



INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

Programa 1 - ECAs

Unidad de aprendizaje: Computing Selected Topics

Grupo: 3CM19

Alumno:
Sanchez Mendez Edmundo Josue

Profesor:
Juarez Martinez Genaro

Índice

| | |
|--|-----------|
| 1. Introducción | 3 |
| 2. Definición | 3 |
| 3. Elementos de un AC | 3 |
| 3.1. Un espacio rectangular | 3 |
| 3.2. Conjunto de estados | 3 |
| 3.3. Configuración inicial o tiempo 0 | 3 |
| 3.4. Función local o Función de transición local | 3 |
| 4. Tipos de límites o fronteras | 4 |
| 4.1. Frontera abierta | 4 |
| 4.2. Frontera reflectora | 4 |
| 4.3. Frontera periódica o circular | 4 |
| 4.4. Sin frontera | 4 |
| 5. Clasificación de los Autómatas Celulares | 4 |
| 5.1. Clase I | 4 |
| 5.2. Clase II | 5 |
| 5.3. Clase III | 5 |
| 5.4. Clase IV | 5 |
| 6. Programa | 6 |
| 6.1. Descripción | 6 |
| 6.2. Pruebas | 7 |
| 6.3. Añadidos | 20 |
| 6.4. Código | 23 |
| 7. Conclusiones | 31 |
| Referencias | 31 |

Resumen

En este programa 1 se aborda la creación de un Autómata Celular Elemental o por sus siglas en inglés (ECA) tomando como regla a usar la regla 30(00011110_2), nuestro objetivo es poder generar dicho autómata con dicha regla y poder visualizar como este va evolucionando, permitiendo así poder generar un autómata como el siguiente ejemplo.

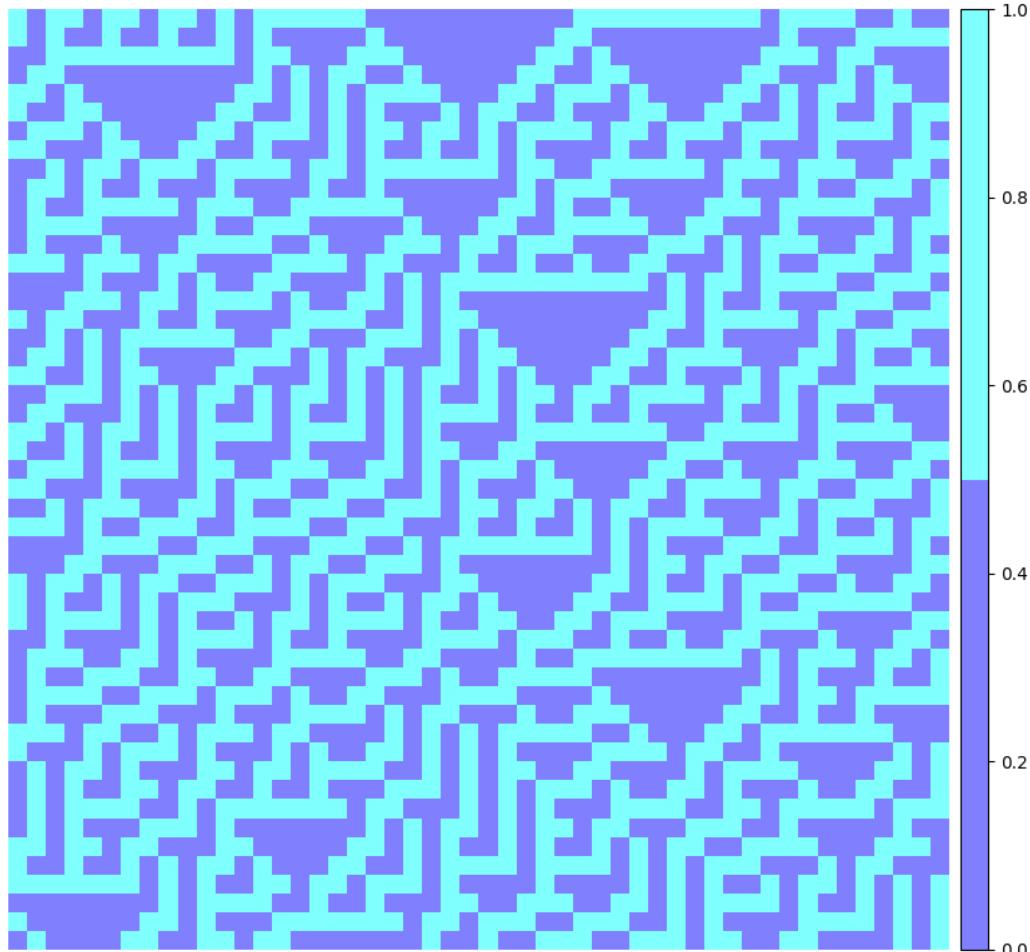


Figura 1: Autómata celular elemental aleatorio de 50 células evolucionado 50 veces

1. Introducción

Los autómatas celulares o por sus siglas (AC) surgen en la década de 1940 con John Von Neumann y fue descrito en su libro "Theory of Self-reproducing Automata", este tipo de autómatas son modelos matemáticos que valga la redundancia modelan sistemas dinámicos, los cuales evolucionan con el paso de tiempo, John Von Neumann tenía como objetivo modelar una máquina, que fuese capaz de auto replicarse, al intentar esto llegó a un modelo matemático, el cual describe a dicha máquina con ciertas reglas sobre una red rectangular. Su nombre se debe a esta similitud con el crecimiento de las células.

2. Definición

Como ya se mencionado anteriormente, un autómata celular es un modelo matemático para un sistema dinámico, este sistema evoluciona con el paso del tiempo. El autómata celular está compuesto por un conjunto células o celdas las cuales adquieren distintos valores o estados. Al ser este un sistema dinámico y al evolucionar a través del tiempo estos estados o valores que poseen las células son alterados de un instante a otro en un tiempo discreto, es decir, conocemos que valores toman nuestras células o celda en un cierto punto en el tiempo, en otras palabras es posible hacer una cuantización. Siendo así, el conjunto de células evolucionan según la expresión matemática, la cual evolucionará según los estados de las células vecinas, a esto se le conoce como regla de transición local.

3. Elementos de un AC

3.1. Un espacio rectangular

El autómata celular está definido ya sea en un espacio de dos dimensiones o bien en un espacio de n dimensiones, este es el espacio de evoluciones y cada una de las divisiones de este espacio es llamada célula.

3.2. Conjunto de estados

Los estados son finitos y cada elemento de la célula tomará un valor de este conjunto de estados. A cada vecindad diferente le corresponde un elemento del conjunto de estados.

3.3. Configuración inicial o tiempo 0

Es la asignación inicial de un estado a cada una de las células del espacio.

3.4. Función local o Función de transición local

Es la regla de evolución que determina el comportamiento del autómata celular. Esta regla está conformada por una célula central y sus vecindades. También esta define como debe cambiar de estado cada una de las células dependiendo de los estados de las vecindades anteriores. Esta función puede ser representada como una función algebraica o como un conjunto de ecuaciones.

4. Tipos de límites o fronteras

Podemos hacer una representación visual de los autómatas celulares, y para que podamos entenderlo de mejor manera es necesario mencionar los límites y las fronteras, del espacio en el cual existe el autómata celular

4.1. Frontera abierta

Considera que todas las células fuera del espacio del autómata tienen un valor el cual es fijo.

4.2. Frontera reflectora

Las células fuera del espacio del autómata toman los valores que están dentro como si se tratase de un espejo.

4.3. Frontera periódica o circular

Las células que están en los límites o en la frontera interaccionan con sus vecinos inmediatos y con las células que están en el extremo opuesto del arreglo, como si el plano estuviese doblado a manera de cilindro.

4.4. Sin frontera

La representación des autómata no tiene limites, en otras palabras es infinito.

5. Clasificación de los Autómatas Celulares

Stephen Wolfram comenzó a trabajar en autómatas celulares a mediados de 1981 después de considerar cómo los patrones complejos parecían formarse en la naturaleza en violación de la segunda ley de la termodinámica. Sus investigaciones fueron inicialmente impulsados por un interés en sistemas de modelado, como las redes neuronales. Tras ver la inesperada complejidad del comportamiento de estas reglas simples Wolfram llevó a sospechar que la complejidad en la naturaleza puede ser debida a mecanismos similares. En 2002 Wolfram publicó su libro *A New Kind of Science*, que sostiene ampliamente que los descubrimientos sobre autómatas celulares no son hechos aislados sino que son robustos y tienen importancia para todas las disciplinas de la ciencia.

Wolfram define cuatro clases en las que los AC. Mientras que los estudios anteriores en autómatas celulares tienden a tratar de identificar el tipo de patrones de reglas específicas, la clasificación de Wolfram fue el primer intento de clasificación global. En orden de complejidad las clases que identifica son:

5.1. Clase I

Casi todos los patrones iniciales evolucionan rápidamente en un estado estable y homogéneo. Cualquier aleatoriedad en el patrón inicial desaparece.

5.2. Clase II

Casi todos los patrones iniciales evolucionan rápidamente hacia estructuras estables u oscilantes. Parte de la aleatoriedad del patrón inicial puede permanecer, pero solo algunos restos. Los cambios locales en el patrón inicial tienden a permanecer locales.

5.3. Clase III

Casi todos los patrones iniciales evolucionan de forma pseudo-aleatoria o caótica. Las estructuras estables que aparecen son destruidas rápidamente por el ruido circundante. Los cambios locales en el patrón inicial tienden a propagarse indefinidamente.

5.4. Clase IV

Casi todos los patrones iniciales evolucionan en las estructuras que interactúan de manera compleja e interesante, con la formación de las estructuras locales que son capaces de sobrevivir por largos períodos de tiempo. Podría ser el caso de que apareciesen estructuras estables u oscilantes, pero el número de pasos necesarios para llegar a este estado puede ser muy grande, incluso cuando el patrón inicial es relativamente simple. Los cambios locales en el patrón inicial pueden extenderse indefinidamente. Wolfram ha conjecturado que muchos, si no todos, los AC de esta clase son capaces de realizar computación universal. Algo que ha sido demostrado para el autómata 110 y para el juego de la vida de John Conway.

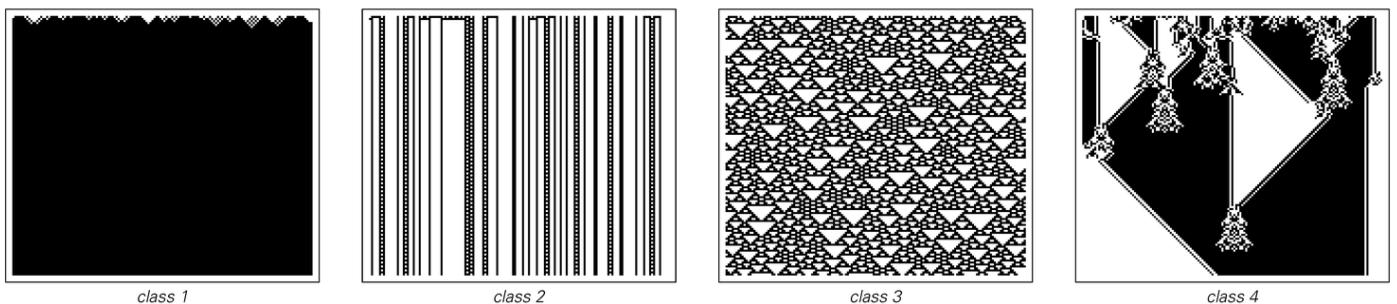


Figura 2: Ejemplo de la clasificación de los autómatas celulares

6. Programa

6.1. Descripción

Para este primer programa se ha creado nuestro primer autómata celular, el autómata celular a crear debido a sus características se le conoce como un autómata celular elemental (ECA) que se regirá por la regla 30 (hay 256 reglas de la 0 hasta la 255), pero ¿Cuáles son esas características para que nuestro autómata celular sea considerado como un ECA? Las características son las siguientes:

- Espacio unidimensional.
- Tiene dos posibles estados o valores para cada célula (0 o 1).
- La reglas que dependen solo de los valores del vecino más cercano.
- Es de tipo frontera periódica o circular.

Nuestro ECA sera regido por la regla 30 o 00011110_2 , la cual nos estable nuestra función local, es decir, nos define como las células cambiarian de estado en la siguiente iteración dependiendo de los estados o valores de la generación anterior teniendo en cuenta que necesitaremos de una célula central y dos células vecinas, una a cada lado, por lo que el comportamiento seria el siguiente:

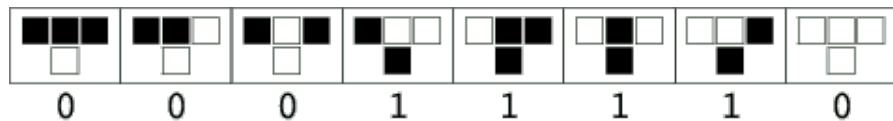


Figura 3: Comportamiento de la regla 30

En la imagen 3 podemos ver 8 figuras diferentes entre si, si vemos la primera fila compuesta de 3 células son las combinaciones posibles de los estados (0 o 1, blanco o negro) la célula de abajo nos da el estado que poseerá la hija de la siguiente iteración, es decir, si encontramos 3 células con estado 1 (color negro) en la siguiente iteración nos generaran una célula con el estado 0 (color blanco) o por ejemplo si encontramos que nuestras dos primeras células tienen estado 0 y la ultima tiene el estado 1 entonces la célula de la próxima iteración tendrá el valor 1.

Para el desarrollo de este programa se uso el lenguaje de programación Python usando Tkinter y Matplotlib para la creación de nuestra GUI y las gráficas correspondientes, el entorno de desarrollo utilizado fue Visual Studio Code. Mencionar que aunque nuestro programa nos permite usar cualquiera de las 256 reglas, en esta ocasión solo nos enfocaremos en la regla 30. Mencionar que nuestro archivo principal y el que se debe de ejecutar es el archivo con el nombre Interfaz.py ya que el archivo Logica.py solo contiene la parte lógica del programa y parte de la graficación de los resultados.

6.2. Pruebas

Este programa tiene varios aspectos que han sido cubiertos. Para iniciar las pruebas crearemos un autómata de 1000 células con 1000 iteraciones, algo especial de este autómata es que tenemos solo una célula con valor 1 en medio de todas las demás células que tienen valor 0.

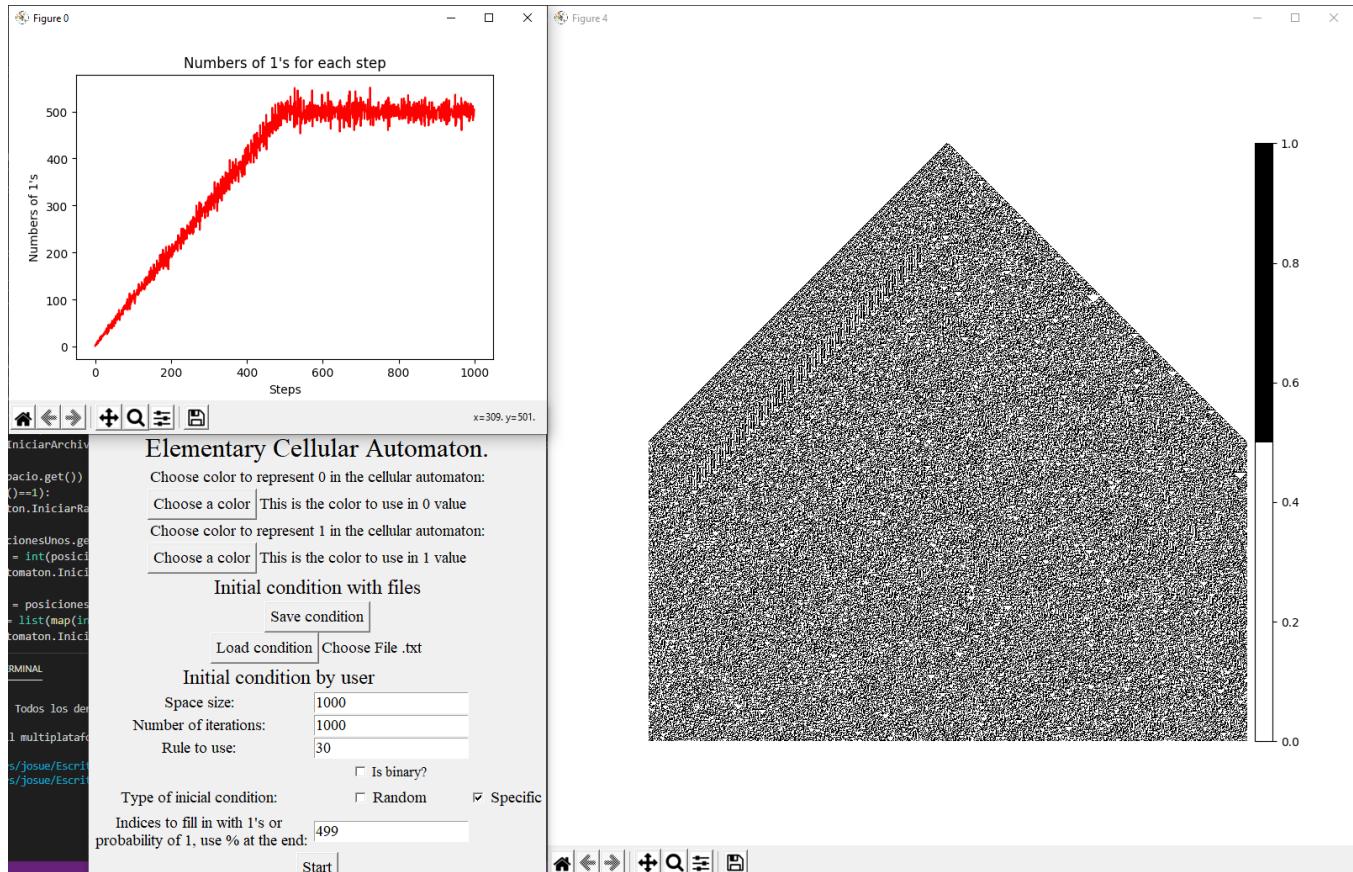


Figura 4: ECA con solo una célula con valor 1. Regla 30

Como vemos en la figura 4 tenemos en el lado derecho de la imagen nuestro ECA resultante después de aplicar 1000 iteraciones con la regla 30 y en la parte superior izquierda tenemos el crecimiento de células con valor 1 durante esas 1000 iteraciones, podemos apreciar que su crecimiento nos recuerda a un crecimiento logarítmico. En ambas gráficas tenemos la posibilidad de agrandar el tamaño mediante el ícono de la lupa que tenemos en la parte inferior de ambas ventanas, una vez seleccionada la lupa podremos seleccionar un área rectangular en nuestra gráfica para poder agrandar lo que abarque dicha área, esto con el fin de visualizar aun mejor nuestro autómata y el crecimiento de nuestras células con valor 1 como se ve en la figura 5.

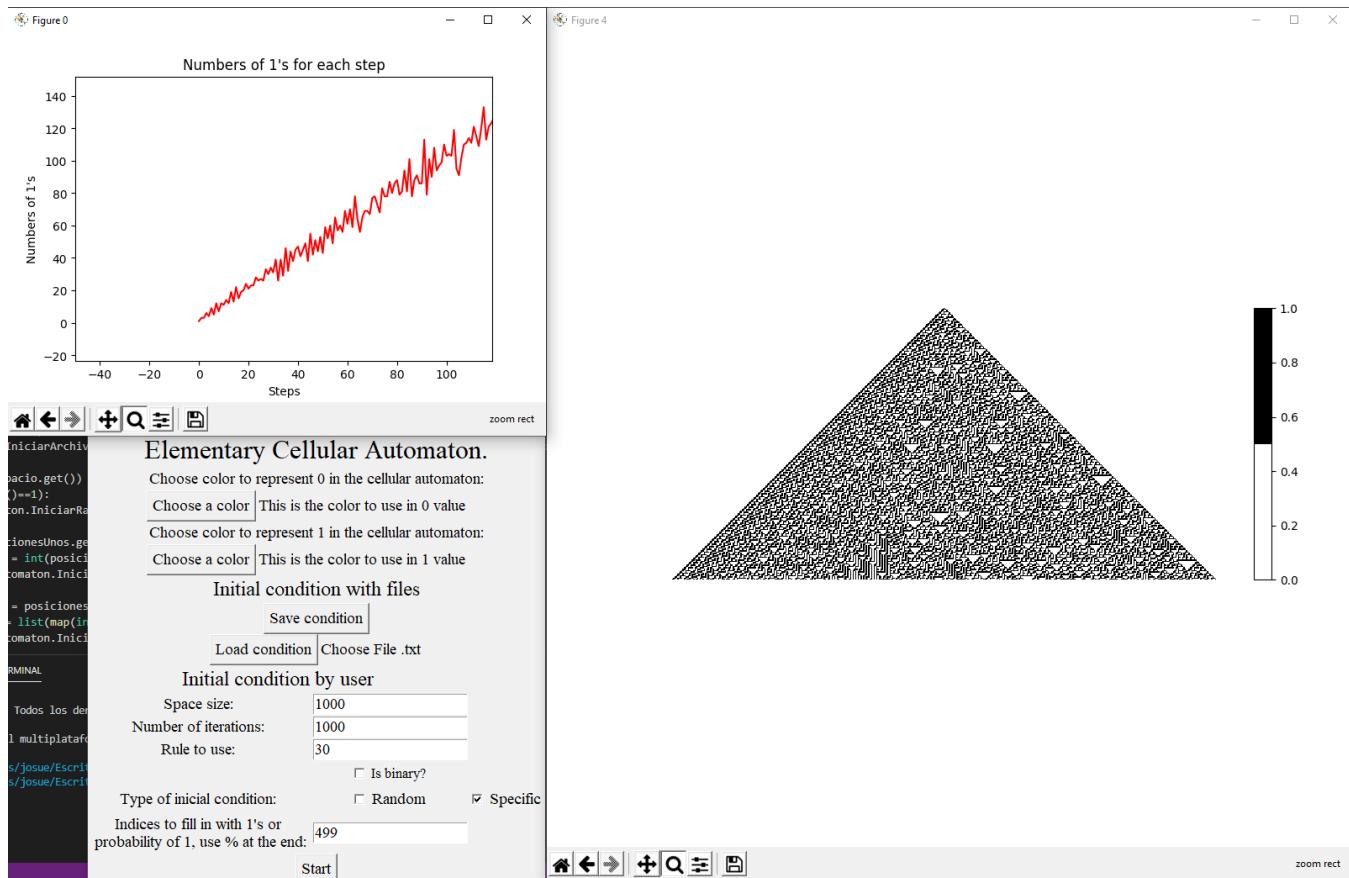


Figura 5: ECA con solo una célula con valor 1 agrandando el tamaño para poder visualizar aun mejor el comportamiento. Regla 30

Nuestro programa también nos permite guardar nuestra configuración de nuestro ECA, en este caso solo toma la iteración 0 y es guardado en un archivo con el siguiente nombre "celulasIniciales.txt" en caso de existir sobre escribirá la nueva información, en caso contrario lo creara desde cero, mencionar que si se desea se puede modificar el nombre pero no la extensión posterior de haber creado el archivo. En este caso nuestro programa nos arroja un mensaje diciendo que el archivo fue guardo exitosamente y se nos crea un archivo con el nombre y extensión anteriormente mencionados (ver figura 6 y 7).

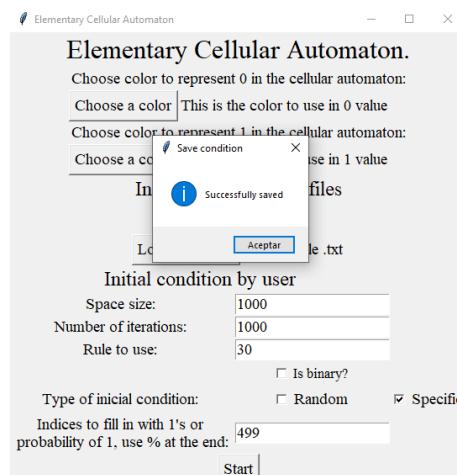


Figura 6: Iteración 0 de nuestro ECA guardada exitosamente

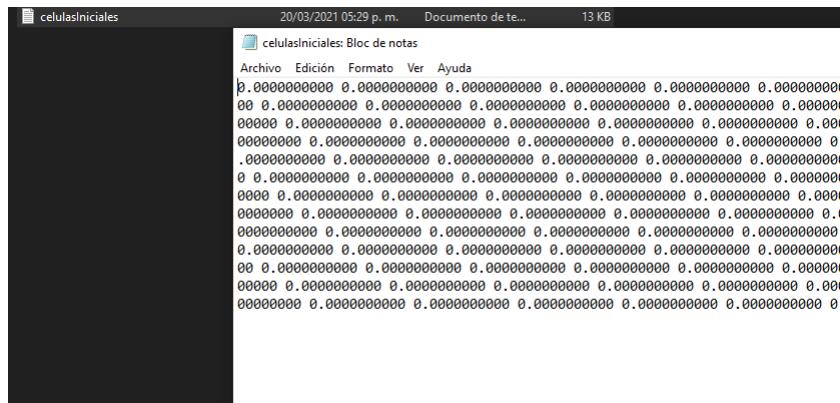


Figura 7: Contenido del archivo

También como era de esperarse se puede cargar archivo de texto al programa para volver a usarlo, algo importante a mencionar es que no guarda ni el numero de iteraciones, los colores para los valores 0 y 1 ni la regla usada esto con fines de facilitar al usuario poder usar un mismo inicio con diferentes reglas y numero de iteraciones cada vez que le plazca hacerlo. En la figura 8 cargamos nuestro archivo y solo modificamos el color que se le asigna a los valores 0 y 1, colocamos la regla a usar (regla 30) y nuestro numero de iteraciones sera de nuevo 1000, el panel para poder elegir el color se puede observar en la figura 9.

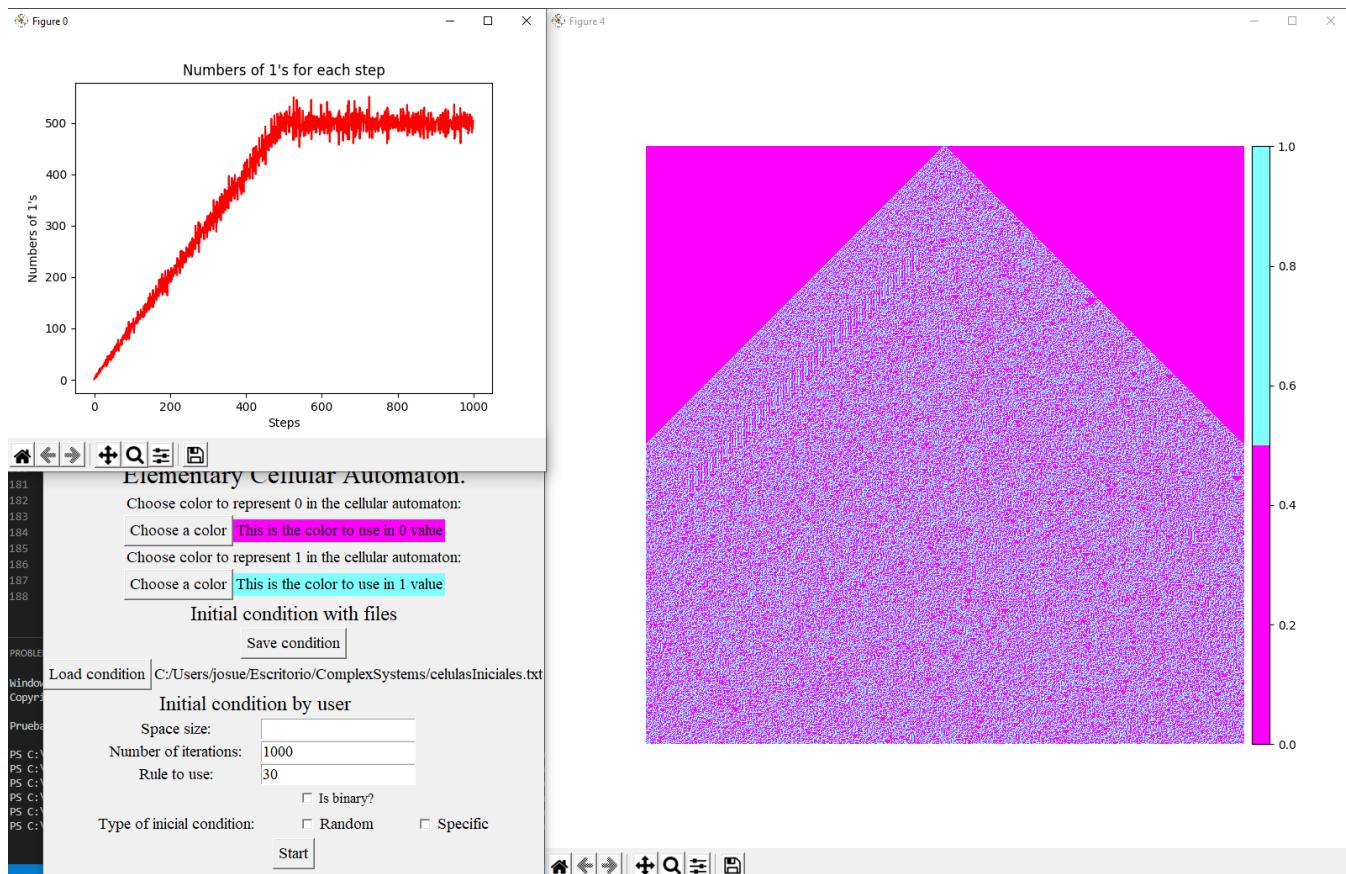


Figura 8: ECA con diferentes colores para los valores 0 y 1

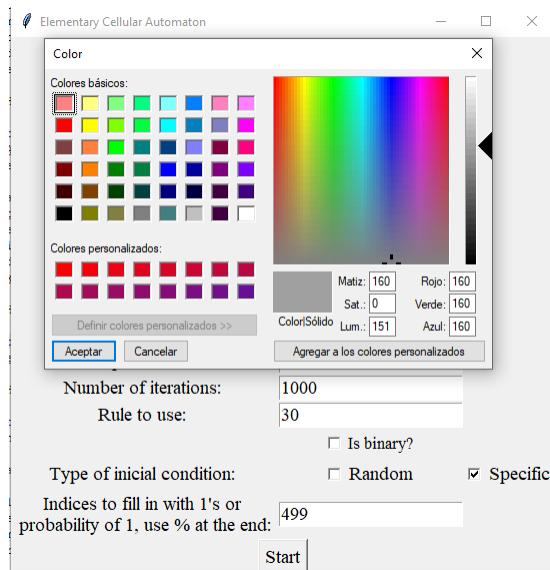


Figura 9: Panel para poder elegir color para valores 0 y 1

Realizando otra prueba crearemos un ECA con un tamaño de 500 células con 500 iteraciones, pero en esta ocasión la regla 30 sera ingresada como numero binario (00011110_2) y serán asignadas células con valor 1 de forma aleatoria.

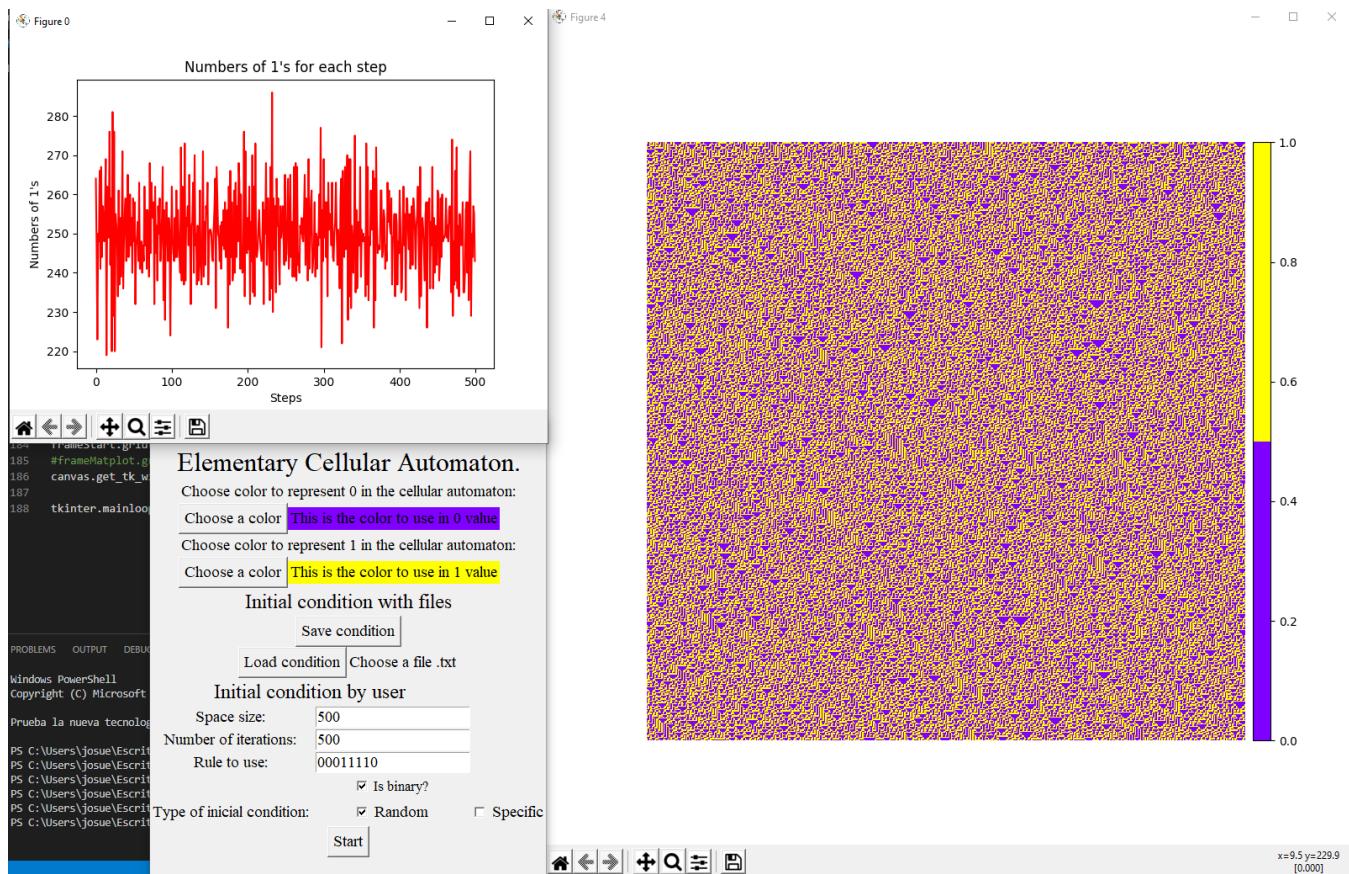


Figura 10: ECA de 500 células por 500 iteraciones creado de forma aleatoria

Como ultimas pruebas realizaremos lo siguiente: mediante las reglas 15, 22, 30, 54, 110 y 126 para la construcción de nuestro ECA mostraremos 3 evoluciones por cada regla en espacios de 400 células por 400 iteraciones, una empezara con un uno al centro, otra iniciando con un 50 % de probabilidad de unos y la última iniciando con un 95 % de probabilidad de unos.

Empecemos así con nuestra regla 15, veremos a través de las imágenes 11 a la 14 el comportamiento de nuestro ECA con base en donde se encuentren las células iniciales con valor 1.

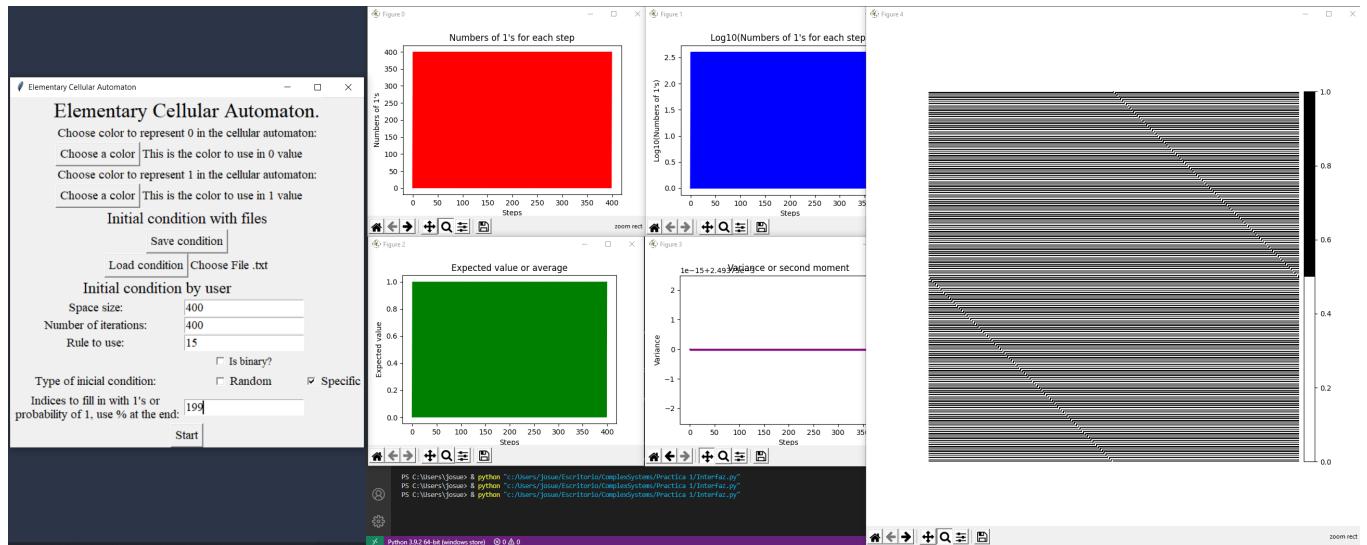


Figura 11: ECA de 400 células por 400 iteraciones con una célula inicial central, regla 15

Para visualizar mejor la información nuestras gráficas hagamos una ampliación de las mismas para las primeras 50 iteraciones.

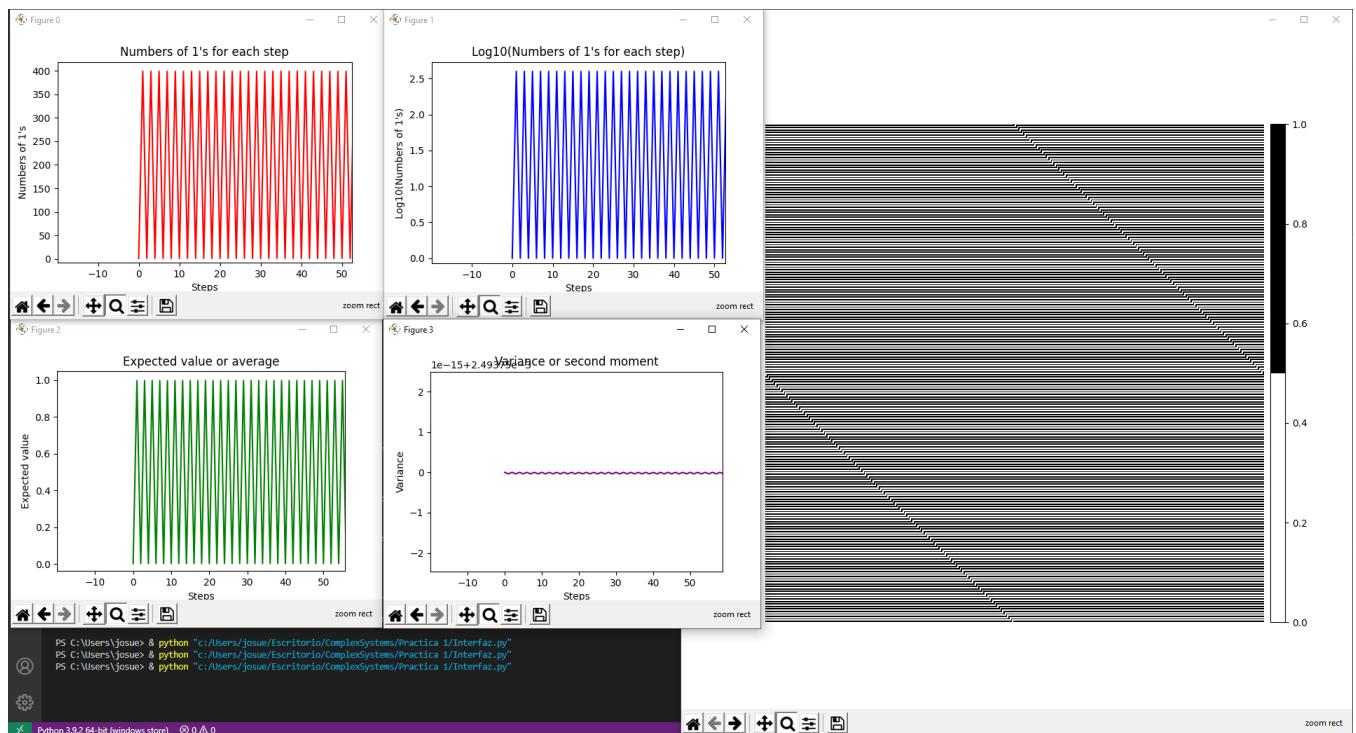


Figura 12: ECA de 400 células por 400 iteraciones con una célula inicial central con aumento, regla 15

Como vemos el comportamiento de nuestro ECA es de una forma constante y es que en la iteración tenemos 1 célula viva a pasar a tener 399 células y a la siguiente volvemos a tener solo una, como si de un ciclo se tratase.

Pasando a la siguiente prueba, usaremos el 50 % de unos y observemos su comportamiento

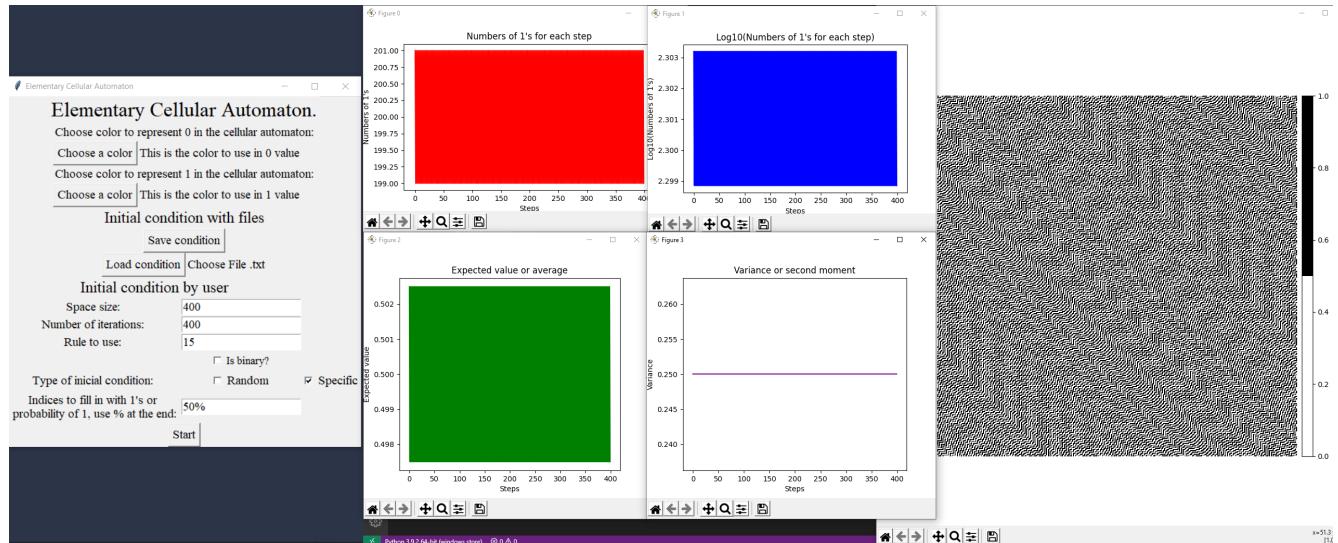


Figura 13: ECA de 400 células por 400 iteraciones con 50 % de probabilidad de 1's, regla 15

Como vemos tiene un comportamiento similar a la primera prueba solo que el numero de células vivas por iteración cambiaron, ahora son menos que en la prueba anterior. Como ultima prueba para esta regla, la probabilidad a usar sera del 95 % así que veamos el resultado

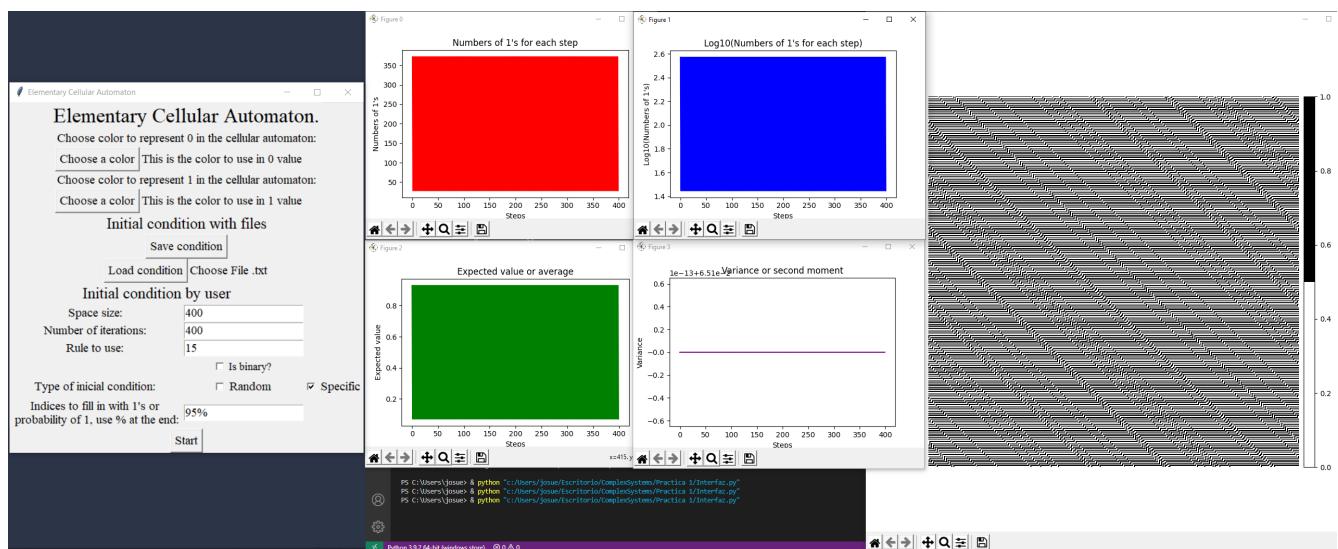


Figura 14: ECA de 400 células por 400 iteraciones con 95 % de probabilidad de 1's, regla 15

El resultado nuevamente es muy similar a los anteriores, por lo que podemos suponer que si probamos diferentes configuraciones iniciales siempre serán parecidas, solo cambiarian el numero de 1's.

Continuando con las pruebas, ahora vayamos con la regla 22 de aquí en adelante se mostraran las 3 imágenes correspondientes con los 3 valores iniciales diferentes para posteriormente dar un pequeño

estudio de los resultados.

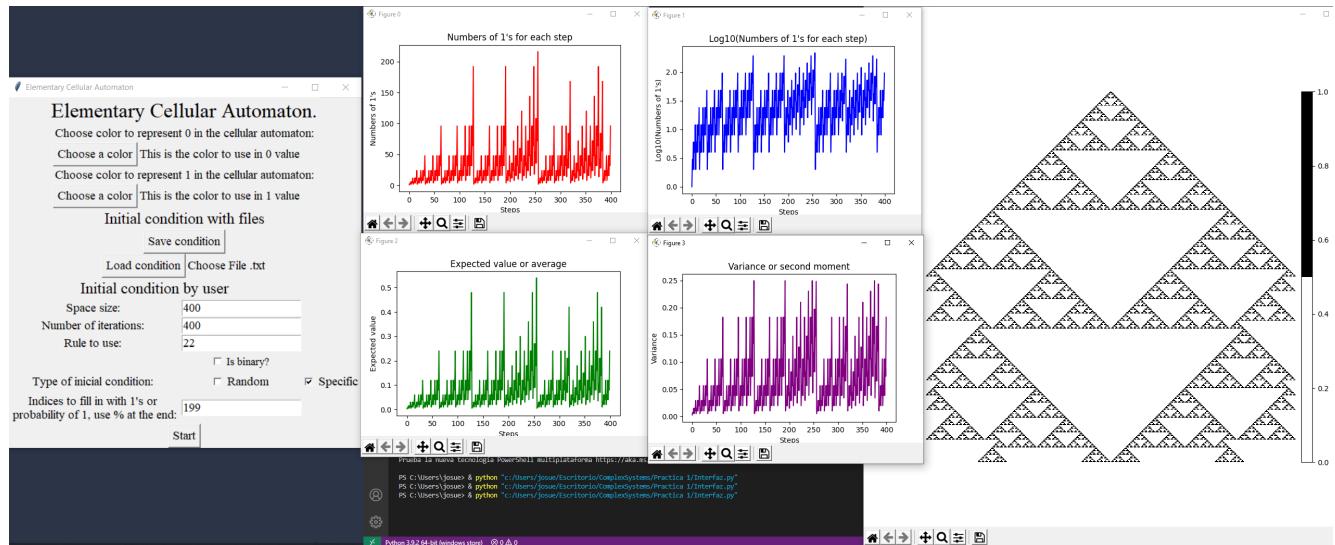


Figura 15: ECA de 400 células por 400 iteraciones con una célula inicial central, regla 22

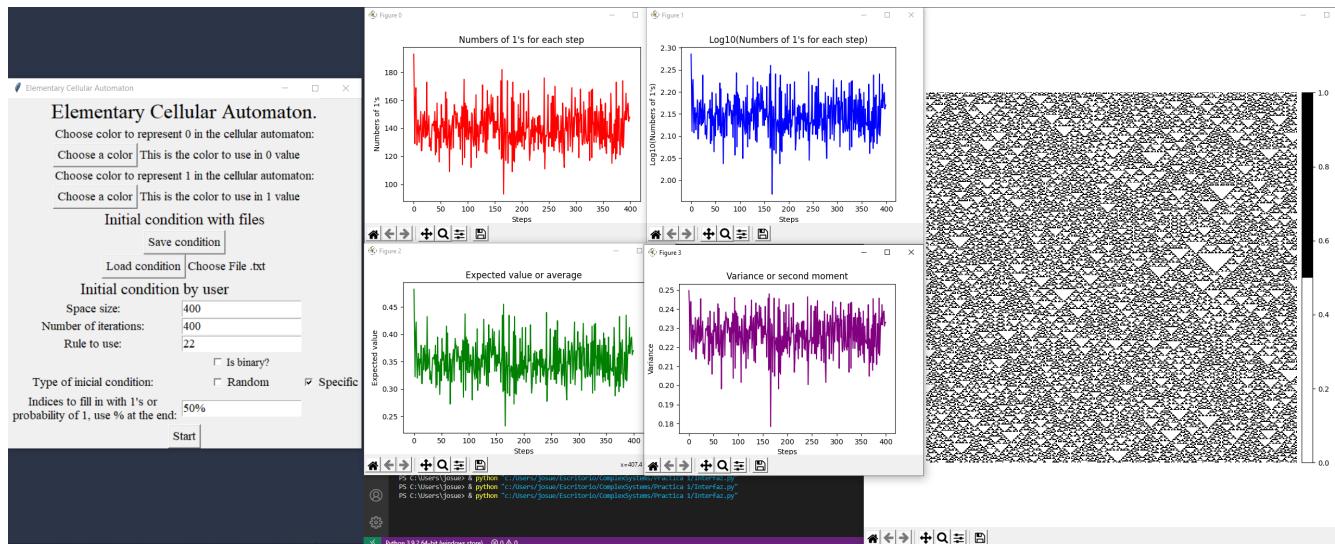


Figura 16: ECA de 400 células por 400 iteraciones con 50 % de probabilidad de 1's, regla 22

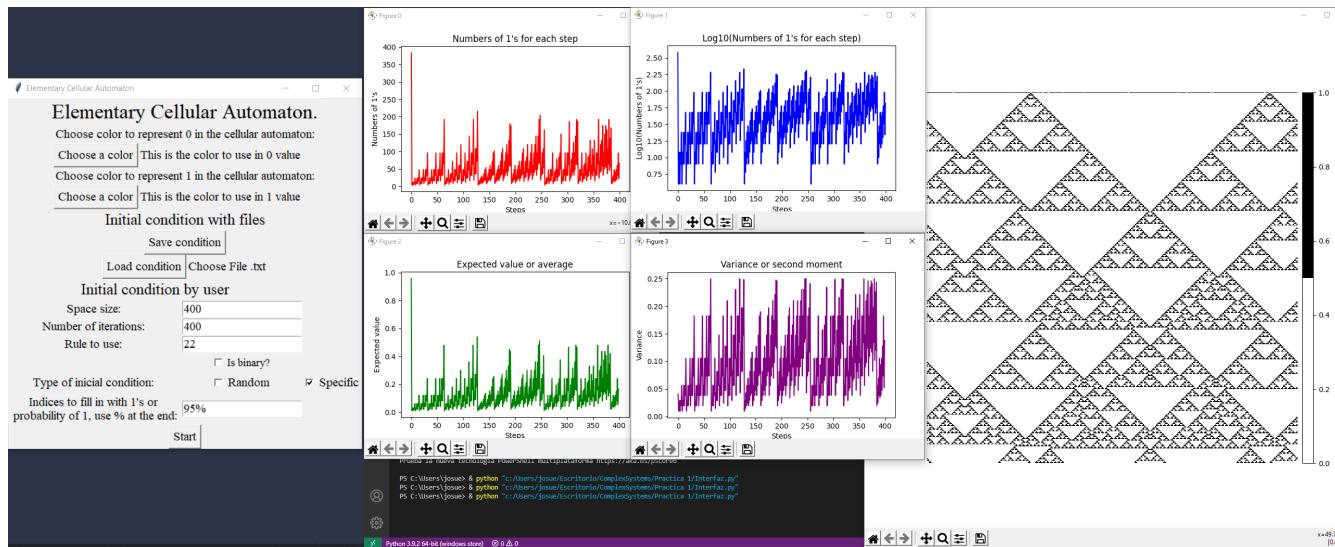


Figura 17: ECA de 400 células por 400 iteraciones con 95 % de probabilidad de 1's, regla 22

Como se ven desde la figura 15 a la 17 con esta regla si cambia el comportamiento de nuestras gráficas con base a cada tipo de condición inicial, aunque la figura 15 y 17 tiene ciertas similitudes pero no son lo mismo, empezzamos a ver que tan influyente es la regla en el comportamiento de nuestro ECA.

Ahora pasemos a la regla 30 con las mismas pruebas.

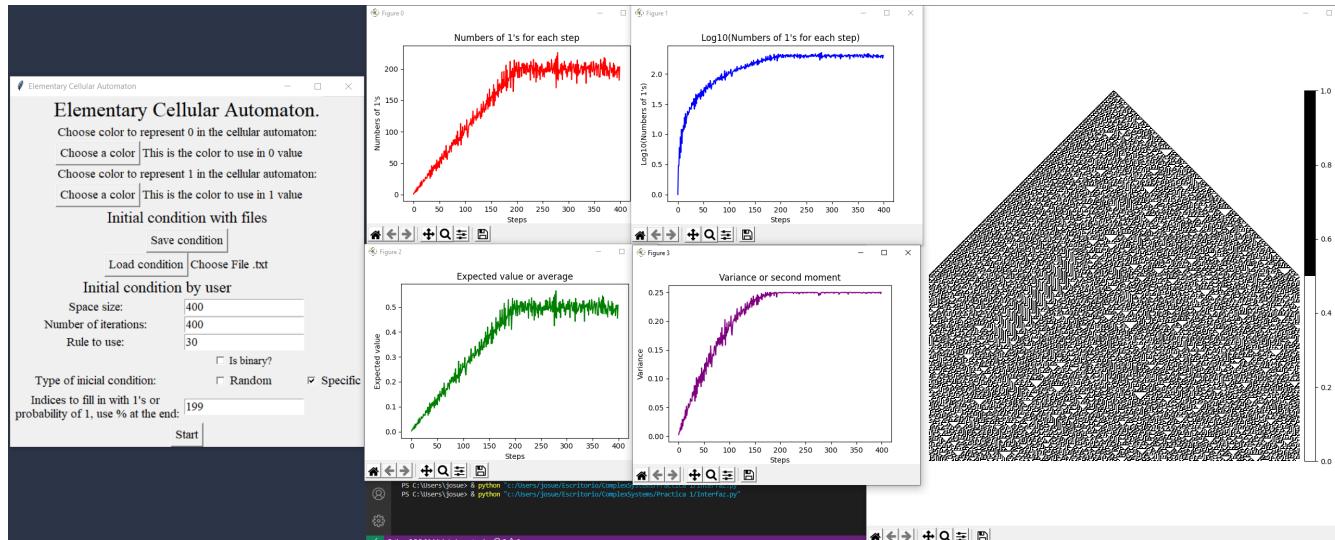


Figura 18: ECA de 400 células por 400 iteraciones con una célula inicial central, regla 30

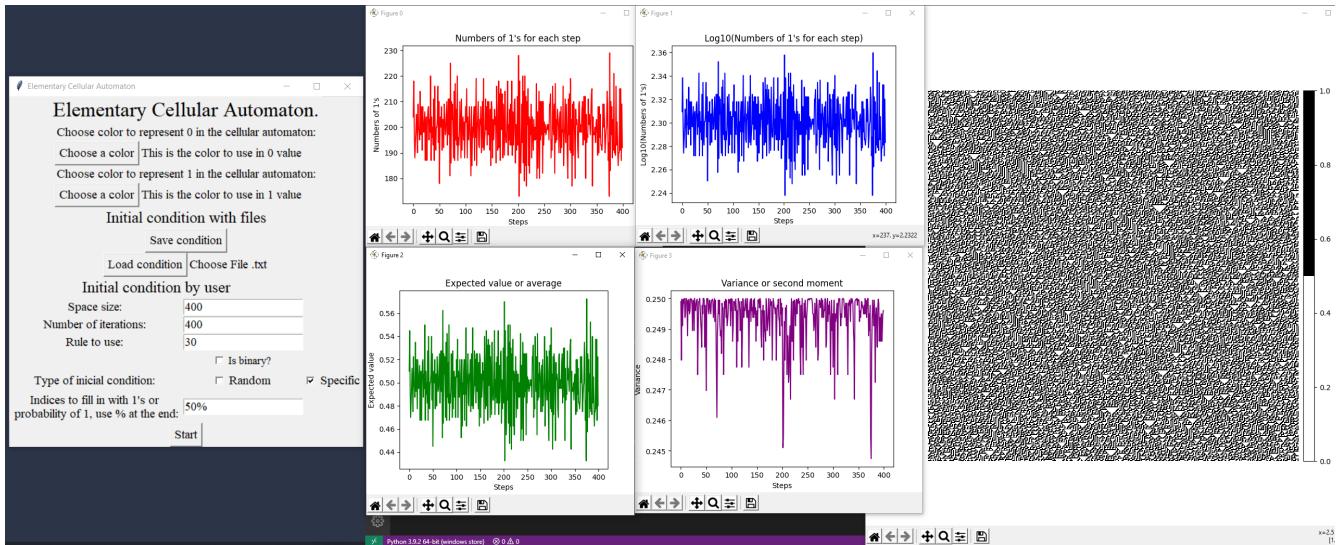


Figura 19: ECA de 400 células por 400 iteraciones con 50 % de probabilidad de 1's, regla 30

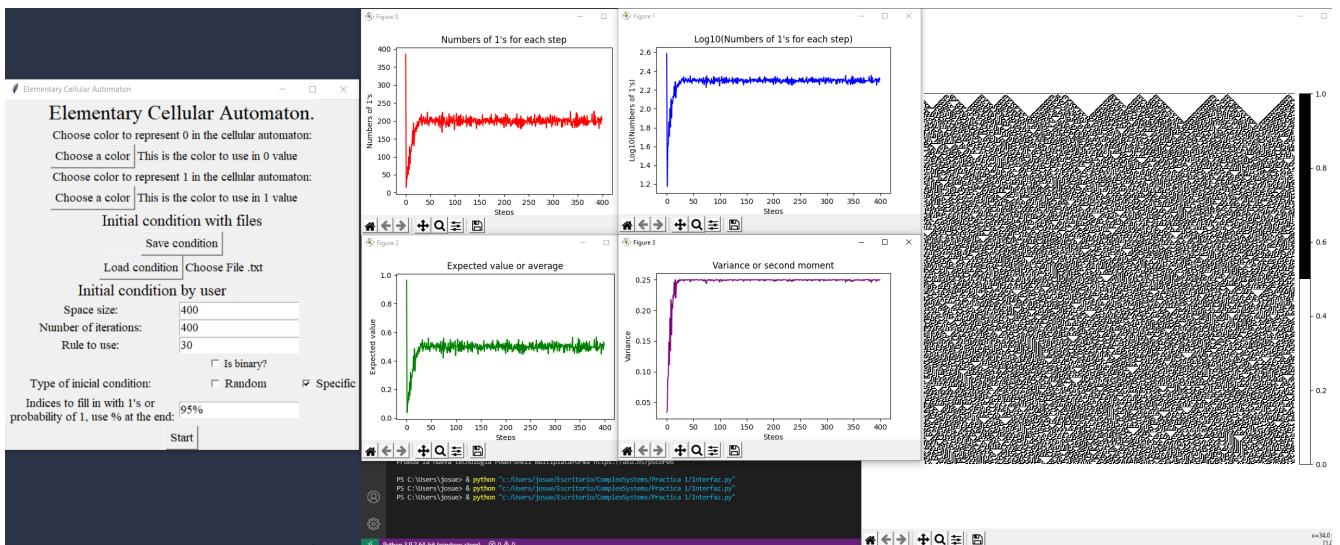


Figura 20: ECA de 400 células por 400 iteraciones con 95 % de probabilidad de 1's, regla 30

Como vemos en las figuras 18 a la 20 esta regla nos recuerda el comportamiento de la regla anterior, en el sentido de que las figuras 18 y 20 son muy similares en su comportamiento el cual es como un crecimiento logarítmico, en cambio la figura 19 tiene un comportamiento caótico.

Continuando, ahora veremos la regla 54.

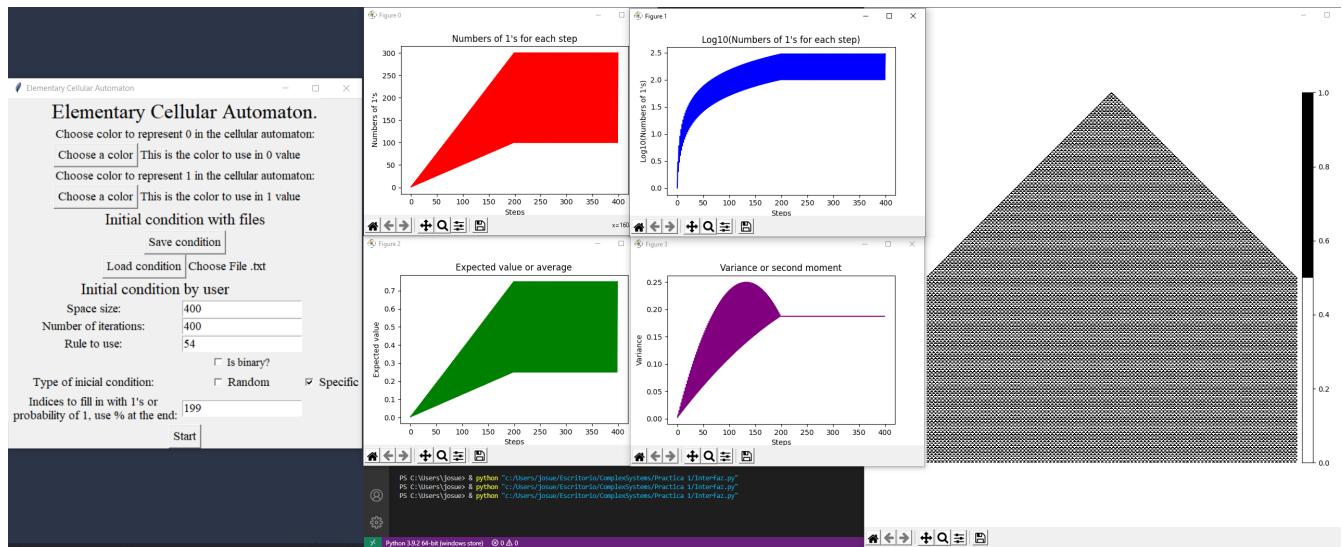


Figura 21: ECA de 400 células por 400 iteraciones con una célula inicial central, regla 54

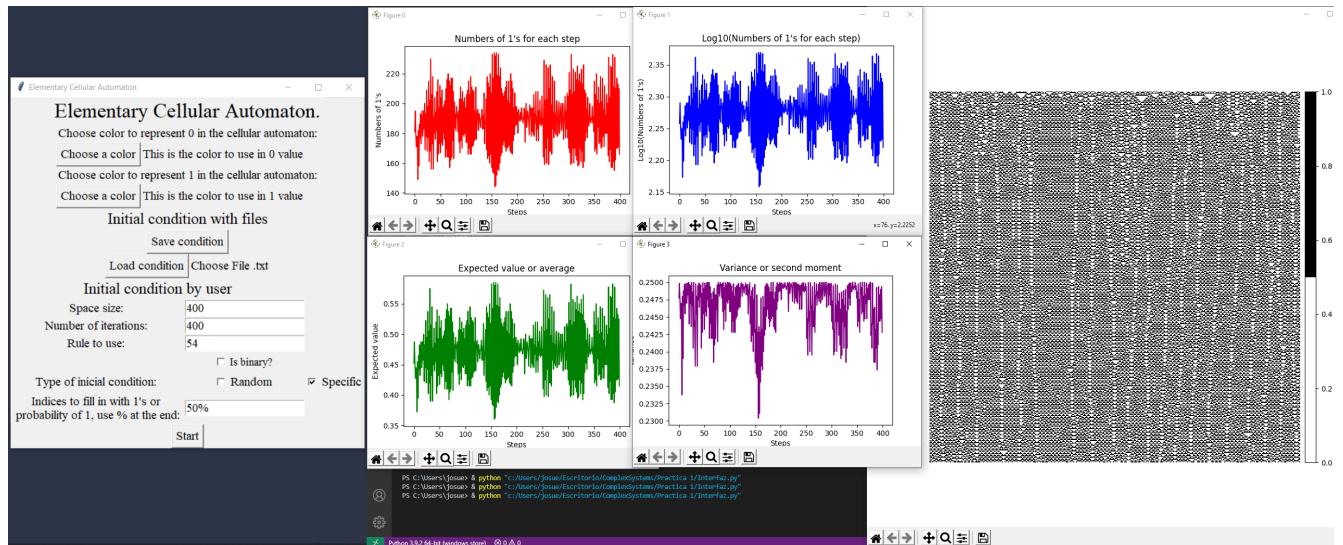


Figura 22: ECA de 400 células por 400 iteraciones con 50 % de probabilidad de 1's, regla 54

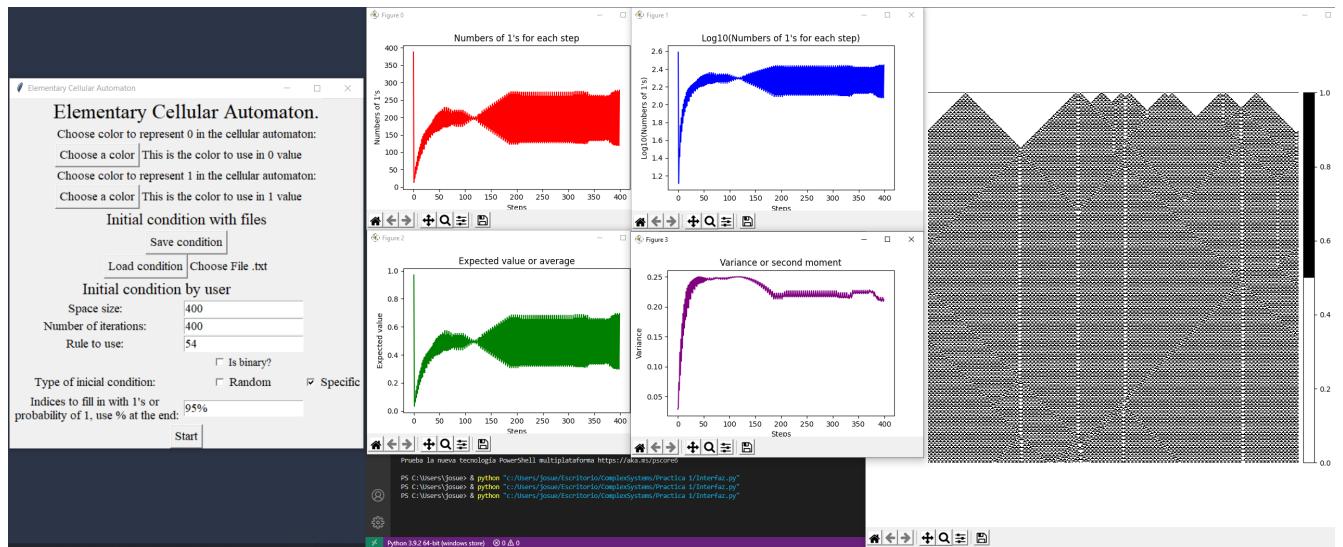


Figura 23: ECA de 400 células por 400 iteraciones con 95 % de probabilidad de 1's, regla 54

Con esta regla si observamos un cambio en nuestras 3 pruebas que las 3 son diferentes entre si, pero, si observamos la figura 22 y la 23 encontramos una similitud un poco sutil en nuestro ECA y es que ambos generan lineas rectas que "dividen" a nuestro ECA de formar vertical, caso contrario a la figura 21 que podríamos decir que esa división pasa a su forma horizontal

Pasando con la penúltima regla a probar la cual es la regla 110.

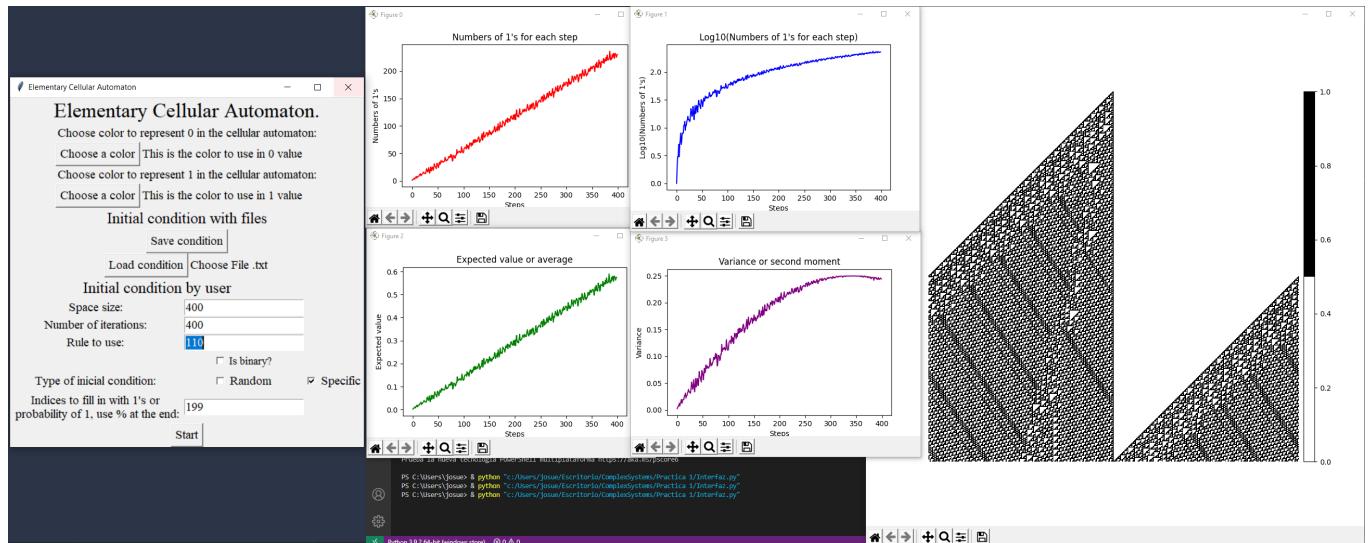


Figura 24: ECA de 400 células por 400 iteraciones con una célula inicial central, regla 110

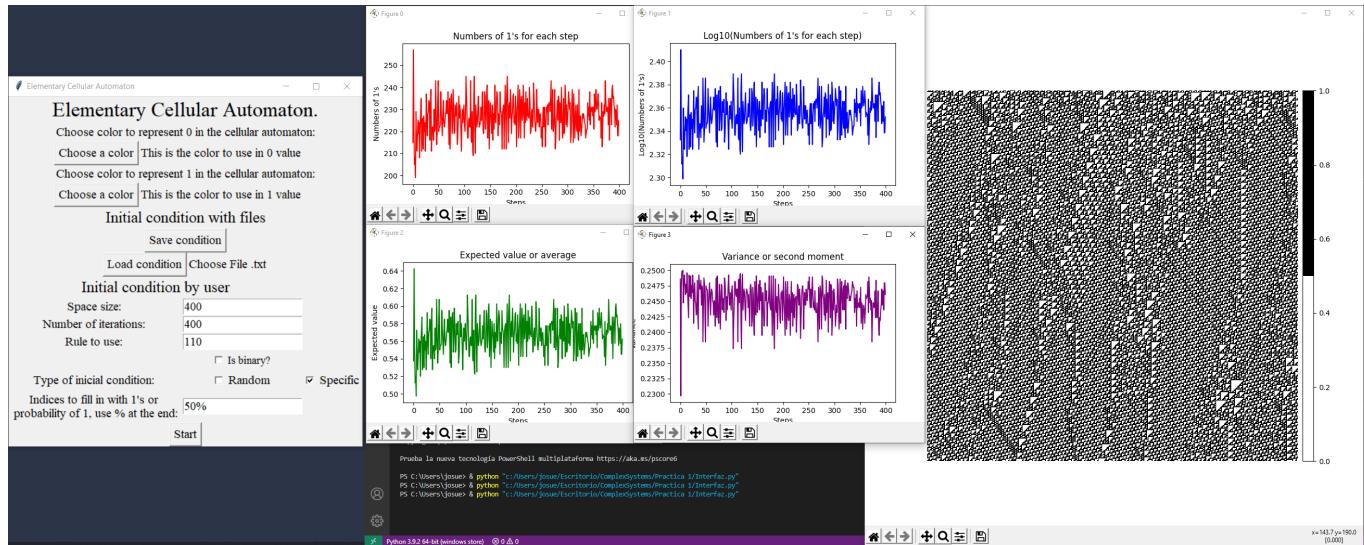


Figura 25: ECA de 400 células por 400 iteraciones con 50 % de probabilidad de 1's, regla 110

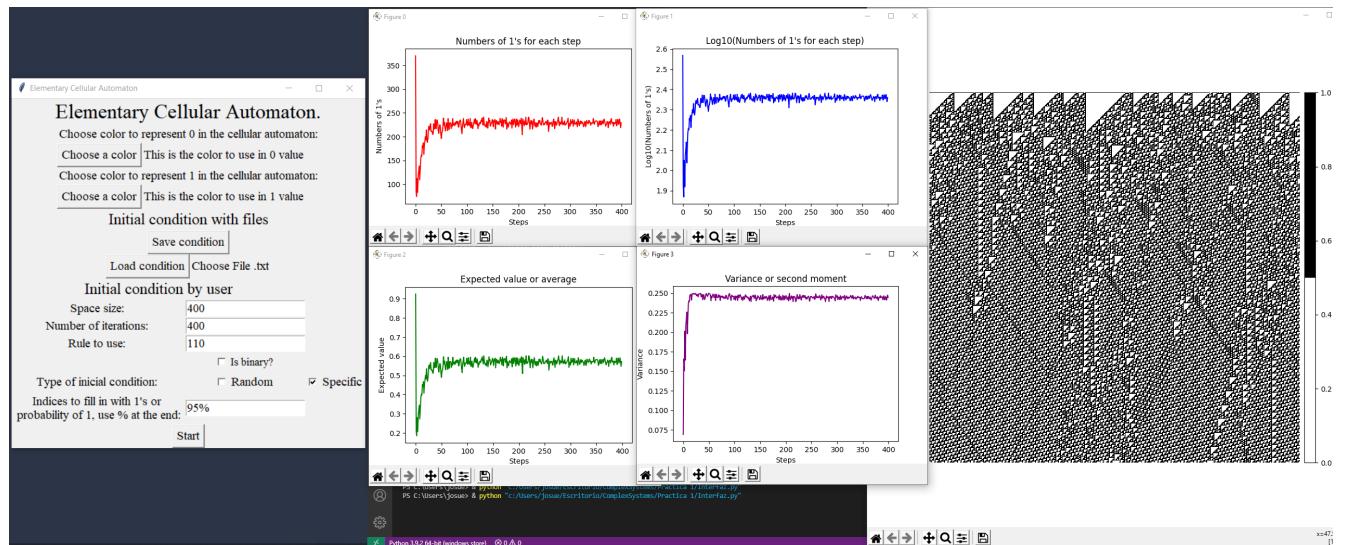


Figura 26: ECA de 400 células por 400 iteraciones con 95 % de probabilidad de 1's, regla 110

Sin duda es la regla con un comportamiento muy interesante, en la figura 24 vemos un crecimiento lineal en los números de células con valor 1, caso contrario de las figuras 25 y 26 las cuales tienen un comportamiento diferente la cual nos recuerda al crecimiento logarítmico, pero, mencionar que la figura 25 no se visualiza tan claramente este comportamiento, de hecho pareciera que estuvieramos algún tipo de filtro en las gráficas de la figura 25 y así obtenemos las gráficas de la figura 26.

Finalmente, tenemos la regla 126.

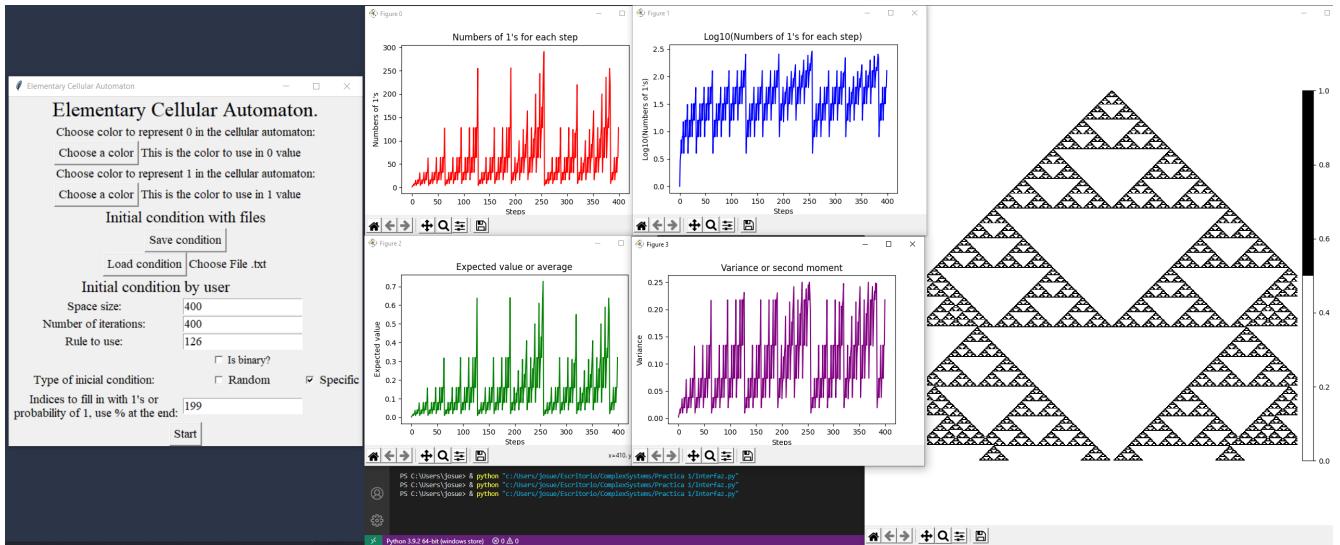


Figura 27: ECA de 400 células por 400 iteraciones con una célula inicial central, regla 126

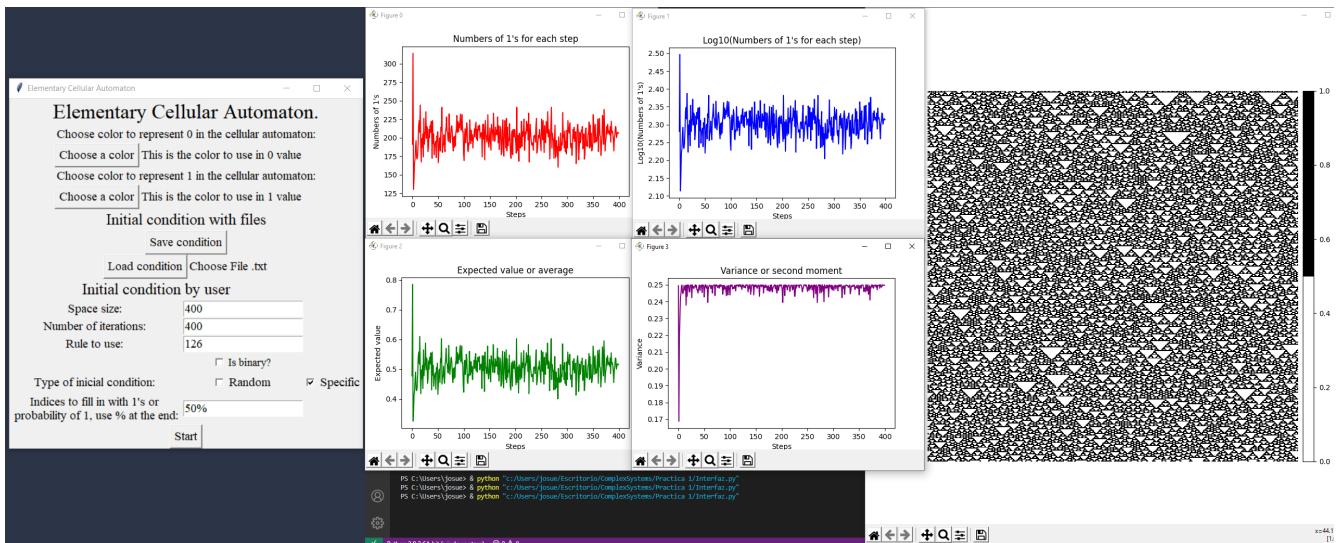


Figura 28: ECA de 400 células por 400 iteraciones con 50 % de probabilidad de 1's, regla 126

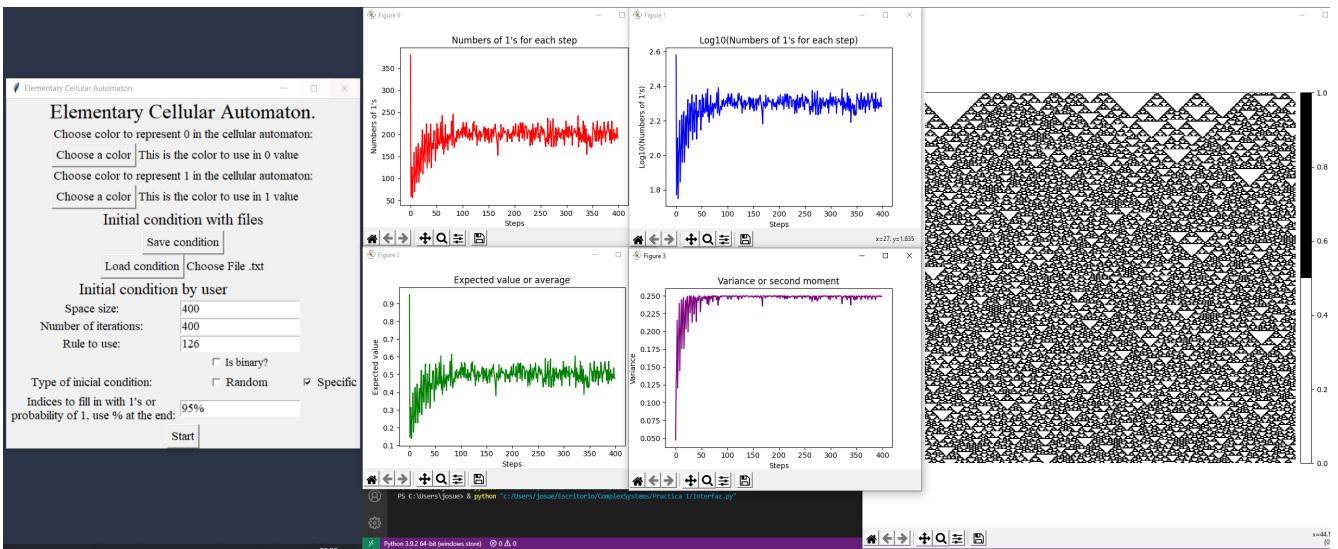


Figura 29: ECA de 400 células por 400 iteraciones con 95 % de probabilidad de 1's, regla 126

En el caso de la regla 126 nos recuerda a la regla 22 y es que las imágenes resultantes son muy parecidas, sin embargo, en los casos del 50 % y 95 % vemos como ahora las gráficas son mas definidas en contraste a la regla 22 y que de hecho se parecen a gráficas anteriores en donde mencionamos que se nos recuerda a un crecimiento logarítmico.

6.3. Añadidos

Como añadidos al programa se agregaron 3 tipos de gráficas, las cuales son las siguientes:

- Gráfica de logaritmo base 10 de la cantidad de células con valor 1 por cada iteración.
- Gráfica del valor esperado o promedio.
- Gráfica de la varianza o segundo momento.

La primera gráfica aplica el log a la cantidad de células con el valor 1 por cada iteración . Para la segunda gráfica se nos muestra el valor esperado para valores discretos el cual esta definido por la siguiente formula:

$$\mu = E[x] = \sum_x x * f(x)$$

En donde x toma los valores de 0 y nuestra $f(x)$ es la probabilidad para que x sea 0 o 1, para sacar esa probabilidad solo se cuenta la cantidad de unos por iteración y se divide por el tamaño de nuestro autómata, si nos detenemos por un momento en la formula nos damos cuenta de que no es necesario tener $f(x = 0)$ ya que se multiplicaría por un 0, por lo que nuestra formula para esa gráfica nos quedaría lo siguiente:

$$E[x] = \sum_x x * f(x) = 1 * f(x = 1) + 0 * f(x = 0) = f(x = 1)$$

$$E[x] = f(x = 1)$$

Por lo que $E[x]$ tomaría el valor de la cantidad de números de 1's entre la longitud del autómata.

Finalmente para la gráfica de la varianza o segundo momento se usa la siguiente formula para el caso discreto.

$$\sigma^2 = E[(x - \mu)^2]$$

La cual puede ser reducida mediante desarrollo matemático quedando de la siguiente manera:

$$\sigma^2 = E[(x - \mu)^2] = E[x^2 - 2x\mu + \mu^2]$$

$$\sigma^2 = E[x^2] - E[2x\mu] + E[\mu^2]$$

$$\sigma^2 = E[x^2] - 2\mu E[x] + \mu^2$$

$$\sigma^2 = E[x^2] - 2\mu\mu + \mu^2$$

$$\sigma^2 = E[x^2] - 2\mu^2 + \mu^2$$

Finalmente tenemos que:

$$\sigma^2 = E[x^2] - \mu^2$$

Por lo que, retomando lo mencionado para el desarrollo de la gráfica 2 o la gráfica de nuestro promedio o valor esperado solo necesitamos el valor de $f(x = 1)$, nuestra formula para la varianza y finalmente saber que $E[x^n] = \sum_x x^n * f(x)$, así, sustituyendo valores obtendríamos lo siguiente sera:

$$\sigma^2 = (1^2 * f(x = 1)) - f(x = 1)^2 = f(x = 1) - f(x = 1)^2$$

Finalmente la formula a usar para la varianza:

$$\sigma^2 = f(x = 1) - f(x = 1)^2$$

Ahora hagamos una prueba, en la cual la regla a usar sera la regla 30, un tamaño de 1000 células con 1000 iteraciones, colores por defecto y asignación de células con valor 1 de manera aleatoria.

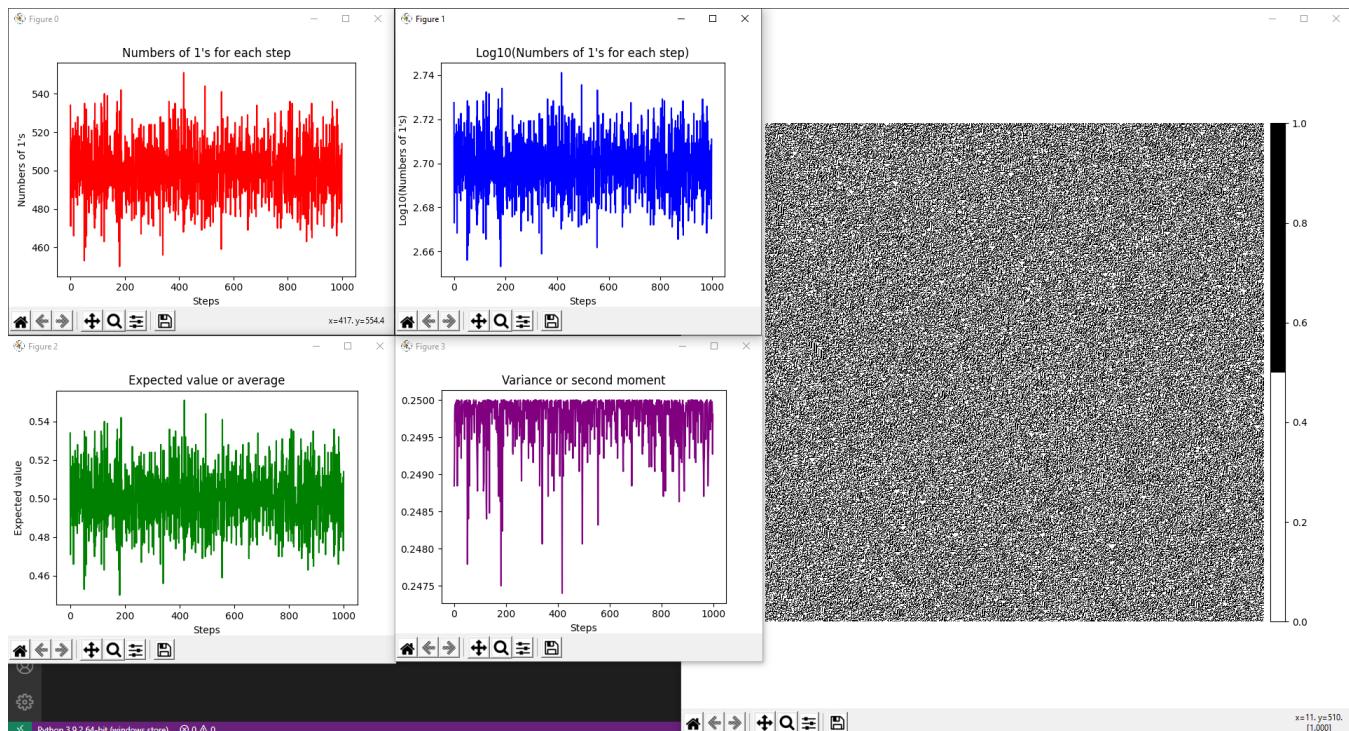


Figura 30: ECA de 1000 células por 1000 iteraciones creado de forma aleatoria

Como podemos observar tenemos 3 graficas que tienen una forma similar o casi iguales las cuales son la cantidad de 1's por iteración, \log de la cantidad de 1's y el valor esperado, esto es lógico ya que solo estamos cambiando la "escala" de una misma representación, la única gráfica que difiere de las demás es

la varianza y esto resulta, como dato adicional al momento de usar diferentes reglas el comportamiento de esta difiere e incluso en ocasiones llega a ser similar con las otras 3. Como nota adicional mencionar que a cada gráfica le podemos incrementar el tamaño de la gráfica para poder visualizar de una mejor visualización de los datos, lo anterior se realiza exactamente igual que a como se menciono unas hojas atrás.

Adicionalmente a lo anterior, la barra de opciones que tiene cada gráfica nos permite realizar diferentes cosas que en seguida enlistaremos, para comodidad tomar como referencia la figura 12 y ver los iconos de izquierda a derecha.

1. El icono de la casa nos permite reajustar la vista de nuestra gráfica a como lo era originalmente.
2. El icono de la flecha a la izquierda o derecha nos permite navegar entre las distintas vistas que hayamos realizado en una gráfica, es decir, si hicimos dos aumentos y queremos regresar al primero hacemos de la flecha hacia la izquierda y en caso de que estando en el primer aumento queremos regresar al segundo hacemos uso de la flecha a la derecha.
3. El icono que nos recuerda a una cruz nos permite mover la gráfica a voluntad, un ejemplo de uso seria que al hacer zoom en un área y querer mover ese mismo zoom en un área que tenemos a lado hacemos uso de esta función para poder mover nuestra gráfica.
4. El icono de la lupa como ya se ha mencionado nos permite aumentar el tamaño de cierta área en nuestra gráfica.
5. El icono que nos recuerda a configuración no recomiendo usarlo ya que cambia la configuración inicial de nuestra gráfica y en ocasiones nos arruina el como se ve la gráfica
6. El ultimo icono, que es el icono de disquete nos permite, como es incluso lógico, guardar la gráfica como una imagen con varias extensiones a elegir, esto nos permite poder guardar resultados como una imagen para el posterior uso.



Figura 31: Barra de opciones

6.4. Código

A continuación se anexa el código generado para la creación de este primer programa. Primero tenemos la parte de la interfaz de nuestro programa que se crea con el siguiente código.

```

1 #IMPORTAMOS LIBRERIAS NECESARIAS.
2 import tkinter
3 from tkinter import colorchooser
4 from tkinter.filedialog import askopenfilename
5 import numpy as np
6 import Logica as cellularAutomaton
7 #-----CREAR VENTANA
8
8 root = tkinter.Tk()
9 root.wm_title("Elementary Cellular Automaton")
10 framet = tkinter.Frame(root)
11 frameColor0 = tkinter.Frame(root)
12 frameColor1 = tkinter.Frame(root)
13 frameCondicion = tkinter.Frame(root)
14 frameMainInformacion = tkinter.Frame(root)
15 frameStart = tkinter.Frame(root)
16 #-----FUNCIONES
17
17 global color0
18 global color1
19 global arregloInicial
20 esBinario = tkinter.IntVar()
21 reglaValor = tkinter.StringVar()
22 tamEspacio = tkinter.StringVar()
23 numIteraciones = tkinter.StringVar()
24 esAleatorio = tkinter.IntVar()
25 esEspecifico = tkinter.IntVar()
26 posicionesUnos = tkinter.StringVar()
27
28 color0 = "#ffffff"
29 color1 = "#000000"
30 arregloInicial = []
31 def chageLabel():
32     if(esEspecifico.get() == 0):
33         labelEspecifico.grid_remove()
34         las.grid_remove()
35     else:
36         las.grid()
37         labelEspecifico.grid()
38
39 def colorto0():
40     global color0
41     color0 = colorchooser.askcolor()[1]
42     showColor0.config(bg=color0)
43

```

```

44 def colorto1():
45     global color1
46     color1 = colorchooser.askcolor()[1]
47     showColor1.config(bg=color1)
48
49 def saveCondic():
50     celulasIniciales = cellularAutomaton.resultadoAutomata[0,:]
51     with open('celulasIniciales.txt', 'wb') as f:
52         np.savetxt(f, np.column_stack(celulasIniciales), fmt='%.10f')
53     tkinter.messagebox.showinfo("Save condition", "Successfully saved")
54
55 def fileselectD():
56     global arregloInicial
57     try:
58         filename = askopenfilename()
59         if (filename.find(".txt")!=-1):
60             labelLoadCondicion.config(text = filename)
61             arregloInicial = np.loadtxt(filename)
62         else:
63             arregloInicial = []
64             labelLoadCondicion.config(text = "Choose a file .txt")
65     except:
66         arregloInicial = []
67         labelLoadCondicion.config(text = "File Error")
68
69 def iniciar():
70     numPasos = int(numIteraciones.get())
71     ruleEntero = int(reglaValor.get())
72     if(esBinario.get() == 1):
73         ruleString = reglaValor.get()
74         ruleEntero = int(ruleString,2)
75     if(len(arregloInicial) != 0):
76         cellularAutomaton.IniciarArchivo(color0,color1,numPasos,
77                                         ruleEntero,arregloInicial)#color 0, color 1, pasos,regla,
77     else:
78         tamEsp = int(tamEspacio.get())
79         if(esAleatorio.get()==1):
80             cellularAutomaton.IniciarRandom(color0,color1,numPasos,
81                                             ruleEntero,tamEsp)
81         else:
82             if '%' in posicionesUnos.get():
83                 unosEntero = int(posicionesUnos.get().replace('%', ''))
84                 )/100
84                 cellularAutomaton.IniciarProbabilidad(color0,color1,
85                 numPasos,ruleEntero,tamEsp,unosEntero)
85             else:
86                 unosString = posicionesUnos.get().split(',')
87                 unosArray = list(map(int, unosString))

```

```
88         cellularAutomaton.IniciarEspecifico(color0,color1,
89             numPasos,ruleEntero,tamEsp,unosArray)
90 #-----CREAR INTERFAZ
91 -----
92 titulo = tkinter.Label(framet, text="Elementary Cellular Automaton.",
93     font=("times new roman", 24))
93 titulo.pack(side="top")
94 -----
95 #-----COLOR
96 0-----
97 color0Label = tkinter.Label(frameColor0, text="Choose color to
98     represent 0 in the cellular automaton:",font=("times new roman",
99     14))
100 color0Label.pack(side="top")
101 buttonColor0 = tkinter.Button(frameColor0, text = "Choose a color",
102     command=colorto0,font=("times new roman", 14))
103 buttonColor0.pack(side="left")
104 showColor0 = tkinter.Label(frameColor0, text="This is the color to
105     use in 0 value",font=("times new roman", 14))
106 showColor0.pack(side="left")
107 -----
108 color1Label = tkinter.Label(frameColor1, text="Choose color to
109     represent 1 in the cellular automaton:",font=("times new roman",
110     14))
111 color1Label.pack(side="top")
112 buttonColor1 = tkinter.Button(frameColor1, text = "Choose a color",
113     command=colorto1,font=("times new roman", 14))
114 buttonColor1.pack(side="left")
115 showColor1 = tkinter.Label(frameColor1, text="This is the color to
116     use in 1 value",font=("times new roman", 14))
117 showColor1.pack(side="left")
118 -----
119 #-----CONDICIONES ARCHIVO
120 -----
121 tittleCondicion = tkinter.Label(frameCondicion, text = "Initial
122     condition with files" ,font=("times new roman", 18))
123 tittleCondicion.pack(side="top")
124 saveCondicion = tkinter.Button(frameCondicion, text = "Save condition
125     ", command=saveCondic ,font=("times new roman", 14))
126 saveCondicion.pack(side="top")
127 loadCondicion = tkinter.Button(frameCondicion, text = "Load condition
128     ", command=fileselectD ,font=("times new roman", 14))
129 loadCondicion.pack(side="left")
```

```

121  labelLoadCondicion = tkinter.Label(frameCondicion, text = "Choose File
122      .txt", font=("Times New Roman", 14))
123  labelLoadCondicion.pack(side="left")
124
124 #-----CONDICION INICIAL USUARIO
125
125 tittleCondicionUsuario = tkinter.Label(frameMainInformacion, text = "Initial condition by user" ,font=("times new roman", 18))
126 tittleCondicionUsuario.grid(row = 0, column = 0, columnspan=2)
127
128 tamanoEspacio = tkinter.Label(frameMainInformacion, text = "Space
129     size: " , font=("times new roman", 14))
129 tamanoEspacio.grid(row=1, column=0)
130 entradaEspacio = tkinter.Entry(frameMainInformacion, font=("times new
131     roman", 14), textvariable = tamEspacio)
131 entradaEspacio.grid(row=1, column=1)
132
133 numeroIteracion = tkinter.Label(frameMainInformacion, text = "Number
134     of iterations: " ,font=("times new roman", 14))
134 numeroIteracion.grid(row=2, column=0)
135 entradaIteracion = tkinter.Entry(frameMainInformacion, font=("times
136     new roman", 14), textvariable = numIteraciones)
136 entradaIteracion.grid(row=2, column=1)
137
138 reglaUsar = tkinter.Label(frameMainInformacion, text = "Rule to use:
139     " ,font=("times new roman", 14))
139 reglaUsar.grid(row=3, column=0)
140 entradaRegla = tkinter.Entry(frameMainInformacion, font=("times new
141     roman", 14), textvariable = reglaValor)
141 entradaRegla.grid(row=3, column=1)
142 binary = tkinter.Checkbutton(frameMainInformacion, text="Is binary?", 
143     variable=esBinario, font=("times new roman", 13))
143 binary.grid(row=4, column=1)
144
145
146 labelCondicion = tkinter.Label(frameMainInformacion, text = "Type of
147     inicial condition: " , font=("times new roman", 14))
147 labelCondicion.grid(row=5, column = 0)
148 checkAleatoria = tkinter.Checkbutton(frameMainInformacion, text="Random",
149     variable=esAleatorio, font=("times new roman", 14))
149 checkAleatoria.grid(row=5, column=1)
150 tipoCondicion = tkinter.Checkbutton(frameMainInformacion, text = "Specific",
151     variable=esEspecifico, command=chageLabel, font=("times
152         new roman", 14))
151 tipoCondicion.grid(row=5, column = 3)
152
153 labelEspecifico = tkinter.Label(frameMainInformacion, text = "Indices
154     to fill in with 1's or \n probability of 1, use % at the end: " ,
154     font=("times new roman", 14))
154 labelEspecifico.grid(row=6, column = 0)

```

```

155 las = tkinter.Entry(frameMainInformacion, font=("times new roman", 14)
156     , textvariable = posicionesUnos)
157 las.grid(row=6, column = 1)
158 labelEspecifico.grid_remove()
159 las.grid_remove()
160 #-----BOTON MAESTRO
161 -----
162 buttonStart = tkinter.Button(frameStart, text = "Start", command=
163     iniciar ,font=("times new roman", 14))
164 buttonStart.pack(side="top")
165 framet.grid(row=0, column=0, columnspan=1)
166 frameColor0.grid(row=1, column=0)
167 frameColor1.grid(row=2, column=0)
168 frameCondicion.grid(row=3, column=0)
169 frameMainInformacion.grid(row=4, column=0)
170 frameStart.grid(row=5, column=0)
171
172 tkinter.mainloop()

```

Por ultimo tenemos el código que nos proporciona la parte lógica del programa y que ademas es el encargado de hacer las gráficas, este código es el cerebro de nuestro programa.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from mpl_toolkits.axes_grid1 import make_axes_locatable
4 import matplotlib.colors as colors
5
6 global unoxPaso
7 global probabilidadUnoxPaso
8
9 def GraficaVarianza():#Momento 2 = Sum(x^2*f(x)) - E[x]^2=> Momento 2
10    = 0*f(x) + 1*f(x) - (1*f(x=1))^2 = f(x=1) - f(x=1)^2
11    global probabilidadUnoxPaso
12    plot4 = plt.figure(3)
13    plt.title("Variance or second moment")
14    plt.xlabel("Steps")
15    plt.ylabel("Variance")
16    plt.plot(probabilidadUnoxPaso-np.power(probabilidadUnoxPaso,2),
17              color ="purple")
18
19 def GraficaMedia():#E[x] = Sum(x*f(x))=> E[x] = 0*f(x) + 1*f(x) = f(x
20    = 1) = Probabilidad de X=1, osea Probabilidad de uno
21    global probabilidadUnoxPaso
22    plot3 = plt.figure(2)
23    plt.title("Expected value or average")
24    plt.xlabel("Steps")
25    plt.ylabel("Expected value")
26    plt.plot(probabilidadUnoxPaso , color = "green")

```

```

24
25 def GraficaDensidad():
26     global unosxPaso
27     plot1 = plt.figure(0)
28     plt.title("Numbers of 1's for each step")
29     plt.xlabel("Steps")
30     plt.ylabel("Numbers of 1's")
31     plt.plot(unosxPaso, color ="red")
32
33 def GraficaLog10():
34     global unosxPaso
35     plot2 = plt.figure(1)
36     plt.title("Log10(Numbers of 1's for each step)")
37     plt.xlabel("Steps")
38     plt.ylabel("Log10(Numbers of 1's)")
39     plt.plot(np.log10(unosxPaso), color ="blue")
40
41 #Ej: Regla 30 = 00011110 base 2
42     #Si vemos la combinacion de las 3 celulas nos permite formar
        combinaciones y cada combinacion tiene un respectivo valor de
        celula de salida
43     #En este caso 111 base 2 o 7 base 10 la salida es 0, por lo que
        tenemos
44     #Output: [0 0 0 1 1 1 1 0]
45     #Indice: [7 6 5 4 3 2 1 0] --Valor decimal de la combinacion que
        genera la salida
46     #Es decir si tenemos el caso 7. Tomamos 7 (numero max obtenido de
        las combinaciones) le restamos nuestro caso para obtener 0 (
        indice del arreglo de salida)
47     #Otro caso: Si tenemos el caso 3. 7 - 3 = 4 (indice en el arreglo
        de salida), salida 1
48     #Return: Arreglo con indices para la regla de las combinaciones
        de las celulas
49 def IndiceRegla(tresCelulas):
50     izq, centro, der = tresCelulas
51     index = 7 - (4*izq + 2*centro + der) #Representacion binaria de
        las 3 celulas, Se resta 7 para coincidir con la representacion
        binaria de la regla
52     return int(index)
53
54 def GenerarAutomataCelular(arreglo,nPasos, reglaNumero):
55     global unosxPaso
56     global probabilidadUnoxPaso
57     reglaString = np.binary_repr(reglaNumero, width=8) #Representamos
        la regla ahora en base 2 como una cadena (8 bits) es un
        numero
58     regla = np.array([int(bit) for bit in reglaString])
59     arregloAutomataCelular = np.zeros((nPasos,len(arreglo)))
60     unosxPaso = np.empty(nPasos, dtype = int)
61     probabilidadUnoxPaso = np.empty(nPasos, dtype = float)

```

```

62     arregloAutomataCelular[0,:] = arreglo #Fila 1
63     unosxPaso[0] = np.count_nonzero(arregloAutomataCelular[0,:] == 1)
64     probabilidadUnoxPaso[0] = unosxPaso[0] / len(arreglo)
65     for pasos in range (1,nPasos):
66         combinaciones3Celulas = np.stack(
67             [
68                 np.roll(arregloAutomataCelular[pasos - 1, :], 1), #  
Movemos a la derecha nuestro arreglo
69                 arregloAutomataCelular[pasos - 1, :], # t actual
70                 np.roll(arregloAutomataCelular[pasos - 1, :], -1), #  
Movemos a la izquierda nuestro arreglo
71             ]
72         )#Obtenemos nuestras combinaciones posibles de 3 celulas
73         arregloAutomataCelular[pasos,:] = regla[np.apply_along_axis(
74             IndiceRegla, 0, combinaciones3Celulas)]#Toma las 3  
columnas, para llamar la funcion indiceRegla
75         #despues empareja cada valor de nuestros indices con el valor  
en la regla 30, guardamos en el arreglo final
76         unosxPaso[pasos] = np.count_nonzero(arregloAutomataCelular[
77             pasos,:]== 1)
78         probabilidadUnoxPaso[pasos] = unosxPaso[pasos] / len(arreglo)
79     return arregloAutomataCelular
80
81 def IniciarArchivo(Color0, Color1, Pasos, Regla, CelulasIniciales):
82     global resultadoAutomata
83     resultadoAutomata = GenerarAutomataCelular(CelulasIniciales, Pasos
84         ,Regla)
85     GraficaDensidad()
86     GraficaLog10()
87     GraficaMedia()
88     GraficaVarianza()
89     cmap = colors.ListedColormap([Color0, Color1]) #0,1
90     fig, ax = plt.subplots(figsize=(16, 9))
91     im = ax.matshow(resultadoAutomata, cmap=cmap)
92     ax.axis(False)
93     divider = make_axes_locatable(ax)
94     cax = divider.append_axes("right", size="3%", pad=0.1)
95     plt.colorbar(im, cax=cax)
96     plt.show()
97
98 def IniciarRandom(Color0, Color1, Pasos, Regla, TamanioAutomata):
99     global resultadoAutomata
100    CelulasIniciales = np.random.randint(2, size=TamanioAutomata)
101    resultadoAutomata = GenerarAutomataCelular(CelulasIniciales, Pasos
102        ,Regla)
103    GraficaDensidad()
104    GraficaLog10()
105    GraficaMedia()
106    GraficaVarianza()
107    cmap = colors.ListedColormap([Color0, Color1]) #0,1

```

```

104     fig, ax = plt.subplots(figsize=(16, 9))
105     im = ax.matshow(resultadoAutomata, cmap=cmap)
106     ax.axis(False)
107     divider = make_axes_locatable(ax)
108     cax = divider.append_axes("right", size="3%", pad=0.1)
109     plt.colorbar(im, cax=cax)
110     plt.show()
111
112 def IniciarEspecifico(Color0, Color1, Pasos, Regla, TamanioAutomata,
113   LugaresUnos):
114     global resultadoAutomata
115     CelulasIniciales = np.zeros((TamanioAutomata,), dtype=int)
116     for i in LugaresUnos:
117       CelulasIniciales[i] = 1
118     resultadoAutomata = GenerarAutomataCelular(CelulasIniciales, Pasos
119       ,Regla)
120     GraficaDensidad()
121     GraficaLog10()
122     GraficaMedia()
123     GraficaVarianza()
124     cmap = colors.ListedColormap([Color0, Color1]) #0,1
125     fig, ax = plt.subplots(figsize=(16, 9))
126     im = ax.matshow(resultadoAutomata, cmap=cmap)
127     ax.axis(False)
128     divider = make_axes_locatable(ax)
129     cax = divider.append_axes("right", size="3%", pad=0.1)
130     plt.colorbar(im, cax=cax)
131     plt.show()
132
133 def IniciarProbabilidad(Color0, Color1, Pasos, Regla, TamanioAutomata
134   , LugaresUnos):
135     global resultadoAutomata
136     CelulasIniciales = np.random.choice([0,1], TamanioAutomata, p
137       =[1-LugaresUnos, LugaresUnos])
138     resultadoAutomata = GenerarAutomataCelular(CelulasIniciales, Pasos
139       ,Regla)
140     GraficaDensidad()
141     GraficaLog10()
142     GraficaMedia()
143     GraficaVarianza()
144     cmap = colors.ListedColormap([Color0, Color1]) #0,1
145     fig, ax = plt.subplots(figsize=(16, 9))
146     im = ax.matshow(resultadoAutomata, cmap=cmap)
147     ax.axis(False)
148     divider = make_axes_locatable(ax)
149     cax = divider.append_axes("right", size="3%", pad=0.1)
150     plt.colorbar(im, cax=cax)
151     plt.show()

```

7. Conclusiones

Esta programa es bastante interesante, ya que es increíble ver al cambiar los valores de las células de nuestra iteración cero cambia por mucho el comportamiento de nuestro ECA que se podria pensar que no es tan significativo pero eso es un pensamiento errado ya que vemos que cambia demasiado, podríamos llegar a decir que una célula de mas o de menos provoca un caos en nuestro ECA y por tanto un ECA diferente cada vez que cambiemos nuestra iteración inicial. Finalmente solo me queda comentar que al haber ya probado diferentes reglas con la misma iteración inicial el resultado es muy diferente con cada regla.

Referencias

- Weisstein, E., 2021. Elementary Cellular Automaton - from Wolfram MathWorld. [online] Mathworld.wolfram.com. Available at: <https://mathworld.wolfram.com/ElementaryCellularAutomaton.html> [Accessed 15 March 2021].
- Lees, E., 2020. Elementary Cellular Automata . Matplotlibblog. [online] Matplotlib.org. Available at: <https://matplotlib.org/matplotlibblog/posts/elementary-cellular-automata/> [Accessed 15 March 2021].
- Caparrini, F., 2016. Autómatas Celulares - Fernando Sancho Caparrini. [online] Cs.us.es. Available at: <http://www.cs.us.es/~fsancho/?e=66> [Accessed 15 March 2021].