Proyecto Semestral Autómata Celular

Catalina Bustos

Gonzalo Paillacan

***Abstract*— The objective of this project is to provide a brief explanation of the concept of a cellular automaton and its elements, highlighting the behavior of three cellular automata..**

***Resumen*— El objetivo de este proyecto es entregar una breve explicación del concepto de un autómata celular y sus elementos destacando el comportamiento tres autómatas celulares.**

I. INTRODUCCIÓN

La humanidad siempre se ha tentado a explorar lo desconocido, a entender cómo funciona el mundo y el porque de las cosas, tratando de dar soluciones a todas las dudas que se nos presentan.

Pero no es tan sencillo encontrar la respuesta indicada, algunos problemas tienden a ser muy complejos, tales como la propagación de enfermedades, desastres meteorológicos, etc. Los Autómatas Celulares nos ayudan a responder algunas de estas interrogantes, ya que nos permite poder simular variados sistemas del universo, haciendo de estas herramientas muy útiles para conocer y comprender el mundo real.

II. AUTÓMATA CELULAR Y SUS ELEMENTOS

Los autómatas celulares surgen con el trabajo de John von Neumann, matemático húngaro-estadounidense, quien en un intento de automatizar sistemas planteo la posibilidad de crear una máquina que fuera capaz de auto-reproducirse.
Así John Von Neumann logro inventar un sistema con esas características el cual se denominó Autómata Celular, la idea de la auto-reproducción inicialmente fue interpretada como un conjunto de células que crecían, reproducían y morían a través del tiempo.
De esta similitud con el crecimiento de las células se le debe su nombre.

Un Autómata Celular es un modelo matemático para un sistema dinámico que consiste en una serie de celdas o células que adquieren distintos valores dependiendo de su estado actual y de los estados de sus vecinos inmediatos.

Estos conjuntos de células logran una evolución según una determinada expresión matemática, la cual se conoce como "Regla De Transición Local".

Elementos de los autómatas celulares

Los autómatas celulares se componen de un espacio discreto, pasos de tiempo discretos, una condición inicial, condiciones de frontera y reglas. Una condición inicial es el estado (o valores) de cada una de las celdas en el tiempo=0. La evolución del sistema dependerá de esta condición inicial. Las condiciones de frontera se refieren a qué ocurre en las celdas que tocan los bordes del espacio.

Los elementos básicos son:

Arreglo Regular: Ya sea un plano de 2 dimensiones o un espacio n-dimensional, este es el espacio de evoluciones, y cada división homogénea de arreglo es llamada célula.

Conjunto de Estados. Es finito y cada elemento o célula del arreglo toma un valor de este conjunto de estados. También se denomina alfabeto. Puede ser expresado en valores o colores.

Configuración Inicial. Consiste en asignar un estado a cada una de las células del espacio de evolución inicial del sistema Vecindades. Define el conjunto contiguo de células y posición relativa respecto a cada una de ellas. A cada vecindad diferente corresponde un elemento del conjunto de estados

Función Local. Es la regla de evolución que determina el comportamiento del AC. Se conforma de una célula central y sus vecindades. Define como debe cambiar de estado cada célula dependiendo de los estados anteriores de sus vecindades. Puede ser una expresión algebraica

Adicionalmente para poder entender mejor su representación visual, se requiere mencionar los tipos de limites o fronteras, del plano en el cual se desarrolla, en los cuales se clasifica:

• Fronteras abiertas: se establece que toda la célula fuera del espacio del autómata toma un valor fijo.

• Frontera reflectora: las células fuera del espacio del autómata toman los valores dentro de este como si se tratara de un espejo.

• Frontera periódica o circular: una celda que está en la frontera interacciona con sus vecinos inmediatos y con las celdas que están en el extremo opuesto del autómata, como si fuera en círculos.

• Sin frontera: el autómata no tiene límites es infinito

III. MARCO MATEMATICO

"*Los Autómatas Celulares unidimensionales son los más simples.*
Cada fila de celdas es dependiente de la fila anterior. Por ejemplo, una celda puede ser dependiente en los estados de las tres celdas sobre él (inmediatamente superior, superior-izquierdo, superior-derecho).

Los estados pueden comprender una variedad de valores intermedios, pero por el momento nosotros usaremos un simple "vivo" o "muerto" (binario).

Imagina una tabla de reglas dónde cada regla es especificada según las tres celdas superiores:
{(x-1, y-1), (x, y-1), (x+1, y-1)}.

Un ejemplo de tabla de reglas podría ser:

Tabla de reglas
(x-1, y-1) (x, y-1) (x+1, y-1) =

0	0	0	0
1	0	0	1
0	1	0	1
0	0	1	1
1	1	0	1
0	1	1	1
1	0	1	1
1	1	1	0

…" [1]

"*La configuración global de un autómata celular en el tiempo t, se especifica por los*
valores de sus N sitios, y puede representarse por un polinomio característico,

[Ec. 1] $$A^{(t)}(x) = \sum_{i=0}^{N-1} a_i(t)\, x_i$$

donde el valor del sitio i en el tiempo t está representado por el valor del coeficiente $a_i(t)$.

Los valores de los sitios evolucionan en pasos discretos de tiempo según reglas determinísticas. Por lo tanto, la evolución depende de los valores tanto de los k elementos como del número r de vecinos que la regla tome en cuenta. El número de reglas distintas en función de k y r está dado por:

$$k^{k^{2r+1}}$$ …" [2]

IV. APLICACIONES

1. <u>*Bioinformática*</u>

"*La Bioinformática consiste en analizar, comprender y predecir procesos biológicos con la ayuda de herramientas computacionales. Puede ser vista como la disciplina que unas dos ciencias: la Biología y la Computación…"* [3]

En la Biología, los sistemas son tan complejos que no pueden resolver con modelos lineales ni Ecuaciones Diferenciales Ordinarias. Normalmente, estos sistemas se modelan con Ecuaciones Diferenciales en Derivadas Parciales, pero este método requiere una computadora con grandes cantidades de memoria.

Aquí es cuando la fusión de la Biología con la Informática se vuelve importante, pues la Bioinformática maneja y analiza datos biológicos almacenados como secuencias y aplica métodos computacionales para manipular esta información.

Alan Turing en sus escritos sobre la morfogénesis y la filotaxis, son ejemplos de patrones resultantes en un proceso auto-organizativo en la naturaleza, al igual que la idea de la auto-reproducción de los Autómatas Celulares.

Dependiendo de la naturaleza compleja del sistema y de la posibilidad de identificar estados locales y reglas generales de evolución, se podrían simular los fenómenos naturales por medio de un Autómata Celular.

Algunos de estos pueden ser: propagación de virus, epidemias y bacterias, comportamiento de glóbulos en el cuerpo, contaminación, evolución galáctica, ecosistemas, genética, etc.

Por ejemplo: "*Según (Martí & Turjanski, 2009) consiste en realizar una simulación de comportamiento entornos. La simulación puede establecer la capacidad biomolecular, principalmente proteínas y sus entornos. La simulación puede establecer la capacidad de que una droga si puede tratar ciertas enfermedades, permitiendo así con más rapidez el descubrimiento y proceso óptimo de estas medicinas...*" [4]

2. *Arquitectura*

Los autómatas celulares al ser capaces de generar patrones y de forma estructurada nos permite visualizar formas arquitectónicas ya que son capaces de trabajar en n dimensiones. Por lo tanto, según el patrón que uno les dé puede presenciar distintas figuras, la cual al derivarla al espacio en 3 dimensiones nos dará una base para el modelamiento que uno necesite en este caso ocupar espacios para la construcción de edificios, casas etc.

3.- *Incendios Forestales*

Las simulaciones de incendios forestales son de gran utilidad para la planificación y mantenimiento de grandes áreas boscosas, ya sean naturales o artificiales, puesto que permiten conocer de antemano dónde deben de existir zonas para la contención de incendios y contribuye a la toma de decisiones en caso de un evento perjudicial.

Se ha de considerar como principales factores las características del terreno, condiciones meteorológicas, condiciones geográficas y el epicentro o foco del incendio.

Estos factores son de suma importancia porque se desea obtener resultados fiables y en el menor tiempo posible, para combatir y minimizar el área afectada por el fuego.

V. ELECCION DEL LENGUAJE

En este proyecto utilizamos "C++", esto debido más que nada a la familiaridad que tenemos con este Lenguaje y el mayor manejo que tenemos de sus características. Además, al ser este Proyecto una Evaluación Importante, decidimos utilizar un lenguaje confiable y con el cual tenemos experiencia, para así asegurarnos de trabajar sin mayores inconvenientes y con la mayor efectividad posible.

Algoritmo

1.- *Se Crean las Reglas*:

- Se Repite el Procedimiento en cada una de las reglas. Solo se cambia en cada condición el valor del guardado en el vectorTransición[i] dependiendo de lo que dicte la regla que se esté programando.
- "i" es la posición donde se iniciará a analizar el vector inicial, y donde se guardará en el vector de transición (auxiliar) el valor obtenido de aplicar la regla.
- "Pos1" y "pos2" son valores para analizar los siguientes dos valores del vector inicial, cobra importante función cuando se llegue al final del vector, permite que el vuelvan a analizar los primeros datos junto a los últimos y así cerrar el vector en una especie de "ciclo".
- "vectorTransición[]" (variable global): vector auxiliar para depositar los datos obtenidos de la regla y luego copiarlos en el vector inicial para volver analizarlos.

voidRegla1(entero vector [], i, pos1, pos2)
{
Si vector[i]=0, vector[i+pos1]=0 y vector[i+pos2]=0 //000
Entonces *vectorTransición[i]=0*

Si vector[i]=0, vector[i+pos1]=0 y vector[i+pos2]=1 //001
Entonces *vectorTransición[i]=1*

Si vector[i]=0, vector[i+pos1]=1 y vector[i+pos2]=0 //010
Entonces *vectorTransición[i]=1*

Si vector[i]=0, vector[i+pos1]=1 y vector[i+pos2]=1 //011
Entonces vectorTransición[i]=0

Si vector[i]=1, vector[i+pos1]=0 y vector[i+pos2]=0 //100
Entonces *vectorTransición[i]=1*

Si vector[i]=1, vector[i+pos1]=0 y vector[i+pos2]=1 //101
Entonces *vectorTransición[i]=0*

Si vector[i]=1, vector[i+pos1]=1 y vector[i+pos2]=0 //110
Entonces *vectorTransición[i]=0*

Si vector[i]=1, vector[i+pos1]=1 y vector[i+pos2]=1 //111
Entonces *vectorTransición[i]=0*
}

```
voidReglaMayoria(entero vector [], i, pos1, pos2)
{
   Si vector[i]=0, vector[i+pos1]=0 y vector[i+pos2]=0  //000
      Entonces vectorTransición[i]=0

   Si vector[i]=0, vector[i+pos1]=0 y vector[i+pos2]=1  //001
      Entonces vectorTransición[i]=0

   Si vector[i]=0, vector[i+pos1]=1 y vector[i+pos2]=0  //010
      Entonces vectorTransición[i]=0

   Si vector[i]=0, vector[i+pos1]=1 y vector[i+pos2]=1  //011
      Entonces vectorTransición[i]=1

   Si vector[i]=1, vector[i+pos1]=0 y vector[i+pos2]=0  //100
      Entonces vectorTransición[i]=0

   Si vector[i]=1, vector[i+pos1]=0 y vector[i+pos2]=1 //101
      Entonces vectorTransición[i]=1

   Si vector[i]=1, vector[i+pos1]=1 y vector[i+pos2]=0  //110
      Entonces vectorTransición[i]=1

   Si vector[i]=1, vector[i+pos1]=1 y vector[i+pos2]=1  //111
      Entonces vectorTransición[i]=1
}
```

```
void Regla110(entero vector [], i, pos1, pos2)
{
   Si vector[i]=0, vector[i+pos1]=0 y vector[i+pos2]=0  //000
      Entonces vectorTransición[i]=0

   Si vector[i]=0, vector[i+pos1]=0 y vector[i+pos2]=1  //001
      Entonces vectorTransición[i]=1

   Si vector[i]=0, vector[i+pos1]=1 y vector[i+pos2]=0  //010
      Entonces vectorTransición[i]=1

   Si vector[i]=0, vector[i+pos1]=1 y vector[i+pos2]=1  //011
      Entonces vectorTransición[i]=1

   Si vector[i]=1, vector[i+pos1]=0 y vector[i+pos2]=0  //100
      Entonces vectorTransición[i]=0

   Si vector[i]=1, vector[i+pos1]=0 y vector[i+pos2]=1 //101
      Entonces vectorTransición[i]=1

   Si vector[i]=1, vector[i+pos1]=1 y vector[i+pos2]=0  //110
      Entonces vectorTransición[i]=1

   Si vector[i]=1, vector[i+pos1]=1 y vector[i+pos2]=1  //111
      Entonces vectorTransición[i]=0
}
```

2.-*Se Aplican Las Reglas*

- Se Repite el Procedimiento en cada una de las reglas. Solo se cambia la Función a la Regla se desee ocupar, los parámetros de estas funciones se mantienen contantes en las Reglas.

- Se ejecuta el Primer Ciclo la cantidad de veces que requiera. En nuestro caso haremos una matriz 500x500.

- El Segundo Ciclo de la Función es para desplazarse y analizar el vector inicial hasta la antepenúltima posición, pues este es el último dato en el cual se pueden analizar los dos siguientes datos sin volver al inicio del vector.

- Una vez fuera del segundo ciclo se analizan los dos datos faltantes:
  - En el primer caso, la posición a analizar "i" la cual tendrá el valor=(largo-1) y la *pos1=1* para que inicie en el siguiente y ultimo valor, mientras que la *pos2=(1-largo)* para que vuelva a la posición inicial "0".

  - En el segundo caso, la posición a analizar "i" tendrá el valor del largo y por ende la *pos1=(-largo)* para que vuelva a la posición inicial "0" y la *pos2=(1-largo)* para que vuelva a la segunda posición "1"

- Se copia el vectorTransición al vector inicial.

- Finalmente se muestra el vector y se crean salto de línea para dar la forma de Matriz.

```
void AplicarRegla1(intv[])
{
   Para cada x=0 hasta 500 hacer
      Para cada i=0 hasta i=largo-2 hacer
         Regla1(v,i,1,2)
      Fin para
   Fin para
   Regla1(v, (largo-1), 1, (1-largo))
   Regla1(v, largo, (-largo), (1-largo))
   copiarVector(v,vectorTransición)

}
```

```
void AplicarReglaMayoria(int v[])
{
   Para cada x=0 hasta 500 hacer
      Para cada i=0 hasta i=largo-2 hacer
         ReglaMayoria(v,i,1,2)
      Fin para
   Fin para

   ReglaMayoria(v, (largo-1), 1, (1-largo))
   ReglaMayoria(v, largo, (-largo), (1-largo))

   copiarVector(v,vectorTransición)
   mostrarVector(v)
   cout<<endl;
}
```

```
void AplicarRegla110(intv[])
{
   Para cada x=0 hasta 500 hacer
      Para cada i=0 hasta i=largo-2 hacer
         Regla110(v,i,1,2)
      Fin para
   Fin para

   Regla110(v, (largo-1), 1, (1-largo))
   Regla110(v, largo, (-largo), (1-largo))

   copiarVector(v,vectorTransición)
   mostrarVector(v)
   cout<<endl;
}
```

2.-*Salida de Pantalla*

- Se Incorpora la Estructura _console_font_infoex, perteneciente a la librería "Windows.h", se encuentra como código libre en:

  https://docs.microsoft.com/en-us/windows/console/console-font-infoex

  https://docs.microsoft.com/en-us/windows/console/setcurrentconsolefontex.

- Se Crea la función para cambiar el tamaño de la Fuente usada en la consola:

```
void setFontSize(int alto,int ancho)
{
  //se establece el nombre de la variable a usar
  CONSOLE_FONT_INFOEX cfi

  //para manipular la salida de pantalla.
  GetStdHandle(STD_OUTPUT_HANDLE)

  // para conocer el tamaño de fuente actual
  cfi.cbSize = sizeof(cfi)

  //modifican el tamaño de la fuente elegido por el usuario,
  cfi.dwFontSize.X=ancho
  cfi.dwFontSize.Y=alto

  //se llama finalmente al método de la estructura para ejecutar os cambios.
  SetCurrentConsoleFontEx(outcon, NULL, &cfi),
}
```

- Se crea la función que simula el acceso de teclado "Alt+Enter"

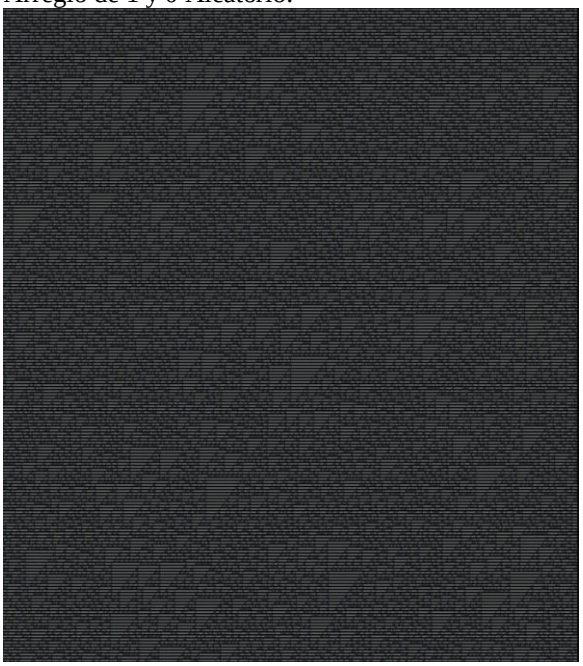
```
void AltEnter()
{
 //Simula presion"Alt"
    keybd_event(VK_MENU,0x38,0,0);

 // Simula presion"Enter"
    keybd_event(VK_RETURN,0x1c,0,0);

 //Simula cuando se suelta "Enter"
    keybd_event(VK_RETURN,0x1c,KEYEVENTF_KEYUP,0);

 //Simula cuando se suelta "Alt"
    keybd_event(VK_RETURN,0x38,KEYEVENTF_KEYUP,0);

 return
 }
```

VI. Aplicación de las Reglas

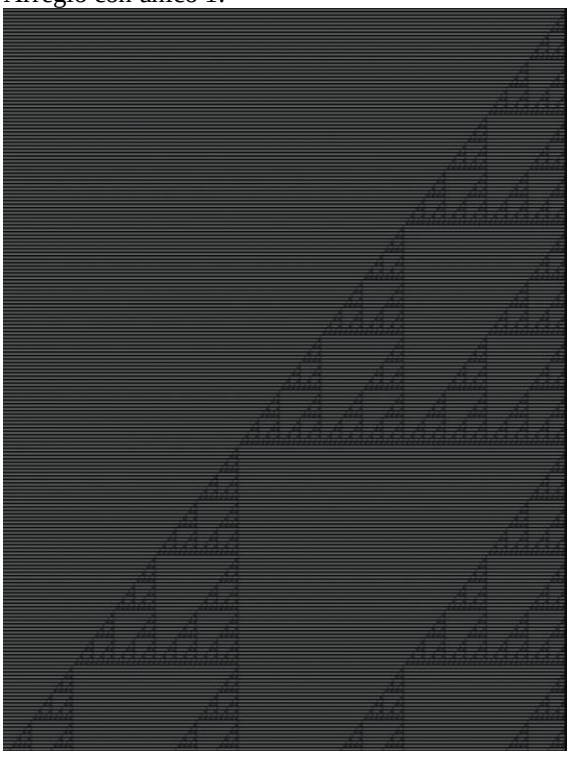
Imágenes

1. *Regla del 1:*

Arreglo de 1 y 0 Aleatorio:



Arreglo con único 1:



- *Regla del 1:*

La Regla del 1 se produce al poner un uno solo si el arreglo de tres números contiene solo un 1.

También llamada Regla "104", dado que:
104d = 01101000b.

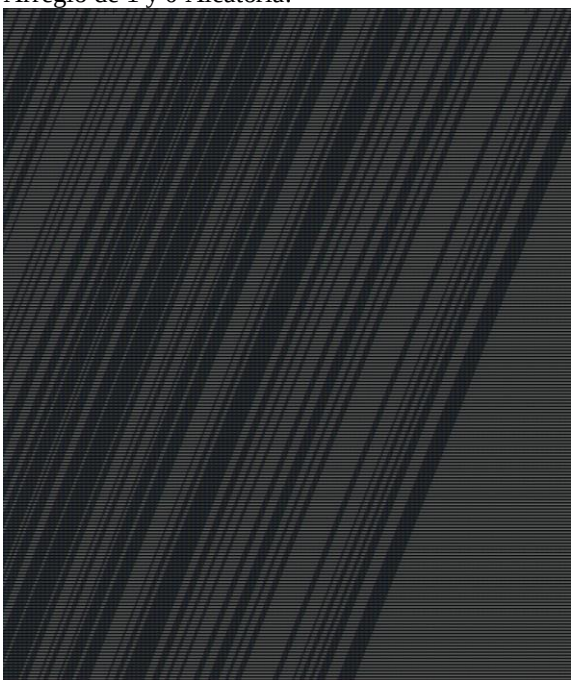
000	001	010	011	100	101	110	111
0	1	1	0	1	0	0	0

En esta Regla podemos ver la formación de varios triángulos formados por otros triángulos más pequeños, ósea, se puede decir que es una figura[1] fractal conocida como el triángulo de Sierpinski.

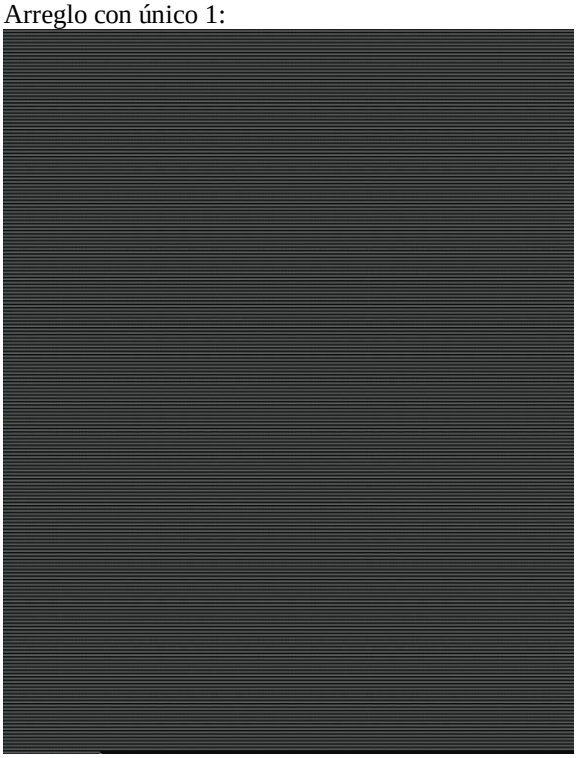
1. Fractal: objeto geométrico cuya estructura básica se repite en distintas escalas.

2. *Regla de la Mayoría:*

Arreglo de 1 y 0 Aleatoria:



Arreglo con único 1:



- *Regla de la Mayoría:*

  La Regla de la Mayoría es generada cuando en el arreglo de tres números, el 1 es mayoría.

  También llamada Regla "11", dado que:
  11d = 0001011b.

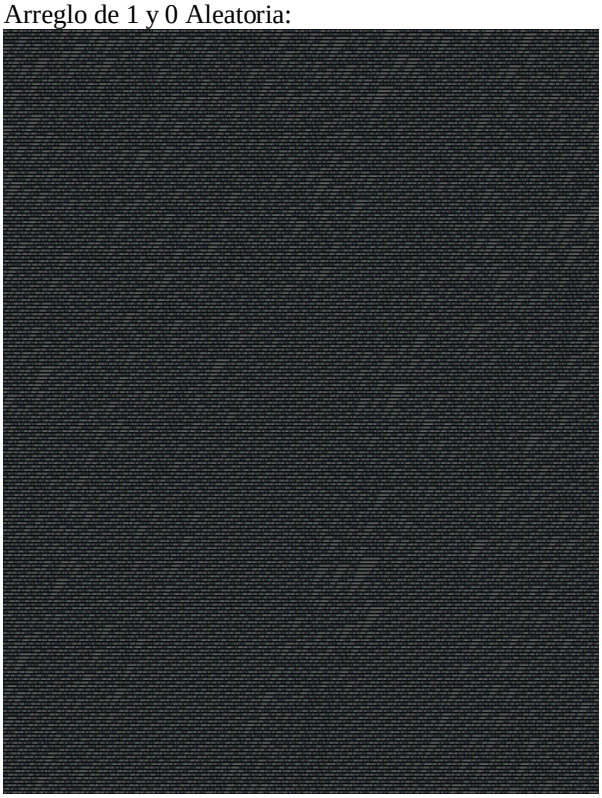
000	001	010	011	100	101	110	111
0	0	0	1	0	1	1	1

  En esta Regla el Arreglo Aleatoria es más significativo para su estudio, pues tal y como se observa en la imagen, no tiene sentido analizar el Arreglo con un único 1.

  Se puede interpretar el resultado como dos posturas, indicadas por los 0 y 1, y analizar el resultado, para ver cuál postura se impone a la otra.

3. *Regla del 110:*

Arreglo de 1 y 0 Aleatoria:



Arreglo con único 1:

- *Regla del 110:*

Llamada Regla “110”, dado que:
110d = 01101110b

111	110	101	100	011	010	001	000
0	1	1	0	1	1	1	0

Es una Regla de gran importancia para la computación, pues esta puede simular una Maquina de Turing, además es una regla muy interesante de estudiar, pues su comportamiento está en el límite entre estabilidad y caos.

“*En 1985, Wolfram conjeturo que la regla 110 era computacionalmente universal, pero esto no fue probado sino hasta 2004 en por Matthew Cook. Para simular una maquina de Turing, este autómata celular tiene que simular primero un sistema Tag cíclico . El sistema tag cíclico a su vez puede simular un sistema Tag tradicional, mismo que tiene la capacidad de simular una máquina de Turing como lo demuestra Matthew Cook…”*[5]

Al igual que en la Regla 1 se produce una figura formada por fractales pero a diferencia de esta , la Regla del 110 no presenta un orden exacto en su formación.

- *Diferencias en los tipos de Arreglos:*

En el Arreglo con un único 1, se puede distinguir claro y ordenado el patrón generado por el Autómata Celular, en el caso de la Regla del 1 y del 110, se puede observar una figura formada por fractales.

En el Arreglo Aleatorio, se puede apreciar un patrón similar anterior, pero pierde todo orden y crea formaciones complejas las cuales se auto-organizan y aun así conservan la estructura básica de la Regla aplicada.

REFERENCES

Basic format for books:

[1] Dacia Arminda Flores Ojeda “Introducción a los Autómatas Celulares CAs”
http://www.revistasbolivianas.org.bo/pdf/rits/n1/n1a28.pdf

[2] Álvaro Álvarez Parrilla, Aurora Espinoza Valdéz “Autómatas Celulares Aditivos: la regla 150 vs. la regla 90”
http://132.248.9.34/hevila/RevistadelCentrodeInvestigacionUniversidadlaSalle/2008/vol8/no30/2.pdf

[3] David Alejandro Reyes Gómez “Descripción y Aplicaciones de los Autómatas Celulares”
http://delta.cs.cinvestav.mx/~mcintosh/cellularautomata/Summer_Research_files/Arti_Ver_Inv_2011_DARG.pdf

[4] Simon Orozco Arias, Jeferson Arango López “Aplicación de la inteligencia artificial en la bioinformática, avances, definiciones y herramientas”
http://revistas.ugca.edu.co/index.php/ugciencia/article/download/494/1072

[5] Juárez Martínez Sergio Eduardo Manzano Mendoza Cesar Iván “Maquinas de Turing y el problema de la universalidad como sistemas dinámicos discretos”
http://uncomp.uwe.ac.uk/genaro/Papers/Thesis_files/CAasTM.pdf

[6] Genaro Juárez Martínez “Introducción a la simulación de procesos con autómata celular”
https://2006.igem.org/wiki/images/1/1b/IntoCA.pdf

[7] Raúl Rechtman “Una Introducción a Autómatas Celulares”
http://www.ejournal.unam.mx/cns/no24/CNS02405.pdf

[8] Guido Gorostiaga Marin “Autómatas celulares y su aplicación”
http://www.revistasbolivianas.org.bo/pdf/rits/n4/n4a03.pdf

[9] Ana Miriam López Salinas “Introducción a La Vida Artificial y Autómatas Celulares”
http://uncomp.uwe.ac.uk/genaro/Papers/Veranos_McIntosh_files/vida_artificial_Miriam.pdf

[10] Steven Weinberg “¿Es el universo un ordenador?”
https://www.revistadelibros.com/articulo_imprimible_pdf.php?art=2875&t=articulos

[11] Alberto Cano Rojas, Ángela Rojas Matas “Autómatas celulares y aplicaciones”
http://www.fisem.org/www/union/revistas/2016/46/01_13-307-1-ED.pdf