625 Programming Assignment 1

Unnati Eramangalath

UIN: 930001393

PART 2: GAME PLAYING

AIM: implement simple min-max and alpha-beta pruning algorithms for a game-tree search. The game will be given as a simple LISP list representing a game tree. Leaves will represent end-game state. Values at the leaves will represent the utility value.
Write two functions min-max and alpha-beta to conduct game tree search.
• The function call should be as follows:
Example: (min-max '(1 (5 7) 4)) (min-max tree)
Example: (alpha-beta '(1 (5 7) 4)) (alpha-beta tree)
• The output should be as follows: – min-max: solution path (a series of numbers indicating the path taken). For example, given a tree '(1 (5 7) 4), the solution path will be 2, 1. Max node root will select the second child (subtree (5 7)), which is a Min node, and it will choose the first child (leaf 5). – alpha-beta: on top of the solution path, you should also indicate the MIN and MAX cuts.
(1) Indicate whether it is a MIN cut or a MAX cut.
(2) Output the local context where the cut is made: e.g., MIN cut after (2 3) in subtree (1 (2 3) 4).

MINIMAX DECISION:

In game theory, **minimax** is a decision rule used to minimize the worst-case potential loss; in other words, a player considers all of the best opponent responses to his strategies, and selects the strategy such that the opponent's best strategy gives a payoff as large as possible.

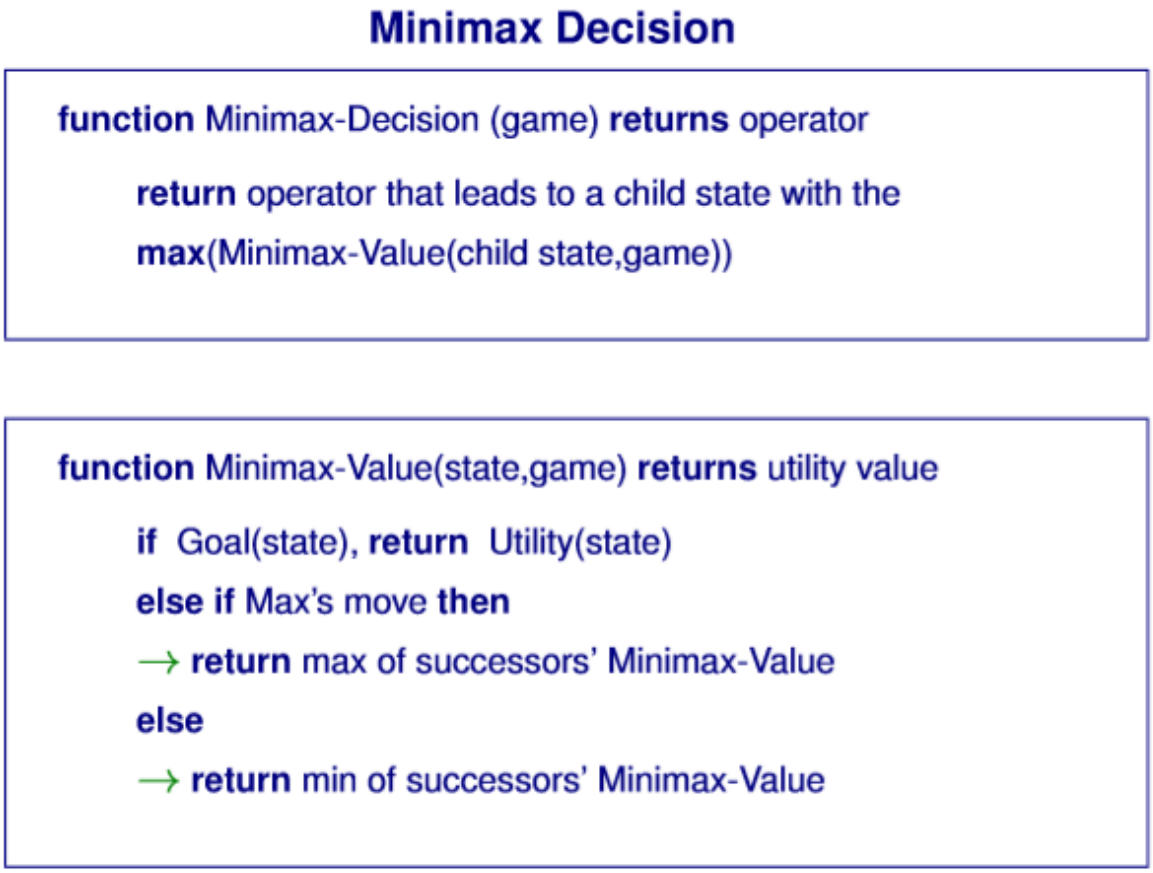
Here, I have implemented 2 functions: Minimax-Decision and Minimax-Value to get the required path.

Results:

```
* (Minimax-Decision '((4 (7 9 8) 8) (((3 6 4) 2 6) ((9 2 9) 4 7 (6 4 5)))) )
 PATH TAKEN : 2 PATH TAKEN : 1 PATH TAKEN : 3
NIL
* (Minimax-Decision '(((1 4) (3 (5 2 8 0) 7 (5 7 1)) (8 3)) (((3 6 4) 2 (9 3 0)) ((8 1 9) 8 (3 4 )))) )
 PATH TAKEN : 1 PATH TAKEN : 1 PATH TAKEN : 2
NIL
* (Minimax-Decision '(5 (((4 7 -2) 7) 6)) )
 PATH TAKEN : 2 PATH TAKEN : 2
NIL
* (Minimax-Decision '((8 (7 9 8) 4) (((3 6 4) 2 1) ((6 2 9) 4 7 (6 4 5)))) )
 PATH TAKEN : 1 PATH TAKEN : 3
NIL
* (Minimax-Decision '(((1 (4 7)) (3 ((5 2) (2 8 9) 0 -2) 7 (5 7 1)) (8 3)) (((8 (9 3 2) 5) 2 (9 (3 2) 0)) ((3 1 9) 8 (3 4 ))))
)
 PATH TAKEN : 2 PATH TAKEN : 1 PATH TAKEN : 1 PATH TAKEN : 3
```

Analysis:

As observed from the results above, I have implemented 2 functions -Minimax-Decision and Minimax-Value to get the path parsed through to perform minimax operation, following the below mentioned algorithm:



Minimax Strategy:
• generate the whole tree, and apply util function to the leaves
• go back upward assigning utility value to each node
• at MIN node, assign min(successors' utility)
• at MAX node, assign max(successors' utility)
• assumption: the opponent acts optimally

Minimax is complete if the game tree is finite. It is optimal if the opponent is optimal.
Time Complexity – O(b^m)
Space Complexity – O(bm) (only when utility function values of all nodes are known)
(where b =Branching factor, m= max depth).

ALPHA-BETA PRUNING

Alpha–beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. It is an adversarial search algorithm used commonly for machine playing of two-player games. It stops evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.

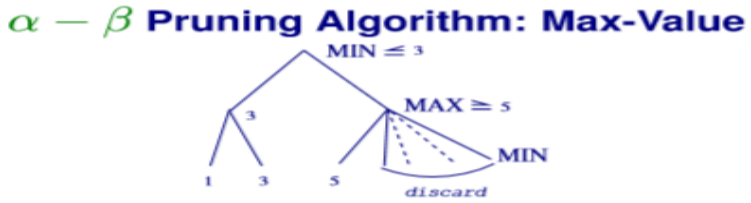
Results:

```
* (Minimax-DecisionAB '((4 (7 9 8) 8) (((3 6 4) 2 6) ((9 2 9) 4 7 (6 4 5)))) )

  MAX CUT AFTER = 7
  MIN CUT AFTER =3
  MIN CUT AFTER =2
  MAX CUT AFTER = 7
  MAX CUT AFTER = 7
  MAX CUT AFTER = 7 PATH TAKEN WITH ALPHA BETA PRUNING : 2
  MAX CUT AFTER = 7 PATH TAKEN WITH ALPHA BETA PRUNING : 1 PATH TAKEN WITH ALPHA BETA PRUNING : 3
NIL
* (Minimax-DecisionAB '(((1 4) (3 (5 2 8 0) 7 (5 7 1)) (8 3)) (((3 6 4) 2 (9 3 0)) ((8 1 9) 8 (3 4 )))) )

  MIN CUT AFTER =2
  MAX CUT AFTER = 7
  MAX CUT AFTER = 8
  MIN CUT AFTER =3
  MIN CUT AFTER =3
  MIN CUT AFTER =((3 6 4) 2 (9 3 0))
  MIN CUT AFTER =2
  MAX CUT AFTER = 7
  MAX CUT AFTER = 8 PATH TAKEN WITH ALPHA BETA PRUNING : 1
  MIN CUT AFTER =2
  MAX CUT AFTER = 7
  MAX CUT AFTER = 8 PATH TAKEN WITH ALPHA BETA PRUNING : 1 PATH TAKEN WITH ALPHA BETA PRUNING : 2
NIL
* (Minimax-DecisionAB '(5 (((4 7 -2) 7) 6)) )

  MIN CUT AFTER =4 PATH TAKEN WITH ALPHA BETA PRUNING : 2 PATH TAKEN WITH ALPHA BETA PRUNING : 2
NIL
* (Minimax-DecisionAB '((8 (7 9 8) 4) (((3 6 4) 2 1) ((6 2 9) 4 7 (6 4 5)))) )

  MAX CUT AFTER = 9
  MIN CUT AFTER =3
  MIN CUT AFTER =((3 6 4) 2 1)
  MAX CUT AFTER = 9 PATH TAKEN WITH ALPHA BETA PRUNING : 1
  MAX CUT AFTER = 9 PATH TAKEN WITH ALPHA BETA PRUNING : 3
NIL
```

```
* (Minimax-DecisionAB '(((1 (4 7)) (3 ((5 2) (2 8 9) 0 -2) 7 (5 7 1)) (8 3)) (((8 (9 3 2) 5) 2 (9 (3 2) 0)) ((3 1 9) 8 (3 4 ))))
)

  MAX CUT AFTER = 5
  MAX CUT AFTER = 8
  MIN CUT AFTER =0
  MAX CUT AFTER = 7
  MAX CUT AFTER = 8
  MAX CUT AFTER = 9
  MIN CUT AFTER =(3 2)
  MIN CUT AFTER =3
  MAX CUT AFTER = 8
  MAX CUT AFTER = 5
  MAX CUT AFTER = 8
  MIN CUT AFTER =0
  MAX CUT AFTER = 7
  MAX CUT AFTER = 8
  MAX CUT AFTER = 9
  MIN CUT AFTER =(3 2)
  MAX CUT AFTER = 8 PATH TAKEN WITH ALPHA BETA PRUNING : 2
  MAX CUT AFTER = 9
  MIN CUT AFTER =(3 2)
  MAX CUT AFTER = 8
  MAX CUT AFTER = 9
  MIN CUT AFTER =(3 2) PATH TAKEN WITH ALPHA BETA PRUNING : 1
  MAX CUT AFTER = 9
  MIN CUT AFTER =(3 2)
  MAX CUT AFTER = 9 PATH TAKEN WITH ALPHA BETA PRUNING : 1
  MAX CUT AFTER = 9 PATH TAKEN WITH ALPHA BETA PRUNING : 3
```

Analysis

As observed from the results above, the implemented alpha-Beta pruning algorithm prints the path taken for the given 5 cases along with their min-cut and max-cut values, following the given algorithm:



function Max-Value (state, game, α, β) **return** utility value
α: best MAX on path to *state* ; β: best MIN on path to *state*
if Cutoff(state) **then return** Utility(state)
$v \leftarrow -\infty$
for each *s* in Successor(state) **do**
- $v \leftarrow$ Max(α, Min-Value(s,game,α,β))
- **if** $v \geq \beta$ **then return** v /* CUT!! */
- $\alpha \leftarrow$Max(α, v)

end
return v

$\alpha - \beta$ Pruning Algorithm: Min-Value

function Min-Value (state, game, α, β) **return** utility value

α: best MAX on path to *state* ; β: best MIN on path to *state*

if Cutoff(state) **then return** Utility (state)

$v \leftarrow \infty$

for each *s* in Successor(state) **do**

- $v \leftarrow$ Min(β, Max-Value(s,game,α,β))
- **if** $v \leq \alpha$ **then return** v /* **CUT!!** */
- $\beta \leftarrow$Min(β,v)

end

return v

At a MAX node:

1. Only α is updated with the MAX of successors.
2. Cut is done by checking if returned v ≥ β.
3. If all fails, MAX(v of succesors) is returned.

At a MIN node:

1. Only β is updated with the MIN of successors.
2. Cut is done by checking if returned v ≤ α.
3. If all fails, MIN(v of succesors) is returned.

With perfect ordering**, time complexity** = O(b^(m/2)) →doubles depth of search
(where b = Branching factor, m= depth)

In the worst case, where there is no node to be pruned, the full tree will be examined (or the complete tree up to the cutoff at a depth m). In the best-case scenario, each node will examine 2b-1 grandchildren to decide on its value. But in the worst-case scenario, the node will examine b^2 grandchildren. Logically this means that the overall algorithm examined O(b^(m/2)) nodes, the same as a worst-case algorithm whose cutoff is half of m.