Nim Minimax Algorithm Code

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

def is_terminal(state):
    return all(h == 0 for h in state)

def get_moves(state):
    moves = []
    for i, heap in enumerate(state):
        if heap > 0:
            for remove in range(1, heap + 1):
                new_state = list(state)
                new_state[i] -= remove
                moves.append((state, tuple(new_state)))
    return moves

def minimax(state, maximizing_player=True, graph=None, parent=None, best_path=None):
    """
    Minimax algorithm for Nim with flat printing of moves.
    """
    if graph is None:
        graph = nx.DiGraph()
    if best_path is None:
        best_path = {}

    node_label = str(state)
    if parent is not None:
        graph.add_edge(str(parent), node_label)

    if is_terminal(state):
        value = -1 if maximizing_player else 1
        print(f"Terminal state {state} → {value}")
        graph.nodes[node_label]["label"] = f"{state}\nVal={value}"
        return value, graph, best_path

    if maximizing_player:
        print(f"MAX turn at {state}")
        value = float('-inf')
        best_child = None
        for move in get_moves(state):
            print(f"  Move: {move[0]} → {move[1]}")
            child_val, graph, best_path = minimax(move[1], False, graph, state, best_path)
            if child_val > value:
                value = child_val
                best_child = move[1]
        graph.nodes[node_label]["label"] = f"{state}\nMAX={value}"
        if best_child:
            best_path[state] = best_child
        print(f"Best value for MAX at {state} = {value}")
        return value, graph, best_path
    else:
        print(f"MIN turn at {state}")
        value = float('inf')
        best_child = None
        for move in get_moves(state):
            print(f"  Move: {move[0]} → {move[1]}")
            child_val, graph, best_path = minimax(move[1], True, graph, state, best_path)
            if child_val < value:
                value = child_val
                best_child = move[1]
        graph.nodes[node_label]["label"] = f"{state}\nMIN={value}"
        if best_child:
            best_path[state] = best_child
        print(f"Best value for MIN at {state} = {value}")
        return value, graph, best_path

def draw_tree(graph, best_path, root):
    """Draw the minimax game tree with best path highlighted."""
    pos = nx.spring_layout(graph, seed=42)
    labels = nx.get_node_attributes(graph, "label")

    # Collect best-path edges
    path_edges = []
    current = root
    while current in best_path:
        nxt = best_path[current]
        path_edges.append((str(current), str(nxt)))
        current = nxt

    plt.figure(figsize=(12, 8))
    nx.draw(graph, pos, with_labels=False, node_size=2000, node_color="lightblue", edge_color="gray")
    nx.draw_networkx_labels(graph, pos, labels, font_size=8)
    nx.draw_networkx_edges(graph, pos, edgelist=path_edges, edge_color="red", width=2)
    plt.title("Nim Minimax Tree with Best Path Highlighted")
    plt.show()


# -----------------------------
# Example run
# -----------------------------
initial_state = (1,2)   # small example

val, graph, best_path = minimax(initial_state, True)
draw_tree(graph, best_path, initial_state)

print(f"\nFinal minimax value from {initial_state} = {val}")
print(f"Best path (state → next optimal state): {best_path}")
```