DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF E&ME, NUST, RAWALPINDI

AI & Decision Support Systems

Lab Report #6

Student Name: Nawab Aarij Imam

Degree/ Syndicate: 43 CE - A

Task:

Code:

```
import numpy as np

def generate_population(num_chromosomes):
    return np.random.randint(8, size=(num_chromosomes, 8))

def calculate_fitness(chromosome):
    non_attacking_pairs = 28
    for i in range(8):
        for j in range(i+1, 8):
            if chromosome[i] == chromosome[j] or abs(chromosome[i] - chromosome[j]) == abs(i - j):
                non_attacking_pairs -= 1
    return non_attacking_pairs

def calculate_fitness_percentages(chromosomes):
    fitnesses = np.apply_along_axis(calculate_fitness, axis=1, arr=chromosomes)
    return fitnesses / np.sum(fitnesses)

def select_parents(chromosomes):
    fitness_percentages = calculate_fitness_percentages(chromosomes)
    cumulative_probabilities = np.cumsum(fitness_percentages)

    selected_pairs = []
    for _ in range(len(chromosomes)):
        parents = []
        while len(parents) < 2:
            rand = np.random.random()
            parent_index = np.argwhere(cumulative_probabilities > rand)[0][0]
            if parent_index not in parents:
                parents.append(parent_index)
        selected_pairs.append(parents)

    return selected_pairs
```

```
def crossover_and_mutate(parent1, parent2):
    crossover_point = np.random.randint(0, 8)
    child = np.concatenate((parent1[:crossover_point], parent2[crossover_point:]))

    mutation_gene = np.random.randint(0, 8)
    child[mutation_gene] = np.random.randint(0, 8)

    return child

def evolve_population(chromosomes):
    new_population = []
    parent_pairs = select_parents(chromosomes)

    for pair in parent_pairs:
        new_chromosome = crossover_and_mutate(chromosomes[pair[0]], chromosomes[pair[1]])
        new_population.append(new_chromosome)

    return np.array(new_population)

def solve_eight_queens(population_size, max_generations):
    population = generate_population(population_size)

    for generation in range(max_generations):
        fitnesses = np.apply_along_axis(calculate_fitness, axis=1, arr=population)

        if generation % 1000 == 0:
            print(f'Generation {generation}: Max fitness = {np.max(fitnesses)}')

        if 28 in fitnesses:
            solution_index = np.where(fitnesses == 28)[0][0]
            return population[solution_index], generation

        population = evolve_population(population)

    return None, max_generations
```

```python
def main():
    population_size = 20
    max_generations = 10000

    solution, generations = solve_eight_queens(population_size, max_generations)

    if solution is not None:
        print(f'\nSolution found in generation {generations}:')
        print(solution)

        print('\nBoard representation:')
        for row in range(8):
            board_row = ['Q' if solution[row] == col else '.' for col in range(8)]
            print(' '.join(board_row))
    else:
        print('\nNo solution found within the maximum number of generations.')

if __name__ == "__main__":
    main()
```

Output:

```
❯ python task.py
Generation 0: Max fitness = 25
Generation 1000: Max fitness = 23

Solution found in generation 1164:
[4 7 3 0 2 5 1 6]

Board representation:
. . . . Q . . .
. . . . . . . Q
. . . Q . . . .
Q . . . . . . .
. . Q . . . . .
. . . . . Q . .
. Q . . . . . .
. . . . . . Q .
```