Exp- 6 MinMax Algorithm

Team- Automata lab

RA1911031010128-VEER VISWAJEET SWAMI

RA1911031010129-VIJAY RENGARAJ R

RA1911031010130-THINAKAR R

RA1911031010131-NAMARATA MISHRA

RA1911031010138-HUSNA QASIM

MinMax ALGORITHM

Problem chosen: TIC-TAC TOE

Problem statement: Program a two-person game of Tic -Tac- Toe. The game is played on a three by three board. Each player has a marker. One player has an 'X', the other an 'O'. Players alternate turns to place their marker on the board. The first player to get three in a row either diagonally, horizontally, or vertically, wins the games. In the event all squares are taken on the board without a winner then it is a tie. The program should set up the game by asking for the names of the players. Player one should be assigned an 'X' as their marker, player two should be assigned the 'O'. After the game has been completed, the program should congratulate the winner by name. The players should then have the option to play again. If they decide to play again, then the program should keep track of the number of times each player has won and display that information at the end of each game. You may not assume that any input the user provides you is initially valid.If the information provided by the user at any stage of the program is invalid, the program should reprompt until valid information is provided.

Code & Output:

```
from math import inf

import sys, os


HUMAN = 1

COMP = -1
```

```
board = [[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]]

MSG = "Welcome to Unbeatable Tic Tac Toe.\n" \
      "Our A.I can foreseen your moves ahead.\n" \
      "Are you sure to continue ? (y/n)"



def evaluate(state):
   if wins(state, COMP):
      score = -1
   elif wins(state, HUMAN):
      score = 1
   else:
      score = 0

   return score

def empty_cells(state):
   """Extract the remainder of board"""
   cells = [] # it contains all empty cells

   # Use enumerate for easy indexing
   for i, row in enumerate(state):
      for j, col in enumerate(row):
```

```
            if state[i][j] == 0:
                cells.append([i, j])

    return cells

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]],
        [state[2][0], state[1][1], state[0][2]],
    ]

    if [player, player, player] in win_state:
        return True
    else:
        return False

def game_over(state):
    """Check game over condition"""
    return wins(state, HUMAN) or wins(state, COMP)
```

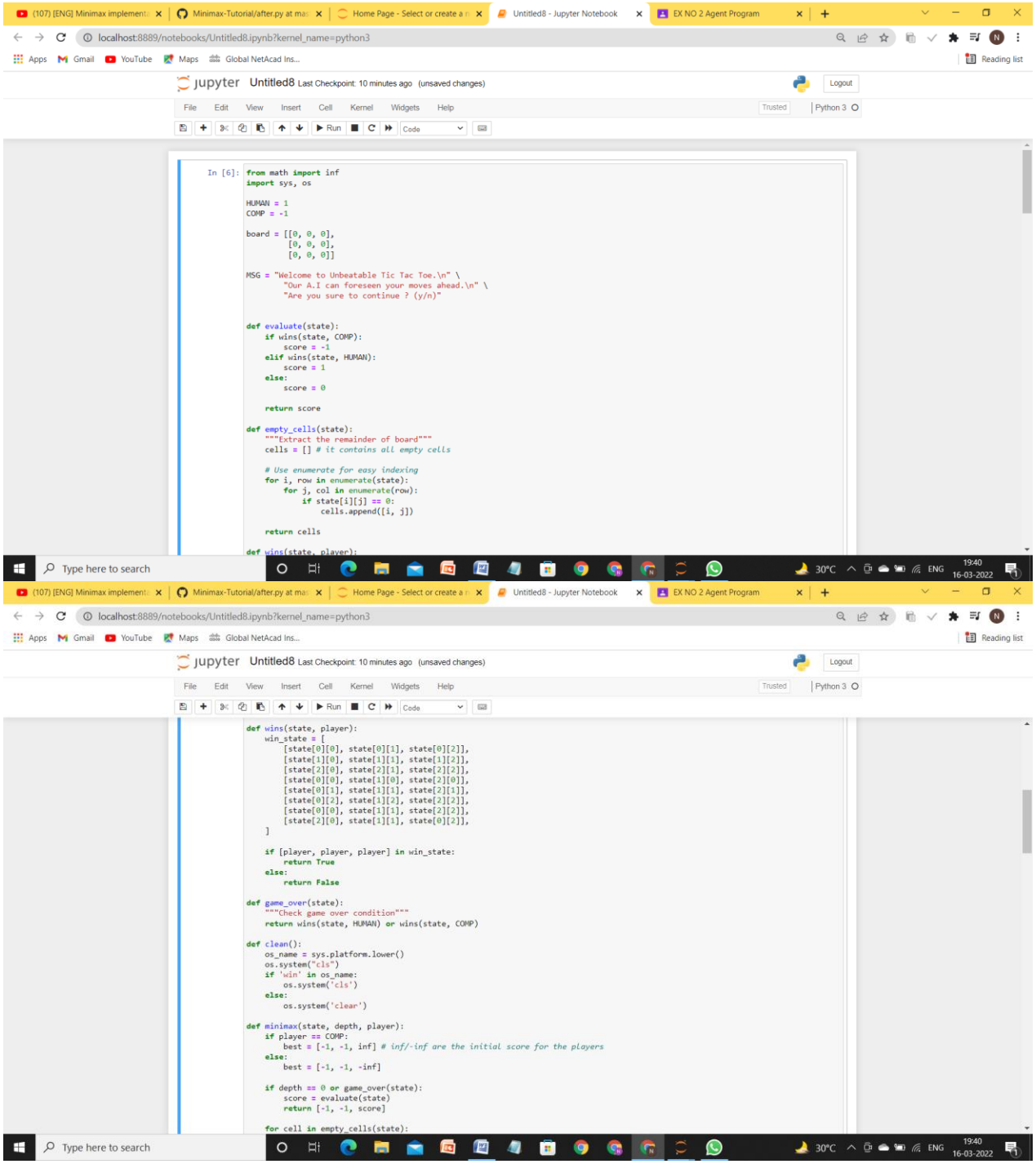
```
def clean():
    os_name = sys.platform.lower()
    os.system("cls")
    if 'win' in os_name:
        os.system('cls')
    else:
        os.system('clear')

def minimax(state, depth, player):
    if player == COMP:
        best = [-1, -1, inf] # inf/-inf are the initial score for the players
    else:
        best = [-1, -1, -inf]

    if depth == 0 or game_over(state):
        score = evaluate(state)
        return [-1, -1, score]

    for cell in empty_cells(state):
        # Fill the empty cells with the player symbols
        x, y = cell[0], cell[1]
        state[x][y] = player
        #
        score = minimax(state, depth - 1, -player)
        state[x][y] = 0
        score[0], score[1] = x, y
```

```
        if player == COMP:
            if score[2] < best[2]:
                best = score
        else:
            if score[2] > best[2]:
                best = score

    return best

def human_turn(state):
    # All possible moves
    moves = {
        1: [0, 0], 2: [0, 1], 3: [0, 2],
        4: [1, 0], 5: [1, 1], 6: [1, 2],
        7: [2, 0], 8: [2, 1], 9: [2, 2],
    }

    remain = empty_cells(state)
    isTurn = True
    print("Human Turn")
    while isTurn:
        try:
            move = int(input("Enter your move (1-9) :"))
            # When the player move is valid
            if moves.get(move) in remain:
```

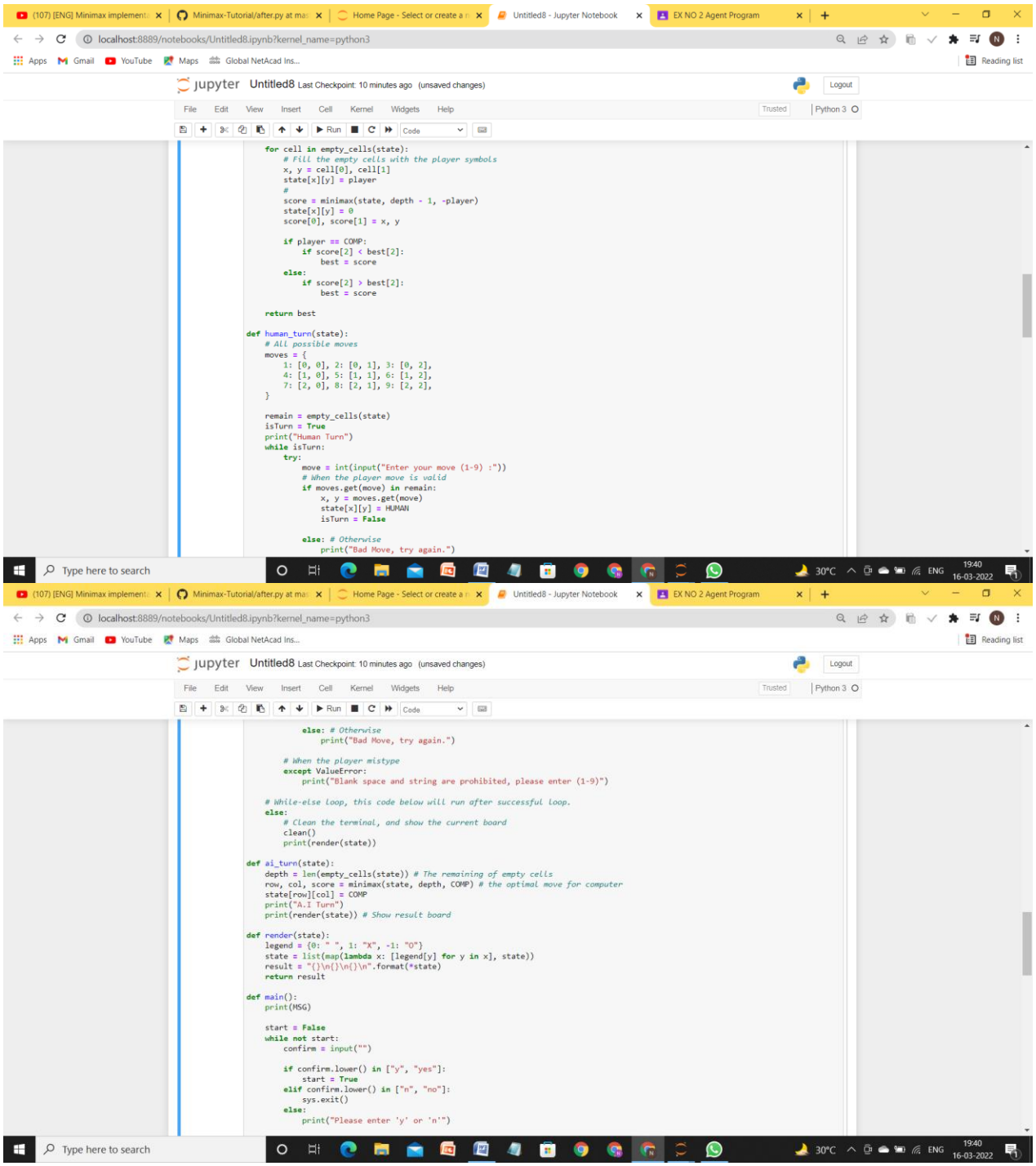
```
            x, y = moves.get(move)
            state[x][y] = HUMAN
            isTurn = False

        else: # Otherwise
            print("Bad Move, try again.")

    # When the player mistype
    except ValueError:
        print("Blank space and string are prohibited, please enter (1-9)")

# While-else loop, this code below will run after successful loop.
else:
    # Clean the terminal, and show the current board
    clean()
    print(render(state))

def ai_turn(state):
    depth = len(empty_cells(state)) # The remaining of empty cells
    row, col, score = minimax(state, depth, COMP) # the optimal move for computer
    state[row][col] = COMP
    print("A.I Turn")
    print(render(state)) # Show result board

def render(state):
    legend = {0: " ", 1: "X", -1: "O"}
```

```
    state = list(map(lambda x: [legend[y] for y in x], state))
    result = "{}\n{}\n{}\n".format(*state)
    return result

def main():
    print(MSG)

    start = False
    while not start:
        confirm = input("")

        if confirm.lower() in ["y", "yes"]:
            start = True
        elif confirm.lower() in ["n", "no"]:
            sys.exit()
        else:
            print("Please enter 'y' or 'n'")

    else:
        clean()
        print("Game is settled !\n")
        print(render(board), end="\n")

    while not wins(board, COMP) and not wins(board, HUMAN):
            human_turn(board)
            if len(empty_cells(board)) == 0: break
```

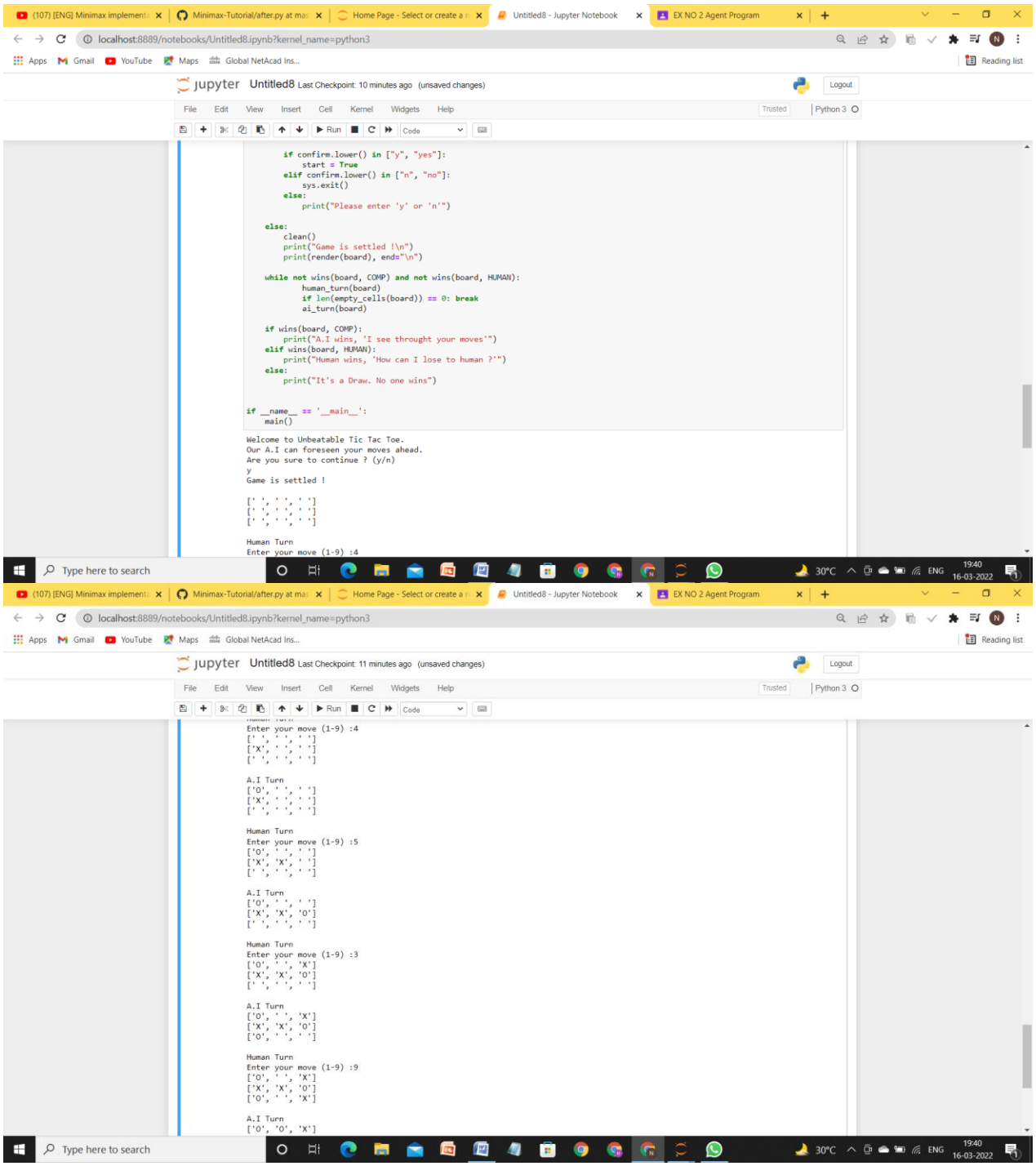
```
            ai_turn(board)

    if wins(board, COMP):
        print("A.I wins, 'I see throught your moves'")
    elif wins(board, HUMAN):
        print("Human wins, 'How can I lose to human ?'")
    else:
        print("It's a Draw. No one wins")



if __name__ == '__main__':
    main()
```



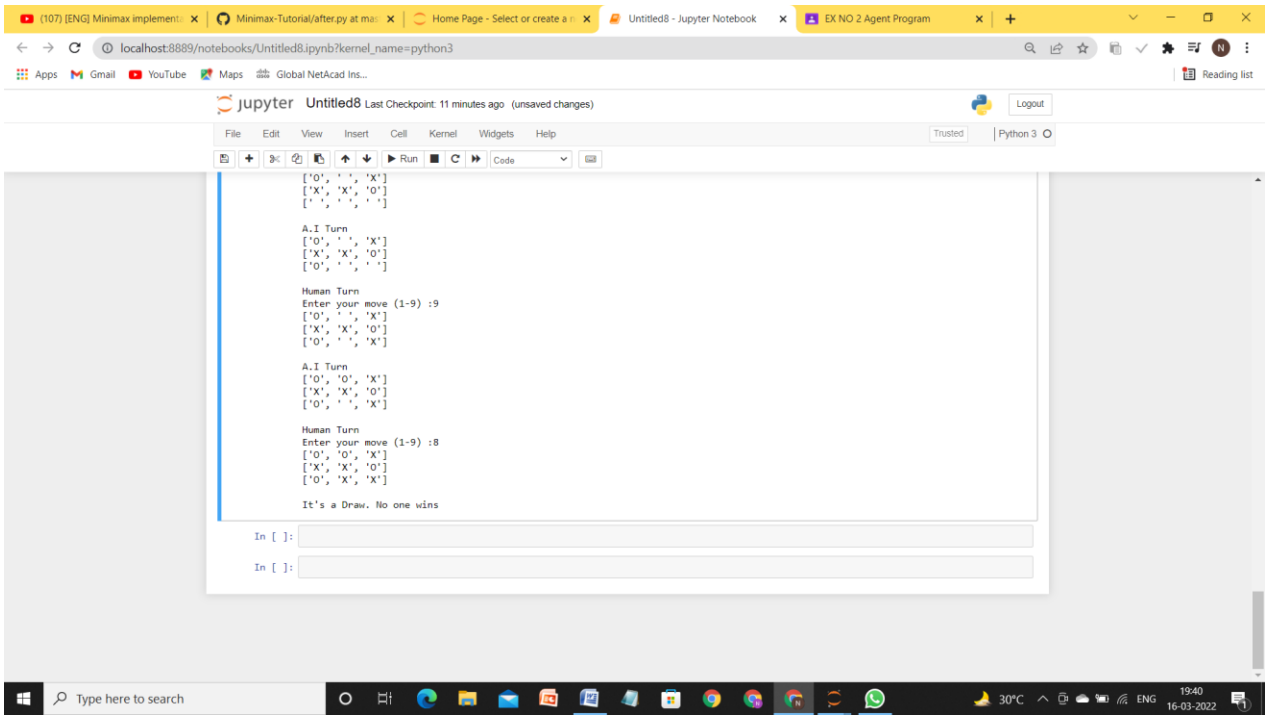
```
In [6]: from math import inf
        import sys, os

        HUMAN = 1
        COMP = -1

        board = [[0, 0, 0],
                 [0, 0, 0],
                 [0, 0, 0]]

        MSG = "Welcome to Unbeatable Tic Tac Toe.\n" \
              "Our A.I can foreseen your moves ahead.\n" \
              "Are you sure to continue ? (y/n)"


        def evaluate(state):
            if wins(state, COMP):
                score = -1
            elif wins(state, HUMAN):
                score = 1
            else:
                score = 0

            return score

        def empty_cells(state):
            """Extract the remainder of board"""
            cells = [] # it contains all empty cells

            # Use enumerate for easy indexing
            for i, row in enumerate(state):
                for j, col in enumerate(row):
                    if state[i][j] == 0:
                        cells.append([i, j])

            return cells

        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]],
                [state[2][0], state[1][1], state[0][2]],
            ]

            if [player, player, player] in win_state:
                return True
            else:
                return False

        def game_over(state):
            """Check game over condition"""
            return wins(state, HUMAN) or wins(state, COMP)

        def clean():
            os_name = sys.platform.lower()
            os.system("cls")
            if 'win' in os_name:
                os.system('cls')
            else:
                os.system('clear')

        def minimax(state, depth, player):
            if player == COMP:
                best = [-1, -1, inf] # inf/-inf are the initial score for the players
            else:
                best = [-1, -1, -inf]

            if depth == 0 or game_over(state):
                score = evaluate(state)
                return [-1, -1, score]

            for cell in empty_cells(state):
```


(107) [ENG] Minimax implementa
Minimax-Tutorial/after.py at mas
Home Page - Select or create a n
Untitled8 - Jupyter Notebook
EX NO 2 Agent Program
localhost:8889/notebooks/Untitled8.ipynb?kernel_name=python3
Apps
Gmail
YouTube
Maps
Global NetAcad Ins...
Reading list
jupyter Untitled8 Last Checkpoint: 10 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
for cell in empty_cells(state):
# Fill the empty cells with the player symbols
x, y = cell[0], cell[1]
state[x][y] = player
#
score = minimax(state, depth - 1, -player)
state[x][y] = 0
score[0], score[1] = x, y
if player == COMP:
if score[2] < best[2]:
best = score
else:
if score[2] > best[2]:
best = score
return best
def human_turn(state):
# All possible moves
moves = {
1: [0, 0], 2: [0, 1], 3: [0, 2],
4: [1, 0], 5: [1, 1], 6: [1, 2],
7: [2, 0], 8: [2, 1], 9: [2, 2],
}
remain = empty_cells(state)
isTurn = True
print("Human Turn")
while isTurn:
try:
move = int(input("Enter your move (1-9) :"))
# When the player move is valid
if moves.get(move) in remain:
x, y = moves.get(move)
state[x][y] = HUMAN
isTurn = False
else: # Otherwise
print("Bad Move, try again.")
Type here to search
30°C
ENG
19:40
16-03-2022

(107) [ENG] Minimax implementa
Minimax-Tutorial/after.py at mas
Home Page - Select or create a n
Untitled8 - Jupyter Notebook
EX NO 2 Agent Program
localhost:8889/notebooks/Untitled8.ipynb?kernel_name=python3
Apps
Gmail
YouTube
Maps
Global NetAcad Ins...
Reading list
jupyter Untitled8 Last Checkpoint: 10 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
else: # Otherwise
print("Bad Move, try again.")
# When the player mistype
except ValueError:
print("Blank space and string are prohibited, please enter (1-9)")
# While-else loop, this code below will run after successful loop.
else:
# Clean the terminal, and show the current board
clean()
print(render(state))
def ai_turn(state):
depth = len(empty_cells(state)) # The remaining of empty cells
row, col, score = minimax(state, depth, COMP) # the optimal move for computer
state[row][col] = COMP
print("A.I Turn")
print(render(state)) # Show result board
def render(state):
legend = {0: " ", 1: "X", -1: "O"}
state = list(map(lambda x: [legend[y] for y in x], state))
result = "{}\n{}\n{}\n".format(*state)
return result
def main():
print(MSG)
start = False
while not start:
confirm = input("")
if confirm.lower() in ["y", "yes"]:
start = True
elif confirm.lower() in ["n", "no"]:
sys.exit()
else:
print("Please enter 'y' or 'n'")
Type here to search
30°C
ENG
19:40
16-03-2022


Untitled8 - Jupyter Notebook
EX NO 2 Agent Program
localhost:8889/notebooks/Untitled8.ipynb?kernel_name=python3
jupyter Untitled8 Last Checkpoint: 10 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
if confirm.lower() in ["y", "yes"]:
start = True
elif confirm.lower() in ["n", "no"]:
sys.exit()
else:
print("Please enter 'y' or 'n'")
else:
clean()
print("Game is settled !\n")
print(render(board), end="\n")
while not wins(board, COMP) and not wins(board, HUMAN):
human_turn(board)
if len(empty_cells(board)) == 0: break
ai_turn(board)
if wins(board, COMP):
print("A.I wins, 'I see throught your moves'")
elif wins(board, HUMAN):
print("Human wins, 'How can I lose to human ?'")
else:
print("It's a Draw. No one wins")
if __name__ == '__main__':
main()
Welcome to Unbeatable Tic Tac Toe.
Our A.I can foreseen your moves ahead.
Are you sure to continue ? (y/n)
y
Game is settled !
[' ', ' ', ' ']
[' ', ' ', ' ']
[' ', ' ', ' ']
Human Turn
Enter your move (1-9) :4
Type here to search
30°C
ENG
19:40
16-03-2022

Untitled8 - Jupyter Notebook
EX NO 2 Agent Program
localhost:8889/notebooks/Untitled8.ipynb?kernel_name=python3
jupyter Untitled8 Last Checkpoint: 11 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Enter your move (1-9) :4
[' ', ' ', ' ']
['X', ' ', ' ']
[' ', ' ', ' ']
A.I Turn
['O', ' ', ' ']
['X', ' ', ' ']
[' ', ' ', ' ']
Human Turn
Enter your move (1-9) :5
['O', ' ', ' ']
['X', 'X', ' ']
[' ', ' ', ' ']
A.I Turn
['O', ' ', ' ']
['X', 'X', 'O']
[' ', ' ', ' ']
Human Turn
Enter your move (1-9) :3
['O', ' ', 'X']
['X', 'X', 'O']
[' ', ' ', ' ']
A.I Turn
['O', ' ', 'X']
['X', 'X', 'O']
['O', ' ', ' ']
Human Turn
Enter your move (1-9) :9
['O', ' ', 'X']
['X', 'X', 'O']
['O', ' ', 'X']
A.I Turn
['O', 'O', 'X']
Type here to search
30°C
ENG
19:40
16-03-2022


(107) [ENG] Minimax implementa
Minimax-Tutorial/after.py at mas
Home Page - Select or create a n
Untitled8 - Jupyter Notebook
EX NO 2 Agent Program
localhost:8889/notebooks/Untitled8.ipynb?kernel_name=python3
Apps
Gmail
YouTube
Maps
Global NetAcad Ins...
Reading list
jupyter
Untitled8
Last Checkpoint: 11 minutes ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
['O', ' ', 'X']
['X', 'X', 'O']
[' ', ' ', ' ']
A.I Turn
['O', ' ', 'X']
['X', 'X', 'O']
['O', ' ', ' ']
Human Turn
Enter your move (1-9) :9
['O', ' ', 'X']
['X', 'X', 'O']
['O', ' ', 'X']
A.I Turn
['O', 'O', 'X']
['X', 'X', 'O']
['O', ' ', 'X']
Human Turn
Enter your move (1-9) :8
['O', 'O', 'X']
['X', 'X', 'O']
['O', 'X', 'X']
It's a Draw. No one wins
In [ ]:
In [ ]:
Type here to search
30°C
ENG
19:40
16-03-2022

Result:

The problem statement for MinMax algorithm(TIC-TAC TOE) is solved.