ECA Rules - Cellular automaton

Gael Hernández Solís

December 17, 2023

Contents

1 Introducción

Los autómatas celulares elementales son una clase fascinante de sistemas dinámicos discretos que se utilizan en diversos campos de la ciencia y la computación, incluyendo la física, la biología, la informática y la teoría de la complejidad. Fueron introducidos por primera vez por el matemático británico John von Neumann y el físico estonio-estadounidense John Ulam en la década de 1940 como una forma de simular procesos de auto-replicación en sistemas artificiales. Sin embargo, fue el trabajo del matemático británico Stephen Wolfram en la década de 1980 lo que catapultó a los autómatas celulares elementales a la prominencia en la investigación científica y la exploración de la complejidad emergente.

Un autómata celular elemental se puede visualizar como una roja unidimensional de celdas, donde cada celda puede estar en uno de dos estados posibles, generalmente etiquetados como 0 y 1. El autómata evoluciona en pasos discretos a lo largo de una línea, donde el estado de cada celda en un paso de tiempo dado depende de su estado y el estado de sus celdas vecinas en el paso de tiempo anterior, siguiendo una regla de transición específica.

Una característica clave de los autómatas celulares elementales es que sus reglas de transición son extremadamente simples pero generan una rica diversidad de comportamientos y patrones a lo largo del tiempo. En total, existen 256 reglas de transición posibles para autómatas celulares elementales, numeradas del 0 al 255. Cada regla se identifica mediante un número binario de 8 bits, donde cada bit corresponde a una configuración específica de tres celdas vecinas y el resultado de la regla para esa configuración.

Estas reglas se dividen en varias categorías en función de sus propiedades y comportamientos emergentes. Algunas de las categorías más notables incluyen:

Reglas Clase 1 : Estas reglas tienden a producir patrones uniformes y homogéneos en el tiempo, con poca o ninguna complejidad emergente. Un ejemplo es la regla 0, donde todas las celdas evolucionan hacia el mismo estado y permanecen así en el tiempo.

Reglas Clase 2 : En esta categoría, las reglas generan patrones estables y periódicos, como alternancia entre 0 y 1 o patrones oscilantes más complejos. Un ejemplo es la regla 18, que produce un patrón de periódico de dos períodos.

Reglas Clase 3 : Estas reglas son conocidas por generar complejidad emergente y patrones caóticos, a menudo con estructuras de ciclo atractor. El ciclo del atractor se refiere a un patrón que se repite periódicamente pero de manera no trivial. La regla 30 es un ejemplo icónico de una regla de Clase 3, que exhibe un comportamiento caótico y la generación de ciclo atractor.

Reglas Clase 4 : Esta categoría es particularmente interesante porque las reglas generan comportamientos altamente complejos y ricos en detalles. La regla 110 es uno de los ejemplos más estudiados y famosos de una regla de Clase 4, capaz de simular la computación universal y demostrar la capacidad de autómatas celulares para realizar cálculos.

Los campos de atracción son patrones recurrentes en la evolución de un autómata celular. Pueden ser simples, como una secuencia periódica de estados, o más complejas, como estructuras que se repiten de manera irregular pero determinista a lo largo del tiempo. Los campos de atracción son esenciales para comprender la dinámica a largo plazo de los autómatas celulares elementales y se utilizan para caracterizar su comportamiento y complejidad. A continuación, se proporciona más información sobre los atractores cíclicos:

Tipos de Ciclos : Los atractores cíclicos pueden tomar varias formas y tamaños. Los ciclos más simples son los ciclos fijos, que consisten en un único estado o patrón que se repite una y otra vez. Los ciclos periódicos son más complejos y se componen de una secuencia de estados que se repite en un patrón regular. También pueden existir ciclos más complejos y caóticos, que tienen estructuras más irregulares y no se repiten en intervalos fijos.

Importancia en la Dinámica : Los atractores cíclicos son importantes porque ayudan a describir y clasificar el comportamiento de un sistema dinámico. Pueden indicar la presencia de ciertas regularidades o patrones estables en medio de la complejidad aparente de un sistema. Los ciclos a menudo representan estados en los que el sistema puede "quedarse atrapado" durante períodos prolongados antes de evolucionar hacia otro estado o ciclo.

Detección y Análisis : En autómatas celulares y otros sistemas similares, detectar y analizar atractores cíclicos puede ser una tarea desafiante. Esto se debe a que los ciclos pueden tener longitudes diferentes y no siempre son fáciles de identificar a simple vista. Los investigadores utilizan técnicas computacionales, como la simulación a largo plazo y el análisis de patrones, para descubrir y caracterizar atractores cíclicos.

Relevancia en la Computación y la Teoría de la Complejidad : En el contexto de la teoría de la

computación y la teoría de la complejidad, los atractores cíclicos pueden tener implicaciones importantes. Por ejemplo, en autómatas celulares como el "Juego de la vida" de Conway, los ciclos pueden usarse para construir circuitos lógicos y, en última instancia, demostrar la capacidad de estos sistemas para realizar cálculos universales.

Aplicaciones en Ciencia y Tecnología : Además de su importancia teórica, los atractores cíclicos tienen aplicaciones prácticas en una variedad de campos. Por ejemplo, se utilizan en modelos de dinámica de poblaciones en ecología, en el análisis de secuencias genéticas y en la síntesis de patrones en la computación gráfica y la generación de arte generativo.

Evolución de los Ciclos : En muchos sistemas dinámicos, los atractores cíclicos pueden interactuar y evolucionar con el tiempo. Pueden fusionarse para formar ciclos más largos o dividirse en ciclos más cortos, lo que da como resultado una dinámica rica y compleja.

2 Desarrollo

A continuación, se adjuntarán capturas del proceso y se explicará a detalle cómo se elaboró el programa que permite simular todas las reglas de los autómatas celulares elementales y los campos de atracción para realizar una clasificación de ellos.

Cabe mencionar que este simulador se realizó con Processing 4.3, el cuál es un entorno de programación y desarrollo creativo que se utiliza para crear aplicaciones y obras de arte interactivas, principalmente en el ámbito de la programación visual y generativa. Processing se utiliza en la creación de gráficos, animaciones, visualizaciones de datos y proyectos interactivos. El lenguaje utilizado fue Java 8.

2.1 Selección de Reglas

Primero mostraré la vista principal del simulador, donde se llevan a cabo todas las configuraciones para mostrar todas las características que son posibles en nuestro programa.

Figure 1: Vista general del simulador

Como podemos ver en la vista principal del simulador, tenemos los siguientes botones y secciones:

Botones

- Generar: Sirve para generar la cuadrícula donde se simulará el autómata celular.
- Automático: Sirve para que se muestre en pantalla las generaciones con un timer de 0.1 milisegundos.

- Llenar Aleatoriamente: Sirve para llenar de manera aleatoria nuestra primera generación de celdas vivas y muertas, es posible cambiar la densidad con la que se muestran los unos oprimiendo los botones de porcentajes de densidad.
- Guardar Configuración: Sirve para guardar en un archivo .txt la configuración de unos y ceros de la primera generación en caso de que nos interese.
- Aplicar Regla: Se oprime cada que queramos ver la siguiente generación, es decir, de manera manual podemos mostrar en pantalla las generaciones.
- Cargar Patrón: Obtiene la configuración de unos y ceros de un archivo .txt hecho por nosotros para cargarla en el programa y ponerla como la primera generación.
- Densidades: Son los cuatro porcentajes solicitados que nos indican con qué densidad de unos queremos tener en nuestra primera generación, claro generados de manera aleatoria.
- Guardar Última Fila: Guarda en un archivo .txt la configuración de la última generación mostrada de acuerdo al tamaño de pantalla indicado.

Secciones

- Colores: Nos permite indicar el color que queramos de la celda muerta y viva en nuestra simulación.
- Tamaño de cuadro: Permite indicar el tamaño en pixeles que queremos que tengan nuestras celdas, desde un pixel hasta un tamaño de 10 pixeles.
- Tamaño de ventana: Permite indicar el tamaño de ventana que queremos, desde 10x10 cuadros hasta un tamaño de pantalla completa, es decir, 1920x1080 cuadros.
- Reglas: Una lista de reglas desde la 0 hasta la 255 que permite elegir por supuesto qué regla queremos que se lleve a cabo.

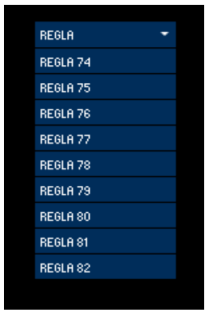

Figure 2: Vista intermedia de las reglas de la 0 a la 255

2.2 Evaluación de Espacios

Para esta sección, a pesar de que si pude inicializar las 10000x10000 celdas solicitadas, no las incluí debido a que el programa tardaba mucho en responder o simplemente no respondía aún con scroll y zoom incluido, por lo que decidí quitarlo y poner como tamaño más grande el de la pantalla que es de 1920x1080 pixeles de tamaño uno claro, aún con este tamaño, el programa es algo lento.

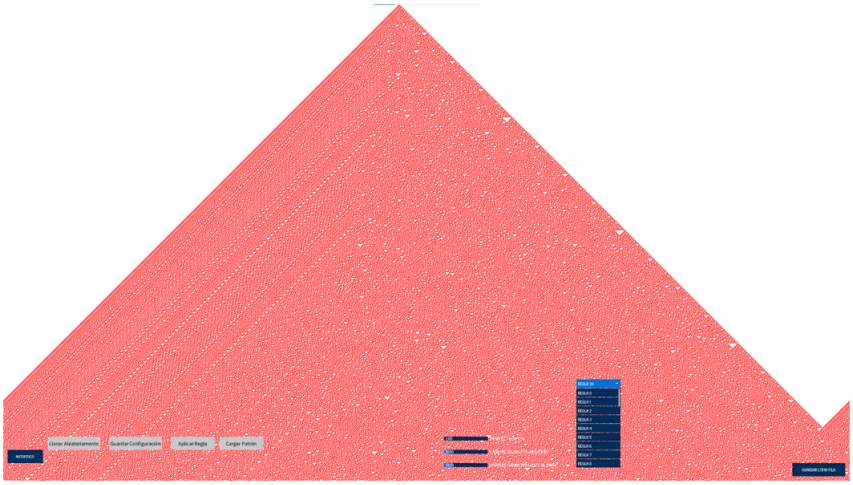

Figure 3: Pantalla completa del simulador (regla 30)

2.3 Cambio de Colores de Estados

La característica de cambio de colores se empleó con un selector de colores para cada celda, tanto la viva como la muerta, utilizando cuatro controles deslizantes (scrolls) que van desde 0 hasta 255 para elegir un color RGBA (Rojo, Verde, Azul, Alfa). Los valores de los deslizadores están en el rango de 0 a 255 como ya lo mencioné, donde 0 significa ausencia de color y 255 es la intensidad máxima. El valor alfa controla la transparencia, donde 0 es completamente transparente y 255 es completamente opaco. A continuación se muestra una configuración de colores, sin embargo, podemos elegir el que queramos.

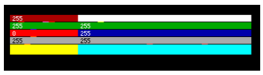

Figure 4: Configuración de colores para el ejemplo

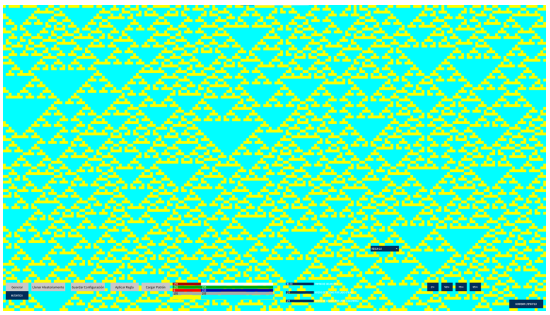

Figure 5: Ejemplo con los colores seleccionados (regla 22 con tamaño de 10 pixeles)

2.4 Inicialización de Espacios

Para esta característica, en nuestro simulador hay tres maneras diferentes de inicializar nuestra primera generación, la primera es de manera manual, es decir, editamos directamente nuestro espacio de evoluciones, con clic izquiero la celda viva (1), y con clic derecho la celda muerta (0). Mostraré cómo se vería seleccionar sólo cuatro celdas y las demás muertas en un tamaño muy grande para que sea visible.

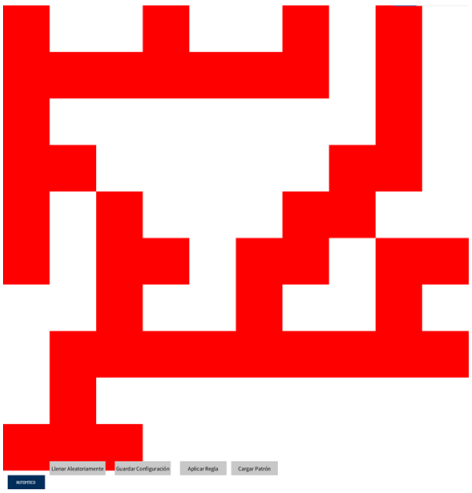

Figure 6: Relleno manual de 4 celdas vivas en un tamaño de ventana de 10x10 y tamaño de 100 pixeles por celda

La otra forma es mediante un archivo de nombre "carga.txt", donde tenemos que poner la configuración de ceros y unos que queremos tener inicialmente, es importante decir que la cantidad de ceros y unos debe ser igual que el tamaño de columnas de nuestra generación, en otro caso marcará error. A continuación se muestra el archivo "carga.txt" con una configuración cualquiera y cómo se ve cargado en el simulador.

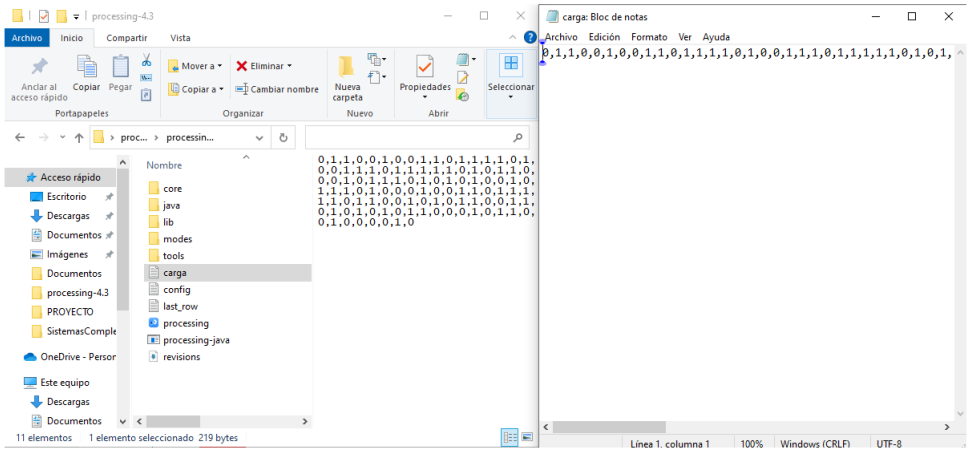

Figure 7: Vista resumida y general de la configuración de unos y ceros del archivo "carga.txt" de 110 cuadros de tamaño

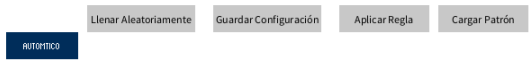

Figure 8: Carga del archivo con el tamaño mencionado en el simulador

Finalmente, la última forma es mediante el botón "Generar Aleatoriamente", primero debemos seleccionar la densidad de unos que queremos, ya se explicó qué botones son, y después oprimiremos el botón. A continuación se muestra un ejemplo de cómo se vería una densidad de unos del 2 porciento de manera aleatoria.

Figure 9: Densidad de unos del 2 porciento (Uno es verde, cero es rosa)

2.5 Salvado y Levantado de Archivos

La parte de levantado de archivos ya se demostró en el anterior punto. Para la parte del salvado, sólo guardo la última fila o generación que haya sido creada de acuerdo al tamaño de filas de nuestra ventana, el principal uso de esta opción es para poder cargar esa configuración final como una inicial y ver cómo sigue evolucionando nuestro autómata celular. A continuación muestro una configuración final de un autómata y el archivo .txt generado con dicha generación.

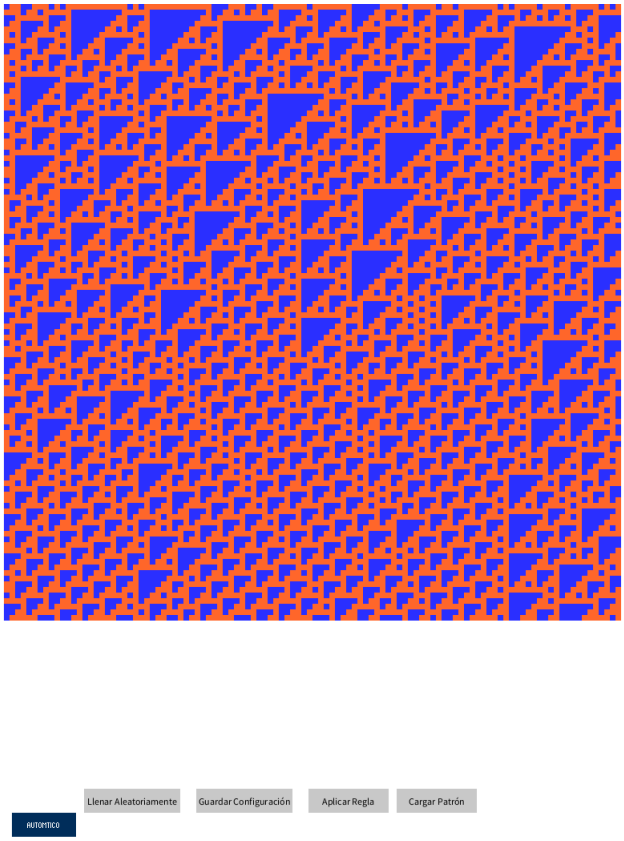

Figure 10: Configuración final después de aplicar la regla 110 en un tamaño de cuadrícula de 110x110

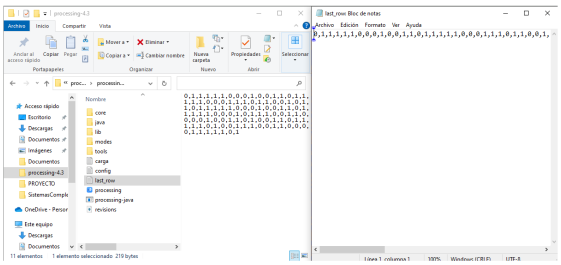

Figure 11: Vista previa y general del archivo "lastrow.txt" donde se guardó dicha configuración

2.6 Graficación de Datos

Para esta característica del simulador, se solicitó graficar el número de unos de cada generación (densidad), también el logaritmo base 10 de la función densidad anteriormente mencionada, la media de unos de cada generación, la varianza de unos de cada generación y finalmente la entroía de Shannon la cual mide la incertidumbre de una fuente de información, también se puede considerar como la cantidad de información promedio que contienen los símbolos usados. Los símbolos con menor probabilidad son los que aportan mayor información. En nuestro caso, los símbolos serán las subcadenas que conforman nuestras reglas de autómatas celulares. Cabe mencionar que las gráficas y gran parte de los cálculos se hicieron con apoyo de la herramienta Matlab, de nuestro simulador se generaron todos los archivos .txt con la información necesaria para poder hacer las gráficas.
A continuación se explicará cómo se realizó cada cálculo y se mostrarán las gráficas para las reglas más especiales.

Densidad y su Logaritmo base 10

Para este cálculo, sólo se requirió calcular el número de unos y de ceros de cada generación, una función sencilla y reiniciar dichos contadores cada que avance de generación. Una vez calculados para una, dividimos el número de unos entre el número de ceros y lo agregamos a un arreglo el cuál se irá imprimiendo en un archivo llamado DENSIDAD.txt. Finalmente, en el archivo tenemos dos columnas, una es la generación y la otra es su densidad de unos, lo mandamos a abrir en MATLAB y guardamos dichos valores en dos arreglos los cuales graficaremos. Para el cálculo de dichos valores en logaritmo base 10, simplemente se le aplicó al arreglo donde guardamos las densidades el logaritmo y lo guardamos en otro arreglo para graficarlo todo en una misma gráfica, a continuación se muestra el archivo generado y la gráfica en MATLAB.

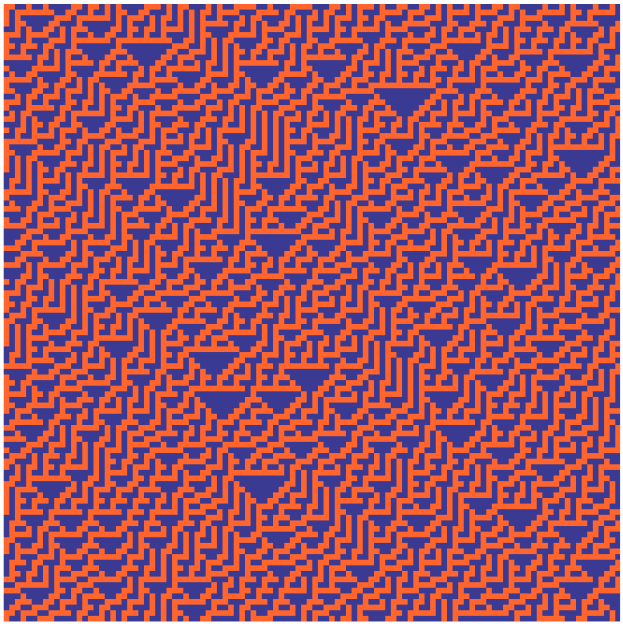

Figure 12: Regla 30 con 50 porciento de densidad de unos inicial (unos-naranja, ceros-morado)

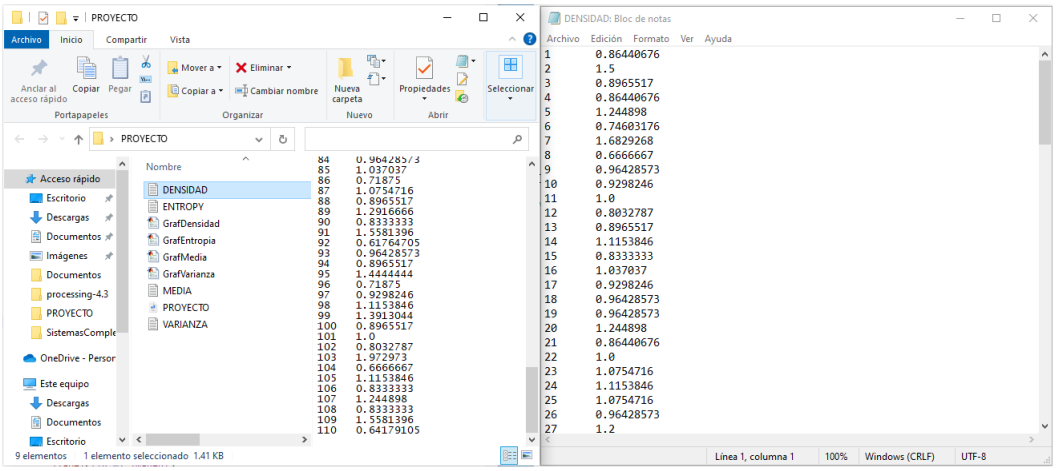

Figure 13: Archivo DENSIDAD.txt con las densidades de la regla 30 en un espacio de 110x110

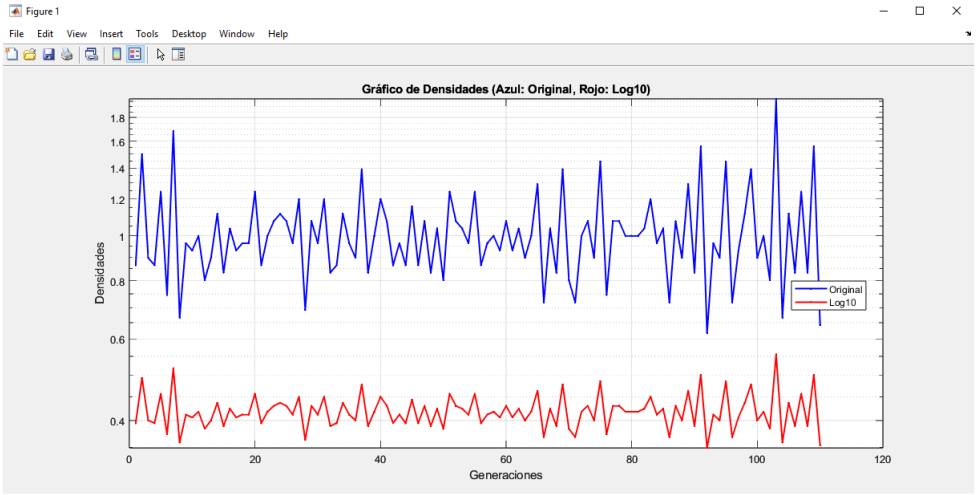

Figure 14: Gráfica de densidad de unos para la regla 30 con 110 generaciones y su logaritmo base 10 aplicado

Media

Para este cálculo, se utilizaron los mismos contadores de unos y ceros, la única diferencia es que para calcular la media de unos, se dividió el número de unos entre el número total de celdas que tiene la generación, en el ejemplo que estamos revisando, el tamaño es de 110. El proceso en MATLAB fue el mismo, se guardó la información extraída del archivo MEDIA.txt en dos arreglos y se graficaron, a continuación se muestra el archivo generado y la gráfica.

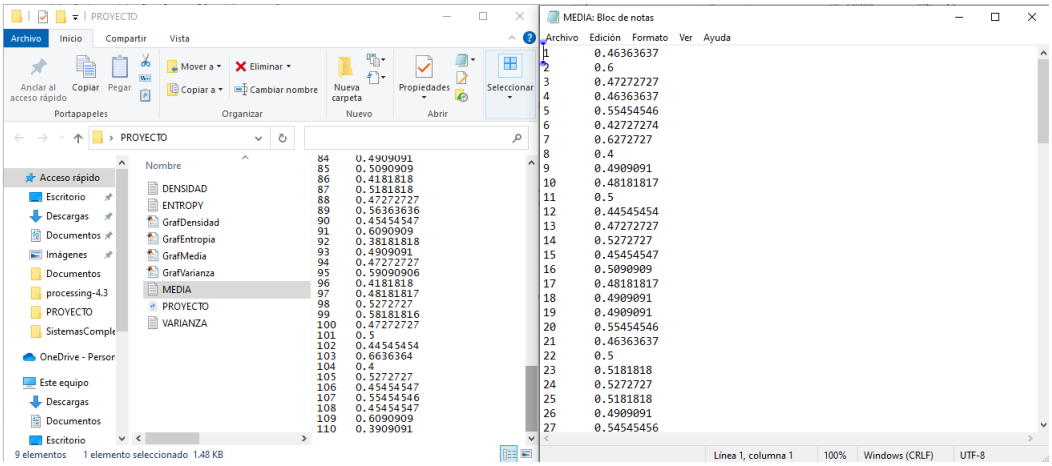

Figure 15: Archivo MEDIA.txt con la media de unos de la regla 30 en un espacio de 110x110

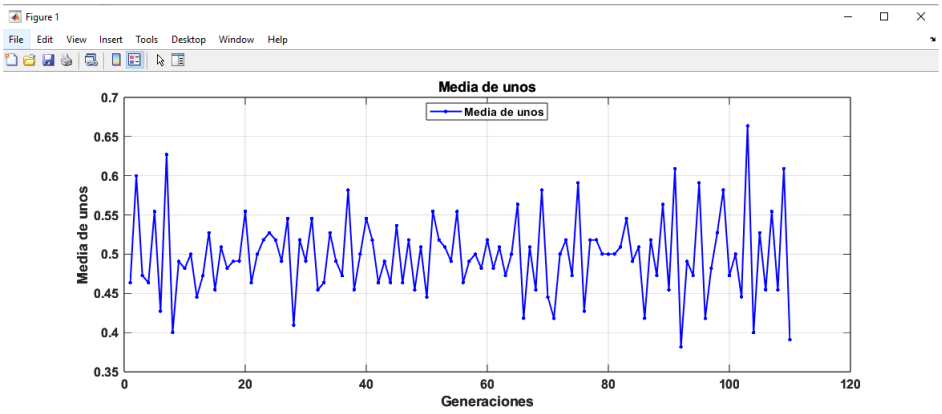

Figure 16: Gráfica de media de unos para la regla 30 con 110 generaciones

Varianza

La varianza es una medida de dispersión que se utiliza para cuantificar cuán dispersos o esparcidos están los valores en un conjunto de datos. Para este caso, se calculará la variación de las proporciones de unos en todas las generaciones. Para este proceso, se mandó a un archivo llamado VARIANZA.trt todas las cadenas de unos y ceros de las generaciones, una vez hecho esto, en MATLAB guardamos dichas cadenas en un arreglo de cadenas y vamos calculando con la función var() la varianza para cada posición de nuestro arreglo. El resultado lo vamos guardando en otra cadena para así graficar la varianza de cada generación, a continuación se muestra el archivo .txt generado y la gráfica respectiva para el ejemplo que estamos viendo.

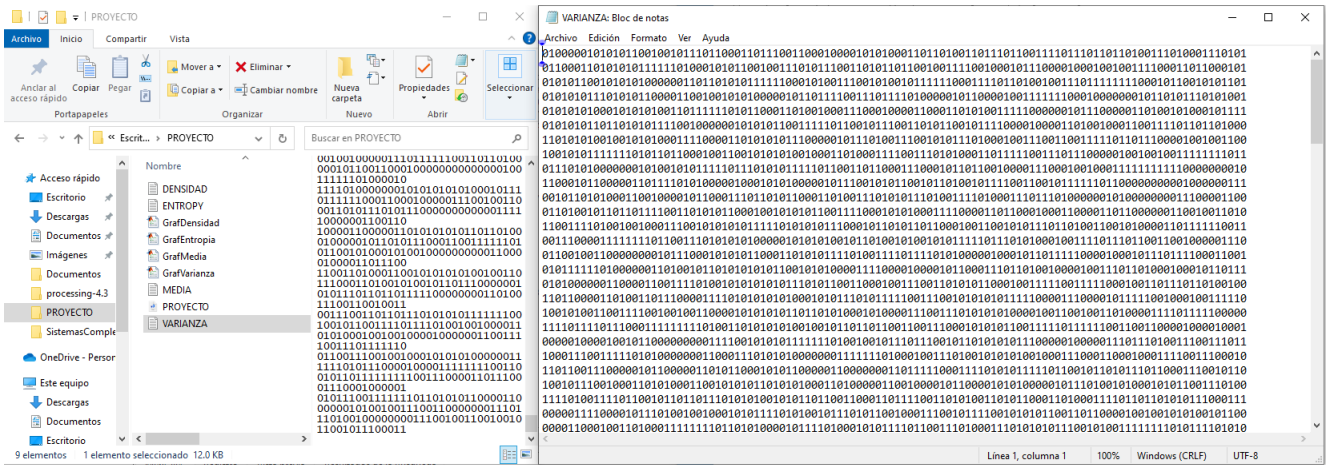

Figure 17: Archivo VARIANZA.txt con las cadenas de unos y ceros de la regla 30 en un espacio de 110x110

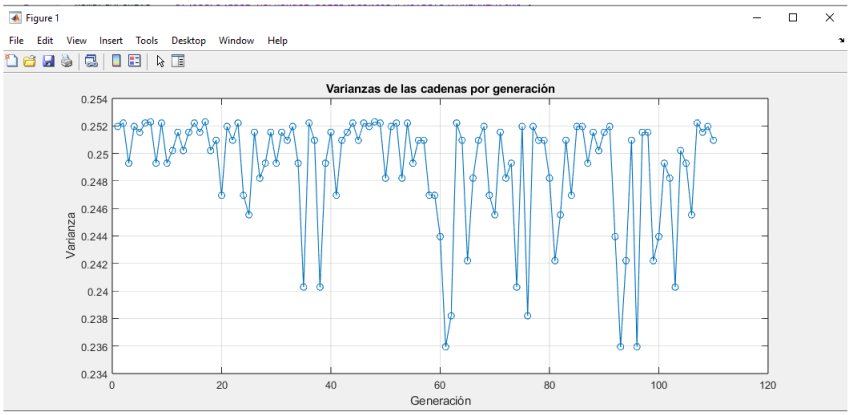

Figure 18: Gráfica de la varianza de unos para la regla 30 con 110 generaciones

Entropía de Shannon

La entropía de Shannon es una medida de la información y la incertidumbre en un sistema o fuente de datos. Se utiliza para cuantificar la cantidad de información o desorden en un conjunto de datos. Cuando se aplica a autómatas celulares elementales, se utiliza para evaluar la complejidad y la evolución de los patrones generados por estos autómatas.

Entonces, se utiliza para medir la diversidad y la complejidad de los patrones que emergen de la evolución de un autómata celular elemental. Para calcular la entropía se siguieron los siguientes pasos:

- Primero, se inició un arreglo que contenía todas las subcadenas que íbamos a contar dentro de nuestra generación, estas son las que conforman las reglas de los autómatas celulares.
- Se usó un ciclo para guardar en otro arreglo el número de subcadenas que había en cada generación usando el primer arreglo que definimos.

- Una vez obtenidos estos datos, usamos otro ciclo para calcular y guardar en otro arreglo las probabilidades pi, es decir, la proporción de estas subcadenas en la generación, para ello, dividimos el número de cada subcadena entre el número de celdas que había en la generación.

- Finalmente, usamos un último ciclo para calcular la entropía de Shannon de cada generación y lo vamos guardando en un archivo llamado ENTROPY.txt el cuál utilizaremos en MATLAB para graficarlo.

La fórmula de la Entropía de Shannon es la siguiente:

$$\text{Entropía} = -\sum(p_i \log_2(p_i))$$

Donde pi es la proporción de subcadenas en cada generación.

Una vez explicado de manera resumida el proceso, se mostrará a continuación el archivo generado con el ejemplo que estamos viendo y la gráfica hecha en MATLAB.

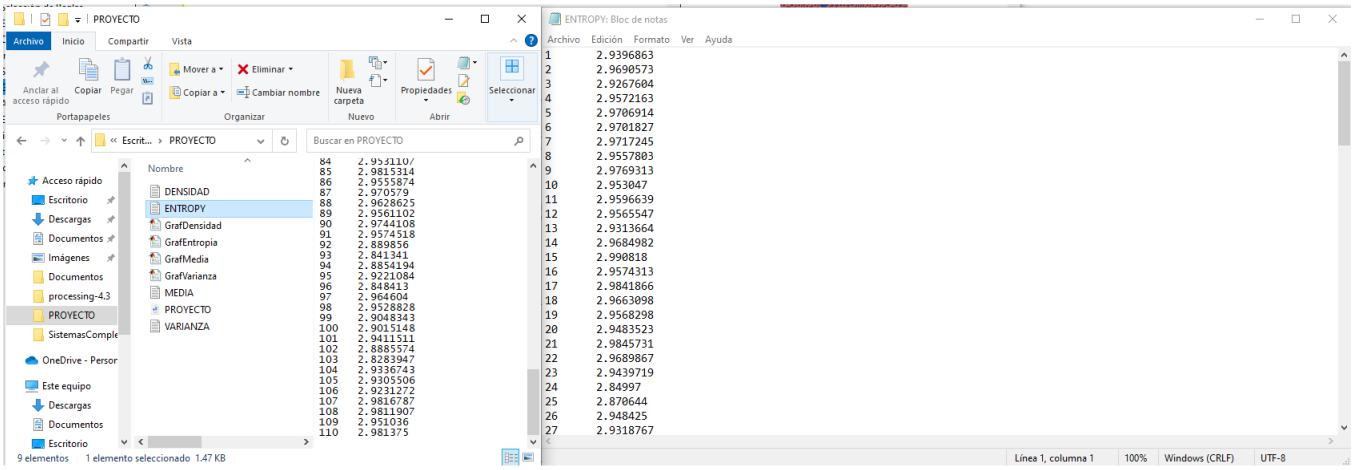

Figure 19: Archivo ENTROPY.txt con el cálculo de la entropía de Shannon para cada generación de la regla 30 en un espacio de 110x110

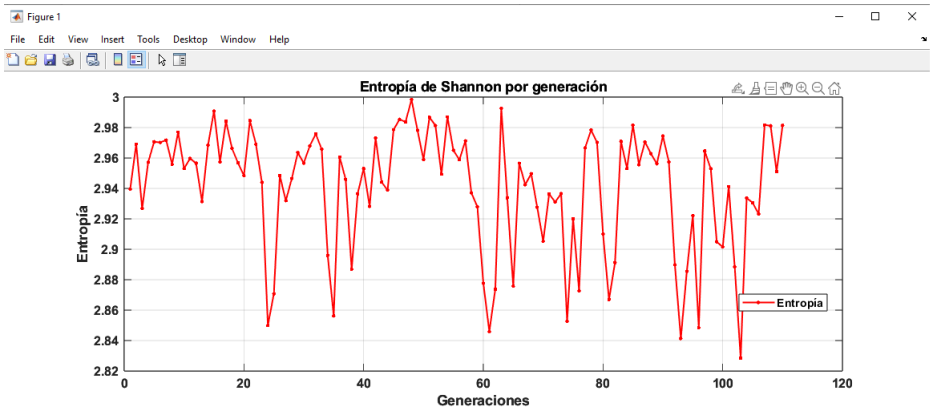

Figure 20: Gráfica de la Entropía de Shannon para cada generación de la regla 30 con 110 generaciones

Con esto que se acaba de ejemplificar, podemos calcular la densidad, la media, varianza y la entropía de Shannon para cualquiera de las reglas de los autómatas celulares elementales, ya con esto, podremos hacer en los siguientes capítulos un análisis más detallado de las reglas para diferentes densidades iniciales.

2.7 Análisis de Atractores

Un ciclo atractor o campo atractor se refiere a un patrón recurrente o conjunto de estados que se repiten en la evolución temporal del autómata celular. Un ciclo atractor es un patrón que se repite de manera constante en la evolución del autómata celular, y atrae a otros ajustes iniciales hacia sí mismo. En otras palabras, actúa como un punto de estabilidad o un patrón recurrente en el espacio de estados del autómata. Estos ciclos atractores pueden ser simples, como un único patrón de celdas que se repite una y otra vez, o más complejos, con múltiples patrones que se alternan. Esto nos ermite comprender mejor cómo evolucionan y se comportan estos sistemas a lo largo del tiempo, y cómo ciertos patrones pueden emerger y persistir en su dinámica, incluso a partir de condiciones iniciales aleatorias o no triviales.

A continuación se muestran los círculos atractores con un tamaño de generación de n=10 para las reglas 30, 110, 22 y 15.

Figure 21: Campo de atracción para la regla 30 con un tamaño n=10

Dado que la regla 30 es bien conocida por generar complejidad y patrones interesantes, se ha observado que no tiene un ciclo atractor simple. La regla 30 tiende a producir patrones complejos y caóticos a partir

de configuraciones iniciales simples.

Se ejecutó la regla 30 con una fila inicial de L=10 celdas y una densidad del 50 porceinto, se observó que con el tiempo se fueron generando patrones que no se repiten de manera regular y no forman ciclos atractores estables. En su lugar, se crean estructuras complejas y caóticas que parecen aleatorias y son difíciles de predecir a largo plazo. Esta es una de las características intrigantes de la regla 30 y otras autómatas celulares similares, ya que demuestran cómo a partir de reglas de evolución simples, pueden surgir patrones complejos y aparentemente aleatorios.

Figure 22: Campo de atracción para la regla 110 con un tamaño n=10

Este ciclo atractor de la regla 110 con L=10 muestra un patrón complejo y recurrente que se repite después de un cierto número de pasos. Las transiciones entre estos estados parecen ser no lineales y difíciles de predecir una simple vista.

No se observa desde este tamaño de simulación pero podría generar estructuras como bordes caóticos en un tamaño más grande, esto debido a la complejidad inherente de la regla 110.

Figure 23: Campo de atracción para la regla 22 con un tamaño n=10

Dado que la regla 22 es una de las reglas más simples, el ciclo atractror que generó es relativamente facil de describir pues parece que incluye patrones que se repiten de manera constante en la evolución del autómata, creando una estructura cíclica.

Entonces podemos decir que el campo de atracción para esta regla tiene un patrón recurrente y periódico que se repite sin improtar el tamaño que se le de al autómata.

Figure 24: Campo de atracción para la regla 15 con un tamaño n=10

Dado que esta regla también es sencilla y no permiten cambios que generen patrones complejos, el campo de atracción para la regla 15 se observa que es predecible y constante, no se esperaría tener patrones dinámicos para un tamaño mayor debido a que el autómata se estabiliza después de cierto periodo en un número finito de pasos, creando un campo de atracción estable en el que todas las celdas tienen el mismo estado.

2.8 Estudio Estadístico

Para esta última sección del reporte, haremos un estudio estadístico usando todas las herramientas que incluye nuestro simulador para determinar si existe alguna diferencia o cambio en alguna regla que usemos en específico aplicando de manera inicial densidades del 2, 50, 75 y 95 porciento de unos sobre ceros. Para este caso, estudiaremos la regla 184 que se dice es relevante en el contexto del tráfico pues modela el comportamiento de los automóviles en una carretera. A continuación se muestran los resultados obtenidos para cada densidad, finalmente se da una conclusión.

- **Densidad del 2 porciento**.

  Primero inicializaremos nuestra primera generación con una densidad del dos porciento, cabe decir que la celda viva para este estudio será de color amarillo, y la celda muerta de color negro.

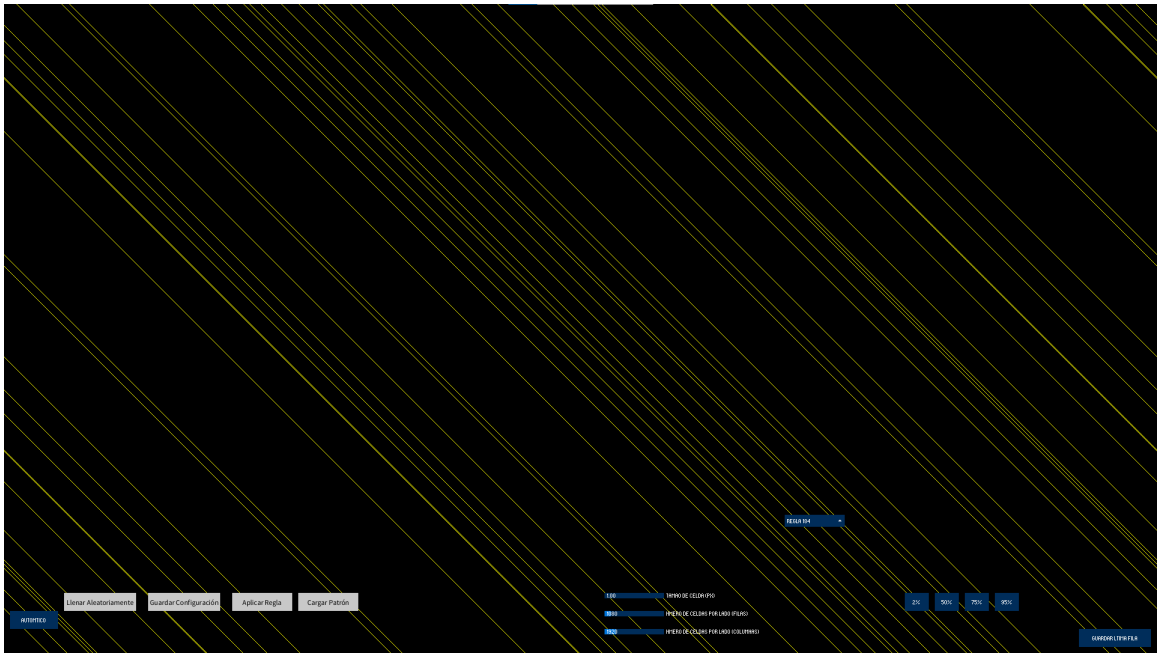

Figure 25: Regla 184 con densidad inicial del dos porciento

Como podemos ver, en términos simples, la regla 184 modela el flujo de vehículos en una carretera de una sola vía con una velocidad máxima de un automóvil por paso de tiempo. Cuando un automóvil se encuentra junto a una celda vacía, avanza una posición hacia la derecha en el siguiente paso de tiempo. Si hay dos o más automóviles adyacentes, se produce una congestión y un automóvil se detiene, lo que se traduce en una celda vacía en el siguiente paso de tiempo.

Con el archivo que nos generó de DENSIDAD.txt, veremos a continuación la gráfica con la densidad de unos que tiene este autómata celular elemental hecho con la regla 184 y el logartimo base 10 aplicado a esta gráfica.

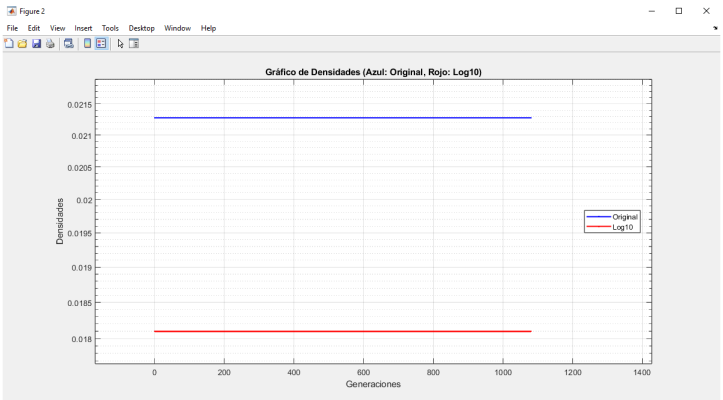

Figure 26: Densidad de unos de la regla 184 y logaritmo base 10 aplicado

Como ya vimos, la regla 184 se utiliza para modelar el tráfico en una carretera, y es una regla que conserva la densidad. Esto significa que la densidad total de vehículos (unos) se mantiene constante a lo largo del tiempo, a menos que ocurran condiciones iniciales inusuales o se introduzcan interacciones externas.

Como se observa en la gráfica, la densidad de unos para este caso es demasiado baja, además de que se mantiene constante debido a que la densidad de celdas vivas es muy bajo, además es necesario mencionar que estamos hablando de 1080 celdas a lo largo, sin embargo, no necesariamente se mantendrá constante siempre. La evolución de la densidad de unos en un autómata celular depende de las siguientes consideraciones:

Condiciones iniciales: La densidad de unos en las primeras generaciones depende de las condiciones iniciales que se establezcan en el autómata celular. Si se comienza con una sola celda viva rodeada de celdas muertas, la densidad de unos será inicialmente baja. Sin embargo, si se comienza con una cuadrícula llena de celdas vivas, la densidad de unos será alta desde el principio.

Reglas de evolución: La regla 184 tiene reglas específicas que determinan cómo las celdas cambian de estado en cada generación. Por ejemplo, si hay una congestión de celdas ocupadas, la densidad de unos podría disminuir, ya que algunos automóviles se convierten en celdas vacías.

Patrones emergentes: A medida que el autómata celular evoluciona, pueden aparecer patrones complejos y estructuras en la cuadrícula. Estos patrones pueden influir en la densidad de unos. Por ejemplo, los patrones que se desplazan pueden aumentar o disminuir la densidad de unos en la medida que interactúan con otras estructuras.

Ahora veremos su gráfica donde muestra la media de unos.

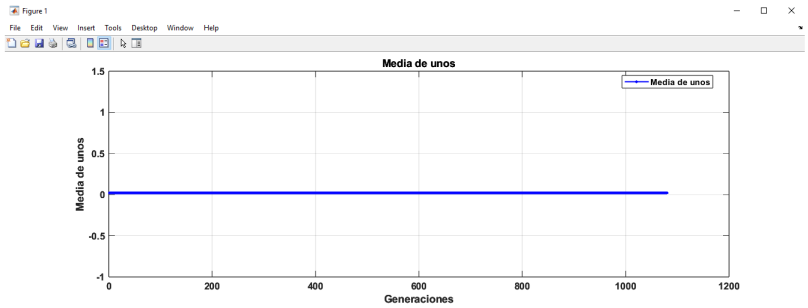

Figure 27: Media de unos de la regla 184 en un espacio de 1920x1080 pixeles

De igual manera, como se ve en la gráfica y en nuestro autómata celular, la media de unos se mantiene constate pues la cantidad de unos con la que iniciamos fue muy poca, del 2 porciento.

Observemos ahora la varianza de las generaciones que conforman nuestro autómata celular.

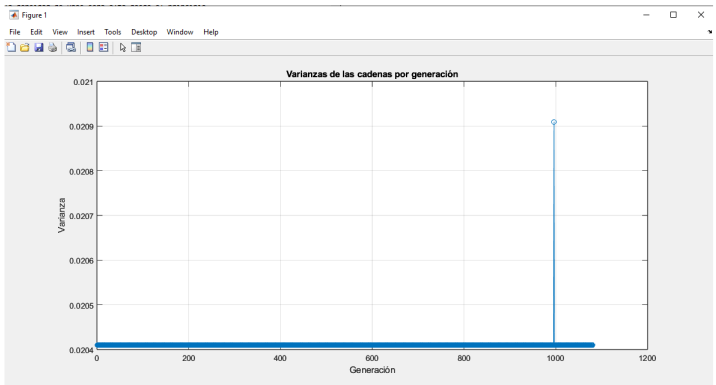

Figure 28: Varianza de la regla 184 en un espacio de 1920x1080 pixeles

Como podemos observar, tampoco hay cambio en la gráfica de nuestra varianza lo cual es algo interesante pues como bien sabemos, la varianza mide cuánto se dispersan los valores de un conjunto de datos alrededor de la media o promedio de esos valores, si lo traducimos al concepto de la regla 184, podemos decir que es una regla determinista, lo que significa que el estado de una celda en una generación posterior está completamente determinado por el estado de las celdas adyacentes en la generación anterior. Dado que las reglas de evolución son consistentes y predecibles, las propiedades estadísticas, tienden a mantenerse estables a menos que ocurran cambios significativos en las condiciones iniciales.

También estamos tratando con una regla que tiene un equilibrio dinámico, es decir, después de un número suficiente de generaciones, los patrones alcanzan un equilibrio. En este estado, los patrones tienden a repetirse o mantenerse relativamente estables con el tiempo. Cuando los patrones son estables, la variación tiende a mantenerse constante o cambiar en una escala limitada, el cual fue nuestro caso.

Finalmente, veamos cómo resulta la entropía de Shannon para este caso en particular.

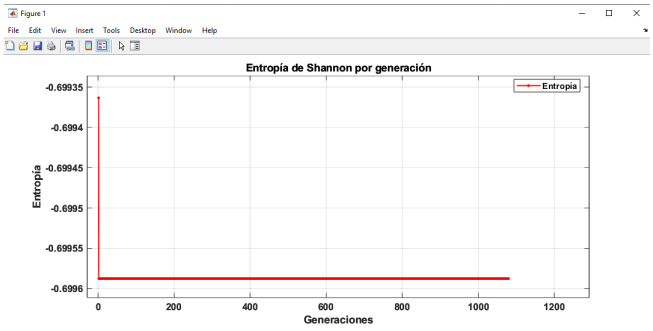

Figure 29: Entropía de Shannon para la regla 184 en un espacio de 1920x1080 pixeles

Como bien sabemos, la entropía de Shannon es una medida de la información y la incertidumbre en un sistema. En el contexto de un autómata celular, como la regla 184, la entropía de Shannon se utiliza para cuantificar la complejidad y la variabilidad de los patrones generados por el autómata a lo largo del tiempo.

Para este caso, vemos que la entropía también se mantiene constante después de avanzar de la primera generación o de las condiciones iniciales, entonces podemos decir que que la cantidad de información y la incertidumbre en ese sistema no están cambiando significativamente. En otras palabras, la entropía constante sugiere que la complejidad o la variabilidad en el sistema se mantiene en un nivel estable y predecible, porque como ya sabemos, es una regla determinística.

- **Densidad del 50 porciento**

  Inicializando nuestra primera generación con una densidad del 50 porciento, se observaría de la siguiente manera nuestro autómata celular.

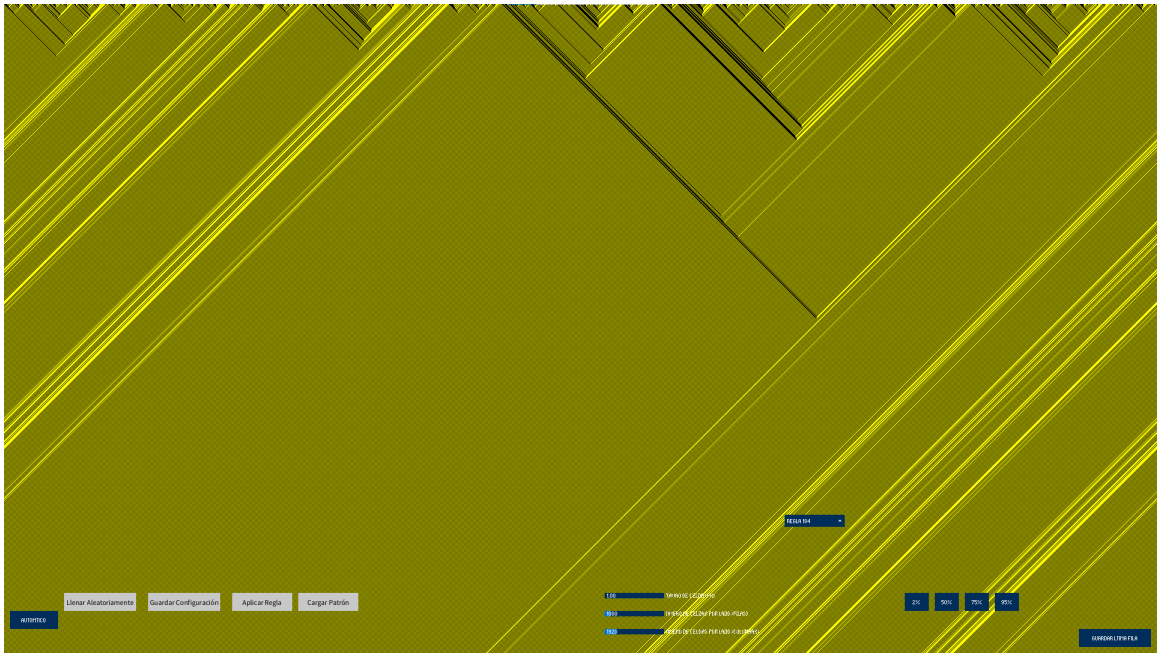

Figure 30: Regla 184 con densidad inicial del 50 porciento

Con una densidad inicial del 50 porciento, podemos ver en el contexto de la regla 184 que la carretera está inicialmente equilibrada en términos de tráfico, con una cantidad significativa de vehículos ocupando la cuadrícula, a medida que el tiempo avanza, vemos como en la primeras generaciones hay mucha congestión y separación de vehículos, pero la densidad relativa de vehículos ocupados se mantiene cerca del 50 porciento, esto debido a que hay muchos carros adyacentes entre sí por lo que provoca congestión, sin embargo, a partir de la mitad del autómata hasta su fin, el tráfico se nivela y se llega al equilibrio dinámica del que tanto hemos hablado.

Con esto, observemos la gráfica de densidad.

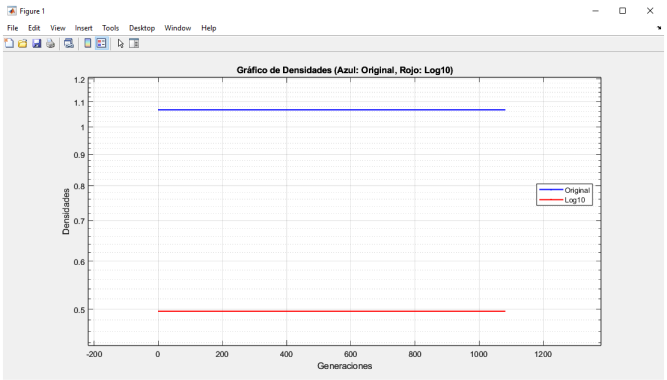

Figure 31: Regla 184 con densidad inicial del 50 porciento

De igual manera, la densidad de unos sobre ceros se mantiene constante pues estamos tratando con una regla deterministica, lo que sí cambió fue que la densidad aumentó hasta un valor 1 aproximadamente, esto debido a que aumentamos la cantidad de unos hasta un 50 porciento.

Ahora veremos la media de unos.

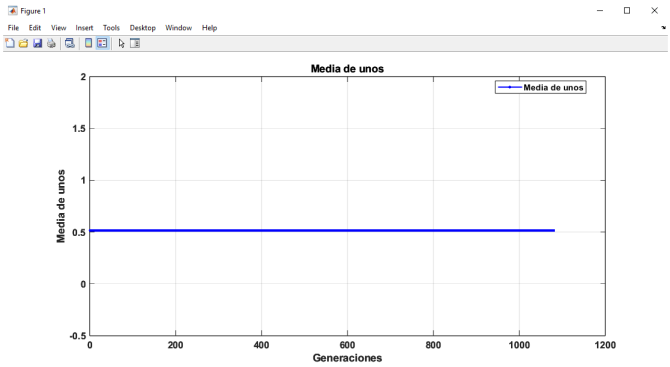

Figure 32: Media de unos de la regla 184 en un espacio de 1920x1080 pixeles

La media de unos se sigue manteniendo constante, sin embargo, como iniciamos con una densidad del 50 porciento y por las razones que ya explicamos, sube la grafica hasta 0.5 en la media y se mantiene constante.

Observemos ahora la varianza de las generaciones que conforman nuestro autómata celular para esta densidad del 50 porciento.

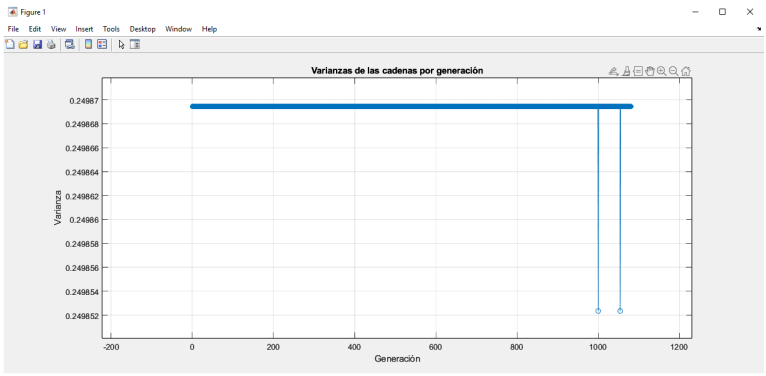

Figure 33: Varianza de la regla 184 en un espacio de 1920x1080 pixeles

Con esta gráfica, a comparación de la primera que analizamos, vemos que aumenta la varianza en cada generación, y este cambio se mantiene constante. Con esto podemos decir que un aumento en la densidad inicial, en nuestro caso, del 50 porciento, da como resultado una mayor dispersión en las densidades iniciales. La mayor dispersión en las densidades iniciales podría contribuir a un aumento en la variación de la densidad. Además, a pesar de que la varianza aumenta debido a una mayor dispersión en las densidades iniciales, la constancia de la varianza a lo largo del tiempo sugiere que, a medida que el autómata evoluciona, los patrones tienden a estabilizarse y mantener un equilibrio dinámico. Esto significa que, aunque la densidad inicial puede variar, el comportamiento del autómata tiende a alcanzar un estado de equilibrio en el que las variaciones en la densidad se compensan entre generaciones, lo que mantiene la varianza constante.

Por último, veamos la entropía de Shannon para este caso.

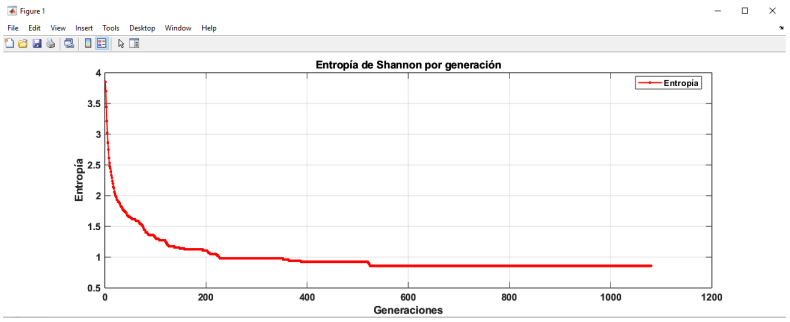

Figure 34: Entropía de Shannon para la regla 184 en un espacio de 1920x1080 pixeles

Aquí vemos algo diferente e interesante, observamos que nuestra gráfica parte de un valor aproximado a 4 y de ahí va disminuyendo hasta estabilizarce o mantenerse constante en un valor menor a uno pero mayor que cero. Con esto, podemos decir que cuando la densidad inicial es del 50 porciento, hay una cantidad significativa de información en la distribución de celdas ocupadas y vacías en la primera generación.

Esto se refleja en una entropía inicial de 4, que es relativamente alta y sugiere una alta incertidumbre en el sistema. Con el tiempo, el autómata celular tiende a establecer patrones y comportamientos recurrentes debido a la regla de evolución determinista. Esto puede reducir la incertidumbre y, como resultado, disminuir la entropía a 3, 2 y 1 a medida que los patrones se vuelven más predecibles y regulares.

Cuando la entropía alcanza niveles más bajos (3, 2, 1), sugiere que los patrones se han estabilizado y que el sistema se ha vuelto más predecible. La entropía constante en estos niveles indica que, aunque los patrones pueden cambiar de una generación a otra, la variabilidad es limitada y los patrones generados son más regulares y predecibles.

- **Densidad del 75 porciento**

  Nuestro autómata celular con una densidad inicial del 75 porciento se vería de la siguiente manera.

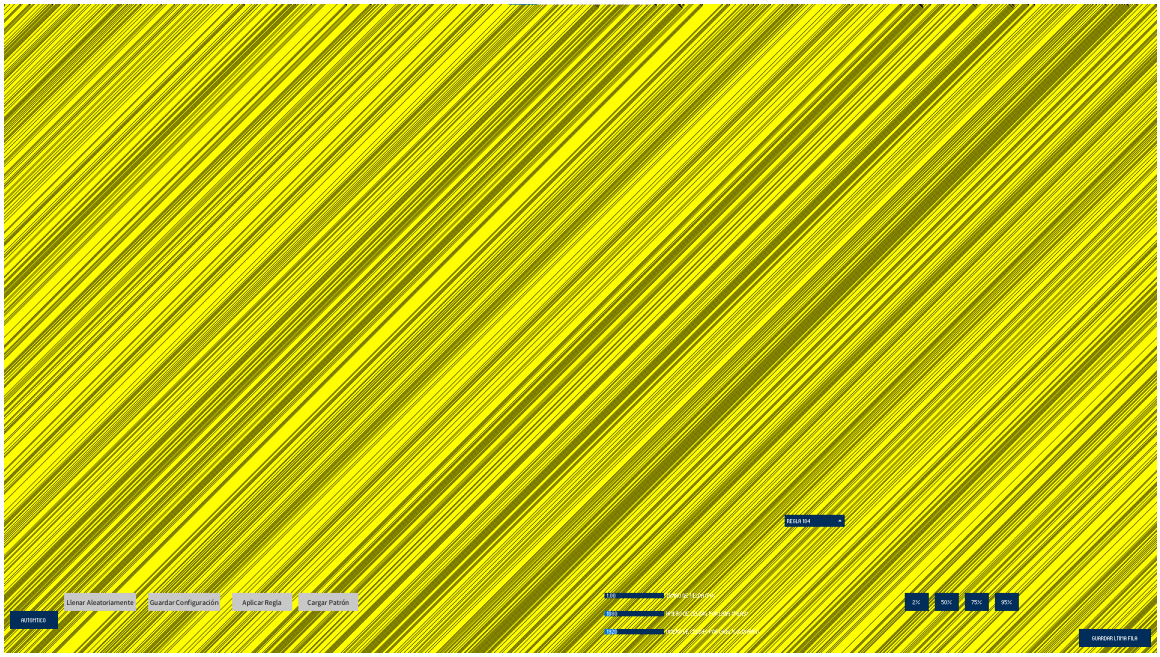

Figure 35: Regla 184 con densidad inicial del 75 porciento

En este caso, comenzamos con una densidad inicial del 75 porciento, lo que significa que el 75 porciento de las celdas están ocupadas con vehículos y el 25 porciento están vacías en la primera generación.

La alta densidad inicial indica una congestión significativa en la carretera desde el principio. A medida que el autómata evoluciona a lo largo de las generaciones, se observa un comportamiento que refleja las condiciones iniciales.

En las primeras generaciones, la congestión inicial dará como resultado un movimiento más lento y una tendencia a la formación de grupos de vehículos.

Podemos ver como en un inicio, la carretera experimenta fluctuaciones en la densidad, sin embargo, con el tiempo el sistema alcanzó un equilibrio dinámico en el que las congestiones y las liberaciones de tráfico se producen de manera cíclica.

Ahora observemos la gráfica de densidad.

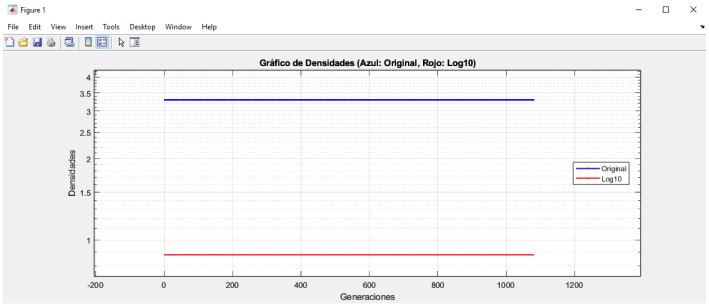

Figure 36: Densidades dela Regla 184

Al ver la gráfica, afimamos lo que ya mencionamos, iniciamos con una densidad de .75 y se mantiene constante pues es una regla que llega siempre al equilibrio dinámico. Claro el valor de la densidad de unos aumentó comparándolo con las anteriores gráficas.

Ahora veamos la media de unos.

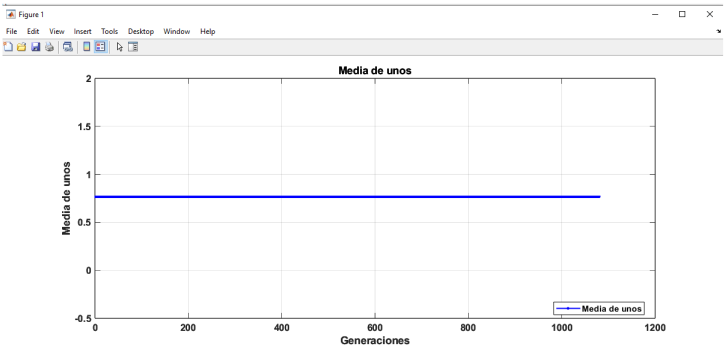

Figure 37: Media de unos de la regla 184 en un espacio de 1920x1080 pixeles

La media de unos también se mantiene constante, la diferencia claro con nuestras anteriores gráficas es que sube la grafica hasta 0.75 debido al aumento de unos.

Observemos ahora la varianza de las generaciones.

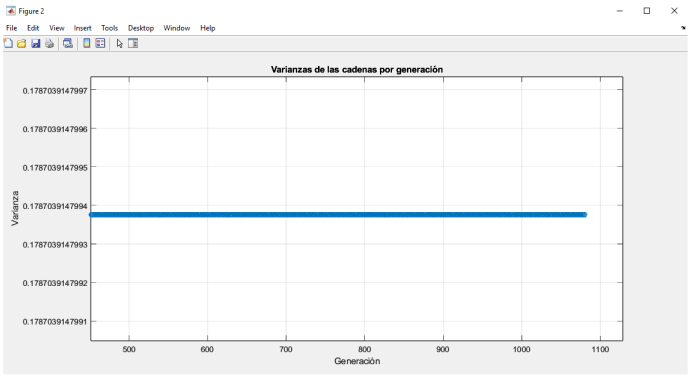

Figure 38: Varianza de la regla 184 en un espacio de 1920x1080 pixeles

Aquí podemos observar que el valor de la varianza por generación se mantiene constante, sin embargo, es menor al que teníamos cuándo analizamos el caso de una densidad inicial del 50 porciento, esto podría deberse a que la alta densidad inicial del 75 porciento lleva a patrones de congestión más estables y regulares en comparación con una densidad del 50 porciento, lo que resulta en una disminución en la variabilidad de la densidad de unos y, por lo tanto , una variación menor.

La entropía de Shannon para una densidad del 75 porciento se ve de la siguiente manera.

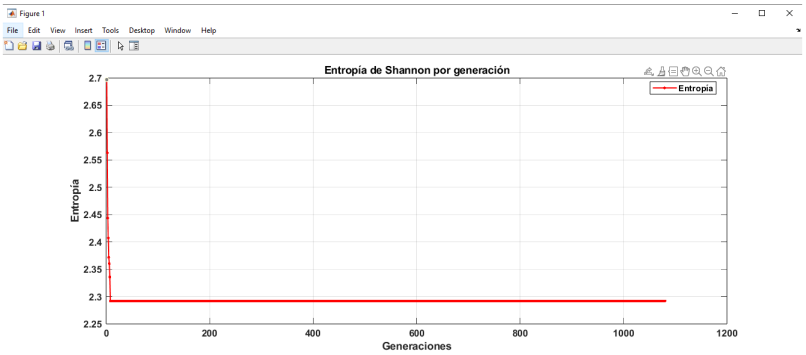

Figure 39: Entropía de Shannon para la regla 184 en un espacio de 1920x1080 pixeles

Vemos que esta gráfica se comporta similar a la que vimos con una densidad del 50 porciento, sin embargo, esta no tarda tantas generaciones en llegar al equilibrio comparada con la anterior, lo que nos dice que la entropía no es tan alta, ni siquiera al inicio pues nos da una medida de 2.7 aproximadamente, y sugiere una baja incertidumbre en el sistema.

Además, con el paso del tiempo, el autómata celular tiende a ser determinista por la regla, por lo que se estabiliza y el sistema se vuelve más predecible. La entropía constante en el nivel de 2.3 indica que existe un nivel de incertidumbre y aunque los patrones cambien de una generación a otra, la variabilidad es limitada.

- **Densidad del 95 porciento**

  Finalmente, toca analizar estadísticamente esta última densidad inicial, por lo que así se vería nuestro autómata celula bajos estas condiciones.

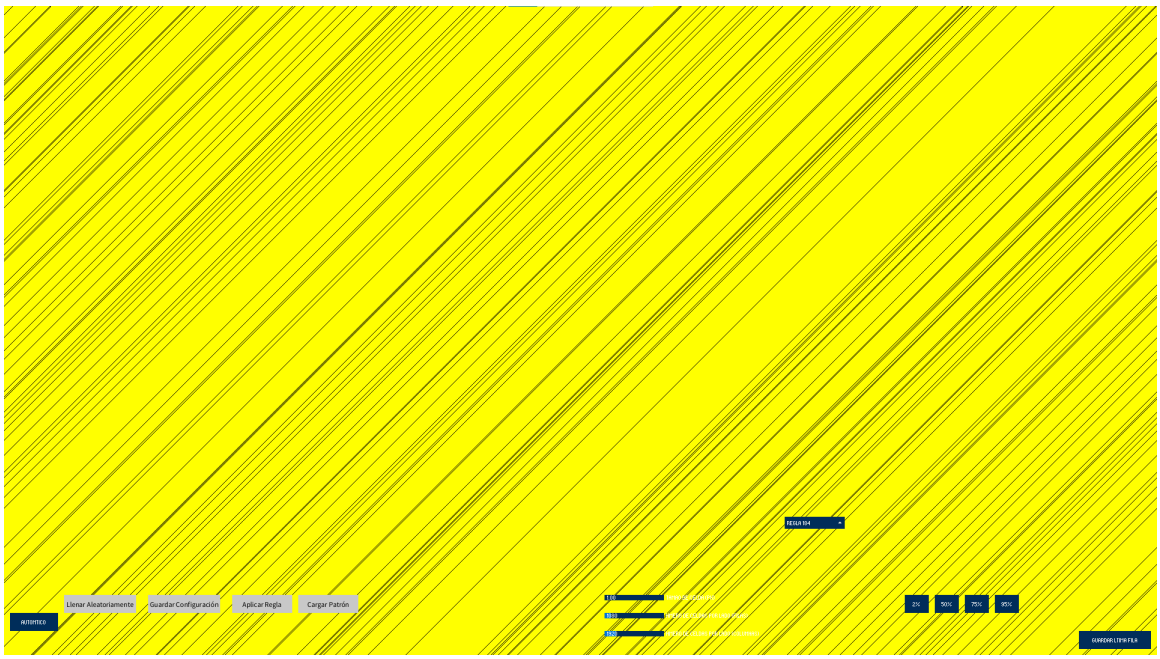

Figure 40: Regla 184 con densidad inicial del 95 porciento

Como vemos en el autómata, al iniciarlo con una densidad del 95 porciento de unos, el escenario representa una densidad de tráfico extremadamente alta en la carretera simulada. En la primera generación se representa una congestión masiva en la carretera. Esto significa que casi todas las celdas en la cuadrícula están ocupadas por vehículos, y el tráfico está prácticamente detenido.

Para las generaciones posteriores, dado que la densidad inicial es extremadamente alta, el tráfico en la carretera tiende a estar altamente congestionado en las generaciones posteriores, lo que significa que la mayoría de las celdas continúan ocupadas.

A medida que evoluciona, el sistema experimentará una congestión constante, con pocos movimientos de vehículos. El tráfico se mantiene prácticamente paralizado en la mayoría de las generaciones debido a la densidad inicial tan alta de vehículos.

Veamos la gráfica de densidad.

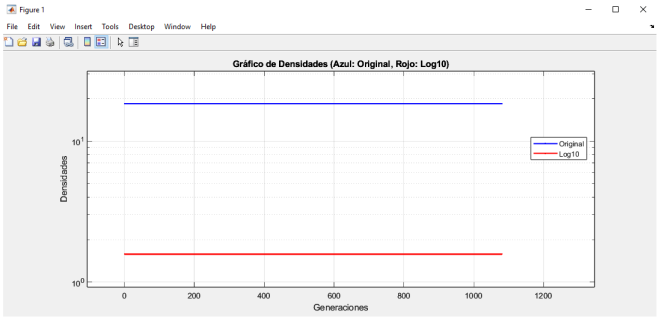

Figure 41: Densidades de la Regla 184

Podemos observar que la densidad de unos aumentó aproximadamente a 18 y esto claro porque empezamos con una alta densidad de unos, además por la naturaleza determinística de la regla es que se mantiene constante.

Veamos la media de unos.

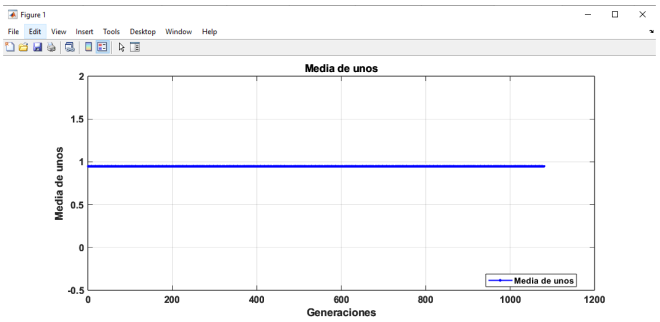

Figure 42: Media de unos de la Regla 184

La media de unos subió a .95 claro por la densidad que establecimos en un inicio y se mantiene constante por la naturaleza de la regla 184.

Ahora analicemos la varianza.

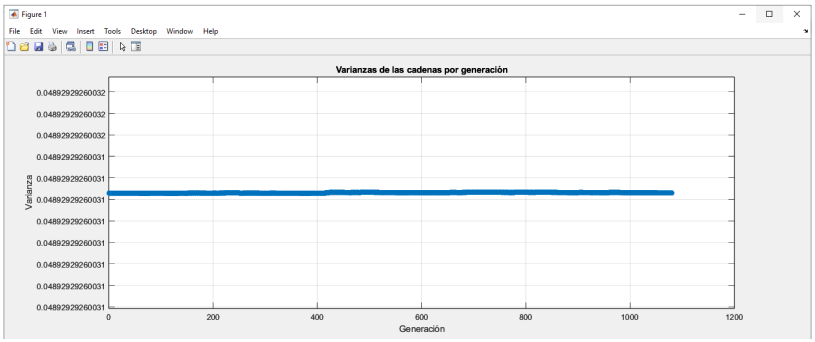

Figure 43: Varianza de las generaciones de la Regla 184

Podemos ver que además de que se mantiene constante, el valor de la varianza volvió a bajar parecido al nivel de una densidad del 2 porciento de unos inicial, esto se debe a la naturaleza de la regla pues nuestras condiciones pueden interpretarse como muy deterministas, por lo que el sistema se vuelve muy estable y predecible desde el inicio.

Finalmente, veamos la entropía de Shannon.

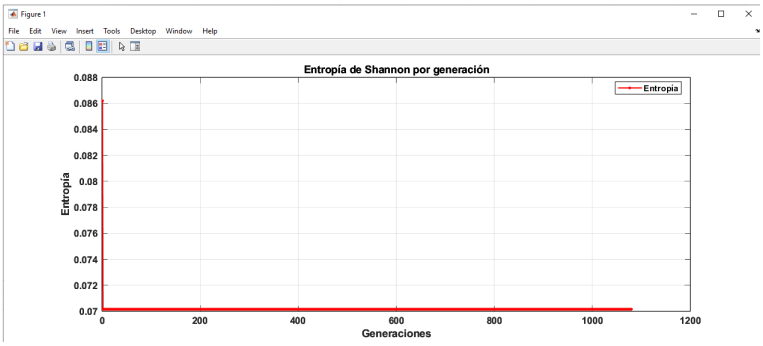

Figure 44: Entropía de Shannon para la regla 184

Podemos ver como la gráfica vuelve a ser la misma que la primera que vimos con densidad del dos porciento de unos, sin embargo ahora ya no toma valores negativos, aunque toma valores de casi cero y se mantiene constante ahí. Con lo que ya hemos explicado, decimos que la cantidad de información y la incertidumbre en ese sistema no están cambiando significativamente. En otras palabras, la entropía constante sugiere que la complejidad o la variabilidad en el sistema se mantiene en un nivel estable y predecible.

3 Código Fuente de Processing 4.3

```
import controlP5.*;
import java.io.*;
import java.util.ArrayList;

import javax.swing.*;
import processing.awt.PSurfaceAWT.SmoothCanvas;


int minCellSize = 1;  // Tamanio minimo de celda en pixeles
int maxCellSize = 100; // Tamanio maximo de celda en pixeles
int minCellsPerSide = 10; // Minimo numero de celdas por lado
int maxCellsPerSide = 10000; // Maximo numero de celdas por lado
int cols, rows; // Numero de columnas y filas basado en el tamanio de la ventana
color[][] grid; // Matriz para almacenar el color de cada celda
int[][] nextGeneration; // Matriz para la siguiente generacion
color colorViva = color(255, 0, 0); // Color para las celdas vivas (rojo)
color colorMuerta = color(255); // Color para las celdas muertas (blanco)
int generation = 1; // Generacion actual--
```

```
float percentageOnes = 0.02; // Porcentaje inicial de unos (2%)
boolean automaticMode = false; // Indica si estamos en modo automatico
float interval = 0.1; // Intervalo de tiempo en milisegundos
int lastTime = 0; // ultimo tiempo en el que se aplico la regla
boolean generateGrid = false; // Bandera para generar la cuadricula
Button generateButton; // Boton para generar la cuadricula
Button fillRandomButton; // Boton para llenar aleatoriamente la primera linea
Button saveConfigButton; // Boton para guardar la configuracion
Button applyRuleButton; // Boton para aplicar la regla
Button loadPatternButton; // Boton para cargar un patron desde un archivo
ColorPicker colorPickerViva; // Selector de color para celdas vivas
ColorPicker colorPickerMuerta; // Selector de color para celdas muertas
DropdownList ruleSelector; // Selector de regla
boolean hideGenerateButton = false; // Bandera para ocultar el boton "Generar"
boolean showUI = true; // Mostrar la interfaz de usuario

int currentRow = 0; // Variable para realizar un seguimiento del estado actual de la fila
int unosGeneracion = 0; // Contador de unos en la generacion actual
int cerosGeneracion = 0; // Contador de ceros en la generacion actual
int[] subStringCounts = new int[8];
PrintWriter writer;
PrintWriter writer1;
PrintWriter writer2;
ArrayList<Float> densidades = new ArrayList<Float>();
ArrayList<Integer> generaciones = new ArrayList<Integer>();
ArrayList<Float> medias = new ArrayList<Float>();
boolean visualizarAtractor = false;
boolean mostrarCicloAtractor = false;
PGraphics graphCanvas; // Lienzo para el ciclo atractor

ControlP5 cp5;


// Define las reglas en una matriz bidimensional
int[][][][] rules = new int[256][2][2][2];

void setup() {
        fullScreen();
        cols = width / minCellSize; // Inicialmente, se establece segun el tamanio minimo de celda
        rows = height / minCellSize; // Inicialmente, se establece segun el tamanio minimo de celda
        grid = new color[cols][rows];
        nextGeneration = new int[cols][rows];
        cp5 = new ControlP5(this);



        // Establecer la posici\'{o}n de los controles
        float xPosition = 10;
        float yPosition = height - 100; // Ajusta esta coordenada hacia arriba
        float spacing = 10;

        writer = createWriter("DENSIDAD.txt");
        writer1 = createWriter("MEDIA.txt");
        writer2 = createWriter("ENTROPY.txt");
        // Crear bot\'{o}n para generar la cuadr\'{i}cula
        generateButton = new Button(xPosition, yPosition, 80, 30, "Generar");
        xPosition += 90; // Espacio entre botones

        // Crear boton para llenar aleatoriamente la primera linea
        fillRandomButton = new Button(xPosition, yPosition, 120, 30, "Llenar Aleatoriamente");
        xPosition += 140; // Espacio entre botones

        // Crear boton para guardar la configuracion
        saveConfigButton = new Button(xPosition, yPosition, 120, 30, "Guardar Configuracion");
        xPosition += 140; // Espacio entre botones

        // Crear boton para aplicar la regla
        applyRuleButton = new Button(xPosition, yPosition, 100, 30, "Aplicar Regla");
        xPosition += 110; // Espacio entre botones

        // Crear boton para cargar un patron desde un archivo
        loadPatternButton = new Button(xPosition, yPosition, 100, 30, "Cargar Patron");
        xPosition += 110; // Espacio entre botones

        // Crear control deslizante para ajustar el tamanio de la celda
        cp5.addSlider("cellSize")
        .setPosition(1000, height - 100)
        .setRange(1, maxCellSize) // Establecer el valor minimo a 1
        .setValue(1) // Establecer el valor inicial a 1
        .setLabel("Tamanio de Celda (px)");

        // Crear control deslizante para el numero de celdas por lado (filas)
        cp5.addSlider("rows")
        .setPosition(1000, height - 70)
        .setRange(minCellsPerSide, 10000) // Establecer el valor maximo a 10000
        .setValue(rows)
        .setLabel("Numero de Celdas por Lado (Filas)");

        // Crear control deslizante para el numero de celdas por lado (columnas)
        cp5.addSlider("cols")
        .setPosition(1000, height - 40)
        .setRange(minCellsPerSide, 10000) // Establecer el valor maximo a 10000
        .setValue(cols)
        .setLabel("Numero de Celdas por Lado (Columnas)");

        // Crear el selector de regla
        ruleSelector = cp5.addDropdownList("Rule Selector")
        .setPosition(1300, 850) // Mover mas a la derecha de la ventana
        .setSize(100, 200)
        .setBarHeight(20)
        .setItemHeight(20)
        .setLabel("Regla");

        for (int i = 0; i < 256; i++) {
                ruleSelector.addItem("Regla " + i, i);
        }

        cp5.addButton("Automatico")
        .setPosition(10, height - 70)
        .setSize(80, 30)
        .setLabel("Automatico");

        // Crear selectores de color para celdas vivas y muertas
        colorPickerViva = cp5.addColorPicker("Color Viva")
        .setPosition(xPosition, yPosition)
        .setColorValue(colorViva)
```

```
.setColorBackground(colorViva);
xPosition += 100; // Espacio entre selectores

colorPickerMuerta = cp5.addColorPicker("Color-Muerta")
.setPosition(xPosition, yPosition)
.setColorValue(colorMuerta)
.setColorBackground(colorMuerta);

// Crear boton para guardar la ultima fila
cp5.addButton("Guardar-Ultima-Fila")
.setPosition(width - 130, height - 40)
.setSize(120, 30)
.setLabel("Guardar-Ultima-Fila");

cp5.addButton("Visualizar-Ciclo-Atractor")
.setPosition(10, height - 40)
.setSize(180, 30)
.setLabel("Visualizar-Ciclo-Atractor");

cp5.addButton("2%").setPosition(xPosition + 800, yPosition).setSize(40, 30);
cp5.addButton("50%").setPosition(xPosition + 850, yPosition).setSize(40, 30);
cp5.addButton("75%").setPosition(xPosition + 900, yPosition).setSize(40, 30);
cp5.addButton("95%").setPosition(xPosition + 950, yPosition).setSize(40, 30);

// Define las reglas en una matriz bidimensional
// Regla 0
rules[0][0][0][0] = 0; rules[0][0][0][1] = 0; rules[0][0][1][0] = 0; rules[0][0][1][1] = 0;
    rules[0][1][0][0] = 0; rules[0][1][0][1] = 0; rules[0][1][1][0] = 0; rules[0][1][1][1] =
    0;
rules[1][0][0][0] = 1;rules[1][0][0][1] = 0;rules[1][0][1][0] = 0;rules[1][0][1][1] =
    0;rules[1][1][0][0] = 0;rules[1][1][0][1] = 0;rules[1][1][1][0] = 0;rules[1][1][1][1] = 0;
rules[2][0][0][0] = 0;rules[2][0][0][1] = 1;rules[2][0][1][0] = 0;rules[2][0][1][1] =
    0;rules[2][1][0][0] = 0;rules[2][1][0][1] = 0;rules[2][1][1][0] = 0;rules[2][1][1][1] = 0;
rules[3][0][0][0] = 1;rules[3][0][0][1] = 1;rules[3][0][1][0] = 0;rules[3][0][1][1] =
    0;rules[3][1][0][0] = 0;rules[3][1][0][1] = 0;rules[3][1][1][0] = 0;rules[3][1][1][1] = 0;
rules[4][0][0][0] = 0;rules[4][0][0][1] = 0;rules[4][0][1][0] = 1;rules[4][0][1][1] =
    0;rules[4][1][0][0] = 0;rules[4][1][0][1] = 0;rules[4][1][1][0] = 0;rules[4][1][1][1] = 0;
rules[5][0][0][0] = 1;rules[5][0][0][1] = 0;rules[5][0][1][0] = 1;rules[5][0][1][1] =
    0;rules[5][1][0][0] = 0;rules[5][1][0][1] = 0;rules[5][1][1][0] = 0;rules[5][1][1][1] = 0;
rules[6][0][0][0] = 0;rules[6][0][0][1] = 1;rules[6][0][1][0] = 1;rules[6][0][1][1] =
    0;rules[6][1][0][0] = 0;rules[6][1][0][1] = 0;rules[6][1][1][0] = 0;rules[6][1][1][1] = 0;
rules[7][0][0][0] = 1;rules[7][0][0][1] = 1;rules[7][0][1][0] = 1;rules[7][0][1][1] =
    0;rules[7][1][0][0] = 0;rules[7][1][0][1] = 0;rules[7][1][1][0] = 0;rules[7][1][1][1] = 0;
rules[8][0][0][0] = 0;rules[8][0][0][1] = 0;rules[8][0][1][0] = 0;rules[8][0][1][1] =
    1;rules[8][1][0][0] = 0;rules[8][1][0][1] = 0;rules[8][1][1][0] = 0;rules[8][1][1][1] = 0;
rules[9][0][0][0] = 1;rules[9][0][0][1] = 0;rules[9][0][1][0] = 0;rules[9][0][1][1] =
    1;rules[9][1][0][0] = 0;rules[9][1][0][1] = 0;rules[9][1][1][0] = 0;rules[9][1][1][1] = 0;
rules[10][0][0][0] = 0;rules[10][0][0][1] = 1;rules[10][0][1][0] = 0;rules[10][0][1][1] =
    1;rules[10][1][0][0] = 0;rules[10][1][0][1] = 0;rules[10][1][1][0] = 0;rules[10][1][1][1]
    = 0;
rules[11][0][0][0] = 1;rules[11][0][0][1] = 1;rules[11][0][1][0] = 0;rules[11][0][1][1] =
    1;rules[11][1][0][0] = 0;rules[11][1][0][1] = 0;rules[11][1][1][0] = 0;rules[11][1][1][1]
    = 0;
rules[12][0][0][0] = 0;rules[12][0][0][1] = 0;rules[12][0][1][0] = 1;rules[12][0][1][1] =
    1;rules[12][1][0][0] = 0;rules[12][1][0][1] = 0;rules[12][1][1][0] = 0;rules[12][1][1][1]
    = 0;
rules[13][0][0][0] = 1;rules[13][0][0][1] = 0;rules[13][0][1][0] = 1;rules[13][0][1][1] =
    1;rules[13][1][0][0] = 0;rules[13][1][0][1] = 0;rules[13][1][1][0] = 0;rules[13][1][1][1]
    = 0;
rules[14][0][0][0] = 0;rules[14][0][0][1] = 1;rules[14][0][1][0] = 1;rules[14][0][1][1] =
    1;rules[14][1][0][0] = 0;rules[14][1][0][1] = 0;rules[14][1][1][0] = 0;rules[14][1][1][1]
    = 0;
rules[15][0][0][0] = 1;rules[15][0][0][1] = 1;rules[15][0][1][0] = 1;rules[15][0][1][1] =
    1;rules[15][1][0][0] = 0;rules[15][1][0][1] = 0;rules[15][1][1][0] = 0;rules[15][1][1][1]
    = 0;
rules[16][0][0][0] = 0;rules[16][0][0][1] = 0;rules[16][0][1][0] = 0;rules[16][0][1][1] =
    0;rules[16][1][0][0] = 1;rules[16][1][0][1] = 0;rules[16][1][1][0] = 0;rules[16][1][1][1]
    = 0;
rules[17][0][0][0] = 1;rules[17][0][0][1] = 0;rules[17][0][1][0] = 0;rules[17][0][1][1] =
    0;rules[17][1][0][0] = 1;rules[17][1][0][1] = 0;rules[17][1][1][0] = 0;rules[17][1][1][1]
    = 0;
rules[18][0][0][0] = 0;rules[18][0][0][1] = 1;rules[18][0][1][0] = 0;rules[18][0][1][1] =
    0;rules[18][1][0][0] = 1;rules[18][1][0][1] = 0;rules[18][1][1][0] = 0;rules[18][1][1][1]
    = 0;
rules[19][0][0][0] = 1;rules[19][0][0][1] = 1;rules[19][0][1][0] = 0;rules[19][0][1][1] =
    0;rules[19][1][0][0] = 1;rules[19][1][0][1] = 0;rules[19][1][1][0] = 0;rules[19][1][1][1]
    = 0;
rules[20][0][0][0] = 0;rules[20][0][0][1] = 0;rules[20][0][1][0] = 1;rules[20][0][1][1] =
    0;rules[20][1][0][0] = 1;rules[20][1][0][1] = 0;rules[20][1][1][0] = 0;rules[20][1][1][1]
    = 0;
rules[21][0][0][0] = 1;rules[21][0][0][1] = 0;rules[21][0][1][0] = 1;rules[21][0][1][1] =
    0;rules[21][1][0][0] = 1;rules[21][1][0][1] = 0;rules[21][1][1][0] = 0;rules[21][1][1][1]
    = 0;
rules[22][0][0][0] = 0;rules[22][0][0][1] = 1;rules[22][0][1][0] = 1;rules[22][0][1][1] =
    0;rules[22][1][0][0] = 1;rules[22][1][0][1] = 0;rules[22][1][1][0] = 0;rules[22][1][1][1]
    = 0;
rules[23][0][0][0] = 1;rules[23][0][0][1] = 1;rules[23][0][1][0] = 1;rules[23][0][1][1] =
    0;rules[23][1][0][0] = 1;rules[23][1][0][1] = 0;rules[23][1][1][0] = 0;rules[23][1][1][1]
    = 0;
rules[24][0][0][0] = 0;rules[24][0][0][1] = 0;rules[24][0][1][0] = 0;rules[24][0][1][1] =
    1;rules[24][1][0][0] = 1;rules[24][1][0][1] = 0;rules[24][1][1][0] = 0;rules[24][1][1][1]
    = 0;
rules[25][0][0][0] = 1;rules[25][0][0][1] = 0;rules[25][0][1][0] = 0;rules[25][0][1][1] =
    1;rules[25][1][0][0] = 1;rules[25][1][0][1] = 0;rules[25][1][1][0] = 0;rules[25][1][1][1]
    = 0;
rules[26][0][0][0] = 0;rules[26][0][0][1] = 1;rules[26][0][1][0] = 0;rules[26][0][1][1] =
    1;rules[26][1][0][0] = 1;rules[26][1][0][1] = 0;rules[26][1][1][0] = 0;rules[26][1][1][1]
    = 0;
rules[27][0][0][0] = 1;rules[27][0][0][1] = 1;rules[27][0][1][0] = 0;rules[27][0][1][1] =
    1;rules[27][1][0][0] = 1;rules[27][1][0][1] = 0;rules[27][1][1][0] = 0;rules[27][1][1][1]
    = 0;
rules[28][0][0][0] = 0;rules[28][0][0][1] = 0;rules[28][0][1][0] = 1;rules[28][0][1][1] =
    1;rules[28][1][0][0] = 1;rules[28][1][0][1] = 0;rules[28][1][1][0] = 0;rules[28][1][1][1]
    = 0;
rules[29][0][0][0] = 1;rules[29][0][0][1] = 0;rules[29][0][1][0] = 1;rules[29][0][1][1] =
    1;rules[29][1][0][0] = 1;rules[29][1][0][1] = 0;rules[29][1][1][0] = 0;rules[29][1][1][1]
    = 0;
rules[30][0][0][0] = 0;rules[30][0][0][1] = 1;rules[30][0][1][0] = 1;rules[30][0][1][1] =
    1;rules[30][1][0][0] = 1;rules[30][1][0][1] = 0;rules[30][1][1][0] = 0;rules[30][1][1][1]
    = 0;
rules[31][0][0][0] = 1;rules[31][0][0][1] = 1;rules[31][0][1][0] = 1;rules[31][0][1][1] =
    1;rules[31][1][0][0] = 1;rules[31][1][0][1] = 0;rules[31][1][1][0] = 0;rules[31][1][1][1]
    = 0;
rules[32][0][0][0] = 0;rules[32][0][0][1] = 0;rules[32][0][1][0] = 0;rules[32][0][1][1] =
    0;rules[32][1][0][0] = 0;rules[32][1][0][1] = 1;rules[32][1][1][0] = 0;rules[32][1][1][1]
    = 0;
```

```
rules[33][0][0][0] = 1;rules[33][0][0][1] = 0;rules[33][0][1][0] = 0;rules[33][0][1][1] =
    0;rules[33][1][0][0] = 0;rules[33][1][0][1] = 1;rules[33][1][1][0] = 0;rules[33][1][1][1]
    = 0;
rules[34][0][0][0] = 0;rules[34][0][0][1] = 1;rules[34][0][1][0] = 0;rules[34][0][1][1] =
    0;rules[34][1][0][0] = 0;rules[34][1][0][1] = 1;rules[34][1][1][0] = 0;rules[34][1][1][1]
    = 0;
rules[35][0][0][0] = 1;rules[35][0][0][1] = 1;rules[35][0][1][0] = 0;rules[35][0][1][1] =
    0;rules[35][1][0][0] = 0;rules[35][1][0][1] = 1;rules[35][1][1][0] = 0;rules[35][1][1][1]
    = 0;
rules[36][0][0][0] = 0;rules[36][0][0][1] = 0;rules[36][0][1][0] = 1;rules[36][0][1][1] =
    0;rules[36][1][0][0] = 0;rules[36][1][0][1] = 1;rules[36][1][1][0] = 0;rules[36][1][1][1]
    = 0;
rules[37][0][0][0] = 1;rules[37][0][0][1] = 0;rules[37][0][1][0] = 1;rules[37][0][1][1] =
    0;rules[37][1][0][0] = 0;rules[37][1][0][1] = 1;rules[37][1][1][0] = 0;rules[37][1][1][1]
    = 0;
rules[38][0][0][0] = 0;rules[38][0][0][1] = 1;rules[38][0][1][0] = 1;rules[38][0][1][1] =
    0;rules[38][1][0][0] = 0;rules[38][1][0][1] = 1;rules[38][1][1][0] = 0;rules[38][1][1][1]
    = 0;
rules[39][0][0][0] = 1;rules[39][0][0][1] = 1;rules[39][0][1][0] = 1;rules[39][0][1][1] =
    0;rules[39][1][0][0] = 0;rules[39][1][0][1] = 1;rules[39][1][1][0] = 0;rules[39][1][1][1]
    = 0;
rules[40][0][0][0] = 0;rules[40][0][0][1] = 0;rules[40][0][1][0] = 0;rules[40][0][1][1] =
    1;rules[40][1][0][0] = 0;rules[40][1][0][1] = 1;rules[40][1][1][0] = 0;rules[40][1][1][1]
    = 0;
rules[41][0][0][0] = 1;rules[41][0][0][1] = 0;rules[41][0][1][0] = 0;rules[41][0][1][1] =
    1;rules[41][1][0][0] = 0;rules[41][1][0][1] = 1;rules[41][1][1][0] = 0;rules[41][1][1][1]
    = 0;
rules[42][0][0][0] = 0;rules[42][0][0][1] = 1;rules[42][0][1][0] = 0;rules[42][0][1][1] =
    1;rules[42][1][0][0] = 0;rules[42][1][0][1] = 1;rules[42][1][1][0] = 0;rules[42][1][1][1]
    = 0;
rules[43][0][0][0] = 1;rules[43][0][0][1] = 1;rules[43][0][1][0] = 0;rules[43][0][1][1] =
    1;rules[43][1][0][0] = 0;rules[43][1][0][1] = 1;rules[43][1][1][0] = 0;rules[43][1][1][1]
    = 0;
rules[44][0][0][0] = 0;rules[44][0][0][1] = 0;rules[44][0][1][0] = 1;rules[44][0][1][1] =
    1;rules[44][1][0][0] = 0;rules[44][1][0][1] = 1;rules[44][1][1][0] = 0;rules[44][1][1][1]
    = 0;
rules[45][0][0][0] = 1;rules[45][0][0][1] = 0;rules[45][0][1][0] = 1;rules[45][0][1][1] =
    1;rules[45][1][0][0] = 0;rules[45][1][0][1] = 1;rules[45][1][1][0] = 0;rules[45][1][1][1]
    = 0;
rules[46][0][0][0] = 0;rules[46][0][0][1] = 1;rules[46][0][1][0] = 1;rules[46][0][1][1] =
    1;rules[46][1][0][0] = 0;rules[46][1][0][1] = 1;rules[46][1][1][0] = 0;rules[46][1][1][1]
    = 0;
rules[47][0][0][0] = 1;rules[47][0][0][1] = 1;rules[47][0][1][0] = 1;rules[47][0][1][1] =
    1;rules[47][1][0][0] = 0;rules[47][1][0][1] = 1;rules[47][1][1][0] = 0;rules[47][1][1][1]
    = 0;
rules[48][0][0][0] = 0;rules[48][0][0][1] = 0;rules[48][0][1][0] = 0;rules[48][0][1][1] =
    0;rules[48][1][0][0] = 1;rules[48][1][0][1] = 1;rules[48][1][1][0] = 0;rules[48][1][1][1]
    = 0;
rules[49][0][0][0] = 1;rules[49][0][0][1] = 0;rules[49][0][1][0] = 0;rules[49][0][1][1] =
    0;rules[49][1][0][0] = 1;rules[49][1][0][1] = 1;rules[49][1][1][0] = 0;rules[49][1][1][1]
    = 0;
rules[50][0][0][0] = 0;rules[50][0][0][1] = 1;rules[50][0][1][0] = 0;rules[50][0][1][1] =
    0;rules[50][1][0][0] = 1;rules[50][1][0][1] = 1;rules[50][1][1][0] = 0;rules[50][1][1][1]
    = 0;
rules[51][0][0][0] = 1;rules[51][0][0][1] = 1;rules[51][0][1][0] = 0;rules[51][0][1][1] =
    0;rules[51][1][0][0] = 1;rules[51][1][0][1] = 1;rules[51][1][1][0] = 0;rules[51][1][1][1]
    = 0;
rules[52][0][0][0] = 0;rules[52][0][0][1] = 0;rules[52][0][1][0] = 1;rules[52][0][1][1] =
    0;rules[52][1][0][0] = 1;rules[52][1][0][1] = 1;rules[52][1][1][0] = 0;rules[52][1][1][1]
    = 0;
rules[53][0][0][0] = 1;rules[53][0][0][1] = 0;rules[53][0][1][0] = 1;rules[53][0][1][1] =
    0;rules[53][1][0][0] = 1;rules[53][1][0][1] = 1;rules[53][1][1][0] = 0;rules[53][1][1][1]
    = 0;
rules[54][0][0][0] = 0;rules[54][0][0][1] = 1;rules[54][0][1][0] = 1;rules[54][0][1][1] =
    0;rules[54][1][0][0] = 1;rules[54][1][0][1] = 1;rules[54][1][1][0] = 0;rules[54][1][1][1]
    = 0;
rules[55][0][0][0] = 1;rules[55][0][0][1] = 1;rules[55][0][1][0] = 1;rules[55][0][1][1] =
    0;rules[55][1][0][0] = 1;rules[55][1][0][1] = 1;rules[55][1][1][0] = 0;rules[55][1][1][1]
    = 0;
rules[56][0][0][0] = 0;rules[56][0][0][1] = 0;rules[56][0][1][0] = 0;rules[56][0][1][1] =
    1;rules[56][1][0][0] = 1;rules[56][1][0][1] = 1;rules[56][1][1][0] = 0;rules[56][1][1][1]
    = 0;
rules[57][0][0][0] = 1;rules[57][0][0][1] = 0;rules[57][0][1][0] = 0;rules[57][0][1][1] =
    1;rules[57][1][0][0] = 1;rules[57][1][0][1] = 1;rules[57][1][1][0] = 0;rules[57][1][1][1]
    = 0;
rules[58][0][0][0] = 0;rules[58][0][0][1] = 1;rules[58][0][1][0] = 0;rules[58][0][1][1] =
    1;rules[58][1][0][0] = 1;rules[58][1][0][1] = 1;rules[58][1][1][0] = 0;rules[58][1][1][1]
    = 0;
rules[59][0][0][0] = 1;rules[59][0][0][1] = 1;rules[59][0][1][0] = 0;rules[59][0][1][1] =
    1;rules[59][1][0][0] = 1;rules[59][1][0][1] = 1;rules[59][1][1][0] = 0;rules[59][1][1][1]
    = 0;
rules[60][0][0][0] = 0;rules[60][0][0][1] = 0;rules[60][0][1][0] = 1;rules[60][0][1][1] =
    1;rules[60][1][0][0] = 1;rules[60][1][0][1] = 1;rules[60][1][1][0] = 0;rules[60][1][1][1]
    = 0;
rules[61][0][0][0] = 1;rules[61][0][0][1] = 0;rules[61][0][1][0] = 1;rules[61][0][1][1] =
    1;rules[61][1][0][0] = 1;rules[61][1][0][1] = 1;rules[61][1][1][0] = 0;rules[61][1][1][1]
    = 0;
rules[62][0][0][0] = 0;rules[62][0][0][1] = 1;rules[62][0][1][0] = 1;rules[62][0][1][1] =
    1;rules[62][1][0][0] = 1;rules[62][1][0][1] = 1;rules[62][1][1][0] = 0;rules[62][1][1][1]
    = 0;
rules[63][0][0][0] = 1;rules[63][0][0][1] = 1;rules[63][0][1][0] = 1;rules[63][0][1][1] =
    1;rules[63][1][0][0] = 1;rules[63][1][0][1] = 1;rules[63][1][1][0] = 0;rules[63][1][1][1]
    = 0;
rules[64][0][0][0] = 0;rules[64][0][0][1] = 0;rules[64][0][1][0] = 0;rules[64][0][1][1] =
    0;rules[64][1][0][0] = 0;rules[64][1][0][1] = 0;rules[64][1][1][0] = 1;rules[64][1][1][1]
    = 0;
rules[65][0][0][0] = 1;rules[65][0][0][1] = 0;rules[65][0][1][0] = 0;rules[65][0][1][1] =
    0;rules[65][1][0][0] = 0;rules[65][1][0][1] = 0;rules[65][1][1][0] = 1;rules[65][1][1][1]
    = 0;
rules[66][0][0][0] = 0;rules[66][0][0][1] = 1;rules[66][0][1][0] = 0;rules[66][0][1][1] =
    0;rules[66][1][0][0] = 0;rules[66][1][0][1] = 0;rules[66][1][1][0] = 1;rules[66][1][1][1]
    = 0;
rules[67][0][0][0] = 1;rules[67][0][0][1] = 1;rules[67][0][1][0] = 0;rules[67][0][1][1] =
    0;rules[67][1][0][0] = 0;rules[67][1][0][1] = 0;rules[67][1][1][0] = 1;rules[67][1][1][1]
    = 0;
rules[68][0][0][0] = 0;rules[68][0][0][1] = 0;rules[68][0][1][0] = 1;rules[68][0][1][1] =
    0;rules[68][1][0][0] = 0;rules[68][1][0][1] = 0;rules[68][1][1][0] = 1;rules[68][1][1][1]
    = 0;
rules[69][0][0][0] = 1;rules[69][0][0][1] = 0;rules[69][0][1][0] = 1;rules[69][0][1][1] =
    0;rules[69][1][0][0] = 0;rules[69][1][0][1] = 0;rules[69][1][1][0] = 1;rules[69][1][1][1]
    = 0;
rules[70][0][0][0] = 0;rules[70][0][0][1] = 1;rules[70][0][1][0] = 1;rules[70][0][1][1] =
    0;rules[70][1][0][0] = 0;rules[70][1][0][1] = 0;rules[70][1][1][0] = 1;rules[70][1][1][1]
    = 0;
rules[71][0][0][0] = 1;rules[71][0][0][1] = 1;rules[71][0][1][0] = 1;rules[71][0][1][1] =
    0;rules[71][1][0][0] = 0;rules[71][1][0][1] = 0;rules[71][1][1][0] = 1;rules[71][1][1][1]
```

```
    = 0;
rules[72][0][0][0] = 0;rules[72][0][0][1] = 0;rules[72][0][1][0] = 0;rules[72][0][1][1] =
    1;rules[72][1][0][0] = 0;rules[72][1][0][1] = 0;rules[72][1][1][0] = 1;rules[72][1][1][1]
    = 0;
rules[73][0][0][0] = 1;rules[73][0][0][1] = 0;rules[73][0][1][0] = 0;rules[73][0][1][1] =
    1;rules[73][1][0][0] = 0;rules[73][1][0][1] = 0;rules[73][1][1][0] = 1;rules[73][1][1][1]
    = 0;
rules[74][0][0][0] = 0;rules[74][0][0][1] = 1;rules[74][0][1][0] = 0;rules[74][0][1][1] =
    1;rules[74][1][0][0] = 0;rules[74][1][0][1] = 0;rules[74][1][1][0] = 1;rules[74][1][1][1]
    = 0;
rules[75][0][0][0] = 1;rules[75][0][0][1] = 1;rules[75][0][1][0] = 0;rules[75][0][1][1] =
    1;rules[75][1][0][0] = 0;rules[75][1][0][1] = 0;rules[75][1][1][0] = 1;rules[75][1][1][1]
    = 0;
rules[76][0][0][0] = 0;rules[76][0][0][1] = 0;rules[76][0][1][0] = 1;rules[76][0][1][1] =
    1;rules[76][1][0][0] = 0;rules[76][1][0][1] = 0;rules[76][1][1][0] = 1;rules[76][1][1][1]
    = 0;
rules[77][0][0][0] = 1;rules[77][0][0][1] = 0;rules[77][0][1][0] = 1;rules[77][0][1][1] =
    1;rules[77][1][0][0] = 0;rules[77][1][0][1] = 0;rules[77][1][1][0] = 1;rules[77][1][1][1]
    = 0;
rules[78][0][0][0] = 0;rules[78][0][0][1] = 1;rules[78][0][1][0] = 1;rules[78][0][1][1] =
    1;rules[78][1][0][0] = 0;rules[78][1][0][1] = 0;rules[78][1][1][0] = 1;rules[78][1][1][1]
    = 0;
rules[79][0][0][0] = 1;rules[79][0][0][1] = 1;rules[79][0][1][0] = 1;rules[79][0][1][1] =
    1;rules[79][1][0][0] = 0;rules[79][1][0][1] = 0;rules[79][1][1][0] = 1;rules[79][1][1][1]
    = 0;
rules[80][0][0][0] = 0;rules[80][0][0][1] = 0;rules[80][0][1][0] = 0;rules[80][0][1][1] =
    0;rules[80][1][0][0] = 1;rules[80][1][0][1] = 0;rules[80][1][1][0] = 1;rules[80][1][1][1]
    = 0;
rules[81][0][0][0] = 1;rules[81][0][0][1] = 0;rules[81][0][1][0] = 0;rules[81][0][1][1] =
    0;rules[81][1][0][0] = 1;rules[81][1][0][1] = 0;rules[81][1][1][0] = 1;rules[81][1][1][1]
    = 0;
rules[82][0][0][0] = 0;rules[82][0][0][1] = 1;rules[82][0][1][0] = 0;rules[82][0][1][1] =
    0;rules[82][1][0][0] = 1;rules[82][1][0][1] = 0;rules[82][1][1][0] = 1;rules[82][1][1][1]
    = 0;
rules[83][0][0][0] = 1;rules[83][0][0][1] = 1;rules[83][0][1][0] = 0;rules[83][0][1][1] =
    0;rules[83][1][0][0] = 1;rules[83][1][0][1] = 0;rules[83][1][1][0] = 1;rules[83][1][1][1]
    = 0;
rules[84][0][0][0] = 0;rules[84][0][0][1] = 0;rules[84][0][1][0] = 1;rules[84][0][1][1] =
    0;rules[84][1][0][0] = 1;rules[84][1][0][1] = 0;rules[84][1][1][0] = 1;rules[84][1][1][1]
    = 0;
rules[85][0][0][0] = 1;rules[85][0][0][1] = 0;rules[85][0][1][0] = 1;rules[85][0][1][1] =
    0;rules[85][1][0][0] = 1;rules[85][1][0][1] = 0;rules[85][1][1][0] = 1;rules[85][1][1][1]
    = 0;
rules[86][0][0][0] = 0;rules[86][0][0][1] = 1;rules[86][0][1][0] = 1;rules[86][0][1][1] =
    0;rules[86][1][0][0] = 1;rules[86][1][0][1] = 0;rules[86][1][1][0] = 1;rules[86][1][1][1]
    = 0;
rules[87][0][0][0] = 1;rules[87][0][0][1] = 1;rules[87][0][1][0] = 1;rules[87][0][1][1] =
    0;rules[87][1][0][0] = 1;rules[87][1][0][1] = 0;rules[87][1][1][0] = 1;rules[87][1][1][1]
    = 0;
rules[88][0][0][0] = 0;rules[88][0][0][1] = 0;rules[88][0][1][0] = 0;rules[88][0][1][1] =
    1;rules[88][1][0][0] = 1;rules[88][1][0][1] = 0;rules[88][1][1][0] = 1;rules[88][1][1][1]
    = 0;
rules[89][0][0][0] = 1;rules[89][0][0][1] = 0;rules[89][0][1][0] = 0;rules[89][0][1][1] =
    1;rules[89][1][0][0] = 1;rules[89][1][0][1] = 0;rules[89][1][1][0] = 1;rules[89][1][1][1]
    = 0;
rules[90][0][0][0] = 0;rules[90][0][0][1] = 1;rules[90][0][1][0] = 0;rules[90][0][1][1] =
    1;rules[90][1][0][0] = 1;rules[90][1][0][1] = 0;rules[90][1][1][0] = 1;rules[90][1][1][1]
    = 0;
rules[91][0][0][0] = 1;rules[91][0][0][1] = 1;rules[91][0][1][0] = 0;rules[91][0][1][1] =
    1;rules[91][1][0][0] = 1;rules[91][1][0][1] = 0;rules[91][1][1][0] = 1;rules[91][1][1][1]
    = 0;
rules[92][0][0][0] = 0;rules[92][0][0][1] = 0;rules[92][0][1][0] = 1;rules[92][0][1][1] =
    1;rules[92][1][0][0] = 1;rules[92][1][0][1] = 0;rules[92][1][1][0] = 1;rules[92][1][1][1]
    = 0;
rules[93][0][0][0] = 1;rules[93][0][0][1] = 0;rules[93][0][1][0] = 1;rules[93][0][1][1] =
    1;rules[93][1][0][0] = 1;rules[93][1][0][1] = 0;rules[93][1][1][0] = 1;rules[93][1][1][1]
    = 0;
rules[94][0][0][0] = 0;rules[94][0][0][1] = 1;rules[94][0][1][0] = 1;rules[94][0][1][1] =
    1;rules[94][1][0][0] = 1;rules[94][1][0][1] = 0;rules[94][1][1][0] = 1;rules[94][1][1][1]
    = 0;
rules[95][0][0][0] = 1;rules[95][0][0][1] = 1;rules[95][0][1][0] = 1;rules[95][0][1][1] =
    1;rules[95][1][0][0] = 1;rules[95][1][0][1] = 0;rules[95][1][1][0] = 1;rules[95][1][1][1]
    = 0;
rules[96][0][0][0] = 0;rules[96][0][0][1] = 0;rules[96][0][1][0] = 0;rules[96][0][1][1] =
    0;rules[96][1][0][0] = 0;rules[96][1][0][1] = 1;rules[96][1][1][0] = 1;rules[96][1][1][1]
    = 0;
rules[97][0][0][0] = 1;rules[97][0][0][1] = 0;rules[97][0][1][0] = 0;rules[97][0][1][1] =
    0;rules[97][1][0][0] = 0;rules[97][1][0][1] = 1;rules[97][1][1][0] = 1;rules[97][1][1][1]
    = 0;
rules[98][0][0][0] = 0;rules[98][0][0][1] = 1;rules[98][0][1][0] = 0;rules[98][0][1][1] =
    0;rules[98][1][0][0] = 0;rules[98][1][0][1] = 1;rules[98][1][1][0] = 1;rules[98][1][1][1]
    = 0;
rules[99][0][0][0] = 1;rules[99][0][0][1] = 1;rules[99][0][1][0] = 0;rules[99][0][1][1] =
    0;rules[99][1][0][0] = 0;rules[99][1][0][1] = 1;rules[99][1][1][0] = 1;rules[99][1][1][1]
    = 0;
rules[100][0][0][0] = 0;rules[100][0][0][1] = 0;rules[100][0][1][0] = 1;rules[100][0][1][1] =
    0;rules[100][1][0][0] = 0;rules[100][1][0][1] = 1;rules[100][1][1][0] =
    1;rules[100][1][1][1] = 0;
rules[101][0][0][0] = 1;rules[101][0][0][1] = 0;rules[101][0][1][0] = 1;rules[101][0][1][1] =
    0;rules[101][1][0][0] = 0;rules[101][1][0][1] = 1;rules[101][1][1][0] =
    1;rules[101][1][1][1] = 0;
rules[102][0][0][0] = 0;rules[102][0][0][1] = 1;rules[102][0][1][0] = 1;rules[102][0][1][1] =
    0;rules[102][1][0][0] = 0;rules[102][1][0][1] = 1;rules[102][1][1][0] =
    1;rules[102][1][1][1] = 0;
rules[103][0][0][0] = 1;rules[103][0][0][1] = 1;rules[103][0][1][0] = 1;rules[103][0][1][1] =
    0;rules[103][1][0][0] = 0;rules[103][1][0][1] = 1;rules[103][1][1][0] =
    1;rules[103][1][1][1] = 0;
rules[104][0][0][0] = 0;rules[104][0][0][1] = 0;rules[104][0][1][0] = 0;rules[104][0][1][1] =
    1;rules[104][1][0][0] = 0;rules[104][1][0][1] = 1;rules[104][1][1][0] =
    1;rules[104][1][1][1] = 0;
rules[105][0][0][0] = 1;rules[105][0][0][1] = 0;rules[105][0][1][0] = 0;rules[105][0][1][1] =
    1;rules[105][1][0][0] = 0;rules[105][1][0][1] = 1;rules[105][1][1][0] =
    1;rules[105][1][1][1] = 0;
rules[106][0][0][0] = 0;rules[106][0][0][1] = 1;rules[106][0][1][0] = 0;rules[106][0][1][1] =
    1;rules[106][1][0][0] = 0;rules[106][1][0][1] = 1;rules[106][1][1][0] =
    1;rules[106][1][1][1] = 0;
rules[107][0][0][0] = 1;rules[107][0][0][1] = 1;rules[107][0][1][0] = 0;rules[107][0][1][1] =
    1;rules[107][1][0][0] = 0;rules[107][1][0][1] = 1;rules[107][1][1][0] =
    1;rules[107][1][1][1] = 0;
rules[108][0][0][0] = 0;rules[108][0][0][1] = 0;rules[108][0][1][0] = 1;rules[108][0][1][1] =
    1;rules[108][1][0][0] = 0;rules[108][1][0][1] = 1;rules[108][1][1][0] =
    1;rules[108][1][1][1] = 0;
rules[109][0][0][0] = 1;rules[109][0][0][1] = 0;rules[109][0][1][0] = 1;rules[109][0][1][1] =
    1;rules[109][1][0][0] = 0;rules[109][1][0][1] = 1;rules[109][1][1][0] =
    1;rules[109][1][1][1] = 0;
rules[110][0][0][0] = 0;rules[110][0][0][1] = 1;rules[110][0][1][0] = 1;rules[110][0][1][1] =
```

```
    1;rules[110][1][0][0] = 0;rules[110][1][0][1] = 1;rules[110][1][1][0] =
    1;rules[110][1][1][1] = 0;
rules[111][0][0][0] = 1;rules[111][0][0][1] = 1;rules[111][0][1][0] = 1;rules[111][0][1][1] =
    1;rules[111][1][0][0] = 0;rules[111][1][0][1] = 1;rules[111][1][1][0] =
    1;rules[111][1][1][1] = 0;
rules[112][0][0][0] = 0;rules[112][0][0][1] = 0;rules[112][0][1][0] = 0;rules[112][0][1][1] =
    0;rules[112][1][0][0] = 1;rules[112][1][0][1] = 1;rules[112][1][1][0] =
    1;rules[112][1][1][1] = 0;
rules[113][0][0][0] = 1;rules[113][0][0][1] = 0;rules[113][0][1][0] = 0;rules[113][0][1][1] =
    0;rules[113][1][0][0] = 1;rules[113][1][0][1] = 1;rules[113][1][1][0] =
    1;rules[113][1][1][1] = 0;
rules[114][0][0][0] = 0;rules[114][0][0][1] = 1;rules[114][0][1][0] = 0;rules[114][0][1][1] =
    0;rules[114][1][0][0] = 1;rules[114][1][0][1] = 1;rules[114][1][1][0] =
    1;rules[114][1][1][1] = 0;
rules[115][0][0][0] = 1;rules[115][0][0][1] = 1;rules[115][0][1][0] = 0;rules[115][0][1][1] =
    0;rules[115][1][0][0] = 1;rules[115][1][0][1] = 1;rules[115][1][1][0] =
    1;rules[115][1][1][1] = 0;
rules[116][0][0][0] = 0;rules[116][0][0][1] = 0;rules[116][0][1][0] = 1;rules[116][0][1][1] =
    0;rules[116][1][0][0] = 1;rules[116][1][0][1] = 1;rules[116][1][1][0] =
    1;rules[116][1][1][1] = 0;
rules[117][0][0][0] = 1;rules[117][0][0][1] = 0;rules[117][0][1][0] = 1;rules[117][0][1][1] =
    0;rules[117][1][0][0] = 1;rules[117][1][0][1] = 1;rules[117][1][1][0] =
    1;rules[117][1][1][1] = 0;
rules[118][0][0][0] = 0;rules[118][0][0][1] = 1;rules[118][0][1][0] = 1;rules[118][0][1][1] =
    0;rules[118][1][0][0] = 1;rules[118][1][0][1] = 1;rules[118][1][1][0] =
    1;rules[118][1][1][1] = 0;
rules[119][0][0][0] = 1;rules[119][0][0][1] = 1;rules[119][0][1][0] = 1;rules[119][0][1][1] =
    0;rules[119][1][0][0] = 1;rules[119][1][0][1] = 1;rules[119][1][1][0] =
    1;rules[119][1][1][1] = 0;
rules[120][0][0][0] = 0;rules[120][0][0][1] = 0;rules[120][0][1][0] = 0;rules[120][0][1][1] =
    1;rules[120][1][0][0] = 1;rules[120][1][0][1] = 1;rules[120][1][1][0] =
    1;rules[120][1][1][1] = 0;
rules[121][0][0][0] = 1;rules[121][0][0][1] = 0;rules[121][0][1][0] = 0;rules[121][0][1][1] =
    1;rules[121][1][0][0] = 1;rules[121][1][0][1] = 1;rules[121][1][1][0] =
    1;rules[121][1][1][1] = 0;
rules[122][0][0][0] = 0;rules[122][0][0][1] = 1;rules[122][0][1][0] = 0;rules[122][0][1][1] =
    1;rules[122][1][0][0] = 1;rules[122][1][0][1] = 1;rules[122][1][1][0] =
    1;rules[122][1][1][1] = 0;
rules[123][0][0][0] = 1;rules[123][0][0][1] = 1;rules[123][0][1][0] = 0;rules[123][0][1][1] =
    1;rules[123][1][0][0] = 1;rules[123][1][0][1] = 1;rules[123][1][1][0] =
    1;rules[123][1][1][1] = 0;
rules[124][0][0][0] = 0;rules[124][0][0][1] = 0;rules[124][0][1][0] = 1;rules[124][0][1][1] =
    1;rules[124][1][0][0] = 1;rules[124][1][0][1] = 1;rules[124][1][1][0] =
    1;rules[124][1][1][1] = 0;
rules[125][0][0][0] = 1;rules[125][0][0][1] = 0;rules[125][0][1][0] = 1;rules[125][0][1][1] =
    1;rules[125][1][0][0] = 1;rules[125][1][0][1] = 1;rules[125][1][1][0] =
    1;rules[125][1][1][1] = 0;
rules[126][0][0][0] = 0;rules[126][0][0][1] = 1;rules[126][0][1][0] = 1;rules[126][0][1][1] =
    1;rules[126][1][0][0] = 1;rules[126][1][0][1] = 1;rules[126][1][1][0] =
    1;rules[126][1][1][1] = 0;
rules[127][0][0][0] = 1;rules[127][0][0][1] = 1;rules[127][0][1][0] = 1;rules[127][0][1][1] =
    1;rules[127][1][0][0] = 1;rules[127][1][0][1] = 1;rules[127][1][1][0] =
    1;rules[127][1][1][1] = 0;
rules[128][0][0][0] = 0;rules[128][0][0][1] = 0;rules[128][0][1][0] = 0;rules[128][0][1][1] =
    0;rules[128][1][0][0] = 0;rules[128][1][0][1] = 0;rules[128][1][1][0] =
    0;rules[128][1][1][1] = 1;
rules[129][0][0][0] = 1;rules[129][0][0][1] = 0;rules[129][0][1][0] = 0;rules[129][0][1][1] =
    0;rules[129][1][0][0] = 0;rules[129][1][0][1] = 0;rules[129][1][1][0] =
    0;rules[129][1][1][1] = 1;
rules[130][0][0][0] = 0;rules[130][0][0][1] = 1;rules[130][0][1][0] = 0;rules[130][0][1][1] =
    0;rules[130][1][0][0] = 0;rules[130][1][0][1] = 0;rules[130][1][1][0] =
    0;rules[130][1][1][1] = 1;
rules[131][0][0][0] = 1;rules[131][0][0][1] = 1;rules[131][0][1][0] = 0;rules[131][0][1][1] =
    0;rules[131][1][0][0] = 0;rules[131][1][0][1] = 0;rules[131][1][1][0] =
    0;rules[131][1][1][1] = 1;
rules[132][0][0][0] = 0;rules[132][0][0][1] = 0;rules[132][0][1][0] = 1;rules[132][0][1][1] =
    0;rules[132][1][0][0] = 0;rules[132][1][0][1] = 0;rules[132][1][1][0] =
    0;rules[132][1][1][1] = 1;
rules[133][0][0][0] = 1;rules[133][0][0][1] = 0;rules[133][0][1][0] = 1;rules[133][0][1][1] =
    0;rules[133][1][0][0] = 0;rules[133][1][0][1] = 0;rules[133][1][1][0] =
    0;rules[133][1][1][1] = 1;
rules[134][0][0][0] = 0;rules[134][0][0][1] = 1;rules[134][0][1][0] = 1;rules[134][0][1][1] =
    0;rules[134][1][0][0] = 0;rules[134][1][0][1] = 0;rules[134][1][1][0] =
    0;rules[134][1][1][1] = 1;
rules[135][0][0][0] = 1;rules[135][0][0][1] = 1;rules[135][0][1][0] = 1;rules[135][0][1][1] =
    0;rules[135][1][0][0] = 0;rules[135][1][0][1] = 0;rules[135][1][1][0] =
    0;rules[135][1][1][1] = 1;
rules[136][0][0][0] = 0;rules[136][0][0][1] = 0;rules[136][0][1][0] = 0;rules[136][0][1][1] =
    1;rules[136][1][0][0] = 0;rules[136][1][0][1] = 0;rules[136][1][1][0] =
    0;rules[136][1][1][1] = 1;
rules[137][0][0][0] = 1;rules[137][0][0][1] = 0;rules[137][0][1][0] = 0;rules[137][0][1][1] =
    1;rules[137][1][0][0] = 0;rules[137][1][0][1] = 0;rules[137][1][1][0] =
    0;rules[137][1][1][1] = 1;
rules[138][0][0][0] = 0;rules[138][0][0][1] = 1;rules[138][0][1][0] = 0;rules[138][0][1][1] =
    1;rules[138][1][0][0] = 0;rules[138][1][0][1] = 0;rules[138][1][1][0] =
    0;rules[138][1][1][1] = 1;
rules[139][0][0][0] = 1;rules[139][0][0][1] = 1;rules[139][0][1][0] = 0;rules[139][0][1][1] =
    1;rules[139][1][0][0] = 0;rules[139][1][0][1] = 0;rules[139][1][1][0] =
    0;rules[139][1][1][1] = 1;
rules[140][0][0][0] = 0;rules[140][0][0][1] = 0;rules[140][0][1][0] = 1;rules[140][0][1][1] =
    1;rules[140][1][0][0] = 0;rules[140][1][0][1] = 0;rules[140][1][1][0] =
    0;rules[140][1][1][1] = 1;
rules[141][0][0][0] = 1;rules[141][0][0][1] = 0;rules[141][0][1][0] = 1;rules[141][0][1][1] =
    1;rules[141][1][0][0] = 0;rules[141][1][0][1] = 0;rules[141][1][1][0] =
    0;rules[141][1][1][1] = 1;
rules[142][0][0][0] = 0;rules[142][0][0][1] = 1;rules[142][0][1][0] = 1;rules[142][0][1][1] =
    1;rules[142][1][0][0] = 0;rules[142][1][0][1] = 0;rules[142][1][1][0] =
    0;rules[142][1][1][1] = 1;
rules[143][0][0][0] = 1;rules[143][0][0][1] = 1;rules[143][0][1][0] = 1;rules[143][0][1][1] =
    1;rules[143][1][0][0] = 0;rules[143][1][0][1] = 0;rules[143][1][1][0] =
    0;rules[143][1][1][1] = 1;
rules[144][0][0][0] = 0;rules[144][0][0][1] = 0;rules[144][0][1][0] = 0;rules[144][0][1][1] =
    0;rules[144][1][0][0] = 1;rules[144][1][0][1] = 0;rules[144][1][1][0] =
    0;rules[144][1][1][1] = 1;
rules[145][0][0][0] = 1;rules[145][0][0][1] = 0;rules[145][0][1][0] = 0;rules[145][0][1][1] =
    0;rules[145][1][0][0] = 1;rules[145][1][0][1] = 0;rules[145][1][1][0] =
    0;rules[145][1][1][1] = 1;
rules[146][0][0][0] = 0;rules[146][0][0][1] = 1;rules[146][0][1][0] = 0;rules[146][0][1][1] =
    0;rules[146][1][0][0] = 1;rules[146][1][0][1] = 0;rules[146][1][1][0] =
    0;rules[146][1][1][1] = 1;
rules[147][0][0][0] = 1;rules[147][0][0][1] = 1;rules[147][0][1][0] = 0;rules[147][0][1][1] =
    0;rules[147][1][0][0] = 1;rules[147][1][0][1] = 0;rules[147][1][1][0] =
    0;rules[147][1][1][1] = 1;
rules[148][0][0][0] = 0;rules[148][0][0][1] = 0;rules[148][0][1][0] = 1;rules[148][0][1][1] =
    0;rules[148][1][0][0] = 1;rules[148][1][0][1] = 0;rules[148][1][1][0] =
    0;rules[148][1][1][1] = 1;
```

```
rules[149][0][0][0] = 1;rules[149][0][0][1] = 0;rules[149][0][1][0] = 1;rules[149][0][1][1] =
    0;rules[149][1][0][0] = 1;rules[149][1][0][1] = 0;rules[149][1][1][0] =
    0;rules[149][1][1][1] = 1;
rules[150][0][0][0] = 0;rules[150][0][0][1] = 1;rules[150][0][1][0] = 1;rules[150][0][1][1] =
    0;rules[150][1][0][0] = 1;rules[150][1][0][1] = 0;rules[150][1][1][0] =
    0;rules[150][1][1][1] = 1;
rules[151][0][0][0] = 1;rules[151][0][0][1] = 1;rules[151][0][1][0] = 1;rules[151][0][1][1] =
    0;rules[151][1][0][0] = 1;rules[151][1][0][1] = 0;rules[151][1][1][0] =
    0;rules[151][1][1][1] = 1;
rules[152][0][0][0] = 0;rules[152][0][0][1] = 0;rules[152][0][1][0] = 0;rules[152][0][1][1] =
    1;rules[152][1][0][0] = 1;rules[152][1][0][1] = 0;rules[152][1][1][0] =
    0;rules[152][1][1][1] = 1;
rules[153][0][0][0] = 1;rules[153][0][0][1] = 0;rules[153][0][1][0] = 0;rules[153][0][1][1] =
    1;rules[153][1][0][0] = 1;rules[153][1][0][1] = 0;rules[153][1][1][0] =
    0;rules[153][1][1][1] = 1;
rules[154][0][0][0] = 0;rules[154][0][0][1] = 1;rules[154][0][1][0] = 0;rules[154][0][1][1] =
    1;rules[154][1][0][0] = 1;rules[154][1][0][1] = 0;rules[154][1][1][0] =
    0;rules[154][1][1][1] = 1;
rules[155][0][0][0] = 1;rules[155][0][0][1] = 1;rules[155][0][1][0] = 0;rules[155][0][1][1] =
    1;rules[155][1][0][0] = 1;rules[155][1][0][1] = 0;rules[155][1][1][0] =
    0;rules[155][1][1][1] = 1;
rules[156][0][0][0] = 0;rules[156][0][0][1] = 0;rules[156][0][1][0] = 1;rules[156][0][1][1] =
    1;rules[156][1][0][0] = 1;rules[156][1][0][1] = 0;rules[156][1][1][0] =
    0;rules[156][1][1][1] = 1;
rules[157][0][0][0] = 1;rules[157][0][0][1] = 0;rules[157][0][1][0] = 1;rules[157][0][1][1] =
    1;rules[157][1][0][0] = 1;rules[157][1][0][1] = 0;rules[157][1][1][0] =
    0;rules[157][1][1][1] = 1;
rules[158][0][0][0] = 0;rules[158][0][0][1] = 1;rules[158][0][1][0] = 1;rules[158][0][1][1] =
    1;rules[158][1][0][0] = 1;rules[158][1][0][1] = 0;rules[158][1][1][0] =
    0;rules[158][1][1][1] = 1;
rules[159][0][0][0] = 1;rules[159][0][0][1] = 1;rules[159][0][1][0] = 1;rules[159][0][1][1] =
    1;rules[159][1][0][0] = 1;rules[159][1][0][1] = 0;rules[159][1][1][0] =
    0;rules[159][1][1][1] = 1;
rules[160][0][0][0] = 0;rules[160][0][0][1] = 0;rules[160][0][1][0] = 0;rules[160][0][1][1] =
    0;rules[160][1][0][0] = 0;rules[160][1][0][1] = 1;rules[160][1][1][0] =
    0;rules[160][1][1][1] = 1;
rules[161][0][0][0] = 1;rules[161][0][0][1] = 0;rules[161][0][1][0] = 0;rules[161][0][1][1] =
    0;rules[161][1][0][0] = 0;rules[161][1][0][1] = 1;rules[161][1][1][0] =
    0;rules[161][1][1][1] = 1;
rules[162][0][0][0] = 0;rules[162][0][0][1] = 1;rules[162][0][1][0] = 0;rules[162][0][1][1] =
    0;rules[162][1][0][0] = 0;rules[162][1][0][1] = 1;rules[162][1][1][0] =
    0;rules[162][1][1][1] = 1;
rules[163][0][0][0] = 1;rules[163][0][0][1] = 1;rules[163][0][1][0] = 0;rules[163][0][1][1] =
    0;rules[163][1][0][0] = 0;rules[163][1][0][1] = 1;rules[163][1][1][0] =
    0;rules[163][1][1][1] = 1;
rules[164][0][0][0] = 0;rules[164][0][0][1] = 0;rules[164][0][1][0] = 1;rules[164][0][1][1] =
    0;rules[164][1][0][0] = 0;rules[164][1][0][1] = 1;rules[164][1][1][0] =
    0;rules[164][1][1][1] = 1;
rules[165][0][0][0] = 1;rules[165][0][0][1] = 0;rules[165][0][1][0] = 1;rules[165][0][1][1] =
    0;rules[165][1][0][0] = 0;rules[165][1][0][1] = 1;rules[165][1][1][0] =
    0;rules[165][1][1][1] = 1;
rules[166][0][0][0] = 0;rules[166][0][0][1] = 1;rules[166][0][1][0] = 1;rules[166][0][1][1] =
    0;rules[166][1][0][0] = 0;rules[166][1][0][1] = 1;rules[166][1][1][0] =
    0;rules[166][1][1][1] = 1;
rules[167][0][0][0] = 1;rules[167][0][0][1] = 1;rules[167][0][1][0] = 1;rules[167][0][1][1] =
    0;rules[167][1][0][0] = 0;rules[167][1][0][1] = 1;rules[167][1][1][0] =
    0;rules[167][1][1][1] = 1;
rules[168][0][0][0] = 0;rules[168][0][0][1] = 0;rules[168][0][1][0] = 0;rules[168][0][1][1] =
    1;rules[168][1][0][0] = 0;rules[168][1][0][1] = 1;rules[168][1][1][0] =
    0;rules[168][1][1][1] = 1;
rules[169][0][0][0] = 1;rules[169][0][0][1] = 0;rules[169][0][1][0] = 0;rules[169][0][1][1] =
    1;rules[169][1][0][0] = 0;rules[169][1][0][1] = 1;rules[169][1][1][0] =
    0;rules[169][1][1][1] = 1;
rules[170][0][0][0] = 0;rules[170][0][0][1] = 1;rules[170][0][1][0] = 0;rules[170][0][1][1] =
    1;rules[170][1][0][0] = 0;rules[170][1][0][1] = 1;rules[170][1][1][0] =
    0;rules[170][1][1][1] = 1;
rules[171][0][0][0] = 1;rules[171][0][0][1] = 1;rules[171][0][1][0] = 0;rules[171][0][1][1] =
    1;rules[171][1][0][0] = 0;rules[171][1][0][1] = 1;rules[171][1][1][0] =
    0;rules[171][1][1][1] = 1;
rules[172][0][0][0] = 0;rules[172][0][0][1] = 0;rules[172][0][1][0] = 1;rules[172][0][1][1] =
    1;rules[172][1][0][0] = 0;rules[172][1][0][1] = 1;rules[172][1][1][0] =
    0;rules[172][1][1][1] = 1;
rules[173][0][0][0] = 1;rules[173][0][0][1] = 0;rules[173][0][1][0] = 1;rules[173][0][1][1] =
    1;rules[173][1][0][0] = 0;rules[173][1][0][1] = 1;rules[173][1][1][0] =
    0;rules[173][1][1][1] = 1;
rules[174][0][0][0] = 0;rules[174][0][0][1] = 1;rules[174][0][1][0] = 1;rules[174][0][1][1] =
    1;rules[174][1][0][0] = 0;rules[174][1][0][1] = 1;rules[174][1][1][0] =
    0;rules[174][1][1][1] = 1;
rules[175][0][0][0] = 1;rules[175][0][0][1] = 1;rules[175][0][1][0] = 1;rules[175][0][1][1] =
    1;rules[175][1][0][0] = 0;rules[175][1][0][1] = 1;rules[175][1][1][0] =
    0;rules[175][1][1][1] = 1;
rules[176][0][0][0] = 0;rules[176][0][0][1] = 0;rules[176][0][1][0] = 0;rules[176][0][1][1] =
    0;rules[176][1][0][0] = 1;rules[176][1][0][1] = 1;rules[176][1][1][0] =
    0;rules[176][1][1][1] = 1;
rules[177][0][0][0] = 1;rules[177][0][0][1] = 0;rules[177][0][1][0] = 0;rules[177][0][1][1] =
    0;rules[177][1][0][0] = 1;rules[177][1][0][1] = 1;rules[177][1][1][0] =
    0;rules[177][1][1][1] = 1;
rules[178][0][0][0] = 0;rules[178][0][0][1] = 1;rules[178][0][1][0] = 0;rules[178][0][1][1] =
    0;rules[178][1][0][0] = 1;rules[178][1][0][1] = 1;rules[178][1][1][0] =
    0;rules[178][1][1][1] = 1;
rules[179][0][0][0] = 1;rules[179][0][0][1] = 1;rules[179][0][1][0] = 0;rules[179][0][1][1] =
    0;rules[179][1][0][0] = 1;rules[179][1][0][1] = 1;rules[179][1][1][0] =
    0;rules[179][1][1][1] = 1;
rules[180][0][0][0] = 0;rules[180][0][0][1] = 0;rules[180][0][1][0] = 1;rules[180][0][1][1] =
    0;rules[180][1][0][0] = 1;rules[180][1][0][1] = 1;rules[180][1][1][0] =
    0;rules[180][1][1][1] = 1;
rules[181][0][0][0] = 1;rules[181][0][0][1] = 0;rules[181][0][1][0] = 1;rules[181][0][1][1] =
    0;rules[181][1][0][0] = 1;rules[181][1][0][1] = 1;rules[181][1][1][0] =
    0;rules[181][1][1][1] = 1;
rules[182][0][0][0] = 0;rules[182][0][0][1] = 1;rules[182][0][1][0] = 1;rules[182][0][1][1] =
    0;rules[182][1][0][0] = 1;rules[182][1][0][1] = 1;rules[182][1][1][0] =
    0;rules[182][1][1][1] = 1;
rules[183][0][0][0] = 1;rules[183][0][0][1] = 1;rules[183][0][1][0] = 1;rules[183][0][1][1] =
    0;rules[183][1][0][0] = 1;rules[183][1][0][1] = 1;rules[183][1][1][0] =
    0;rules[183][1][1][1] = 1;
rules[184][0][0][0] = 0;rules[184][0][0][1] = 0;rules[184][0][1][0] = 0;rules[184][0][1][1] =
    1;rules[184][1][0][0] = 1;rules[184][1][0][1] = 1;rules[184][1][1][0] =
    0;rules[184][1][1][1] = 1;
rules[185][0][0][0] = 1;rules[185][0][0][1] = 0;rules[185][0][1][0] = 0;rules[185][0][1][1] =
    1;rules[185][1][0][0] = 1;rules[185][1][0][1] = 1;rules[185][1][1][0] =
    0;rules[185][1][1][1] = 1;
rules[186][0][0][0] = 0;rules[186][0][0][1] = 1;rules[186][0][1][0] = 0;rules[186][0][1][1] =
    1;rules[186][1][0][0] = 1;rules[186][1][0][1] = 1;rules[186][1][1][0] =
    0;rules[186][1][1][1] = 1;
rules[187][0][0][0] = 1;rules[187][0][0][1] = 1;rules[187][0][1][0] = 0;rules[187][0][1][1] =
    1;rules[187][1][0][0] = 1;rules[187][1][0][1] = 1;rules[187][1][1][0] =
```

```
0;rules[187][1][1][1] = 1;
rules[188][0][0][0] = 0;rules[188][0][0][1] = 0;rules[188][0][1][0] = 1;rules[188][0][1][1] =
1;rules[188][1][0][0] = 1;rules[188][1][0][1] = 1;rules[188][1][1][0] =
0;rules[188][1][1][1] = 1;
rules[189][0][0][0] = 1;rules[189][0][0][1] = 0;rules[189][0][1][0] = 1;rules[189][0][1][1] =
1;rules[189][1][0][0] = 1;rules[189][1][0][1] = 1;rules[189][1][1][0] =
0;rules[189][1][1][1] = 1;
rules[190][0][0][0] = 0;rules[190][0][0][1] = 1;rules[190][0][1][0] = 1;rules[190][0][1][1] =
1;rules[190][1][0][0] = 1;rules[190][1][0][1] = 1;rules[190][1][1][0] =
0;rules[190][1][1][1] = 1;
rules[191][0][0][0] = 1;rules[191][0][0][1] = 1;rules[191][0][1][0] = 1;rules[191][0][1][1] =
1;rules[191][1][0][0] = 1;rules[191][1][0][1] = 1;rules[191][1][1][0] =
0;rules[191][1][1][1] = 1;
rules[192][0][0][0] = 0;rules[192][0][0][1] = 0;rules[192][0][1][0] = 0;rules[192][0][1][1] =
0;rules[192][1][0][0] = 0;rules[192][1][0][1] = 0;rules[192][1][1][0] =
1;rules[192][1][1][1] = 1;
rules[193][0][0][0] = 1;rules[193][0][0][1] = 0;rules[193][0][1][0] = 0;rules[193][0][1][1] =
0;rules[193][1][0][0] = 0;rules[193][1][0][1] = 0;rules[193][1][1][0] =
1;rules[193][1][1][1] = 1;
rules[194][0][0][0] = 0;rules[194][0][0][1] = 1;rules[194][0][1][0] = 0;rules[194][0][1][1] =
0;rules[194][1][0][0] = 0;rules[194][1][0][1] = 0;rules[194][1][1][0] =
1;rules[194][1][1][1] = 1;
rules[195][0][0][0] = 1;rules[195][0][0][1] = 1;rules[195][0][1][0] = 0;rules[195][0][1][1] =
0;rules[195][1][0][0] = 0;rules[195][1][0][1] = 0;rules[195][1][1][0] =
1;rules[195][1][1][1] = 1;
rules[196][0][0][0] = 0;rules[196][0][0][1] = 0;rules[196][0][1][0] = 1;rules[196][0][1][1] =
0;rules[196][1][0][0] = 0;rules[196][1][0][1] = 0;rules[196][1][1][0] =
1;rules[196][1][1][1] = 1;
rules[197][0][0][0] = 1;rules[197][0][0][1] = 0;rules[197][0][1][0] = 1;rules[197][0][1][1] =
0;rules[197][1][0][0] = 0;rules[197][1][0][1] = 0;rules[197][1][1][0] =
1;rules[197][1][1][1] = 1;
rules[198][0][0][0] = 0;rules[198][0][0][1] = 1;rules[198][0][1][0] = 1;rules[198][0][1][1] =
0;rules[198][1][0][0] = 0;rules[198][1][0][1] = 0;rules[198][1][1][0] =
1;rules[198][1][1][1] = 1;
rules[199][0][0][0] = 1;rules[199][0][0][1] = 1;rules[199][0][1][0] = 1;rules[199][0][1][1] =
0;rules[199][1][0][0] = 0;rules[199][1][0][1] = 0;rules[199][1][1][0] =
1;rules[199][1][1][1] = 1;
rules[200][0][0][0] = 0;rules[200][0][0][1] = 0;rules[200][0][1][0] = 0;rules[200][0][1][1] =
1;rules[200][1][0][0] = 0;rules[200][1][0][1] = 0;rules[200][1][1][0] =
1;rules[200][1][1][1] = 1;
rules[201][0][0][0] = 1;rules[201][0][0][1] = 0;rules[201][0][1][0] = 0;rules[201][0][1][1] =
1;rules[201][1][0][0] = 0;rules[201][1][0][1] = 0;rules[201][1][1][0] =
1;rules[201][1][1][1] = 1;
rules[202][0][0][0] = 0;rules[202][0][0][1] = 1;rules[202][0][1][0] = 0;rules[202][0][1][1] =
1;rules[202][1][0][0] = 0;rules[202][1][0][1] = 0;rules[202][1][1][0] =
1;rules[202][1][1][1] = 1;
rules[203][0][0][0] = 1;rules[203][0][0][1] = 1;rules[203][0][1][0] = 0;rules[203][0][1][1] =
1;rules[203][1][0][0] = 0;rules[203][1][0][1] = 0;rules[203][1][1][0] =
1;rules[203][1][1][1] = 1;
rules[204][0][0][0] = 0;rules[204][0][0][1] = 0;rules[204][0][1][0] = 1;rules[204][0][1][1] =
1;rules[204][1][0][0] = 0;rules[204][1][0][1] = 0;rules[204][1][1][0] =
1;rules[204][1][1][1] = 1;
rules[205][0][0][0] = 1;rules[205][0][0][1] = 0;rules[205][0][1][0] = 1;rules[205][0][1][1] =
1;rules[205][1][0][0] = 0;rules[205][1][0][1] = 0;rules[205][1][1][0] =
1;rules[205][1][1][1] = 1;
rules[206][0][0][0] = 0;rules[206][0][0][1] = 1;rules[206][0][1][0] = 1;rules[206][0][1][1] =
1;rules[206][1][0][0] = 0;rules[206][1][0][1] = 0;rules[206][1][1][0] =
1;rules[206][1][1][1] = 1;
rules[207][0][0][0] = 1;rules[207][0][0][1] = 1;rules[207][0][1][0] = 1;rules[207][0][1][1] =
1;rules[207][1][0][0] = 0;rules[207][1][0][1] = 0;rules[207][1][1][0] =
1;rules[207][1][1][1] = 1;
rules[208][0][0][0] = 0;rules[208][0][0][1] = 0;rules[208][0][1][0] = 0;rules[208][0][1][1] =
0;rules[208][1][0][0] = 1;rules[208][1][0][1] = 0;rules[208][1][1][0] =
1;rules[208][1][1][1] = 1;
rules[209][0][0][0] = 1;rules[209][0][0][1] = 0;rules[209][0][1][0] = 0;rules[209][0][1][1] =
0;rules[209][1][0][0] = 1;rules[209][1][0][1] = 0;rules[209][1][1][0] =
1;rules[209][1][1][1] = 1;
rules[210][0][0][0] = 0;rules[210][0][0][1] = 1;rules[210][0][1][0] = 0;rules[210][0][1][1] =
0;rules[210][1][0][0] = 1;rules[210][1][0][1] = 0;rules[210][1][1][0] =
1;rules[210][1][1][1] = 1;
rules[211][0][0][0] = 1;rules[211][0][0][1] = 1;rules[211][0][1][0] = 0;rules[211][0][1][1] =
0;rules[211][1][0][0] = 1;rules[211][1][0][1] = 0;rules[211][1][1][0] =
1;rules[211][1][1][1] = 1;
rules[212][0][0][0] = 0;rules[212][0][0][1] = 0;rules[212][0][1][0] = 1;rules[212][0][1][1] =
0;rules[212][1][0][0] = 1;rules[212][1][0][1] = 0;rules[212][1][1][0] =
1;rules[212][1][1][1] = 1;
rules[213][0][0][0] = 1;rules[213][0][0][1] = 0;rules[213][0][1][0] = 1;rules[213][0][1][1] =
0;rules[213][1][0][0] = 1;rules[213][1][0][1] = 0;rules[213][1][1][0] =
1;rules[213][1][1][1] = 1;
rules[214][0][0][0] = 0;rules[214][0][0][1] = 1;rules[214][0][1][0] = 1;rules[214][0][1][1] =
0;rules[214][1][0][0] = 1;rules[214][1][0][1] = 0;rules[214][1][1][0] =
1;rules[214][1][1][1] = 1;
rules[215][0][0][0] = 1;rules[215][0][0][1] = 1;rules[215][0][1][0] = 1;rules[215][0][1][1] =
0;rules[215][1][0][0] = 1;rules[215][1][0][1] = 0;rules[215][1][1][0] =
1;rules[215][1][1][1] = 1;
rules[216][0][0][0] = 0;rules[216][0][0][1] = 0;rules[216][0][1][0] = 0;rules[216][0][1][1] =
1;rules[216][1][0][0] = 1;rules[216][1][0][1] = 0;rules[216][1][1][0] =
1;rules[216][1][1][1] = 1;
rules[217][0][0][0] = 1;rules[217][0][0][1] = 0;rules[217][0][1][0] = 0;rules[217][0][1][1] =
1;rules[217][1][0][0] = 1;rules[217][1][0][1] = 0;rules[217][1][1][0] =
1;rules[217][1][1][1] = 1;
rules[218][0][0][0] = 0;rules[218][0][0][1] = 1;rules[218][0][1][0] = 0;rules[218][0][1][1] =
1;rules[218][1][0][0] = 1;rules[218][1][0][1] = 0;rules[218][1][1][0] =
1;rules[218][1][1][1] = 1;
rules[219][0][0][0] = 1;rules[219][0][0][1] = 1;rules[219][0][1][0] = 0;rules[219][0][1][1] =
1;rules[219][1][0][0] = 1;rules[219][1][0][1] = 0;rules[219][1][1][0] =
1;rules[219][1][1][1] = 1;
rules[220][0][0][0] = 0;rules[220][0][0][1] = 0;rules[220][0][1][0] = 1;rules[220][0][1][1] =
1;rules[220][1][0][0] = 1;rules[220][1][0][1] = 0;rules[220][1][1][0] =
1;rules[220][1][1][1] = 1;
rules[221][0][0][0] = 1;rules[221][0][0][1] = 0;rules[221][0][1][0] = 1;rules[221][0][1][1] =
1;rules[221][1][0][0] = 1;rules[221][1][0][1] = 0;rules[221][1][1][0] =
1;rules[221][1][1][1] = 1;
rules[222][0][0][0] = 0;rules[222][0][0][1] = 1;rules[222][0][1][0] = 1;rules[222][0][1][1] =
1;rules[222][1][0][0] = 1;rules[222][1][0][1] = 0;rules[222][1][1][0] =
1;rules[222][1][1][1] = 1;
rules[223][0][0][0] = 1;rules[223][0][0][1] = 1;rules[223][0][1][0] = 1;rules[223][0][1][1] =
1;rules[223][1][0][0] = 1;rules[223][1][0][1] = 0;rules[223][1][1][0] =
1;rules[223][1][1][1] = 1;
rules[224][0][0][0] = 0;rules[224][0][0][1] = 0;rules[224][0][1][0] = 0;rules[224][0][1][1] =
0;rules[224][1][0][0] = 0;rules[224][1][0][1] = 1;rules[224][1][1][0] =
1;rules[224][1][1][1] = 1;
rules[225][0][0][0] = 1;rules[225][0][0][1] = 0;rules[225][0][1][0] = 0;rules[225][0][1][1] =
0;rules[225][1][0][0] = 0;rules[225][1][0][1] = 1;rules[225][1][1][0] =
1;rules[225][1][1][1] = 1;
rules[226][0][0][0] = 0;rules[226][0][0][1] = 1;rules[226][0][1][0] = 0;rules[226][0][1][1] =
```

```
            0;rules[226][1][0][0] = 0;rules[226][1][0][1] = 1;rules[226][1][1][0] =
            1;rules[226][1][1][1] = 1;
        rules[227][0][0][0] = 1;rules[227][0][0][1] = 1;rules[227][0][1][0] = 0;rules[227][0][1][1] =
            0;rules[227][1][0][0] = 0;rules[227][1][0][1] = 1;rules[227][1][1][0] =
            1;rules[227][1][1][1] = 1;
        rules[228][0][0][0] = 0;rules[228][0][0][1] = 0;rules[228][0][1][0] = 1;rules[228][0][1][1] =
            0;rules[228][1][0][0] = 0;rules[228][1][0][1] = 1;rules[228][1][1][0] =
            1;rules[228][1][1][1] = 1;
        rules[229][0][0][0] = 1;rules[229][0][0][1] = 0;rules[229][0][1][0] = 1;rules[229][0][1][1] =
            0;rules[229][1][0][0] = 0;rules[229][1][0][1] = 1;rules[229][1][1][0] =
            1;rules[229][1][1][1] = 1;
        rules[230][0][0][0] = 0;rules[230][0][0][1] = 1;rules[230][0][1][0] = 1;rules[230][0][1][1] =
            0;rules[230][1][0][0] = 0;rules[230][1][0][1] = 1;rules[230][1][1][0] =
            1;rules[230][1][1][1] = 1;
        rules[231][0][0][0] = 1;rules[231][0][0][1] = 1;rules[231][0][1][0] = 1;rules[231][0][1][1] =
            0;rules[231][1][0][0] = 0;rules[231][1][0][1] = 1;rules[231][1][1][0] =
            1;rules[231][1][1][1] = 1;
        rules[232][0][0][0] = 0;rules[232][0][0][1] = 0;rules[232][0][1][0] = 0;rules[232][0][1][1] =
            1;rules[232][1][0][0] = 0;rules[232][1][0][1] = 1;rules[232][1][1][0] =
            1;rules[232][1][1][1] = 1;
        rules[233][0][0][0] = 1;rules[233][0][0][1] = 0;rules[233][0][1][0] = 0;rules[233][0][1][1] =
            1;rules[233][1][0][0] = 0;rules[233][1][0][1] = 1;rules[233][1][1][0] =
            1;rules[233][1][1][1] = 1;
        rules[234][0][0][0] = 0;rules[234][0][0][1] = 1;rules[234][0][1][0] = 0;rules[234][0][1][1] =
            1;rules[234][1][0][0] = 0;rules[234][1][0][1] = 1;rules[234][1][1][0] =
            1;rules[234][1][1][1] = 1;
        rules[235][0][0][0] = 1;rules[235][0][0][1] = 1;rules[235][0][1][0] = 0;rules[235][0][1][1] =
            1;rules[235][1][0][0] = 0;rules[235][1][0][1] = 1;rules[235][1][1][0] =
            1;rules[235][1][1][1] = 1;
        rules[236][0][0][0] = 0;rules[236][0][0][1] = 0;rules[236][0][1][0] = 1;rules[236][0][1][1] =
            1;rules[236][1][0][0] = 0;rules[236][1][0][1] = 1;rules[236][1][1][0] =
            1;rules[236][1][1][1] = 1;
        rules[237][0][0][0] = 1;rules[237][0][0][1] = 0;rules[237][0][1][0] = 1;rules[237][0][1][1] =
            1;rules[237][1][0][0] = 0;rules[237][1][0][1] = 1;rules[237][1][1][0] =
            1;rules[237][1][1][1] = 1;
        rules[238][0][0][0] = 0;rules[238][0][0][1] = 1;rules[238][0][1][0] = 1;rules[238][0][1][1] =
            1;rules[238][1][0][0] = 0;rules[238][1][0][1] = 1;rules[238][1][1][0] =
            1;rules[238][1][1][1] = 1;
        rules[239][0][0][0] = 1;rules[239][0][0][1] = 1;rules[239][0][1][0] = 1;rules[239][0][1][1] =
            1;rules[239][1][0][0] = 0;rules[239][1][0][1] = 1;rules[239][1][1][0] =
            1;rules[239][1][1][1] = 1;
        rules[240][0][0][0] = 0;rules[240][0][0][1] = 0;rules[240][0][1][0] = 0;rules[240][0][1][1] =
            0;rules[240][1][0][0] = 1;rules[240][1][0][1] = 1;rules[240][1][1][0] =
            1;rules[240][1][1][1] = 1;
        rules[241][0][0][0] = 1;rules[241][0][0][1] = 0;rules[241][0][1][0] = 0;rules[241][0][1][1] =
            0;rules[241][1][0][0] = 1;rules[241][1][0][1] = 1;rules[241][1][1][0] =
            1;rules[241][1][1][1] = 1;
        rules[242][0][0][0] = 0;rules[242][0][0][1] = 1;rules[242][0][1][0] = 0;rules[242][0][1][1] =
            0;rules[242][1][0][0] = 1;rules[242][1][0][1] = 1;rules[242][1][1][0] =
            1;rules[242][1][1][1] = 1;
        rules[243][0][0][0] = 1;rules[243][0][0][1] = 1;rules[243][0][1][0] = 0;rules[243][0][1][1] =
            0;rules[243][1][0][0] = 1;rules[243][1][0][1] = 1;rules[243][1][1][0] =
            1;rules[243][1][1][1] = 1;
        rules[244][0][0][0] = 0;rules[244][0][0][1] = 0;rules[244][0][1][0] = 1;rules[244][0][1][1] =
            0;rules[244][1][0][0] = 1;rules[244][1][0][1] = 1;rules[244][1][1][0] =
            1;rules[244][1][1][1] = 1;
        rules[245][0][0][0] = 1;rules[245][0][0][1] = 0;rules[245][0][1][0] = 1;rules[245][0][1][1] =
            0;rules[245][1][0][0] = 1;rules[245][1][0][1] = 1;rules[245][1][1][0] =
            1;rules[245][1][1][1] = 1;
        rules[246][0][0][0] = 0;rules[246][0][0][1] = 1;rules[246][0][1][0] = 1;rules[246][0][1][1] =
            0;rules[246][1][0][0] = 1;rules[246][1][0][1] = 1;rules[246][1][1][0] =
            1;rules[246][1][1][1] = 1;
        rules[247][0][0][0] = 1;rules[247][0][0][1] = 1;rules[247][0][1][0] = 1;rules[247][0][1][1] =
            0;rules[247][1][0][0] = 1;rules[247][1][0][1] = 1;rules[247][1][1][0] =
            1;rules[247][1][1][1] = 1;
        rules[248][0][0][0] = 0;rules[248][0][0][1] = 0;rules[248][0][1][0] = 0;rules[248][0][1][1] =
            1;rules[248][1][0][0] = 1;rules[248][1][0][1] = 1;rules[248][1][1][0] =
            1;rules[248][1][1][1] = 1;
        rules[249][0][0][0] = 1;rules[249][0][0][1] = 0;rules[249][0][1][0] = 0;rules[249][0][1][1] =
            1;rules[249][1][0][0] = 1;rules[249][1][0][1] = 1;rules[249][1][1][0] =
            1;rules[249][1][1][1] = 1;
        rules[250][0][0][0] = 0;rules[250][0][0][1] = 1;rules[250][0][1][0] = 0;rules[250][0][1][1] =
            1;rules[250][1][0][0] = 1;rules[250][1][0][1] = 1;rules[250][1][1][0] =
            1;rules[250][1][1][1] = 1;
        rules[251][0][0][0] = 1;rules[251][0][0][1] = 1;rules[251][0][1][0] = 0;rules[251][0][1][1] =
            1;rules[251][1][0][0] = 1;rules[251][1][0][1] = 1;rules[251][1][1][0] =
            1;rules[251][1][1][1] = 1;
        rules[252][0][0][0] = 0;rules[252][0][0][1] = 0;rules[252][0][1][0] = 1;rules[252][0][1][1] =
            1;rules[252][1][0][0] = 1;rules[252][1][0][1] = 1;rules[252][1][1][0] =
            1;rules[252][1][1][1] = 1;
        rules[253][0][0][0] = 1;rules[253][0][0][1] = 0;rules[253][0][1][0] = 1;rules[253][0][1][1] =
            1;rules[253][1][0][0] = 1;rules[253][1][0][1] = 1;rules[253][1][1][0] =
            1;rules[253][1][1][1] = 1;
        rules[254][0][0][0] = 0;rules[254][0][0][1] = 1;rules[254][0][1][0] = 1;rules[254][0][1][1] =
            1;rules[254][1][0][0] = 1;rules[254][1][0][1] = 1;rules[254][1][1][0] =
            1;rules[254][1][1][1] = 1;
        rules[255][0][0][0] = 1;rules[255][0][0][1] = 1;rules[255][0][1][0] = 1;rules[255][0][1][1] =
            1;rules[255][1][0][0] = 1;rules[255][1][0][1] = 1;rules[255][1][1][0] =
            1;rules[255][1][1][1] = 1;
        // Cargar un patron inicial si existe un archivo "pattern.txt"
        loadPatternIfExists();
}

void draw() {
        background(255);

        // Inicializar el tamanio de la cuadricula en funcion del numero de celdas por lado (filas y
            columnas)
        cols = int(cp5.getController("cols").getValue());
        rows = int(cp5.getController("rows").getValue());

        if (generateGrid) {
                generateGrid();
                generateGrid = false; // Evita regenerar la cuadricula en cada frame
                hideGenerateButton = true; // Oculta el boton "Generar" despues de generar la
                    cuadricula
        }

        if (showUI) {
                // Mostrar controles de usuario
                cp5.show();
        }

        // Dibujar la cuadricula en cada frame
        displayGrid();
```

```
        if (!hideGenerateButton) {
                // Mostrar boton de "Generar" si no se ha ocultado
                generateButton.display();
        }

        if (automaticMode && currentRow == rows - 2) {
                automaticMode = false;
                if (!visualizarAtractor) {
                        applyRule(); // Aplicar la regla si no estamos visualizando el ciclo atractor
                }
                lastTime = millis();
                currentRow++;
        }

        if (automaticMode) {
                applyRule();
                displayGrid();
                lastTime = millis(); // Actualizar el ultimo tiempo
                currentRow++;
        }

        if (mostrarCicloAtractor) {
                generarCicloAtractor(); // Generar y mostrar el ciclo atractor
        }

        fillRandomButton.display();
        saveConfigButton.display();
        applyRuleButton.display();
        loadPatternButton.display();


}

void generateGrid() {
        // Ajusta el tamanio de la cuadricula en funcion del numero de celdas por lado (filas y
             columnas)
        generation = 1;
        cols = int(cp5.getController("cols").getValue());
        rows = int(cp5.getController("rows").getValue());
        grid = new color[cols][rows]; // Crear una nueva cuadricula con el tamanio actualizado
        nextGeneration = new int[cols][rows]; // Crear una nueva matriz para la siguiente generacion
        currentRow = 0; // Restablecer el numero de fila actual cuando se genera una nueva cuadricula
        // Restablecer los contadores
        resetCounters();

        for (int i = 0; i < cols; i++) {
                for (int j = 0; j < rows; j++) {
                        grid[i][j] = colorMuerta; // Inicializar todas las celdas como muertas
                }
        }
        // Ocultar selectores de color despues de generar la cuadricula
        colorPickerViva.hide();
        colorPickerMuerta.hide();
}

void displayGrid() {
        int cellSize = int(cp5.getController("cellSize").getValue()); // Obtener el tamanio de celda
             actual

        noStroke(); // Quitar el contorno de los rectangulos

        for (int i = 0; i < cols; i++) {
                for (int j = 0; j < rows; j++) {
                        float x = i * cellSize;
                        float y = j * cellSize;
                        fill(grid[i][j]);
                        rect(x, y, cellSize, cellSize);
                }
        }
}

void mousePressed() {
        int i = int(mouseX / int(cp5.getController("cellSize").getValue()));
        int j = int(mouseY / int(cp5.getController("cellSize").getValue()));

        if (i >= 0 && i < cols && j >= 0 && j < rows) {
                // Cambiar el color de la celda al hacer clic
                if (mouseButton == LEFT) {
                        grid[i][j] = colorPickerViva.getColorValue(); // Celda viva
                } else if (mouseButton == RIGHT) {
                        grid[i][j] = colorPickerMuerta.getColorValue(); // Celda muerta
                }
        }

        // Manejar clics en el boton y los selectores de color
        generateButton.mousePressed();
        fillRandomButton.mousePressed();
        saveConfigButton.mousePressed();
        applyRuleButton.mousePressed();
        loadPatternButton.mousePressed();
}

void loadPatternIfExists() {
        File file = new File("carga.txt"); // Nombre del archivo de patron
        if (file.exists()) {
                loadPattern();
        }
}

void loadPattern() {
        File file = new File("carga.txt"); // Nombre del archivo de patron
        BufferedReader reader;

        try {
                reader = new BufferedReader(new FileReader(file));
                String line = reader.readLine();
                if (line != null) {
                        String[] values = line.split(",");
                        if (values.length == cols) {
                                // Asegurate de que la linea del archivo tenga el mismo numero de
                                     celdas que la cuadricula actual
                                for (int i = 0; i < cols; i++) {
                                        if (values[i].equals("0")) {
                                                grid[i][0] = colorMuerta;
                                        } else if (values[i].equals("1")) {
                                                grid[i][0] = colorViva;
```

```
                                        } else {
                                                println("Error: El archivo contiene valores no
                                                    validos.");
                                                return;
                                        }
                                }
                                println("Patron cargado correctamente.");
                        } else {
                                println("Error: El archivo no tiene el mismo numero de celdas que la
                                    cuadricula.");
                        }
                } else {
                        println("Error: El archivo esta vacio.");
                }
                reader.close();
        } catch (IOException e) {
                println("Error al cargar el archivo: " + e.getMessage());
        }
}

class Button {
        float x, y, w, h;
        String label;

        Button(float x, float y, float w, float h, String label) {
                this.x = x;
                this.y = y;
                this.w = w;
                this.h = h;
                this.label = label;
        }

        void display() {
                fill(200);
                rect(x, y, w, h);
                fill(0);
                textSize(12);
                textAlign(CENTER, CENTER);
                text(label, x + w / 2, y + h / 2);
        }

        boolean isMouseOver() {
                return mouseX >= x && mouseX <= x + w && mouseY >= y && mouseY <= y + h;
        }

        void mousePressed() {
                if (isMouseOver() && label.equals("Generar")) {
                        generateGrid = true;
                        hideGenerateButton = true; // Oculta el boton despues de hacer clic
                } else if (isMouseOver() && label.equals("Llenar Aleatoriamente")) {
                        fillRandomLine();
                } else if (isMouseOver() && label.equals("Guardar Configuracion")) {
                        saveConfiguration();
                } else if (isMouseOver() && label.equals("Aplicar Regla")) {
                        applyRule();
                } else if (isMouseOver() && label.equals("Cargar Patron")) {
                        loadPattern();
                }
        }
}

void fillRandomLine() {
        color colorVivaActual = colorPickerViva.getColorValue(); // Obtener el color actual de celda
            viva
        color colorMuertaActual = colorPickerMuerta.getColorValue(); // Obtener el color actual de
            celda muerta

        for (int i = 0; i < cols; i++) {
                float randomValue = random(1); // Generar un valor aleatorio entre 0 y 1
                if (randomValue <= percentageOnes) { // Comprobar si el valor esta dentro del
                    porcentaje de unos deseado
                        grid[i][0] = colorVivaActual; // Celda viva
                } else {
                        grid[i][0] = colorMuertaActual; // Celda muerta
                }
        }
}

void saveConfiguration() {
        color colorVivaActual = colorPickerViva.getColorValue(); // Obtener el color actual de celda
            viva
        color colorMuertaActual = colorPickerMuerta.getColorValue(); // Obtener el color actual de
            celda muerta
        try {
                PrintWriter writer = new PrintWriter("config.txt");
                for (int j = 0; j < rows; j++) {
                        for (int i = 0; i < cols; i++) {
                                if (i == 0) {
                                        writer.print(grid[i][j] == colorMuertaActual ? "0" : "1");
                                } else {
                                        writer.print("," + (grid[i][j] == colorMuertaActual ? "0" :
                                            "1"));
                                }
                        }
                        writer.println(); // Nueva linea para la siguiente fila
                }
                writer.close();
                println("Configuracion guardada en 'config.txt'");
        } catch (Exception e) {
                println("Error al guardar la configuracion: " + e.getMessage());
        }
}

void applyRule() {
        color colorVivaActual = colorPickerViva.getColorValue(); // Obtener el color actual de celda
            viva
        color colorMuertaActual = colorPickerMuerta.getColorValue(); // Obtener el color actual de
            celda muerta

        float[] probabilities = new float[8];
        float entropy = 0.0;
        String[] subStrings = {"000", "001", "010", "011", "100", "101", "110", "111"};

        // Copiar el estado actual a la proxima generacion
        for (int i = 0; i < cols; i++) {
                for (int j = 0; j < rows; j++) {
```

```
                        nextGeneration[i][j] = grid[i][j];
                }
        }

        // Contar unos y ceros en la generacion actual
        for (int i = 0; i < cols; i++) {
                if (grid[i][generation - 1] == colorVivaActual) {
                        unosGeneracion++;
                } else {
                        cerosGeneracion++;
                }
        }

        for (int i = 0; i < subStrings.length; i++) {
                String subString = subStrings[i];
                for (int z = 0; z < cols; z++) {
                        String cellState = (grid[z][(generation - 1) % cols] == colorVivaActual) ?
                             "1" : "0";
                        String cellStateRight = (grid[(z + 1) % cols][(generation - 1) % cols] ==
                             colorVivaActual) ? "1" : "0";
                        String cellStateLeft = (grid[(z - 1 + cols) % cols][(generation - 1) % cols]
                            == colorVivaActual) ? "1" : "0";
                        String neighborhood = cellStateLeft + cellState + cellStateRight;
                        if (neighborhood.equals(subString)) {
                                subStringCounts[i]++;
                        }
                }
        }

        // Calcular probabilidades
        for (int i = 0; i < 8; i++) {
                probabilities[i] = subStringCounts[i] / (float) (rows);

        }

        // Calcular la entropia de Shannon
        for (int i = 0; i < 8; i++) {
                if (probabilities[i] > 0) {
                        entropy -= probabilities[i] * log(probabilities[i]) / log(2);

                }
        }

        // Guardar el resultado en un archivo de texto
        writer2.println(generation + "\t" + entropy);
        writer2.flush();




        // Calcular densidad y guardar en archivo
        calculateDensity();

        //Calcular media de unos y guarda en archivo
        calculateMedia();

        calculateVar();


        // Restablecer los contadores para la siguiente generacion
        resetCounters();

        // Obtener la regla seleccionada desde el DropdownList
        int selectedRule = int(cp5.getController("Rule-Selector").getValue());

        // Aplicar la regla seleccionada para calcular la siguiente generacion
        for (int i = 0; i < cols; i++) {
                int left = (i - 1 + cols) % cols; // Usar modulo para obtener el vecino izquierdo
                int center = i;
                int right = (i + 1) % cols; // Usar modulo para obtener el vecino derecho

                int leftState = grid[left][generation - 1] == colorVivaActual ? 1 : 0;
                int centerState = grid[center][generation - 1] == colorVivaActual ? 1 : 0;
                int rightState = grid[right][generation - 1] == colorVivaActual ? 1 : 0;

                // Obtener el resultado de la regla actual desde la matriz de reglas
                int nextState = rules[selectedRule][leftState][centerState][rightState];

                // Actualizar la proxima generacion
                nextGeneration[center][generation] = (nextState == 1) ? colorVivaActual :
                     colorMuertaActual;
        }



        // Copiar la proxima generacion de nuevo a la cuadricula actual
        for (int i = 0; i < cols; i++) {
                for (int j = 0; j < rows; j++) {
                        grid[i][j] = nextGeneration[i][j];

                }
        }

        // Cambiar la generacion actual a la siguiente generacion
        generation++;
}

void controlEvent(ControlEvent event) {
        if (event.isController()) {
                String eventName = event.getController().getName();
                if (eventName.equals("Guardar-Ultima-Fila")) {
                        saveLastRow();
                } else if (eventName.equals("Automatico")) {
                        automaticMode = !automaticMode;
                        if (automaticMode) {
                                applyRule();
                                lastTime = millis();
                        }
                } else if (eventName.equals("2%")) {
                        percentageOnes = 0.02;
                } else if (eventName.equals("50%")) {
                        percentageOnes = 0.50;
                } else if (eventName.equals("75%")) {
                        percentageOnes = 0.75;
                } else if (eventName.equals("95%")) {
```

```
                        percentageOnes = 0.95;
                } else if (eventName.equals("Visualizar Ciclo Atractor")) {
                        mostrarCicloAtractor = !mostrarCicloAtractor;
                }
        }
}

void saveLastRow() {
        color colorVivaActual = colorPickerViva.getColorValue(); // Obtener el color actual de celda
             viva
        color colorMuertaActual = colorPickerMuerta.getColorValue(); // Obtener el color actual de
             celda muerta
        try {
                PrintWriter writer = new PrintWriter("last_row.txt");
                for (int i = 0; i < cols; i++) {
                        if (i == 0) {
                                writer.print(grid[i][rows - 1] == colorMuertaActual ? "0" : "1");
                        } else {
                                writer.print("," + (grid[i][rows - 1] == colorMuertaActual ? "0" :
                                     "1"));
                        }
                }
                writer.close();
                println("Ultima fila guardada en 'last_row.txt'");
        } catch (Exception e) {
                println("Error al guardar la ultima fila:" + e.getMessage());
        }
}

void resetCounters() {
        unosGeneracion = 0;
        cerosGeneracion = 0;
        for (int i = 0; i < 8; i++) {
                subStringCounts[i] = 0;
        }
}

void calculateDensity() {
        float density = unosGeneracion / (float)cerosGeneracion; // Calcular densidad
        densidades.add(density); // Agregar densidad a la lista
        generaciones.add(generation); // Agregar numero de generacion a la lista
        writer.println(generation + "\t" + density);
        writer.flush();
}

void calculateMedia() {
        float media = unosGeneracion / (float) (cerosGeneracion + unosGeneracion); // Calcular
             densidad
        medias.add(media); // Agregar densidad a la lista
        generaciones.add(generation); // Agregar numero de generacion a la lista
        writer1.println(generation + "\t" + media);
        writer1.flush();
}

void calculateVar() {

        color colorMuertaActual = colorPickerMuerta.getColorValue(); // Obtener el color actual de
             celda muerta
        try {
                PrintWriter varWriter = createWriter("VARIANZA.txt");


                for (int j = 0; j < rows; j++) {
                        for (int i = 0; i < cols; i++) {
                                if (i == 0) {
                                        varWriter.print(grid[i][j] == colorMuertaActual ? "0" : "1");
                                } else {
                                        varWriter.print((grid[i][j] == colorMuertaActual ? "0" :
                                             "1"));
                                }
                        }
                        varWriter.println(); // Nueva linea para la siguiente fila
                }

                varWriter.flush();
                varWriter.close();
        } catch (Exception e) {
                println("Error al guardar las cadenas:" + e.getMessage());
        }
}


void exit() {
        // Cerrar el archivo "DENSIDAD.txt" al finalizar el programa
        writer.close();
        super.exit();
}

void visualizarCicloAtractor() {
        int generacionActual = generation - 1;


        PGraphics graphCanvas = createGraphics(width, height);
        graphCanvas.beginDraw();
        graphCanvas.background(255);


        int nodeRadius = 10;
        int xOffset = 50;
        int yOffset = 50;


        for (int i = 0; i < generacionActual; i++) {
                for (int j = 0; j < rows; j++) {
                        if (grid[i][j] == colorViva) {
                                int x = i * nodeRadius + xOffset;
                                int y = j * nodeRadius + yOffset;
                                graphCanvas.ellipse(x, y, nodeRadius, nodeRadius);

                                // Dibuja conexiones entre nodos
                                if (i < generacionActual - 1) {
                                        for (int k = -1; k <= 1; k++) {
                                                int nextX = (i + 1) * nodeRadius + xOffset;
                                                int nextY = (j + k) * nodeRadius + yOffset;
                                                if (grid[i + 1][j] == colorViva) {
                                                        graphCanvas.line(x, y, nextX, nextY);
```

```
                                                }
                                        }
                                }
                        }
                }
        }

        graphCanvas.endDraw();
        image(graphCanvas, 0, 0);
}

void generarCicloAtractor() {
        int generacionActual = generation - 1;


        graphCanvas = createGraphics(width, height);
        graphCanvas.beginDraw();
        graphCanvas.background(255);


        int nodeRadius = 10;
        int xOffset = 50;
        int yOffset = 50;


        for (int i = 0; i < generacionActual; i++) {
                for (int j = 0; j < rows; j++) {
                        if (grid[i][j] == colorViva) {
                                int x = i * nodeRadius + xOffset;
                                int y = j * nodeRadius + yOffset;
                                graphCanvas.ellipse(x, y, nodeRadius, nodeRadius);


                                if (i < generacionActual - 1) {
                                        for (int k = -1; k <= 1; k++) {
                                                int nextX = (i + 1) * nodeRadius + xOffset;
                                                int nextY = (j + k) * nodeRadius + yOffset;
                                                if (grid[i + 1][j] == colorViva) {
                                                        graphCanvas.line(x, y, nextX, nextY);
                                                }
                                        }
                                }
                        }
                }
        }

        graphCanvas.endDraw();
        image(graphCanvas, 0, 0);
}
```

4 Código Fuente de MATLAB (Densidad y Log10)

```
data = load('C:\Users\Gael-Hernandez-Solis\Desktop\PROYECTO\DENSIDAD.txt');
min_value = min(data(:, 2));
min_value = abs(min_value) + 1;

figure;

semilogy(data(:, 1), data(:, 2), 'b.-', 'LineWidth', 1.5);
hold on;

semilogy(data(:, 1), log10(data(:, 2) + min_value), 'r.-', 'LineWidth', 1.5);

grid on;

xlabel('Generaciones');
ylabel('Densidades');
title('Grafico-de-Densidades-(Azul:-Original,-Rojo:-Log10)');

legend('Original', 'Log10', 'Location', 'Best');
```

5 Código Fuente de MATLAB (Media)

```
data = load('C:\Users\Gael-Hernandez-Solis\Desktop\PROYECTO\MEDIA.txt');

generaciones = data(:, 1);
mediaUnos = data(:, 2);

figure;

plot(generaciones, mediaUnos, 'b.-', 'LineWidth', 1.5, 'MarkerSize', 10);

grid on;

xlabel('Generaciones');
ylabel('Media-de-unos');
title('Media-de-unos');

set(gca, 'FontName', 'Arial', 'FontSize', 12, 'FontWeight', 'bold');
set(gcf, 'Color', 'w');
legend('Media-de-unos', 'Location', 'Best');
```

6 Código Fuente de MATLAB (Varianza)

```
nombreArchivo = 'C:\Users\Gael-Hernandez-Solis\Desktop\PROYECTO\VARIANZA.txt';

fid = fopen(nombreArchivo, 'r');
if fid == -1
error('No se pudo abrir el archivo.');
end

cadenas = {};
linea = fgetl(fid);
while ischar(linea)
cadenas{end+1} = linea;
linea = fgetl(fid);
end

fclose(fid);

generaciones = 1:length(cadenas);

for i = 1:length(cadenas)
varianzas(i) = var(cadenas{i});
end

figure;
plot(generaciones, varianzas, 'o-');
xlabel('Generacion');
ylabel('Varianza');
title('Varianzas de las cadenas por generacion');
grid on;
```

7 Código Fuente de MATLAB (Entropía de Shannon)

```
data = load('C:\Users\Gael-Hernandez-Solis\Desktop\PROYECTO\ENTROPY.txt');

generaciones = data(:, 1);
entropiaS = data(:, 2);

figure;

plot(generaciones, entropiaS, 'r.-', 'LineWidth', 1.5, 'MarkerSize', 10);

grid on;

xlabel('Generaciones');
ylabel('Entropia');
title('Entropia de Shannon por generacion');


set(gca, 'FontName', 'Arial', 'FontSize', 12, 'FontWeight', 'bold');
set(gcf, 'Color', 'w');
legend('Entrop\'{i}a', 'Location', 'Best');
```

8 Captura de Pantalla del simulador

Figure 45: Simulador completo

9 Conclusiones

Los autómatas celulares son un campo fascinante de la teoría de sistemas dinámicos y la informática teórica que ha estado en constante evolución desde su introducción en la década de 1940 por John von Neumann y Stanislaw Ulam. Estos modelos, que se basan en células dispuestas en una cuadrícula, cada una de las cuales puede estar en uno de varios estados posibles, se han convertido en herramientas esenciales para comprender y simular una amplia gama de fenómenos complejos. Su valor radica en su capacidad para capturar procesos emergentes a partir de reglas simples y en la exploración de ciclos atractivos que son fundamentales para la comprensión de la complejidad propia de sistemas dinámicos aparentemente caóticos.

Una de las razones por las cuales es fundamental estudiar los autómatas celulares es su capacidad para modelar una gran diversidad de fenómenos de la vida real y fenómenos físicos. Los autómatas celulares se utilizan para investigar problemas en biología, física, ecología, química, informática y muchas otras disciplinas. Por ejemplo, los CA pueden modelar la propagación de incendios en bosques, el crecimiento de patrones en cultivos celulares y el comportamiento de fluidos en medios porosos. La simplicidad de las reglas de evolución de los autómatas celulares permite una representación efectiva de procesos complejos que de otra manera serían difíciles de analizar.

En cuanto a su clasificación, existen 256 reglas de autómatas celulares elementales, numeradas del 0 al 255. Estas reglas se clasifican en cuatro categorías principales según su comportamiento: reglas periódicas, reglas caóticas, reglas complejas y reglas homogéneas. Las reglas periódicas generan patrones que se repiten en ciclos atractivos, mientras que las reglas caóticas crean patrones impredecibles y cambiantes. Las reglas complejas generan patrones más estructurados y ricos en contenido, y las reglas homogéneas llevan a una evolución uniforme en la que todas las celdas adoptan eventualmente el mismo estado. Estas categorías demuestran la diversidad de comportamientos que pueden surgir de reglas de evolución aparentemente simples.

Las aplicaciones de los autómatas celulares son igualmente diversas. Se utilizan en el procesamiento de imágenes para crear efectos visuales y algoritmos de filtrado, en criptografía para generar secuencias aleatorias y en la modelización de sistemas complejos para simular fenómenos naturales. En la biología, los CA se aplican para modelar el crecimiento de poblaciones y la evolución de especies, y en la física, se usan para analizar fenómenos de transporte y difusión.

La programación de este simulador y sus herramientas extra sí fue desafío debido a la necesidad de codificar las reglas de evolución y gestionar el estado de un gran número de celdas en la cuadrícula. Sin embargo, a medida que la computación se ha vuelto más accesible y potente, la implementación de simuladores de autómatas celulares se ha vuelto más factible. Aunque se requiere de un entendimiento sólido de las reglas y principios subyacentes, así como habilidades de programación, la programación de este simuladores fue una tarea gratificante. Me permitió explorar y visualizar las propiedades emergentes de estos sistemas dinámicos de una manera interactiva.

Los autómatas celulares son un poderoso recordatorio de que la simplicidad en las reglas puede conducir a la emergencia de una riqueza inesperada en la estructura y el comportamiento.

10 Referencias

References

[1] Campos de Atracción. (s. f.). `https://delta.cs.cinvestav.mx/~mcintosh/oldweb/s1997/todos/node16.html`

[2] Sfairopoulos, K. (2023, 14 septiembre). Cellular automata in D dimensions and ground states of spin models in $(D+1)$ dimensions. *arXiv.org*. `https://arxiv.org/abs/2309.08059`

[3] Search — ARXIV e-print Repository. (s. f.). `https://arxiv.org/search/?query=celular+rules&searchtype=all&source=header`

[4] Search — ARXIV e-print Repository. (s. f.-b). `https://arxiv.org/search/?query=celular+automata&searchtype=all&source=header`

11 Anexo

Rule	Size of MMS	Equivalent rules			Rule	Size of MMS	Equivalent rules		
0	0	255			56	3	98	185	227
1	3	127			57	3	99		
2	3	16	191	247	58	3	114	163	177
3	2	17	63	119	60	2	102	153	195
4	3	223			62	3	118	131	145
5	2	95			72	3	237		
6	3	20	159	215	73	3	109		
7	3	21	31	87	74	3	88	173	229
8	3	64	239	253	76	3	205		
9	3	65	111	125	77	3			
10	2	80	175	245	78	3	92	141	197
11	3	47	81	117	90	2	165		
12	2	68	207	221	94	3	133		
13	3	69	79	93	104	3	233		
14	3	84	143	213	105	3			
15	1	85			106	3	120	169	225
18	3	183			108	3	201		
19	3	55			110	3	124	137	193
22	3	151			122	3	161		
23	3				126	3	129		
24	3	66	189	231	128	3	254		
25	3	61	67	103	130	3	144	190	246
26	3	82	167	181	132	3	222		
27	3	39	53	83	134	3	148	158	214
28	3	70	157	199	136	2	192	238	252
29	3	71			138	3	174	208	244
30	3	86	135	149	140	3	196	206	220
32	3	251			142	3	212		
33	3	123			146	3	182		
34	2	48	187	243	150	3			
35	3	49	59	115	152	3	188	194	230
36	3	219			154	3	166	180	210
37	3	91			156	3	198		
38	3	52	155	211	160	2	250		
40	3	96	235	249	162	3	176	186	242
41	3	97	107	121	164	3	218		
42	3	112	171	241	168	3	224	234	248
43	3	113			170	1	240		
44	3	100	203	217	172	3	202	216	228
45	3	75	89	101	178	3			
46	3	116	139	209	184	3	226		
50	3	179			200	3	236		
51	1				204	1			
54	3	147			232	3			

Figure 46: Reglas equivalentes

- **Class I**: uniform behavior.
- **Class II**: periodic behavior.
- **Class III**: chaotic behavior.
- **Class IV**: complex behavior.

	Rules
classes	
Class I	0 8 32 40 128 136 160 168
Class II	1 2 3 4 5 6 7 9 10 11 12 13 14 15 19 23 24 25 26 27 28 29 33 34 35 36 37 38 42 43 44 46 50 51 56 57 58 62 72 73 74 76 77 78 94 104 108 130 132 134 138 140 142 152 154 156 162 164 170 172 178 184 200 204 232
Class III	18 22 30 45 60 90 105 122 126 146 150
Class IV	41 54 106 110

Figure 47: Clasificación por comportamiento de las reglas