Worksheet 3.2

Student Name: Rydem **UID: 25MCI10275**
Branch: MCA (AI & ML) **Section/Group: MAM-1 (A)**
Semester: I **Date of Performance: 02/11/25**
Subject Name: Artificial Intelligence Tools - I **Subject Code: 25CAP-614**

1. Aim/Overview of the practical:

Experiment-1:
Implement Minimax Algorithm

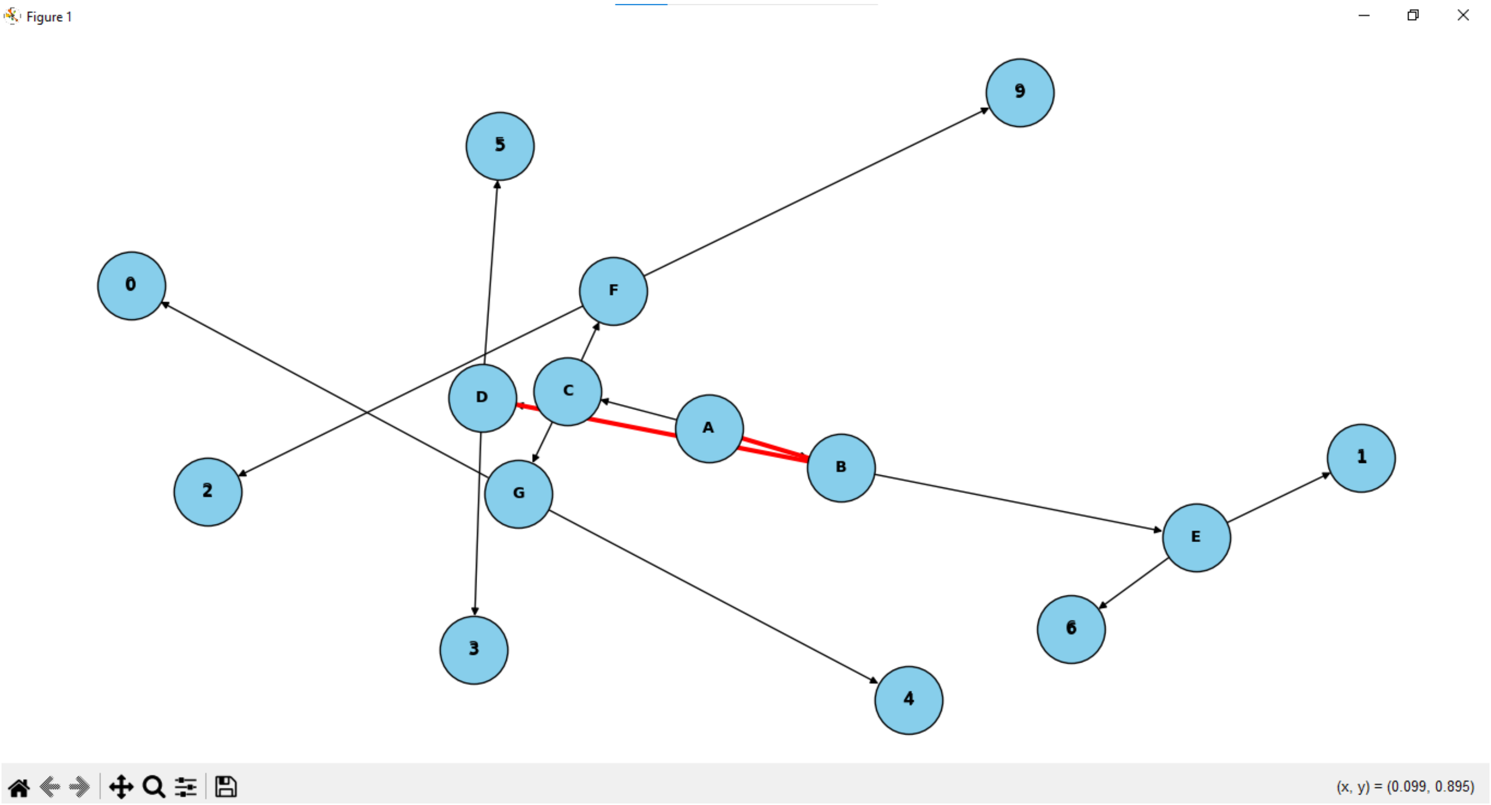
2. Coding:

EXAMPLE-1

```
import networkx as nx
import matplotlib.pyplot as plt

class GameTree:
    def __init__(self):
        self.tree = {
            'A': ['B', 'C'],
            'B': ['D', 'E'],
            'C': ['F', 'G'],
            'D': [3, 5],
            'E': [6, 1],
            'F': [2, 9],
            'G': [0, 4]
        }
    def minimax(self, node, is_max_turn=True):
        """Recursive Minimax algorithm with value return"""
        if isinstance(self.tree[node][0], int):  # leaf node
            return max(self.tree[node]) if is_max_turn else min(self.tree[node])

        child_values = {}
        for child in self.tree[node]:
            val = self.minimax(child, not is_max_turn)
            child_values[child] = val

        if is_max_turn:
            best_child = max(child_values, key=child_values.get)
        else:
            best_child = min(child_values, key=child_values.get)

        # Store best child for visualization
        self.best_path[node] = best_child
        return child_values[best_child]

    def build_graph(self):
        """Builds a NetworkX graph from the tree structure"""
        G = nx.DiGraph()
        for parent, children in self.tree.items():
            for child in children:
                G.add_edge(parent, str(child))
        return G


# --- Driver Code ---
game = GameTree()
game.best_path = {}  # store best choices during minimax
optimal_value = game.minimax('A', True)

# Build graph
G = game.build_graph()

# Node positions for visualization
```

```
pos = nx.spring_layout(G, seed=42)

# Determine all edges and optimal path edges
all_edges = list(G.edges())
optimal_edges = []
node = 'A'
while node in game.best_path:
    child = game.best_path[node]
    optimal_edges.append((node, str(child)))
    node = str(child)

# Draw all edges
nx.draw(G, pos, with_labels=True, node_size=2000, node_color="#87CEEB",
        font_size=10, font_weight="bold", edgecolors="black")

# Highlight optimal path
nx.draw_networkx_edges(G, pos, edgelist=optimal_edges, edge_color="red", width=3)

# Show labels for leaf nodes (numeric values)
leaf_labels = {str(v): str(v) for k, vals in game.tree.items() if isinstance(vals[0], int) for v in vals}
nx.draw_networkx_labels(G, pos, labels=leaf_labels, font_color="black")

# Title
plt.title(f"Minimax Game Tree (Optimal Value at Root = {optimal_value})", fontsize=12, fontweight="bold")
plt.show()
```

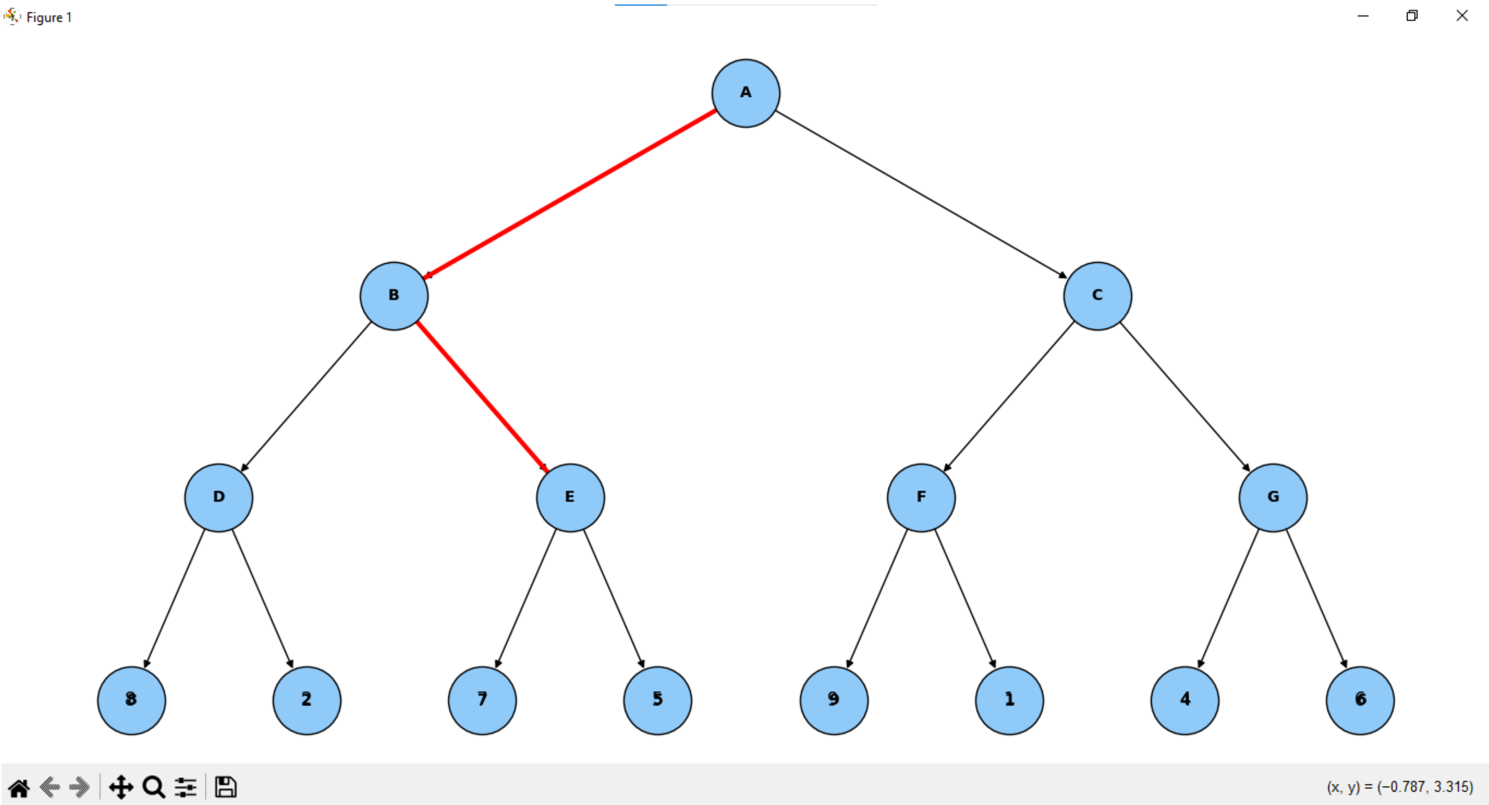
OUTPUT:-



EXAMPLE-2

```
import networkx as nx
import matplotlib.pyplot as plt

class GameTree:
    def __init__(self):
        self.tree = {
            'A': ['B', 'C'],
            'B': ['D', 'E'],
            'C': ['F', 'G'],
            'D': [8, 2],
            'E': [7, 5],
            'F': [9, 1],
            'G': [4, 6]
        }

    def minimax(self, node, is_max_turn=True):
        """Recursive Minimax algorithm"""
        if isinstance(self.tree[node][0], int):  # leaf node
            return max(self.tree[node]) if is_max_turn else min(self.tree[node])

        child_values = {}
        for child in self.tree[node]:
            val = self.minimax(child, not is_max_turn)
            child_values[child] = val

        if is_max_turn:
            best_child = max(child_values, key=child_values.get)
        else:
            best_child = min(child_values, key=child_values.get)

        # Store best child for visualization
        self.best_path[node] = best_child
        return child_values[best_child]

    def build_graph(self):
        """Builds a NetworkX graph from the tree"""
        G = nx.DiGraph()
        for parent, children in self.tree.items():
            for child in children:
                G.add_edge(parent, str(child))
        return G


# --- Driver Code ---
game = GameTree()
game.best_path = {}
optimal_value = game.minimax('A', True)

# Build graph
G = game.build_graph()

# Layered positions for a clean tree layout
pos = {
    'A': (0, 3),
    'B': (-2, 2), 'C': (2, 2),
    'D': (-3, 1), 'E': (-1, 1), 'F': (1, 1), 'G': (3, 1),
    '8': (-3.5, 0), '2': (-2.5, 0),
    '7': (-1.5, 0), '5': (-0.5, 0),
    '9': (0.5, 0), '1': (1.5, 0),
    '4': (2.5, 0), '6': (3.5, 0)
}

# Determine edges
all_edges = list(G.edges())
optimal_edges = []
node = 'A'
while node in game.best_path:
    child = game.best_path[node]
    optimal_edges.append((node, str(child)))
    node = str(child)
```

```
# Draw graph
nx.draw(G, pos, with_labels=True, node_size=2000, node_color="#90CAF9",
        font_size=10, font_weight="bold", edgecolors="black")

# Highlight optimal path
nx.draw_networkx_edges(G, pos, edgelist=optimal_edges, edge_color="red", width=3)

# Show labels for leaf values
leaf_labels = {str(v): str(v) for k, vals in game.tree.items() if isinstance(vals[0], int) for v in vals}
nx.draw_networkx_labels(G, pos, labels=leaf_labels, font_color="black")

# Title
plt.title(f"Minimax Game Tree (Optimal Value at Root = {optimal_value})", fontsize=12, fontweight="bold")
plt.axis('off')
plt.show()
```



3. Learning outcomes (What I have learnt):

- Learners will be able to explain how the Minimax algorithm helps two-player (adversarial) games make optimal decisions by minimizing possible losses.
- Learners will understand the roles of maximizing and minimizing players and how they alternately choose the best or worst possible outcomes at each level.
- Learners will develop the ability to implement recursive logic to evaluate game trees, compute optimal values, and trace decision paths.
- Learners will be able to construct and visualize a game tree using Python libraries (like NetworkX and Matplotlib) and identify the optimal path visually.
- Learners will gain insight into the computational complexity of Minimax and understand why optimization techniques like **Alpha-Beta Pruning** are needed for larger trees.