

INSTITUTO FEDERAL CATARINENSE - IFC

RAFAEL JOSÉ CAMARGO BEKHAUSER

8 RAINHAS: Aplicação de algoritmos genéticos

Rio do Sul, 2022

RAFAEL JOSÉ CAMARGO BEKHAUSER

8 RAINHAS: Aplicação de algoritmos genéticos

Relatório técnico apresentado como requisito parcial para obtenção de aprovação na disciplina Inteligência Artificial, no Curso Bacharel em Ciência da Computação, no Instituto Federal Catarinense.

Prof. Me. Daniel Gomes Soares

Rio do Sul, 2022

RESUMO

Este presente relatório técnico visa apresentar um modelo de solução para o problema das 8 rainhas, por meio do uso do método de inteligência artificial, os algoritmos genéticos. Logo, serão feitas maiores explicações acerca da problemática e sua complexidade, juntamente com o método de algoritmos genéticos, abordando desta forma um modelo de solução ao problema e sua codificação em uma linguagem de programação.

Palavras-chave: Algoritmos Genéticos. Inteligência artificial. Problema das 8 rainhas.

SUMÁRIO

1	INTRODUÇÃO.....	4
2	DESENVOLVIMENTO.....	5
2.1	OBJETIVO GERAL.....	5
2.1.1	Objetivos específicos.....	5
2.2	METODOLOGIA.....	6
2.3	RESULTADOS.....	10
3	CONCLUSÕES	13
	APÊNDICE A – Código fonte do algoritmo.....	14
	REFERÊNCIAS.....	18

1 INTRODUÇÃO

Os avanços na ciência e na tecnologia aproximaram cada vez mais o desenvolvimento de computadores capazes de realizar tarefas que somente os humanos podem realizar hoje. A inteligência artificial é o ramo da ciência da computação que busca entender como a mente funciona, permitindo que os computadores raciocinam de maneira semelhante.

Levine (1988, p. 3) define a Inteligência Artificial como um método de fazer um computador pensar de forma inteligente. Ou seja, simula o funcionamento da mente humana para que os computadores tomem decisões e resolvem problemas. Existem diversas técnicas e modelos de inteligência artificial, cada qual com sua característica, o presente relatório visa a aplicação da técnica de algoritmos genéticos.

Segundo Rosa (2009), Algoritmo Genético (AG) é uma técnica de Inteligência Artificial que busca otimizar a solução de problemas complexos. Esta técnica tem seus conceitos e fundamentos na genética e nas teorias de evolução e de seleção natural de Charles Darwin.

Em suma a técnica consiste na análise evolucionar de uma população de indivíduos, logo geramos uma população com n indivíduos, onde cada um carrega consigo uma carga genética fenotípica (características/informações que podem vir a ser uma solução ao problema analisado), enquanto não for encontrado um indivíduo que contenha uma solução satisfatória é feito o cruzamento entre os indivíduos, o cruzamento é feito de dois em dois e gera outros dois indivíduos, gerando assim uma nova população que permanece neste ciclo até atingir o objetivo.

Cada indivíduo gerado é avaliado por uma função fitness, que tem por objetivo definir quão próximo da solução, ou quão “bom” ele é. Similar a natureza os indivíduos de um algoritmos genéticos estão propensos a sofrerem de mutação em seus cromossomos.

2 DESENVOLVIMENTO

O problema das 8 rainhas é um problema combinatório exponencial envolvendo a colocação de 8 rainhas sem colisões no tabuleiro de xadrez. Sendo considerado uma colisão quando duas ou mais rainhas estão na mesma linha, coluna ou diagonal.

Com base em um tabuleiro de xadrez 8X8, é preciso encontrar uma solução para que as oito rainhas desse tabuleiro obedeçam às regras propostas pelo problema. Podemos expressar matematicamente a quantidade total de possibilidades como sendo:

$$C_{64,8} = \frac{64!}{(8! \cdot (64-8)!)} = \frac{64!}{(8! \cdot 56!)} = 4426165368$$

Se reduzirmos o espaço de busca tendo em vista que as rainhas ocupam diferentes colunas temos:

$$C_8 = 8^8 = 1677216$$

Logo, existem 4.426.165.368 arranjos possíveis totais, onde destes considerando a premissa de que as rainhas ocupam diferentes colunas temos 1.677.216 possibilidades. Destes, apenas 92 conseguem atingir a solução do problema apresentando o estado de zero colisões.

2.1 OBJETIVO GERAL

Aplicar o método de algoritmos genéticos, na resolução do problema das 8 rainhas, visando a solução ideal de zero colisões

2.1.1 Objetivos específicos

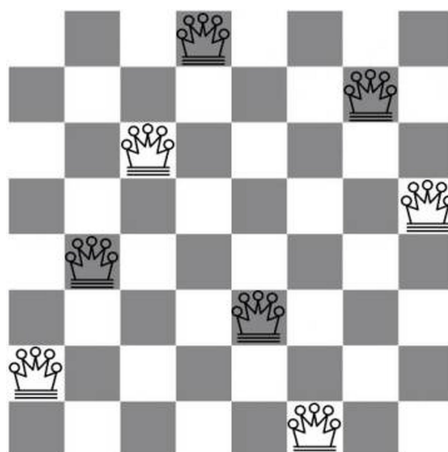
1. Determinar um modelo de solução com algoritmos genéticos, utilizando a linguagem de programação Python;
2. Analisar quais os melhores ajustes inerentes ao problema.

2.2 METODOLOGIA

Em um primeiro momento podemos definir alguns parâmetros necessários ao funcionamento do algoritmo, como sendo, um número de indivíduos que nosso algoritmo irá gerar, um número máximo de gerações (subentende-se geração como sendo o resultado da população resultante de um cruzamento de indivíduos, esta constante tem por objetivo prevenir que o algoritmo não atinja um *loop* infinito), a taxa de mutação da população, bem como a taxa de cruzamento.

No caso do problema das 8 rainhas, existe um número de estados possíveis para cada rainha, analisando o tabuleiro podemos afirmar que existem 8 colunas e 8 posições, logo desta forma tendo em vista as condições que satisfazem o problema teremos apenas uma rainha por linha, logo podemos enumerar as colunas indo de 1 até 8, portanto uma rainha pode assumir 8 estados possíveis dentro de uma linha, outrossim o mesmo se aplica se invertermos onde podemos ter apenas 1 rainha por coluna e a mesma possui 8 linhas possíveis.

Figura 1 - Tabuleiro 8x8, com 8 rainhas dispostas.

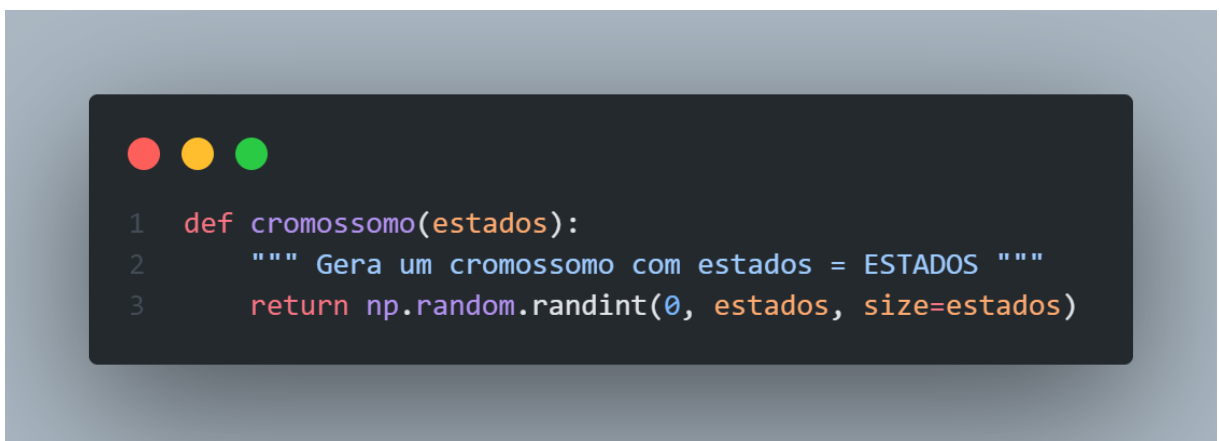


Fonte: Isabel Rubio, 2019.

Cada indivíduo contém um cromossomo, este qual carrega a informação genética do mesmo, ou seja uma possível solução ao problema. Para representar um cromossomo em termos de programação, podemos definir um vetor que irá receber os estados possíveis definidos. Logo desta forma, podemos definir um indivíduo como sendo um vetor de 8 posições, onde cada posição recebe um estado possível aleatoriamente (no caso, indo de 0 até 7).

Por meio do uso da biblioteca numpy do Python podemos escrever uma função que nos retorna um array com tamanho igual a quantidade de estados, onde cada posição recebe um valor inteiro aleatório entre 0 e a quantidade de estados.

Figura 2 - Representação do cromossomo.

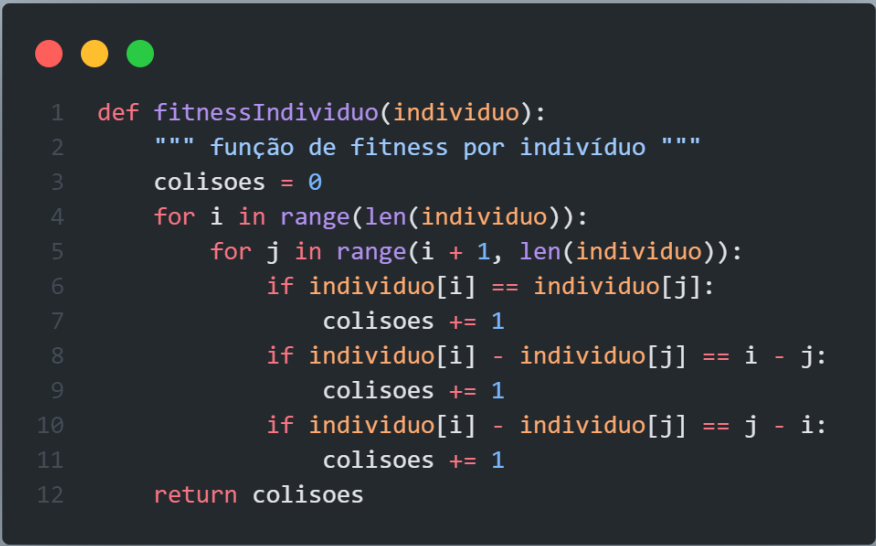
A screenshot of a code editor window with a dark background and light-colored text. The window has three colored window control buttons (red, yellow, green) in the top-left corner. The code is a Python function definition for a chromosome, using the numpy library. The function is named 'cromossomo' and takes 'estados' as an argument. It includes a docstring and returns a numpy array of random integers.

```
1 def cromossomo(estados):  
2     """ Gera um cromossomo com estados = ESTADOS """  
3     return np.random.randint(0, estados, size=estados)
```

Fonte: De autoria própria.

Como forma de avaliar cada indivíduo e definir o quão próximo ele está da resolução que satisfaça o problema, logo utilizamos de uma função fitness. O fitness é calculado tendo como base o número de colisões/ataques que cada rainha tem sobre as demais. Levando em consideração que as rainhas não podem estar na mesma linha, coluna ou diagonal. Para que o indivíduo seja considerado como perfeito sendo a solução, o fitness deve ser 0.

Figura 3 - Função de avaliação fitness.



```
1 def fitnessIndividuo(individuo):
2     """ função de fitness por indivíduo """
3     colisoes = 0
4     for i in range(len(individuo)):
5         for j in range(i + 1, len(individuo)):
6             if individuo[i] == individuo[j]:
7                 colisoes += 1
8             if individuo[i] - individuo[j] == i - j:
9                 colisoes += 1
10            if individuo[i] - individuo[j] == j - i:
11                colisoes += 1
12    return colisoes
```

Fonte: De autoria própria.

Após definidos os indivíduos de uma população e não encontrado um, tal qual possua um fitness igual a zero, se faz necessário realizar uma seleção entre os indivíduos para que, a cada dois, sejam cruzados a fim de gerar dois novos indivíduos com possíveis soluções ao problema. Para cada indivíduo selecionado atribuímos o valor de fitness como sendo 1000, para que não sejam escolhidos novamente para cruzamento.

No modelo proposto a seleção dos indivíduos pais ocorre de forma elitista tendo como base o valor do fitness de cada um, ou seja, os indivíduos com fitness menor são selecionados para cruzamento.

O cruzamento ou crossover, ocorre de acordo com a taxa de cruzamento definida, ou seja sorteamos aleatoriamente um número entre 0,0 e 1,0, caso o número gerado seja menor que a taxa de cruzamento, os pais terão seus cromossomos divididos por um ponto de corte (escolhido aleatoriamente, dentro do tamanho do indivíduo, indo de 1 até a quantidade de estados). Após feito o corte entre ambos, as partes são mescladas, formando dois novos indivíduos, intitulados filhos. Caso o número gerado seja maior que a taxa de cruzamento, os pais são mantidos para a próxima geração.

Figura 4 - Função de seleção e cruzamento de indivíduos.

```

1  def selecaoCrossover(populacao):
2      fit = fitness(populacao)
3      populacao = np.array(populacao)
4
5      pais = []
6      for i in range(len(populacao)):
7          pais.append(populacao[np.argmin(fit)])
8          fit[np.argmin(fit)] = 1000
9      pais = np.array(pais)
10
11     filhos = []
12     for i in range(0, len(pais), 2):
13         if np.random.random() < TAXA_CRUZAMENTO:
14             ponto_corte = np.random.randint(1, ESTADOS)
15             filho1 = np.concatenate((pais[i, :ponto_corte], pais[i + 1, ponto_corte:]))
16             filho2 = np.concatenate((pais[i + 1, :ponto_corte], pais[i, ponto_corte:]))
17             filhos.append(filho1)
18             filhos.append(filho2)
19         else:
20             filhos.append(pais[i])
21             filhos.append(pais[i + 1])
22
23     return filhos

```

Fonte: De autoria própria.

Após realizada a geração de uma nova população, os indivíduos que a compõem sofrem um processo de mutação cromossômica aleatória, ou seja, sorteamos aleatoriamente um número entre 0,0 e 1,0, caso este seja menor que a taxa de mutação definida, o indivíduo, sofrerá uma alteração em um de seus estados escolhido aleatoriamente, recebendo um novo valor gerado aleatoriamente, dentro dos estados possíveis.

Figura 5 - Representação de mutação no cromossomo.

```

1  def mutacao(populacao):
2      for i in range(len(populacao)):
3          if np.random.random() < MUTACAO:
4              populacao[i][np.random.randint(0, ESTADOS)] = np.random.randint(0, ESTADOS)
5      return populacao

```

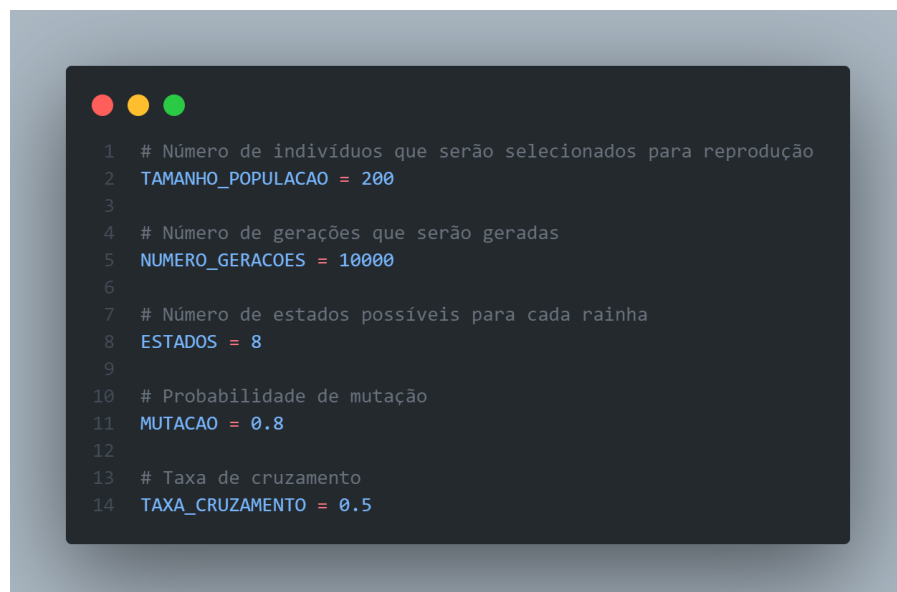
Fonte: De autoria própria.

Em suma podemos definir o processo de um algoritmo genético como sendo a geração de uma população inicial de indivíduos, que serão selecionados e cruzados, sofrendo uma possível mutação em seus cromossomos. Se ao final deste processo não forem encontrados indivíduos capazes de satisfazer a condição do problema, o ciclo é prosseguido a partir da geração anterior de indivíduos, até atingir um ponto de parada.

2.3 RESULTADOS

Para a realização dos testes definimos como constantes do algoritmo, os seguintes parâmetros, conforme imagem abaixo.

Figura 6 - Definição inicial das constantes de controle.



```
1 # Número de indivíduos que serão selecionados para reprodução
2 TAMANHO_POPULACAO = 200
3
4 # Número de gerações que serão geradas
5 NUMERO_GERACOES = 10000
6
7 # Número de estados possíveis para cada rainha
8 ESTADOS = 8
9
10 # Probabilidade de mutação
11 MUTACAO = 0.8
12
13 # Taxa de cruzamento
14 TAXA_CRUZAMENTO = 0.5
```

Fonte: De autoria própria.

Figura 7 - Resultados iniciais.

```

Teste 1
Numero de geracoes: 3261
Numero de individuos gerados: 326100
Melhor individuo: [0 4 7 5 2 6 1 3]
Fitness do melhor individuo: 0
--- 23.63 seconds ---

-----

Teste 2
Numero de geracoes: 10000
Numero de individuos gerados: 1000000
Melhor individuo: [0 3 6 7 2 6 1 7]
Fitness do melhor individuo: 4
--- 76.82 seconds ---

```

Fonte: De autoria própria.

Após realizar a execução do algoritmo com tais índices, obtivemos como resultados a solução do problema. Porém, podemos notar que o segundo teste não chegou a uma resposta perfeita (como fitness igual a zero), outro detalhe em ambos se diz quanto ao número de indivíduos gerados e tempo de execução total do algoritmo. Ao analisar os parâmetros definidos, faremos algumas alterações, no caso, aumentando a taxa de cruzamento, reduzindo a mutação e gerando menos indivíduos por população, portanto teremos:

Figura 8 - Constantes de controle ajustadas.

```

1  # Número de indivíduos que serão selecionados para reprodução
2  TAMANHO_POPULACAO = 100
3
4  # Número de gerações que serão geradas
5  NUMERO_GERACOES = 10000
6
7  # Número de estados possíveis para cada rainha
8  ESTADOS = 8
9
10 # Probabilidade de mutação
11 MUTACAO = 0.5
12
13 # Taxa de cruzamento
14 TAXA_CRUZAMENTO = 0.7

```

Fonte: De autoria própria.

Figura 9 - Resultados aprimorados.

```

Teste 1
Numero de geracoes: 315
Numero de individuos gerados: 31500
Melhor individuo: [1 3 5 7 2 0 6 4]
Fitness do melhor individuo: 0
--- 2.29 seconds ---

-----

Teste 2
Numero de geracoes: 415
Numero de individuos gerados: 41500
Melhor individuo: [3 1 4 7 5 0 2 6]
Fitness do melhor individuo: 0
--- 2.98 seconds ---

-----

Teste 3
Numero de geracoes: 1500
Numero de individuos gerados: 150000
Melhor individuo: [1 7 5 0 2 4 6 3]
Fitness do melhor individuo: 0
--- 10.87 seconds ---

-----

Teste 4
Numero de geracoes: 1076
Numero de individuos gerados: 107600
Melhor individuo: [0 6 4 7 1 3 5 2]
Fitness do melhor individuo: 0
--- 7.78 seconds ---

```

Fonte: De autoria própria.

Os resultados obtidos após os ajustes nas constantes de controle se mostram efetivos, visto que o número de indivíduos gerados e o tempo de processamento do algoritmo sofreram reduções, apresentando em todos os casos uma solução perfeita ao problema das 8 rainhas.

3 CONCLUSÕES

Com o estudo do método de algoritmos genéticos, observou-se a viabilidade do seu uso para resolver problemas complexos, seguindo modelos de aperfeiçoamento presentes na natureza, como a problemática em dispor 8 rainhas em um tabuleiro sem que nenhuma ataque as demais. Percebeu-se que a eficiência do algoritmo está diretamente ligada com seus parâmetros de controle, onde por meio destes é possível reduzir tempo e resultados insatisfatórios na implementação desta técnica de inteligência artificial.

APÊNDICE A – Código fonte do algoritmo.

```
import numpy as np

# Número de indivíduos que serão selecionados para reprodução
TAMANHO_POPULACAO = 100

# Número de gerações que serão geradas
NUMERO_GERACOES = 10000

# Número de estados possíveis para cada rainha
ESTADOS = 8

# Probabilidade de mutação
MUTACAO = 0.5

# Taxa de cruzamento
TAXA_CRUZAMENTO = 0.7

def cromossomo(estados):
    """ Gera um cromossomo com estados = ESTADOS """
    return np.random.randint(0, estados, size=estados)

def gerarPopulacaoInicial(tamanho_populacao, estados):
    """ Gera uma população inicial de tamanho = tamanho_populacao
    com estados = ESTADOS """
    populacao = []
    for i in range(tamanho_populacao):
        populacao.append(cromossomo(estados))
    return populacao

def fitnessIndividuo(individuo):
    """ função de fitness por indivíduo """
    colisoes = 0
    for i in range(len(individuo)):
        for j in range(i + 1, len(individuo)):
            if individuo[i] == individuo[j]:
                colisoes += 1
            if individuo[i] - individuo[j] == i - j:
                colisoes += 1
            if individuo[i] - individuo[j] == j - i:
```

```

        colisoos += 1
    return colisoos

def fitness(populacao):
    """
    Calcula o fitness de cada indivíduo da população

    O fitness é calculado tendo como base o número de
    colisões/ataques
    que cada rainha tem sobre as demais. Levando em consideração
    que
    as rainhas não podem estar na mesma linha, coluna ou diagonal,
    para que o individuo seja perfeito, o fitness deve ser 0.
    """
    fit = []
    for i in range(len(populacao)):
        fit.append(fitnessIndividuo(populacao[i]))
    return fit

def selecaoCrossover(populacao):
    """
    Seleciona os indivíduos que serão reproduzidos e realiza o
    crossover

    A seleção dos pais, ou seja os indivíduos que serão
    reproduzidos, é feita
    tendo como base o fitness de cada indivíduo. Os indivíduos com
    fitness menor
    são selecionados, após isto o fitness do indivíduo selecionado
    é atribuído o valor
    de 1000 para que ele não seja selecionado novamente. O
    crossover é realizado
    com base na taxa de cruzamento, e pelo ponto de corte (é
    escolhido aleatoriamente,
    dentro do tamanho do indivíduo) é feito o cruzamento dos
    indivíduos selecionados.

    Os filhos gerados são adicionados a uma nova população que será
    retornada.
    """
    fit = fitness(populacao)
    populacao = np.array(populacao)

```



```

pais = []
for i in range(len(populacao)):
    pais.append(populacao[np.argmin(fit)])
    fit[np.argmin(fit)] = 1000
pais = np.array(pais)

filhos = []
for i in range(0, len(pais), 2):
    if np.random.random() < TAXA_CRUZAMENTO:
        ponto_corte = np.random.randint(1, ESTADOS)
        filho1 = np.concatenate((pais[i, :ponto_corte], pais[i
+ 1, ponto_corte:]))
        filho2 = np.concatenate((pais[i + 1, :ponto_corte],
pais[i, ponto_corte:]))
        filhos.append(filho1)
        filhos.append(filho2)
    else:
        filhos.append(pais[i])
        filhos.append(pais[i + 1])

return filhos

def mutacao(populacao):
    """
    Realiza a mutação dos indivíduos da população
    tendo como base a probabilidade de mutação,
    logo se o número aleatório gerado (entre {0.0 e 1.0})
    for menor que a probabilidade de mutação, o indivíduo
    sofrerá mutação. Esta qual consiste em trocar o valor
    de um dos estados do indivíduo por um valor aleatório.
    """
    for i in range(len(populacao)):
        if np.random.random() < MUTACAO:
            populacao[i][np.random.randint(0, ESTADOS)] =
np.random.randint(0, ESTADOS)
    return populacao

def imprimeTabuleiro(individuo):
    """ Essa função imprime o tabuleiro especificado com as rainhas
    posicionadas """
    tabuleiro = np.zeros((ESTADOS, ESTADOS))

```

```

for i in range(len(individuo)):
    tabuleiro[individuo[i]][i] = 1

# trocar 1 por 🏰
tabuleiro = np.where(tabuleiro == 1, '🏰', tabuleiro)

# trocar 0 por 🟩
tabuleiro = np.where(tabuleiro == '0.0', '🟩', tabuleiro)

return tabuleiro

def estatisticas(geracao, melhorIndividuo):
    print('Numero de gerações: ', geracao)
    print('Numero de individuos gerados: ', geracao *
TAMANHO_POPULACAO)
    print('Melhor indivíduo: ', melhorIndividuo)
    print('Fitness do melhor indivíduo: ',
fitnessIndividuo(melhorIndividuo))
    print('Tabuleiro: ')
    print(imprimeTabuleiro(melhorIndividuo))

def algoritmoGenetico():
    """ Função principal do algoritmo genético """
    populacao = gerarPopulacaoInicial(TAMANHO_POPULACAO, ESTADOS)
    geracao = 0
    for i in range(NUMERO_GERACOES):
        filhos = selecaoCrossover(populacao)
        filhos = np.array(filhos)
        filhos = mutacao(filhos)
        populacao = filhos
        if np.min(fitness(populacao)) == 0:
            break
        # print("Geração: ", i)
        # melhorIndividuo =
populacao[np.argmin(fitness(populacao))]
        # print("Melhor indivíduo: ", melhorIndividuo)
        # print("Fitness do melhor indivíduo: ",
fitnessIndividuo(melhorIndividuo))
        geracao += 1
    return geracao, populacao[np.argmin(fitness(populacao))]

```

REFERÊNCIAS

LEVINE, Robert L.; DRANG, Diane E.; EDELSON, Barry. **Inteligência Artificial e Sistemas Especialistas**: Aplicações e Exemplos Práticos. São Paulo: McGraw-Hill, 1988.

ROSA, Alexsander: **Uma Solução para o Problema das N Rainhas usando Threads em Java**. Disponível em: <<http://www.inf.ufrgs.br/gppd/disc/cmp157/trabalhos/urcamp99-1/alexsand/LogicaRainha.html>>.