis\_search\_x(Z, Game, NextMove). |
|  |  |
|  | % This thing describes all kinds of valid games. |
|  | valid\_game(Turn, Game, LastGame, Result) :- |
|  | victory\_condition(Winner, Game) -> |
|  | (Game = LastGame, Result = win(Winner)) ; |
|  | valid\_continuing\_game(Turn, Game, LastGame, Result). |
|  |  |
|  | valid\_continuing\_game(Turn, Game, LastGame, Result) :- |
|  | valid\_moves(Moves, Game, Turn), |
|  | tie\_or\_next\_game(Moves, Turn, Game, LastGame, Result). |
|  |  |
|  | tie\_or\_next\_game([], \_, Game, Game, tie). |
|  | tie\_or\_next\_game(Moves, Turn, Game, LastGame, Result) :- |
|  | valid\_gameplay\_move(Moves, NextGame, Game), |
|  | opponent(Turn, NextTurn), |
|  | valid\_game(NextTurn, NextGame, LastGame, Result). |
|  |  |
|  | % Victory conditions for tic tac toe. |
|  | victory(P, Game, Begin) :- |
|  | valid\_gameplay(Game, Begin), |
|  | victory\_condition(P, Game). |
|  |  |
|  | victory\_condition(P, Game) :- |
|  | (X = 0; X = 1; X = 2), |
|  | member(move(P, X, 0), Game), |
|  | member(move(P, X, 1), Game), |
|  | member(move(P, X, 2), Game). |
|  |  |
|  | victory\_condition(P, Game) :- |
|  | (Y = 0; Y = 1; Y = 2), |
|  | member(move(P, 0, Y), Game), |
|  | member(move(P, 1, Y), Game), |
|  | member(move(P, 2, Y), Game). |
|  |  |
|  | victory\_condition(P, Game) :- |
|  | member(move(P, 0, 2), Game), |
|  | member(move(P, 1, 1), Game), |
|  | member(move(P, 2, 0), Game). |
|  |  |
|  | victory\_condition(P, Game) :- |
|  | member(move(P, 0, 0), Game), |
|  | member(move(P, 1, 1), Game), |
|  | member(move(P, 2, 2), Game). |
|  |  |
|  | % This describes a valid form of gameplay. |
|  | % Which player did the move is disregarded. |
|  | valid\_gameplay(Start, Start). |
|  |  |
|  | valid\_gameplay(Game, Start) :- |
|  | valid\_gameplay(PreviousGame, Start), |
|  | valid\_moves(Moves, PreviousGame, \_), |
|  | valid\_gameplay\_move(Moves, Game, PreviousGame). |
|  |  |
|  | valid\_gameplay\_move([X|\_], [X|PreviousGame], PreviousGame). |
|  | valid\_gameplay\_move([\_|Z], Game, PreviousGame) :- |
|  | valid\_gameplay\_move(Z, Game, PreviousGame). |
|  |  |
|  | % The set of valid moves must not be affected by the decision making |
|  | % of the prolog interpreter. |
|  | % Therefore we have to retrieve them like this. |
|  | % This is equivalent to the (∀x∈0..2)(∀y∈0..2)(.... |
|  | % uh wait.. There's no way to represent this using those quantifiers. |
|  | valid\_moves(Moves, Game, Turn) :- |
|  | valid\_moves\_column(0, M1, [], Game, Turn), |
|  | valid\_moves\_column(1, M2, M1, Game, Turn), |
|  | valid\_moves\_column(2, Moves, M2, Game, Turn). |
|  |  |
|  | valid\_moves\_column(X, M3, M0, Game, Turn) :- |
|  | valid\_moves\_cell(X, 0, M1, M0, Game, Turn), |
|  | valid\_moves\_cell(X, 1, M2, M1, Game, Turn), |
|  | valid\_moves\_cell(X, 2, M3, M2, Game, Turn). |
|  |  |
|  | valid\_moves\_cell(X, Y, M1, M0, Game, Turn) :- |
|  | member(move(\_, X, Y), Game) -> M0 = M1 ; M1 = [move(Turn,X,Y) | M0]. |
|  |  |
|  | % valid\_move(X, Y, Game) :- |
|  | % (X = 0; X = 1; X = 2), |
|  | % (Y = 0; Y = 1; Y = 2), |
|  | % not(member(move(\_, X, Y), Game)). |
|  |  |
|  | opponent(x, o). |
|  | opponent(o, x). |

**Output:**



CODE in Python:

import pygame

from pygame.locals import \*

Board\_width = 3 # number of columns in the board

Board\_height = 3 # number of rows in the board

Tile\_size = 100

Window\_width = 480

Window\_height = 480

FPS = 30 # Frames per second

Blank = None

# R G B

Black = ( 0, 0, 0)

White = (255, 255, 255)

Green = ( 0, 204, 0)

Dark\_turquoise = ( 3, 54, 73)

Magenta = ( 255, 0, 255)

Background\_color = Dark\_turquoise

Tile\_color = Magenta

Text\_color = White

Border\_color = Green

Font\_size = 20

Button\_color = White

Button\_text\_color = Black

Message\_color = White

Blank = 10

Player\_O = 11

Player\_X = 21

Player\_O\_win = Player\_O \* 3

Player\_X\_win = Player\_X \* 3

Continue\_Game = 10

Draw\_Game = 20

Quit\_Game = 30

X\_margin = int((Window\_width - (Tile\_size \* Board\_width + (Board\_width - 1))) / 2)

Y\_margin = int((Window\_height - (Tile\_size \* Board\_height + (Board\_height - 1))) / 2)

choice = 0

def Check\_Winner(board):

def Check\_Draw():

return sum(board)%10 == 9

def check\_horizontal(player): # Horizontal Win

for i in [0, 3, 6]:

if sum(board[i:i+3]) == 3 \* player:

return player

def check\_vertical(player): # Vertical Win

for i in range(3):

if sum(board[i::3]) == 3 \* player:

return player

def check\_diagonals(player): # Main Diagonal Win

if (sum(board[0::4]) == 3 \* player) or (sum(board[2:7:2]) == 3 \* player):

return player

for player in [Player\_X, Player\_O]:

if any([check\_horizontal(player), check\_vertical(player), check\_diagonals(player)]):

return player

return Draw\_Game if Check\_Draw() else Continue\_Game

def unit\_score(winner, depth):

if winner == Draw\_Game:

return 0

else:

return 10 - depth if winner == Player\_X else depth - 10

def get\_available\_move(board):

return [i for i in range(9) if board[i] == Blank]

def minimax(board, depth):

global choice

result = Check\_Winner(board)

if result != Continue\_Game:

return unit\_score(result, depth)

depth += 1 # index of the node in the game tree

scores = [] # an array of scores

steps = [] # an array of moves(steps)

for step in get\_available\_move(board):

score = minimax(update\_state(board, step, depth), depth)

scores.append(score)

steps.append(step)

if depth % 2 == 1:

max\_value\_index = scores.index(max(scores))

choice = steps[max\_value\_index]

return max(scores)

else:

min\_value\_index = scores.index(min(scores))

choice = steps[min\_value\_index]

return min(scores)

def update\_state(board, step, depth):

board = list(board)

board[step] = Player\_X if depth % 2 else Player\_O

return board

def update\_board(board, step, player):

board[step] = player

def change\_to\_player(player):

if player == Player\_O:

return 'O'

elif player == Player\_X:

return 'X'

elif player == Blank:

return '-'

def Draw\_Board(board, message):

displaySurf.fill(Background\_color)

if message:

textSurf, textRect = makeText(message, Message\_color, Background\_color, 5, 5)

displaySurf.blit(textSurf, textRect)

for tile\_x in range(3):

for tile\_y in range(3):

if board[tile\_x\*3+tile\_y] != Blank:

drawTile(tile\_x, tile\_y, board[tile\_x\*3+tile\_y])

left, top = get\_Left\_Top\_Of\_Tile(0, 0)

width = Board\_width \* Tile\_size

height = Board\_height \* Tile\_size

pygame.draw.rect(displaySurf, Border\_color, (left - 5, top - 5, width + 11, height + 11), 4)

displaySurf.blit(New\_surf, New\_rect)

displaySurf.blit(New\_surf2, New\_rect2)

def get\_Left\_Top\_Of\_Tile(tile\_X, tile\_Y):

left = X\_margin + (tile\_X \* Tile\_size) + (tile\_X - 1)

top = Y\_margin + (tile\_Y \* Tile\_size) + (tile\_Y - 1)

return (left, top)

def makeText(text, color, bgcolor, top, left):

textSurf = Basic\_font.render(text, True, color, bgcolor)

textRect = textSurf.get\_rect()

textRect.topleft = (top, left)

return (textSurf, textRect)

def drawTile(tile\_x, tile\_y, symbol, adj\_x=0, adj\_y=0):

left, top = get\_Left\_Top\_Of\_Tile(tile\_x, tile\_y)

pygame.draw.rect(displaySurf, Tile\_color, (left + adj\_x, top + adj\_y, Tile\_size, Tile\_size))

textSurf = Basic\_font.render(symbol\_to\_str(symbol), True, Text\_color)

textRect = textSurf.get\_rect()

textRect.center = left + int(Tile\_size / 2) + adj\_x, top + int(Tile\_size / 2) + adj\_y

displaySurf.blit(textSurf, textRect)

def symbol\_to\_str(symbol):

if symbol == Player\_O:

return 'O'

elif symbol == Player\_X:

return 'X'

def get\_spot\_clicked(x, y):

for tile\_X in range(3):

for tile\_Y in range(3):

left, top = get\_Left\_Top\_Of\_Tile(tile\_X, tile\_Y)

tileRect = pygame.Rect(left, top, Tile\_size, Tile\_size)

if tileRect.collidepoint(x, y):

return (tile\_X, tile\_Y)

return None

def board\_to\_step(spot\_x, spot\_y):

return spot\_x \* 3 + spot\_y

def check\_valid\_move(coords, board):

step = board\_to\_step(\*coords)

return board[step] == Blank

def main():

global FPS\_clock, displaySurf, Basic\_font, New\_surf, New\_rect, New\_surf2, New\_rect2

two\_player = False #by default false

pygame.init()

FPS\_clock = pygame.time.Clock()

displaySurf = pygame.display.set\_mode((Window\_width, Window\_height))

pygame.display.set\_caption('Tic Tac Toe')

Basic\_font = pygame.font.Font('freesansbold.ttf', Font\_size)

New\_surf, New\_rect = makeText('vs AI', Text\_color, Tile\_color, Window\_width - 120, Window\_height - 60)

New\_surf2, New\_rect2 = makeText('vs Human', Text\_color, Tile\_color, Window\_width - 240, Window\_height - 60)

board = [Blank] \* 9

game\_over = False

x\_turn = True

msg = "Welcome to this game" # Contains the message to show in the upper left corner.

Draw\_Board(board, msg)

pygame.display.update() # pygame.display.update() is called to draw the display Surface object on the actual computer screen

while True:

coords = None

for event in pygame.event.get(): # event handling loop

if event.type == MOUSEBUTTONUP: # If the type of event was a MOUSEBUTTONUP event (that is, the player had released a mouse button somewhere over the window), then we pass the mouse coordinates to our getSpotClicked() function which will return the board coordinates of the spot on the board the mouse release happened. The event.pos[0] is the X coordinate and event.pos[1] is the Y coordinate.

coords = get\_spot\_clicked(event.pos[0], event.pos[1])

if not coords and New\_rect.collidepoint(event.pos):

board = [Blank] \* 9

game\_over = False

msg = "Welcome to this game"

Draw\_Board(board, msg)

pygame.display.update()

two\_player = False

if not coords and New\_rect2.collidepoint(event.pos):

board = [Blank] \* 9

game\_over = False

msg = "Welcome to this game"

Draw\_Board(board, msg)

pygame.display.update()

two\_player = True

if coords and check\_valid\_move(coords, board) and not game\_over:

if two\_player:

next\_step = board\_to\_step(\*coords)

if x\_turn:

update\_board(board, next\_step, Player\_X)

x\_turn = False

else:

update\_board(board, next\_step, Player\_O)

x\_turn = True

Draw\_Board(board, msg)

pygame.display.update()

if not two\_player:

next\_step = board\_to\_step(\*coords)

update\_board(board, next\_step, Player\_X)

Draw\_Board(board, msg)

pygame.display.update()

minimax(board, 0)

update\_board(board, choice, Player\_O)

result = Check\_Winner(board)

game\_over = (result != Continue\_Game)

if result == Player\_X:

msg = "The winner of this game is X"

elif result == Player\_O:

msg = "The winner of this game is O"

elif result == Draw\_Game:

msg = "Draw Game"

Draw\_Board(board, msg)

pygame.display.update()

if \_\_name\_\_ == '\_\_main\_\_':

main()

Output:

