

Implement Alpha-Beta Pruning.

alpha_beta.py

from typing import Any, Dict, List, Optional, Tuple

State = Dict[str, Any]

Action = Dict[str, Any]

def alpha_beta_search(state: State) -> Tuple[float, Optional[Action]]:

"""

Returns the best action and its evaluated value using Alpha-Beta pruning.

Returns (value, best_action). best_action is None for terminal states.

"""

value, move = max_value(state, float('-inf'), float('inf'))

return value, move

def max_value(state: State, alpha: float, beta: float) -> Tuple[float, Optional[Action]]:

if terminal_test(state):

return utility(state), None

value = float('-inf')

best_move: Optional[Action] = 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:

Beta cutoff

return value, best_move

alpha = max(alpha, value)

return value, best_move

def min_value(state: State, alpha: float, beta: float) -> Tuple[float, Optional[Action]]:

```

if terminal_test(state):
    return utility(state), None

value = float('inf')
best_move: Optional[Action] = 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:
        # Alpha cutoff
        return value, best_move
    beta = min(beta, value)

return value, best_move

```

```

# -----

```

```

# Example Toy Game Functions

```

```

# -----

```

```

def actions(state: State) -> List[Action]:
    """Return all possible actions from this state."""
    return state.get('actions', [])

```

```

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

```

```

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

```

```

def utility(state: State) -> float:
    """Return the utility (score) of a terminal state."""

```

```

return float(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 (MIN layer). Add a small 'name' in actions for readability.
    B = {'actions': [
        {'name': 'B->leaf1', 'next': leaf1},
        {'name': 'B->leaf2', 'next': leaf2}
    ]}

    C = {'actions': [
        {'name': 'C->leaf3', 'next': leaf3},
        {'name': 'C->leaf4', 'next': leaf4}
    ]}

    # Root (MAX layer)
    root = {'actions': [
        {'name': 'Root->B', 'next': B},
        {'name': 'Root->C', 'next': C}
    ]}

    best_value, best_move = alpha_beta_search(root)

    print("Best move value (evaluated utility):", best_value)
    if best_move is not None:
        # Print the action name if available, otherwise print the whole action dict
        print("Best move chosen:", best_move.get('name', best_move))
    else:
        print("No move (terminal state).")

```

Output:

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