15/11/24

LAB-5

Simulated Annealing for N-Queens problem:

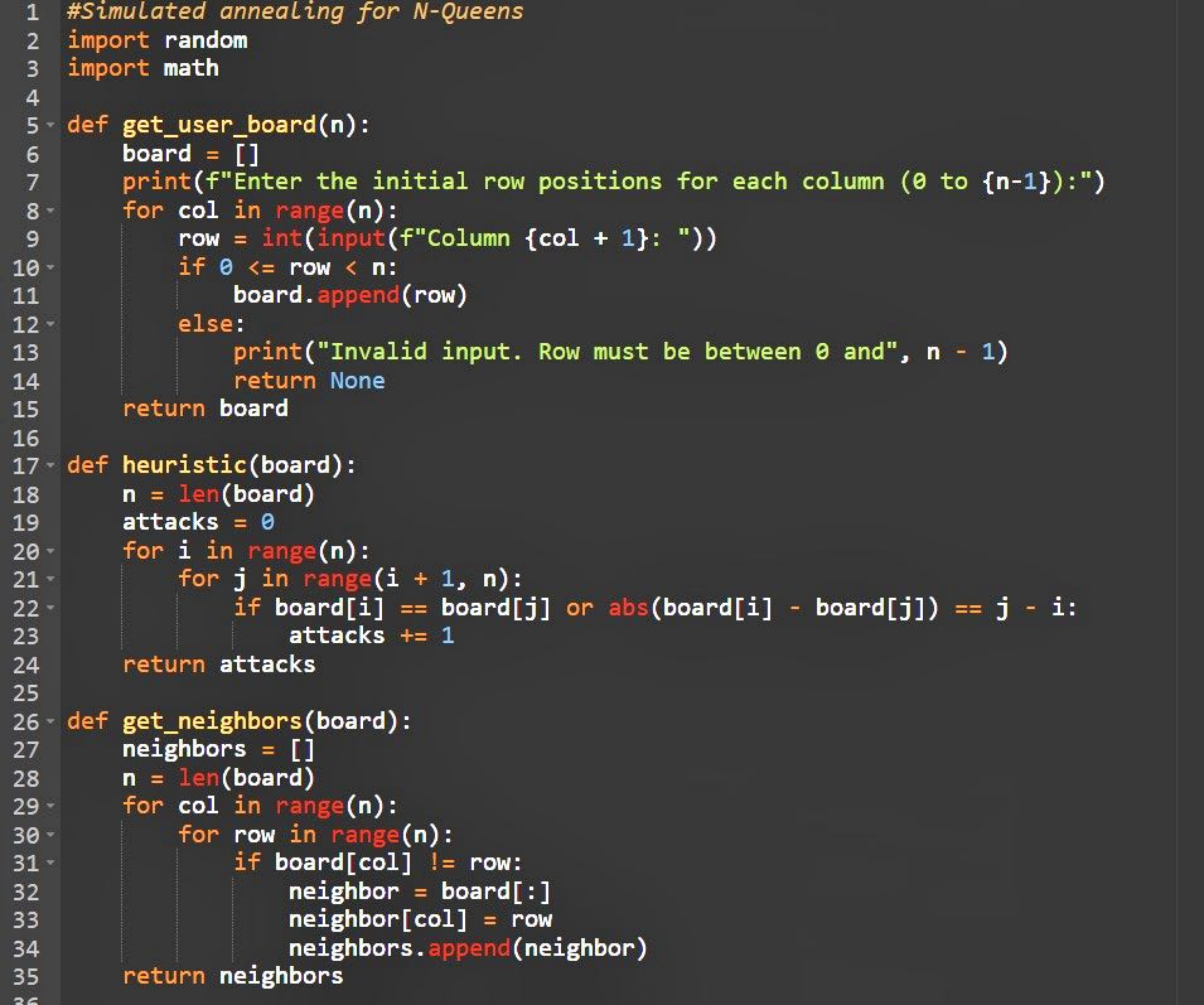
```
current ← initial state
T ← a large positive value
while T > 0 do:
        next ← a random neighbor of current
        ΔE ← current. cost - next. cost
        if ΔE > 0 then
                current ← next
        else:
                current ← next with probability $p = e^{\frac{ΔE}{T}}$
        end if
      decrease T
end while
return current
```

OUTPUT:

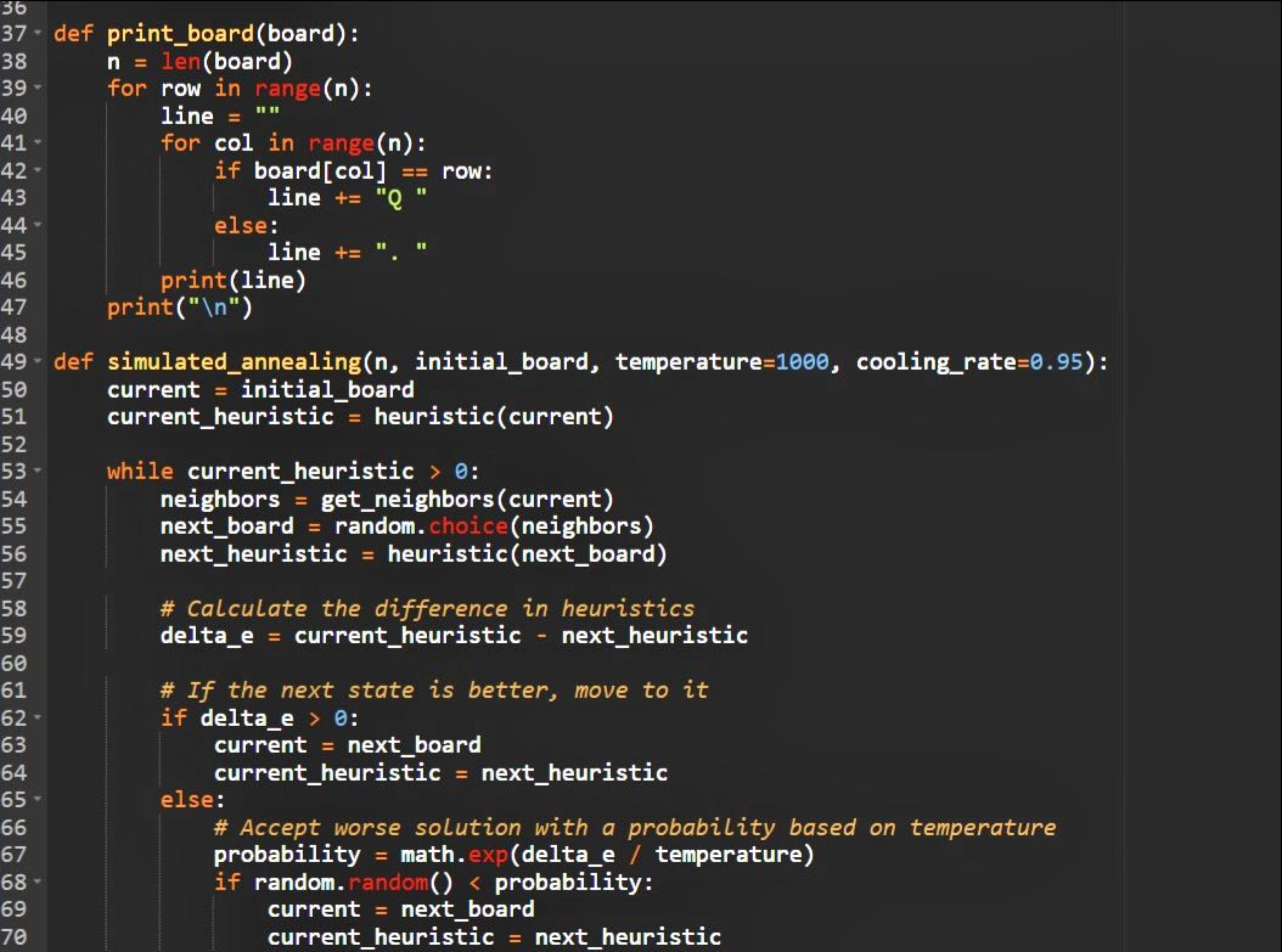
Queen positions in each row : [2, 5, 7, 0, 3, 6, 4, 7]

```
0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
```

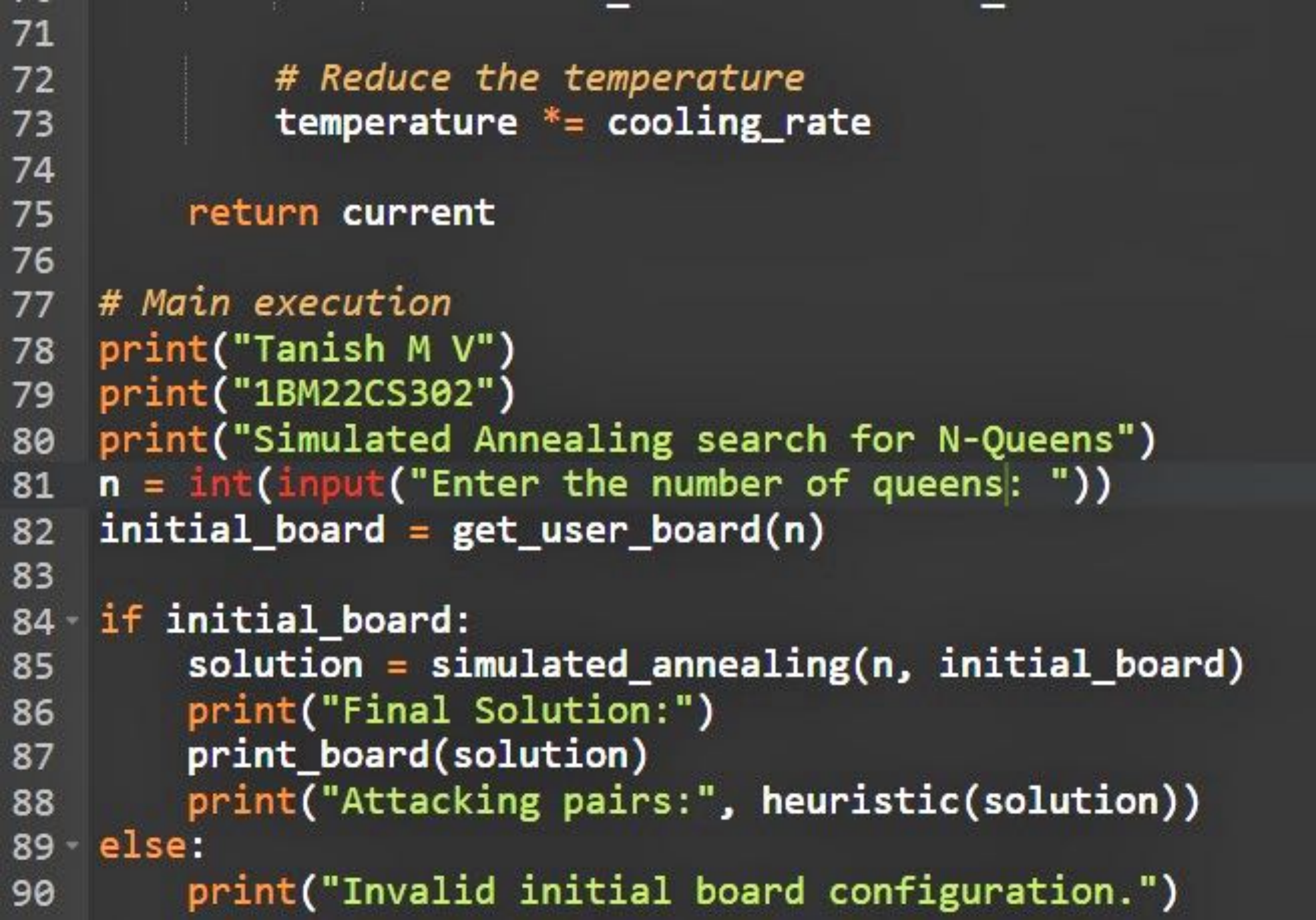
15/11/24



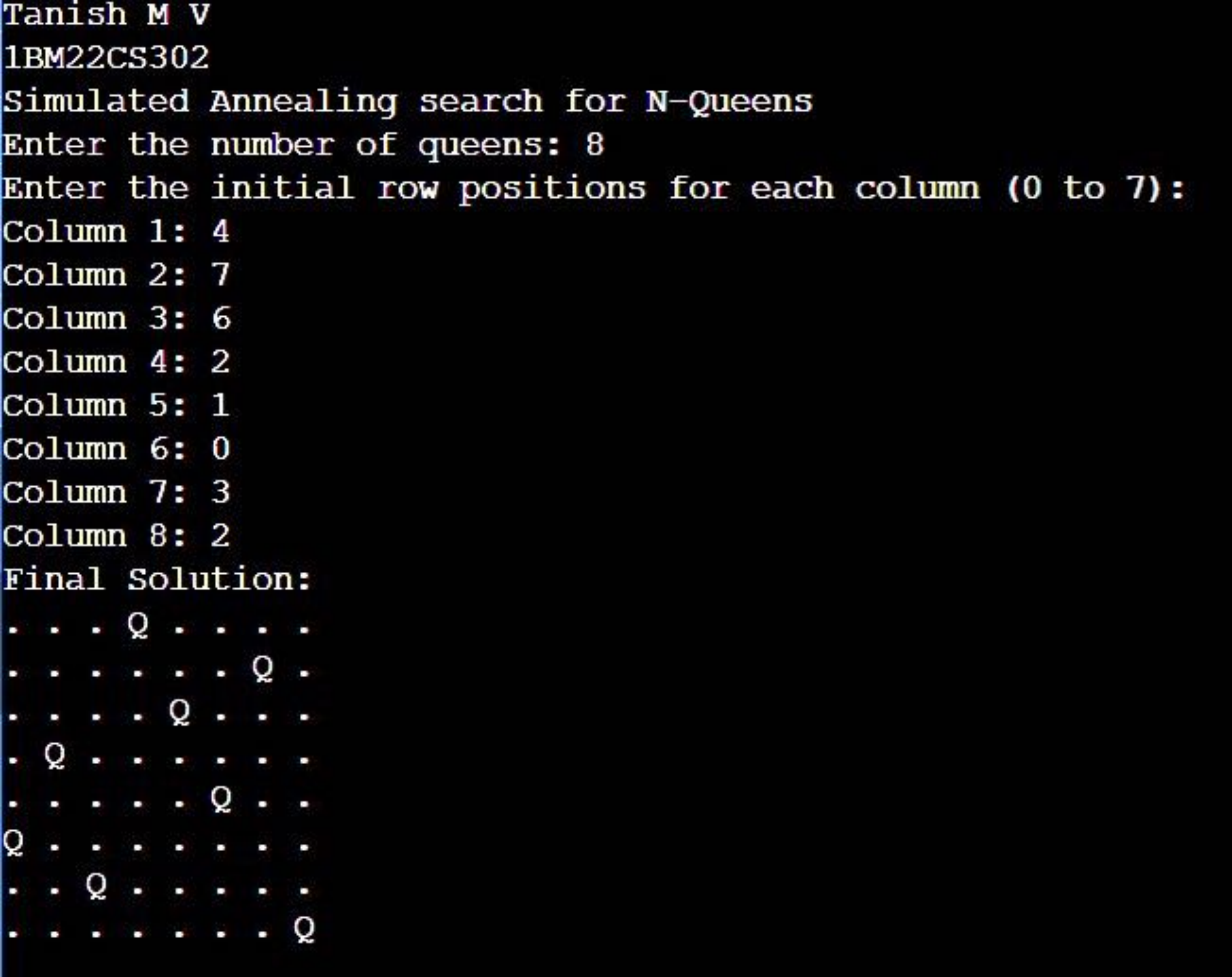
```
#Simulated annealing for N-Queens
import random
import math

def get_user_board(n):
    board = []
    print(f"Enter the initial row positions for each column (0 to {n-1}):")
    for col in range(n):
        row = int(input(f"Column {col + 1}: "))
        if 0 <= row < n:
            board.append(row)
        else:
            print("Invalid input. Row must be between 0 and", n - 1)
            return None
    return board

def heuristic(board):
    n = len(board)
    attacks = 0
    for i in range(n):
        for j in range(i + 1, n):
            if board[i] == board[j] or abs(board[i] - board[j]) == j - i:
                attacks += 1
    return attacks

def get_neighbors(board):
    neighbors = []
    n = len(board)
    for col in range(n):
        for row in range(n):
            if board[col] != row:
                neighbor = board[:]
                neighbor[col] = row
                neighbors.append(neighbor)
    return neighbors
```



```python

def print_board(board):
    n = len(board)
    for row in range(n):
        line = ""
        for col in range(n):
            if board[col] == row:
                line += "Q "
            else:
                line += ". "
        print(line)
    print("\n")

def simulated_annealing(n, initial_board, temperature=1000, cooling_rate=0.95):
    current = initial_board
    current_heuristic = heuristic(current)

    while current_heuristic > 0:
        neighbors = get_neighbors(current)
        next_board = random.choice(neighbors)
        next_heuristic = heuristic(next_board)

        # Calculate the difference in heuristics
        delta_e = current_heuristic - next_heuristic

        # If the next state is better, move to it
        if delta_e > 0:
            current = next_board
            current_heuristic = next_heuristic
        else:
            # Accept worse solution with a probability based on temperature
            probability = math.exp(delta_e / temperature)
            if random.random() < probability:
                current = next_board
                current_heuristic = next_heuristic
```



```python

        # Reduce the temperature
        temperature *= cooling_rate

    return current

# Main execution
print("Tanish M V")
print("1BM22CS302")
print("Simulated Annealing search for N-Queens")
n = int(input("Enter the number of queens: "))
initial_board = get_user_board(n)

if initial_board:
    solution = simulated_annealing(n, initial_board)
    print("Final Solution:")
    print_board(solution)
    print("Attacking pairs:", heuristic(solution))
else:
    print("Invalid initial board configuration.")
```



```
Tanish M V
1BM22CS302
Simulated Annealing search for N-Queens
Enter the number of queens: 8
Enter the initial row positions for each column (0 to 7):
Column 1: 4
Column 2: 7
Column 3: 6
Column 4: 2
Column 5: 1
Column 6: 0
Column 7: 3
Column 8: 2
Final Solution:
. . . Q . . . .
. . . . . . Q .
. . . . Q . . .
. Q . . . . . .
. . . . . Q . .
Q . . . . . . .
. . Q . . . . .
. . . . . . . Q
```