

SRM INSTITUTE OF SCIENCE & TECHNOLOGY DEPARTMENT OF NETWORKING & COMMUNICATIONS **18CSC305J-ARTIFICIAL INTELLIGENCE**

SEMESTER – 6 BATCH-2

|  |  |
| --- | --- |
| **REGISTRATION NUMBER** | **RA1911003011043** |
| **NAME** | **RAMNATH** |

# B.Tech- CSE / CC, Third Year (Section: H2)

**Faculty In charge: Dr. S. Prabakeran, B.Tech, M.E, PhD**

**Assistant Professor**

**School of Computing - Department of**

**Networking and Communications**

**Year 2021-2022 / Even Semester**

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ex No** | **DATE** | **Title** | **Page No** | **Marks** |
| 1 | 14/01/2022 | Toy Problem: Tic Tac Toe |  |  |

## Exercise: 1

**Date : 21-01-2021**

**TOY PROBLEM**

Problem Statement :

Two players, named ‘player1’ and ‘player2’, play a tic-tac-toe game on a grid of size ‘3 x 3’. Given an array ‘moves’ of size ‘n’, where each element of the array is a tuple of the form (row, column) representing a position on the grid. Players place their characters alternatively in the sequence of positions given in ‘moves’. Consider that ‘player1’ makes the first move. Your task is to return the winner of the game, i.e., the winning player’s name. If there is no winner and some positions remain unmarked, return ‘uncertain’. Otherwise, the game ends in a draw, i.e., when all positions are marked without any winner, return ‘draw’.

## Algorithm :

 The game is to be played between two people (in this program between HUMAN and COMPUTER).

 One of the player chooses ‘O’ and the other ‘X’ to mark their respective cells.

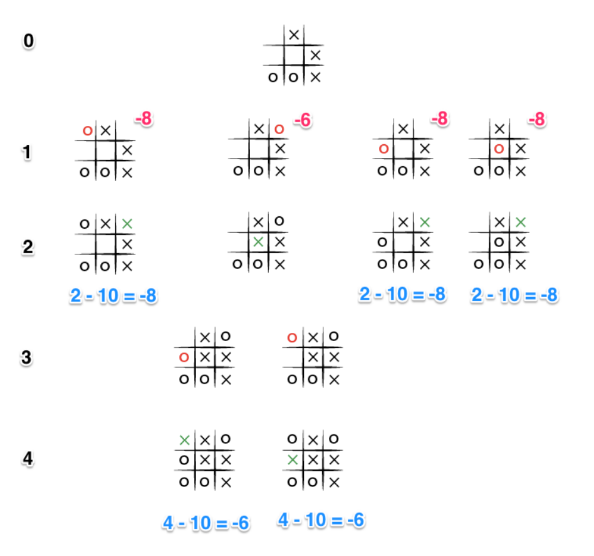
 The game starts with one of the players and the game ends when one of the players has one whole row/ column/ diagonal filled with his/her respective character (‘O’ or ‘X’).

 If no one wins, then the game is said to be draw.

**Optimization technique :** The key is to use Minimax algorithm . A back and forth between the two players, where the player whose "turn it is" desires to pick the move with the maximum score. In turn, the scores for each of the available moves are determined by the opposing player deciding which of its available moves has the minimum score. And the scores for the opposing players moves are again determined by the turn-taking player trying to maximize its score and so on all the way down the move tree to an end state.

A description for the algorithm, assuming X is the "turn taking player,"

* If the game is over, return the score from X's perspective.
* Otherwise get a list of new game states for every possible move
* Create a scores list
* For each of these states add the minimax result of that state to the scores list
* If it's X's turn, return the maximum score from the scores list
* If it's O's turn, return the minimum score from the scores list



**Tool :** VS Code and Python 3.9.0

## Programming code :

import numpy as np

import random

from time import sleep

# Creates an empty board

def create\_board():

return(np.array([[0, 0, 0],

[0, 0, 0],

[0, 0, 0]]))

# Check for empty places on board

def possibilities(board):

l = []

for i in range(len(board)):

for j in range(len(board)):

if board[i][j] == 0:

l.append((i, j))

return(l)

# Select a random place for the player

def random\_place(board, player):

selection = possibilities(board)

current\_loc = random.choice(selection)

board[current\_loc] = player

return(board)

# Checks whether the player has three

# of their marks in a horizontal row

def row\_win(board, player):

for x in range(len(board)):

win = True

for y in range(len(board)):

if board[x, y] != player:

win = False

continue

if win == True:

return(win)

return(win)

# Checks whether the player has three

# of their marks in a vertical row

def col\_win(board, player):

for x in range(len(board)):

win = True

for y in range(len(board)):

if board[y][x] != player:

win = False

continue

if win == True:

return(win)

return(win)

# Checks whether the player has three

# of their marks in a diagonal row

def diag\_win(board, player):

win = True

y = 0

for x in range(len(board)):

if board[x, x] != player:

win = False

if win:

return win

win = True

if win:

for x in range(len(board)):

y = len(board) - 1 - x

if board[x, y] != player:

win = False

return win

# Evaluates whether there is

# a winner or a tie

def evaluate(board):

winner = 0

for player in [1, 2]:

if (row\_win(board, player) or

col\_win(board,player) or

diag\_win(board,player)):

winner = player

if np.all(board != 0) and winner == 0:

winner = -1

return winner

# Main function to start the game

def play\_game():

board, winner, counter = create\_board(), 0, 1

print(board)

sleep(2)

while winner == 0:

for player in [1, 2]:

board = random\_place(board, player)

print("Board after " + str(counter) + " move")

print(board)

sleep(2)

counter += 1

winner = evaluate(board)

if winner != 0:

break

return(winner)

# Driver Code

print("Winner is: " + str(play\_game()))

## Output screen shots :

## 

## 

**Result :**  The Tic Tac Toe problem was implemented successfully using minmax algorithm to evaluate the best moves with the highest score.