8 Puzzle game

from collections import deque

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

# Function to print the board nicely

def print\_board(state):

for i in range(0,9,3):

print(state[i], state[i+1], state[i+2])

print()

# Generate all legal neighbor states

def neighbors(state):

i = state.index(0) # blank position

r, c = divmod(i, 3)

for j in (i-3, i+3, i-1, i+1):

if 0 <= j < 9:

# prevent wrap-around in rows

if (j == i-1 and c == 0) or (j == i+1 and c == 2):

continue

s = list(state)

s[i], s[j] = s[j], s[i]

yield tuple(s)

# Check if the puzzle is solvable

def is\_solvable(state):

arr = [x for x in state if x != 0]

inv = 0

for i in range(len(arr)):

for j in range(i+1, len(arr)):

if arr[i] > arr[j]:

inv += 1

return inv % 2 == 0

# BFS to solve the puzzle

def bfs(start):

if not is\_solvable(start):

return None

queue = deque([start])

parent = {start: None} # keep track of moves

while queue:

cur = queue.popleft()

if cur == GOAL:

# reconstruct path

path = []

while cur is not None:

path.append(cur)

cur = parent[cur]

return path[::-1] # reverse to get start → goal

for nb in neighbors(cur):

if nb not in parent:

parent[nb] = cur

queue.append(nb)

return None

# Main program

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

print("Enter the 8-puzzle start state (0 for blank) row by row, separated by spaces:")

user\_input = []

for i in range(3):

row = input(f"Row {i+1}: ").split()

if len(row) != 3:

print("Please enter exactly 3 numbers per row.")

exit()

user\_input.extend(int(x) for x in row)

start = tuple(user\_input)

print("\nStart Board:")

print\_board(start)

print("Solvable?", is\_solvable(start))

path = bfs(start)

if path:

print("Solved in", len(path)-1, "moves")

for step, state in enumerate(path):

print(f"Step {step}:")

print\_board(state)

else:

print("No solution (unsolvable)")

