Name:Meruva Kodanda Suraj Roll no: 241020446

Code:

import math

HUMAN = "X"

SAIRAM = "O"

EMPTY = " "

# ---------- Board Helpers ----------

def print\_board(board):

print()

for r in range(3):

row = [board[r\*3+c] if board[r\*3+c] != EMPTY else str(r\*3+c+1) for c in range(3)]

print(" | ".join(row))

if r < 2: print("-"\*5)

print()

def is\_moves\_left(board):

return EMPTY in board

def evaluate(board):

for r in range(3):

if board[r\*3] == board[r\*3+1] == board[r\*3+2] != EMPTY:

return +10 if board[r\*3] == SAIRAM else -10

for c in range(3):

if board[c] == board[c+3] == board[c+6] != EMPTY:

return +10 if board[c] == SAIRAM else -10

if board[0] == board[4] == board[8] != EMPTY:

return +10 if board[0] == SAIRAM else -10

if board[2] == board[4] == board[6] != EMPTY:

return +10 if board[2] == SAIRAM else -10

return 0

# ---------- Minimax with Alpha-Beta + Node Counter ----------

ab\_nodes = 0

def minimax\_ab(board, depth, is\_maximizing, alpha, beta):

global ab\_nodes

ab\_nodes += 1

score = evaluate(board)

if score == 10: return score - depth

if score == -10: return score + depth

if not is\_moves\_left(board): return 0

if is\_maximizing: # SAIRAM

best = -math.inf

for i in range(9):

if board[i] == EMPTY:

board[i] = SAIRAM

best = max(best, minimax\_ab(board, depth+1, False, alpha, beta))

board[i] = EMPTY

alpha = max(alpha, best)

if beta <= alpha: break

return best

else: # Human

best = math.inf

for i in range(9):

if board[i] == EMPTY:

board[i] = HUMAN

best = min(best, minimax\_ab(board, depth+1, True, alpha, beta))

board[i] = EMPTY

beta = min(beta, best)

if beta <= alpha: break

return best

# ---------- Best Move ----------

def best\_move(board):

global ab\_nodes

best\_val = -math.inf

best\_index = -1

ab\_nodes = 0 # reset counter

for i in range(9):

if board[i] == EMPTY:

board[i] = SAIRAM

move\_val = minimax\_ab(board, 0, False, -math.inf, math.inf)

board[i] = EMPTY

if move\_val > best\_val:

best\_val = move\_val

best\_index = i

print(f"SAIRAM evaluated {ab\_nodes} nodes for this move.")

return best\_index

# ---------- Play Game ----------

def play():

board = [EMPTY]\*9

human\_turn = True

print("Welcome to Tic Tac Toe (3x3)")

print("You are X, SAIRAM is O\n")

while True:

print\_board(board)

score = evaluate(board)

if score == 10:

print("SAIRAM wins!")

break

elif score == -10:

print("You win!")

break

elif not is\_moves\_left(board):

print("Draw!")

break

if human\_turn:

move = int(input("Your move (1-9): ")) - 1

if 0 <= move < 9 and board[move] == EMPTY:

board[move] = HUMAN

human\_turn = False

else:

print("Invalid move, try again.")

else:

print("SAIRAM is thinking...")

move = best\_move(board)

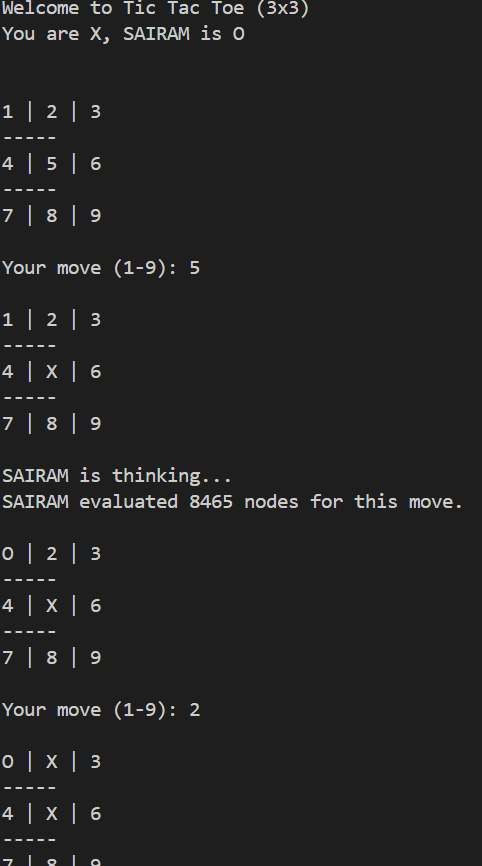
board[move] = SAIRAM

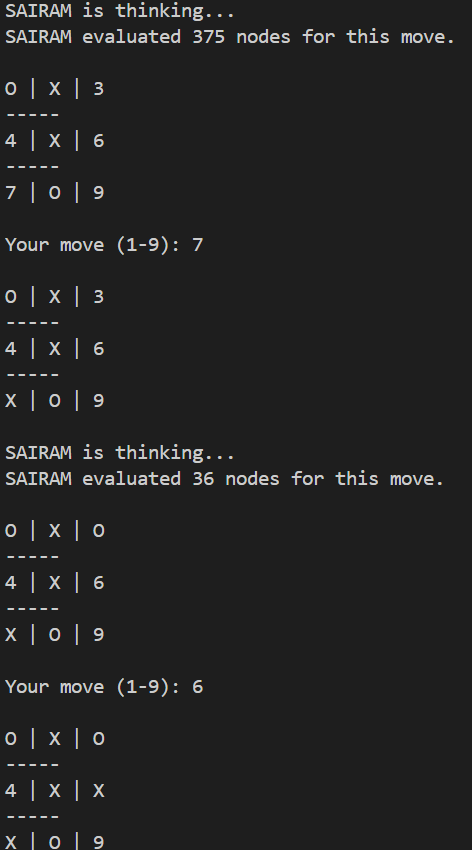
human\_turn = True

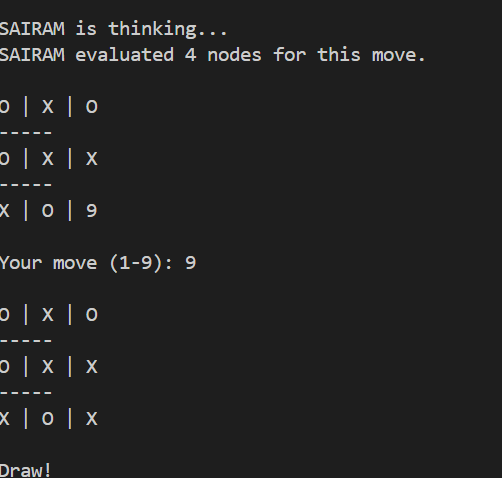
if \_\_name\_\_ == "\_\_main\_\_":

play()

Output:







Assumptions:   
As 4x4 Tic Tac Toe has very large runtime   
I have proceeded with 3x3 tic tac toe version.