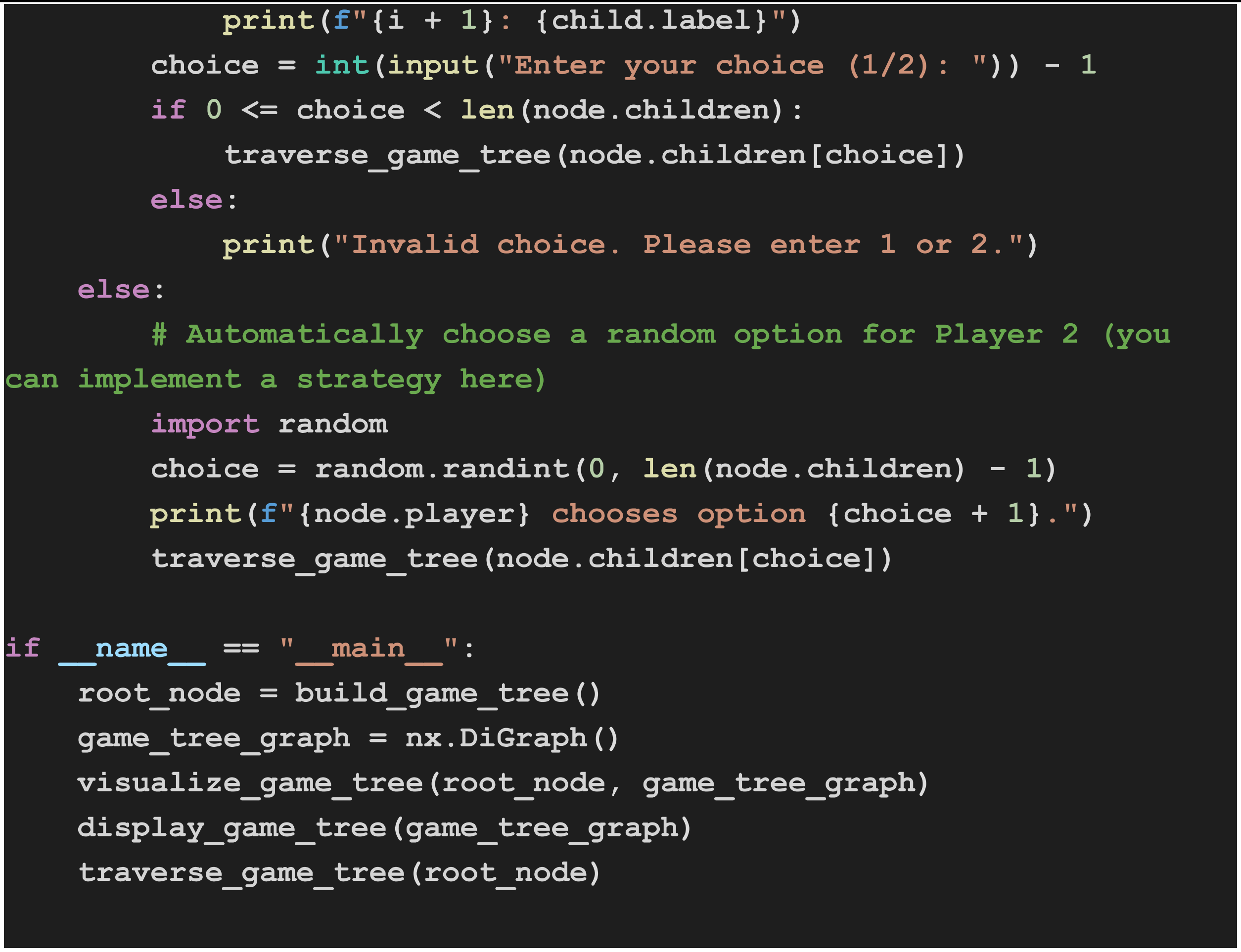
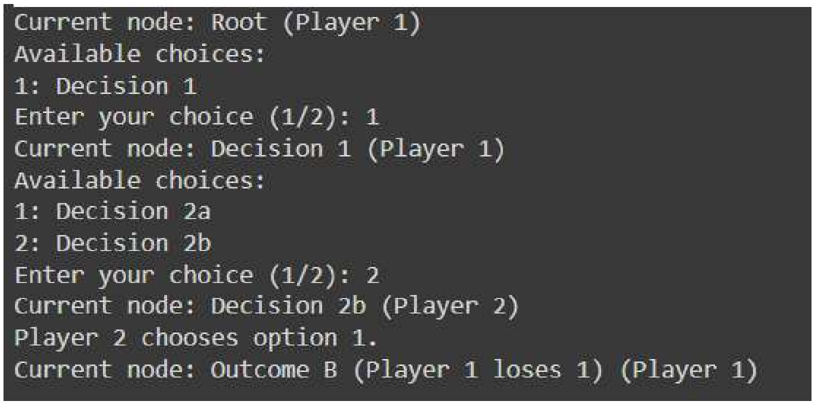
Program :

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

class GameNode:
    def __init__(self, player, label):
        self.player = player
        self.label = label
        self.children = []

    def add_child(self, child_node):
        self.children.append(child_node)

def build_game_tree():
    root = GameNode("Player 1", "Root")

    decision_node_1 = GameNode("Player 1", "Decision 1")
    root.add_child(decision_node_1)

    decision_node_2a = GameNode("Player 2", "Decision 2a")
    decision_node_2b = GameNode("Player 2", "Decision 2b")
    decision_node_1.add_child(decision_node_2a)
    decision_node_1.add_child(decision_node_2b)

    terminal_node_1a = GameNode("Player 1", "Outcome A (Player 1 wins 3)")
    terminal_node_1b = GameNode("Player 1", "Outcome B (Player 1 loses 1)")
    terminal_node_2a = GameNode("Player 2", "Outcome A (Player 2 loses 3)")
    terminal_node_2b = GameNode("Player 2", "Outcome B (Player 2 wins 1)")

    decision_node_2a.add_child(terminal_node_1a)
    decision_node_2a.add_child(terminal_node_2a)
    decision_node_2b.add_child(terminal_node_1b)
    decision_node_2b.add_child(terminal_node_2b)

    return root

def visualize_game_tree(node, graph, parent=None):
    graph.add_node(node.label, player=node.player)
    if parent is not None:
        graph.add_edge(parent.label, node.label)
    for child in node.children:
        visualize_game_tree(child, graph, node)

def display_game_tree(graph):
    pos = nx.spring_layout(graph)
    labels = {node: f"{node}\n({graph.nodes[node]['player']})" for node in graph.nodes}
    nx.draw(graph, pos, with_labels=True, labels=labels, node_size=800, node_color="lightblue", font_size=5)
    plt.title("Game Tree")
    plt.show()

def traverse_game_tree(node):
    print(f"Current node: {node.label} ({node.player})")
    if not node.children:
        return  # Reached a terminal node

    if node.player == "Player 1":
        print("Available choices:")
        for i, child in enumerate(node.children):
```



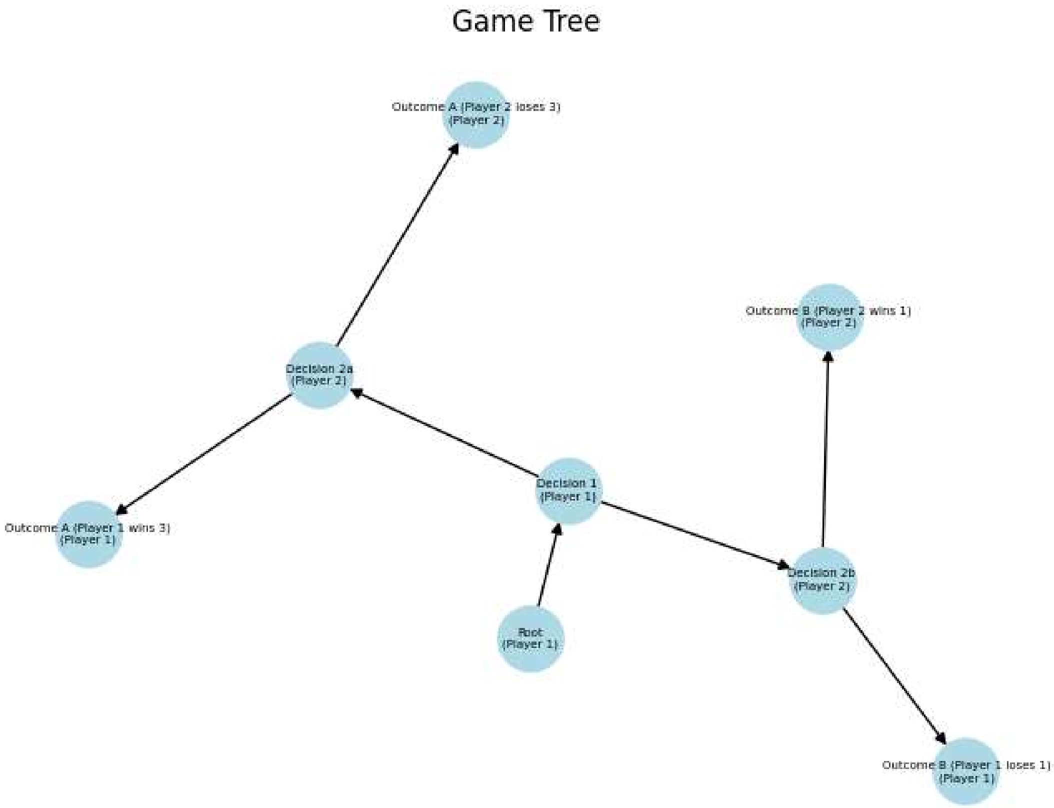
```
            print(f"{i + 1}: {child.label}")
        choice = int(input("Enter your choice (1/2): ")) - 1
        if 0 <= choice < len(node.children):
            traverse_game_tree(node.children[choice])
        else:
            print("Invalid choice. Please enter 1 or 2.")
    else:
        # Automatically choose a random option for Player 2 (you can implement a strategy here)
        import random
        choice = random.randint(0, len(node.children) - 1)
        print(f"{node.player} chooses option {choice + 1}.")
        traverse_game_tree(node.children[choice])

if __name__ == "__main__":
    root_node = build_game_tree()
    game_tree_graph = nx.DiGraph()
    visualize_game_tree(root_node, game_tree_graph)
    display_game_tree(game_tree_graph)
    traverse_game_tree(root_node)
```

Output



```
Current node: Root (Player 1)
Available choices:
1: Decision 1
Enter your choice (1/2): 1
Current node: Decision 1 (Player 1)
Available choices:
1: Decision 2a
2: Decision 2b
Enter your choice (1/2): 2
Current node: Decision 2b (Player 2)
Player 2 chooses option 1.
Current node: Outcome B (Player 1 loses 1) (Player 1)
```

Game Tree


Game Tree
Outcome A (Player 2 loses 3)
(Player 2)
Outcome B (Player 2 wins 1)
(Player 2)
Decision 2a
(Player 2)
Decision 1
(Player 1)
Outcome A (Player 1 wins 3)
(Player 1)
Decision 2b
(Player 2)
Root
(Player 1)
Outcome B (Player 1 loses 1)
(Player 1)