1. **Tic-Tac-Toe**

def print\_board(board):

for i in range(3): print(' | '.join(board[i]) or '-+-+-')

def check\_winner(board, player):

return any(all(board[i][j] == player for j in range(3)) or

all(board[j][i] == player for j in range(3)) for i in range(3)) or \

all(board[i][i] == player for i in range(3)) or \

all(board[i][2-i] == player for i in range(3))

def tic\_tac\_toe():

board = [[' ']\*3 for \_ in range(3)]

for turn in range(9):

print\_board(board)

player = 'X' if turn % 2 == 0 else 'O'

move = int(input(f"Player {player}, enter your move (1-9): ")) - 1

row, col = divmod(move, 3)

if board[row][col] == ' ':

board[row][col] = player

if check\_winner(board, player):

print\_board(board)

print(f"Player {player} wins!")

return

else:

print("Invalid move, try again.")

print\_board(board)

print("It's a draw!")

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

tic\_tac\_toe()

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. **Water Jug Problem**

import math

def can\_measure\_water(jug1\_capacity, jug2\_capacity, target):

if target > max(jug1\_capacity, jug2\_capacity):

return False

if target % math.gcd(jug1\_capacity, jug2\_capacity) != 0:

return False

def can\_measure(jug1\_cap, jug2\_cap, target):

jug1 = jug2 = 0

while jug1 != target and jug2 != target:

if jug1 == 0:

jug1 = jug1\_cap

transfer = min(jug1, jug2\_cap - jug2)

jug1 -= transfer

jug2 += transfer

if jug1 == target or jug2 == target:

return True

if jug2 == jug2\_cap:

jug2 = 0

return False

return can\_measure(jug1\_capacity, jug2\_capacity, target) or can\_measure(jug2\_capacity, jug1\_capacity, target)

jug1\_capacity = 4

jug2\_capacity = 3

target = 2

if can\_measure\_water(jug1\_capacity, jug2\_capacity, target):

print(f"Solution found: You can measure exactly {target} liters.")

else:

print(f"No solution: It's impossible to measure exactly {target} liters.")

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**OR**

import math

def can\_measure\_water(jug1, jug2, target):

# Check if target is within the possible range and divisible by gcd

return target <= max(jug1, jug2) and target % math.gcd(jug1, jug2) == 0

jug1\_capacity = 4

jug2\_capacity = 3

target = 2

if can\_measure\_water(jug1\_capacity, jug2\_capacity, target):

print(f"Solution found: You can measure exactly {target} liters.")

else:

print(f"No solution: It's impossible to measure exactly {target} liters.")

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. **Min-Max Algorithm**

import math

def minimax(depth, node\_index, maximizing\_player, values, alpha, beta):

if depth == 3:

return values[node\_index]

if maximizing\_player:

best = -math.inf

for i in range(2):

val = minimax(depth + 1, node\_index \* 2 + i, False, values, alpha, beta)

best = max(best, val)

alpha = max(alpha, best)

if beta <= alpha:

break

return best

else:

best = math.inf

for i in range(2):

val = minimax(depth + 1, node\_index \* 2 + i, True, values, alpha, beta)

best = min(best, val)

beta = min(beta, best)

if beta <= alpha:

break

return best

values = [3,4,2,1,7,8,0,9]

optimal\_value = minimax(0, 0, True, values, -math.inf, math.inf)

print("The optimal value is:", optimal\_value)

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. **Alpha Beta Pruning**

import math

def minimax(depth, node, is\_max, values, alpha, beta):

if depth == 3:

return values[node]

if is\_max:

best = -math.inf

for i in range(2):

best = max(best, minimax(depth + 1, node \* 2 + i, False, values, alpha, beta))

alpha = max(alpha, best)

if beta <= alpha:

break

return best

else:

best = math.inf

for i in range(2):

best = min(best, minimax(depth + 1, node \* 2 + i, True, values, alpha, beta))

beta = min(beta, best)

if beta <= alpha:

break

return best

values = [3, 5, 6, 9, 1, 2, 0, -1]

print("The optimal value is:", minimax(0, 0, True, values, -math.inf, math.inf))

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***