**Problem 1**

**Max: [4308: 30 Points, 5360: 25 Points]**

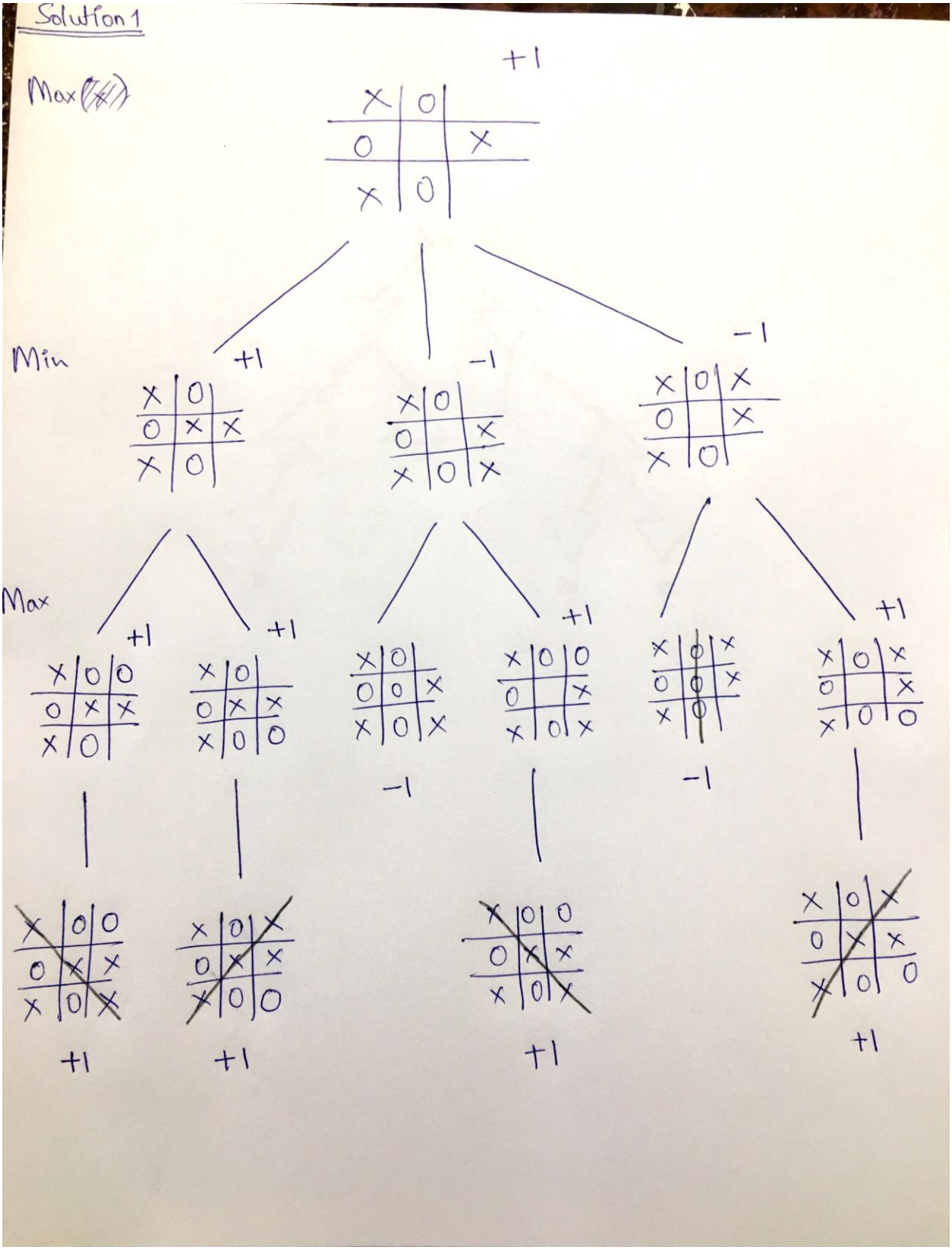
| | | |
|---|---|---|
| X | O | |
| O | | X |
| X | O | |

Figure 1. A tic-tac-toe board state.

Consider the tic-tac-toe board state shown in Figure 1. Draw the full minimax search tree starting from this state, and ending in terminal nodes. Show the utility value for each terminal and non-terminal node. Also show which move the Minimax algorithm decides to play. Utility values are +1 if X wins, 0 for a tie, and -1 if O wins. Assume that X makes the next move (X is the MAX player).

## Solution:

Solution 1



**Problem 2**

**Max: [4308: 25 Points, 5360: 20 Points]**

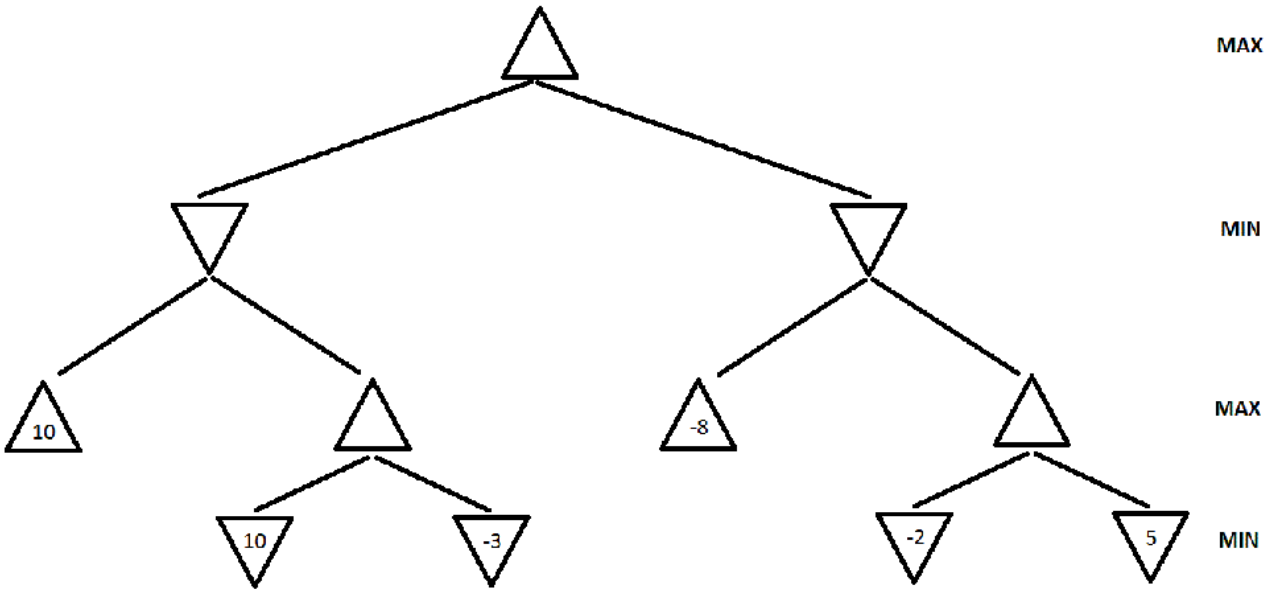
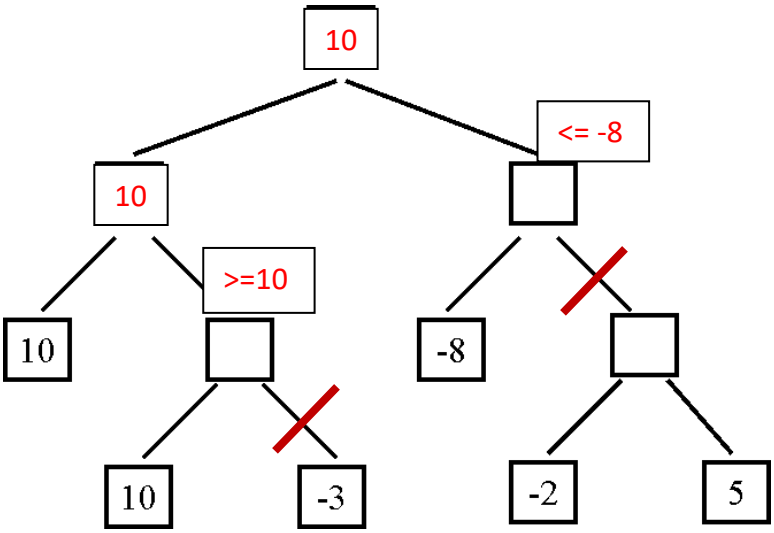



Figure 2. A game search tree.

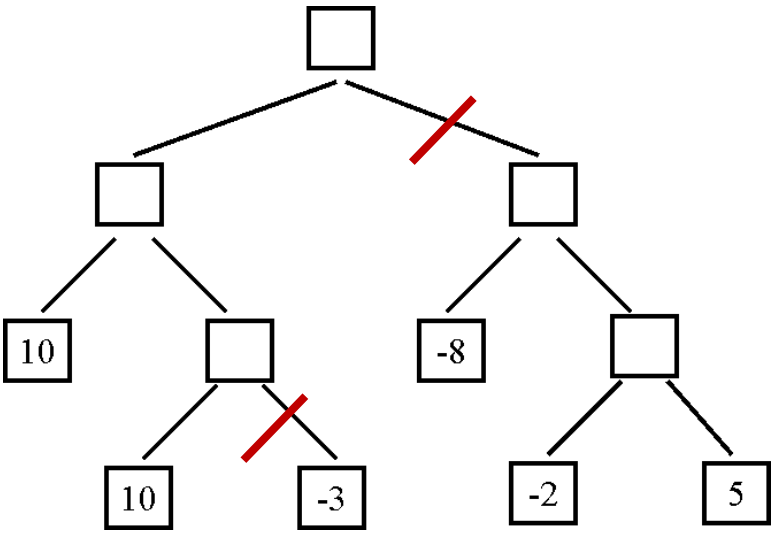
a. (4308: 20 points, 5360: 15 points) In the game search tree of Figure 2, indicate what nodes will be pruned using alpha-beta search, and what the estimated utility values are for the rest of the nodes. Assume that, when given a choice, alpha-beta search expands nodes in a left-to-right order. Also, assume the MAX player plays first. Finally incidcate which action the Minmax algorithm will pick to exectute.

## Solution:



b. (4308: 5 points, 5360: 5 points) This question is also on the game search tree of Figure 2. Suppose we are given some additional knowledge about the game: the maximum utility value is 10, i.e., it is not mathematically possible for the MAX player to get an outcome greater than 10. How can this knowledge be used to further improve the efficiency of alpha-beta search? Indicate the nodes that will be pruned using this improvement. Again, assume that, when given a choice, alpha-beta search expands nodes in a left-to-right order, and that the MAX player plays first.

## Solution:



**Problem 3**

**Max: [4308: 20 Points, 5360: 15 Points]**

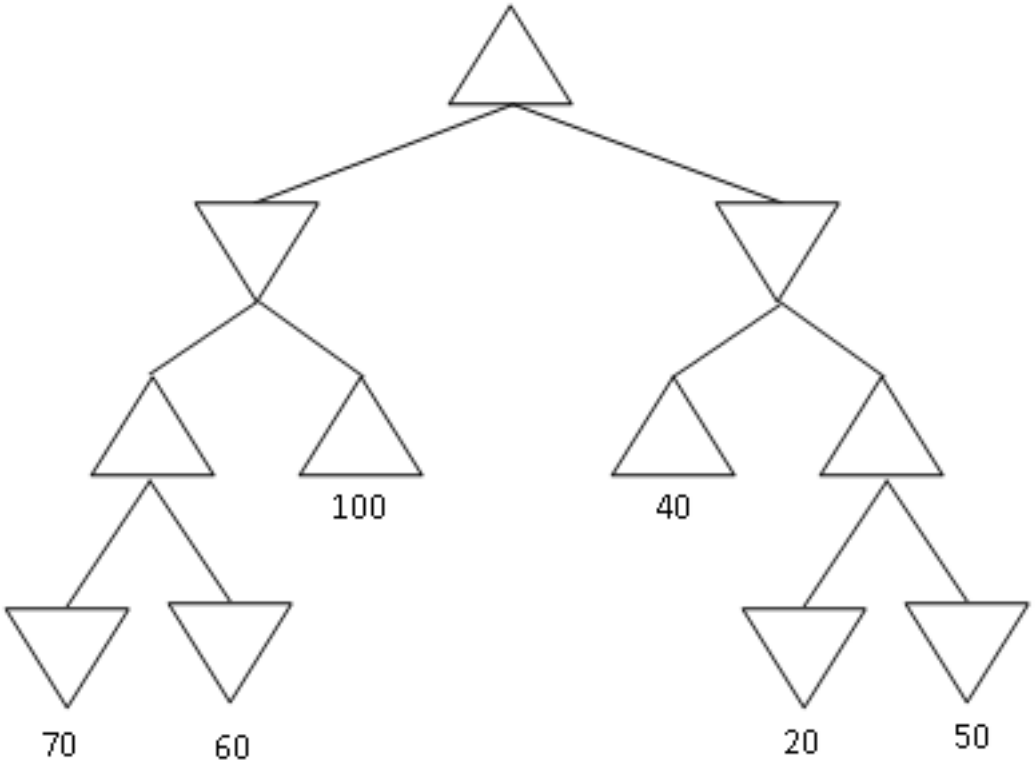



Figure 3: Yet another game search tree

Consider the MINIMAX tree above. Suppose that we are the MAX player, and we follow the MINIMAX algorithm to play a full game against an opponent. However, **we do not know what algorithm the opponent uses**.

Under these conditions, what is the best possible outcome of playing the full game for the MAX player? What is the worst possible outcome for the MAX player? Justify your answer.

NOTE: the question is not asking you about what MINIMAX will compute for the start node. It is asking you what is the best and worst outcome of a **complete game** under the assumptions stated above.

## Solution:

Min max is a backtracking algorithm. In order to get the best solution, we would use backtracking. From the question the max player (maximizer) is the one who gets the high score in the game. Min player is the one with less score than the maximizer.

Since I am the MaxPlayer I would get the first chance to play. Maximizer can either move to left or right.

From the values 70 and 60 the maximizer would choose 70, from 70 and 100 the minimizer would choose 70. From 20 and 50 the maximizer would choose 50, from 40 and 50 the minimizer would choose 40. The maximizer would choose 70 from the values 70 and 40.

The worst case is when the root node has the value 70.

For the best-case scenario,

From the values 70 and 60, the value 70 would be chosen. From 70 and 100, the value 100 would be chosen. From 20 and 50, the value 50 would be chosen. From the values 40 and 50 the value 50 would be chosen. From the values 100 and 50 the value 100 would be chosen.

The best case is when the root node has the value 100.

**Problem 4**

**Max: [4308: 25 Points, 5360: 20 Points]**

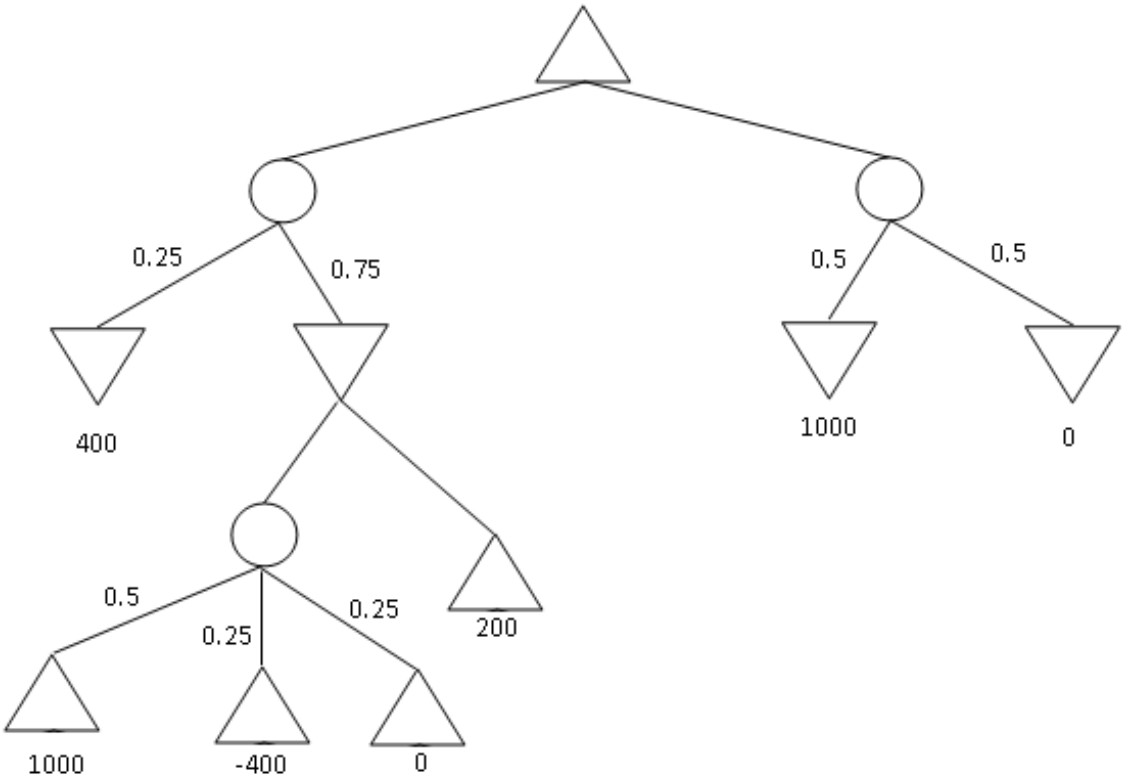
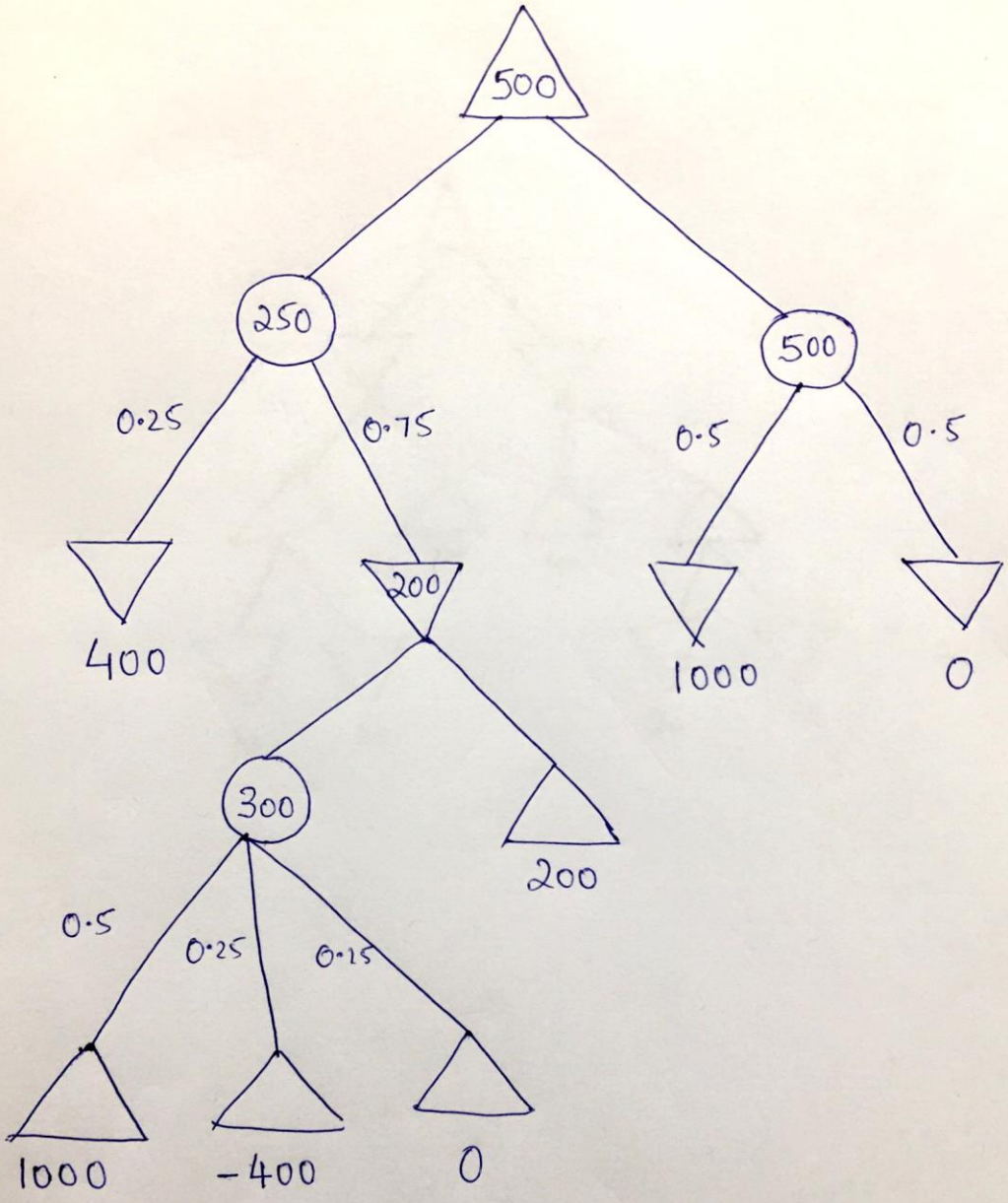



Figure 4: An Expectiminmax tree.

Find the value of every non-terminal node in the expectiminmax tree given above. Also indicate which action will be performed by the algorithm

## Solution:

Solution 4


500
250
500
0.25
0.75
0.5
0.5
400
200
1000
0
300
200
0.5
0.25
0.25
1000
-400
0

**Problem 5 (Extra Credit for 4308, Required for 5360)**

**Max: [4308: 20 Points EC, 5360: 20 Points]**

Suppose that you want to implement an algorithm tht will compete on a two-player deterministic game of perfect information. Your opponent is a supercomputer called DeepGreen. DeepGreen does not use Minimax. You are given a library function DeepGreenMove(S), that takes any state S as an argument, and returns the move that DeepGreen will choose for that state S (more precisely, DeepGreenMove (S) returns the state resulting from the opponent's move).

Write an algorithm in pseudocode (following the style of the Minimax pseudocode) that will always make an optimal decision given the knowledge we have about DeepGreen. You are free to use the library function DeepGreenMove(S) in your pseudocode. If you think regular Minimax is the answer, just state so.

# Solution:

Modifying the Minimax pseudo code.

**function** MINIMAX-DECISION(state) **returns** an action **return** arg $\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(state, a))

**function** MAX-VALUE(state) **returns** a utility value

**if** TERMINAL-TEST(state) **then return** UTILITY(state) v ← −∞

**for each** a **in** ACTIONS(state) **do**

v ← MAX(v, MIN-VALUE(RESULT(s, a))) **return** v

**function** MIN-VALUE(state) **returns** a utility value

**if** TERMINAL-TEST(state) **then return** UTILITY(state) v←∞

**for each** a **in** ACTIONS(state) **do**

v ← MIN(v, MAX-VALUE(RESULT(s, a)))

```
if v <= UTILITY (DeepGreenMove(state))
        then return MAX(v, DeepGreenMove(state))
else
```

```
        return v
```

**----------END-OF-DOCUMENT----------**