Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Experiment # 6: Implementing Min-Max algorithm to solve given graph

Aim/Objective:

Implement Min-Max algorithm to solve given graph.

Description:

The aim of this lab Experiment is to implement the Mini-Max algorithm for solving the Tic-Tac-Toe game, we'll create a program that allows the computer to play optimally against a human player. Students will learn how to play optimally using the Mini-Max algorithm, ensuring a challenging opponent for the human player in the game of Tic-Tac-Toe.

Pre-Requisites:

- Basic understanding of machine learning concepts, including neural networks and classification algorithms.
- Familiarity with programming concepts and basic knowledge of Python programming language

Pre-Lab:

- Describe the rules of Tic-Tac-Toe. What are the win conditions and how is a draw determined?

Players take turns putting their mark in empty squares.. The first player to get 3 of her mark in empty squares. The first player to get 3 of their mark in row, column, diagnally is winner.

- What is the role of recursion in the Mini-Max algorithm? Provide an example of how recursion is used in the algorithm.

Min Max algorithm uses recursion to search through game tree. Min-max algorithm is mostly used for playing in AI such as chess and checker.

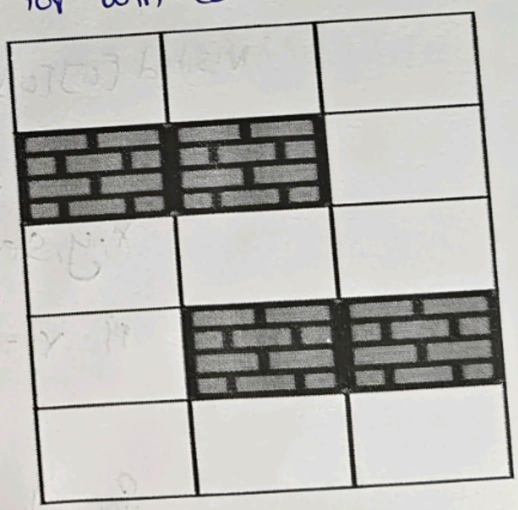
- Define what utility function means in the context of game playing algorithms.

The player in game theory must known which action they prefers in a pair wise.
Such preference in decision making can be represented by a utility function sometimes called payoff function which assigns a number for every possible action in decision making

- Describe how you would represent the Tic-Tac-Toe board and manage game states within your implementation.

The tic tac toe board is represented as 3x3 matrix and game states ~~are~~ managed by updating the board with moves ~~and~~ checking for win conditions or ties

In-Lab:



Description: Students will implement Min-Max algorithm to solve a given problem.

You are given an m x n integer matrix grid where each cell is either 0 (empty) or 1 (obstacle). You can move up, down, left, or right from and to an empty cell in one step. Return *the minimum number of steps to walk from the upper left corner* (0, 0) *to the lower right corner* (m - 1, n - 1) *given that you can eliminate at most* k *obstacles*. If it is not possible to find such walk return -1.

Input: grid = [[0,0,0],[1,1,0],[0,0,0],[0,1,1],[0,0,0]], k = 1

Output: 6

Procedure/Program:

```
from collections import deque
def shortestPath (grid, k):
    m = len(grid)
    n = len(grid[0])
    directions = [(0,1), (1,0), (0,-1), (-1,0)]
    queue = deque([0,0,0,0])
    visited = [[False]*(k+1) for _ in range(n)
                for _ in range(m)]

    visited[0][0][0] = True

    while queue:
        x,y,steps,abs = queue.popleft()
        if x == m-1 and y == n-1:
            return steps
        for dx, dy in directions:
            nx, ny = x+dx, y+dy
```

```
if 0<=nx and nx<m and 0<=ny and ny<n:
    if grid[nx][ny]=00 and 0<=ny;
        if grid[nx][ny]==0 and not visited[nx][ny]
                                    [obs]:
            visited[nx][ny][obs]=True
            queue.append((nx,ny,steps,obs))

        elif grid[nx][ny]==1 and obs<k and
            not visited[nx][ny][obs+1]:
            visited[nx][ny][obs+1]=True
            queue.append((nx,ny,steps+1,obs+1))
        return 1
grid=[[0,0,0],[1,1,0],[0,0,0],[0,1,1],[0,0,0]]
k=1
print (shortest path (grid,k))
```

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Data and Results:

The input of the given program is

grid = [[0,0,0],
[1,1,0],
[0,0,0],
[0,1,1],
[0,0,0]]

k = 1

Output :- 6

Analysis and Inferences:

The Queue initializes start from 0,0 with steps and obstacles eliminated

Visited array will keeps the track of the states to avoid redundant process

It will return the numbers of steps to reach the destination otherwise returns 1

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

VIVA-VOCE Questions (In-Lab):

1. How does the Mini-Max algorithm work in the context of a Tic-Tac-Toe game?
2. What is the purpose of the evaluation function in the Mini-Max algorithm?
3. How do you implement alpha-beta pruning to optimize the Mini-Max algorithm?
4. What are the main components of a cyber-security system for incident response?
5. How do you prioritize incidents and determine the appropriate response in a cyber-security system?

1) The Min Max algorithm in Tic Tac Toe evaluate all possible moves to choose the optimal one by minimizing the opponents maximum potential score

2) Alpha beta pruning optimizes Min Max by discarding branches of game tree that want influences the final decision reducing the no. of nodes

3) ~~A~~ cyber security incident response system includes components for detection, analysis and remediation while prioritizing incidents based on severity

4) The main components of a cyber security Incident response system include incident detection and monitoring analysis and investigation containment and eradication and recovery

5) Incidents are prioritized based on their impact urgency and value of affected assets. The response is determined by assessing severity potential bussiness imapct and available resource to address issue

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Post-Lab:

Implement a Cyber Security System for Optimizing defense strategies in network security games where an attacker and defender have conflicting objectives. Use the Minimax algorithm to model and solve the interaction between attacker and defender as a strategic game on a network graph

Description: The Min-Max algorithm is a recursive algorithm used in decision-making and game theory. It is designed for two-player zero-sum games, where one player maximizes their score (often referred to as the maximizing player, typically the AI), and the other player minimizes the score (the minimizing player, often the human opponent).

Procedure/Program:

```
def minimam(graph, node, depth, max, attacked,
            defended, critical):
    if depth==0 or cortical in attacked or
        critical in defended:
            return 100 if critical in attacked else -bo
    best = -float('inf') \
    if max else float('inf')
    for neighbor in grap[node]:
        score = minimax.(graph, neighbor, depth-1,
            not maximum, attacked, defended, crital)
        best = max (best, score if minimize
            else min (best, score)
    return best
```

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Data and Results:

The procedure helps to implements . the security System for optimizing strategies in network security games. where attacker and defender have the conflicting objectives. using min.max algorithm.

Analysis and Inferences:

The algorithm return 100 indicating that the attacker successfully reached the critical node within the given depth limit

This is optimal outcome for attacker when starting from node A with given depth.

Evaluator Remark (if Any):	Marks Secured: 47 out of 50
	Signature of the Evaluator with Date