Daniel Cu Sánchez
A01703613
Proyecto Final: Cadenas de Markov
Video:

https://drive.google.com/drive/folders/1h39IYJA0lg1mcB_e9bG1pHTJVXvPQOow?usp=sharing

Instrucciones:

Este proyecto se puede hacer máximo por parejas, pero los videos deben ser individuales.

Simulación de una batalla entre grupos antagonistas utilizando Cadenas de Markov.

Utilizando tu lenguaje de programación favorito, simularás una batalla mortal entre distintos grupos enemigos utilizando cadenas de Markov y la generación de números aleatorios.

Instrucciones generales.

- Comienza estableciendo una matriz estocástica inicial (dada por el usuario o generada de manera aleatoria, pero asegúrate que sea estocástica). 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. Obviamente un guerrero no puede atacar a otro del mismo grupo.
- La cantidad de guerreros por grupo puede ser dada por el usuario o generada de manera aleatoria.
- Para decidir qué grupo ataca en cada turno, genera un número aleatorio uniforme. Por ejemplo, si se genera el número 2, éste será el grupo que atacará. Para decidir a qué grupo va a atacar, utiliza el método visto en clase.
- Cada que un grupo se queda sin guerreros, la matriz estocástica debe ser reconfigurada de manera automática.
- El grupo que queda con guerreros vivos es el ganador.

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.

(El archivo puede ser similar a este: output-1.txt

Calificaciones

1) **(85 puntos)** Tu simulación funciona con N = 3 grupos.

2) **(100 points)** Tu simulación funciona con N grupos.

Entregables: Código fuente. Archivos de texto generados por tu programa (***por lo menos 3 archivos, y si decidiste hacerlo para N grupos*** cambia la cantidad de grupos en cada caso). Enlace a una carpeta de Goggle Drive con videos individuales, explicando de manera general el código, pero explicando a detalle la parte de cómo usan la matriz estocástica para determinar el grupo ha ser atacado. El video es obligatorio para tener calificación.

Código solución del proyecto:

```
import sys
import numpy as np

# Simula el ataque de un grupo a otro
def atacar(matriz, grupos):
    # Creamos un vector para registrar el número original de cada grupo
    grupo_numeros = np.arange(len(grupos)) + 1
    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 y tienen posibles víctimas
        posibles_atacantes = np.where((grupos > 0) & (np.sum(matriz > 0, axis=1) > 0))[0]


        if len(posibles_atacantes) == 0:  # Si no hay posibles atacantes, se sale del bucle
            break
        atacante = np.random.choice(posibles_atacantes)
        # Seleccionamos las posibles víctimas del ataque (aquellos grupos a los que el grupo
atacante puede atacar)
        posibles_victimas = np.where(matriz[atacante] > 0)[0]
```

Daniel Cu Sánchez
A01703613
Proyecto Final: Cadenas de Markov

```
        # 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
        print(f"Grupo {grupo_numeros[atacante]} ataco al Grupo {grupo_numeros[victima]}!")

        # Si el grupo víctima ha sido aniquilado, reconfiguramos la matriz estocástica y el
vector de grupos
        if grupos[victima] == 0:
            print(f"Grupo {grupo_numeros[victima]} ha sido aniquilado!\n")
            matriz = reconfigurar_matriz(matriz, victima)
            grupos = np.delete(grupos, victima)  # Elimina la entrada correspondiente al grupo
aniquilado
            grupo_numeros = np.delete(grupo_numeros, victima)  # Actualiza el vector de
números de grupo
            print("Reconfigurando matriz estocastica\n")
            imprimir_matriz(matriz)
        print("Numero de guerreros por cada grupo")
        imprimir_grupos(grupos)
    print("================[Fin de batalla]==============\n")
    ganador = np.where(grupos > 0)[0]
    if ganador.size > 0:
        print("El ganador es el grupo", grupo_numeros[ganador[0]], "\n")




# Reconfigura 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



# 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("\n")
```

Daniel Cu Sánchez
A01703613
Proyecto Final: Cadenas de Markov

```
    print("================[Matriz actualizada]================\n")

# Imprimir los grupos
def imprimir_grupos(grupos):
    for i in range(len(grupos)):
        print(f"Grupo {i + 1}: {grupos[i]}")
    print("================[Grupos actualizados]================\n")


# Genera la matriz estocástica de forma aleatoria
def crear_matriz_aleatoria(n):
    matriz = np.random.random((n, n)) # Crea la matriz con numeros aleatorios
    np.fill_diagonal(matriz, 0) # Previene el ataca entre grupos
    # Normalizamos las filas para que sean estocásticas
    matriz /= matriz.sum(axis=1, keepdims=True)
    return matriz

# Genera la cantidad de guerreros por grupo de forma aleatoria
def generar_guerreros(n):
    guerreros = np.random.randint(5, 15, n) # Guerreros por grupo entre 5 y 15
    return guerreros

# Función principal del programa
def main():
    # Número de grupos
    n = int(input("Ingresa a cantidad de grupos 🥷: "))

    # Generamos los grupos y la matriz estocástica de forma aleatoria
    grupos = generar_guerreros(n)
    matriz = crear_matriz_aleatoria(n)

    # Permite enviar a un texto la salida por consola
    sys.stdout = open('output.txt', 'w')
    print("========= Matriz inicial =========")
    imprimir_matriz(matriz)
    print("========= Numero de guerreros inicial  =========")
    imprimir_grupos(grupos)

    # Esta función permite simular la batalla. ⚔
    atacar(matriz, grupos)

# Condicional de arranque
if __name__ == "__main__":
    main()
```

Descripción:

Este proyecto se divide en 6 funciones de operación y 1 de arranque. En total 7.

La función main llama en orden secuencial la ejecución del programa.

Daniel Cu Sánchez
A01703613
Proyecto Final: Cadenas de Markov
Función atacar: Simula el ataque de un grupo a otro.

Función reconfigurar_matriz: Reconfigura la matriz estocástica cuando un grupo es aniquilado.

Función imprimir_matriz: Imprime la matriz.

Función imprimir_grupos: Imprime los grupos.

Función crear_matriz_aleatoria: Genera la matriz estocástica de forma aleatoria.

Función generar_guerreros: Genera la cantidad de guerreros por grupo de forma aleatoria

Conclusión:

En este proyecto, me enfrenté a diversos retos trabajando con simulaciones y matrices estocásticas, tales como errores de índice, bucles infinitos y asegurar la aleatoriedad del proceso. Aprendí que estos desafíos se pueden superar con un enfoque metódico y persistente, corrigiendo los problemas uno por uno hasta obtener un código que funcioné como se desea.

Me di cuenta de que, en la programación, los errores son parte del camino, pero con paciencia y perseverancia, se pueden manejar de manera efectiva. Entendí la importancia de validar los resultados y corregir los bugs a medida que aparecen. Al final, logré desarrollar una simulación de batalla que funcionó correctamente, lo que demuestra que, con esfuerzo y dedicación, es posible crear sistemas complejos y eficientes. Sin duda, estos aprendizajes me serán de gran utilidad en mi futuro como ingeniero.