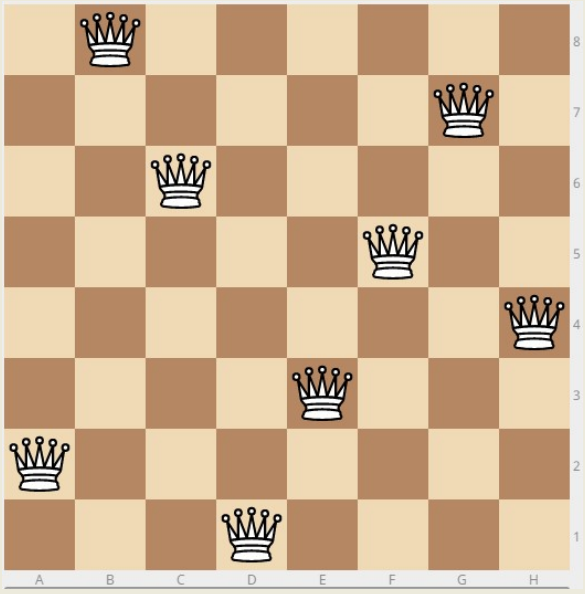
CE_PRACTICA_02	Resolver el problema de ajedrez de las 8 damas	CE

El problema consiste en colocar 8 reinas en un tablero de ajedrez (8x8) cumpliendo la condición de que no se puedan "comer" entre ellas como se muestra en la siguiente figura.



Mediante computación evolutiva se busca la solución de mejor resultado (fitness)

SOLUCION

El pseudocódigo principal está en el método "algoritmo", de la clase Agenetico en el código.

```
def algoritmo(self, poblacion):

  while True:
      npobla=[]
      for i in range(len(poblacion)):
          x=self.seleccion(poblacion)
          y=self.seleccion(poblacion)
          hijo=self.reproducir(x, y)
          if rnd.randint(1, 20)<3:
              hijo=self.mutar(hijo)
          npobla.append(hijo)
          if npobla[i].fitness>=(self.casillas*self.casillas-self.casillas):
              return npobla[i]
      poblacion=npobla
```

Inicialmente creamos una población aleatoria y vamos modificándola en base a la función reproducir y aleatoriamente mutamos un gen del individuo hijo.

De esta forma, generamos npobla (otro array) donde almacenamos la nueva población, evaluamos la idoneidad (fitness) del individuo actual y devolvemos el individuo idóneo (solución válida) para posteriormente finalizar el proceso al mostrar la solución.

Las soluciones que encontramos al problema de las 8 reinas pueden variar, pero en nuestra ejecución son las siguientes:

```
. . R .
R . . .
. . . R
. R . .
fitness: 12  0.020
. . R . .
R . . . .
. . . R .
. R . . .
. . . . R
fitness: 20  0.028
. . R . . .
. . . . . R
. R . . . .
. . . . R .
R . . . . .
. . . R . .
fitness: 30  2.270
. . . R . . .
R . . . . . .
. . . . R . .
. R . . . . .
. . . . . R .
. . R . . . .
. . . . . . R
fitness: 42  0.059
. . R . . . . .
. . . . . R . .
. . . . . . . R
R . . . . . . .
. . . R . . . .
. . . . . . R .
. . . . R . . .
. R . . . . . .
fitness: 56  0.069
. . . . . . . . R
. . . R . . . . .
. . . . . R . . .
. . . . . . . R .
. R . . . . . .
. . . . . R . .
R . . . . . . .
. . R . . . . .
. . . . R . . . .
fitness: 72  862.279
```

Variando la ejecución desde 4 hasta 9 casillas, se observa que para 9 casillas se dispara el tiempo de procesado siendo necesaria una mayor optimización del código.

CÓDIGO EN PYTHON

```
import random as rnd
import time as tm

class Cromosoma:
    def __init__(self, genes, fitness):
        self.genes=genes
        self.fitness=fitness

    def len(self):
        return len(self.genes)

    def __str__(self):
        s="%s %d" % (" ".join(map(str, self.genes)), self.fitness)
        s2=[]
        n=len(self.genes)
        for i in range(n):
            s2.append(list("."*n))
            for k in range(n):
                if self.genes[k]==i:
                    s2[i][k]="R"
            s2[i]=" ".join(s2[i])

        return "\n".join(s2)+"\nfitness: " + str(self.fitness)

class AGenetico:
    def __init__(self, casillas):
        self.casillas=casillas
        self.genes=range(casillas)

    def diagonales(self, gen, posicion):
        genesDiagonal1=[]
        genesDiagonal2=[]
        px=posicion+1
        py=gen+1
        for x in range(self.casillas):
            if x==posicion:
                genesDiagonal1.append(gen)
                genesDiagonal2.append(gen)
                continue
            y1 = (x-posicion)*((py-gen)/(px-posicion))+gen
            y2 = (x-posicion)*((py-gen)/((posicion-1)-posicion))+gen
            if y1>=0 and y1<self.casillas and y1!=gen:
                genesDiagonal1.append(y1)
            else:
                genesDiagonal1.append(-1)
            if y2>=0 and y2<self.casillas and y2!=gen:
                genesDiagonal2.append(y2)
            else:
                genesDiagonal2.append(-1)
        return genesDiagonal1, genesDiagonal2
```

```
def fidoneidad(self, genes):
    fitness=0
    i=0
    for k in range(len(genes)):
        diagonal1, diagonal2 = self.diagonales(genes[k], k)
        for i in range(len(genes)):
            if genes[k]==genes[i]:   # reina en horizontal
                continue
            elif (len(diagonal1)>i and genes[i]==diagonal1[i]) or (len(diagonal2)>i
            and genes[i]==diagonal2[i]): # reina en diagonal
                continue
            else:
                fitness+=1
    return fitness

def newParent(self):
    genes=rnd.sample(self.genes, self.casillas)
    fitness=self.fidoneidad(genes)
    return Cromosoma(genes, fitness)

def reproducir(self, x, y):
    c=rnd.randint(0, x.len())
    genes=x.genes[:c]+y.genes[c:]
    fitness=self.fidoneidad(genes)
    return Cromosoma(genes, fitness)

def mutar(self, hijo):
    n=rnd.randint(0, hijo.len()-1)
    genes=hijo.genes
    genes[n]=-1
    genX=genY=max(genes)
    while genX in genes and genY in genes:
        genX, genY = rnd.sample(self.genes, 2)
    genes[n]= genX if genX in genes else genY
    hijo.fitness=self.fidoneidad(genes)
    hijo.genes=genes
    return hijo

def seleccion(self, poblacion):
    if len(poblacion)>0:
        return rnd.choice(poblacion)
    else:
        return self.newParent()
```

```
    def algoritmo(self, poblacion):

        while True:
            npobla=[]
            for i in range(len(poblacion)):
                x=self.seleccion(poblacion)
                y=self.seleccion(poblacion)
                hijo=self.reproducir(x, y)
                if rnd.randint(1, 20)<3:
                    hijo=self.mutar(hijo)
                npobla.append(hijo)
                if npobla[i].fitness>=(self.casillas*self.casillas-self.casillas):
                    return npobla[i]
            poblacion=npobla

    def run(self):
        poblacion=[]
        for i in range(350):
            poblacion.append(self.newParent())
            print("Individuo",i)
            print(poblacion[i])

        print()
        print("Solución al problema de las DAMAS con tablero de ", self.casillas,"
        x ",self.casillas," casillas")
        print(self.algoritmo(poblacion)),

# PROGRAMA PRINCIPAL

for n in [4, 5, 6, 7, 8, 9]:
    t=tm.time()
    alg=AGenetico(n)
    alg.run()

print("Tiempo de Ejecución n=",n," %02.03f" % (tm.time()-t)," segundos")
```