El juego de la vida

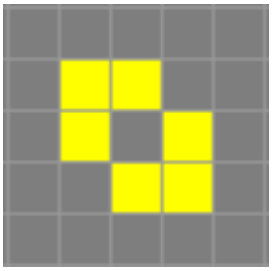
Krystifer Campos Muñoz - 1926189

4 de abril de 2021

1. En una cuadrícula finita con todas sus celdas vivas, todas las celdas en el interior están rodeadas por más de 3 celdas vivas, luego estas mueren en el siguiente estado, las celdas en los bordes (sin contar las esquinas) tienen 5 vecinas vivas, luego estas también mueren, por último, las celdas en las esquinas tienen exactamente 3 vecinas vivas, con lo cual estas sobreviven, el siguiente estado sería entonces el de una cuadrícula con todas las celdas muertas, excepto las esquineras. Luego, ninguna celda muerta en la cuadrícula está rodeada de 3 celdas vivas, ninguna resucita, y las esquineras ya no tienen celdas vecinas vivas, luego estas mueren, la sucesión de estados se ve de la siguiente manera para un arreglo de 4×4.

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

2. El siguiente es otro estado estable



3. Dado que cada casilla puede contener 2 valores (0 o 1), y en un arreglo de $m \times n$ hay mn casillas, el número de estados posibles es 2^{nm}, en el caso de $m = n = 2$, se tendrían exactamente 2^4 posibles estados, estos se listan a continuación.

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$
$$\begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}$$
$$\begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$
$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

4. En una cuadrícula 100×100 (aunque esto ocurrirá en general para una de $n \times n$), las celdas en la diagonal, sin contar las esquinas, están rodeadas por exactamente 2 vecinas vivas, luego en el siguiente estado estas sobreviven, las dos celdas en las esquinas sin embargo; están rodeadas solo por 1 celda viva, luego estas mueren. El proceso se repite para la nueva diagonal, eliminando las celdas en el final, después de $\frac{n}{2}$ pasos (si n es par) o $\frac{n}{2} + 1$ (si n es impar), todas las casillas estarán muertas.

5. El siguiente programa se basa en el código de Python presentado, las diferencias principales son, que en este programa la cuadrícula base es dinámica, es decir; se expande si es necesario para contener todo el patrón que define el estado siguiente. Para ello, entre otras cosas se generaliza la función **exten(mat)** a **expand(mat, level)**, en esta última además de la matriz se debe pasar un entero que defina el nivel de expansión, si $level = 1$, es una expansión de primer nivel, la función genera la matriz pasada como parámetro pero ahora con un borde de ceros adicionales, si $level = 2$, se genera mat pero con dos bordes de ceros adicionales, y así sucesivamente. La misma tarea realiza la función **decrease(mat,level)** pero eliminando los bordes. **around_sum(i, j, arr)** no cambia. La función **rules(mat)** realiza la misma tarea, pero tiene algunos cambios pequeños en los índices. Si **mat** es una matriz de $n \times n$, esta función regresa una matriz de $(n+2) \times (n+2)$, pues se tiene en cuenta el estado de las casillas que rodean los bordes de la matriz inicial. La función **maps(arr)** genera un mapa de color de la matriz, note que es necesario "voltear" de arriba a abajo (o viceversa) la matriz antes de pintarla, para que se lean primero las filas inferiores y en ese orden se pinte, así el dibujo coincide con la representación del array en forma de matriz, esto se hace con **np.flipud()**. Finalmente, en la función **play(arr,steps)**, **arr** es el estado inicial y **steps** es el número de estados que se quieren observar. Estos se muestran a no ser que aparezca un patrón periódico o un estado estable antes de terminar todas las iteraciones, en cuyo caso se termina el programa.

Game Of Life

[10]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors
from matplotlib.colors import ListedColormap
```

[11]:
```
def expand(mat,level):
    n = len(mat)
    ex = np.zeros((n+2*level,n+2*level), dtype = int)
    ex[level:n+level,level:n+level] = mat
    return ex
```

[12]:
```
def decrease(mat,level):
    n = len(mat)
    dec = np.zeros((n-2*level,n-2*level), dtype = int)
    dec = mat[level:n-level,level:n-level]
    return dec
```

[13]:
```
def around_sum(i,j,arr):
    return ( arr[i,j-1] + arr[i,j+1] + arr[i-1,j-1] + arr[i-1,j] + arr[i-1,j+1] +␣
 ↪arr[i+1,j-1] + arr[i+1,j]+ arr[i+1,j+1] )
```

[14]:
```
def rules(mat):
    n = len(mat)
    ex = expand(mat,2)
    nMat = np.zeros((n+2,n+2), dtype=int)
    for i in range(n+2):
        for j in range(n+2):
            s = around_sum(i+1,j+1,ex)
            if s>3 or s <=1:
                nMat[i,j] = 0
            if s==3 and ex[i+1,j+1]== 0:
                nMat[i,j] = 1
            if (s==3 or s==2) and ex[i+1,j+1]== 1:
                nMat[i,j] = 1

    return nMat
```

[15]:
```
def maps(arr):
    Map = ListedColormap(["gray", "blue"])
    plt.figure(figsize=(5,5))
    Arr=np.flipud(arr)
    plt.pcolormesh(Arr, cmap = Map , edgecolors = "silver", lw=0.005, norm= matplotlib.
↪colors.Normalize(0,1))
    ax = plt.gca()
    ax.axes.xaxis.set_visible(False)
    ax.axes.yaxis.set_visible(False)
    plt.show()
```

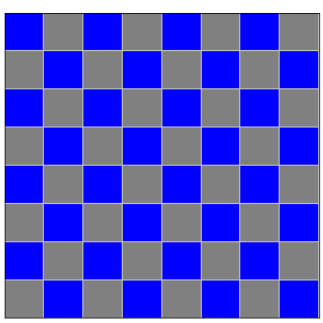
[17]:
```
def play(arr, steps):
    dict={}
    for i in range(steps+1):
        dict[i]=arr
        print("Estado número:",i)
        maps(arr)
        arr = rules(arr)
        for j in dict:
            Arr = dict[j]
            Arr = expand(Arr,int((len(arr)-len(Arr))/2))
            booln_arr = np.equal(arr, Arr)
            if booln_arr.all():
                if i+1-j>1:
                    print("Estado número:",i+1)
                    maps(arr)
                    print("------Aquí empieza un patrón periódico, con período:", i+1-j,␣
↪"pasos.-------")
                else:
                    print("----------Este estado es estable.------------")
                return True
```

Tomando como estado inicial un tablero de Ajedrez, y ejecutando el programa, se obtienen los siguientes estados:

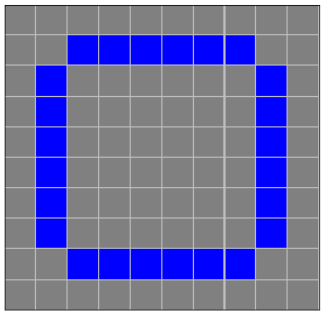
[19]:
```
arr1=np.array(
        [[1,0,1,0,1,0,1,0],
         [0,1,0,1,0,1,0,1],
         [1,0,1,0,1,0,1,0],
         [0,1,0,1,0,1,0,1],
         [1,0,1,0,1,0,1,0],
         [0,1,0,1,0,1,0,1],
         [1,0,1,0,1,0,1,0],
         [0,1,0,1,0,1,0,1]])

play(arr1,5)
```

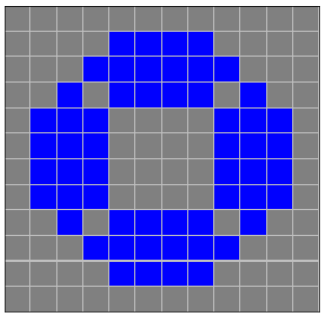
```
Estado número: 0
```



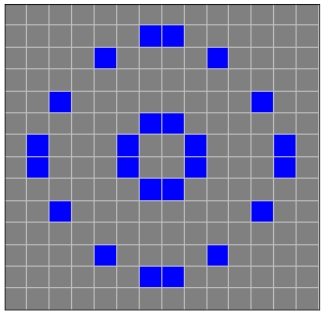
Estado número: 1



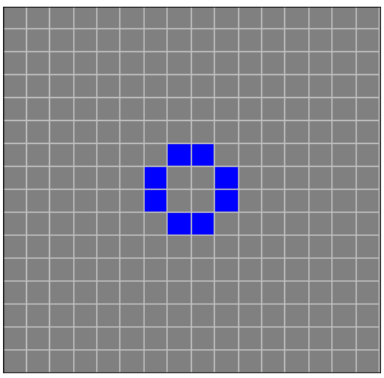
Estado número: 2



Estado número: 3



Estado número: 4



----------Este estado es estable.------------

El espacio métrico del juego de la vida (Conjetura)

Por la forma en la que están definidas las reglas (sobre todo la regla de resurrección), puede que en cada estado sucesivo se amplíe la cuadrícula y lo haga indefinidamente, sin embargo; un estado siempre estará en una cuadrícula finita. Definamos cada estado como $(x)_q$ con q fijo, y en donde q puede variar para definir diferentes estados, $q = 1, 2, 3...z$. Cada estado es una sucesión de sucesiones, $x_q = (x_k)_q^m$, con $k = 1, 2, ..., n$ y $m = 1, 2, ..n$ (para un n propio de cada estado, de forma que la matriz $n \times n$ que lo representa, contenga todo el "dibujo"). Para un q fijo, $(x_k)_q^1$ es la primera fila de la cuadrícula, $(x_k)_q^2$ es la segunda fila y así sucesivamente, y en donde cada (x_k) es una sucesión en $\mathbb{R}$ (de ceros y unos). De los espacios vistos, se podrían considerar l^∞, s y l^p. Como cada estado está definido en una cuadrícula finita, para las sucesiones (x_k) podríamos tomar l^∞ con todos sus elementos iguales a cero a partir de x_{n+1}, y lo mismo para l^p. Por tener mayor familiaridad con la métrica de l^p, y por tener este las propiedades de ser separable y completo (que hasta ahora supongo es una ventaja), tomaremos este como el espacio métrico de cada estado, con lo cual, cada estado se define mediante una sucesión (x_q) en donde cada elemento es una sucesión en l^p. La distancia entre dos elementos de (x_q) estaría definida con la métrica de l^p, d_p. Entonces podemos definir la distancia entre dos estados (x_q) y (x_t) como la suma de las distancias (con d_p) entre $(x_k)_q^j$ y $(x_k)_t^j$, para cada $j = 1, 2, ..n$.

$$d(x_q, x_t) = \sum_{j=0}^{n} d_p((x_k)_q^j, (x_k)_p^j) \tag{1}$$

Considerando ahora el espacio del juego en sí, cada estado es una sucesión de sucesiones, y a su vez los diferentes estados conforman una sucesión $(x)_q$ para $q = 1, 2, 3...z$. Pienso que para describir este espacio habría que hacer una generalización a un espacio, digamos W, en donde cada elemento (partida de juego) en W sea una sucesión de sucesiones de sucesiones en l^p. En cuanto a la métrica, se podría tomar la distancia entre dos partidas o sucesiones de estados, como la suma de las distancias entre cada uno de los respectivos estados (el primer estado de la partida A con el primer estado de la partida B, el segundo estado de A con el segundo de B, etc), en donde la distancia entre estados está dada por (1).
Note que, si por ejemplo hay dos estados cuyas matrices de representación difieren en tamaño, se tomaría el tamaño de la mayor, y la más pequeña se completaría con ceros hasta que tenga el mismo tamaño de la mayor, y si dos partidas de juego tienen un número diferente de estados o de pasos, se tomaría el tamaño o número de pasos de la partida que tiene el mayor número de pasos, y se completarían los pasos de la más corta con estados nulos.