AI LAB | 5 September 2023

Name: Maloth Aditya

Roll No.: 120CS0124

N Queens problem using Genetic Algorithm

Code:

```
# 120CS0124 Maloth Aditya

# n-queens problem using Genetic algorithm

# Generate a random state of the 8-queens and arrange them in such a way that no queens attach each other


import random


def random_chromosome(size):
    return [ random.randint(1, nq) for _ in range(nq) ]


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)) #28-(2+3)=23


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


def random_pick(population, probabilities):
    populationWithProbabilty = zip(population, probabilities)
    total = sum(w for c, 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
    assert False, "Shouldn't get here"


def reproduce(x, y): #doing cross_over between two chromosomes
    n = len(x)
    c = 3
    return x[0:c] + y[c:n]


def mutate(x):  #randomly changing the value of a random index of a
chromosome
    n = len(x)
    c = random.randint(0, n - 1)
    m = random.randint(1, n)
    x[c] = m
    return x


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) #best chromosome 1
        y = random_pick(population, probabilities) #best chromosome 2
        child = reproduce(x, y) #creating two new chromosomes from the best 2
chromosomes
```

```
        if random.random() < mutation_probability:
            child = mutate(child)
        print_chromosome(child)
        new_population.append(child)
        if fitness(child) == maxFitness: break
    return new_population


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


if __name__ == "__main__":
    nq = int(input("Enter Number of Queens: ")) #say N = 8
    maxFitness = (nq*(nq-1))/2  # 8*7/2 = 28
    population = [random_chromosome(nq) for _ in range(100)]

    generation = 1

    while not maxFitness in [fitness(chrom) for chrom in population]:
        print("=== Generation {} ===".format(generation))
        population = genetic_queen(population, fitness)
        print("")
        print("Maximum Fitness = {}".format(max([fitness(n) for n in population])))
        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(["x"] * nq)

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



def print_board(board):
   for row in board:
      print (" ".join(row))

print()
print_board(board)
```

OUTPUT:

```
Chromosome = [7, 1, 8, 4, 5, 7, 3, 6],  Fitness = 26
Chromosome = [7, 1, 8, 4, 5, 8, 3, 7],  Fitness = 25
Chromosome = [7, 2, 3, 4, 5, 7, 3, 6],  Fitness = 25
Chromosome = [5, 1, 8, 4, 2, 7, 3, 6],  Fitness = 28

Maximum Fitness = 28
Solved in Generation 13095!

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

x x Q x x x x x
x x x x x Q x x
x x x x x x x Q
Q x x x x x x x
x x x Q x x x x
x x x x x x Q x
x x x x Q x x x
x Q x x x x x x
```