

# Práctica 2 Autómata Celular

Denisse Leyva

Febrero 24, 2021

## 1. Introducción

En esta segunda práctica trabajaremos con autómatas celulares en dos dimensiones, particularmente el famoso juego de la vida. El estado del autómata se representa con una matriz booleana (es decir, contiene ceros y unos). Cada celda es o viva (uno) o muerta (cero). En cada paso, la supervivencia de cada celda (verde) se determina a partir de los valores de sus ocho vecinos (amarillos):

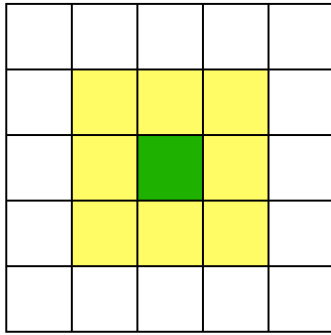



Figura 1: Matriz Booleana

En los extremos de la matriz, las celdas simplemente tienen menos vecinos. Otra alternativa sería considerar el espacio como un torus -pareciendo una dona- donde el extremo de abajo se reúne con el extremo de arriba igual como los lados izquierdo y derecho uno con otro.

La regla de supervivencia es sencilla: una celda está viva si exactamente 3 vecinos suyos están vivos[1].

## 2. Objetivo

Se debe diseñar y ejecutar un experimento para determinar el efecto de la regla de supervivencia (por lo menos 5 reglas) en la vida de la colonia en una malla de 12 por 12 celdas hasta que se mueran todas o que se hayan cumplido las 30 iteraciones, teniendo cada celda o viva o muerta con la misma probabilidad al inicio. Graficar y tabular los hallazgos[1].

## 3. Código

Se debe diseñar y ejecutar un experimento para determinar el efecto de la regla de supervivencia (por lo menos 5 reglas) en la vida de la colonia en una malla de 12 por 12 celdas hasta que se mueran todas o que se hayan cumplido las 30 iteraciones, teniendo cada celda o viva o muerta con la misma probabilidad al inicio. Graficar y tabular los hallazgos [1].

Con el siguiente código se realizan 5 reglas de supervivencia con el fin de graficar su comportamiento de vida durante 30 iteraciones.

La primera regla que se ejecuta es la regla general u obligatoria de 3 vecinos para vivir. La segunda regla parte de la esencia de la primera regla que vive con 3 vecinos, pero tiene un 50 por ciento de probabilidad de vivir con 4 y 2 vecinos, con más o menos vecinos muere.

La tercera regla sigue siendo una extensión de la primera y la segunda ya que en base al comportamiento de la segunda regla, si en la primera iteración logro vivir con 2 o 4 vecinos tendrá la oportunidad de no morir con 1 o 5 hasta 8 vecinos, esto pretendiendo suponer que tiene la posibilidad de ser inmortal.

La cuarta regla ya se independiza de las primeras 3 dando diferentes probabilidades de vida según la cantidad de vecinos que tiene.

Vecinos	Porcentaje de Vida
0	01 %
1	20 %
2	30 %
3	50 %
4	35 %
5	60 %
6	05 %
7	85 %
8	02 %

Cuadro 1: Parámetros de porcentaje para la regla cuatro.

La quinta y última regla nos dice que si tienes un número par de vecinos se tiene el 50 por ciento de probabilidad de vivir y con número impar solo se tiene el 1 por ciento de probabilidad de vida.

Para este código se usó la alternativa de considerar el espacio como un torus (pareciendo una dona). Y se usó el código base de la Dra. Schaeffer que se obtuvo de su repositorio de GitHub[2].

```

1
2 def paso(pos, reglas):
3     fila = pos // dim
4     columna = pos % dim
5     actual_o = actual
6     if fila == 0:
7         actualn = actual_o[[dim-1],:]
8         actualm = actual_o[[i for i in range(dim-1)],:]
9         actual_o = np.append(actualn, actualm, axis=0)
10        fila += 1
11
12    elif fila == dim-1:
13        actualn = actual_o[[0],:]
14        actualm = actual_o[[i for i in range(1, dim)],:]
15        actual_o = np.append(actualm, actualn, axis=0)
16        fila -= 1
17
18    if columna == 0:
19        actualn = actual_o[:, [dim-1]]
20        actualm = actual_o[:, [i for i in range(dim-1)]]
21        actual_o = np.append(actualn, actualm, axis=1)
22        columna += 1
23
24    elif columna == dim-1:
25        actualn = actual_o[:, [0]]
26        actualm = actual_o[:, [i for i in range(1, dim)]]
27        actual_o = np.append(actualm, actualn, axis=1)
28        columna -= 1

```

Listing 1: Código para considerar el espacio como un torus

```

1  if reglas == 1:
2      return 1 * (vecinos == 3)
3
4  elif reglas == 2:
5      if vecinos == 3:
6          regla = True
7
8      elif vecinos == 2:
9          regla = 1 * (random() < 0.5)
10     elif vecinos == 4:
11         regla = 1 * (random() < 0.5)
12     else:
13         regla = False
14
15     return 1 * (regla)

```

Listing 2: Código para reglas de supervivencia

## 4. Resultados

En la siguiente imagen se puede apreciar el comportamiento de la gráfica con las 5 reglas en sus 30 iteraciones.

Gráficas de vida durante las iteraciones, de la regla 1 a la 5

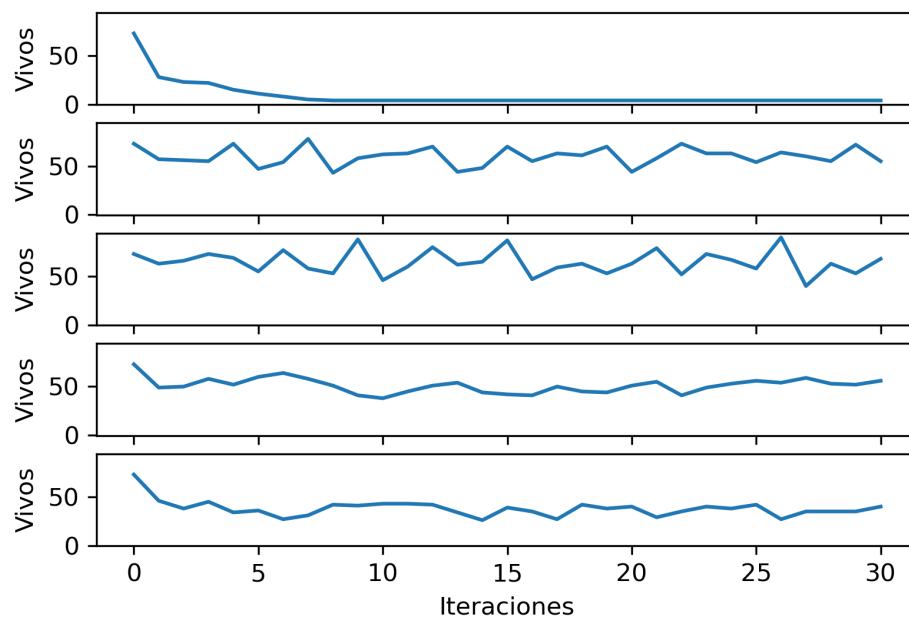


Figura 2: Gráfica de las 5 reglas de supervivencia.

A continuación, se muestran el estado inicial y final del comportamiento de la matriz para cada regla. Las imágenes en .gif para poder observar bien el proceso en las 30 iteraciones se encuentra en el Github [3].

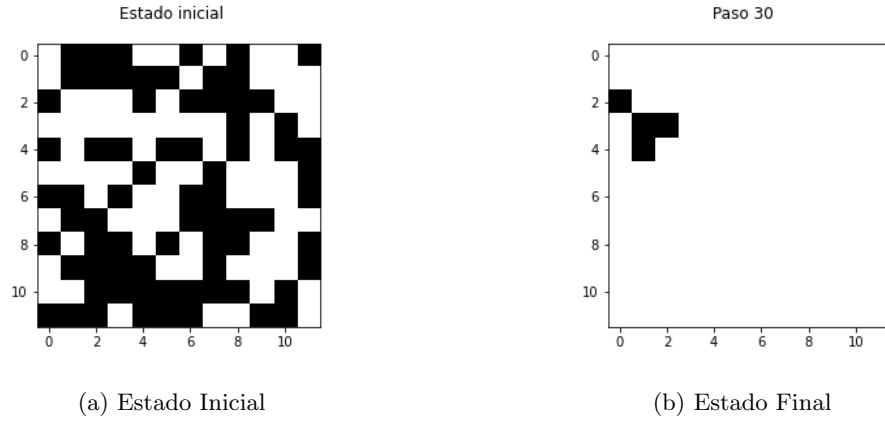


Figura 3: Imagen del estado inicial y final de la regla 1 de supervivencia

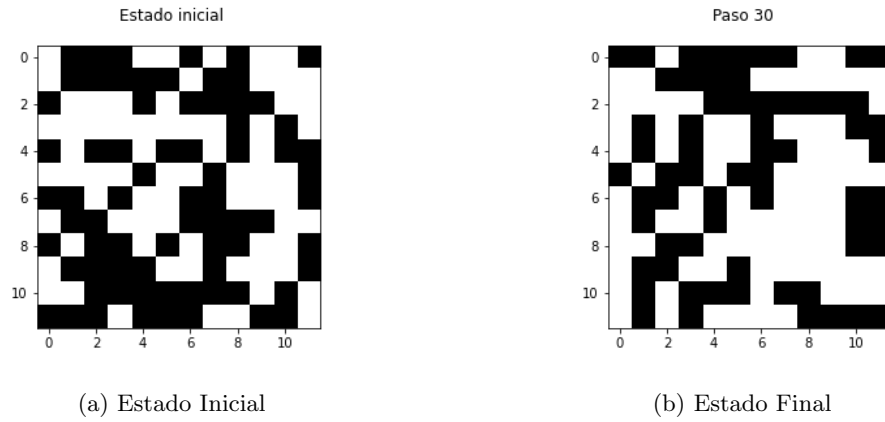


Figura 4: Imagen del estado inicial y final de la regla 2 de supervivencia

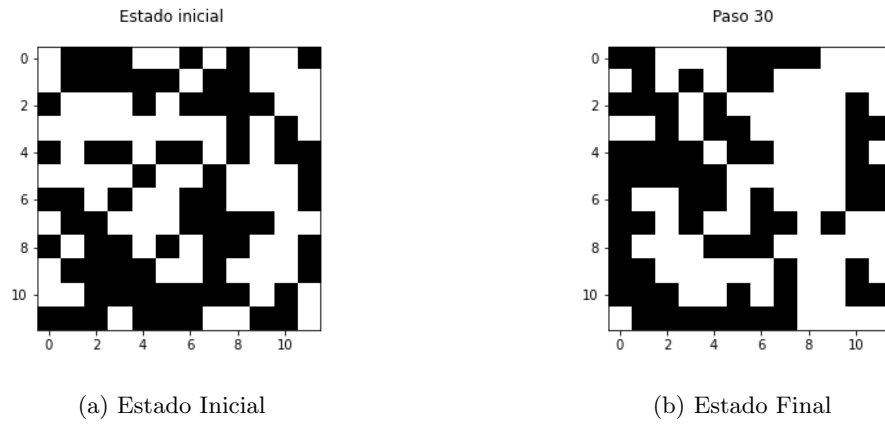


Figura 5: Imagen del estado inicial y final de la regla 3 de supervivencia

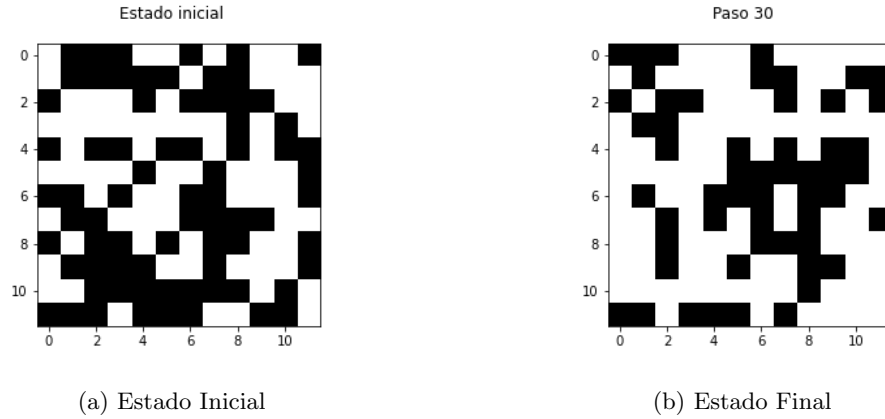


Figura 6: Imagen del estado inicial y final de la regla 4 de supervivencia

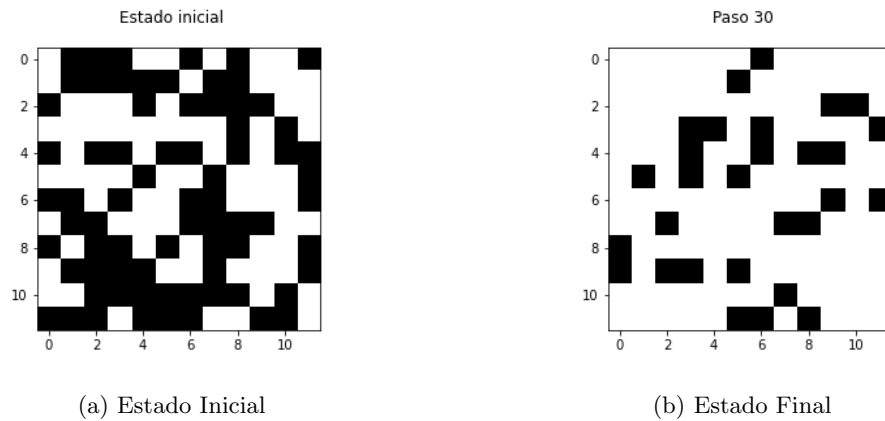


Figura 7: Imagen del estado inicial y final de la regla 5 de supervivencia

## 5. Reto 1

Para este primer reto se debe modificar la simulación para que modele algún tipo de crecimiento (cristalización) en la microestructura de un material. Los núcleos aparecen al azar en celdas desocupadas y se expanden con una tasa constante a celdas vecinas hasta agotar el espacio disponible. Se debe examinar la distribución de los tamaños de los núcleos que no toquen el borde al finalizar la simulación, eligiendo el tamaño de la zona y el número de semillas de tal forma que sean por lo menos la mitad.

```

1
2 dim = 100
3 semilla = 40
4 num = dim**2
5 semill = [x for x in range(1,semilla+1)]
6 valores = [0 for x in range(num)]
7 valores[:semilla] = semill
8 random.shuffle(valores)
9 actual = np.reshape(valores,(dim,dim))

```

Listing 3: Código para creación y posicionamiento de semillas.

```

1 while 0 in m:
2     valor = []
3     for pos in range(num):
4         fila = pos // dim
5         columna = pos % dim
6         na = actual[fila,columna]
7         if na > 0:
8             while True:
9                 for f in range(max(0, fila -1),min(dim, fila + 2)):
10                     for c in range(max(0, columna -1),min(dim, columna +2)):
11                         if actual_n[f,c] == 0:
12                             actual_n[f,c] = na
13                             ac = actual_n[max(0, fila -1):min(dim, fila + 2),
14                                         max(0, columna -1):min(dim, columna +2)].ravel()
15                             acc = np.count_nonzero(ac == 0)
16                             if acc == 0:
17                                 break
18             for i in range(num):
19                 f = i // dim
20                 c = i % dim
21                 valor.append(actual_n[f,c])
22

```

Listing 4: Código para expansión de semillas.

```

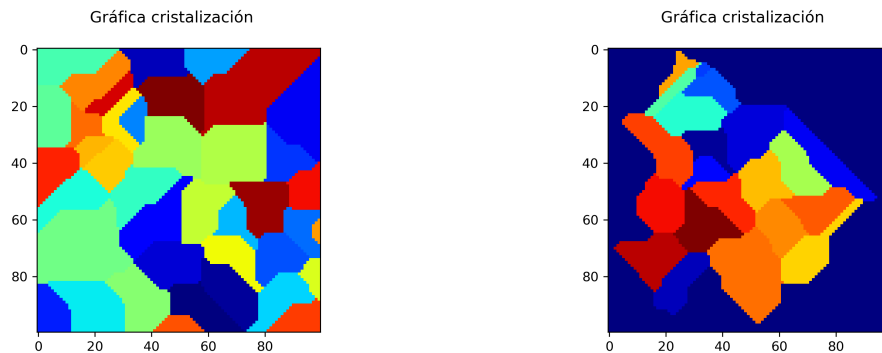
1 for i in actual[0,:]:
2     no.append(i)
3 for i in actual[:,0]:
4     no.append(i)
5 for i in actual[dim-1,:]:
6     no.append(i)
7 for i in actual[:,dim-1]:
8     no.append(i)
9 no = np.unique(no)
10 for i in range(len(no)):
11     for x in range(len(valor)):
12         a = no[i]
13         b = valor[x]
14         if a == b:
15             valor[x] = 0

```

Listing 5: Código para eliminar la orilla

Para mostrar la cristalización que se produjo se creó un archivo tipo gif que se encuentra en el repositorio junto con el código completo[3].

En la figura 8 (a) se muestra la imagen del crecimiento total de las semillas y la 8 (b) eliminando las semillas que tocaron el borde. Para la figura 8 (b) se elimino el 50.73 % de la zona.



(a) Crecimiento total

(b) Eliminación de las semillas que tocaron el borde

Figura 8: Imagen de cristalización

## 6. Reto 2

Para este segundo reto se ejecutará un autómata celular en un modelo tridimensional.

La regla de supervivencia propuesta para este código toma el estado en el que se encuentra la mayoría de sus vecinos es decir si tiene más de dos vecinos en "true" su valor será "true" de no ser así será "false".

En las imágenes de la 9 a la 11 podemos visualizar mejor la regla anterior con el color azul que representa el "true" el color rojo a "false". El código completo para la obtención de estas imagenes se encuentra en Github [3].

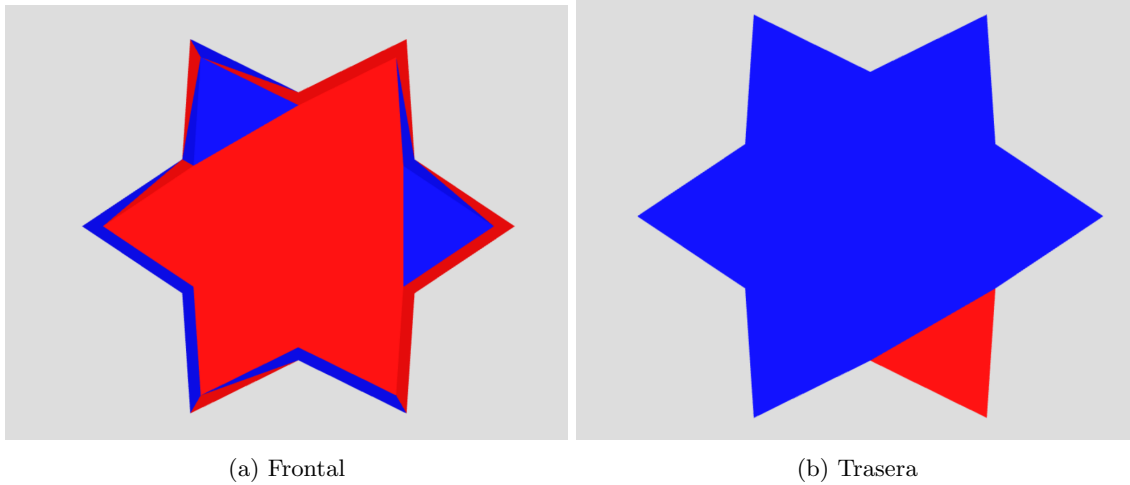


Figura 9: Estado Inicial

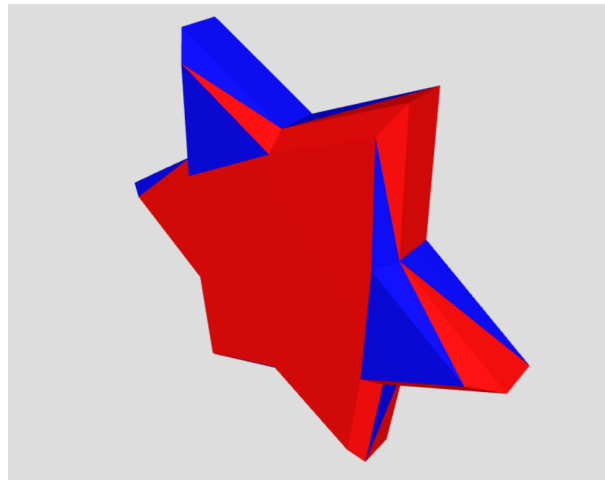


Figura 10: Isométrica

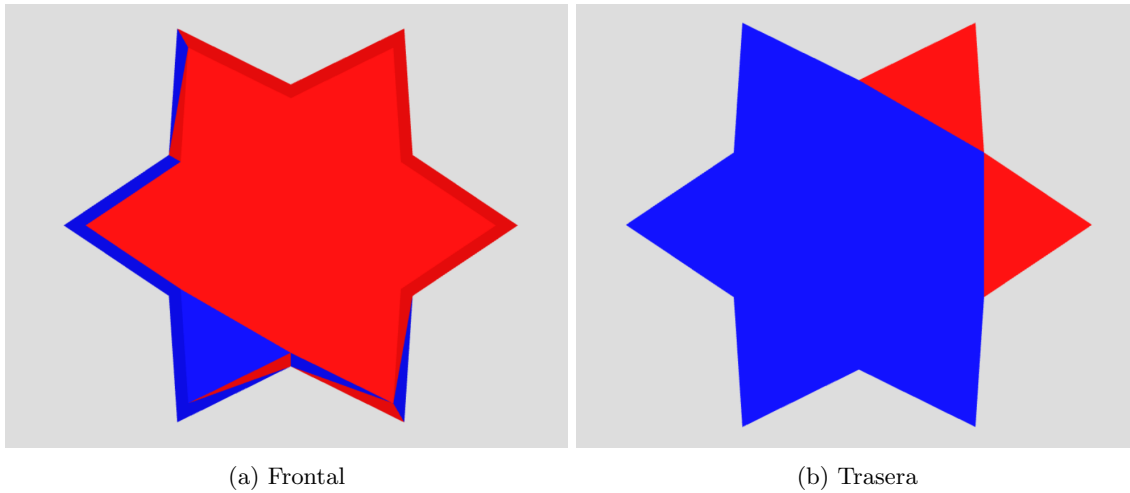


Figura 11: Estado Final

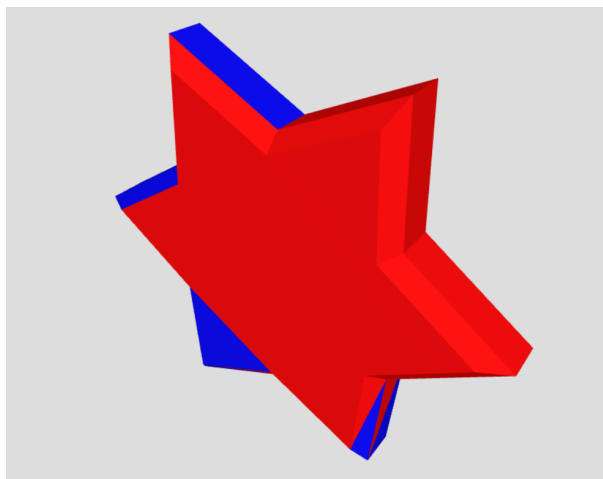


Figura 12: Isométrica

## Referencias

- [1] *Practica 2*. URL: <https://elisa.dyndns-web.com/teaching/comp/par/p2.html>.
- [2] *Repositorio*. URL: <https://github.com/satuelisa/Simulation>.
- [3] *Repositorio*. URL: <https://github.com/Denisse251/Simulation/tree/main/Tarea.2>.