

Instituto Politécnico Nacional

”La técnica al servicio de la patria”



Escuela Superior de Cómputo

AUTÓMATA CELULAR: EL JUEGO DE LA VIDA

Alumno: Cristian Hernández Resendiz

Optativa: Computing selected topics

Materia: Complex systems

Profesor: Genaro Juarez Martínez

Grupo: 3CM9

Boleta: 2016602153

Índice general

1. Introducción	5
2. Desarrollo	7
2.1. El juego de la vida, un autómata celular	7
2.1.1. Reglas	7
2.1.2. Patrones	8
2.2. Atractores, caos y complejidad	9
2.3. Descripción del software de aplicación	12
2.3.1. Requerimientos para el desarrollo del software	12
2.4. Ejemplo de funcionamiento del software	14
2.4.1. Life	14
2.4.2. Generación de atractores	16
2.4.3. Clasificación de los atractores	23
2.5. Código fuente	24
3. Conclusiones	47
4. Referencias	49

Capítulo 1

Introducción

Un ecosistema, o el medioambiente en general, son sistemas complejos. Las organizaciones y estructuras humanas (económicas, sociales, políticas, etc.) son aún más complejas. Cualquier análisis o estudio territorial, a la escala que sea, cualquier plan de desarrollo, ya sea urbanístico o económico, cualquier decisión política y toda intervención en un ámbito social determinado incide en un medio muy dinámico y cambiante; muchas veces autónomo en su evolución. Actuaciones que persiguen un beneficio o la mejora de una situación dada pueden fácilmente tener efectos inesperados y provocar a la postre un efecto contrario. El estudio de un sistema de estas características es hoy en día un reto para cualquier disciplina científica. Los economistas por su lado, los urbanistas por otro; los sociólogos en su esfera o los polítólogos en la suya, todos se enfrentan a la dificultad de 'manejar' adecuadamente estas estructuras dinámicas. Más si cabe cuando la meta es pronosticar la evolución futura del sistema en cuestión. Y de ello dependen muchos intereses y a veces hasta muchas vidas. La naturaleza impredecible de estos sistemas se asume en general como una característica intrínseca y propia y lo que es peor, inevitable. Se trate de la Bolsa, de la economía local o internacional, de la evolución de la moda, de la competencia industrial en los mercados, de la respuesta social a fenómenos culturales o del estudio de los índices de gobernanza de un país, resulta escasa la precisión en la descripción de la situación en el presente mientras que se convierte en un arte adivinadorio la predicción de su evolución futura. Para poder diagnosticar correctamente un sistema de este tipo habría que recopilar y luego integrar una cantidad de datos tan inmensa que los ordenadores que disponemos en la actualidad se quedan cortos. Pero el problema no es sólo ese. Suponiendo que pudieramos disponer de la tecnología adecuada aún quedaría por solucionar otro asunto. Un sistema complejo es tan dinámico que cambia continuamente. Por lo tanto, los datos que se obtuvieron ayer y que se analizan hoy igual ya no tienen nada que ver con la realidad de mañana. Hasta ahora la estrategia de aproximación al estudio de lo complejo se basa en el cálculo estadístico. Se seleccionan una serie limitada de variables del sistema, las que a priori se considera que más influyen en los procesos internos de la estructura y se trabaja con ellas estableciendo correlaciones y sacando así conclusiones. Muchas veces este modo de aproximación funciona bien y permite obtener una visión muy realista de la situación y, a veces, hasta sirve para anticiparse a los cambios que se producen. Sin embargo también ocurre con demasiada frecuencia que cuando el investigador cree controlar todas las variables importantes, aparece una nueva e inesperada que ha adquirido un protagonismo imprevisto en el funcionamiento del sistema y por extensión, en su evolución. Por ello, estos modos de aproximación, estas técnicas de estudio, tienen un valor muy relativo y una utilidad bastante limitada. El ejemplo más ilustrativo de un sistema complejo es un organismo biológico. Una célula, un órgano, una planta o un animal responden a este patrón. Pero así mismo lo es la estructura social en la que se organizan las poblaciones de animales como termitas, hormigas, abejas, una familia de leones o una manada de antílopes. Lo es también una comunidad de seres vivos, animales y vegetales, que comparten un hábitat en

cualquier ecosistema en general. Y lo es así mismo un bioma, a una escala mucho más amplia y referida a las condiciones climáticas de una zona del planeta. Cada uno de ellos, a su nivel, actúa como una unidad autónoma, como un organismo individual. Dispone de un 'dentro' organizado y de un 'afuera' con el que interactúa. Aunque pueda parecer a primera vista que el límite de un sistema está bien definido, lo que nos resulta casi evidente en el caso de una célula, de un órgano, de una planta o de un animal, no podemos olvidar que el sistema no existe sin su medio exterior con el cual intercambia constantemente energía y materia. La respiración y la alimentación son buena prueba de ello. Por otra parte, un individuo siempre es parte constituyente de otra individualidad superior. Y a su vez, ese mismo individuo está compuesto de partes constituyentes que son individualidades a su vez, en una escala inferior. Esto crea una estructura jerárquica de sistemas que engloban otros subsistemas en una sucesión interminable. Este es el caso de la célula que visualizamos perfectamente como un ente individual y autónomo. Sin embargo, esa misma célula, integrada en un tejido y tal vez en un órgano, es parte de un ser superior. Es un simple ladrillo de la arquitectura de ese ser superior. Pero la célula no es indivisible. Todo lo contrario, ella misma está 'fabricada' por componentes menores que pueden describirse en una escala descendente. Así, podemos establecer una primera división en núcleo, citoplasma y membrana y luego podemos ir descendiendo al nivel de los organélos celulares y de allí a la escala de las macromoléculas. A su vez, el ser superior de esa célula, ya sea una planta o un animal, comparte su existencia con sus congéneres. En un espacio común, en un trozo de terreno específico, interactúan entre todos ellos y crean una unidad superior dotada de 'individualidad' propia. Es la manada o la bandada o la familia o la colmena, etc. Dada esta idea, a continuación se presenta un juego que simula el movimiento de células dentro de un espacio bidimensional; se trata del juego de la vida.

Capítulo 2

Desarrollo

2.1. El juego de la vida, un autómata celular

En el año 1970, el matemático inglés John Horton Conway creó un juego matemático de simulación que consistía en un universo cuadriculado donde es posible determinar un estado inicial y luego hacerlo correr para observar su evolución. Lo llamó 'el Juego de la Vida 2' es en realidad un autómata celular. Esto es, un modelo matemático para un sistema dinámico que va evolucionando en pasos discretos. El juego es un autómata celular bidimensional en el cual cada celda (célula) puede estar en uno de los dos estados posibles, viva o muerta. Partiendo de un estado inicial, la simulación va haciendo evolucionar al autómata en base a unas sencillas funciones de transición. Una célula va a estar en un estado concreto, el cual sera determinado únicamente del estado inmediatamente anterior de las células vecinas y el de la propia célula.

2.1.1. Reglas

El conjunto de reglas básicas que propuso Conway son las siguientes:

- Una célula muerta con exactamente 3 células vecinas vivas 'nace' (es decir, al turno siguiente estará viva).
- Una célula viva con 2 o 3 células vecinas vivas sigue viva, en otro caso muere (por 'soledad' o 'superpoblación').

Cada célula viva o muerta en el espacio bidimensional se rige en sus 8 células vecinas cercanas de la siguiente manera.

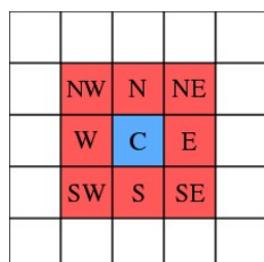


Figura 2.1: Vecindad de Moore

La célula C esta rodeada de 8 celdas de color rojo. Estas últimas son células que pueden estar vivas o muertas. Todas las células en el espacio bidimensional determinan su estado mediante sus 8 vecinas cercanas, siendo estas vivas o muertas en ese momento.

2.1.2. Patrones

Un patrón es una configuración de células que permanecen vivas en un momento determinado. Existen básicamente 3 clases de patrones en el juego de la vida de Conway.

- Patrones estáticos: Un conjunto de células vivas que se mantiene estático, sin que se produzcan nuevos nacimientos o muertes.

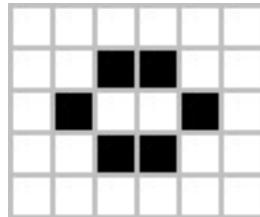


Figura 2.2: Patrones estáticos

- Patrones recurrentes: Un conjunto de células vivas que no se mueve por el mundo, pero que no es estático, ya que se producen nacimientos y muertes, produciendo transiciones que se repiten continuamente.

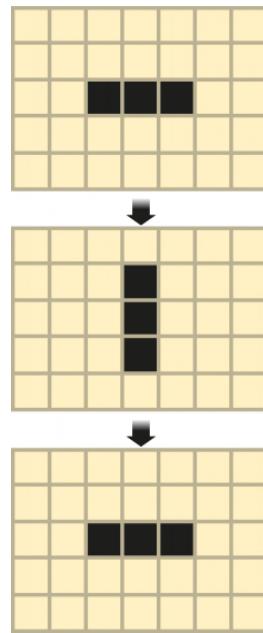


Figura 2.3: Patrones recurrentes

- Patrones que se trasladan: Un conjunto de células vivas que permanece con la misma forma, pero que se desplaza por el tablero.

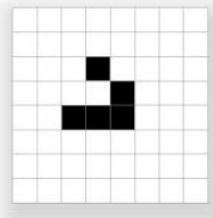


Figura 2.4: Patrones en translación

2.2. Atractores, caos y complejidad

Las condiciones para que un sistema sea no caótico (lineal) son que: 1) si situamos un objeto en una posición determinada, podremos calcular con exactitud y de forma sencilla en que posición se encontrará al cabo de cierto tiempo; 2) si situamos dos objetos en posiciones iniciales cercanas, al cabo del tiempo dichos objetos permanecerán más o menos cercanos, ya que sobre ellos actuarán fuerzas de parecida magnitud. En el análisis de los sistemas complejos, la base del análisis es lo nolineal. Se habla de sistemas lineales cuando estos sistemas son predecibles, previsibles. Un sistema caótico es, sin duda, un sistema más difícil de tratar, ya que debido a su imprevisibilidad siempre tiene alguna sorpresa que ofrecer (Nicholls 1998) . Si un sistema dinámico es caótico, tiene un componente de impredicibilidad, un componente de irreducibilidad, pero aun así tiene un tercer componente de regularidad (Upm 1998). Un caos totalmente aleatorio conlleva la irregularidad en su comportamiento, y el análisis científico actual ha avanzado poco en su conocimiento.

Ha quedado señalado que en la Tcaos actual existen dos enfoques básicos para entender el desorden de los sistemas: el “enfoque de las estructuras disipativas”, y el “enfoque de los atractores”. En el primero, el caos se considera como el precursor y socio del orden y no como su opuesto. Aquí se centra la atención en el surgimiento espontáneo de “autoorganizaciones” que emergen del caos, o, según la terminología del campo, en las “estructuras disipativas” que surgen en sistemas fuera de equilibrio, cuando la producción de entropía es alta. La comprensión de que los sistemas ricos en entropía facilitan en vez de impedir la auto-organización ha sido una coyuntura decisiva para la revaluación contemporánea del caos.

El segundo enfoque destaca el orden oculto que existe dentro de los sistemas caóticos. Usado de este modo, el término “caos” difiere de la total aleatoriedad, porque se puede demostrar que contiene estructuras profundamente codificadas, llamadas “atractores extraños”. Mientras que los sistemas verdaderamente aleatorios no muestran un esquema discernible cuando se les organiza en el espacio de fase, los sistemas caóticos se concentran en una región limitada y trazan modelos complejos dentro de ella (Hayles 1998: 29).

El enfoque de las “estructuras disipativas” permite entender que el conjunto de los diversos sistemas naturales, biológicos y humanos (supersistema), generan durante su convivencia intercambios de energía, recursos o informaciones, lo que da origen a la entropía. Esta, en lugar de degenerar o perderse, es aprovechada por algunos sistemas para revitalizarse, o transformarse, lo cual puede dar origen, o recrear, nuevas estructuras (sistemas). La naturaleza emplea el caos en forma constructiva. Al amplificar las pequeñas fluctuaciones provee sistemas que permiten el acceso a la creatividad. Los sistemas complejos pueden generar anticaos, es decir, un sistema desordenado que “cristaliza” en orden. El caos puede proporcionar los medios para estructurar los cambios azarosos, ofreciendo la posibilidad de poner la diversificación bajo el control de la evolución (Schifter 1996: 100-103). De esta manera, el supersistema se autorganiza a partir del caos.

Por su parte, el enfoque de los “atractores” proporciona herramientas para entender o delimitar (“medir”) dicho caos. Atractor es el término técnico para la figura o trayectoria básica del estado final al que tiende el sistema (Gleick 1987:121-153). Por ejemplo, si se pone una canica en un embudo, acabará siempre en el agujero, independientemente de la pared del embudo donde se haya colocado inicialmente. El agujero del embudo sería el atractor del sistema. Un gran atractor contiene todos los posibles estados finales de un sistema dinámico, una especie de metaembudo. El ejemplo clásico es el clima, el cual sería el gran atractor de la meteorología (Navaltropo 1998). La complejidad de este atractor ha hecho que se le denomine como extraño atractor.

Es importante apreciar que los patrones de fase espacial de un sistema caótico nunca coinciden. Si esto ocurriera, el sistema se transformaría en periódico. Un atractor de un sistema

caótico de las ciencias duras es una maraña de trayectorias, una “turbulencia”. Superficialmente, el atractor parece estar completamente desorganizado. Sin embargo, una observación más cuidadosa de la fase espacial revela que el atractor está organizado pero de una manera no convencional. A diferencia de las ciencias duras, en las “ciencias blandas” los atractores tienden a ser “elementos”, “factores”, “cuellos de botella” o “causas” que originan desórdenes en la sociedad, su economía, cultura o política. En las regiones los atractores son “elementos” generadores de entropía (activa o inactiva) (Briggs 1994: 19-23). Por consiguiente, los “attractores regionales” generan a la vez bienestar y caos en las regiones, y son resultado de la acumulación de experiencias, situaciones, conocimientos y actitudes resultado de la interacción de la sociedad, la economía, la cultura, la ecología y el territorio de las propias regiones. Se convierten en referentes que en ocasiones repentinamente son activados por situaciones que se asemejan a las experiencias precedentes: están siempre presentes en espera de ser puestos en operación por la propia interacción de las regiones. A través de los atractores, las regiones confirman su carácter complejo, cambiante, su comportamiento no lineal, oscilante entre el orden y el caos, entre el bienestar y el desorden.

En un sistema caótico complejo como el de la región es posible la existencia de atractores múltiples (Nicholls y Tagarev 1998). Esto significa que los sistemas caóticos pueden tener diversos niveles de estabilidad-inestabilidad cuando operan a favor del bienestar o del desorden. El clima de la tierra es un buen ejemplo de esta forma de comportamiento, pues se estima que el clima actual sufre comportamientos inesperados, pasando de una situación a otra. En las regiones, los cambios pueden deberse a la presencia de diversos atractores, algunos básicos, como la pobreza o el desempleo, cuya presencia afecta la estructura del sistema; otros intermedios, como la aparición de un orden social alternativo, los cuales generan cambios parciales en los sistemas sociales; y también la existencia de atractores profundos como la destrucción cultural o ecológica, los cuales tienden a ocasionar el cambio radical del sistema. En estos casos el caos se genera debido a la activación de un atractor.

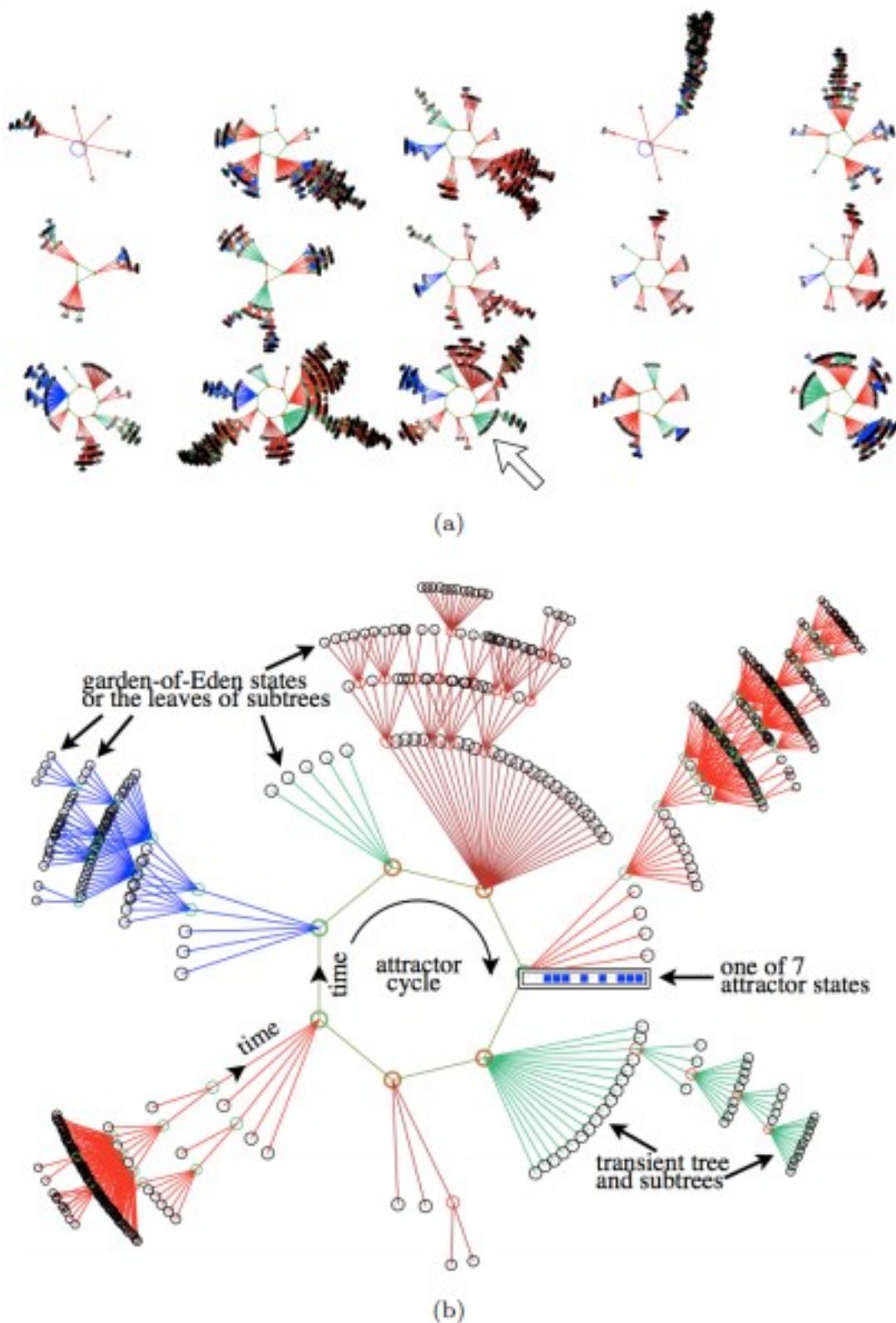


Figura 2.5: En (a) se ilustra el campo del conjunto de atracción de una red booleana aleatoria (RBN). En (b) se presenta en detalle un conjunto de atracción, como configuraciones de bits (flecha de arriba indicada en (a)) con 604 estados de los cuales 523 son hojas y el atractor es de periodo igual a 7. La dirección del tiempo es hacia dentro del atractor y con orientación al sentido de las manecillas del reloj.

2.3. Descripción del software de aplicación

Se ha realizado una versión del juego de la vida de Conway en un universo plano empleando el lenguaje de programación interpretado llamado **python**. Esta versión cuenta con las siguientes funcionalidades:

- **Control del juego:** Es posible pausar y reproducir el juego cuando el usuario lo deseé.
- **Cambiar el color de las células:** El programa permite cambiar el color de las células vivas y muertas que están en el espacio celular de evolución
- **Cambiar el tamaño de la célula:** La célula puede tener tamaño de hasta 10 px (pixeles).
- **Cargar configuraciones:** Esta función le permite al usuario cargar patrones de células vivas guardados en archivos .npy
- **Guardar configuraciones:** El programa tiene un módulo para guardar patrones de células vivas que estén en el espacio celular plano
- **Ver gráfica de la densidad de células vivas:** En este punto, el usuario tiene la opción de visualizar las células vivas que han estado en cada fase de evolución. Estos datos están expresados en una gráfica.
- **Añadir y borrar células en el espacio celular:** Cuando el usuario de un click izquierdo o derecho sobre una celula muerta, esta se convierte a una célula viva; y viceversa.
- **Cambiar el tamaño del espacio celular:** El universo celular plano puede ser de tamaño de 100x100, 500x500, 1000x1000, 5000x5000 y 10000x10000.
- **Cambiar las reglas del juego:** El programa acepta cadenas con la siguiente expresión regular: **B0-8/S0-8**. Estas cadenas indican que nace (B) una célula muerta cuando tiene 0,1,2,..., u 8 células vecinas vivas; mientras que, para que sobreviva (S) una célula viva, deben existir 0,1,2,..., u 8 células vecinas vivas.
- **Reiniciar juego:** Cuando el usuario reinicie el juego, todas las células vivas pasarán a ser células muertas.
- **Generación de los árboles (trees) o atractores:** Como módulo extra, el programa calcula los atractores que existen en los universos celulares de tamaño 2x2, 3x3, 4x4 y 5x5. Estos atractores se representan de forma gráfica mediante una red o varias redes.
- **Ver los atractores generados:** Teniendo calculados los atractores, el usuario puede ver las redes o grafos del universo celular plano 2x2, 3x3, 4x4 o 5x5.

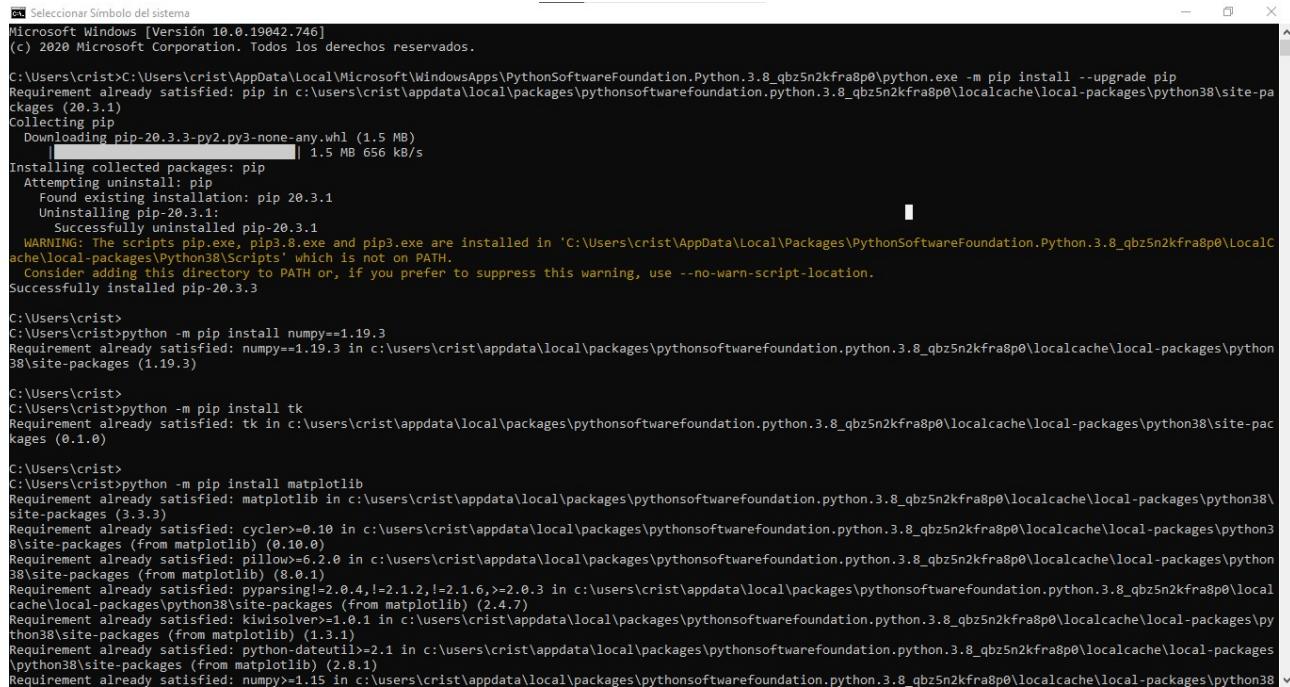
2.3.1. Requerimientos para el desarrollo del software

A continuación, se enumeran todos los paquetes de software de instalación que se usaron para crear este software:

1. Instalar **python versión 3.8** desde la tienda de Microsoft.
2. Instalar y configurar el instalador de paquetes (pip) de python.
3. Teniendo el instalador pip python, instalar los siguientes paquetes:
 - Tkinter

- Matplotlib
- Networkx
- Scipy

En los puntos 2 y 3, colocar los siguientes comandos (respectivamente):



```
C:\Users\crist>C:\Users\crist\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.8_qbz5n2kfra8p0\python.exe -m pip install --upgrade pip
Requirement already satisfied: pip in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\python38\site-packages (20.3.1)
Collecting pip
  Downloading pip-20.3.3-py2.py3-none-any.whl (1.5 MB)
    100% |████████████████████████████████| 1.5 MB 656 kB/s
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 20.3.1
    Uninstalling pip-20.3.1:
      Successfully uninstalled pip-20.3.1
      WARNING: The scripts pip.exe, pip3.8.exe and pip3.exe are installed in 'C:\Users\crist\AppData\Local\ Packages\PythonSoftwareFoundation.Python.3.8_qbz5n2kfra8p0\LocalCache\local-packages\Python38\Scripts' which is not on PATH.
      Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed pip-20.3.3

C:\Users\crist>
C:\Users\crist>python -m pip install numpy==1.19.3
Requirement already satisfied: numpy==1.19.3 in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\python38\site-packages (1.19.3)

C:\Users\crist>
C:\Users\crist>python -m pip install tk
Requirement already satisfied: tk in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\python38\site-packages (0.1.0)

C:\Users\crist>
C:\Users\crist>python -m pip install matplotlib
Requirement already satisfied: matplotlib in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\python38\site-packages (3.3.3)
Requirement already satisfied: cycler>=0.10 in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\python38\site-packages (from matplotlib) (0.10.0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\python38\site-packages (from matplotlib) (8.0.1)
Requirement already satisfied: pyarsing!=2.0.4,!!=2.1.2,!!=2.1.6,>=2.0.3 in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\pythonsite-packages (from matplotlib) (2.4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\python38\site-packages (from matplotlib) (1.3.1)
Requirement already satisfied: python-dateutil!=2.1 in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\pythonsite-packages (from matplotlib) (2.8.1)
Requirement already satisfied: numpy>=1.15 in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\python38\site-packages (from matplotlib) (1.19.3)
Requirement already satisfied: six in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\python38\site-packages (from cycler>=0.10->matplotlib) (1.15.0)
```

Figura 2.6: Instalación de paquetes 1



```
C:\Users\crist>
C:\Users\crist>python -m pip install tk
Requirement already satisfied: tk in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\python38\site-packages (0.1.0)

C:\Users\crist>
C:\Users\crist>python -m pip install matplotlib
Requirement already satisfied: matplotlib in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\python38\site-packages (3.3.3)
Requirement already satisfied: cycler>=0.10 in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\python38\site-packages (from matplotlib) (0.10.0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\python38\site-packages (from matplotlib) (8.0.1)
Requirement already satisfied: pyarsing!=2.0.4,!!=2.1.2,!!=2.1.6,>=2.0.3 in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\pythonsite-packages (from matplotlib) (2.4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\python38\site-packages (from matplotlib) (1.3.1)
Requirement already satisfied: python-dateutil!=2.1 in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\pythonsite-packages (from matplotlib) (2.8.1)
Requirement already satisfied: numpy>=1.15 in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\python38\site-packages (from matplotlib) (1.19.3)
Requirement already satisfied: six in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\python38\site-packages (from cycler>=0.10->matplotlib) (1.15.0)

C:\Users\crist>
C:\Users\crist>pip install networkx
Requirement already satisfied: networkx in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\python38\site-packages (2.5)
Requirement already satisfied: decorator>=4.3.0 in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\pythonsite-packages (from networkx) (4.4.2)

C:\Users\crist>
C:\Users\crist>pip install scipy
Requirement already satisfied: scipy in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\python38\site-packages (1.6.0)
Requirement already satisfied: numpy>=1.16.5 in c:\users\crist\appdata\local\packages\pythonsoftwarefoundation.python.3.8_qbz5n2kfra8p0\localcache\local-packages\python38\site-packages (from scipy) (1.19.3)

C:\Users\crist>
```

Figura 2.7: Instalación de paquetes 2

2.4. Ejemplo de funcionamiento del software

En este apartado, se presentan los 2 módulos principales que el software maneja: **simulación del juego de la vida y generación de atractores**.

2.4.1. Life

Inicialmente, agregamos unas cuantas células al universo de tamaño 100x100 y las guardamos en un archivo .npy. Para guardar este patrón celular, se da click en el botón "save".

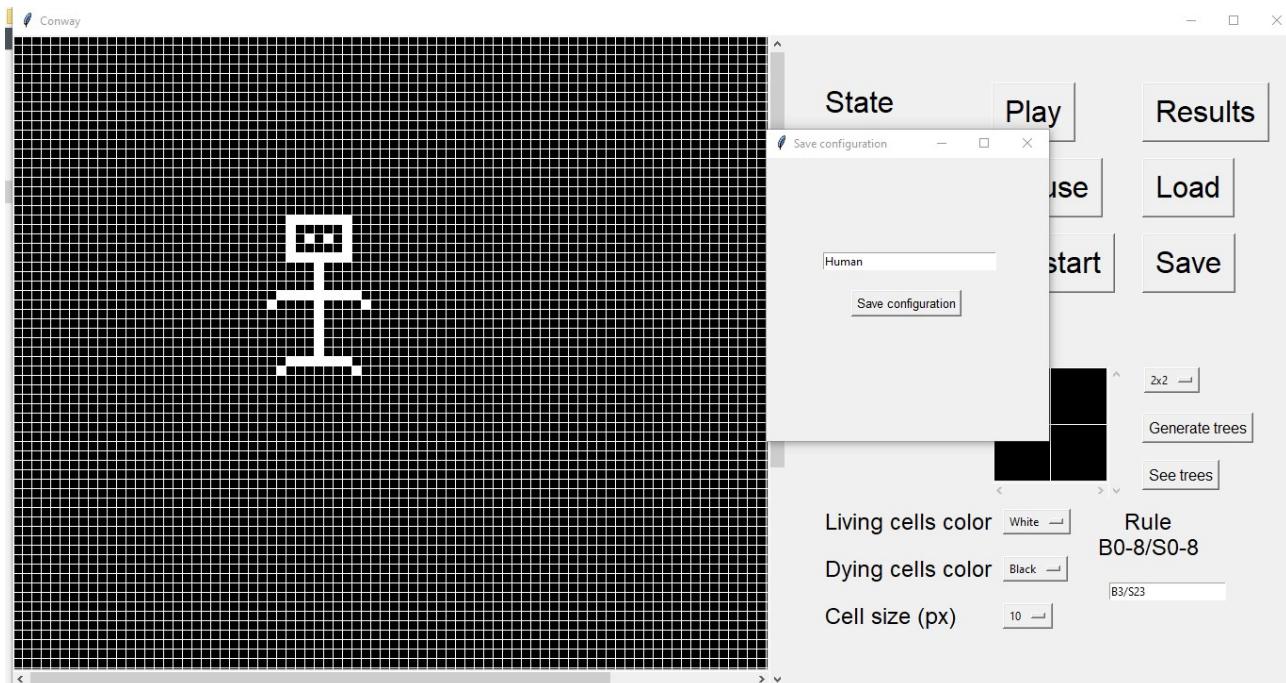


Figura 2.8: Las células vivas están de color blanco



Figura 2.9: Archivo generado llamado 'Human'

Al aplicar las reglas del juego de la vida (dar click en el botón "Play" para iniciar al juego de la vida), el patrón inicial 'Human.npy' permanece por la eternidad en la siguiente figura celular:

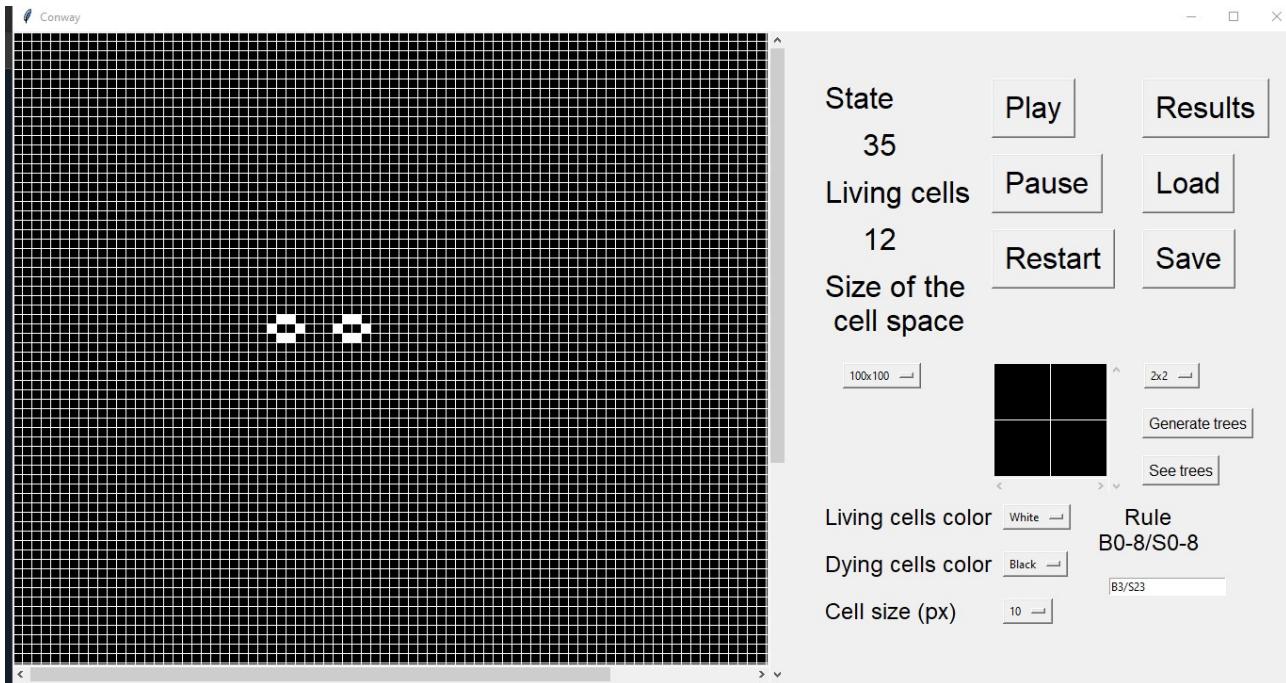


Figura 2.10: Patrón estático

El comportamiento de este patrón se puede verificar en la siguiente gráfica, que contiene la información de la relación 'célulasVivas-estado'. Esta gráfica se consigue dando click en el botón Results"

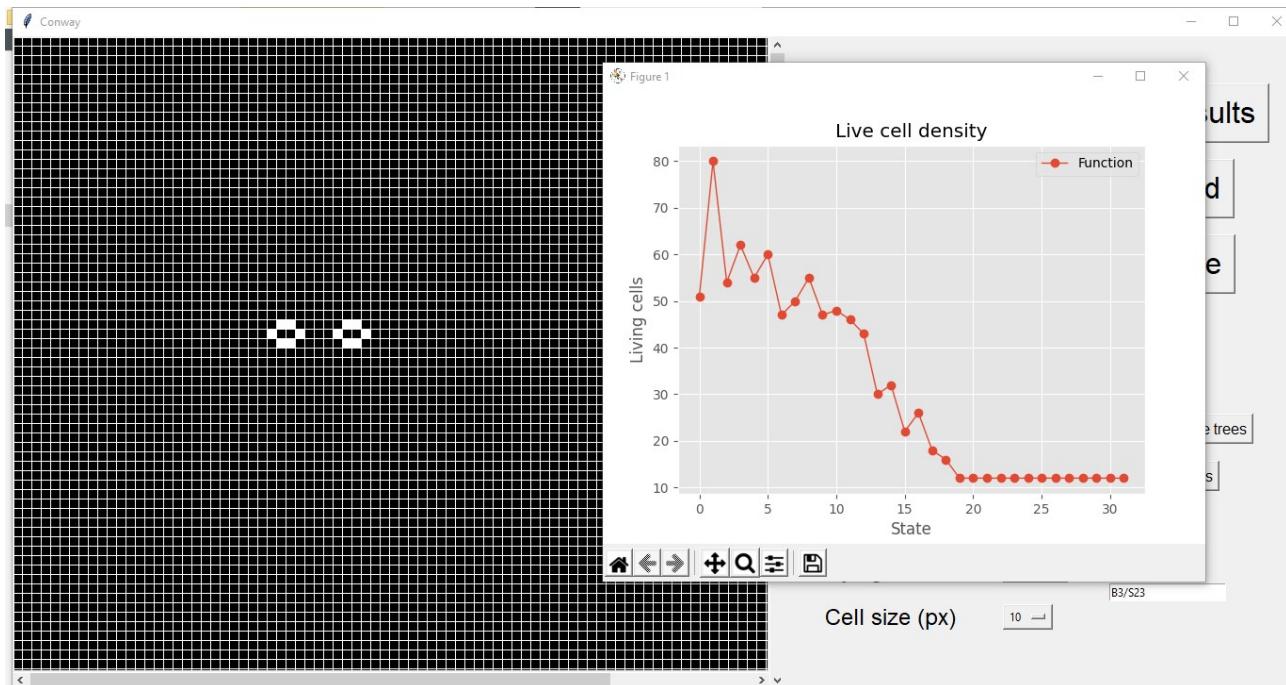


Figura 2.11: Células vivas por estado

Ahora, podemos cargar el patrón de células que se formó inicialmente dando click en el botón "Load".

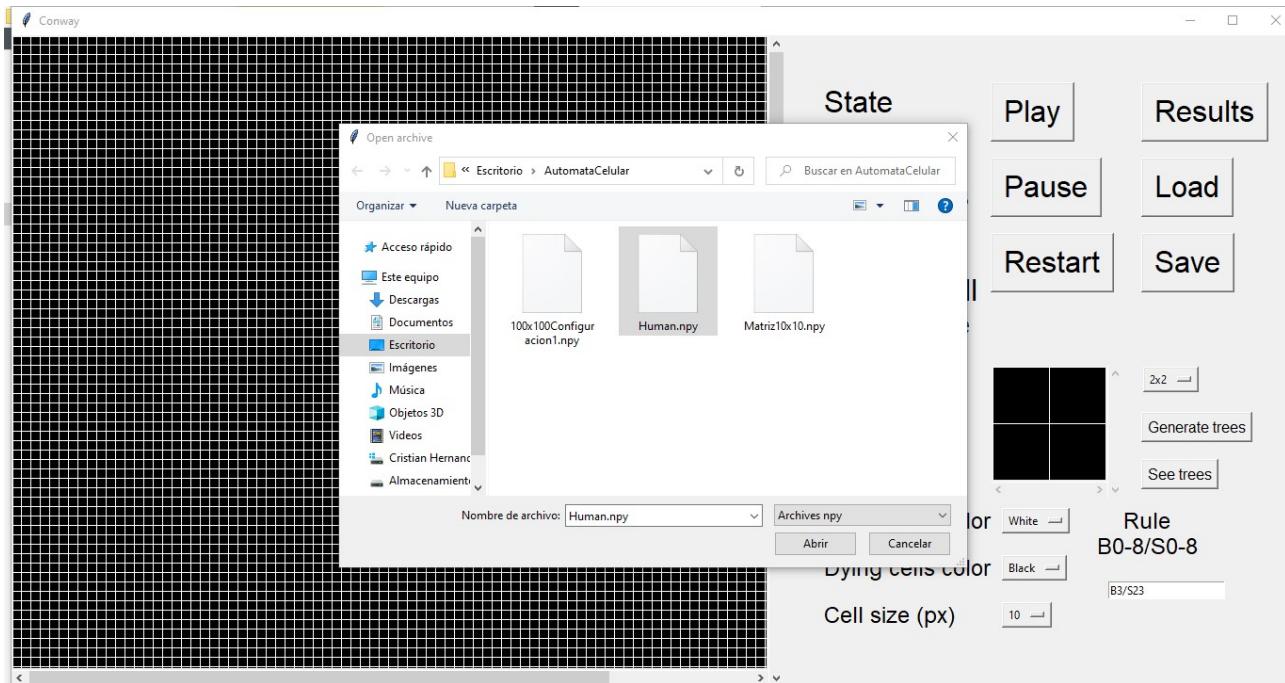


Figura 2.12: Cargar patrón

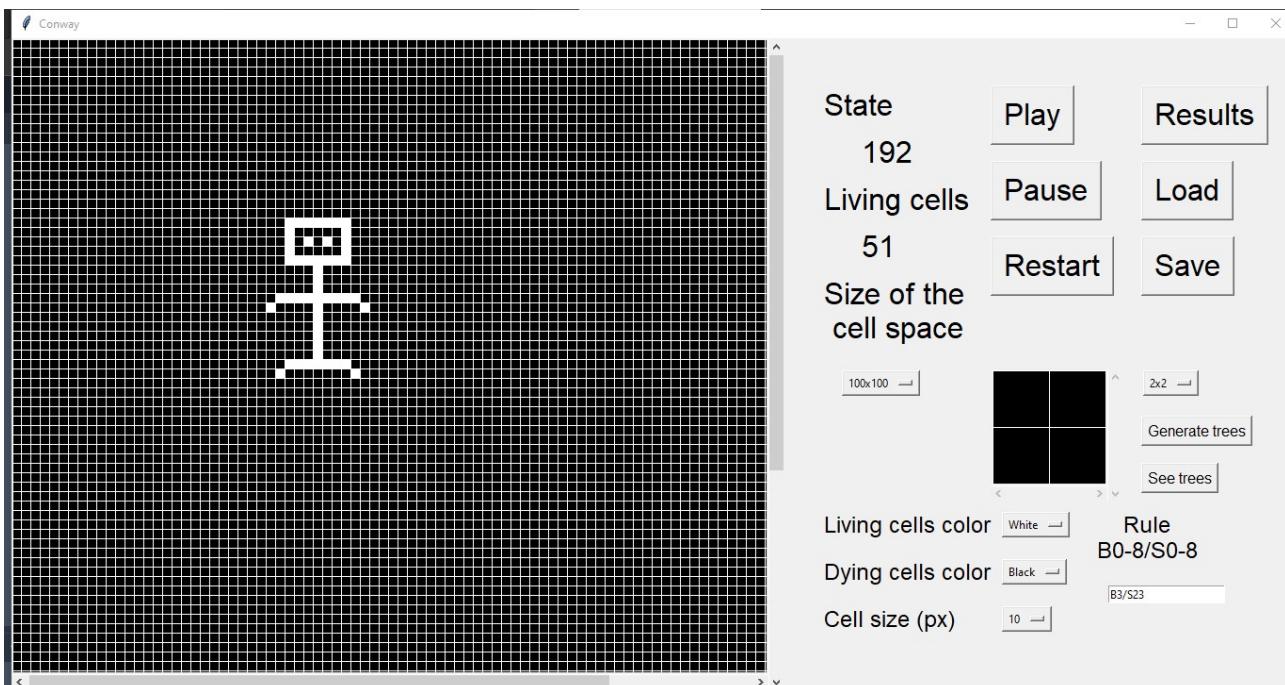


Figura 2.13: Patrón celular cargado

2.4.2. Generación de atractores

Un atractor es un punto, curva u otra figura que tiene como característica que un sistema, en el largo plazo, evoluciona hacia él; este atractor puede estar en universos planos de 2×2 , 3×3 , 4×4 y 5×5 , el cual, en este caso, será hallado aplicando las siguientes reglas:

- B3/S23
- B2/S7

Los atractores son representados mediante grafos, los cuales son generados por las funciones del paquete Networkx de python (para el caso del universo plano de 2x2) y utilizando el software llamado 'Mathematica' (para los universos planos 3x3, 4x4 y 5x5). El programa genera todos los datos necesarios y los guarda en un archivo .txt para que Mathematica pueda ocuparlos para realizar los grafos. Cuando se da click en el botón "Generate trees", generamos los atractores.

¿Cómo se generan los atractores en el programa?

La idea principal de formar los nodos que pueden ir al atractor es la siguiente: *en un universo plano $m \times m$, se establecen $2^{m \times m}$ patrones celulares, siendo estos regidos por las reglas del juego de la vida*. Un ejemplo para el lector, es el siguiente:

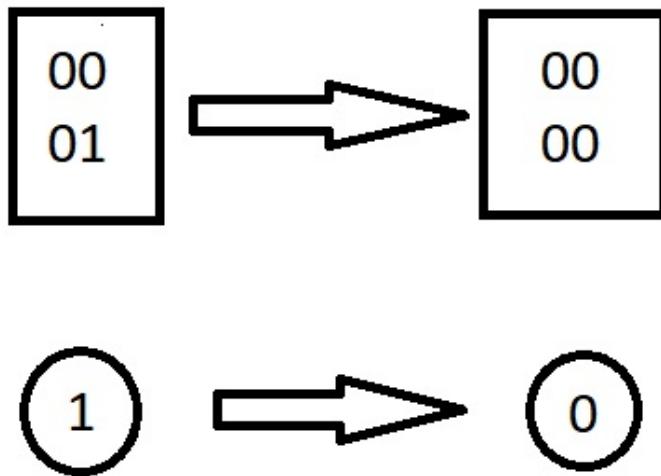


Figura 2.14: Arriba, en el universo celular 2x2, el patrón celular 0001 (en el estado 0) pasando a ser el patrón celular 0000 (en el estado 1). Abajo, la representación de ambos patrones celulares en números decimales dentro de nodos

Si se encuentra en algun estado del juego de la vida un patrón celular que ya había estado, se pasa al siguiente patrón celular, que en este caso, sería 0010.

De esta manera, el programa forma los nodos y nodos atractores. El programa los grafica para poder visualizar quienes son los diagramas de ciclos, para que después puedan ser clasificados y determinar si el sistema es caótico o complejo.

Atractores en el universo 2x2 plano

En las siguientes figuras, se pueden apreciar los atractores que existen en el universo celular 2x2 plano y a qué número decimal pertenece cada uno de ellos. Los atractores que existen aquí son los siguientes para la regla B3/S23: 0 y 15; mientras que para la regla B2/S7, los atractores en el mismo universo son: 0,12,10,9,6,5 y 3.

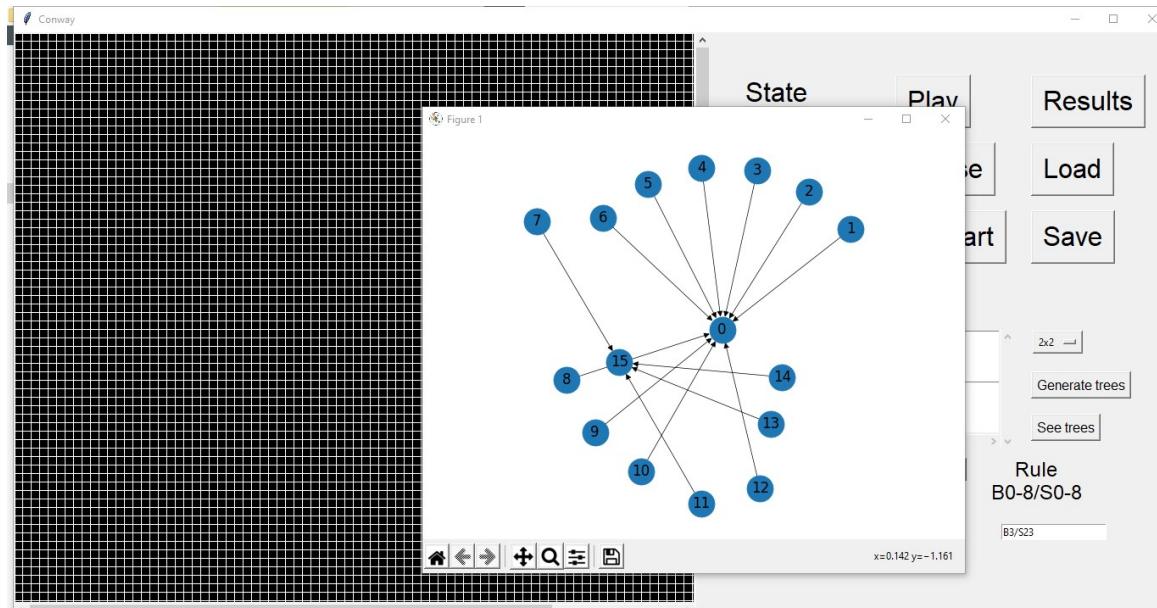


Figura 2.15: Vista de los atractores en el universo plano 2x2 con la regla B3/S23

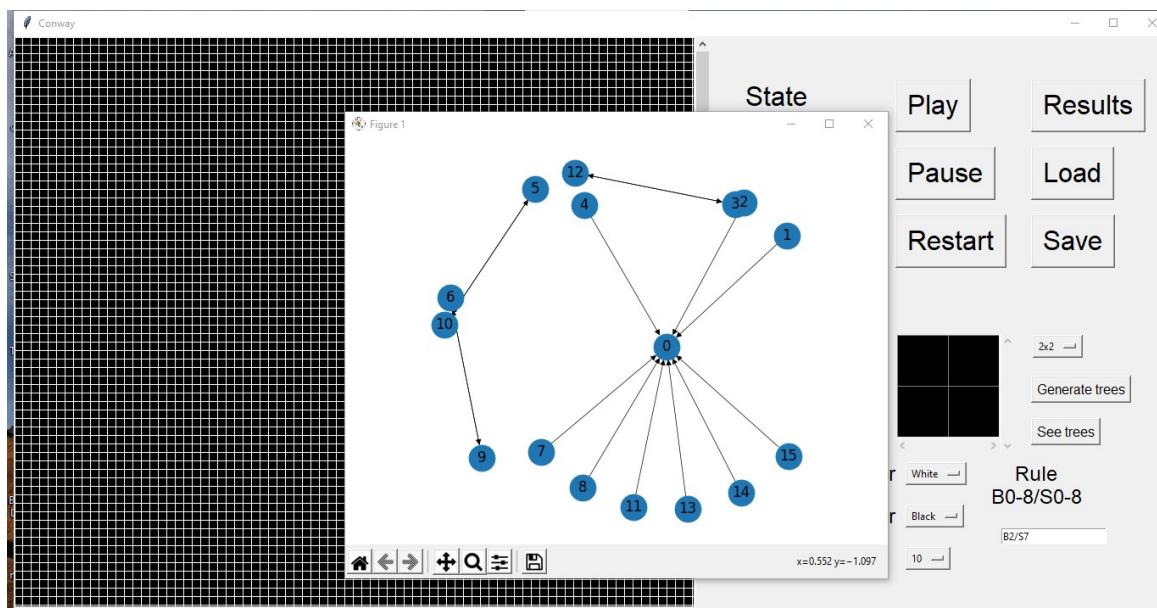


Figura 2.16: Vista de los atractores en el universo plano 2x2 con la regla B2/S7

Atractores en el universo 3x3 plano

En las siguientes figuras, se pueden apreciar los atractores que existen en el universo celular 3x3 plano.

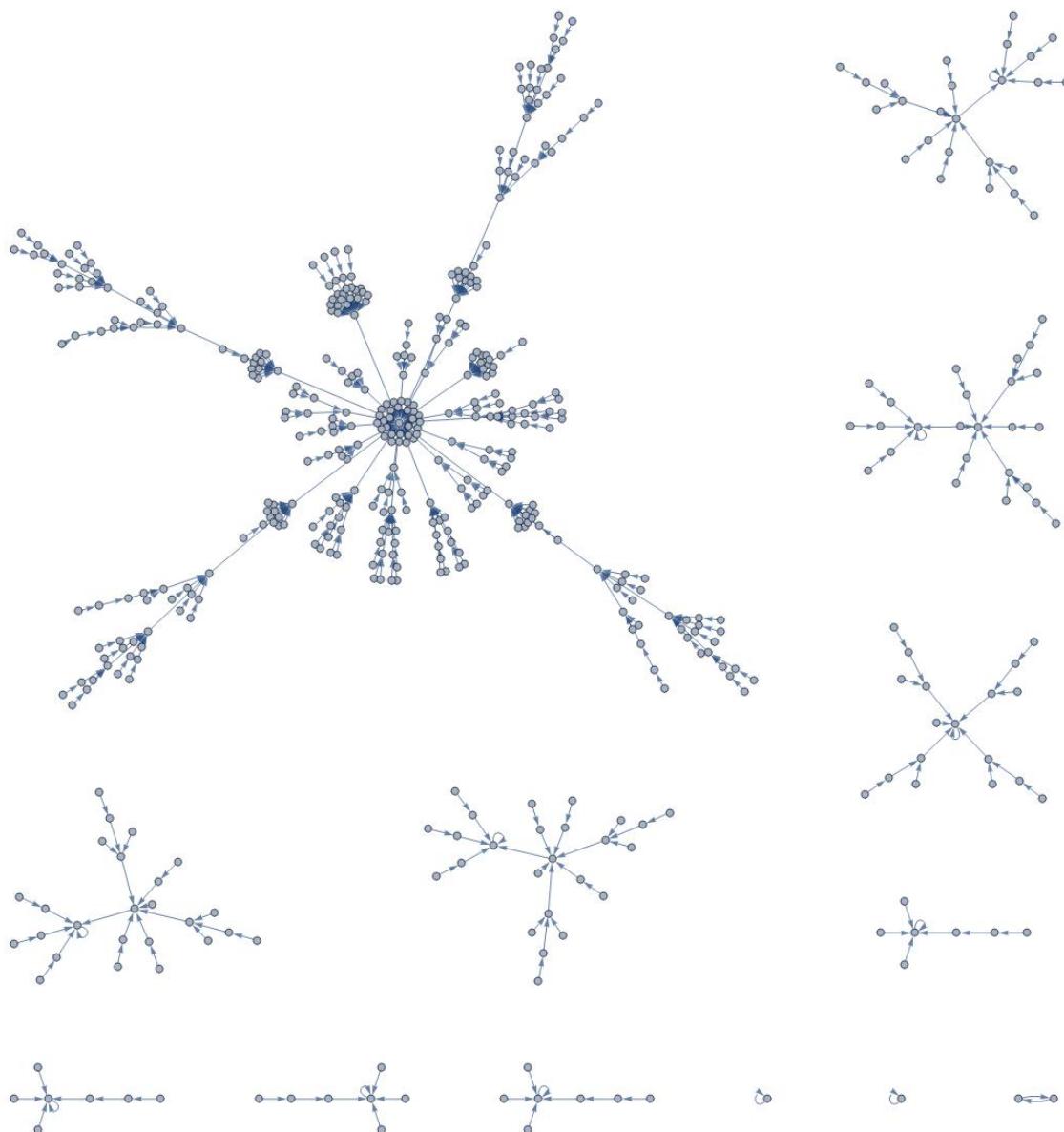


Figura 2.17: Vista de los atractores del universo plano 3x3 con la regla B3/S23

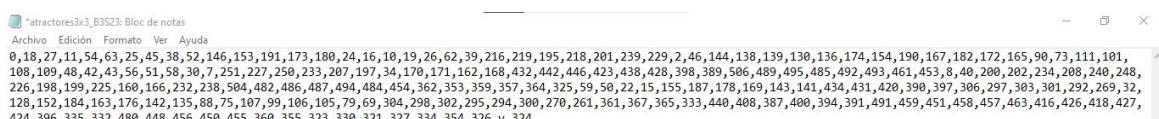


Figura 2.18: Atractores etiquetados del universo plano 3x3 con la regla B3/S23

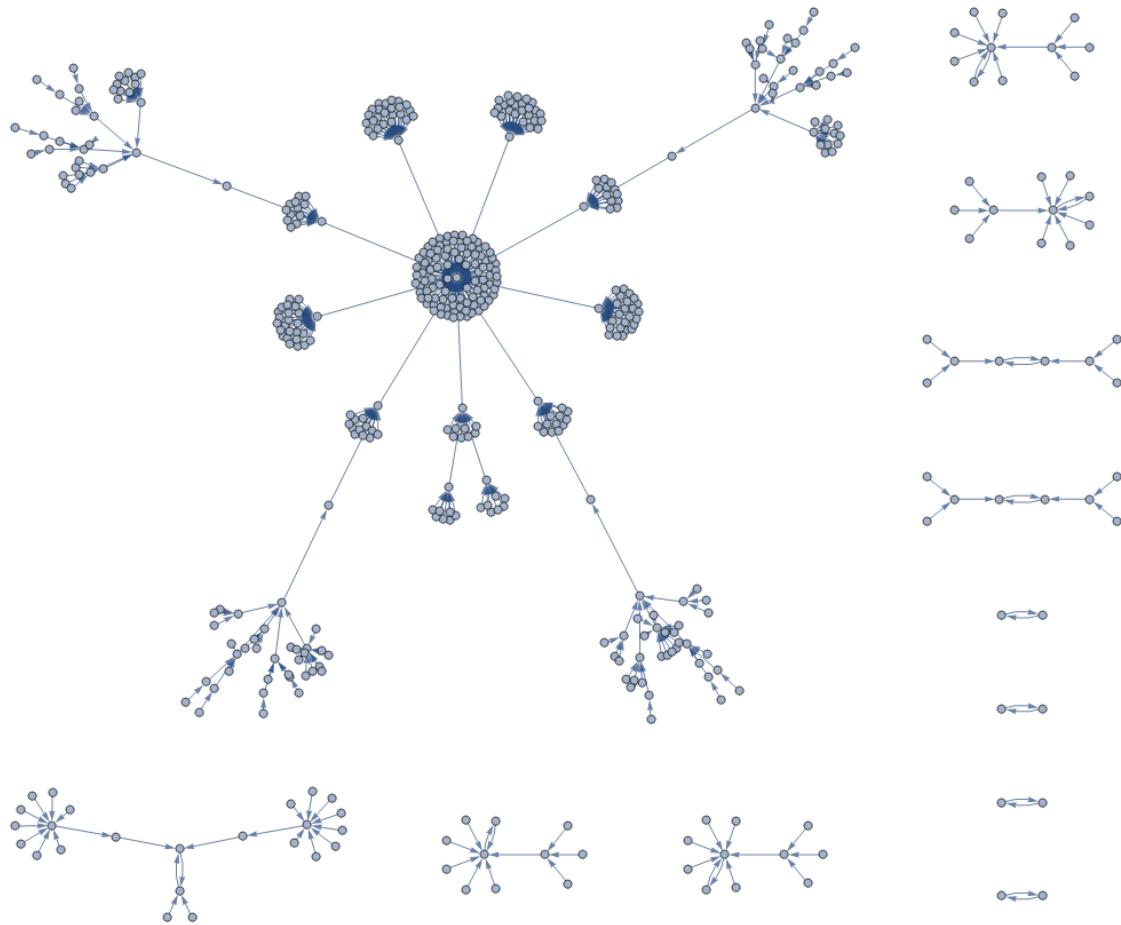


Figura 2.19: Vista de los atractores del universo plano 3x3 con la regla B2/S7

*atractores3x3_B2S7: Bloc de notas
 Archivo Edición Formato Ver Ayuda
 0, 24, 18, 48, 40, 17, 33, 32, 10, 45, 36, 34, 9, 195, 192, 228, 225, 224, 20, 12, 8, 146, 128, 133, 132, 129, 390, 396, 393, 384, 392, 325, 324, 320, 321, 16, 144, 130, 161, 160, 136, 165, 164, 170, 3, 138, 140, 2, 5, 4, 1, 270, 260, 257, 256, 264, 261, 56, 42, 80, 66, 97, 96, 98, 65, 64, 360, 354, 330, 291, 288, 272, 266, 268, 258, 322, 78, 68, 72, 290, 6, 162, 141, 137, 99, 69, 168, 35, 14 y 74

Figura 2.20: Atractores etiquetados del universo plano 3x3 con la regla B2/S7

Atractores en el universo 4x4 plano

En las siguientes figuras, se pueden apreciar los atractores que existen en el universo celular 4x4. Por la cantidad de nodos mostrados, se complica la visualización de los atractores.

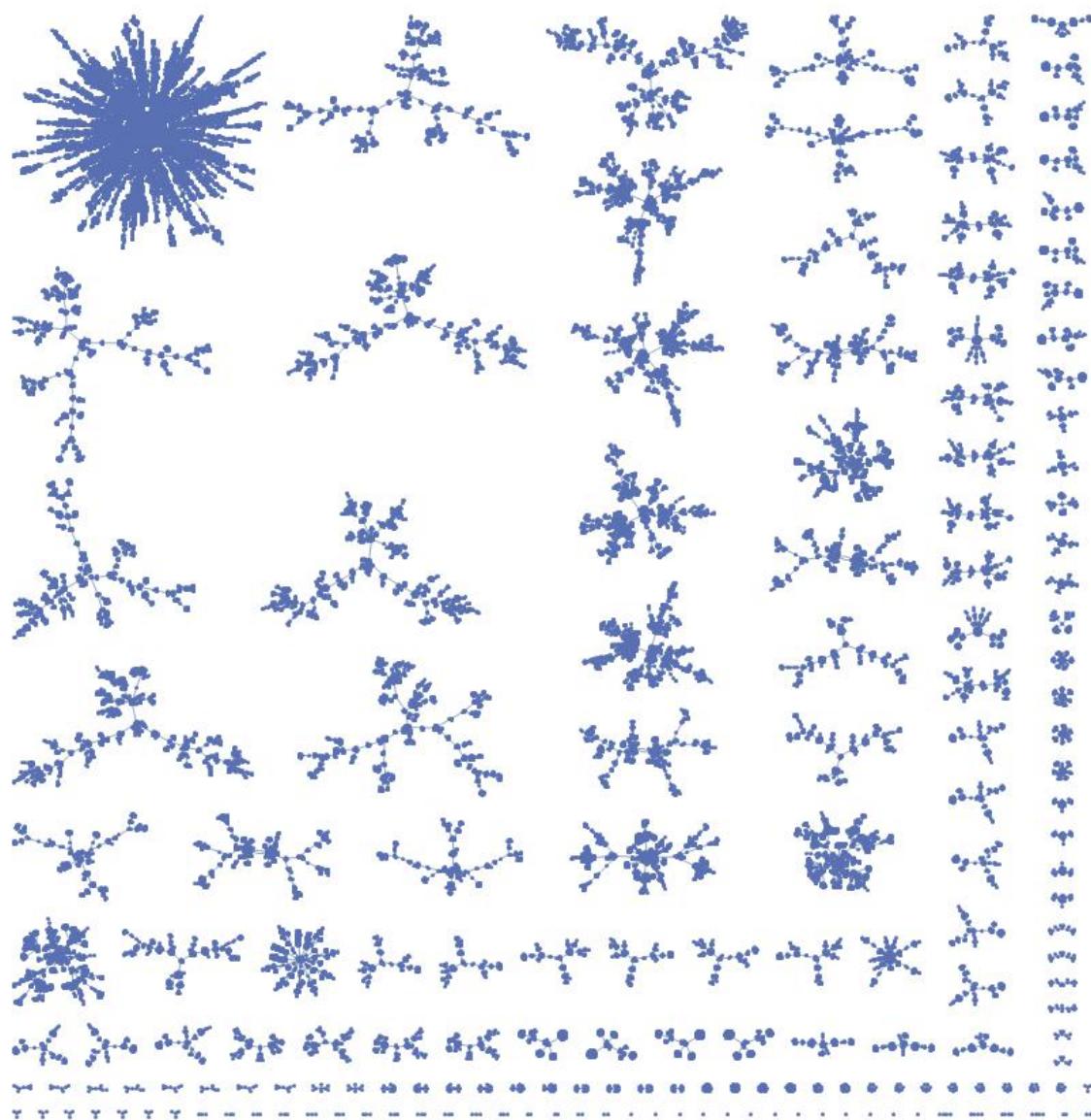


Figura 2.21: Vista de los atractores del universo plano 4x4 con la regla B3/S23

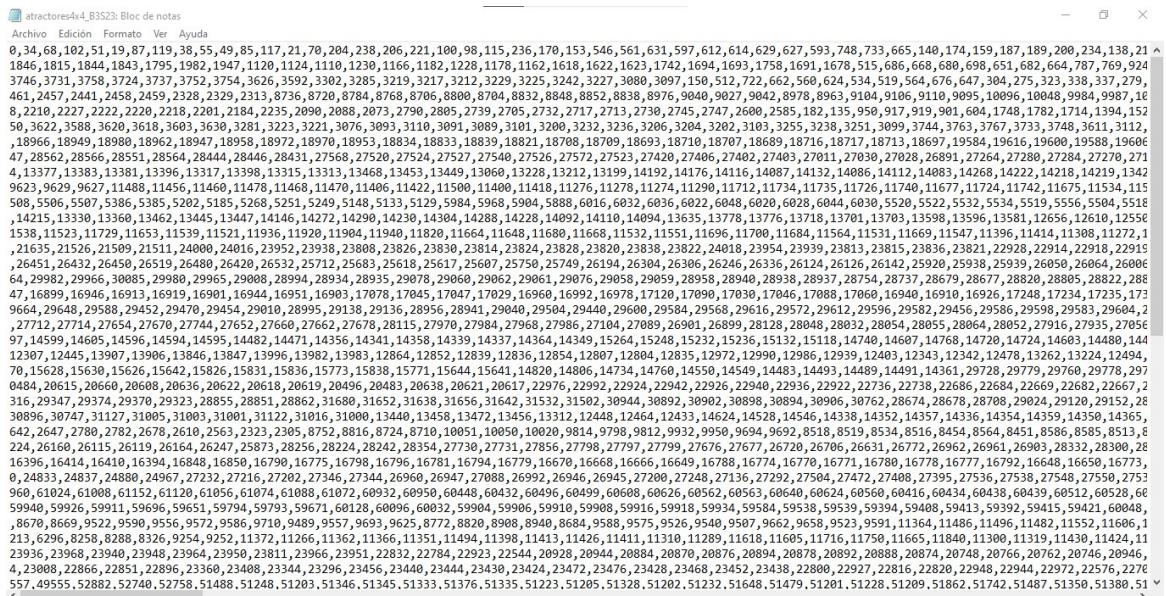


Figura 2.22: Algunos atractores etiquetados del universo plano 4x4 con la regla B3/S23

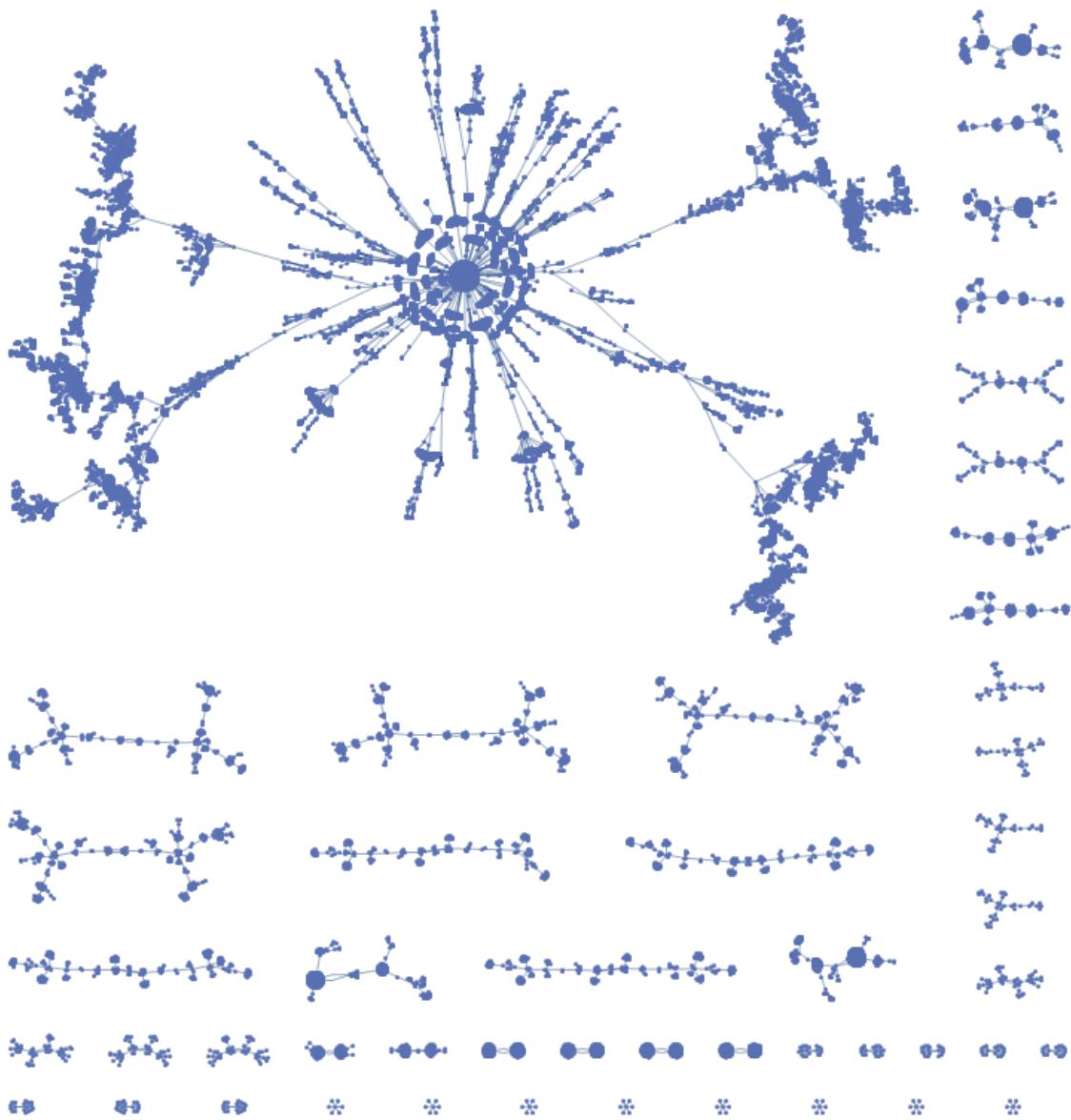


Figura 2.23: Vista de los atractores del universo plano 4x4 con la regla B2/S7

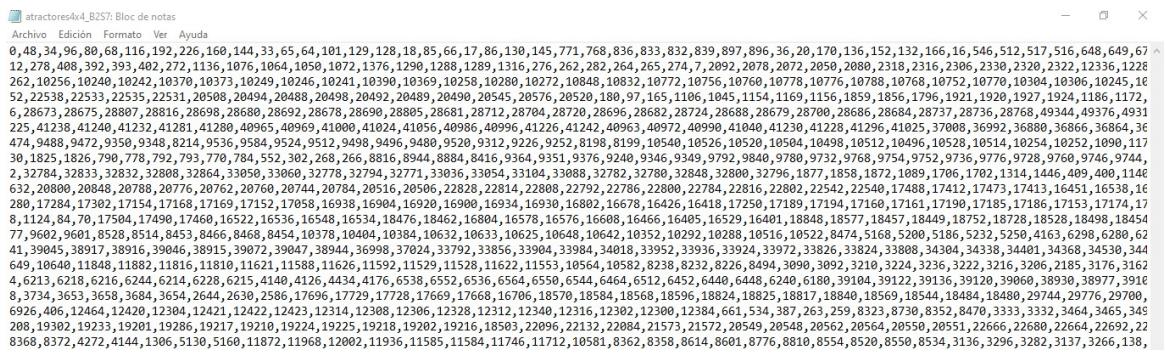


Figura 2.24: Algunos atractores etiquetados del universo plano 4x4 con la regla B2/S7

2.4.3. Clasificación de los atractores

Tomando en cuenta la clasificación Wolfram, los atractores generados en el programa (con las reglas B3/S23 y B2/S7) pertenecen a la clase IV (comportamientos complejos).

2.5. Código fuente

```

import networkx as nx
#import graphviz
import numpy as np#permite crear matrices muy grandes y usar funciones matematicas
import time
import math
import re
import os
import csv
import tkinter
import os #para obtener la ruta de directorio actual de trabajo
import matplotlib.pyplot as plt
from tkinter import messagebox
#from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
#import matplotlib.animation as animation
#agrego y quito celulas con el click izquierdo
#el boton reset solo funciona si esta en pausa el programa, solo checo si la celula es
#PRIMERO SE EJECUTA TODO LO QUE ESTA EN FUNCIONES
global colorelegidoparaCelulaMuerta,colorelegidoparaCelulaViva,var_textNpz,ventanaSalvar
global nombreArchivoNpz,juego,estadoDelJuegoNuevo,estadoDelJuego,anchoCelula,altoCelula,
global listaEstados,listaCelulasVivasPorEstado,otroProceso, posicionDeBarra,bornMin,bornM
global estadoDelJuegoNuevo2,estadoDelJuego2,anchoCelula2,altoCelula2,numFilas2,numColumnas2
global G,numeroEstado2
#Los objetos tkinter son globales. Pueden ser utilizados en cualquier funcion y no es necesario declararlos
def agregarQuitarCelula(event):#al agregar o quitar celula, actualizamos matrices y el canvas
    canvas= event.widget
    posY, posX = canvas.canvasx(event.x),canvas.canvasy(event.y)#donde se da click, obtenemos las coordenadas
    #siendo el espacioCelularCanvas un canvas desplazable, la forma de obtener las coordenadas es
    #print(posX)
    #print(posY)
    global estadoDelJuegoNuevo,estadoDelJuego,anchoCelula,altoCelula,celdulasVivas,colores
    #y, x = int(np.floor(posY/altoCelula)), int(np.floor(posX/anchoCelula))
    y, x = int(np.floor(posY/altoCelula)), int(np.floor(posX/anchoCelula))#averiguamos las coordenadas dentro del triangulo
    print(x)#salen coordenadas dentro del triangulo
    print(y)
    if estadoDelJuegoNuevo[x,y] == 0:
        estadoDelJuegoNuevo[x,y]=1
        estadoDelJuego[x,y]=1
        celulasVivas+=1
        celula = espacioCelularCanvas.create_rectangle((y*anchoCelula) ,(x*altoCelula),(y*anchoCelula)+anchoCelula,(x*altoCelula)+altoCelula)
    else:
        estadoDelJuegoNuevo[x,y]=0
        estadoDelJuego[x,y]=0
        celulasVivas-=1
        celula = espacioCelularCanvas.create_rectangle((y*anchoCelula) ,(x*altoCelula),(y*anchoCelula)+anchoCelula,(x*altoCelula)+altoCelula)

    etiquetaESTADO.config(text=etiquetaESTADO['text'])
    etiquetaNCELVIVAS.config(text=str(celulasVivas))
    #ventanaPrincipal.update()

```

```

def callback1(*args):#PARA COLOREAR CELULAS VIVAS
    global colorelegidoparaCelulaViva,estadoDelJuegoNuevo,numFilas,numColumnas
    colorelegidoparaCelulaViva=format(ColorDeCelulaViva.get())#ColorDeCelulaViva es StringVar de tkinter
    #print(colorelegidoparaCelulaViva)
    for posicionEnX in range(0,numFilas):
        for posicionEnY in range(0,numColumnas):
            #celula = espacioCelularCanvas.create_rectangle((posicionEnY*anchoCelula)
            if estadoDelJuegoNuevo[posicionEnX,posicionEnY] == 1:#dibujo un rectangulo
                celula = espacioCelularCanvas.create_rectangle((posicionEnY*anchoCelula)

def callback2(*args):#PARA COLOREAR CELULAS MUERTAS
    global colorelegidoparaCelulaMuerta,estadoDelJuegoNuevo,numFilas,numColumnas,anchoCelula
    colorelegidoparaCelulaMuerta=format(ColorDeCelulaMuerta.get())#ColorDeCelulaViva es StringVar de tkinter
    #print(colorelegidoparaCelulaViva)
    for posicionEnX in range(0,numFilas):
        for posicionEnY in range(0,numColumnas):
            #celula = espacioCelularCanvas.create_rectangle((posicionEnY*anchoCelula)
            if estadoDelJuegoNuevo[posicionEnX,posicionEnY] == 0:#dibujo un rectangulo
                celula = espacioCelularCanvas.create_rectangle((posicionEnY*anchoCelula)

def callback3(*args):#PARA CAMBIAR EL TAMAÑO DE LAS CELULAS
    global colorelegidoparaCelulaMuerta,colorelegidoparaCelulaViva,estadoDelJuegoNuevo,numeroDeFilas,numeroDeColumnas
    tamElegidoCel=format(TamCelulas.get())#ColorDeCelulaViva es StringVar de tkinter, numeroDeFilas y numeroDeColumnas son IntVar de tkinter
    anchoCelula=int(tamElegidoCel)
    altoCelula=int(tamElegidoCel)
    espacioCelularCanvas.delete(tkinter.ALL)#quitamos todas las células dentro del canvas
    espacioCelularCanvas.config(scrollregion=(0, 0, numColumnas*anchoCelula,numFilas*altoCelula))
    for posicionEnX in range(0,numFilas):
        for posicionEnY in range(0,numColumnas):
            if estadoDelJuegoNuevo[posicionEnX,posicionEnY] == 0:#dibujo un rectangulo
                celula = espacioCelularCanvas.create_rectangle((posicionEnY*anchoCelula)
            else:
                celula = espacioCelularCanvas.create_rectangle((posicionEnY*anchoCelula)
    ventanaPrincipal.update()

def callback4(*args):#PARA REPINTAR EL CANVAS DEL ESPACIO CELULAR 2 QUE REFIEREN A LOS ARBOS
    global estadoDelJuegoNuevo2,numFilas2,numColumnas2,anchoCelula2,altoCelula2,estadoDelJuegoNuevo,numeroDeFilas,numeroDeColumnas
    tamUniArb=format(tamUniversoArboles.get())#ColorDeCelulaViva es StringVar de tkinter
    #print(tamUniArb)
    espacioCelularCanvas2.delete("all")
    if (tamUniArb=="2x2"):
        numColumnas2=2
        numFilas2=2
        anchoCelula2=(120/numColumnas2)
        altoCelula2=(120/numFilas2)
    elif (tamUniArb=="3x3"):
        numColumnas2=3
        numFilas2=3
        anchoCelula2=(120/numColumnas2)

```

```

altoCelula2=(120/numFilas2)
elif (tamUniArb=="4x4"):
    numColumnas2=4
    numFilas2=4
    anchoCelula2=(120/numColumnas2)
    altoCelula2=(120/numFilas2)
elif (tamUniArb=="5x5"):
    numColumnas2=5
    numFilas2=5
    anchoCelula2=(120/numColumnas2)
    altoCelula2=(120/numFilas2)
elif (tamUniArb=="6x6"):
    numColumnas2=6
    numFilas2=6
    anchoCelula2=(120/numColumnas2)
    altoCelula2=(120/numFilas2)

##  

#ESTADO INICIAL de los arboles. LOS REPINTAMOS  

estadoDelJuego2=np.zeros((numFilas2,numColumnas2))
estadoDelJuegoNuevo2=np.copy(estadoDelJuego2)
numeroEstado2=0
celulasVivas2=0
estadoDelJuegoNuevo2=np.copy(estadoDelJuego2)
for posicionEnX in range(0,numFilas2):
    for posicionEnY in range(0,numColumnas2):
        #celula = espacioCelularCanvas.create_rectangle((posicionEnY*anchoCelula)
        if estadoDelJuego2[posicionEnX,posicionEnY] == 0:#dibujo un rectangulo negro
            celula = espacioCelularCanvas2.create_rectangle((posicionEnY*anchoCelula
        else:
            celulasVivas2+=1
            celula = espacioCelularCanvas2.create_rectangle((posicionEnY*anchoCelula

def callback5(*args):
    global estadoDelJuegoNuevo,numFilas,numColumnas,anchoCelula,altoCelula,estadoDelJuegoNuevo
    tamUniLife=format(tamUniversoLife.get())
    espacioCelularCanvas.delete("all")

    if (tamUniLife=="100x100"):
        numFilas=100
        numColumnas=100
    elif (tamUniLife=="500x500"):
        numFilas=500
        numColumnas=500
    elif (tamUniLife=="1000x1000"):
        numFilas=1000
        numColumnas=1000
    elif (tamUniLife=="5000x5000"):
        numFilas=5000

```

```

        numColumnas=5000
    elif (tamUniLife=="10000x10000"):
        numFilas=10000
        numColumnas=10000

espacioCelularCanvas.config(scrollregion=(0, 0, numColumnas*anchoCelula,numFilas*altoCelula))
#####SCROLL fin
estadoDelJuego=np.zeros((numFilas,numColumnas))
estadoDelJuegoNuevo=np.copy(estadoDelJuego)
celulasVivas=0

for posicionEnX in range(0,numFilas):
    for posicionEnY in range(0,numColumnas):
        #celula = espacioCelularCanvas.create_rectangle((posicionEnY*anchoCelula, posicionEnX*altoCelula, (posicionEnY+1)*anchoCelula, (posicionEnX+1)*altoCelula))
        if estadoDelJuego[posicionEnX,posicionEnY] == 0:#dibujo un rectangulo vacio
            celula = espacioCelularCanvas.create_rectangle((posicionEnY*anchoCelula, posicionEnX*altoCelula, (posicionEnY+1)*anchoCelula, (posicionEnX+1)*altoCelula), fill="white")
        else:
            celulasVivas+=1
            celula = espacioCelularCanvas.create_rectangle((posicionEnY*anchoCelula, posicionEnX*altoCelula, (posicionEnY+1)*anchoCelula, (posicionEnX+1)*altoCelula), fill="black")

etiquetaNCELVIVAS=tkinter.Label(ventanaPrincipal,text=str(celulasVivas))
etiquetaNCELVIVAS.config(font=("Arial", 24))
etiquetaNCELVIVAS.place(x= 900.0,y= 200.0)
etiquetaNESTADO=tkinter.Label(ventanaPrincipal,text=str(0))
etiquetaNESTADO.config(font=("Arial", 24))
etiquetaNESTADO.place(x= 900.0,y= 100.0)
ventanaPrincipal.update()

def accionBtnPausar():
    global juego,otroProceso#especificamos que es global estas variables
    if juego==True and otroProceso==True:
        juego=False
    otroProceso=False#esta variable controla los botones
    #campoRegla.configure(state=tkinter.NORMAL)

def accionBtnCargar():
    global juego, celulasVivas,estadoDelJuegoNuevo,estadoDelJuego,anchoCelula,altoCelula
    if (not juego) and (not otroProceso):#si esta en pausa el juego. os.getcwd() contiene la ruta actual
        archivoNpy=tkinter.filedialog.askopenfilename(title="Open archive",initialdir=os.getcwd())
        #print(archivoNpy)
        #el archivo Npy contiene una ruta hacia un archivo .npy, que en realidad se trataria de una matriz
        matrizConfigurada=np.load(archivoNpy)#cargamos ese archivoNPY a una matriz

```

```

for posicionEnX in range(0,numFilas):
    for posicionEnY in range(0,numColumnas):
        #celula = espacioCelularCanvas.create_rectangle((posicionEnY*anchoCelula, posicionEnX*altoCelula, (posicionEnY+1)*anchoCelula, (posicionEnX+1)*altoCelula))
        if matrizConfigurada[posicionEnX,posicionEnY] == 1:#dibujo un rectangulo
            celulasVivas+=1
            celula = espacioCelularCanvas.create_rectangle((posicionEnY*anchoCelula, posicionEnX*altoCelula, (posicionEnY+1)*anchoCelula, (posicionEnX+1)*altoCelula))
            estadoDelJuegoNuevo[posicionEnX,posicionEnY]=1
            estadoDelJuego[posicionEnX,posicionEnY]=1
etiquetaNCELVIVAS.config(text=str(celulasVivas))

def accionBtnGuardar():#de la ventanaPrincipal
    global juego,otroProceso,celulasVivas
    if (not juego) and (not otroProceso) and (celulasVivas>0):#abrimos una ventana extra
        #ventanaPrincipal.withdraw()
        otroProceso=True
        global ventanaSalvarConfiguracion,var_textNpz
        ventanaSalvarConfiguracion=tkinter.Toplevel(ventanaPrincipal)#ventana principal
        ventanaSalvarConfiguracion.title("Save configuration")
        ventanaSalvarConfiguracion.geometry("300x300")
        ventanaSalvarConfiguracion.protocol("WM_DELETE_WINDOW",accionCerrarVentanaSalvarConfiguracion)
        var_textNpz=tkinter.StringVar()# el texto ingresado en campo se va a guardar en el archivo
        campo=tkinter.Entry(ventanaSalvarConfiguracion,textvariable=var_textNpz,width=30)
        botonGuardarConfiguracion=tkinter.Button(ventanaSalvarConfiguracion,command=lambda:accionGuardarNpy())
        campo.pack()
        botonGuardarConfiguracion.pack()

def accionGuardarNpy():#ssi doy click en el boton de la ventana de salvar configuracion
    global nombreArchivoNpz,otroProceso,estadoDelJuegoNuevo,var_textNpz
    otroProceso=False
    #print(nombreConf.get())
    nombreArchivoNpz=var_textNpz.get()
    print(nombreArchivoNpz)
    np.save(nombreArchivoNpz,estadoDelJuegoNuevo)
    ventanaSalvarConfiguracion.destroy()

def accionCerrarVentanaSalvarConfiguracion():#ssi se cierra la ventana de salvar configuracion
    global otroProceso,ventanaSalvarConfiguracion
    otroProceso=False
    ventanaSalvarConfiguracion.destroy()

def realizarGraficas():
    global listaEstados,listaCelulasVivasPorEstado,otroProceso#usamos las variables globales
        #plt es una grafia (plano cartesiano)
    plt.style.use('ggplot')
    plt.title("Live cell density")
    plt.xlabel("State")
    plt.ylabel("Living cells")
    plt.plot(listaEstados,listaCelulasVivasPorEstado,"o-",linewidth=1,label="Function")#
    #plt.plot(listaEstados,listaCelulasVivasPorEstado,linewidth=1,label="Function")
    #plt.plot(listaEstados,listaCelulasVivasPorEstado)
    plt.legend()#se refiere a la funcion

```

```

plt.ion()
plt.show()
"""

plt.style.use('ggplot')
fig, (ax1,ax2) = plt.subplots(2)
fig.suptitle('Results')
ax1.plot(listaEstados, listaCelulasVivasPorEstado)
ax1.set_title("Live cell density")
ax1.set_ylabel("Living cells")
listaMedia=[]
tamListaCelulasVivasPorEstado=0
for i in listaCelulasVivasPorEstado:
    tamListaCelulasVivasPorEstado+=1

for i1 in range(0,tamListaCelulasVivasPorEstado):
    i2=0
    sumaParcial=0#media del actual estado
    mediaActual=0
    while i2<=i1:
        sumaParcial+=listaCelulasVivasPorEstado[i2]
        i2+=1

    mediaActual=(sumaParcial/i2)
    listaMedia.append(mediaActual)

ax2.plot(listaEstados, listaMedia)
ax2.set_title("Media")
ax2.set_xlabel("State")
ax2.set_ylabel("Media")
plt.show()
"""

def accionBtnReiniciar():#borramos celulas cuando juego = false, actualizamos las matrices
    global numFilas,numColumnas,estadoDelJuegoNuevo,estadoDelJuego,anchoCelula,altoCelula
    if (not juego) and (not otroProceso):#if juego==False:
        for posicionEnX in range(0,numFilas):
            for posicionEnY in range(0,numColumnas):
                #celula = espacioCelularCanvas.create_rectangle((posicionEnY*anchoCelula,altoCelula*(posicionEnX+1),posicionEnY*anchoCelula+anchoCelula,altoCelula*(posicionEnX+1)+altoCelula))
                if estadoDelJuegoNuevo[posicionEnX,posicionEnY] == 1:#dibujo un rectangulo
                    celula = espacioCelularCanvas.create_rectangle((posicionEnY*anchoCelula,altoCelula*(posicionEnX+1),posicionEnY*anchoCelula+anchoCelula,altoCelula*(posicionEnX+1)+altoCelula))
                    estadoDelJuegoNuevo[posicionEnX,posicionEnY]=0
                    estadoDelJuego[posicionEnX,posicionEnY]=0
                    celulasVivas-=1

    etiquetaNCELVIVAS.config(text=str(celulasVivas))#esto forma parte del if
    ventanaPrincipal.update()

```

```

def accionBtnResultados():
    global juego
    if (not juego):#si juego es false (pausa el programa), se niega y juego es verdadero
        realizarGraficas()

def accionBtnReproducir():#Ejecuta el juego de la vida. Por cada iteracion, se actualiza
    global juego,otroProceso
    reglaValida=False
    if(len(regla.get())>0):
        reglaValida=validar(regla.get())
        if(reglaValida==False):
            messagebox.showinfo("Warning", "The rule is invalid")
    else:
        reglaValida=False
        messagebox.showinfo("Warning", "The string is empty")

    if (not juego) and (not otroProceso) and reglaValida:
        global colorelegidoparaCelulaMuerta,colorelegidoparaCelulaViva,estadoDelJuegoNuevo
        juego=True
        otroProceso=True
        informacionNacimiento=""
        informacionSupervivencia=""
        #print("La cadena "+cadenaEnMayusculas+" es valida")

        #obtenemos cadena para validar el nacimiento de una celula muerta (Extraemos los caracteres)
        cadenaEnMayusculas=regla.get().upper()
        #messagebox.showinfo("Mayusculas", cadenaEnMayusculas)
        informacionBornInicio=cadenaEnMayusculas[1]
        informacionBornFin=cadenaEnMayusculas[posicionDeBarra-1]

        if(informacionBornInicio=='/' and informacionBornFin=='B'):
            #bornMin=1
            #bornMax=8
            informacionNacimiento="12345678"
        else:
            #bornMin=(ord(informacionBornInicio)-48)
            #bornMax=(ord(informacionBornFin)-48)
            informacionNacimiento=cadenaEnMayusculas[1:posicionDeBarra] #extraccion de subcadenas

        #print("\nBorn minimo: ",bornMin)
        #print("\nBorn maximo: ",bornMax)
        #messagebox.showinfo("Mayusculas", cadenaEnMayusculas)
        #obtenemos cadena para validar la supervivencia de una celula viva (Extraemos los caracteres)
        posicionDeS=posicionDeBarra+1
        if((posicionDeS+1)==len(cadenaEnMayusculas)):#por ejemplo b26/s
            #supervivMin=1
            #supervivMax=8
            informacionSupervivencia="12345678"

```

```

else:
    informacionSupervivencia=cadenaEnMayusculas[(posicionDeS+1):] #extraigo lo q

#messagebox.showinfo("Cadena 1", informacionNacimiento)
#messagebox.showinfo("Cadena 2", informacionSupervivencia)
#print("\nSuperviv minimo: ",supervivMin)
#print("\nSuperviv maximo: ",supervivMax)
#print("\n\nFuncion de transicion\n\n")
#print("f("+str(bornMin)+", "+str(bornMax)+", "+str(supervivMin)+", "+str(superviv

#campoRegla.config(state='disabled')
while juego:
    #celulasVivas=0
    espacioCelularCanvas.delete(tkinter.ALL)#clave para que sea rapido el progr
    if numeroEstado==0:
        #Dibujamos celulas
        estadoDelJuegoNuevo=np.copy(estadoDelJuego)
        celulasVivas=0
        for posicionEnX in range(0,numFilas):
            for posicionEnY in range(0,numColumnas):
                #celula = espacioCelularCanvas.create_rectangle((posicionEnY*a
                if estadoDelJuego[posicionEnX,posicionEnY] == 0:
                    celula = espacioCelularCanvas.create_rectangle((posicionEnY*
                else:
                    celulasVivas+=1
                    celula = espacioCelularCanvas.create_rectangle((posicionEnY*
    #print(posicionEnX)#ESTO FORMA PARTE DEL PRIMER FOR
else:
    estadoDelJuegoNuevo=np.copy(estadoDelJuego)
    #print("hola")
    celulasVivas=0
    #Visitamos cada posicion de la matriz "estadoDelJuego"
    for posicionEnX in range(0,numFilas):
        for posicionEnY in range(0,numColumnas):
            #contamos vecinos vivos
            vecinasVivas= 0
            limiteinferiorY=0
            limitesuperiorY=0
            if (posicionEnY+1)==numColumnas:#es el ultimo
                limitesuperiorY=(posicionEnY)
            else:
                limitesuperiorY=(posicionEnY+1)

            if (posicionEnY-1)>=0:
                limiteinferiorY=(posicionEnY-1)
            else:
                limiteinferiorY=(posicionEnY)

            #analizamos celulas de en medio

```

```

itposY=limiteinferiorY
while itposY<=limitesuperiorY:
    if (itposY!=posicionEnY):
        vecinasVivas=(vecinasVivas+estadoDelJuego[(posicionEnX)]
itposY+=1

#analizamos celulas de arriba
if (posicionEnX-1)>=0:
    itposY=limiteinferiorY
    while itposY<=limitesuperiorY:
        vecinasVivas=(vecinasVivas+estadoDelJuego[(posicionEnX-1)
itposY+=1

#analizamos celulas de abajo
if (posicionEnX+1)<numFilas:
    itposY=limiteinferiorY
    while itposY<=limitesuperiorY:
        vecinasVivas=(vecinasVivas+estadoDelJuego[(posicionEnX+1)
itposY+=1

#FIN DEL CONTEO DE LAS CELULAS
#REGLAS
#tendria que cambiar por igual al ir analizando cadenas B/S (e
#Creo que voy a partir la cadena BNUMEROS Y SNUMEROS
#Convierto vecinasVivas a cadena para usar la funcion re.search
vecinasVivas=int(vecinasVivas)#vecinas vivas al principio es un
vecViv=str(vecinasVivas)
#messagebox.showinfo("Inf", vecViv)
#messagebox.showinfo("Inf", informacionNacimiento)
#messagebox.showinfo("Inf", informacionSupervivencia)
numeroEncontradoNac=re.search(vecViv,informacionNacimiento) #num
numeroEncontradoSup=re.search(vecViv,informacionSupervivencia) #n
#print(numeroEncontradoNac)
#print("\n")
#print(numeroEncontradoSup)
#print("\n")
#Reglas
#messagebox.showinfo("Inf", numeroEncontradoNac)
#messagebox.showinfo("Inf", numeroEncontradoSup)
if estadoDelJuego[posicionEnX,posicionEnY]==0:#si la celula est
    if(numeroEncontradoNac is not None):#y si se hallo el numero
        estadoDelJuegoNuevo[posicionEnX,posicionEnY]=1
    elif estadoDelJuego[posicionEnX,posicionEnY]==1:#si la celula e
        if(numeroEncontradoSup is None):#y si no se hallo el numero
            estadoDelJuegoNuevo[posicionEnX,posicionEnY]=0

#FIN REGLAS
#Creamos celulas y las coloreamos, basandonos en la la matriz
if estadoDelJuegoNuevo[posicionEnX,posicionEnY]==0:

```

```

        celula = espacioCelularCanvas.create_rectangle((posicionEnY*50+50, posicionEnX*50+50, posicionEnY*50+150, posicionEnX*50+150))
    else:
        celulasVivas+=1
        celula = espacioCelularCanvas.create_rectangle((posicionEnY*50+50, posicionEnX*50+50, posicionEnY*50+150, posicionEnX*50+150))

#Luego, actualizamos datos
etiquetaNESTADO.config(text=str(numeroEstado))
etiquetaNCELVIVAS.config(text=str(celulasVivas))
#Llenamos array para las graficas
listaEstados.append(numeroEstado)
listaCelulasVivasPorEstado.append(celulasVivas)
numeroEstado+=1
estadoDelJuego=np.copy(estadoDelJuegoNuevo)
"""
plt.style.use('ggplot')
plt.title("Live cell density")
plt.xlabel("State")
plt.ylabel("Living cells")
#plt.plot(listaEstados,listaCelulasVivasPorEstado,"o-", linewidth=1, label="Live cells")
plt.scatter(listaEstados,listaCelulasVivasPorEstado,color="r")#
plt.pause(0.05)
plt.plot(listaEstados,listaCelulasVivasPorEstado,color="b", linewidth=1, label="Dead cells")
#plt.legend()#se refiere a la funcion
plt.ion()#activa el modo interactivo. La grafica tiene el modo interactivo
plt.show()#mostramos el grafico que estara siempre

"""
ventanaPrincipal.update()
#espacioContenedorCelular.update()

def accionBtnReproducir2():
    global juego, otroProceso, numFilas2, numColumnas2, G, numeroEstado2

    reglaValida=False
    if(len(regla.get())>0):
        reglaValida=validar(regla.get())
        if(reglaValida==False):
            messagebox.showinfo("Warning", "The rule is invalid")
    else:
        reglaValida=False
        messagebox.showinfo("Warning", "The string is empty")

    if (not juego) and (not otroProceso) and reglaValida:
        global colorelegidoparaCelulaMuerta,colorelegidoparaCelulaViva,estadoDelJuegoNuevo
        juego=True
        otroProceso=True
        informacionNacimiento=""
        informacionSupervivencia=""
        #print("La cadena "+cadenaEnMayusculas+" es valida")

```

```

#obtenemos cadena para validar el nacimiento de una celula muerta (Extraemos la cadenaEnMayusculas=regla.get().upper()
#messagebox.showinfo("Mayusculas", cadenaEnMayusculas)
informacionBornInicio=cadenaEnMayusculas[1]
informacionBornFin=cadenaEnMayusculas[posicionDeBarra-1]

if(informacionBornInicio=='/' and informacionBornFin=='B'):
    #bornMin=1
    #bornMax=8
    informacionNacimiento="12345678"
else:
    #bornMin=(ord(informacionBornInicio)-48)
    #bornMax=(ord(informacionBornFin)-48)
    informacionNacimiento=cadenaEnMayusculas[1:posicionDeBarra] #extraccion de subcadenas

#print("\nBorn minimo: ",bornMin)
#print("\nBorn maximo: ",bornMax)
#messagebox.showinfo("Mayusculas", cadenaEnMayusculas)
#obtenemos cadena para validar la supervivencia de una celula viva (Extraemos la posicionDeS=posicionDeBarra+1
if((posicionDeS+1)==len(cadenaEnMayusculas)):#por ejemplo b26/s
    #supervivMin=1
    #supervivMax=8
    informacionSupervivencia="12345678"
else:
    informacionSupervivencia=cadenaEnMayusculas[(posicionDeS+1):] #extraigo lo que sigue

#messagebox.showinfo("Cadena 1", informacionNacimiento)
#messagebox.showinfo("Cadena 2", informacionSupervivencia)
#print("\nSuperviv minimo: ",supervivMin)
#print("\nSuperviv maximo: ",supervivMax)
#print("\n\nFuncion de transicion\n\n")
#print("f("+str(bornMin)+", "+str(bornMax)+", "+str(supervivMin)+", "+str(supervivMax)+")")
numNodos=pow(2,numFilas2*numColumnas2)
print("Numero de nodos: "+str(numNodos))
nodoActual=0
##solo aumento el nodo cuando
G = nx.DiGraph() # crear un grafo
listaCadenasGrafo=[]
listaAtractores=[]

#numeroEstado2=0
#
#campoRegla.config(state='disabled')
rule=""
if regla.get()=="B3/S23":
    rule="B3S23"
elif regla.get()=="B2/S7":
    rule="B2S7"

```

```

else:
    rule=regla.get()

file=None
file2=None
if numNodos==16:
    file = open("D:/grafos2x2_"+rule+".txt", "w")
    file2 = open("D:/atractores2x2_"+rule+".txt", "w")
elif numNodos==512:
    file = open("D:/grafos3x3_"+rule+".txt", "w")
    file2 = open("D:/atractores3x3_"+rule+".txt", "w")
elif numNodos==65536:
    file = open("D:/grafos4x4_"+rule+".txt", "w")
    file2 = open("D:/atractores4x4_"+rule+".txt", "w")
elif numNodos==33554432:
    file = open("D:/grafos5x5_"+rule+".txt", "w")
    file2 = open("D:/atractores5x5_"+rule+".txt", "w")
#file.write("Primera linea" + os.linesep)#os.linesep es un salto de linea
#file.write("Segunda linea")
#file.close()

while nodoActual<numNodos:
    #numDecInicial=nodoActual
    #G.add_node(str(numDecInicial))
    #celulasVivas=0
    #G.add_node(str(nodoActual))
    espacioCelularCanvas2.delete(tkinter.ALL)#clave para que sea rapido el programa
    numBits=(numFilas2*numColumnas2)
    cadenaBinInicial=(format(nodoActual, '0'+str(numBits)+'b'))#convierte numero a binario
    #
    #print("Cadena bin inicial: "+str(cadenaBinInicial))

    posicionCadenaBin=0
    #ESTADO INICIAL de las matrices de los arboles
    estadoDelJuego2=np.zeros((numFilas2,numColumnas2))
    estadoDelJuegoNuevo2=np.copy(estadoDelJuego2)
    #Pongo la cadena en la matriz
    for posicionEnX in range(0,numFilas2):
        for posicionEnY in range(0,numColumnas2):
            #celula = espacioCelularCanvas.create_rectangle((posicionEnY*ancho
            if cadenaBinInicial[posicionCadenaBin] == "0":#dibujo un rectangulo
                estadoDelJuego2[posicionEnX,posicionEnY]=0
            posicionCadenaBin+=1

    else:
        estadoDelJuego2[posicionEnX,posicionEnY]=1
        posicionCadenaBin+=1

```

```

#
#Guardo
#print(estadoDelJuego2)
estadoDelJuegoNuevo2=np.copy(estadoDelJuego2)
#print(estadoDelJuegoNuevo2)#actualizada

#print("Ancho celula2: "+str(anchoCelula2))
#print("Alto celula2: "+str(altoCelula2))

numeroEstado2=0
celulasVivas2=0

#PINTAMOS CELULAS
for posicionEnX in range(0,numFilas2):
    for posicionEnY in range(0,numColumnas2):
        #celula = espacioCelularCanvas.create_rectangle((posicionEnY*ancho
        if estadoDelJuego2[posicionEnX,posicionEnY] == 0:#dibujo un rectangulo
            celula = espacioCelularCanvas2.create_rectangle((posicionEnY*ancho
        else:
            celulasVivas2+=1
            celula = espacioCelularCanvas2.create_rectangle((posicionEnY*ancho

#Hasta aqui hemos inicializado el espacio en cadenas 0000,0001,...,1111 (el
banderaContinuidad=True
listaCadenas=[] #esta lista contiene numeros decimales

#listaCadenasAnterior=np.array([])
#print("Tam lista cadenas: "+str(len(listaCadenas)))#muestra un 0

while banderaContinuidad:

    espacioCelularCanvas2.delete(tkinter.ALL)#clave para que sea rapido el

    #estadoDelJuegoNuevo2=np.copy(estadoDelJuego2)
    #print("hola")
    celulasVivas2=0
    #Visitamos cada posicion de la matriz "estadoDelJuego"
    for posicionEnX in range(0,numFilas2):
        for posicionEnY in range(0,numColumnas2):
            #contamos vecinos vivos
            vecinasVivas= 0
            limiteinferiorY=0
            limitesuperiorY=0
            if (posicionEnY+1)==numColumnas2:#es el ultimo
                limitesuperiorY=(posicionEnY)
            else:
                limitesuperiorY=(posicionEnY+1)

```

```

        if (posicionEnY-1)>=0:
            limiteinferiorY=(posicionEnY-1)
        else:
            limiteinferiorY=(posicionEnY)

#analizamos celulas de en medio
itposY=limiteinferiorY
while itposY<=limitesuperiorY:
    if (itposY!=posicionEnY):
        vecinasVivas=(vecinasVivas+estadoDelJuego2[(posicionEnX,
itposY+=1

#analizamos celulas de arriba
if (posicionEnX-1)>=0:
    itposY=limiteinferiorY
    while itposY<=limitesuperiorY:
        vecinasVivas=(vecinasVivas+estadoDelJuego2[(posicionEnX-
itposY+=1

#analizamos celulas de abajo
if (posicionEnX+1)<numFilas2:
    itposY=limiteinferiorY
    while itposY<=limitesuperiorY:
        vecinasVivas=(vecinasVivas+estadoDelJuego2[(posicionEnX+
itposY+=1

#FIN DEL CONTEO DE LAS CELULAS
#REGLAS
#tendria que cambiar por igual al ir analizando cadenas B/S (es
#Creo que voy a partir la cadena BNUMEROS Y SNUMEROS
#Convierto vecinasVivas a cadena para usar la funcion re.search
vecinasVivas=int(vecinasVivas)#vecinas vivas al principio es un
vecViv=str(vecinasVivas)
#messagebox.showinfo("Inf", vecViv)
#messagebox.showinfo("Inf", informacionNacimiento)
#messagebox.showinfo("Inf", informacionSupervivencia)
numeroEncontradoNac=re.search(vecViv,informacionNacimiento) #numero
numeroEncontradoSup=re.search(vecViv,informacionSupervivencia) #numero
#print(numeroEncontradoNac)

#print(numeroEncontradoSup)

#Reglas
#messagebox.showinfo("Inf", numeroEncontradoNac)
#messagebox.showinfo("Inf", numeroEncontradoSup)
if estadoDelJuego2[posicionEnX,posicionEnY]==0:#si la celula es
    if(numeroEncontradoNac is not None):#y si se hallo el numero
        estadoDelJuegoNuevo2[posicionEnX,posicionEnY]=1

```

```

        elif estadoDelJuego2[posicionEnX, posicionEnY]==1:#si la celula
            if(numeroEncontradoSup is None):#y si no se hallo el numero
                estadoDelJuegoNuevo2[posicionEnX, posicionEnY]=0

        #FIN REGLAS
        #Creamos celulas y las coloreamos, basandonos en la la matriz
        if estadoDelJuegoNuevo2[posicionEnX, posicionEnY]==0:
            celula = espacioCelularCanvas2.create_rectangle((posicionEnX, posicionEnY), (posicionEnX+1, posicionEnY+1), fill="white", outline="black")
        else:
            celulasVivas2+=1
            celula = espacioCelularCanvas2.create_rectangle((posicionEnX, posicionEnY), (posicionEnX+1, posicionEnY+1), fill="black", outline="black")

        print("Nodo o iteracion actual: "+str(nodoActual))
        #print("Representacion binaria: "+format(nodoActual, '0'+str(numBits)+'.'))
        #print("Matriz actual: ")
        #print(estadoDelJuego2)
        #print("Matriz nueva: ")
        #print(estadoDelJuegoNuevo2)

#agrego nodos (el estado actual y el estado nuevo, ambos que estan en
#FORMATO LAS CADENAS
cadBinEstadoActual=""
cadBinEstadoNuevo=""
for posicionEnX in range(0,numFilas2):
    for posicionEnY in range(0,numColumnas2):
        #celula = espacioCelularCanvas.create_rectangle((posicionEnY*ancho, posicionEnX*alto), (posicionEnY*ancho+ancho, posicionEnX*alto+alto), fill="white", outline="black")
        if estadoDelJuego2[posicionEnX, posicionEnY]==0:#dibujo un rectangulo blanco
            cadBinEstadoActual+="0"

        else:
            cadBinEstadoActual+="1"

#
for posicionEnX in range(0,numFilas2):
    for posicionEnY in range(0,numColumnas2):
        #celula = espacioCelularCanvas.create_rectangle((posicionEnY*ancho, posicionEnX*alto), (posicionEnY*ancho+ancho, posicionEnX*alto+alto), fill="white", outline="black")
        if estadoDelJuegoNuevo2[posicionEnX, posicionEnY]==0:#dibujo un rectangulo blanco
            cadBinEstadoNuevo+="0"

        else:
            cadBinEstadoNuevo+="1"

#recorro matrices para formar cadenas
#print("Cad bin actual estado actual: "+cadBinEstadoActual)
#print("Cad bin nuevo estado nuevo: "+cadBinEstadoNuevo)
decimalActual=int(cadBinEstadoActual,base=2)

```

```

decimalNuevo=int(cadBinEstadoNuevo,base=2)
#print("Decimal de Cad bin actual estado actual: "+str(decimalActual))
#print("Decimal de Cad bin nuevo estado nuevo: "+str(decimalNuevo))

G.add_node(str(decimalActual))
G.add_node(str(decimalNuevo))
#file.write(str(decimalActual)+"->"+str(decimalNuevo)+", ")

#Funcion para guardar el par de nodos NODOA -> NODOB en el archivo .tx
#Checo si el par de nodos NODOA -> NODOB esta en el conjunto (array o

if (str(decimalActual)+"->"+str(decimalNuevo) in listaCadenasGrafo):
    #print(str(decimalActual)+"->"+str(decimalNuevo))
    #print(listaCadenasGrafo)
    pass
else:
    file.write(str(decimalActual)+"->"+str(decimalNuevo)+", ")

#arrayParNodos=[[decimalActual,decimalNuevo]]
#writer = csv.writer(file)
#writer.writerows(arrayParNodos)
        #aniado arista (uno a uno, del estado viejo al nuevo)
G.add_edge(str(decimalActual), str(decimalNuevo))
listaCadenas.append(decimalActual)
listaCadenasGrafo.append(str(decimalActual)+"->"+str(decimalNuevo)) #ver
#print(listaCadenasGrafo)

if numColumnas2==5 or numColumnas2==4 or numColumnas2==3 or numColumnas2==2:
    #print(listaCadenas)
    if len(listaCadenas)>1:
        posLC=0
        while (posLC<len(listaCadenas)):
            if listaCadenas[posLC]==decimalActual:
                #print("se encontró un ciclo")
                if decimalActual in listaAtractores:
                    pass
                else:
                    listaAtractores.append(decimalActual)
                    file2.write(str(decimalActual)+", ")
                    banderaContinuidad=False
                    nodoActual+=1
                    break
            else:
                posLC+=1

```

```

        estadoDelJuego2=np.copy(estadoDelJuegoNuevo2)
        ventanaPrincipal.update()
        #print("-----")
        #espacioContenedorCelular.update()

    #FIN DEL WHILE INTERNO


juego=False
otroProceso=False
print("GRAFO GENERADO")
file.close()
file2.close()

def accionBtnVerArboles():
    global G
    nx.draw_kamada_kawai(G, node_size=500, width=0.5, with_labels=True, font_size=12) #
    #nx.draw_random(G, node_size=10, width=0.5, with_labels=False, font_size=8)
    #nx.circular_layout(G, scale=1, center=None, dim=2) # Dibujar el grafo G con una
    plt.axis("equal") # Redimensionar los ejes a longitudes iguales
    plt.show() # Mostrar el grado d-regular por pantalla

def validar(cadenaEnMayusculas):
    cadenaEnMayusculas=cadenaEnMayusculas.upper()
    print("Cadena a analizar: "+cadenaEnMayusculas)
    #Primera parte la cadena
    i=0
    if(cadenaEnMayusculas[i]=='B'):
        i=1
    else:#return false
        return False

    """
    caracter=ord(cadenaEnMayusculas[1])
    caracter es un tipo de dato entero. Ese entero es el valor en codigo ascii del caracter
    if(caracter>=48 and caracter<=56):
        #reglas+=1
        print("Es un numero")
    else:#return false
        valida=False
    """

```

```

textoAE incontrar="//"

textoEncontrado=re.search(textoAE incontrar,cadenaEnMayusculas)#devuelve None si no se

global posicionDeBarra

posicionDeBarra=0
if textoEncontrado is not None:
    #reglas+=1
    #print("Texto hallado //")
    posicionDeBarra=textoEncontrado.start()
    #print("Posicion: "+str(posicionDeBarra))
else:
    return False

caracterActual=ord(cadenaEnMayusculas[i])
caracterSiguiente=ord(cadenaEnMayusculas[i+1])
while(i<posicionDeBarra):
    if(cadenaEnMayusculas[i+1]=='/'):
        break
    elif((ord(cadenaEnMayusculas[i])>=48 and ord(cadenaEnMayusculas[i])<=56) and (ord(cadenaEnMayusculas[i+1])>=48 and ord(cadenaEnMayusculas[i+1])<=56)):
        i+=1
    else:
        return False

#Segunda parte de la cadena
if(cadenaEnMayusculas[posicionDeBarra+1]=='S'):
    i=posicionDeBarra+2
else:
    return False

while(i<len(cadenaEnMayusculas)):
    if(((i+1)==len(cadenaEnMayusculas) and (ord(cadenaEnMayusculas[i])>=48 and ord(cadenaEnMayusculas[i+1])<=56)) or ((ord(cadenaEnMayusculas[i])>=48 and ord(cadenaEnMayusculas[i])<=56) and (ord(cadenaEnMayusculas[i+1])>=48 and ord(cadenaEnMayusculas[i+1])<=56))):
        return True
    elif((ord(cadenaEnMayusculas[i])>=48 and ord(cadenaEnMayusculas[i])<=56) and (ord(cadenaEnMayusculas[i+1])>=48 and ord(cadenaEnMayusculas[i+1])<=56)):
        i+=1
    else:
        return False

return True

```

#FUNCIONES

#SECCION 1 DEL CODIGO. ESTO SE EJECUTA PRIMERO

```

otroProceso=False
#Inicializamos arrays para realizar las graficas
listaEstados=[]
listaCelulasVivasPorEstado=[]
#Inicializamos variables
juego=False
numeroEstado=0
#Dimensiones de la celula
anchoCelula=10
altoCelula=10
#ancho y alto deben ser las mismas
#hasta ahora solo funciona con el ancho y el alto de las celula iguales
#Creamos Celulas
numFilas=100#
numColumnas=100#
#el espacio es cuadrado numFilas = num columnas
#Creamos matriz inicializada en 0
estadoDelJuego=np.zeros((numFilas,numColumnas))
estadoDelJuegoNuevo=np.copy(estadoDelJuego)

#Creamos ventana
ventanaPrincipal=tkinter.Tk()
ventanaPrincipal.title("Conway")
ventanaPrincipal.geometry("1366x768")

#Creamos una ventana 2
espacioContenedorCelular=tkinter.Frame(ventanaPrincipal, width=800, height=670, bg='black')
espacioContenedorCelular.place(x=0.0,y=0.0)
#Creamos canvas encima de espacioContenedorCelular
espacioCelularCanvas=tkinter.Canvas(espacioContenedorCelular, width=800, height=670, bg="green")
espacioCelularCanvas.place(x=0.0,y=0.0)
#####SCROLLIN

#Creamos scrollbar horizontal y vertical al frame
sbarV = tkinter.Scrollbar(espacioContenedorCelular, orient=tkinter.VERTICAL, command=espacioCelularCanvas.yview)
sbarH = tkinter.Scrollbar(espacioContenedorCelular, orient=tkinter.HORIZONTAL, command=espacioCelularCanvas.xview)

#los posiciono en el frame
sbarV.pack(side=tkinter.RIGHT, fill=tkinter.Y)
sbarH.pack(side=tkinter.BOTTOM, fill=tkinter.X)
espacioCelularCanvas.config(yscrollcommand=sbarV.set)
espacioCelularCanvas.config(xscrollcommand=sbarH.set)
espacioCelularCanvas.pack(side=tkinter.LEFT, expand=True, fill=tkinter.BOTH)
espacioCelularCanvas.config(scrollregion=(0, 0, numColumnas*anchoCelula,numFilas*altoCelula))
#####SCROLLfin
"""

celula = espacioCelularCanvas.create_rectangle(0,0,20,20, fill = "yellow",outline="white")
celula = espacioCelularCanvas.create_rectangle(20,0,40,20, fill = "red",outline="white")
"""

```

```

#####
etiquetaMensajeColorCelulaViva=tkinter.Label(ventanaPrincipal,text="Living cells color")
etiquetaMensajeColorCelulaViva.config(font=("Arial", 18))
etiquetaMensajeColorCelulaViva.place(x=860,y=500)
#Agregamos lista de colores para celulas vivas
ColorDeCelulaViva=tkinter.StringVar(ventanaPrincipal)
ColorDeCelulaViva.set('White')
colorlegidoparaCelulaViva="White"
varioscoloresCELULASVIVASyMUERTAS=['Blue','Red','White','Green','Yellow','Gray','Pink','Black']
menuColoresCelulasVivas=tkinter.OptionMenu(ventanaPrincipal,ColorDeCelulaViva,*varioscoloresCELULASVIVASyMUERTAS)
ColorDeCelulaViva.trace("w",callback1) #ESTO FUNCIONA CUANDO ELIJO UN ITEM DEL MENU DESDE LA LISTA DE COLORES
##### esto se ejecuta siempre porque esta en el loop
etiquetaMensajeColorCelulaMuerta=tkinter.Label(ventanaPrincipal,text="Dying cells color")
etiquetaMensajeColorCelulaMuerta.config(font=("Arial", 18))
etiquetaMensajeColorCelulaMuerta.place(x=860,y=550)
#####Agregamos lista de colores de las celulas muertas
ColorDeCelulaMuerta=tkinter.StringVar(ventanaPrincipal)
ColorDeCelulaMuerta.set('Black')
colorlegidoparaCelulaMuerta="Black"
menuColoresCelulasMuertas=tkinter.OptionMenu(ventanaPrincipal,ColorDeCelulaMuerta,*varioscoloresCELULASVIVASyMUERTAS)
ColorDeCelulaMuerta.trace("w",callback2) #ESTO FUNCIONA CUANDO ELIJO UN ITEM DEL MENU DESDE LA LISTA DE COLORES
#####
etiquetaMensajeTamCelulas=tkinter.Label(ventanaPrincipal,text="Cell size (px)")
etiquetaMensajeTamCelulas.config(font=("Arial", 18))
etiquetaMensajeTamCelulas.place(x=860,y=600)
#####
TamCelulas=tkinter.StringVar(ventanaPrincipal) #Creamos una variable que guarde el valor de la celula
TamCelulas.set('10')
varioTamsParaCelulas=['1','2','3','4','5','6','7','8','9','10']
menuTamCels=tkinter.OptionMenu(ventanaPrincipal,TamCelulas,*varioTamsParaCelulas).place(x=860,y=650)
TamCelulas.trace("w",callback3)
#####

#ESTADO INICIAL
numeroEstado=0
celulasVivas=0
estadoDelJuegoNuevo=np.copy(estadoDelJuego)
for posicionEnX in range(0,numFilas):
    for posicionEnY in range(0,numColumnas):
        #celula = espacioCelularCanvas.create_rectangle((posicionEnY*anchoCelula) ,(posicionEnX*altoCelula) , (posicionEnY*anchoCelula)+anchoCelula , (posicionEnX*altoCelula)+altoCelula)
        if estadoDelJuego[posicionEnX,posicionEnY] == 0:#dibujo un rectangulo negro
            celula = espacioCelularCanvas.create_rectangle((posicionEnY*anchoCelula) ,(posicionEnX*altoCelula) , (posicionEnY*anchoCelula)+anchoCelula , (posicionEnX*altoCelula)+altoCelula)
        else:
            celulasVivas+=1
            celula = espacioCelularCanvas.create_rectangle((posicionEnY*anchoCelula) ,(posicionEnX*altoCelula) , (posicionEnY*anchoCelula)+anchoCelula , (posicionEnX*altoCelula)+altoCelula)

#Las etiquetas declaradas aqui podran ser actualizadas en las funciones de arriba
etiqueta1=tkinter.Label(ventanaPrincipal,text="State")#etiqueta 1 es el estado

```

```

etiqueta1.config(font=("Arial", 24))
etiqueta1.place(x= 860.0,y= 50.0)
etiquetaNESTADO=tkinter.Label(ventanaPrincipal,text=str(numeroEstado))
etiquetaNESTADO.config(font=("Arial", 24))
etiquetaNESTADO.place(x= 900.0,y= 100.0)

etiqueta2=tkinter.Label(ventanaPrincipal,text="Living cells")#etiqueta 1 es el estado
etiqueta2.config(font=("Arial", 24))
etiqueta2.place(x= 860.0,y= 150.0)
etiquetaNCELVIVAS=tkinter.Label(ventanaPrincipal,text=str(celulasVivas))
etiquetaNCELVIVAS.config(font=("Arial", 24))
etiquetaNCELVIVAS.place(x= 900.0,y= 200.0)

etiqueta5=tkinter.Label(ventanaPrincipal,text="Size of the\n cell space")#etiqueta 1 es el
etiqueta5.config(font=("Arial", 24))
etiqueta5.place(x= 860.0,y= 250)

tamUniversoLife=tkinter.StringVar(ventanaPrincipal)#Creamos una variable que guarde el
tamUniversoLife.set('100x100')
varioTamsUniversosLife=['100x100','500x500','1000x1000','5000x5000','10000x10000']
menuTamUniversosLife=tkinter.OptionMenu(ventanaPrincipal,tamUniversoLife,*varioTamsUniversosLife)
tamUniversoLife.trace("w",callback5)

estadoDelJuego=np.copy(estadoDelJuegoNuevo)

#Agregamos caracteristica de click en canvas (panel negro) que es el espacio celular
espacioCelularCanvas.bind("<Button-1>",agregarQuitarCelula)
#Creamos botones. Los agregamos a ventana principal
botonReproducir=tkinter.Button(ventanaPrincipal,command=lambda: accionBtnReproducir(),text="Reproducir")
botonPausar=tkinter.Button(ventanaPrincipal,command=lambda: accionBtnPausar(),text="Pausar")
botonReiniciar=tkinter.Button(ventanaPrincipal,command=lambda: accionBtnReiniciar(),text="Reiniciar")
botonResultados=tkinter.Button(ventanaPrincipal,command=lambda: accionBtnResultados(),text="Resultados")
botonCargar=tkinter.Button(ventanaPrincipal,command=lambda: accionBtnCargar(),text="Cargar")
botonGuardar=tkinter.Button(ventanaPrincipal,command=lambda: accionBtnGuardar(),text="Guardar")

etiqueta4=tkinter.Label(ventanaPrincipal,text="Rule\nB0-8/S0-8")#etiqueta 1 es el estado
etiqueta4.config(font=("Arial", 18))
etiqueta4.place(x= 1150.0,y= 500)
regla=tkinter.StringVar(ventanaPrincipal)
regla.set("B3/S23")
campoRegla=tkinter.Entry(ventanaPrincipal, textvariable=regla, width=20).place(x=1165, y=525)

#Creamos espacio para los arboles, que puede ser de 6x6 como maximo
numColumnas2=2
numFilas2=2
anchoCelula2=(120/numColumnas2)
altoCelula2=(120/numFilas2)

```

```

#Creamos una ventana 3
espacioContenedorCelular2=tkinter.Frame(ventanaPrincipal, width=120, height=120, bg='black')
espacioContenedorCelular2.place(x=1040,y=350)
#Creamos canvas encima de espacioContenedorCelular
espacioCelularCanvas2=tkinter.Canvas(espacioContenedorCelular2,width=120,height=120,bg='white')
espacioCelularCanvas2.place(x=1040,y=350)
#####SCROLLIN

#Creamos scrollbar horizontal y vertical al frame
sbarV2 = tkinter.Scrollbar(espacioContenedorCelular2, orient=tkinter.VERTICAL, command=espacioCelularCanvas2.yview)
sbarH2 = tkinter.Scrollbar(espacioContenedorCelular2, orient=tkinter.HORIZONTAL, command=espacioCelularCanvas2.xview)

#los posiciono en el frame
sbarV2.pack(side=tkinter.RIGHT, fill=tkinter.Y)
sbarH2.pack(side=tkinter.BOTTOM, fill=tkinter.X)
espacioCelularCanvas2.config(yscrollcommand=sbarV2.set)
espacioCelularCanvas2.config(xscrollcommand=sbarH2.set)
espacioCelularCanvas2.pack(side=tkinter.LEFT, expand=True, fill=tkinter.BOTH)
espacioCelularCanvas2.config(scrollregion=(0, 0, numColumnas2*anchoCelula2,numFilas2*altoCelula2))
#####SCROLLfin

#ESTADO INICIAL de los arboles
estadoDelJuego2=np.zeros((numFilas2,numColumnas2))
estadoDelJuegoNuevo2=np.copy(estadoDelJuego2)
numeroEstado2=0
celulasVivas2=0
estadoDelJuegoNuevo2=np.copy(estadoDelJuego2)
for posicionEnX in range(0,numFilas2):
    for posicionEnY in range(0,numColumnas2):
        celula = espacioCelularCanvas2.create_rectangle((posicionEnY*anchoCelula) ,(posicionEnX*altoCelula))
        if estadoDelJuego2[posicionEnX,posicionEnY] == 0:#dibujo un rectangulo negro
            celula = espacioCelularCanvas2.create_rectangle((posicionEnY*anchoCelula2) ,(posicionEnX*altoCelula2), fill='black')
        else:
            celulasVivas2+=1
            celula = espacioCelularCanvas2.create_rectangle((posicionEnY*anchoCelula2) ,(posicionEnX*altoCelula2), fill='green')

#####
tamUniversoArboles=tkinter.StringVar(ventanaPrincipal)#Creamos una variable que guarde el tamano del universo
tamUniversoArboles.set('2x2')
varioTamsUniversos=['2x2','3x3','4x4','5x5']
menuTamUniversos=tkinter.OptionMenu(ventanaPrincipal,tamUniversoArboles,*varioTamsUniversos)
tamUniversoArboles.trace("w",callback4)

##Creamos boton play2
botonReproducir2=tkinter.Button(ventanaPrincipal,command=lambda: accionBtnReproducir2())
##Creamos boton seeArboles
botonVerArboles=tkinter.Button(ventanaPrincipal,command=lambda: accionBtnVerArboles())
#Mostramos nuestra interfaz para siempre con mainloop
ventanaPrincipal.mainloop()

```


Capítulo 3

Conclusiones

En este momento, se tiene un juego de la vida en nuestras manos. Los resultados que pueden obtenerse con unas simples reglas son fantásticas, sin embargo, nos dirijimos a la pregunta: **¿es el juego de la vida un sistema caótico o complejo?**. La respuesta a esta pregunta se basa en los atractores generados por el software. Estos atractores tienen nodos relativamente idénticos a su cercanía, además, es complicado analizar la evolución dinámica de cada uno de los nodos, siendo que estos pueden ser atractores o pueden ser nodos atractores que apuntan a otros atractores.

Hace ya bastante tiempo que se demostró que el juego de la vida es equivalente a una máquina universal de Turing, lo que le permitiría, teóricamente, realizar cualquier computación posible si se dispusiera de suficiente tiempo y suficiente memoria, por ejemplo, videojuegos de consolas de novena generación.

Entre otras aplicaciones también están las simulaciones de ciertos procesos químicos o incluso biológicos, tales como la formación de los patrones en la piel, el pelo o las conchas de ciertos animales. En las plantas sucede otro tanto: el comportamiento de los estomas (células de la epidermis de las hojas) es relativamente fácil de simular con unas pocas reglas. Con algo más de complejidad se puede simular también el funcionamiento de las neuronas, creando modelos que pese a su simplicidad acaban mostrando una extraordinaria complejidad cuando se les aplican ciertos patrones o valores iniciales. En el fondo, esa es la grandeza del comportamiento tanto del juego de la vida como de otros autómatas celulares: que de algo tan simple pueda surgir algo elaborado y complejo. Tan elaborado y tan complejo que todavía hoy en día se siguen estudiando.

Capítulo 4

Referencias

1. http://www.ddlab.com/downloads/papers/2011_b-of-a_spanish.pdf
2. <https://www.eumed.net/libros-gratis/2011e/1100/attractor.html>
3. http://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S2448-66552019000200179
4. <http://delta.cs.cinvestav.mx/~mcintosh/oldweb/tesis/genaro/node5.html>
5. <http://delta.cs.cinvestav.mx/~mcintosh/oldweb/s1997/todos/node16.html>