Autómatas celulares
Física Computacional

Escrito por:
Alván Ventura, Edsel Yael
ealvan@unsa.edu.pe

Docente:
Apaza Veliz, Danny Giancarlo
dapazav@unsa.edu.pe

27/07/2022

1 Introducción

Los automatas celulares nacieron con Von Neuman[1] en 1903, al intentar modelar una máquina capaz de autoreplicarse. En este sentido, Von Neuman también buscó reglas para lograr este fin.

Posteriormente Stanishlaw Ulam[1] consideró un arreglo de celdas en las cuales se podía tener uno de los finitos estados permitidos y conforme pasaba el tiempo las celdas tenian la posibilidad de cambiar de acuerdo a funciones de transición. Luego en 1966 se publicó "Theory of Self-Reproducing Automata" por el estudiante de doctorado W. Burks.

En este sentido, se creó el término Autómata Celular, el cual se puede definir como[2]:

> *Un modelo matemático para un sistema dinámico compuesto por un conjunto de celdas o células que adquieren distintos estados o valores*

Se considera por tanto a un Automáta Celular(AC), como un sistema complejo discretizado con respecto a sus estados finitos y a cada unidad de tiempo transcurrido. Además tiene la dificultad de no saber con exactitud cual es la tendencia y propiedades de un AC dado un estado inicial. Pues a cada unidad de tiempo todas las celdas pueden cambiar de estado.

2 Automáta Celular

El automáta celular tiene un comportamiento basado en las configuraciones iniciales y las reglas que dominan el cambio de sus estados finitos, resultando importante saber qué elementos intervienen en un Autómata Celular, es por esto, que se hablará de los componentes principales de los AC.

2.1 Elementos de un AC

Un Automata Celular esta denotado por[1]:

$$AC(L, S, N, f) \tag{1}$$

Donde:

Elementos	Descripción
L	Es el espacio con una dimension "n", donde se desarrolla el AC. Considerando que cada elemento es una célula.
S	Es el conjunto finito de estados permitidos en AC, y cada célula debe adoptar uno de estos valores.
N	Es el conjunto de celulas que se consideran vecinas de una célula. Cuando el espacio es uniforme, la vecindad de cada célula tiene el mismo aspecto(isomorfismo).
f	Es la función de transición entre un estado a otro. Define principalmente como un estado siguiente es influenciado por su vecindad y su estado anterior. $f : S^N -> S$

2.2 Tipos de límites de espacio.

Existen 3 tipos de espacio segun Fernández[2]:

1. **Fronteras periódicas:** Cuando una célula llega al límite del espacio, la célula que está al opuesto extremo del espacio es considerada su vecina, tanto para verticalmente como horizontalmente. Por lo tanto, la figura de un plano con estas condiciones se llama *toroide*.

2. **Fronteras absorventes:** También llamadas fronteras abiertas[2], son las que consideran que el espacio es finito, y por lo tanto, las células en el límite del espacio no tienen vecindades en esa dirección.

3. **Fronteras reflectantes:** En este tipo de frontera, las células en el limite del espacio toman valores dentro del espacio, como si se tratase de un espejo, pues replican los valores que están dentro del espacio.

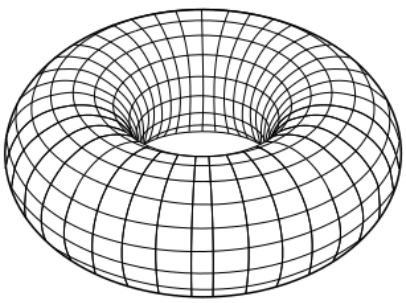


Figure 1: Reprensentación de una frontera periódica.

3 El juego de la vida.

El juego de la vida esta implementado en un Automata Celular, que dadas sus condiciones iniciales podemos observar como evoluciona el sistema en conjunto.

Al igual que un AC, podemos decir que tiene los siguientes componentes:

Elementos	Descripción
L	El espacio esta determinado por NxN, donde N es el número de celulas en su dimension, que puede ser de una dimension d determinada.
S	El conjunto finitos de estados son: • **ON:** La célula esta viva. • **OFF:** La célula esta muerta.
N	Se considera que cada célula tendra en total 8 vecinos adyacentes tanto vertical como horizontalmente.
f	Y que la función de transición tendrá las siguientes 4 reglas: 1. **Primera regla:** Cualquier celda viva con menos de dos vecinas vivas muere. 2. **Segunda regla:** Cualquier celda viva con dos o tres vecinos vivos vive. 3. **Tercera regla:** Cualquier celda viva con más de tres vecinos vivos muere. 4. **Cuarta regla:** Cualquier celda muerta con exactamente tres vecinos vivos se convierte en una celda viva.

3.1 Implementación en Python

El proceso principal del juego de la vida es[3]:

1. Inicializar las celdas en la cuadrícula.
2. A cada paso del tiempo, para cada celda (i, j):
   (a) Actualice el valor de la celda (i, j) en función de sus vecinos, teniendo en cuenta las condiciones de contorno.
   (b) Actualice los valores de la cuadrícula en cada paso del tiempo.

El siguiente código es sacado por un repositorio de Github[1] creado por Mahesh Venkitachalam.

The Game of Life

```
# Python code to implement Conway's Game Of Life
import argparse
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

# setting up the values for the grid
ON = 255
OFF = 0
vals = [ON, OFF]

def randomGrid(N):

        """returns a grid of NxN random values"""
        return np.random.choice(vals, N*N, p=[0.2, 0.8]).reshape(N, N)

def addGlider(i, j, grid):

        """adds a glider with top left cell at (i, j)"""
        glider = np.array([[0, 0, 255],
                                [255, 0, 255],
                                [0, 255, 255]])
        grid[i:i+3, j:j+3] = glider

def addGosperGliderGun(i, j, grid):

        """adds a Gosper Glider Gun with top left
        cell at (i, j)"""
        gun = np.zeros(11*38).reshape(11, 38)

        gun[5][1] = gun[5][2] = 255
        gun[6][1] = gun[6][2] = 255

        gun[3][13] = gun[3][14] = 255
        gun[4][12] = gun[4][16] = 255
        gun[5][11] = gun[5][17] = 255
        gun[6][11] = gun[6][15] = gun[6][17] = gun[6][18] = 255
        gun[7][11] = gun[7][17] = 255
        gun[8][12] = gun[8][16] = 255
        gun[9][13] = gun[9][14] = 255

        gun[1][25] = 255
        gun[2][23] = gun[2][25] = 255
        gun[3][21] = gun[3][22] = 255
        gun[4][21] = gun[4][22] = 255
        gun[5][21] = gun[5][22] = 255
        gun[6][23] = gun[6][25] = 255
        gun[7][25] = 255

        gun[3][35] = gun[3][36] = 255
        gun[4][35] = gun[4][36] = 255

```

[1] https://github.com/electronut/pp/blob/master/conway/conway.py

```
53          grid[i:i+11, j:j+38] = gun
54
55  def update(frameNum, img, grid, N):
56
57          # copy grid since we require 8 neighbors
58          # for calculation and we go line by line
59          newGrid = grid.copy()
60          for i in range(N):
61                  for j in range(N):
62
63                          # calculando los vecinos
64              #con fronteras periodica o toroidales.
65                          total = int((grid[i, (j-1)%N] + grid[i, (j+1)%N] +
66              grid[(i-1)%N, j] + grid[(i+1)%N, j] +
67              grid[(i-1)%N, (j-1)%N] + grid[(i-1)%N, (j+1)%N] +
68              grid[(i+1)%N, (j-1)%N] + grid[(i+1)%N, (j+1)%N])/255)
69
70                          # aplicando las 4 reglas del juego de la vida:
71                          if grid[i, j] == ON:
72              # < 2 vecinos muere o si es > 3 vecinos muere
73                                  if (total < 2) or (total > 3):
74                                          newGrid[i, j] = OFF
75                          else:
76              #si tiene 3 vecinos revive
77                                  if total == 3:
78                                          newGrid[i, j] = ON
79
80          # update data
81          img.set_data(newGrid)
82          grid[:] = newGrid[:]
83          return img,
84
85  # main() function
86  def main():
87
88          # Command line args are in sys.argv[1], sys.argv[2] ..
89          # sys.argv[0] is the script name itself and can be ignored
90          # parse arguments
91          parser = argparse.ArgumentParser(
92          description="Runs Conway's Game of Life simulation.")
93
94          # add arguments
95          parser.add_argument('--grid-size', dest='N', required=False)
96          parser.add_argument('--mov-file', dest='movfile', required=False)
97          parser.add_argument('--interval', dest='interval', required=False)
98          parser.add_argument('--glider', action='store_true', required=False)
99          parser.add_argument('--gosper', action='store_true', required=False)
100         args = parser.parse_args()
101
102         # set grid size
103         N = 100
104         if args.N and int(args.N) > 8:
105                 N = int(args.N)
106
107         # set animation update interval
108         updateInterval = 50
109         if args.interval:
```

```
110                 updateInterval = int(args.interval)
111
112         # declare grid
113         grid = np.array([])
114
115         # viendo cual condicion inicial se elige
116         if args.glider:
117         #primera opcion
118                 grid = np.zeros(N*N).reshape(N, N)
119                 addGlider(1, 1, grid)
120         elif args.gosper:
121         #segunda opcion
122                 grid = np.zeros(N*N).reshape(N, N)
123                 addGosperGliderGun(10, 10, grid)
124         else:
125         #tercera opcion
126                         #mas vivos que muertos
127                 grid = randomGrid(N)
128
129         # Configuracion para el movimiento:
130         fig, ax = plt.subplots()
131         img = ax.imshow(grid, interpolation='nearest')
132         ani = animation.FuncAnimation(fig, update, fargs=(img, grid, N, ),
133                     frames = 10,
134                     interval=updateInterval,
135                     save_count=50)
136
137         # # of cuadros por segundo(fps-frames per second)
138         # set output file
139         if args.movfile:
140                 ani.save(args.movfile, fps=30, extra_args=['-vcodec', 'libx264'])
141
142         plt.show()
143
144 # call main
145 if __name__ == '__main__':
146         main()
```

3.1.1 Análisis

El juego de la vida del script presentado en la anterior sección esta hecho con las siguientes herramientas:

- La librería *numpy*, para los arreglos y cuadriculas del espacio de *NxN*
- La librería *matplotlib*, para el dibujo y la representación de la cuadricula.
- La librería *argparse* para aceptar parametros via comandos.

Para empezar en el script anterior se presentan 3 tipos de condiciones iniciales las cuales son:

1. La *Glider*[4], es un patrón que es la primera nave espacial más pequeña, más común y descubierta por primera vez en el Juego de la Vida.
2. La *Gosper Glider Gun*[5], que configura el primer patrón finito conocido con crecimiento ilimitado, encontrado por Bill Gosper en noviembre de 1970.
3. Y finalmente, el aleatorio, en el que se configura que una celda esta viva o muerta aleatoriamente.

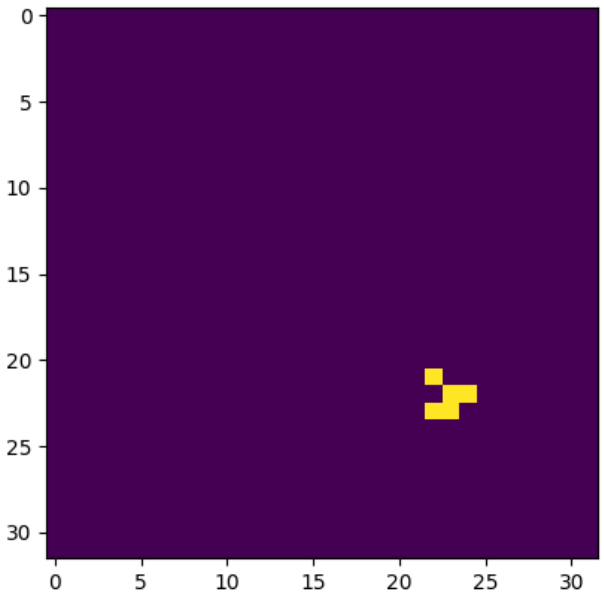
En la siguiente pieza de código se especifica que tipo de configuración inicial se decide, de acuerdo al parametro *–glider*, *–gosper* y si no se proporciona ninguna de esta se elige la opcion aleatoria.

```
# viendo cual condicion inicial se elige
if args.glider:
    #primera opcion->glider
    grid = np.zeros(N*N).reshape(N, N)
    addGlider(1, 1, grid)
elif args.gosper:
    #segunda opcion -> gosper Gun
    grid = np.zeros(N*N).reshape(N, N)
    addGosperGliderGun(10, 10, grid)
else:
    #tercera opcion-> aleatorioamente
        #mas vivos que muertos
    grid = randomGrid(N)
```

Para la 1^{ra} opción resulta con el comando:

```
python vida_game.py --grid-size 32 --interval 500 --glide
```

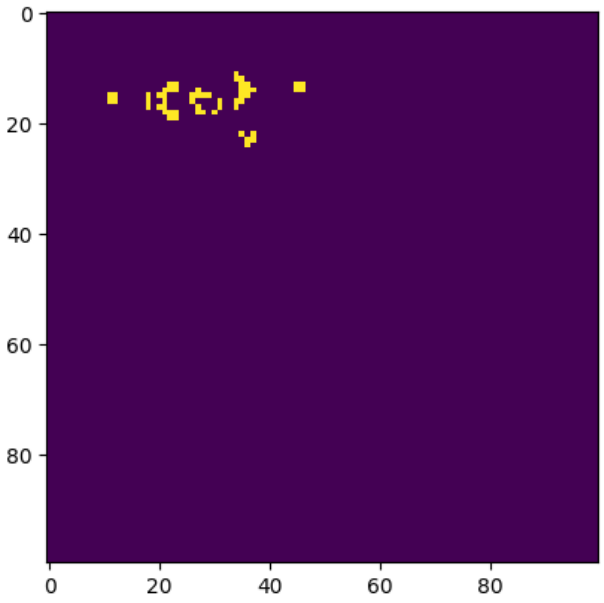
El resultado es:



Para la 2^{da} opción resulta con el comando:

```
python vida_game.py --interval 500 --gosper
```

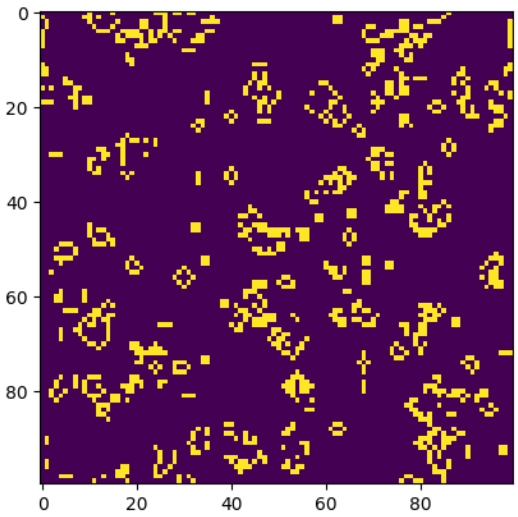
El resultado es:



Para la 3^{ra} opción resulta con el comando:

```
python vida_game.py --interval 500
```

El resultado es:



En la función *update()*, se lleva a cabo el cambio de estado de cada célula siguiendo las 4 reglas de Conway explicadas anteriormente.

A continuación se muestra la implementación de esas 4 reglas y el cambio de estados.

```
for i in range(N):
    for j in range(N):
        # calculando los vecinos
        #con fronteras periodica o toroidales.

        #el %N es para tener el comportamiento de frontera periodica
        #si es que pasa el limite las celulas en las fronteras.
        total = int((grid[i, (j-1)%N] + grid[i, (j+1)%N] +
            grid[(i-1)%N, j] + grid[(i+1)%N, j] +
            grid[(i-1)%N, (j-1)%N] + grid[(i-1)%N, (j+1)%N] +
            grid[(i+1)%N, (j-1)%N] + grid[(i+1)%N, (j+1)%N])/255)

        # aplicando las 4 reglas del juego de la vida:
        if grid[i, j] == ON:
            #< 2 vecinos muere o si es > 3 vecinos muere
            if (total < 2) or (total > 3):
                newGrid[i, j] = OFF
        else:
            #si tiene 3 vecinos revive
            if total == 3:
                newGrid[i, j] = ON
```

Y finalmente la ultima parte esta relacionada la actualización de la cuadricula del espacio de dos dimensiones hecha a través de *matplotlib*.

En la siguiente pieza de código se especifica la función que actualizara la cuadrícula(*update()*).

Se especifica, la unidad de tiempo en la que será actualizado el tablero(*updateInterval*), que puede ser especificado por línea de comandos.

```
# Configuracion para el movimiento:
fig, ax = plt.subplots()
img = ax.imshow(grid, interpolation='nearest')
ani = animation.FuncAnimation(fig, update, fargs=(img, grid, N, ),
                frames = 10,
                interval=updateInterval,
                save_count=50)

# # of cuadros por segundo(fps-frames per second)
# set output file
if args.movfile:
    ani.save(args.movfile, fps=30, extra_args=['-vcodec', 'libx264'])
plt.show()
```

References

[1] Autómatas Celulares y su Aplicación en Computación - Fernández Fraga Santiago Miguel y Rangel Mondragón Jaime. (n.d.). Retrieved July 26, 2022

[2] Autómatas Celulares. Fernando Sancho Caparrini. (n.d.). Retrieved July 26, 2022, from `http://www.cs.us.es/~fsancho/?e=66`.

[3] Conway's Game of Life.(Python Implementation) - GeeksforGeeks. (n.d.). Retrieved July 26, 2022, from `https://www.geeksforgeeks.org/conways-game-life-python-implementation/`

[4] Glider - LifeWiki. (n.d.). Retrieved July 26, 2022, from `https://conwaylife.com/wiki/Glider`

[5] Gosper glider gun - LifeWiki. (n.d.). Retrieved July 26, 2022, from `https://conwaylife.com/wiki/Gosper_glider_gun`