Autómatas Celulares

Ramsés Castro Molano[1]
[1]*Universidad Nacional Autónoma de México, Facultad de Ingeniería, Ciudad de México, México*
4 de noviembre del 2019

***Abstract*-- It will be known what a system is, the types of systems that exist to talk about a cellular automaton, the elements that it possesses among several other things, to finally develop the game of life.**

***Resumen*— Se conocerá lo que es un sistema, los tipos de sistemas que existen para hablar sobre un autómata celular, los elementos que posee entre diversas cosas más, para al final desarrollar el juego de la vida.**

I. Introducción

Un sistema es un conjunto de partes o elementos organizados y relacionados, que interactúan entre en sí, para llegar a un mismo objetivo. Los sistemas reciben (entrada) datos, energía o materia del ambiente y tienen como resultado que proveen (salida) información, energía o materia.

Los sistemas tienen límites o fronteras, que los diferencian del ambiente. Ese límite puede ser físico (el gabinete de una computadora) o conceptual. Si hay algún intercambio entre el sistema y el ambiente a través de ese límite, el sistema es abierto, de lo contrario el sistema sería cerrado. El ambiente es el medio externo que envuelve física o conceptualmente a un sistema. El ambiente también puede ser una amenaza para el sistema.

Los sistemas complejos se caracterizan fundamentalmente porque su comportamiento es imprevisible. Sin embargo, complejidad no es sinónimo de complicación: este último hace referencia a algo enmarañado, enredado, de difícil comprensión. En realidad, y por el momento, no existe una definición precisa y absolutamente aceptada de lo que es un sistema complejo, pero pueden darse algunas peculiaridades comunes.

Está compuesto por una gran cantidad de elementos relativamente idénticos. Por ejemplo, las células en un organismo, o las personas en una sociedad. La interacción entre sus elementos es local y origina un comportamiento emergente que no puede explicarse a partir de dichos elementos tomados aisladamente. Un desierto puede contener billones de granos de arena, pero sus interacciones son excesivamente simples comparadas con las que se verifican en las abejas de un enjambre.

Es muy difícil predecir su evolución dinámica futura; o sea, es prácticamente imposible vaticinar lo que ocurrirá más allá de un cierto horizonte temporal.

En la naturaleza se pueden encontrar una gran cantidad de ejemplos de sistemas complejos que se extienden desde la física hasta la neurología, desde la economía hasta la biología molecular, desde la sociología hasta las matemáticas.

La mayoría de los sistemas complejos son inestables, se mantienen delicadamente equilibrados. Cualquier variación mínima entre sus elementos componentes puede modificar, de forma imprevisible, las interrelaciones y, por lo tanto, el comportamiento de todo el sistema. Así, la evolución de esta clase de sistemas se caracteriza por la fluctuación, situación en la que el orden y el desorden se alternan constantemente. Sus estados evolutivos no transcurren a través de procesos continuos y graduales, sino que suceden por medio de reorganizaciones y saltos.

Los sistemas caóticos y los atractores extraños son conceptos que provienen de la Teoría del caos, impulsada en la década de los sesenta por el matemático y meteorólogo estadounidense Edward Lorenz. Con el tiempo, esta teoría ha terminado aplicándose en campos tan diversos como la economía o las ciencias sociales, en lo que se conoce como caología.

Las tres características principales que cumplen los sistemas caóticos son: la no-linealidad, la extrema sensibilidad que poseen ante cambios muy pequeños de sus condiciones iniciales y que no se puede prever el comportamiento del sistema hasta que el proceso sucede o se calcula. A pesar de este comportamiento impredecible, el sistema es determinista. Es decir, para unos parámetros dados, el sistema está completamente determinado para tiempos futuros, por muchas veces que lo volvamos a calcular o repetir. Eso sí, si cambiamos mínimamente alguno de los parámetros, podremos encontrarnos con una sorpresa: un resultado final muy diferente al original.

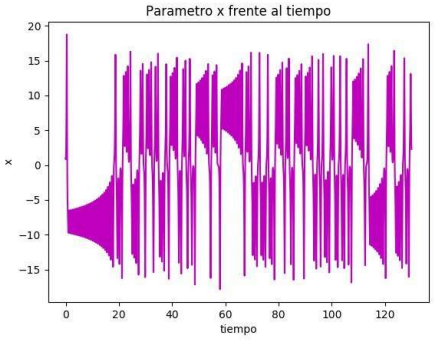


Figura 1

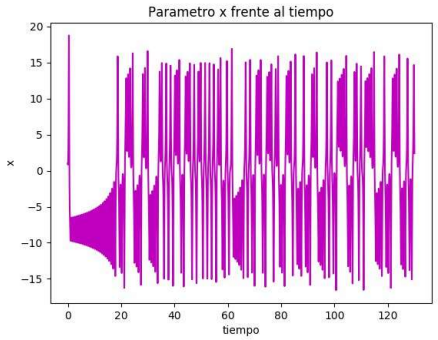


Figura 2

En las figuras presentadas son la representación de uno de los parámetros de las ecuaciones de Lorenz utilizadas para estudiar los fluidos y fenómenos meteorológicos para dos condiciones iniciales distintas. Es un ejemplo de sistema caótico en el que se observa cómo sus parámetros muestran comportamientos impredecibles a lo largo del tiempo que no se repiten. Además, el comportamiento del sistema es diferente dependiendo de los valores de las condiciones iniciales de los parámetros.

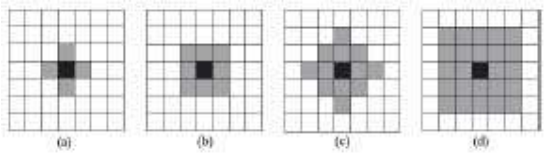
Un autómata celular es un modelo matemático para un sistema dinámico compuesto por un conjunto de celdas o células que adquieren distintos estados o valores. Estos estados son alterados de un instante a otro en unidades de tiempo discreto, es decir, que se puede cuantificar con valores enteros a intervalos regulares. De esta manera este conjunto de células logra una evolución según una determinada expresión matemática, que es sensible a los estados de las células vecinas, y que se conoce como regla de transición local.

Los autómatas celulares (AC) surgen en la década de 1940 con John Von Neumann, que intentaba modelar una máquina que fuera capaz de autorreplicarse, llegando así a un modelo matemático de dicha maquina con reglas complicadas sobre una red rectangular.

El aspecto que más caracteriza a los AC es su capacidad de lograr una serie de propiedades que surgen de la propia dinámica local a través del paso del tiempo y no desde un inicio, aplicándose a todo el sistema en general. Por lo tanto, no es fácil analizar las propiedades globales de un AC desde su comienzo, complejo por naturaleza, si no es por medio de una simulación, partiendo de un estado o configuración inicial de células y cambiando en cada instante los estados de todas ellas de forma síncrona.

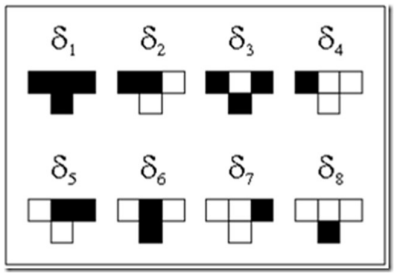
La definición de un AC requiere mencionar sus elementos básicos:

- **Un espacio regular**. Ya sea una línea, un plano de 2 dimensiones o un espacio n-dimensional. Cada división homogénea del espacio es llamada célula.
- **Conjunto de Estados**. Es finito y cada elemento o célula del espacio toma un valor de este conjunto de estados. También se denomina alfabeto. Puede ser expresado en valores o colores.
- **Configuración Inicial**. Es la asignación inicial de un estado a cada una de las células del espacio.
- **Vecindades**. Define el conjunto de células que se consideran adyacentes a una dada, así como la posición relativa respecto a ella. Cuando el espacio es uniforme, la vecindad de cada célula es isomorfa (es decir, que tiene el mismo aspecto).



Ejemplos de vecindades

- **Función de Transición Local**. Es la regla de evolución que determina el comportamiento del AC. Se calcula a partir del estado de la célula y su vecindad. Define cómo debe cambiar de estado cada célula dependiendo su estado anterior y de los estados anteriores de su vecindad. Suele darse como una expresión algebraica o un grupo de ecuaciones.



Ejemplos de Transición

Adicionalmente, es necesario especificar el tipo de limite o frontera del espacio, entre los que podemos destacar:

- **Frontera Abierta**. Se considera que todas las células fuera del espacio del autómata toman un valor fijo.
- **Frontera Reflectora**. Las células fuera del espacio del autómata toman los valores que están dentro, como si se tratara de un espejo.
- **Frontera Periódica o Circular**. las células que están en la frontera interaccionan con sus vecinos inmediatos y con las células que están en el extremo opuesto del espacio, como si lo dobláramos en forma de cilindro.
- **Sin Frontera**. La representación des autómata no tiene límites, es infinito.

Un autómata celular reversible, es un autómata celular en el que cada configuración tiene un predecesor único. Es una cuadrícula regular de celdas, cada una de las cuales contiene un estado dibujado a partir de un conjunto finito de estados. Esta cuadrícula tiene una regla para actualizar todas las celdas simultáneamente en función de los estados de sus vecinos, de modo que el estado anterior de cualquier celda antes de una

actualización se puede determinar únicamente a partir de los estados actualizados de todas las celdas.
Para poder hablar de autómatas celulares reversibles primero se tiene que hablar del concepto de reversibilidad y su relación con los autómatas. Definimos reversibilidad como la cualidad de un sistema de ir a través de una serie de cambios ya sea hacia "adelante" o hacia "atrás" sin presentar alteraciones en estos sistemas, es decir, la cualidad de regresar o avanzar hacia etapas que previamente ya se habían visitado sin que estas presenten cambios.
En concreto hablando de la reversibilidad de una forma matemática tenemos una función inyectiva, o uno a uno, en donde cada entrada solo tiene una salida y viceversa siendo de la forma $y = f(x)$.
Stephen Wolfram comenzó a trabajar en autómatas celulares a mediados de 1981 después de considerar cómo los patrones complejos parecían formarse en la naturaleza en violación de la segunda ley de la termodinámica. Sus investigaciones fueron inicialmente impulsadas por un interés en sistemas de modelado, como las redes neuronales. Tras ver la inesperada complejidad del comportamiento de estas reglas simples Wolfram llevó a sospechar que la complejidad en la naturaleza puede ser debida a mecanismos similares. En 2002 Wolfram publicó un texto de 1.280 páginas, A New Kind of Science, que sostiene ampliamente que los descubrimientos sobre autómatas celulares no son hechos aislados, sino que son robustos y tienen importancia para todas las disciplinas de la ciencia.

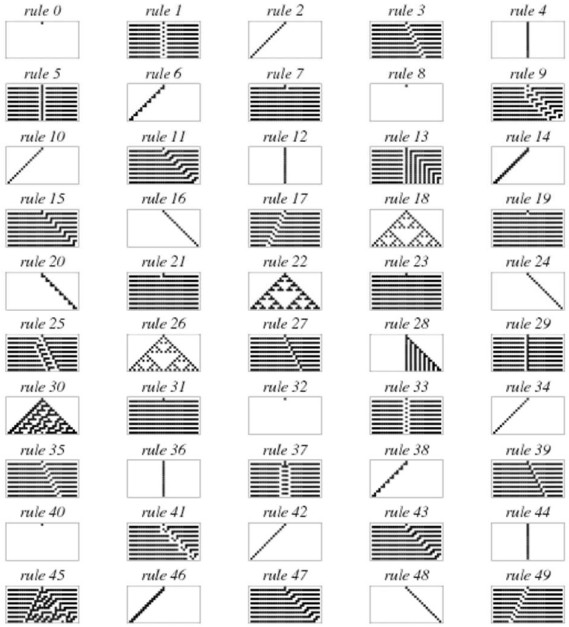
Wolfram define cuatro clases en las que los AC. Mientras que los estudios anteriores en autómatas celulares tienden a tratar de identificar el tipo de patrones de reglas específicas, la clasificación de Wolfram fue el primer intento de clasificación global. En orden de complejidad las clases que identifica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

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

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

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

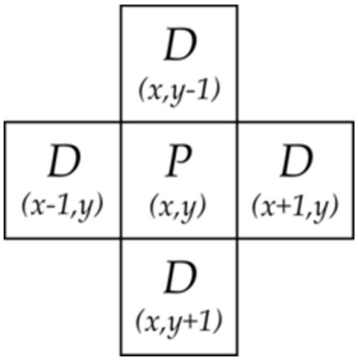
- **Clase IV**: Casi todos los patrones iniciales evolucionan en las estructuras que interactúan de manera compleja e interesante, con la formación de las estructuras locales que son capaces de sobrevivir por largos períodos de tiempo. Podría ser el caso de que apareciesen estructuras estables u oscilantes,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



Reglas de Wolfram

Tenemos que existen dos tipos de geometría de vecindad, la de Von Neumann y la de Moore.

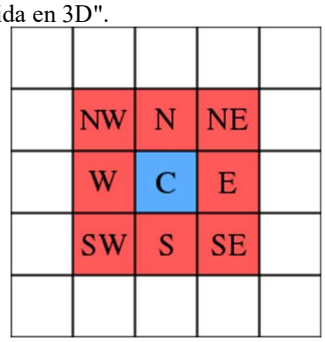
El concepto de vecindad de von Neumann se define como el conjunto de las cuatro celdas que rodean ortogonalmente a una celda central en un enrejado cuadrado bidimensional.
El concepto puede ser extendido a dimensiones más altas, por ejemplo, formando una vecindad octaédrica de 6 celdas para un autómata celular cúbico en tres dimensiones.
La vecindad de von Neumann de un punto es el conjunto de puntos situados a una distancia de Manhattan de valor 1 respecto al punto dado.



Vecindad von Neumann

La Vecindad de Moore se define como el conjunto de las ocho celdas que rodean una celda central en un enrejado cuadrado bidimensional.

El concepto puede ser extendido a dimensiones más altas, por ejemplo, formando una vecindad cúbica de 26-celdas para un autómata celular en tres dimensiones, como los utilizados en los "juegos de la vida en 3D".



Vecindad de Moore

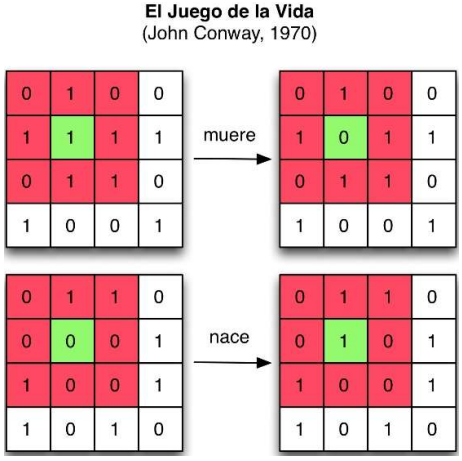
El juego de la vida es un autómata celular diseñado por el matemático británico John Horton Conway en 1970.

Se trata de un juego de cero jugadores, lo que quiere decir que su evolución está determinada por el estado inicial y no necesita ninguna entrada de datos posterior. El "tablero de juego" es una malla plana formada por cuadrados (las "células") que se extiende por el infinito en todas las direcciones. Por tanto, cada célula tiene 8 células "vecinas", que son las que están próximas a ella, incluidas las diagonales.

Las células tienen dos estados: están "vivas" o "muertas" (o "encendidas" y "apagadas"). El estado de las células evoluciona a lo largo de unidades de tiempo discretas (se podría decir que por turnos). El estado de todas las células se tiene en cuenta para calcular el estado de estas al turno siguiente.

Todas las células se actualizan simultáneamente en cada turno, siguiendo estas reglas:

- 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").



Reglas del Juego de la Vida

Juego de la vida

Existen numerosos tipos de patrones de células que pueden tener lugar en el juego de la vida:

- **Osciladores**: Son patrones que son predecesores de sí mismos. En otras palabras, son patrones que tras un número finito de generaciones vuelven a su estado inicial.
- **Vidas Estáticas**: Son patrones que no cambian de una generación a la siguiente. Las vidas estáticas se pueden considerar como osciladores de período 1.
- **Naves Espaciales**: Son patrones que reaparecen en otra posición tras completar su período. Esto es, son patrones que tras un número finito de generaciones vuelven a su estado original, pero en una ubicación diferente.
- **Matusalenes**: Son patrones que pueden evolucionar a lo largo de muchos turnos, o generaciones, antes de estabilizarse.

II. Objetivo

Es el de aprender sobre los autómatas celulares mediante una de sus aplicaciones que es el juego de la vida.

Se realizará en el lenguaje de programación C de manera secuencial para que en un futuro pueda ser paralelizado utilizando MPI.

III. Modelo

Para generar el modelo conceptual que utilizaremos nos iremos basando en los siguientes puntos:

1. Asignar la geometría del espacio celular.
2. Asignar la geometría de la vecindad.
3. Definir el conjunto de estados de las celdas.
4. Asignar las reglas de transición.
5. Asignar las condiciones de frontera.
6. Asignar la condición inicial del autómata celular.
7. Cambiar repetidamente las reglas del AC hasta llegar a una condición de paro.

Por lo que partiendo de los puntos mencionados tenemos que la geometría del espacio que utilizaremos será bidimensional, aplicando la vecindad de Moore.

El estado de las células puede ser dos, estar vivas (1) y estar muertas (0), las reglas de transición a utilizar son las que el juego estipula para su versión estándar, una célula viva con más de tres vecinos vivos o con menos de dos vecinos vivos morirá al siguiente turno, una célula muerta con exactamente 3 vecinos

nacerá al siguiente turno y una célula viva con 2 o 3 vecinos vivos se mantendrá con vida otro turno.

Las condiciones de frontera a utilizar son la de una frontera abierta con valor de 0 las que se encuentren afuera del tablero, la posición inicial del tablero con las células que estarán vivas será establecidas por valores al azar en las posiciones.

En este caso no existe una condición de paro, por lo que el juego seguirá, aunque ya no existan células o el sistema quede de manera estable.

Para el modelo matemático, necesitamos una matriz regular de células que cubra una porción de un espacio bidimensional.

Un conjunto de variables booleanas que nos dirán el estado de la célula en dicho momento de tiempo:

$$M(e,t) = \{M_0(i,j,t), M_1(i,j,t) \dots M_{n*m}(i,j,t)\}$$

Una regla $R = \{R_1, R_2, R_3 \dots R_p\}$ que nos especifica la evolución temporal de los estados de la siguiente forma:

$$\begin{aligned} M_p(i,j,t+1) = R_j(&M(i-1,j-1,t), \quad M(i-1,j,t), \\ &M(i-1,j+1,t), \quad M(i,j-1,t), \\ &M(i,j+1,t), \quad M(i+1,j-1,t), \\ &M(i+1,j,t), \quad M(i+1,j+1,t)) \end{aligned}$$

Donde el conjunto de reglas está dado por las reglas del mismo juego, y la evolución está dado por los vecinos de la célula central.

Para el modelo computacional tenemos al programa que se encargará de generar las respuestas al problema planteado, el código y capturas de la ejecución se muestran más adelante.

IV. Análisis del problema

Para resolver el problema planteado debemos de hacer uso en el modelo planteado en el punto anterior.

Para generar la geometría podemos manejarla con una matriz que manejaremos de un tamaño estático que será de 20 * 20, lo que nos dará un tablero con 400 células.

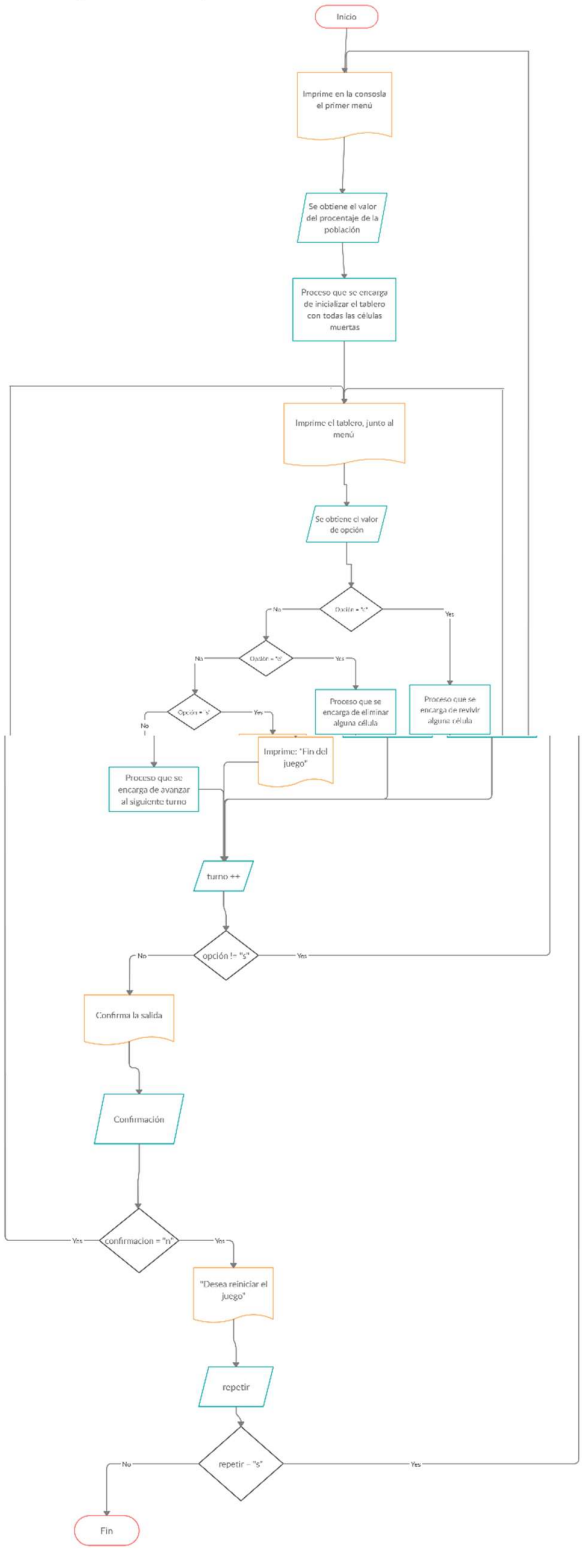
Haremos una copia de la matriz con las células para poder modificarla sin riesgo a perder la información de alguna célula.

Para la comprobación de los vecinos de cada célula se hace un barrido donde se toma a una célula como centro y se comprueba el estado de sus colindantes, con los que se hará una cuenta de los vecinos vivos y comprobar la cuenta para decidir qué opción tomar con respecto al estado de la célula central.

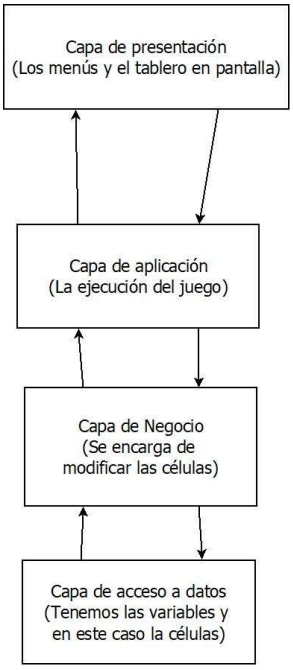
Se utilizará un menú con el cual poder decidir el porcentaje de la población que estará viva en el tablero al inicio del juego, y junto a ello otro menú con el cual podremos revivir alguna célula para dibujar algún patrón, destruir bloques de vida estática por sobrepoblación, otra opción sería para eliminar alguna célula y observar el comportamiento del juego sin ella, y una última será para salir del juego. También se muestra el conteo de turnos y a su vez el número de células existentes.

V. Implementación

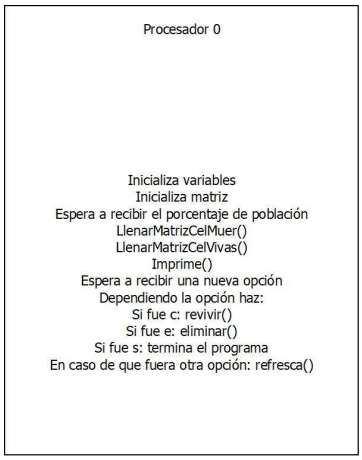
- Diagrama de flujo



- Diagrama de patrón arquitectónico



- Diagrama UML



VI. CÓDIGO FUENTE IMPORTANTE

A continuación, se muestran las funciones importantes utilizadas para el desarrollo del programa:

```
//Función dedicada a actualizar el tablero y las celulas
void Refrescar(int Matriz[Filas][Columnas]){
    int copiaMatriz[Filas][Columnas], NUMcelulasvivas = 0;
    int i, j; //Variables que se utilizan en los ciclos for
        //Copiamos la matriz en una copia auxiliar
        Duplicar(Matriz, copiaMatriz);
        for(i = 0; i < Filas; i++){ //Ciclo para recorrer las filas
            for(j = 0; j < Columnas; j++){ //Ciclo para recorrer las columnas
                //Control de las celulas vecinas vivas
                if(i > 0 && j > 0 && Matriz[i-1][j-1] == 1){//Esquina Superior Izquierda
                    NUMcelulasvivas++;
                }
                if(i > 0 && Matriz[i-1][j] == 1){//Arriba
                    NUMcelulasvivas++;
                }
                if(i > 0 && j < Columnas && Matriz[i-1][j+1] == 1){//Esquina Superior Derecha
                    NUMcelulasvivas++;
                }
                if(j > 0 && Matriz[i][j-1] == 1){ //Derecha
                    NUMcelulasvivas++;
                }
                if(j < Columnas && Matriz[i][j+1] == 1){ //Izquierda
                    NUMcelulasvivas++;
                }
                if(i < Filas && j > 0 && Matriz[i+1][j-1] == 1){ //Esquina Inferior Izquierda
                    NUMcelulasvivas++;
                }
                if(i < Filas && Matriz[i+1][j] == 1){ //Abajo
                    NUMcelulasvivas++;
                }
                if(i < Filas && j < Columnas && Matriz[i+1][j+1] == 1){ //Esquina Inferior Derecha
                    NUMcelulasvivas++;
                }
                if(Matriz[i][j] == 1){//Actuamos sobre las celulas en la copia de la matriz
                    //La celulas vivas con 2 o 3 celulas vivas pegadas, se mantiene vivas.
                    if(NUMcelulasvivas == 2 || NUMcelulasvivas == 3){
                        copiaMatriz[i][j] = 1;
                    }
                    else{//Si no se cumple la condicion, mueren.
                        copiaMatriz[i][j]=0;
                    }
                }
                else{
                    if(NUMcelulasvivas == 3){//Las celulas muertas con 3 celulas vivas pegadas, resucitan.
                        copiaMatriz[i][j] = 1;
                    }
                }
                NUMcelulasvivas = 0;//Ponemos a 0 el contador
            }
        }
        Duplicar(copiaMatriz, Matriz); //Devolvemos los nuevos datos a la matriz original
}

//Función dedicada a revivir alguna célula para modificar el comportamiento del juego
void Resucitar(int Matriz[Filas][Columnas]){
    int i,j,a,b; //Variables que se utilizan como opciones
    do{ //Ciclo de opción de celulas
        printf("Introduce fila <1-%d>: ", Filas);
        scanf("%d",&i);
    }while(i < 1 || i > Filas);
    do{
        printf("\nIntroduce columna <1-%d>: ", Columnas);
        scanf("%d",&j);
    }while(j < 1 || j > Columnas);
    a = Matriz[i-1][j-1]; //Copiamos la el valor de la celula, para poder revertir el cambio
    Matriz[i-1][j-1] = 1; //Resucitamos la celula seleccionada
    system("cls");
    Imprimir(Matriz);
    printf("\nCompruebe la seleccion, desea mantenerla? [s/n]");
    b = getch();
    //Limpiamos el buffer
    fflush(stdin);
    //Si no se mantiene, se devuelve el valor de la celula copiada anteriormente
    if(b == 'n') Matriz[i-1][j-1]=a;
}
 //Función principal dedicada a la ejecución del juego
 void Juego(){
     //Definimos las diversas variables a utilizar
     int poblacion, i, j, fil, col, repetir, confirmar, turno = 0;
     int Matriz [Filas][Columnas], tipo, conteo = 0;
     clock_t t_ini, t_fin;
     double secs;
     char letra;
     do{//Menu inicial
         system("cls");
         printf("Bienvenido al juego de la Vida\n");
         printf("Algunas distribuciones que podrían ser interesantes son:\n");
         printf("\t1)Dispersa 10% \n\t2)Normal 25% \n\t3)Densa 50% \n\t4)Masiva 75% \n");
         printf("Ingresa un porcentaje de distribución (1 - 99): ");
         scanf("%d",&tipo);
         system("cls");
         //Ponemos todas las celulas muertas inicialmente
         t_ini = clock();
         for(i = 0; i < Filas; i++){
             for(j = 0; j < Columnas; j++){
                 Matriz [i][j] = 0;
             }
         }
         poblacion = (((Filas*Columnas)/100)*tipo);
         //Modo aleatorio
         srand(time(0));
         for(i = 0; i < poblacion; i++){
             fil = rand() % Filas;
             col = rand() % Columnas;
           //Si la celula esta muerta, la resucita
           if(Matriz [fil][col] == 0){
               Matriz[fil][col] = 1;
           }
           /*Si ya estaba viva se mantiene, y no lo contamos como
           una posicion añadida, asi aseguramos que se impriman el
           numero de celulas vivas seleccionado. Por ello restamos uno
           al contador*/
           else
               i--;
           }
       t_fin = clock();
       do{
           do{
               system("cls");
               conteo = Vida(conteo, Matriz);
               Imprimir(Matriz);
               t_fin = clock();
               secs = (double)(t_fin - t_ini) / CLOCKS_PER_SEC;
               printf("\nTurno: %d\nCelulas vivas: %d\n", turno, conteo);
               printf("Tiempo de respuesta: %.16g milisegundos\n", secs * 1000.0);
               printf("Pulse una tecla:\n[s] para salir\n[c] para revivir celula\n[e] para eliminar\n");
               //Opciones en el juego
               letra = getch();
               //Limpiamos el buffer
               fflush(stdin);
               if(letra == 's'){
                   system("cls");
                   Imprimir(Matriz);
                   printf("\nFin del juego");
               }
```

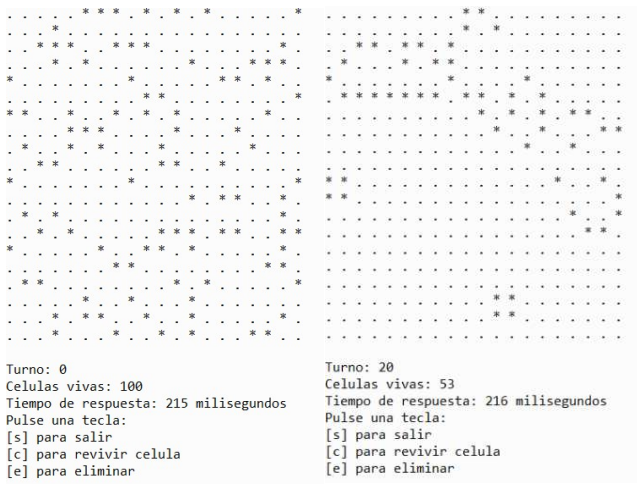
```
            else if(letra == 'c'){
                system("cls");
                Imprimir(Matriz);
                Resucitar(Matriz);
            }
            else if(letra == 'e'){
                system("cls");
                Imprimir(Matriz);
                Eliminar(Matriz);
            }
            else{
                t_ini = clock();
                Refrescar(Matriz);
                turno ++;
            }
        }while(letra != 's');
        //Confirmacion de salida, para prevenir equivocaciones
        system("cls");
        printf("\n\n\n\n\n%20cSeguro que quieres salir? [s/n]", ' ');
        confirmar = getch();
        fflush(stdin);
    }while(confirmar == 'n');
    system("cls");
    //Opcion para volver a empezar, asi no hace falta salir del programa y volver a ejecutar
    printf("\n\n\n\n\n%20cVolver a empezar? [s/n]", ' ');
    turno = 0;
    repetir = getch();
    //Limpiamos el buffer
    fflush(stdin);
}while(repetir == 's');
}
```

VII. Métricas

Las métricas que utilizamos son la de contar el número de vecinos vivos que posee cada célula, el número de células existentes en el tablero, y el número de turnos para tener un mejor manejo del juego. Con lo que respecta al tiempo calculamos el tiempo que tarda en hacer el refresco únicamente.

```
Bienvenido al juego de la Vida
Algunas distribuciones que podrían ser interesantes son:
        1)Dispersa 10
        2)Normal 25
        3)Densa 50
        4)Masiva 75
Ingresa un porcentaje de distribución (1 - 99): ■
```

Juego de la vida 1



Juego de la vida 2

Juego de la vida 3

Como se puede ver en las 3 imágenes anteriores, se ven las métricas utilizadas que se mencionaron anteriormente, de ahí podemos ver que el tiempo de respuesta es en promedio lo mismo.

VIII. Conclusión

En conclusión, tenemos que la aplicación de los autómatas celulares (AC) es diversa y una de ellas es el juego de la vida que implementamos. El tema de AC es complejo pero muy útil ya que con eso se está logrando entender diversas cosas de nuestro entorno como lo son los patrones que poseen los animales o plantas por poner un ejemplo.

Enfocándonos más en lo que es el juego de la vida, de manera filosófica podemos observar como evoluciona una sociedad, de un pequeño conjunto de personas (células) se van expandiendo por el mundo (tablero) reproduciéndose, haciendo más numerosa su especie, hasta llegar a un momento cumbre donde al existir demasiadas personas en un pequeño mundo donde los recursos no alcanzan a cubrir las necesidades de todos estas se comienzan a morir o en otras palabras se comienza una auto destrucción de la especie, hasta el punto de llegar a dejar un tablero en blanco.

De manera computacional nos interesa esto ya que es una prueba computacional ya que es equivalente a una máquina universal de Turing, por lo que todo lo que se pueda computar algorítmicamente se puede computar en dicho juego.

Para la continuación de esto se hará de manera paralela utilizando MPI y lo aprendido en la clase de sistemas distribuidos para desarrollar una mejor implementación que no consuma mucho tiempo al momento de tener un tablero demasiado grande.

IX. Referencias

https://donovan19.wordpress.com/2010/10/25/sistemas-tipos-y-clases-tgs/
https://www.tendencias.kpmg.es/2018/06/sistemas-caoticos-y-compliance/
https://nusgrem.es/sistemas-caoticos-y-teoria-del-caos/
https://www.lawebdelprogramador.com/foros/C-Visual-C/1552530-Sugerencias-para-un-programa-El-juego-de-la-vida.html
https://es.wikipedia.org/wiki/Juego_de_la_vida
http://www.cs.us.es/~fsancho/?e=66

Para revisar de mejor manera el código se anexa el repositorio donde se encuentra:
https://github.com/Ramy34/Sistemas_Distribuidos