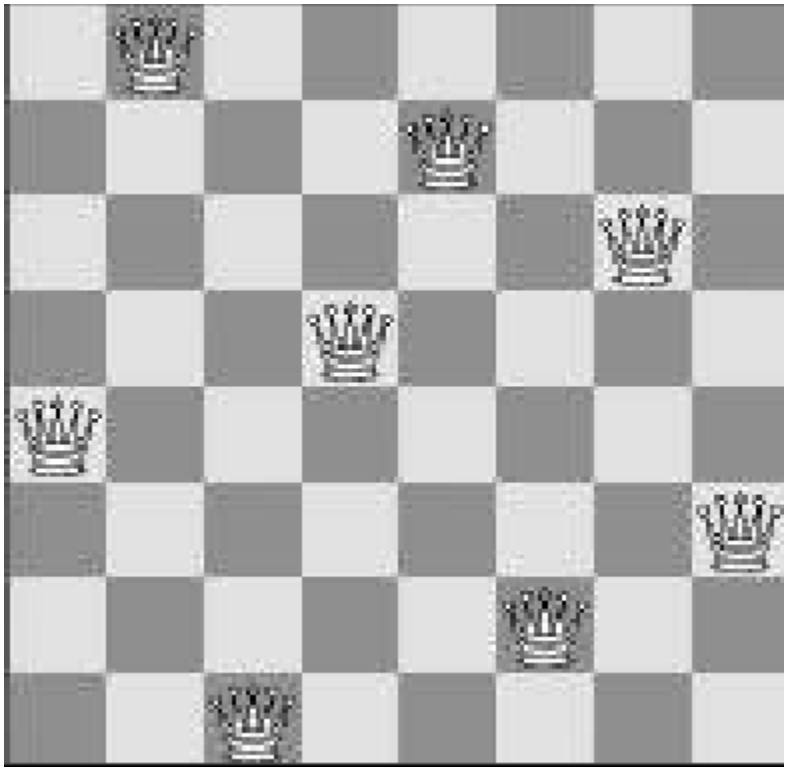
Artificial Intelligence
Lab 2- N-Queen Problem Using Genetic algorithm

Dharamraj Bhatt
Reg.no: 1947216

Aim: The aim of the N-Queens Problem is to place N queens on an N x N chessboard, in a way so that no queen is in conflict with the others.



Terminology

Gene: An individual is characterized by a set of variables
Chromosome: Genes are joined into a string to form a Chromosome (solution). A chromosome is a set of parameters that define a proposed solution to the problem that the genetic algorithm is trying to solve
Population: The set of all solutions
Fitness Function: Pairs of non-attacking queens (say for N=6, Fmax= 6C2 = 6*5/2 = 15)

Crossover: Also called recombination, is a genetic operator used to combine the genetic information of two parents to generate new offspring
Mutation: It alters one or more gene values in a chromosome from its initial state

How the genetic algorithm solves the n-queen problem?

Step 1: A random chromosome is generated
Step 2: The fitness value of the chromosome is calculated
Step 3: If fitness is not equal to Fmax
Step 4: Reproduce (crossover) new chromosome from 2 randomly selected best chromosomes
Step 5: Mutation may take place
Step 6: New chromosome added to the population
Repeat Step 2 to 6 until a chromosome (solution) with Fitness value = Fmax is found

Code:

```
import random

def random_chromosome(size): #making random chromosomes
    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 = random.randint(0, n - 1)
    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)
```