Implement Alpha-Beta Pruning.

```
def alpha_beta_search(state):
    """
    Returns the best action and its evaluated value using Alpha-Beta
pruning.
    """
    value, move = max_value(state, float('-inf'), float('inf'))
    return value, move


def max_value(state, alpha, beta):
    if terminal_test(state):
        return utility(state), None

    value = float('-inf')
    best_move = None
    for action in actions(state):
        v, _ = min_value(result(state, action), alpha, beta)
        if v > value:
            value = v
            best_move = action
        if value >= beta:
            return value, best_move  # Beta cutoff
        alpha = max(alpha, value)

    return value, best_move


def min_value(state, alpha, beta):
    if terminal_test(state):
        return utility(state), None

    value = float('inf')
    best_move = None
    for action in actions(state):
        v, _ = max_value(result(state, action), alpha, beta)
        if v < value:
            value = v
            best_move = action
        if value <= alpha:
            return value, best_move  # Alpha cutoff
        beta = min(beta, value)

    return value, best_move


# --------------------------
# Example Toy Game Functions
```

```
# --------------------------

def actions(state):
    """Return all possible actions from this state."""
    return state.get('actions', [])


def result(state, action):
    """Return the next state after performing action."""
    return action['next']


def terminal_test(state):
    """Check if this is a terminal (leaf) state."""
    return state.get('terminal', False)


def utility(state):
    """Return the utility (score) of a terminal state."""
    return state.get('utility', 0)


# --------------------------
# Example Game Tree
# --------------------------
if __name__ == "__main__":
    # Leaf nodes with known utilities
    leaf1 = {'terminal': True, 'utility': 3}
    leaf2 = {'terminal': True, 'utility': 5}
    leaf3 = {'terminal': True, 'utility': 6}
    leaf4 = {'terminal': True, 'utility': 9}

    # Intermediate nodes
    B = {'actions': [{'next': leaf1}, {'next': leaf2}]}  # MIN layer
    C = {'actions': [{'next': leaf3}, {'next': leaf4}]}  # MIN layer
    root = {'actions': [{'next': B}, {'next': C}]}       # MAX layer

    best_value, best_move = alpha_beta_search(root)
    print("Best move value (evaluated utility):", best_value)
```

Output:

```
Best move value (evaluated utility): 6
```