Koushik Sahu
118CS0597
Artificial Intelligence Lab – IV
22nd September 2021

Code:

```
import random

def fitness(chromosome):
    horizontal_collisions = sum([chromosome.count(queen)-1 for queen in chromosome])/2
    diagonal_collisions = 0

    n = len(chromosome)
    left_diagonal = [0] * 2*n
    right_diagonal = [0] * 2*n
    for i in range(n):
        left_diagonal[i + chromosome[i] - 1] += 1
        right_diagonal[len(chromosome) - i + chromosome[i] - 2] += 1

    diagonal_collisions = 0
    for i in range(2*n-1):
        counter = 0
        if left_diagonal[i] > 1:
            counter += left_diagonal[i]-1
        if right_diagonal[i] > 1:
            counter += right_diagonal[i]-1
        diagonal_collisions += counter / (n-abs(i-n+1))

    return int(maxFitness - (horizontal_collisions + diagonal_collisions))

def probability(chromosome, fitness):
    return fitness(chromosome) / maxFitness

def random_pick(population, probabilities):
    populationWithProbabilty = zip(population, probabilities)
    total = sum(w for _, w in populationWithProbabilty)
    r = random.uniform(0, total)
    upto = 0
    for c, w in zip(population, probabilities):
        if upto + w >= r:
            return c
        upto += w

def reproduce(x, y):
    n = len(x)
    c = random.randint(0, n - 1)
    return x[0:c] + y[c:n]

def mutate(x):
    n = len(x)
    c = random.randint(0, n - 1)
```

```
    m = random.randint(1, n)
    x[c] = m
    return x

def print_chromosome(chrom):
    print("Chromosome = {},  Fitness = {}"
        .format(str(chrom), fitness(chrom)))

def genetic_queen(population, fitness):
    mutation_probability = 0.03
    new_population = []
    probabilities = [probability(n, fitness) for n in population]
    for i in range(len(population)):
        x = random_pick(population, probabilities)
        y = random_pick(population, probabilities)
        child = reproduce(x, y)
        if random.random() < mutation_probability:
            child = mutate(child)
        new_population.append(child)
        if fitness(child) == maxFitness: break
    return new_population

if __name__ == "__main__":
    print('Solving 8 queens problem...')
    nq = 8
    maxFitness = (nq*(nq-1))/2
    def random_chromosome(size):
        return [ random.randint(1, nq) for _ in range(nq) ]
    population = [random_chromosome(nq) for _ in range(100)]

    generation = 1

    while not maxFitness in [fitness(chrom) for chrom in population]:
        population = genetic_queen(population, fitness)
        generation += 1
    chrom_out = []
    print("Solved in Generation {}!".format(generation-1))
    for chrom in population:
        if fitness(chrom) == maxFitness:
            print("");
            print("One of the solutions: ")
            chrom_out = chrom
            print_chromosome(chrom)

    board = []

    for x in range(nq):
        board.append(["*"] * nq)

    for i in range(nq):
        board[nq-chrom_out[i]][i]="Q"
```

```
print()
for row in board:
    print(''.join(row))
```

Sample input:

* No input required *

Sample output:

```
Solving 8 queens problem...
Solved in Generation 388!

One of the solutions:
Chromosome = [6, 4, 2, 8, 5, 7, 1, 3],  Fitness = 28

***Q****
*****Q**
Q*******
****Q***
*Q******
*******Q
**Q*****
******Q*
```