Simular una batalla entre grupos antagonistas utilizando Cadenas de Markov.

Utilizando python con numpy o pandas. Simular una batalla mortal entre distintos grupos enemigos utilizando cadenas de Markov y la generación de números aleatorios.

Realiza las correcciones y sigue las instrucciones generales.

1. Comienza estableciendo una matriz estocástica inicial generada de manera aleatoria cada vez que se corre el programa, asegúrate que sea estocástica.
2. Ir del estado 1 al 2 significa que un guerrero del grupo 1 mata instantáneamente a un guerrero del grupo 2. Cada entrada de la matriz representa qué tan probable es que un grupo ataque a otro. Por ejemplo, si el elemento [1][2] es 0.60, significa que hay un 60% de probabilidad que el grupo 1 ataque al 2.
3. La cantidad de guerreros por grupo debe ser generada de manera aleatoria.
4. Para decidir qué grupo ataca en cada turno, genera un número aleatorio uniforme por medio de la distribución multinomial. Por ejemplo, si se genera el número 2, éste será el grupo que atacará.
5. Cada que un grupo se queda sin guerreros, la matriz estocástica debe ser reconfigurada de manera automática.
6. El grupo que queda con guerreros vivos es el ganador.

Consideraciones:

* Solicita la cantidad de grupos. Genera la función main y el condicional de arranque.
* Detalla las funciones y pasos complejos con comentarios en español y simples de entender.
* Usa nombres de funciones y variables en español.
* Utiliza la técnica dividir y vencer para los problemas complejos del programa. De tal forma que el código sea limpio y legible sin cosas complejas.
* Un guerrero de un grupo no puede atacar a otro del mismo grupo. Ejemplo grupo 3 ataca a grupo 3. Esto no estaría permitido.

Para la salida del programa:

* Genera un archivo de texto con la siguiente información.
* Matriz inicial.
* Cantidad de guerreros por grupo.
* Cada que un ataque se lleve a cabo, muestra el grupo que ataca y a quién ataca. Muestra en cada turno cuántos guerreros tienen todos los grupos.
* Cada que un grupo muere, muestra un mensaje y la nueva matriz estocástica.
* Al final muestra el grupo ganador.

Deberás guardar la salida del programa en un archivo llamado output.txt.

A continuación, te muestro un ejemplo del archivo de ejemplo de un programa similar. Debes utilizar un formato similar a este ejemplo.

1 2 3

1 0.0 0.4 0.6

2 0.2 0.0 0.8

3 0.15 0.85 0.0

Number of warriors for each group

Group 1: 50

Group 2: 40

Group 3: 60

==================================

Group 1 attacked Group 2!

Number of warriors for each group

Group 1: 50

Group 2: 39

Group 3: 60

Group 3 attacked Group 1!

Number of warriors for each group

Group 1: 49

Group 2: 39

Group 3: 60

.

.

.

Group 1 attacked Group 2!

Number of warriors for each group

Group 1: 12

Group 3: 25

Group 2 is annihilated!

=================================

Reconfiguring stochastic matrix

1 3

1 0.0 1

3 1 0.0

Group 3 attacked Group 1!

Number of warriors for each group

Group 1: 11

Group 3: 25

.

.

.

Group 1 attacked Group 3!

Number of warriors for each group

Group 1: 3

Group 3: 0

Group 3 is annihilated!

===============================

Group 1 is the winner!

===============================

Traceback (most recent call last):

File "C:\Users\danie\OneDrive\Escritorio\metodos-y-simulacion\proyecto\_p3\index.py", line 93, in <module>

main()

File "C:\Users\danie\OneDrive\Escritorio\metodos-y-simulacion\proyecto\_p3\index.py", line 89, in main

atacar(matriz, grupos)

File "C:\Users\danie\OneDrive\Escritorio\metodos-y-simulacion\proyecto\_p3\index.py", line 10, in atacar

posibles\_victimas = np.where(matriz[atacante] > 0)[0]

~~~~~~^^^^^^^^^^

IndexError: index 2 is out of bounds for axis 0 with size 2

Agrega una consideración o validación para ese escenario. En el siguiente código.

import sys

import numpy as np

# Función que representa un ataque de un grupo a otro

def atacar(matriz, grupos):

    while np.count\_nonzero(grupos) > 1: # Mientras exista más de un grupo vivo

        # Seleccionamos aleatoriamente el grupo atacante de los grupos que aún tienen guerreros

        atacante = np.random.choice(np.where(grupos > 0)[0])

        # Seleccionamos las posibles víctimas del ataque (aquellos grupos a los que el grupo atacante puede atacar)

        posibles\_victimas = np.where(matriz[atacante] > 0)[0]

        if len(posibles\_victimas) == 0:  # Si no hay a quien atacar, se salta el turno

            continue

        # Seleccionamos aleatoriamente la víctima del ataque en base a las probabilidades de la matriz estocástica

        victima = np.random.choice(posibles\_victimas, p=matriz[atacante][posibles\_victimas])

        # Reducimos el número de guerreros del grupo víctima

        grupos[victima] -= 1

        # Si el grupo víctima ha sido aniquilado, reconfiguramos la matriz estocástica

        if grupos[victima] == 0:

            matriz = reconfigurar\_matriz(matriz, victima)

            print(f"Grupo {victima + 1} ha sido aniquilado!\n")

            print("Reconfigurando matriz estocastica\n")

            imprimir\_matriz(matriz)

        print(f"Grupo {atacante + 1} ataco al Grupo {victima + 1}!")

        print("Numero de guerreros por cada grupo")

        imprimir\_grupos(grupos)

    print("===============================\n")

    print("El ganador es el Grupo", np.where(grupos > 0)[0][0] + 1, "\n")

    print("===============================\n")

# Función para reconfigurar la matriz estocástica cuando un grupo es aniquilado

def reconfigurar\_matriz(matriz, indice):

    matriz = np.delete(matriz, indice, 0) # Eliminamos la fila correspondiente al grupo aniquilado

    matriz = np.delete(matriz, indice, 1) # Eliminamos la columna correspondiente al grupo aniquilado

    # Normalizamos las filas de la matriz para que sigan siendo estocásticas

    for i in range(len(matriz)):

        suma\_de\_fila = np.sum(matriz[i])

        # Si la suma total de la fila es diferente de cero, dividimos cada elemento por la suma total

        if suma\_de\_fila != 0:

            matriz[i] /= suma\_de\_fila

    return matriz

# Función para imprimir la matriz

def imprimir\_matriz(matriz):

    for i in range(len(matriz)):

        for j in range(len(matriz[i])):

            print(round(matriz[i][j], 2), end="\t")

        print()

# Función para imprimir los grupos

def imprimir\_grupos(grupos):

    for i in range(len(grupos)):

        print(f"Grupo {i + 1}: {grupos[i]}")

    print("==================================\n")

# Función para generar la matriz estocástica de forma aleatoria

def generar\_matriz\_aleatoria(n):

    matriz = np.random.random((n, n)) # Genera una matriz nxn con números aleatorios entre 0 y 1

    np.fill\_diagonal(matriz, 0) # Los guerreros de un grupo no pueden atacarse entre sí

    # Normalizamos las filas para que sean estocásticas

    matriz /= matriz.sum(axis=1, keepdims=True)

    return matriz

# Función para generar la cantidad de guerreros por grupo de forma aleatoria

def generar\_guerreros(n):

    guerreros = np.random.randint(5, 15, n) # Genera un número aleatorio de guerreros por grupo entre 5 y 15

    return guerreros

# Función principal

def main():

    # Número de grupos

    n = 3

    # Cambiamos la salida estándar al archivo output.txt

    #sys.stdout = open('output.txt', 'w')

    # Generamos los grupos y la matriz estocástica de forma aleatoria

    grupos = generar\_guerreros(n)

    matriz = generar\_matriz\_aleatoria(n)

    print("Matriz inicial:")

    imprimir\_matriz(matriz)

    print("Numero de guerreros por cada grupo")

    imprimir\_grupos(grupos)

    # Comenzamos la batalla

    atacar(matriz, grupos)

# Ejecutamos la función principal

if \_\_name\_\_ == "\_\_main\_\_":

    main()