```
import math

def alpha_beta_pruning(node, depth, alpha, beta, maximizing_player, evaluate, g
    """
    Perform Alpha-Beta Pruning to find the optimal value of a game tree.

    Args:
        node: The current node (state) of the game tree.
        depth: The maximum depth to explore.
        alpha: The alpha value (initially -infinity).
        beta: The beta value (initially +infinity).
        maximizing_player: Boolean indicating if the current player is maximizi
        evaluate: Function to evaluate the value of a terminal state.
        get_children: Function to generate child nodes from a given node.

    Returns:
        The optimal value for the current node.
    """
    # Base case: terminal node or maximum depth reached
    if depth == 0 or not get_children(node):
        return evaluate(node)

    if maximizing_player:
        max_eval = -math.inf
        for child in get_children(node):
            eval = alpha_beta_pruning(child, depth - 1, alpha, beta, False, eva
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                break  # Beta cut-off
        return max_eval
    else:
        min_eval = math.inf
        for child in get_children(node):
            eval = alpha_beta_pruning(child, depth - 1, alpha, beta, True, eval
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            if beta <= alpha:
                break  # Alpha cut-off
        return min_eval

# Example usage
if __name__ == "__main__":
    # Example game tree structure
    game_tree = {
        "A": ["B", "C", "D"],
        "B": ["E", "F"],
```

```
        "C": ["G", "H"],
        "D": ["I", "J"],
        "E": [],  # Leaf node
        "F": [],  # Leaf node
        "G": [],  # Leaf node
        "H": [],  # Leaf node
        "I": [],  # Leaf node
        "J": []   # Leaf node
    }

    # Leaf node values (evaluation function outputs)
    leaf_values = {
        "E": 3,
        "F": 5,
        "G": 6,
        "H": 9,
        "I": 1,
        "J": 2
    }

    # Define evaluation and child generation functions
    def evaluate(node):
        return leaf_values.get(node, 0)  # Return leaf value, or 0 for non-leaf

    def get_children(node):
        return game_tree.get(node, [])

    # Perform Alpha-Beta Pruning
    result = alpha_beta_pruning("A", depth=3, alpha=-math.inf, beta=math.inf,
                                maximizing_player=True, evaluate=evaluate, get_

    print(f"The optimal value for the root is: {result}")
```

```
The optimal value for the root is: 6
```