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**:

