FACULTAD DE CIENCIAS

INTELIGENCIA ARTIFICIAL

Algoritmos Genéticos

Equipo: `Skynet Scribes`

Número de practica: `05`

Sarah Sophía Olivares García
318360638

Marco Silva Huerta
316205326

Carlos Daniel Cortés Jiménez
420004846

Laura Itzel Tinoco Miguel
316020189

Luis Enrique García Gómez
315063880

Fernando Mendoza Eslava
319097690

Profesora: Cecilia Reyes Peña
Ayudante teoría: Karem Ramos Calpulalpan
Ayudante laboratorio: Tania Michelle Rubí Rojas

Semestre 2024-2

Fecha de entrega:
03 de Abril del 2024

Índice

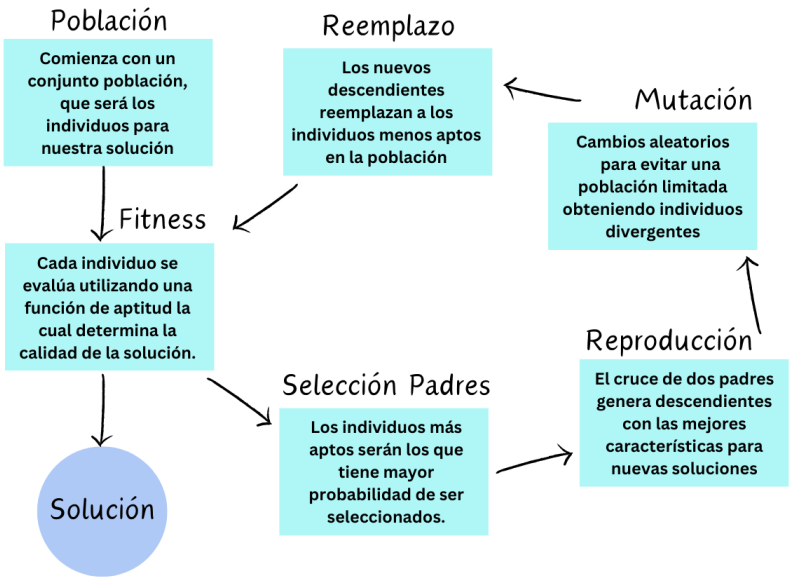
1. Investigación

¿Qué es el algoritmo Genético?

Un algoritmo genético `(AG)` entonces es: una técnica de resolución de problemas que utiliza principios inspirados en la selección natural para solucionar problemas de optimización. La selección natural (evolución) como estrategia implica la creación, reproducción y adaptación de una población de posibles soluciones en un rango de generaciones.

Este enfoque de población se basa en que cada individuo representa una posible solución al problema, y la evolución de estos comienza por, la **creación** de los individuos, la **reproducción** para después hacer la óptima **selección** de los padres en una nueva reproducción y **mutación**, finalmente comprobar o evaluar su **aptitud**

Pasos que sigue



¿Cuál es el juego de la vida?

El Juego de la vida es un autómata celular, es decir es un modelo matemático llevado de manera computacional para observar de manera dinámica cómo evoluciona en el tiempo. Creado por John Horton Conway en 1969 que consiste en una cuadrícula donde se marcan o desmarcan siendo pintados por un color, representando células y reflejando si esta se encuentra viva o muerta.

Iniciado el juego son las reglas las que determinan como acaba. El objetivo es crear un sistema que simulase la vida y su naturaleza. La manera en que se logra es con el nacimiento y supervivencia de estos seres binarios digitales donde depende del número de vecinos que estén a su vez vivos o muertos. Por lo que es posible observar patrones complejos e impredecibles gracias a unas sencillas reglas. Siendo utilizado como un modelo de simulación para estudiar fenómenos biológicos, evolutivos o físicos.

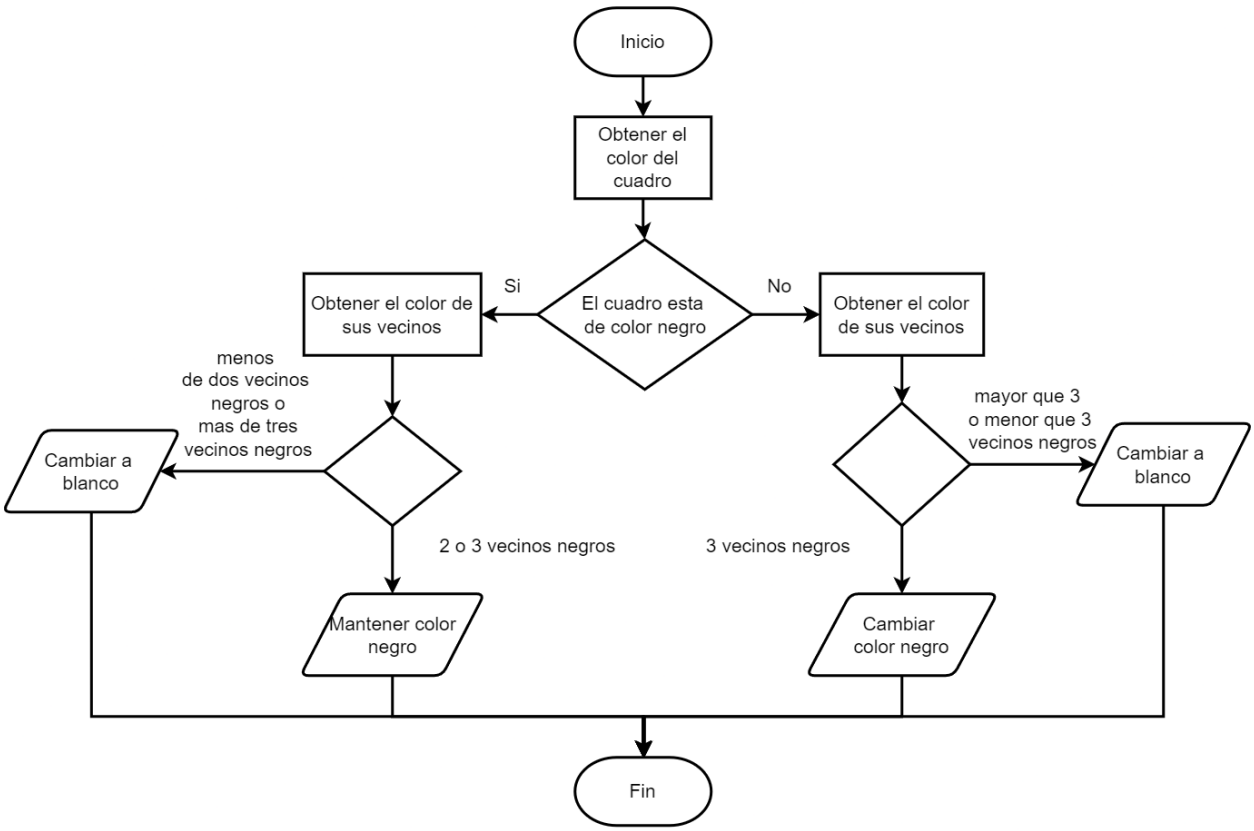


Figura 1: Diagrama de flujo del Juego de la Vida

En el diagrama se puede observar las reglas que se siguen para determinar si una célula vive o muere. En este caso las condicionales del diagrama reflejan la sobrepoblación, la soledad , la reproducción y la supervivencia de las células. Por lo que solo faltaría implementar el código en Python para poder visualizar el juego de la vida.

2. Desarrollo

2.1. Implementación básica del Juego de la Vida

Descripción del código

El código implementa el Juego de la Vida, un autómata celular que simula el comportamiento de un sistema de células vivas y muertas en una cuadrícula, el código consta de las siguientes partes principales:

Creación del estado inicial del juego

La función **crear_matriz_inicial** genera una matriz cuadrada de tamaño $n \times n$ con valores aleatorios de 0 (célula muerta) y 1 (célula viva), esta matriz representa el estado inicial del juego.

```
def crear_matriz_inicial(n):
    """
    Genera una matriz n x n con celulas vivas o muertas aleatoriamente.

    Parametros:
    - n: Tamano de la matriz (numero de filas y columnas).

    Retorna:
    - Matriz n x n con valores aleatorios 0 (muerta) o 1 (viva).
    """
    return np.random.choice([0, 1], size=(n, n))
```

Aplicación de las reglas del Juego de la Vida

La función **aplicar_reglas** toma el estado actual del juego como entrada y aplica las reglas del Juego de la Vida para determinar el siguiente estado, las reglas son:

- Una célula viva con dos o tres vecinas vivas sobrevive.
- Una célula muerta con tres vecinas vivas nace.
- En cualquier otro caso, la célula muere.

```
def aplicar_reglas(estado_actual):
    """
    Aplica las reglas del Juego de la Vida a cada celula de la matriz.

    Parametros:
    - estado_actual: Matriz que representa el estado actual del juego.

    Retorna:
    - Nuevo estado del juego despues de aplicar las reglas.
    """
    n = estado_actual.shape[0]
    nuevo_estado = np.zeros((n, n))

    for i in range(n):
        for j in range(n):
            num_vecinos = np.sum(estado_actual[i-1:i+2, j-1:j+2]) - estado_actual[i, j]

```

```
            # Regla a) Si una celula esta viva y tiene dos o tres vecinas vivas, sobrevive.
            if estado_actual[i, j] == 1 and num_vecinos in [2, 3]:
                nuevo_estado[i, j] = 1

            # Regla b) Si una celula esta muerta y tiene tres vecinas vivas, nace.
            elif estado_actual[i, j] == 0 and num_vecinos == 3:
                nuevo_estado[i, j] = 1
            else:
                nuevo_estado[i, j] = 0

    return nuevo_estado
```

Visualización del estado del juego

La función **visualizar_estado** se utiliza para mostrar el estado actual del juego mediante una imagen en escala de grises.

```
def visualizar_estado(estado, titulo="Estado del Juego de la Vida"):
    """
    Visualiza el estado del juego.

    Parametros:
    - estado: Matriz que representa el estado actual del juego.
    - titulo: Titulo para la visualizacion.
    """
    plt.figure(figsize=(5, 5))
    plt.imshow(estado, cmap='Greys', interpolation='nearest')
    plt.title(titulo)
    plt.show()
```

Actualización de la matriz y visualización a través de múltiples turnos

Esta sección del código implementa un bucle que actualiza la matriz basándose en las reglas y visualiza el estado de la cuadrícula después de cada actualización.

```
# Numero de iteraciones/turnos para visualizar
num_iteraciones = 5
n = 10  # Tamano de la matriz

# Generar el estado inicial del juego
estado_actual = crear_matriz_inicial(n)

# Visualizar el estado inicial
visualizar_estado(estado_actual, "Estado Inicial")

# Bucle para actualizar y visualizar el juego en cada turno
for i in range(num_iteraciones):
    # Aplicar las reglas del Juego de la Vida para obtener el nuevo estado
    estado_actual = aplicar_reglas(estado_actual)
    # Visualizar el estado actual despues de aplicar las reglas
    visualizar_estado(estado_actual, titulo=f"Estado despues de {i + 1} turno(s)")
```

2.2. Introducción a los Algoritmos Genéticos

2.3. Implementación de Algoritmos Genéticos en el Juego de la Vida

Recordemos las reglas iniciales del juego:

1. Si una célula está viva y tiene dos o tres vecinas vivas, sobrevive.
2. Si una célula está muerta y tiene tres vecinas vivas, nace.
3. Si una célula está viva y tiene más de tres vecinas vivas, muere.

   La disposición o patrón inicial de células se llama *semilla*. La siguiente generación nace de aplicar las reglas del juego a todas las células de manera simultánea. Este proceso se puede ejecutar de manera indefinida.

Modificación de las Reglas

Para modificar la simulación del Juego por los cromosomas de una población podemos crear una **representación cromosómica:** donde cada cromosoma será una cadena binaria donde cada bit representa el estado de una célula en el tablero (vivo o muerto).

Así las nuevas reglas para el *Juego de la Vida* que estamos proponiendo es para buscar crear más vida con menos generaciones, las células podrán vivir con más vecinos si se alcanza un cierto número de cromosomas en la población:

- Nacimientos: Una célula muerta con exactamente tres vecinos vivos se convierte en una célula viva.
- Muerte uno: Una célula viva con uno o menos vecinos vivos muere.
- Muerte dos: Una célula viva con más de tres vecinos vivos muere, a menos que se cumpla la condición especial.
- Condición Especial de Supervivencia: Si la población de cromosomas alcanza o supera un n especifica, las células vivas pueden soportar hasta cuatro vecinos vivos sin morir.
- Supervivencia Normal: Si no se cumple la condición especial, una célula viva con dos o tres vecinos vivos sobrevive.
- Muerte tres: Si en n generaciones no se alcanza el objetivo el juego termina.

El objetivo que queremos lograr con este cambio es encontrar un conjunto de células más eficiente en el menor tiempo de generaciones, para este caso definiremos que sea cumplido tras un 70 porciento del total de generaciones (aproximadamente 160 generaciones), es decir, buscamos las células que se mantengan como población estable y amplia cumpliendo los criterios de aptitud.

Pasos del algoritmo

- Inicialización de la Población: Generamos una población inicial de cromosomas de manera aleatoria
- Evaluación de la Aptitud: astronaves
- Selección: Utilizamos un método de selección por torneo

- Reproducción: Cruzamiento en dos puntos, en el que se intercambian los genes que aparecen en el intervalo de genes delimitados por dos puntos.
- Mutación: Cambiar aleatoriamente el estado de algunas células
- Remplazo: Una vez realizada la mutación y aplicar una evaluación fitness (más tranquila que la evaluación de la aptitud) se hace el remplazo

Inicialización de la Población

Cada célula en el tablero de $n \times n$ tiene ocho vecinos, que incluyen las celdas adyacentes horizontal, vertical y diagonalmente. Al comienzo del juego, la población inicial se genera aleatoriamente, cada cromosoma es una secuencia binaria que representa un estado completo del tablero, con 1s para las células vivas y 0s para las muertas.

Función fitness

Para definir la función fitness necesitamos saber que existen patrones básicos dentro del juego y estos son configuraciones de los vecinos para las células que determinan un comportamiento concreto como patrones estáticos que no hay nacimientos ni fallecimientos, y nunca cambian, patrones recurrentes o *osciladores* que evolucionan a través de diversos estados pero vuelven a su forma inicial y patrones que se trasladan por el tablero llamados *spaceships*.

Las astronaves [Dop] son patrones que se caracterizan por desplazarse a través del tablero a lo largo del tiempo, bien sea de forma diagonal o de forma horizontal o vertical. La velocidad de desplazamiento es variable, dependiendo del patrón que se trate. y esta capacidad para moverse eficientemente es un indicador de su robustez y estabilidad dentro del juego

La fórmula de velocidad para las *spaceships*:

$$v = \frac{max(|x|, |y|)}{n} \times c$$

(v) es la velocidad de la astronave.

(x) y (y) son los desplazamientos en dos dimensiones (horizontal y vertical).

(n) es el número de generaciones que tarda la astronave en desplazarse.

(c) es una constante análoga a la velocidad de la luz, que en este contexto se puede considerar como 1 celda por generación.

Esta fórmula es útil para medir la eficiencia con la que un patrón se desplaza a través del tablero. Los patrones que se mueven más rápido (mayor valor de (v)) serían considerados más *aptos* mientras que con velocidades más bajas nos dice que esos patrones son menos eficientes (lentos) o que mueren rápidamente, de manera que calificaran con aptitud menor.

Entonces estamos considerando la distancia máxima recorrida en cualquier dirección y el número de generaciones necesarias para dicho desplazamiento, la ventaja de usar la velocidad para hacer la evaluación es que favorece patrones que pueden sobrevivir y moverse a lo largo de las generaciones, como si fuera una evolución natural sin embargo se añadió otro criterio el cual consiste en sumar puntos si es que los patrones no chocan demasiado con los bordes, ya que células vivas en bordes podrían tener menos vecinos para sobrevivir.

3. Resultados obtenidos

3.1. Juego de la Vida sin algoritmo genético

A continuación se presentan dos casos diferentes de estados iniciales y se visualiza su evolución a través de 5 turnos.

- Estado inicial aleatorio

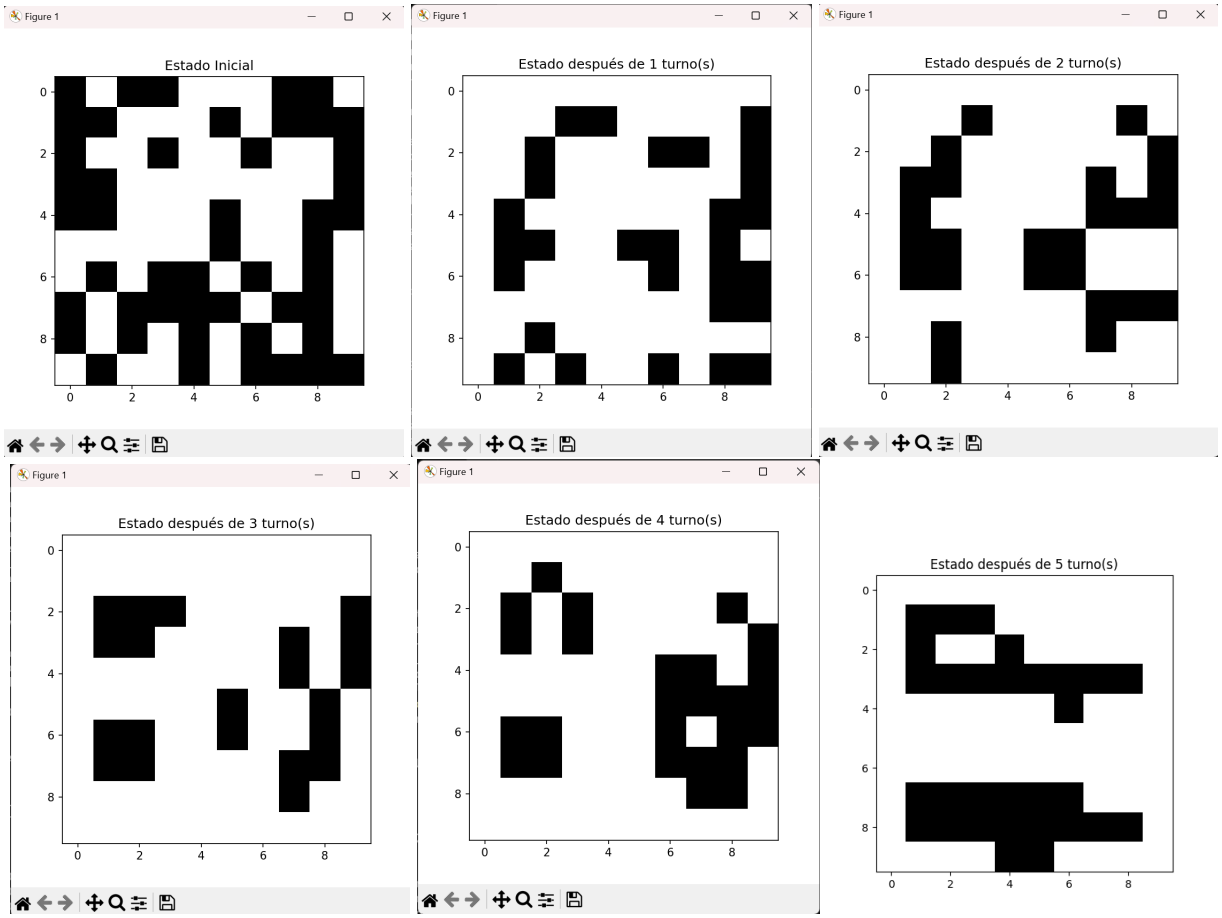
```
# Numero de iteraciones/turnos para visualizar
num_iteraciones = 5
n = 10  # Tamano de la matriz

# Generar el estado inicial del juego
estado_actual = crear_matriz_inicial(n)

# Visualizar el estado inicial
visualizar_estado(estado_actual, "Estado Inicial")

# Bucle para actualizar y visualizar el juego en cada turno
for i in range(num_iteraciones):
    # Aplicar las reglas del Juego de la Vida para obtener el nuevo estado
    estado_actual = aplicar_reglas(estado_actual)
    # Visualizar el estado actual despues de aplicar las reglas
    visualizar_estado(estado_actual, titulo=f"Estado despues de {i + 1} turno(s)")
```

Los estados se muestran de la siguiente forma:



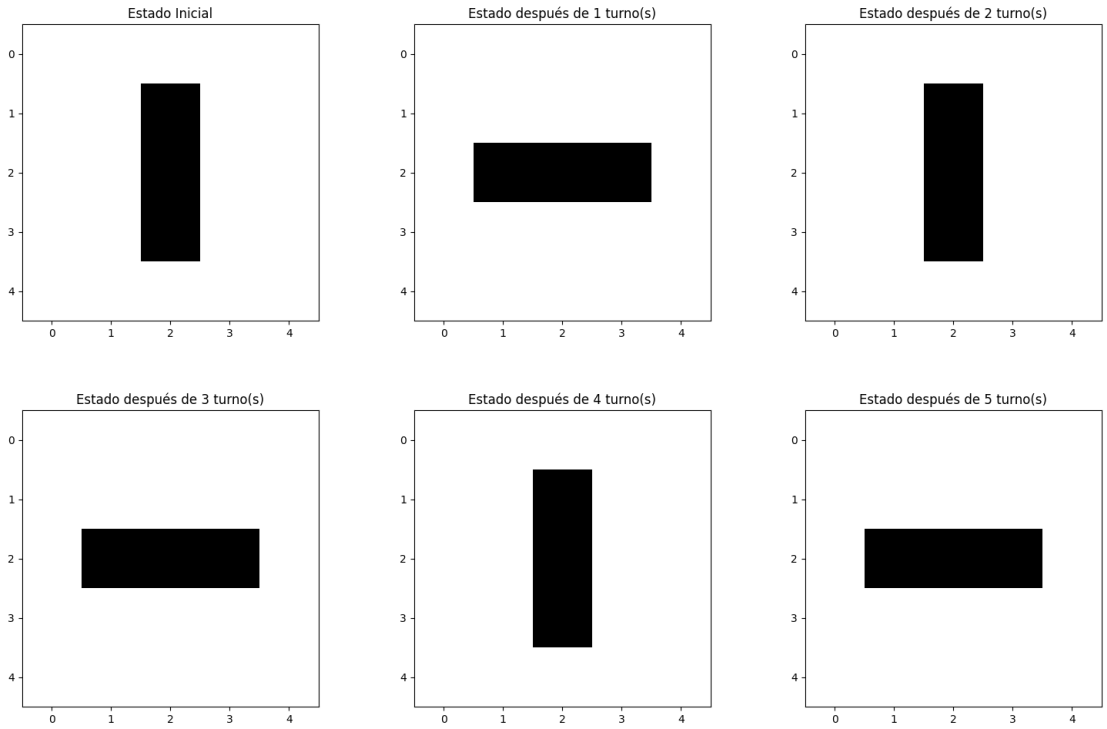
- Estado inicial formado

```
# Estado inicial formando un bloque estable
estado_actual = np.array([[0, 0, 0, 0, 0],
                          [0, 0, 1, 0, 0],
                          [0, 0, 1, 0, 0],
                          [0, 0, 1, 0, 0],
                          [0, 0, 0, 0, 0]])


# Visualizar el estado inicial
visualizar_estado(estado_actual, "Estado Inicial")

# Bucle para actualizar y visualizar el juego en cada turno
for i in range(num_iteraciones):
    # Aplicar las reglas del Juego de la Vida para obtener el nuevo estado
    estado_actual = aplicar_reglas(estado_actual)
    # Visualizar el estado actual despues de aplicar las reglas
    visualizar_estado(estado_actual, titulo=f"Estado despues de {i + 1} turno(s)")
```

Los estados se muestran de la siguiente forma:



3.2. Juego de la Vida con algoritmo genético

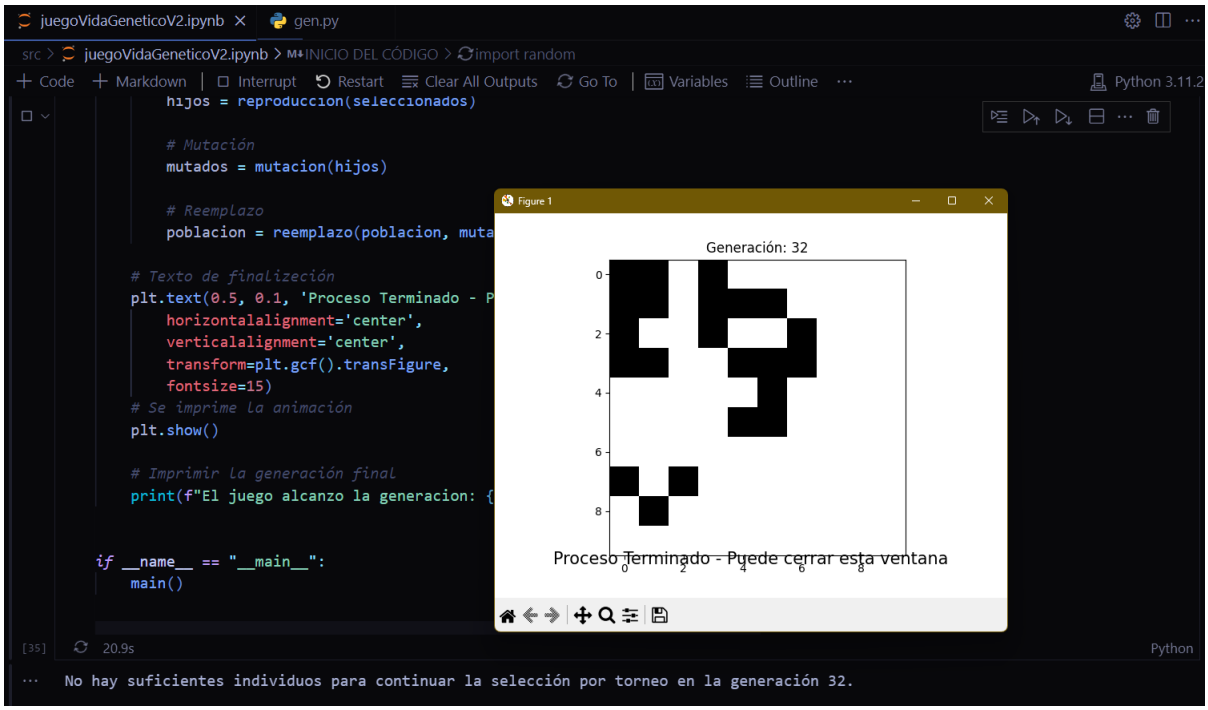
Animación

Usamos la importación de `import matplotlib` para usar `matplotlib.use('TkAgg')` y así poder ejecutar la ventana emergente que muestra la animación del Juego de la vida. Mencionamos esto por dos cosas:

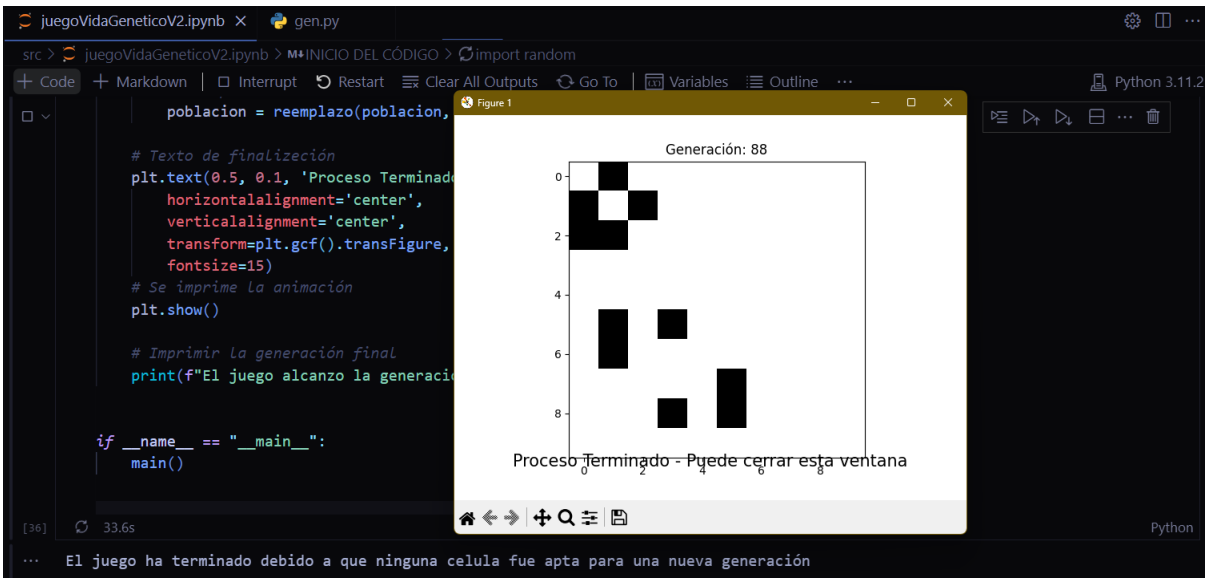
- Notamos que al ejecutar un archivo `.py` funciona, al ejecutar un archivo *.ipynb* en nuestro caso dentro de Visual Studio Code funciona la animación pero si el código es ejecutado en un archivo de **colab** truena y aunque quitemos la importación la animación no se ejecuta.
- Se necesito instalar usando el comando (para linux): `sudo apt install python3-tk`, es para la visualización de gráficos.

Pruebas

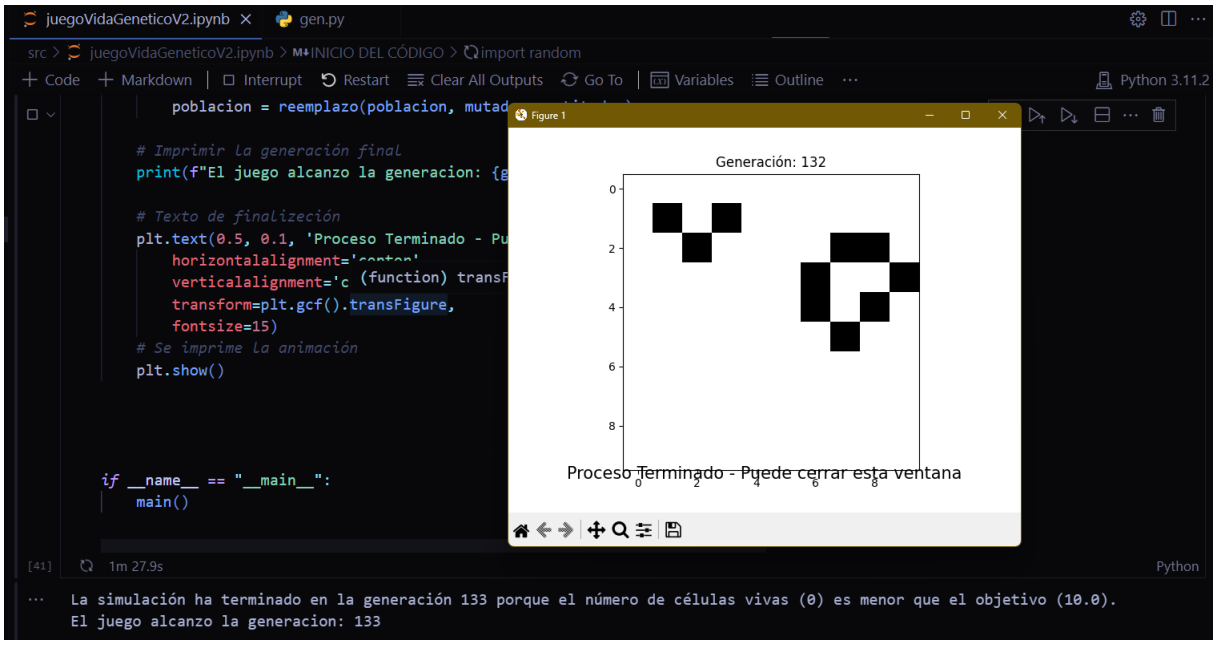
Nuestro primer caso de salida es cuando hemos agotado la población y no hay suficientes para realizar el torneo.



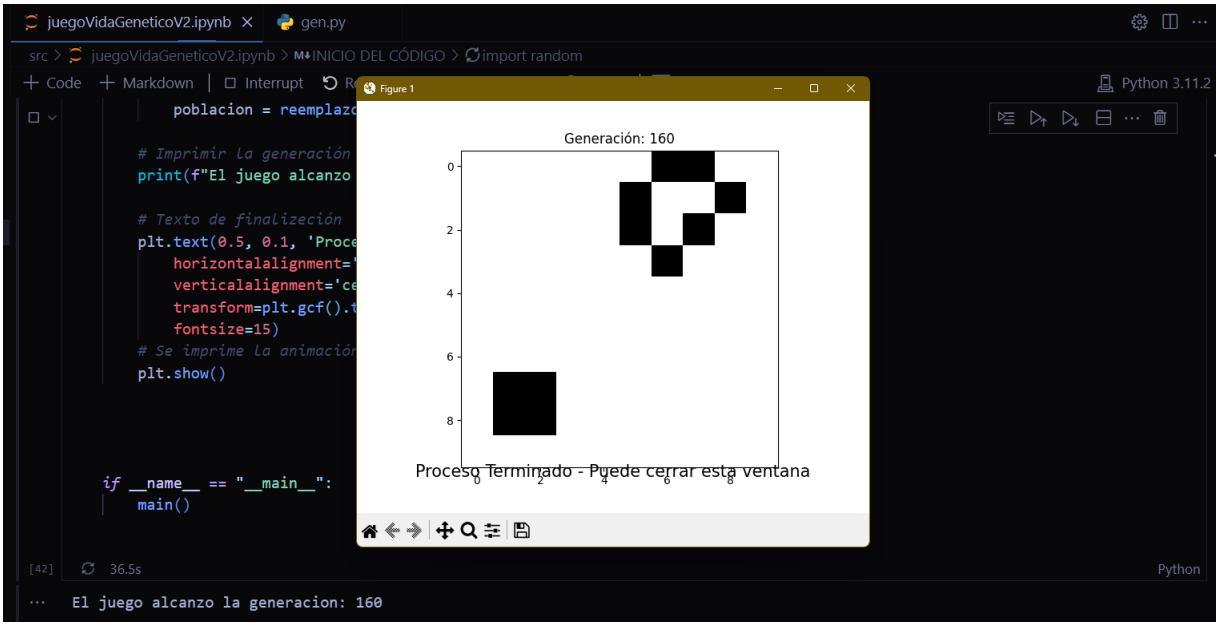
La segunda salida es cuando la generación no es apta bajo los criterios, por lo tanto no hay una nueva generación y tenemos población vacía, entonces terminamos.



Para la tercera salida es el caso especial que condicionamos, donde si nuestro porcentaje de células vivas no es mayor después de cierta generación (elegimos el 70 porciento del total dado) salimos del programa. Esto lo hicimos con la finalidad de encontrar células más aptas en menos generaciones.



Finalmente el caso exitoso donde llegamos al final de las generaciones establecidas.



Notas

Dentro del algoritmo se definieron constantes con las que hemos jugado a lo largo de la creación del algoritmo, comprobamos la importancia de la función de aptitud para llegar a nuestro objetivo a demás de ver como las mutaciones en este caso particular pueden ser buenas y malas.

Las constantes son:

- `TABLERO` Se define un tablero cuadrado y podemos jugar con el tamaño de sus lados para que a medida que células más aptas crezcan podamos observar patrones generados, cosa que no es tan visual con tableos pequeños.

- `POBLACION INICIAL` Comprobamos que para nuestra solución entre menos población habrá menos oportunidades de éxito, una posible mejora para tratar eso sería modificar nuestra función de torneo.
- `GENERACIONES` Son la medida para saber que tanto funciona nuestra función de aptitud junto con el número de células vivas objetivo.
- `OBJETIVO CELULAS VIVAS` El porcentaje que deseamos este cubierto de células vivas dentro del tablero.
- `C` Es la constante que nos ayuda para la formula de las astronaves.

4. Reflexión final

Referencias

[RN16] Stuart Russell y Peter Norvig. *Inteligencia Artificial Un Enfoque moderno*. 2nd. Pearson Prentice Hall, 2016.

[Mat18] MatematIA. *Algoritmos Genéticos*. 2018. URL: https://www.cs.us.es/~fsancho/Blog/posts/Algoritmos_Geneticos.md.html (visitado 16-03-2024).

[Peñ19] Eric Peña. *Cellular Automata Optimization Using Genetic Algorithms*. Inf. téc. Consulta para función fitness. Binghamton: State University of New York, dic. de 2019.

[Gee24] GeeksforGeeks. *Genetic Algorithms*. 2024. URL: https://www.geeksforgeeks.org/genetic-algorithms/ (visitado 16-03-2024).

[Dop] Manuel Romero Dopico. *EL JUEGO DE LA VIDA*. URL: https://eodelgadorcursos.files.wordpress.com/2018/11/juego-de-la-vida-conway.pdf (visitado 24-03-2024).