```python
import numpy as np
import random
from turtle import *

#initial population
def initial_Population(Population_Size):
    return np.random.randint(8 , size=(Population_Size,8))

#fitness => maximum

def Fitness_Function(Population):
    fitness_vals = []
    for individual in Population :
        fitness_score = 0
        for i in range(8):
            for j in range (( i +  1) , 8):
                if individual[i] ==  individual[j]:
                    fitness_score += 1
                elif abs(individual[i] - individual[j]) == abs(i-j):
                    fitness_score += 1
                fitness_val = 28 - fitness_score
        fitness_vals.append(fitness_val)
    return np.array(fitness_vals)

#Selection
def Roulette_Wheel_Selection(Population,Fitness_vals):

    probs = Fitness_vals.copy()
    probs = probs/probs.sum()
    N = len(Population)
    indices = np.arange(N)
    Selected_indices = np.random.choice(indices, size= N , p=probs )
    Selected_Population = Population[Selected_indices]
    return Selected_Population

#-------------------------------------------------------------
# def rank_selection(population):
#     population_size = len(population)
#     ranks = [i+1 for i in range(population_size)]
#     total_rank = sum(ranks)
#     selection_probabilities = [rank / total_rank for rank in ranks]
#     selection = []
#     for j in range(population_size):
#         r = random.uniform(0, 1)
#         for i in range(population_size):
#             if r <= selection_probabilities[i]:
#                 selection.append(population[i])
#                 break
#     return selection
#-------------------------------------------------------------
#Crossover

# def Single_Point_Crossover(parent1,parent2,Pc):
#     P_crossover = np.random.random()
#     if P_crossover < Pc :
#             point = np.random.randint(1,8)
#             child1 = np.concatenate([parent1[ : point] , parent2[point: ]])
#             child2 = np.concatenate([parent2[ : point] , parent1[point : ]])
#     else :
#             child1 = parent1.copy()
#             child2 = parent2.copy()
#     return child1 , child2
# #-------------------------------------------------------------
def Two_Point_Crossover(parent1,parent2,Pc):

        P_crossover = np.random.random()
        if P_crossover < Pc :
                point1 = np.random.randint(1,4)
                point2 =np.random.randint(4,8)
                child1 = np.concatenate([parent1[ : point1] , parent2[point1:point2],parent1[point2:]])
                child2 = np.concatenate([parent2[ : point1] , parent1[point1:point2],parent2[point2:]])
        else :
                child1 = parent1.copy()
                child2 = parent2.copy()

        # print("crossover gene",point1 ,",", point2)
        # print(point1,point2)
        return child1 , child2

```

```python
# #-------------------------------------------------------------
# def Three_Parent_Crossover(parent1 , parent2 , parent3 , Pc ) :
#     P_crossover = np.random.random()
#     child = []
#     if P_crossover < Pc :
#             for i in range (8):
#                 if parent1[i] == parent2[i]:
#                     child.append(parent1[i])
#                 else :
#                     child.append(parent3[i])
#     else :
#             child = parent1.copy()
#     return child
#-------------------------------------------------------------
# def uniform_crossover(parent1, parent2,pc):
#  mask=[]
#  for i in range(8):
#     x=random.randint(0,1)
#     mask.append(x)

#  print('parent1=',parent1)
#  print('parent2=',parent2)
#  print('mask=',mask)

#  child1=[]
#  child2=[]
#  for i in range(8):
#     if mask[i] == 1:
#         child1.append(parent1[i])
#         child2.append(parent2[i])
#     else:
#         child1.append(parent2[i])
#         child2.append(parent1[i])

#  print('child1=',child1)
#  print('child2=',child2)

# #-------------------------------------------------------------

#Mutation
def Flipping_Mutation(chromosome , Pm):
    P_mutation = np.random.random()
    point = np.random.randint(8)
    if P_mutation < Pm :
        chromosome[point] = np.random.randint(8)
    return chromosome
#-------------------------------------------------------------
#crossover and mutation
def Crossover_Mutation(Selected_Population , Pc ,Pm):
    population_size = len(Selected_Population)
    new_population = np.empty((population_size , 8 ) , dtype=int)
    for i in range (0 , population_size , 2):
        parent1 = Selected_Population[i]
        parent2 = Selected_Population[i+1]
        child1 , child2 = Two_Point_Crossover(parent1 , parent2 , Pc)
        new_population[i] = child1
        new_population[i+1] = child2
    for i in range (population_size ):
         Flipping_Mutation(new_population[i] , Pm)
    return new_population
```

```python
# #-------------------------------------------------------------
def Eight_Queen_Problem(population_size, Max_Iteration, Pc=0.70, Pm=0.01):
    Population = initial_Population(population_size)
    # print('initial population: \n', Population, '\n')
    best_fitness_overall = None
    for i in range(Max_Iteration):
        fitness_vals = Fitness_Function(Population)
        best_index = fitness_vals.argmax()
        best_fitness = fitness_vals[best_index]
        if best_fitness_overall is None or best_fitness > best_fitness_overall:
            best_fitness_overall = best_fitness
            best_solution = Population[best_index]
        print(f'\rgeneration = {i:06}   best_fitness = {best_fitness_overall:.3f}', end='')
        if best_fitness == 28:
            print('\nfound best solution')
            break
        Selected_Population = Roulette_Wheel_Selection(Population, fitness_vals)
        Population = Crossover_Mutation(Selected_Population, Pc, Pm)
    return best_solution

# #-------------------------------------------------------------
#board
def print_board(chrom):
    board = []

    for x in range(8):
        board.append(["[ ]"] * 8)

    for i in range(8):
        board[chrom[i]][i] = "[Q]"

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

    print()
    print_board(board)
# #-------------------------------------------------------------
Initial_Population = initial_Population(4)
print('initial populatoin => \n')
print(Initial_Population )

Fitness_Values = Fitness_Function(Initial_Population)
print('Fitness values => \n')
print(Fitness_Values)

Selection = Roulette_Wheel_Selection(Initial_Population,Fitness_Values)
print('selection => \n')
print(Selection)
```

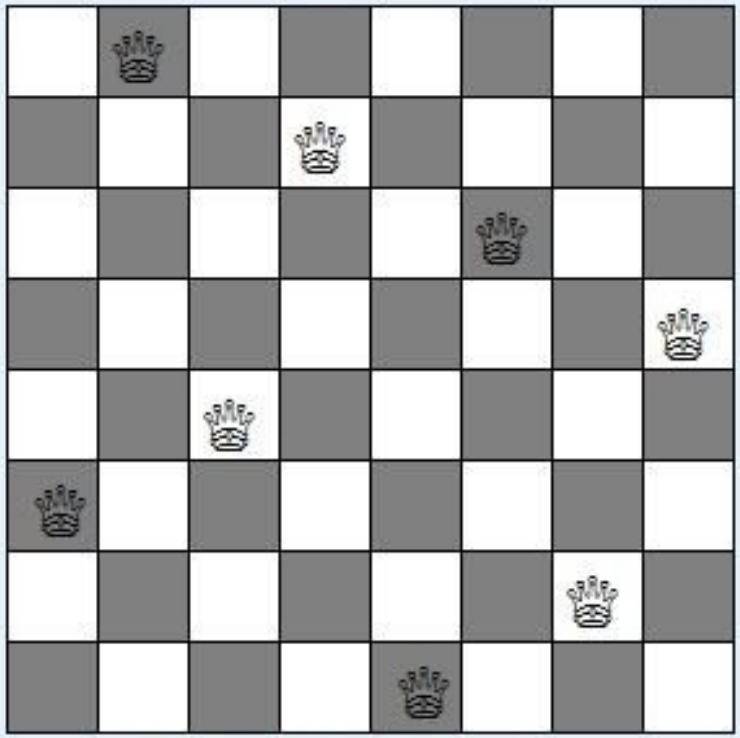
```python
# parent1=Selection[0]
# parent2=Selection[1]
# child1 ,child2 = Two_Point_Crossover(parent1 ,parent2 , Pc=0.70)
# print('crossover')
# print(parent1,'-->',child1)
# print(parent2,'-->',child2)

#child  = uniform_crossover(parent1 ,parent2 , pc=0.70)
#print(parent1,'--->',child)

# mut=Flipping_Mutation(Selection[0],0.01)
# print('mutation=>',mut)

# crosmut=Crossover_Mutation(Selection,0.70,0.01)
# print('crossover and mutation')
# print("new population--> \n",crosmut)



solution = Eight_Queen_Problem(population_size=90, Max_Iteration=1000, Pc=0.7, Pm=0.01)
print('\nbest solution: ', solution)

board=print_board(solution)
print(board)

# #graphic

tu = Turtle()
tu.screen.bgcolor("#e3f2fd")

tu.speed(0)

tu.home()

def draw():
    for i in range(4):
        tu.forward(35)
        tu.left(90)

    tu.forward(35)
if __name__ == "__main__":
    for j in range(8):
        tu.up()
        tu.setpos(-100, 35 * j)
        tu.down()

        for k in range(8):
            if (j + k) % 2 == 0:

                color1 = 'gray'

            else:
                color1 = 'white'
            tu.fillcolor(color1)
            tu.begin_fill()
            draw()
            tu.end_fill()
    tu.up()
    tu.pencolor("black")
    tu.goto(-50,-120)
    tu.write("\n Alaa Atef safan  \n Aya Sabry El Sorady \n Rewan Ahmed Abdelghfar \n Sara Abd El-Kader Mohamed \n Maryam Jamal Dawood"
             ,font=("Arial", 15))
    # tu.hideturtle()
    tu.up()
    tu.goto(50,0)
    tu.pencolor("black")
    tu.write("♕" ,font=("Arial", 18))

    tu.up()
    tu.goto(115,35)
    tu.pencolor("black")
    tu.write("♕" ,font=("Arial", 18))

    tu.up()
    tu.goto(-90,70)
    tu.pencolor("black")
    tu.write("♕" ,font=("Arial", 18))

    tu.up()
    tu.goto(-25,103)
    tu.pencolor("black")
    tu.write("♕" ,font=("Arial", 18))

    tu.up()
    tu.goto(150,138)
    tu.pencolor("black")
    tu.write("♕" ,font=("Arial", 18))

    tu.up()
    tu.goto(80,175)
    tu.pencolor("black")
    tu.write("♕" ,font=("Arial", 18))

    tu.up()
    tu.goto(10,210)
    tu.pencolor("black")
    tu.write("♕" ,font=("Arial", 18))

    tu.up()
    tu.goto(-60,245)
    tu.pencolor("black")
    tu.write("♕" ,font=("Arial", 18))
    tu.up()
    tu.home()
    tu.goto(-300,0)
    tu.pencolor("blue")
    tu.write("initial population\n" ,font=("Arial", 18))
    tu.write("[5,0,4,1,7,2,6,3]" ,font=("Arial", 18))
    done()
```

OUTPUT DEBUG CONSOLE TERMINAL

```
N/Desktop/project_final.py
initial populatoin =>

[[2 7 4 2 5 5 3 1]
 [0 6 1 1 7 5 1 6]
 [0 5 2 6 6 2 6 0]
 [6 1 4 2 6 5 5 5]]
Fitness values =>

[23 21 19 19]
selection =>

[[0 5 2 6 6 2 6 0]
 [2 7 4 2 5 5 3 1]
 [0 6 1 1 7 5 1 6]
 [0 5 2 6 6 2 6 0]]
generation = 000216   best_fitness = 28.000
found best solution

best solution:  [3 1 7 4 6 0 2 5]

[ ][ ][ ][ ][ ][Q][ ][ ]
[ ][Q][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][Q][ ]
[Q][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][Q][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][Q]
[ ][ ][ ][ ][Q][ ][ ][ ]
[ ][ ][Q][ ][ ][ ][ ][ ]
```

Python Turtle Graphics

initial population
[5,0,4,1,7,2,6,3]



Alaa Atef safan
Aya Sabry El Sorady
Rewan Ahmed Abdelghfar
Sara Abd El-Kader Mohamed
Maryam Jamal Dawood