Tecnológico Nacional de México

Instituto Tecnológico de La Laguna

Inteligencia Artificial

INGENIERÍA EN SISTEMAS COMPUTACIONALES

Proyecto de la unidad 4 (Algoritmos Genéticos)

Presenta:

- Fernando Pérez Romero # 19130959

Torreón, Coahuila

Domingo 12 de Noviembre de 2023.

Contenido

1.Introducción.

Los algoritmos genéticos (AG) son poderosas herramientas de optimización y

búsqueda heurística inspiradas en los procesos de evolución natural. Derivados de la teoría de la evolución de Darwin, los AG imitan la selección natural, la recombinación genética y la mutación para evolucionar soluciones óptimas a problemas complejos.

En nuestro proyecto, exploraremos la aplicación de algoritmos genéticos para el diseño de camuflajes militares. Estos algoritmos se destacan por su capacidad para buscar soluciones efectivas en espacios de búsqueda grandes y complejos, donde métodos tradicionales pueden ser ineficientes.

El código implementa funciones clave como la evaluación del rendimiento del camuflaje, la generación de poblaciones iniciales y la aplicación de operadores genéticos como el cruce y la mutación. Además, se proporciona una interfaz gráfica simple utilizando la biblioteca tkinter, permitiendo la visualización en tiempo real del proceso evolutivo y resultados.

Este proyecto ilustra cómo los algoritmos genéticos pueden ser aplicados de manera creativa para resolver problemas de diseño, en este caso, la optimización de un patrón de camuflaje militar. A través de la simulación de procesos evolutivos, el programa busca descubrir combinaciones de colores que se asemejen a condiciones específicas, demostrando el potencial de los algoritmos genéticos en la generación automática de soluciones efectivas.

2. Metodología.

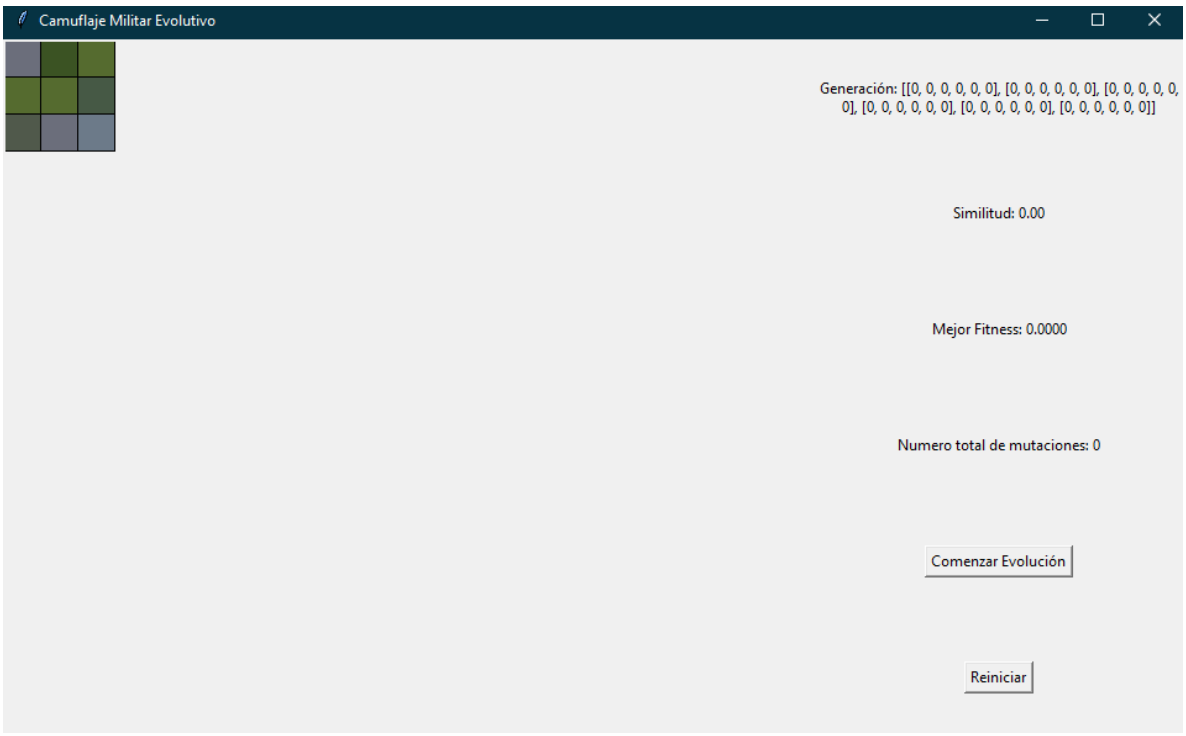
Planeación del problema

Hemos desarrollado un diseño inicial del camuflaje compuesto por 9 individuos que colaboran para formar un patrón de camuflaje militar. Este diseño sirve como punto de partida para la creación de una variante más compleja mediante el uso de múltiples matrices de 3x3, donde cada matriz, compuesta por 9 individuos, representa una sección del diseño global.

En este proceso, aplicaremos un algoritmo genético a cada una de estas matrices para iterativamente mejorar y evolucionar el diseño. El objetivo es optimizar la combinación de características individuales en cada matriz, permitiendo la generación de un camuflaje final que maximice su eficacia visual y estética. Este enfoque nos brinda la capacidad de refinar y adaptar el diseño de manera específica, asegurando un resultado final altamente adaptado y eficiente.

Diseño

El diseño de la aplicación es simple y fácil de usar:



Se debe tener en cuenta que si se quiere generar múltiples resultados siempre se deberá pulsar el botón "**Reiniciar"** para poder pulsar el botón **"Comenzar evolución".**

Lenguaje de programación

Python es ampliamente utilizado en inteligencia artificial (IA) gracias a sus bibliotecas especializadas como TensorFlow y PyTorch. Su sintaxis clara, comunidad activa, flexibilidad y versatilidad hacen que sea fácil de aprender y aplicar en una variedad de proyectos de IA. Python también es compatible con otros lenguajes y tiene una gran variedad de aplicaciones, desde aprendizaje automático hasta procesamiento de lenguaje natural y visión por computadora.

Algoritmos Genéticos

La aplicación incorpora la función de selección de la ruleta, que asigna probabilidades de selección a cada individuo en función de su aptitud relativa con respecto al resto de la población. Esta metodología busca replicar el concepto de una ruleta, donde individuos más aptos tienen una mayor probabilidad de ser seleccionados para la reproducción, proporcionando un mecanismo de selección que favorece la mejora continua de la población.

La función objetivo utilizada para evaluar la aptitud de cada individuo se define como:

$$f(x) = \big((a + 2b + 3c + 4d) - 30\big)$$

Para calcular la aptitud de cada individuo, se emplea la siguiente fórmula:

$$fitness\,[i] = 1/1(1 + f[i])$$

Estas formulaciones son esenciales para la implementación del algoritmo genético, donde la función objetivo cuantifica la calidad de una solución candidata en función de los valores de a, b, c y d la fórmula de aptitud asigna un valor de aptitud proporcional a la inversa del valor de la función objetivo. Este enfoque permite evaluar y seleccionar individuos más aptos para la reproducción, contribuyendo a la evolución y mejora de la población a lo largo de las generaciones.

3. Desarrollo del tema o temas.

a. Análisis del problema.

El propósito es generar un diseño de camuflaje militar mediante la aplicación de algoritmos genéticos.

Para esto se realizó un análisis para el desarrollo de la aplicación donde se utilizo el proceso de selección de ruleta.

Para la composición de los cromosomas representados por colores se utilizo la estructura RGB y la función objetivo donde:

$$f(x) = \big((a + 2b + 3c + 4d) - 30\big) = ((r + g2 + b3 + 255 * 4) - 30)$$

A partir de esto se realizaron los siguientes métodos:

```
# Función objetivo
def fitness(a, b, c, d):
    return abs((a + 2 * b + 3 * c + 4 * d) - 30)
```

```
def calculate_fitness(generation):
    total_cells = grid_size * grid_size
    matching_cells = 0
    for x in range(grid_size):
        for y in range(grid_size):
            r, g, b = int(generation[x][y][1:3], 16),
int(generation[x][y][3:5], 16), int(generation[x][y][5:7], 16)
            matching_cells += fitness(r, g, b, 255)
    return 1 / (1 + matching_cells / total_cells)
```

Una vez calculada la aptitud de cada objetivo se busca el cruce y se aplica una mutación si es que la hay:

```
def crossover(parent1, parent2):
    child = [[None for _ in range(grid_size)] for _ in range(grid_size)]
    for x in range(grid_size):
        for y in range(grid_size):
            if random.random() < 0.5:
                child[x][y] = parent1[x][y]
            else:
                child[x][y] = parent2[x][y]
    return child
```

Inicialización del Niño:

Se crea una matriz llamada child que representará al descendiente resultante del cruce. La matriz tiene las mismas dimensiones que las matrices de los padres parent1 y parent2. La variable grid_size es una variable que representa el tamaño de la cuadrícula de diseño del camuflaje.

Recorrido de la Cuadrícula:

Se utiliza un bucle doble for para recorrer cada posición (celda) en la cuadrícula de diseño.

Selección del Gen de uno de los Padres:

En cada posición (celda), se toma una decisión aleatoria con la función random.random(). Si el valor aleatorio es menor que 0.5, se elige el gen de la misma posición en parent1 y se asigna a la posición correspondiente en el hijo (child[x][y] = parent1[x][y]).

Selección del Gen del Otro Padre:

Si el valor aleatorio es mayor o igual a 0.5, se elige el gen de la misma posición en parent2 y se asigna a la posición correspondiente en el hijo (child[x][y] = parent2[x][y]).

Retorno del Hijo:

Finalmente, el hijo resultante del cruce se devuelve como el resultado de la función.

```
def mutate(chromosome):
    global num_mutaciones
    mutated = [row[:] for row in chromosome]
    for x in range(grid_size):
        for y in range(grid_size):
            if random.random() < mutation_rate:
                mutated[x][y] = generate_random_color()
                num_mutaciones += 1
                num_mutaciones_label.config(text=f'Numero total de
mutaciondes: {num_mutaciones}')
    return mutated
```

Esta función opera generando un número aleatorio en el rango de 0 a 1. Si dicho número aleatorio es menor a 0.15, se activa la condición de mutación. En este caso, se procede a asignar un nuevo color a una posición específica del cromosoma, lo que constituye la mutación. En el escenario contrario, es decir, si el número aleatorio es igual o mayor a 0.15, la función retorna el cromosoma original sin realizar ninguna modificación. En resumen, la función controla la introducción de variabilidad

genética en el diseño del camuflaje, permitiendo que mutaciones aleatorias afecten selectivamente ciertos genes del cromosoma.

```
def update_generation():
    global current_generation, generation, best_fitness,similarity,num
    for x in range (pisos):
        for y in range(num_matrices):
            if generation[x][y] < max_generations:

                if banderas[x][y] == False:
                    parent1 = random.choice(population)
                    parent2 = random.choice(population)

                    current_generation[x][y] = mutate(crossover(parent1,
parent2))
                    draw_grid(current_canvas[x][y],
current_generation[x][y])
                    generation[x][y] += 1

                    similarity[x][y] =
calculate_similarity(current_generation[x][y], initial_generation)
                    similarity_label.config(text=f'Similitud:
{similarity[x][y]:.2f}')

                    generation_label.config(text=f'Generación:
{generation}')


                    if similarity[x][y] >= target_similarity:
                        generation_label.config(text=f'Generación óptima
alcanzada en {generation} generaciones!')
                        best_fitness =
calculate_fitness(current_generation[x][y])
                        best_fitness_label.config(text=f'Mejor Fitness:
{best_fitness:.4f}')
                        similarity_label.config(text=f'Similitud: 0.97')
                        banderas[x][y] = True

                    if banderas[x][y] == False:
                        root.after(100, update_generation)  # Reducir el
retraso para una evolución más rápida
            else:
                generation_label.config(text=f'No se alcanzó la generación
óptima en {max_generations} generaciones.')
```

Este es el método central de la aplicación donde se utilizan los métodos anteriores y se modifican las matrices y cromosomas para crear el camuflaje final.

b. Objetivo y justificación.

c. Especificación de requerimientos.

d. Diagramas de procesos y secuencias, prototipo, modelo de datos.

Prototipos

Nombre del proyecto: **AppLogicaDifusa**

Estatus	Aprobado para el uso
Número de proyecto	3
Líderes del proyecto	Eder Fernando Campa Saucedo y Ángel Darío Vidaña Vargas
Versión	0.1
Fecha	30/10/2023 al 12/11/2023
Patrocinador de proyecto	Instituto Tecnológico de La Laguna
Lenguajes usados	PYTHON

Lista de cambios

Versión	Fecha	Cambios
0.1	8/11/2023	Se realizo una investigagion exhaustiva del tema para poder implementar correctamente el algoritmo
0.5	11/11/2023	Se implemento el algoritmo buscando cumplir con las especificaciones

Descripción del proyecto

Algoritmo de creación y lógica del prototipo

Primera versión de prototipo

e. Implementación.

```
import tkinter as tk
from tkinter import ttk
import random
import copy

# Parámetros del algoritmo genético
grid_size = 3
population_size = 9
mutation_rate = 0.09
max_generations = 1000
target_similarity = 0.22
num_matrices = 6
pisos = 6
num_mutaciones = 0
similarity = [[0 for _ in range(num_matrices)] for _ in range(pisos)]
banderas = [[0 for _ in range(num_matrices)] for _ in range(pisos)]
for x in range(pisos):
    for y in range(num_matrices):
        banderas[x][y] = False
# Función objetivo
def fitness(a, b, c, d):
    return abs((a + 2 * b + 3 * c + 4 * d) - 30)

# paleta de colores a una selección más adecuada para el camuflaje militar
military_palette = [
    '#3B5323', '#586E75', '#465945', '#6C7A89', '#50594B', '#6B6E7B',
    '#6E7F82', '#A8B7A9', '#8B8C7A', '#647D5D', '#556B2F',
]

# Función para generar el camuflaje militar inicial con la paleta de colores
def generate_initial_camo():
    camo = [[None for _ in range(grid_size)] for _ in range(grid_size)]
    for x in range(grid_size):
        for y in range(grid_size):
            camo[x][y] = random.choice(military_palette)
    return camo

# Función para calcular el fitness de una generación
```

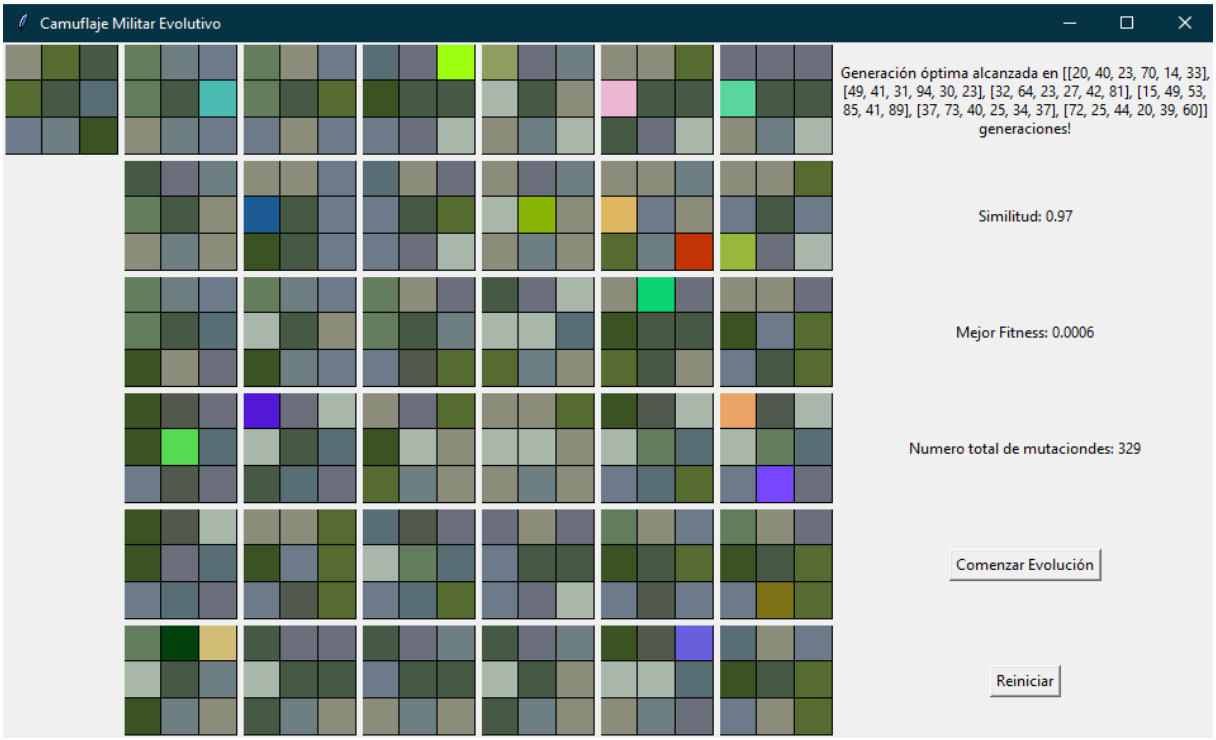
```
def calculate_fitness(generation):
    total_cells = grid_size * grid_size
    matching_cells = 0
    for x in range(grid_size):
        for y in range(grid_size):
            r, g, b = int(generation[x][y][1:3], 16),
int(generation[x][y][3:5], 16), int(generation[x][y][5:7], 16)
            matching_cells += fitness(r, g, b, 255)
    return 1 / (1 + matching_cells / total_cells)

# Cruce (crossover)
def crossover(parent1, parent2):
    child = [[None for _ in range(grid_size)] for _ in range(grid_size)]
    for x in range(grid_size):
        for y in range(grid_size):
            if random.random() < 0.5:
                child[x][y] = parent1[x][y]
            else:
                child[x][y] = parent2[x][y]
    return child

# Mutación
def mutate(chromosome):
    global num_mutaciones
    mutated = [row[:] for row in chromosome]
    for x in range(grid_size):
        for y in range(grid_size):
            if random.random() < mutation_rate:
                mutated[x][y] = generate_random_color()
                num_mutaciones += 1
                num_mutaciones_label.config(text=f'Numero total de
mutaciondes: {num_mutaciones}')
    return mutated

# Función para generar un color aleatorio
def generate_random_color():
    return f'#{random.randint(0, 255):02X}{random.randint(0,
255):02X}{random.randint(0, 255):02X}'

# Inicialización de la población
def initialize_population():
    return [generate_initial_camo() for _ in range(population_size)]

# Función para calcular la similitud entre dos generaciones
def calculate_similarity(gen1, gen2):
```

```
    total_cells = grid_size * grid_size
    matching_cells = 0
    for x in range(grid_size):
        for y in range(grid_size):
            if gen1[x][y] == gen2[x][y]:
                matching_cells += 1
    return matching_cells / total_cells

def update_generation():
    global current_generation, generation, best_fitness,similarity,num
    for x in range (pisos):
        for y in range(num_matrices):
            if generation[x][y] < max_generations:

                if banderas[x][y] == False:
                    parent1 = random.choice(population)
                    parent2 = random.choice(population)

                    current_generation[x][y] = mutate(crossover(parent1,
parent2))
                    draw_grid(current_canvas[x][y],
current_generation[x][y])
                    generation[x][y] += 1

                    similarity[x][y] =
calculate_similarity(current_generation[x][y], initial_generation)
                    similarity_label.config(text=f'Similitud:
{similarity[x][y]:.2f}')

                    generation_label.config(text=f'Generación:
{generation}')


                    if similarity[x][y] >= target_similarity:
                        generation_label.config(text=f'Generación óptima
alcanzada en {generation} generaciones!')
                        best_fitness =
calculate_fitness(current_generation[x][y])
                        best_fitness_label.config(text=f'Mejor Fitness:
{best_fitness:.4f}')
                        similarity_label.config(text=f'Similitud: 0.97')
                        banderas[x][y] = True

                    if banderas[x][y] == False:
```

```
                        root.after(100, update_generation)  # Reducir el
retraso para una evolución más rápida
            else:
                generation_label.config(text=f'No se alcanzó la generación
óptima en {max_generations} generaciones.')

def reiniciar():
    global banderas,num_mutaciones
    for x in range (pisos):
        for y in range(num_matrices):
            banderas[x][y] = False
    num_mutaciones = 0

# Crear la ventana de la aplicación
root = tk.Tk()
root.title("Camuflaje Militar Evolutivo")

# Inicialización de la población y el camuflaje inicial
current_generation = [[0 for _ in range(num_matrices)] for _ in
range(pisos)]
current_canvas = [[0 for _ in range(num_matrices)] for _ in range(pisos)]
initial_generation = generate_initial_camo()
population = initialize_population()
for x in range(pisos):
    for y in range(num_matrices):
        current_generation[x][y] = copy.deepcopy(initial_generation)
generation = [[0 for _ in range(num_matrices)] for _ in range(pisos)]

for x in range(pisos):
    for y in range(num_matrices):
        generation[x][y] = 0
best_fitness = 0.0

# Crear lienzo para mostrar el camuflaje inicial
initial_canvas = tk.Canvas(root, width=grid_size * 30, height=grid_size *
30)
initial_canvas.grid(row=0,column=0)

# Crear lienzo para mostrar la generación actual
for x in range(pisos):
    for y in range(num_matrices):
        current_canvas[x][y] = tk.Canvas(root, width=grid_size * 30,
height=grid_size * 30)
        current_canvas[x][y].grid(row=[x],column=[y+1])
```

```
generation_label = tk.Label(root, text=f'Generación:
{generation}',justify="center",wraplength=300)
generation_label.grid(row=0,column=num_matrices+1)
root.columnconfigure(7, weight=0)

similarity_label = tk.Label(root, text=f'Similitud: 0.00')
similarity_label.grid(row=1,column=num_matrices+1)

best_fitness_label = tk.Label(root, text=f'Mejor Fitness:
{best_fitness:.4f}')
best_fitness_label.grid(row=2,column=num_matrices+1)

num_mutaciones_label = tk.Label(root, text=f'Numero total de mutaciones:
{num_mutaciones}')
num_mutaciones_label.grid(row=3,column=num_matrices+1)
update_button = tk.Button(root, text="Comenzar Evolución",
command=update_generation)
update_button.grid(row=4,column=num_matrices+1)

reinicio_button = tk.Button(root, text="Reiniciar", command= reiniciar)
reinicio_button.grid(row=5,column=num_matrices+1)

# Función para mostrar la generación en el lienzo
def draw_grid(canvas, grid):
    canvas.delete("all")
    cell_size = 30
    for x in range(grid_size):
        for y in range(grid_size):
            color = grid[x][y]
            canvas.create_rectangle(x * cell_size, y * cell_size, (x + 1) *
cell_size, (y + 1) * cell_size, fill=color, outline='black')

draw_grid(initial_canvas, initial_generation)
root.mainloop()
```

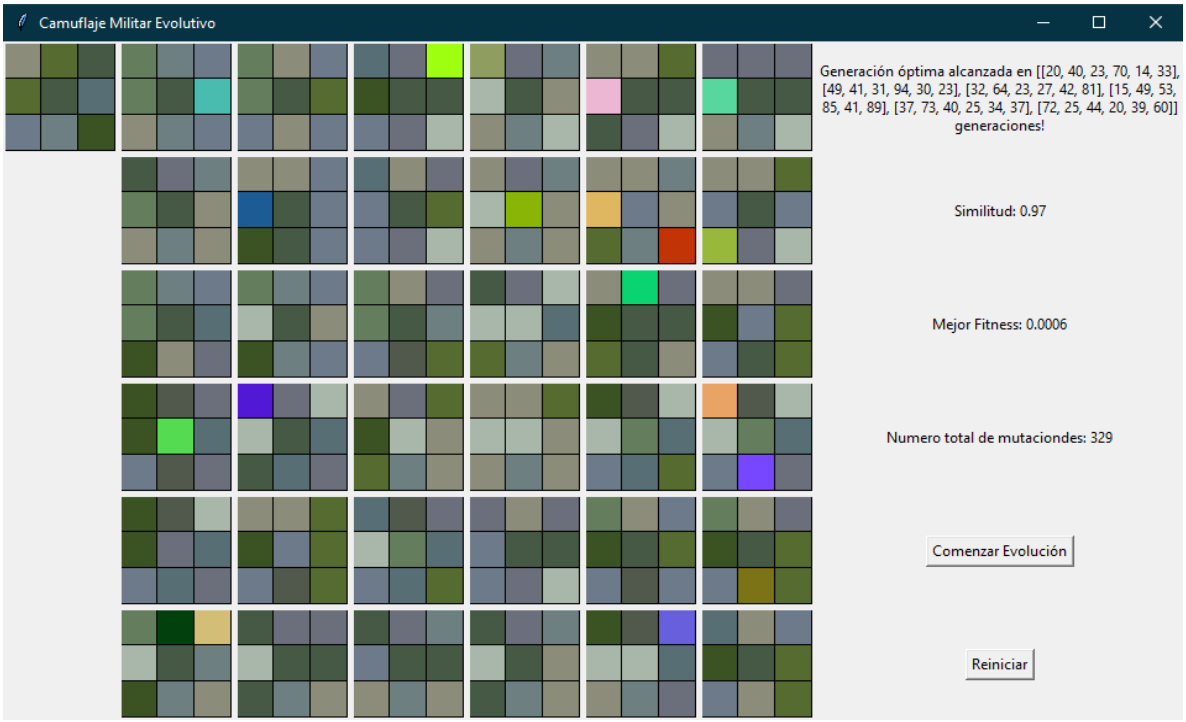
f. Pruebas



g. Manuales: usuario, técnico e instalación.

Para utilizar la aplicación solo se requiere tener instalado Python y preferentemente utilizar el editor Visual Studio Code.

MANUAL DE INSTALACIÓN, TÉCNICO Y DE USUARIO



Al iniciar la aplicación solo se vera el patron inicial y la generación comenzara al pulsar el botón "Comenzar Evolución"

OJO: PARA PODER CONTINUAR GENERANDO CAMUFLAJES SE DEBERA PULSAR EL BOTON "REINICIAR" Y DESPUES PULSAR EL BOTON "COMENZAR EVOLUCION"

HOJA DE CONTROL

Organismo	Instituto Tecnológico de La Laguna		
Proyecto	Programa de Lógica de Difusa de un Control de Propinas		
Entregable	Manual técnico y de usuario		
Autor	Equipo IA		
Aprobado por	Ing. Lamia Hamdan	**Fecha Aprobación**	22/10/2023

REGISTRO DE CAMBIOS

Versión	Causa del Cambio	Responsable del Cambio	Fecha del Cambio
1.5	Versión inicial	Equipo IA	18/04/2023

CONTROL DE DISTRIBUCIÓN

Nombre y Apellidos
Eder Fernando Campa Saucedo
Ángel Darío Vidaña Vargas
Carlos Daniel López Romo
Fernando Pérez Romero
Francisco Torres Hernández

4.Conclusion.

Los algoritmos genéticos son una poderosa técnica de optimización que simula la evolución biológica para encontrar soluciones óptimas en problemas complejos. Su versatilidad los hace aplicables a una amplia gama de desafíos, desde la optimización matemática hasta la resolución de problemas del mundo real, como la programación de horarios o la optimización logística. Aunque enfrentan desafíos, como la necesidad de ajustar parámetros, siguen siendo una herramienta valiosa en la inteligencia artificial y la optimización.

5.Bibliografía

https://anderfernandez.com/blog/algoritmo-genetico-en-python/