**EXPT NO: 01 TIC-TAC-TOE GAME USING ADVERSARIAL SEARCHING ALGORITHM**

**DATE: 03.08.2023**

**AIM:**

To create the tic tac toe game using the adversarial searching algorithm.

**DESCRIPTION:**

1. **Game Initialization:**
   * Create a 3x3 grid to represent the Tic Tac Toe board.
   * Define the two players: "X" and "O".
   * Set a current player to start the game.
2. **Strategy:**
   * The minimax algorithm is a recursive approach that simulates all possible future game states by exploring the game tree.
   * Each node in the game tree represents a possible game state after a specific move.
   * The algorithm assigns a score to each terminal node (win, loss, or draw) based on a predefined evaluation function.
   * The goal is for the maximizing player ("X") to choose the move that maximizes their score, while the minimizing player ("O") aims to minimize their opponent's score.
   * Alpha-Beta Pruning is used to optimize the search by eliminating branches of the tree that are guaranteed to not affect the final decision.
3. **Evaluation Function:**
   * Design an evaluation function that assigns a score to each possible game state based on factors like the number of winning lines the player has, the number of open lines, and strategic positioning.
4. **Game Loop:**
   * Create a loop that continues until the game reaches a terminal state (win, loss, or draw).
   * In each iteration, the current player makes a move based on the adversarial algorithm.
   * The algorithm recursively explores possible moves and their outcomes to make an informed decision.
   * Update the game board based on the chosen move.
5. **Terminal State Detection:**
   * Determine whether the game has reached a terminal state (win, loss, or draw) by checking the current state of the board.
6. **Outcome and Display:**
   * Display the final outcome of the game (win, loss, or draw) once the game loop ends.
7. **User Interface (Optional):**
   * Implement a user-friendly interface to allow players to interact with the game visually, here we are using the javascript for user interface .

By following these steps, we can create a Tic Tac Toe game that utilizes an adversarial searching algorithm to make strategic decisions for both players, resulting in a challenging and engaging gameplay experience.

**CODE:**

**HTML:**

<!DOCTYPE html>

<html>

<head>

<!--

Tic Tac Toe with Minimax AI Algorithm

Author: Clederson Cruz

Year: 2017

License: GNU GENERAL PUBLIC LICENSE (GPL)

-->

<meta charset="UTF-8" />

<meta content="Clederson Cruz" name="author" />

<meta name="viewport" content="width=device-width,initial-scale=1">

<style>

body {

margin: 0;

padding: 0;

}

header {

background-color: #00AEEA;

color: #eee;

text-align: center;

height: 60px;

padding-top: 5px;

display: block;

margin-top: 0px;

margin-bottom: 60px;

box-sizing: border-box;

position: relative;

width: 100%;

}

p {

font-size: 14pt;

font-weight: bold;

font-family: Sans;

text-align: center;

position: relative;

text-align: center;

margin-left: auto;

margin-right: auto;

display: block;

}

a {

color: #00AEEA;

text-decoration: none;

}

a:hover {

color: #00AEEA;

}

a:visited {

color: #00AEEA;

}

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Table style \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

#tab-tic-tac-toe {

margin-left: auto;

margin-right: auto;

padding: 5px;

font-size: 4em;

font-family: Sans;

color: #444;

background: #F8F8F8;

width: 310px;

height: 300px;

text-align: center;

vertical-align: center;

border: 1px solid #CECECE;

border-radius: 5px;

box-shadow: 1px 1px 1px #CCC;

}

/\*Column style\*/

#tab-tic-tac-toe td {

border-collapse:collapse;

border-left: 5px solid #CCC;

border-bottom: 5px solid #CCC;

}

#tab-tic-tac-toe td:first-child {

border-left: none;

}

#tab-tic-tac-toe tr:last-child td {

border-bottom: none;

}

/\*Cells\*/

#tab-tic-tac-toe td {

cursor: pointer;

height: 95px;

width: 95px;

}

#tab-tic-tac-toe td:hover {

background: #ECECEC;

}

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Restart Button \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

#bnt-restart {

display: block;

padding: 10px;

margin-left: auto;

margin-right: auto;

width: 200px;

background: #00AEEA;

font-size: 1.5em;

color: #FFF;

border: none;

border-radius: 6px;

cursor: pointer;

}

#bnt-restart:hover {

background: #1DC5FF;

}

#bnt-restart:active {

background: #0787B3;

}

#bnt-restart:disabled {

color: #444;

background: #CECECE;

}

</style>

</head>

<body>

<br />

<p id="message"></p>

<br />

<table id="tab-tic-tac-toe" cellspacing="0">

<tr><td id="00" onclick="clickedCell(this)"></td><td id="01" onclick="clickedCell(this)"></td><td id="02" onclick="clickedCell(this)"></td></tr>

<tr><td id="10" onclick="clickedCell(this)"></td><td id="11" onclick="clickedCell(this)"></td><td id="12" onclick="clickedCell(this)"></td></tr>

<tr><td id="20" onclick="clickedCell(this)"></td><td id="21" onclick="clickedCell(this)"></td><td id="22" onclick="clickedCell(this)"></td></tr>

</table>

<br><br>

<input type="button" value="Start AI" id="bnt-restart" onclick="restartBnt(this)"/>

<br />

<script>

var board = [

[0, 0, 0],

[0, 0, 0],

[0, 0, 0],

];

var HUMAN = -1;

var COMP = +1;

/\* Function to heuristic evaluation of state. \*/

function evalute(state) {

var score = 0;

if (gameOver(state, COMP)) {

score = +1;

}

else if (gameOver(state, HUMAN)) {

score = -1;

} else {

score = 0;

}

return score;

}

/\* This function tests if a specific player wins \*/

function gameOver(state, player) {

var win\_state = [

[state[0][0], state[0][1], state[0][2]],

[state[1][0], state[1][1], state[1][2]],

[state[2][0], state[2][1], state[2][2]],

[state[0][0], state[1][0], state[2][0]],

[state[0][1], state[1][1], state[2][1]],

[state[0][2], state[1][2], state[2][2]],

[state[0][0], state[1][1], state[2][2]],

[state[2][0], state[1][1], state[0][2]],

];

for (var i = 0; i < 8; i++) {

var line = win\_state[i];

var filled = 0;

for (var j = 0; j < 3; j++) {

if (line[j] == player)

filled++;

}

if (filled == 3)

return true;

}

return false;

}

/\* This function test if the human or computer wins \*/

function gameOverAll(state) {

return gameOver(state, HUMAN) || gameOver(state, COMP);

}

function emptyCells(state) {

var cells = [];

for (var x = 0; x < 3; x++) {

for (var y = 0; y < 3; y++) {

if (state[x][y] == 0)

cells.push([x, y]);

}

}

return cells;

}

/\* A move is valid if the chosen cell is empty \*/

function validMove(x, y) {

var empties = emptyCells(board);

try {

if (board[x][y] == 0) {

return true;

}

else {

return false;

}

} catch (e) {

return false;

}

}

/\* Set the move on board, if the coordinates are valid \*/

function setMove(x, y, player) {

if (validMove(x, y)) {

board[x][y] = player;

return true;

}

else {

return false;

}

}

/\* \*\*\* AI function that choice the best move \*\*\* \*/

// Read more on https://github.com/Cledersonbc/tic-tac-toe-minimax/

function minimax(state, depth, player) {

var best;

if (player == COMP) {

best = [-1, -1, -1000];

}

else {

best = [-1, -1, +1000];

}

if (depth == 0 || gameOverAll(state)) {

var score = evalute(state);

return [-1, -1, score];

}

emptyCells(state).forEach(function (cell) {

var x = cell[0];

var y = cell[1];

state[x][y] = player;

var score = minimax(state, depth - 1, -player);

state[x][y] = 0;

score[0] = x;

score[1] = y;

if (player == COMP) {

if (score[2] > best[2])

best = score;

}

else {

if (score[2] < best[2])

best = score;

}

});

return best;

}

/\* It calls the minimax function \*/

function aiTurn() {

var x, y;

var move;

var cell;

if (emptyCells(board).length == 9) {

x = parseInt(Math.random() \* 3);

y = parseInt(Math.random() \* 3);

}

else {

move = minimax(board, emptyCells(board).length, COMP);

x = move[0];

y = move[1];

}

if (setMove(x, y, COMP)) {

cell = document.getElementById(String(x) + String(y));

cell.innerHTML = "O";

}

}

/\* main \*/

function clickedCell(cell) {

var button = document.getElementById("bnt-restart");

button.disabled = true;

var conditionToContinue = gameOverAll(board) == false && emptyCells(board).length > 0;

if (conditionToContinue == true) {

var x = cell.id.split("")[0];

var y = cell.id.split("")[1];

var move = setMove(x, y, HUMAN);

if (move == true) {

cell.innerHTML = "X";

if (conditionToContinue)

aiTurn();

}

}

if (gameOver(board, COMP)) {

var lines;

var cell;

var msg;

if (board[0][0] == 1 && board[0][1] == 1 && board[0][2] == 1)

lines = [[0, 0], [0, 1], [0, 2]];

else if (board[1][0] == 1 && board[1][1] == 1 && board[1][2] == 1)

lines = [[1, 0], [1, 1], [1, 2]];

else if (board[2][0] == 1 && board[2][1] == 1 && board[2][2] == 1)

lines = [[2, 0], [2, 1], [2, 2]];

else if (board[0][0] == 1 && board[1][0] == 1 && board[2][0] == 1)

lines = [[0, 0], [1, 0], [2, 0]];

else if (board[0][1] == 1 && board[1][1] == 1 && board[2][1] == 1)

lines = [[0, 1], [1, 1], [2, 1]];

else if (board[0][2] == 1 && board[1][2] == 1 && board[2][2] == 1)

lines = [[0, 2], [1, 2], [2, 2]];

else if (board[0][0] == 1 && board[1][1] == 1 && board[2][2] == 1)

lines = [[0, 0], [1, 1], [2, 2]];

else if (board[2][0] == 1 && board[1][1] == 1 && board[0][2] == 1)

lines = [[2, 0], [1, 1], [0, 2]];

for (var i = 0; i < lines.length; i++) {

cell = document.getElementById(String(lines[i][0]) + String(lines[i][1]));

cell.style.color = "red";

}

msg = document.getElementById("message");

msg.innerHTML = "You lose!";

}

if (emptyCells(board).length == 0 && !gameOverAll(board)) {

var msg = document.getElementById("message");

msg.innerHTML = "Draw!";

}

if (gameOverAll(board) == true || emptyCells(board).length == 0) {

button.value = "Restart";

button.disabled = false;

}

}

/\* Restart the game\*/

function restartBnt(button) {

if (button.value == "Start AI") {

aiTurn();

button.disabled = true;

}

else if (button.value == "Restart") {

var htmlBoard;

var msg;

for (var x = 0; x < 3; x++) {

for (var y = 0; y < 3; y++) {

board[x][y] = 0;

htmlBoard = document.getElementById(String(x) + String(y));

htmlBoard.style.color = "#444";

htmlBoard.innerHTML = "";

}

}

button.value = "Start AI";

msg = document.getElementById("message");

msg.innerHTML = "";

}

}

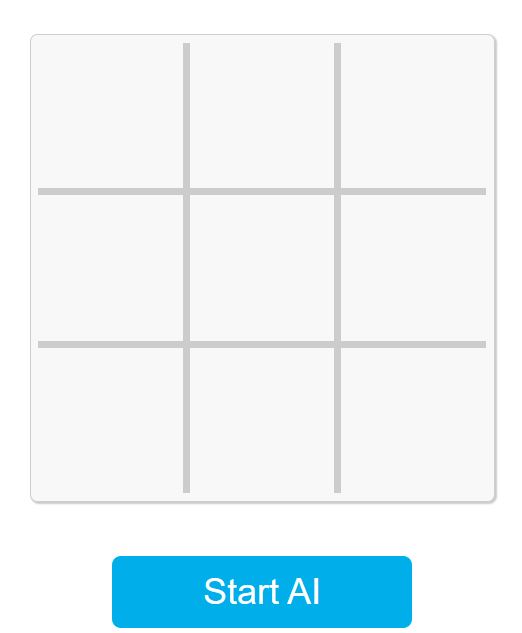
</script>

</body>

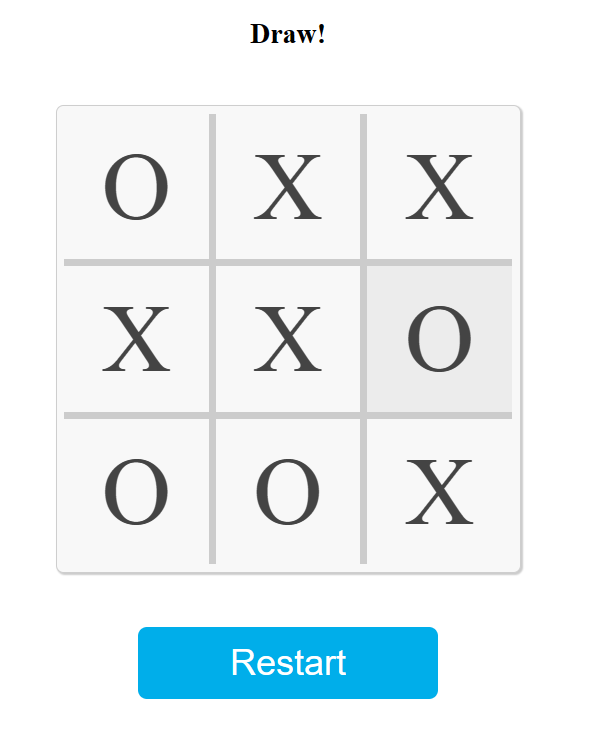
</html>

**OUTPUT:**

Home page of Tic Tac Toe



Either user can start or ai can start based on user choice



Restart button will start the game

|  |  |
| --- | --- |
| OBSERVATION (20) |  |
| RECORD (05) |  |
| TOTAL (25) |  |

**RESULT:**

Thus, the tic tac toe game using the adversarial searching algorithm are successfully implemented by javascript and the outputs are verified.

**Ex No: 4**

**Date:**

**Create the environment for probabilistic inference using a Bayesian network**

**Aim**

To write the program to implement Bayesian network.

**Algorithm**

1. Define the Bayesian Network
2. Specify the variables of interest and their dependencies. In the Burglar's Alarm example, you have variables like "Burglar," "Earthquake," "Alarm," "JohnCalls," and "MaryCalls," with their conditional dependencies.
3. Construct the Conditional Probability Tables (CPTs)
4. For each variable in the network, create its Conditional Probability Table based on domain knowledge or available data. These tables represent the probabilities of each variable given the values of its parent variables.
5. If you have any observed evidence (e.g., John's call was received), set the corresponding variables to their observed values.
6. Implement the variable elimination algorithm to compute the posterior probabilities of unobserved variables.
7. Iterate through the variables in topological order.
8. For each variable, compute the conditional probability given its parents and evidence using the CPTs.
9. Multiply these conditional probabilities to get the joint probability.
10. Sum out (marginalize) the variable to get the updated probabilities.
11. After eliminating all variables, normalize the resulting probability distribution to ensure it sums to 1.
12. Once the network is constructed and evidence is set, you can query the network to answer specific probabilistic questions.
13. For example, you can calculate the probability of the Burglar being present given John's call: P(Burglar | JohnCalls = true).
14. If the network is large and variable elimination is computationally expensive, consider using more efficient inference algorithms like Belief Propagation or Markov Chain Monte Carlo (MCMC) for approximate inference.
15. If new evidence becomes available, update the evidence in the network and re-run the inference process to get updated probabilities.
16. Present the results to the user in a meaningful way, such as probabilities or graphical representations of the network.
17. Implement error handling to account for situations where evidence is inconsistent with the network or other unexpected issues.
18. Validate the implementation by comparing the results to known solutions or simulated data.
19. Optimize the inference process for efficiency, especially if dealing with large or complex networks.

**Program**

import networkx as nx

import matplotlib.pyplot as plt

from pgmpy.models import BayesianModel

from pgmpy.factors.discrete import TabularCPD

from pgmpy.inference import VariableElimination

# Create a Bayesian network

model = BayesianModel([('Burglary', 'Alarm'),

('Earthquake', 'Alarm'),

('Alarm', 'JohnCalls'),

('Alarm', 'MaryCalls')])

# Define conditional probability distributions (CPDs)

cpd\_burglary = TabularCPD(variable='Burglary', variable\_card=2, values=[[0.001], [0.999]])

cpd\_earthquake = TabularCPD(variable='Earthquake', variable\_card=2, values=[[0.002], [0.998]])

cpd\_alarm = TabularCPD(variable='Alarm', variable\_card=2,

values=[[0.95, 0.94, 0.29, 0.001],

[0.05, 0.06, 0.71, 0.999]],

evidence=['Burglary', 'Earthquake'],

evidence\_card=[2, 2])

cpd\_john = TabularCPD(variable='JohnCalls', variable\_card=2,

values=[[0.90, 0.05],

[0.10, 0.95]],

evidence=['Alarm'],

evidence\_card=[2])

cpd\_mary = TabularCPD(variable='MaryCalls', variable\_card=2,

values=[[0.70, 0.01],

[0.30, 0.99]],

evidence=['Alarm'],

evidence\_card=[2])

# Add CPDs to the network structure

model.add\_cpds(cpd\_burglary, cpd\_earthquake, cpd\_alarm, cpd\_john, cpd\_mary)

# Check if the model is valid

assert model.check\_model(), "Model is not valid."

print('Probability distribution, P(Burglary)')

print(cpd\_burglary)

print()

print('Probability distribution, P(Earthquake)')

print(cpd\_earthquake)

print()

print('Joint probability distribution, P(Alarm | Burglary, Earthquake)')

print(cpd\_alarm)

print()

print('Joint probability distribution, P(JohnCalls | Alarm)')

print(cpd\_john)

print()

print('Joint probability distribution, P(MaryCalls | Alarm)')

print(cpd\_mary)

print()

# Plot the model

pos = {'Burglary': (0, 0), 'Earthquake': (1, 1), 'Alarm': (0.5, 0.5), 'JohnCalls': (0, -1), 'MaryCalls': (1, -1)}

nx.draw(model, pos=pos, with\_labels=True, node\_size=1000, font\_size=10, node\_color='skyblue', font\_color='black')

plt.show()

# Perform variable elimination for inference

infer = VariableElimination(model)

# Calculate the probability of burglary if John and Mary call (0: True, 1: False)

posterior\_probability = infer.query(variables=['Burglary'], evidence={'JohnCalls': 0, 'MaryCalls': 0})

# Print posterior probability

print('Posterior probability of Burglary if JohnCalls(True) and MaryCalls(True)')

print(posterior\_probability)

print()

# Calculate the probability of the alarm sounding if there is a burglary and an earthquake (0: True, 1: False)

posterior\_probability = infer.query(variables=['Alarm'], evidence={'Burglary': 0, 'Earthquake': 0})

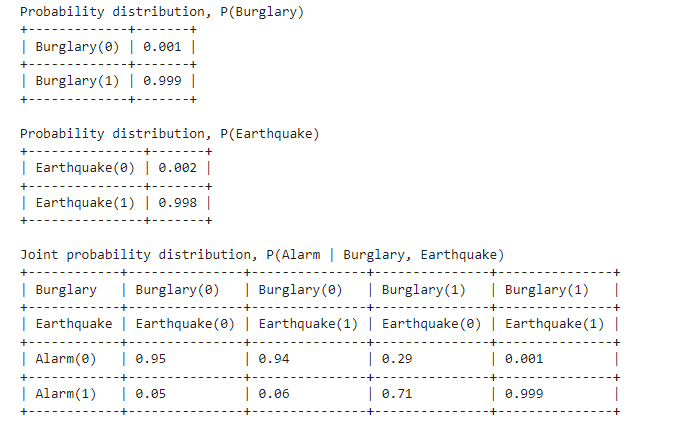
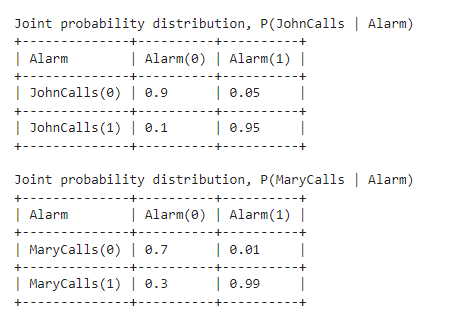
# Print posterior probability

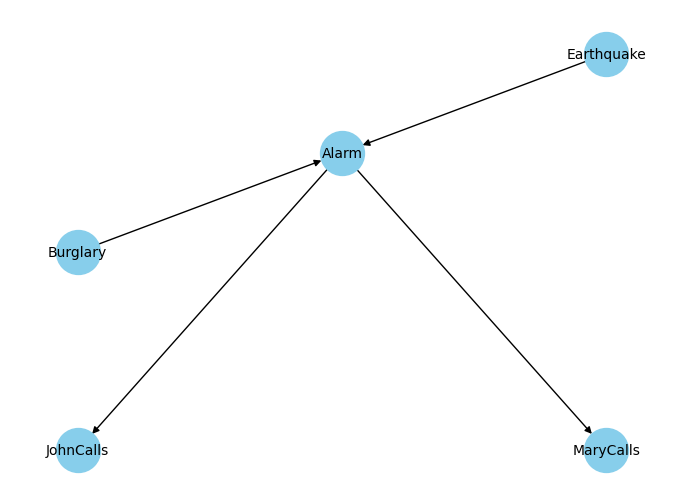
print('Posterior probability of Alarm sounding if Burglary(True) and Earthquake(True)')

print(posterior\_probability)

print()

**Output**



|  |  |
| --- | --- |
| **OBSERVATION (20)** |  |
| **RECORD (05)** |  |
| **TOTAL (25)** |  |

**Result:**

Thus, program used with Bayesian network is successfully run and output is verified.