# Minimax for Tic-Tac-Toe (X = AI, O = human/random)

from typing import List, Optional

X, O, EMPTY = "X", "O", " "

WIN\_LINES = [(0,1,2),(3,4,5),(6,7,8),

(0,3,6),(1,4,7),(2,5,8),

(0,4,8),(2,4,6)]

def winner(board: List[str]) -> Optional[str]:

for a,b,c in WIN\_LINES:

if board[a] != EMPTY and board[a] == board[b] == board[c]:

return board[a]

return None

def terminal(board: List[str]) -> bool:

return winner(board) is not None or all(c != EMPTY for c in board)

def utility(board: List[str]) -> int:

w = winner(board)

if w == X: return 1

if w == O: return -1

return 0

def actions(board: List[str]) -> List[int]:

return [i for i,c in enumerate(board) if c == EMPTY]

def result(board: List[str], move: int, player: str) -> List[str]:

b = board[:]

b[move] = player

return b

def player\_to\_move(board: List[str]) -> str:

x\_count = board.count(X)

o\_count = board.count(O)

return X if x\_count == o\_count else O

def minimax(board: List[str]) -> int:

"""Return best move index for current player (X maximizes, O minimizes)."""

p = player\_to\_move(board)

def max\_value(b):

if terminal(b): return utility(b), None

v, best = -2, None

for m in actions(b):

val, \_ = min\_value(result(b, m, X))

if val > v:

v, best = val, m

if v == 1: break # pruning by optimal value

return v, best

def min\_value(b):

if terminal(b): return utility(b), None

v, best = 2, None

for m in actions(b):

val, \_ = max\_value(result(b, m, O))

if val < v:

v, best = val, m

if v == -1: break # pruning by optimal value

return v, best

\_, move = (max\_value(board) if p == X else min\_value(board))

return move

# Demo play: AI (X) vs random O

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

import random

board = [EMPTY]\*9

while not terminal(board):

p = player\_to\_move(board)

if p == X:

mv = minimax(board)

else:

mv = random.choice(actions(board))

board[mv] = p

print(p, "->", mv, " | ", [board[i:i+3] for i in range(0,9,3)])

print("Winner:", winner(board))