Minimax Algorithm

Ex: NO: 5

Date:

Aim:

To implement Minimax Algorithm problem using python.

Source code:

```
from math import inf as infinity
from random import choice
import platform
import time
from os import system

HUMAN = -1
COMP = +1
board = [
    [ 0, 0, 0 ],
    [ 0, 0, 0 ],
    [ 0, 0, 0 ],
]

def evaluate(state):
if wins(state, COMP):
    score = +1
```

```
elif wins (state, HUMAN):
    score = -1
else
    score = 0
return score

def wins (state, player):
    win_state = [
    [state[0][0], state[0][1], state[0][2]],
    [state[1][0], state[1][1], state[1][2]],
    [state[2][0], state[2][1], state[2][2]],
    [state[0][0], state[1][0], state[2][0]],
    [state[0][1], state[1][1], state[2][1]],
    [state[0][2], state[1][2], state[2][2]],
    [state[0][0], state[1][1], state[2][2]],
]
if [player, player, player] in win_state:
    return TRUE
```

```
else:
    return false
def game_over (state):
    return wins (state, HUMAN) or wins (state,
                                        COMP)
def empty_cells (state):
        cells = [ ]
for x, row in enumerate (state):
for y, cell in enumerate (row):
    if cell = 0:
    cells. append ( [x, y] )
return cells
def valid_move (x, y):
if [x, y] in empty_cells (board):
 return TRUE
else:
  return FALSE.
def render (state, c-choice, h-choice):
  chars = {
    -1 : h-choice,
    +1 : c-choice,
     0 : ' '
  }
```

```
        str_line = "---"
    print("|" + str_line)
    for row in state:
        for cell in row:
            symbol = chars[cell]
            print(f"| {symbol} ", end="")
        print("|" + str_line)
def ai_turn(c_choice, h_choice):
    depth = len(empty_cells(board))
    if depth == 0 or game_over(board):
        return
    clean()
```