Coding Example: Game of life (NumPy approach)

This lesson discusses the case study Game of life and explains its solution using NumPy approach.

NumPy Implementation

Starting from the Python version, the vectorization of the Game of Life requires two parts,

- one responsible for counting the neighbors
- one responsible for enforcing the rules

Neighbor-counting is relatively easy if we remember we took care of adding a null border around the arena. By considering partial views of the arena we can actually access neighbors quite intuitively as illustrated below for the one-dimensional case:

```
     Z[:-2] | 0 | 1 | 1 | 1 | 0 | (left neighbors)
                  ↓
 Z[1:-1] | 0 | 1 | 1 | 1 | 0 | (actual cells)
                  ↑
Z[+2:] | 0 | 1 | 1 | 1 | 0 | (right neighbors)
```

Going to the two dimensional case requires just a bit of arithmetic to make sure to consider all the eight neighbors.

```
N = np.zeros(Z.shape, dtype=int)
N[1:-1,1:-1] += (Z[ :-2, :-2] + Z[ :-2,1:-1] + Z[ :-2,2:] +
                 Z[1:-1, :-2]                + Z[1:-1,2:] +
                 Z[2:  , :-2] + Z[2:  ,1:-1] + Z[2:  ,2:])
```

For the rule enforcement, we can write a first version using NumPy's argwhere method that will give us the indices where a given condition is `true`.

```
# Flatten arrays
N_ = N.ravel() #create a 1 dimenional array N_
Z_ = Z.ravel() #create a 1 dimensional array Z_

# Apply rules
R1 = np.argwhere( (Z_==1) & (N_ < 2) )
R2 = np.argwhere( (Z_==1) & (N_ > 3) )
R3 = np.argwhere( (Z_==1) & ((N_==2) | (N_==3)) )
R4 = np.argwhere( (Z_==0) & (N_==3) )

# Set new values
Z_[R1] = 0
Z_[R2] = 0
Z_[R3] = Z_[R3]
Z_[R4] = 1

# Make sure borders stay null
Z[0,:] = Z[-1,:] = Z[:,0] = Z[:,-1] = 0
```

Even the above code does not use nested loops, it is far from optimal because of the use of the four `argwhere` calls which makes it quite slow. We can instead factorize the rules into cells that will survive (stay at 1) and cells that will give birth. For doing this, we can take advantage of NumPy boolean capability and write quite naturally:

> **Note:** In the above code we did not write Z = 0 as this would simply assign the value 0 to Z that would then become a simple scalar.

```
birth = (N==3)[1:-1,1:-1] & (Z[1:-1,1:-1]==0) #active cell state
survive = ((N==2) | (N==3))[1:-1,1:-1] & (Z[1:-1,1:-1]==1)#deactive cell states
Z[...] = 0
Z[1:-1,1:-1][birth | survive] = 1
```

If you look at the birth and survive lines, you'll see that these two variables are arrays that can be used to set Z values to 1 after having cleared it.

The Game of Life. Gray levels indicate how much a cell has been active in the past.

Complete Solution

Given below is the complete solution that I have implemented using the NumPy approach.

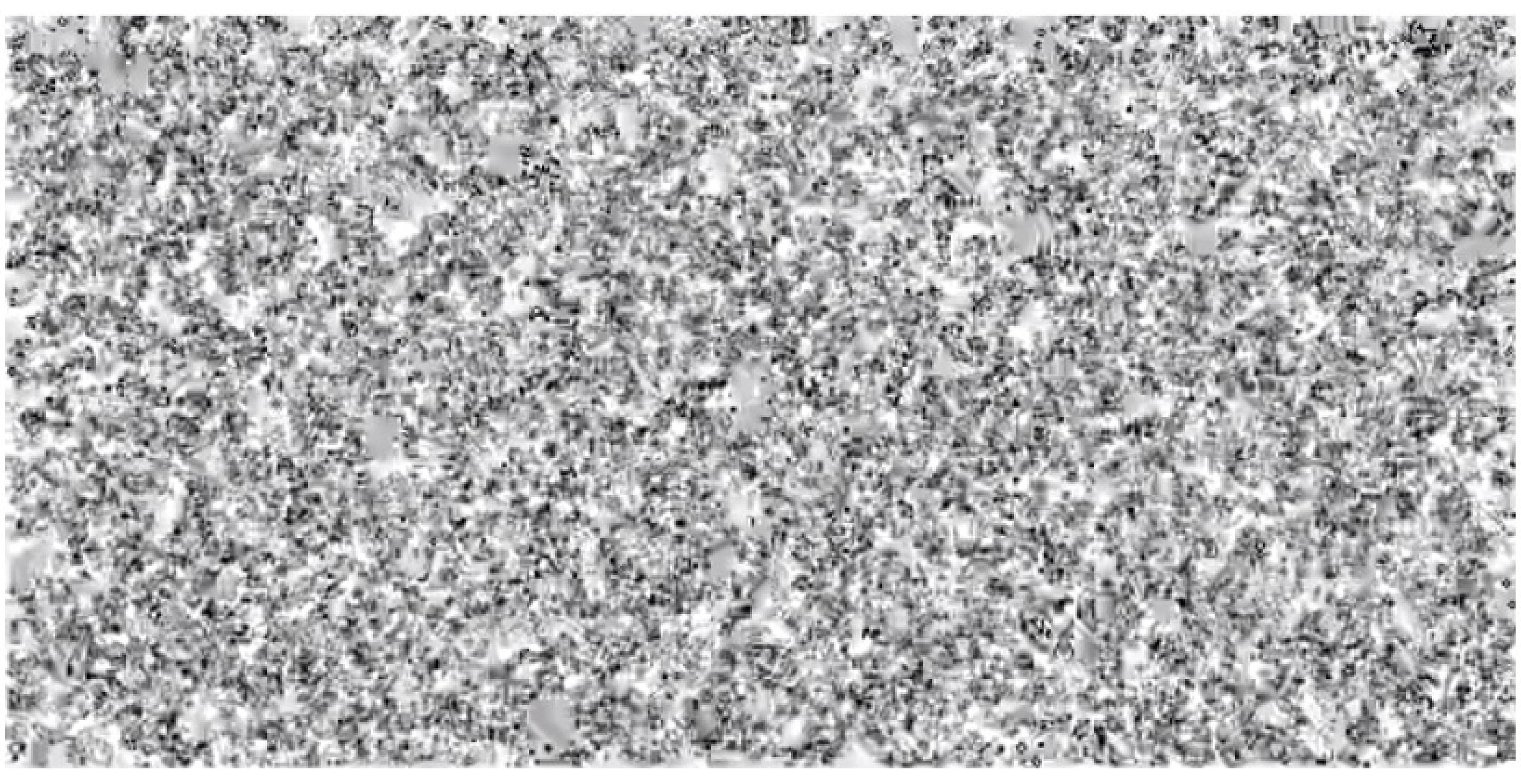
```
# More information at https://github.com/rougier/numpy-book
# -----------------------------------------------------------------------------
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation


def update(*args):
    global Z, M

    N = (Z[0:-2, 0:-2] + Z[0:-2, 1:-1] + Z[0:-2, 2:] +
         Z[1:-1, 0:-2]                 + Z[1:-1, 2:] +
         Z[2:  , 0:-2] + Z[2:  , 1:-1] + Z[2:  , 2:])
    birth = (N == 3) & (Z[1:-1, 1:-1] == 0)
    survive = ((N == 2) | (N == 3)) & (Z[1:-1, 1:-1] == 1)
    Z[...] = 0
    Z[1:-1, 1:-1][birth | survive] = 1

    # Show past activities
    M[M>0.25] = 0.25
    M *= 0.995
    M[Z==1] = 1
    # Direct activity
    # M[...] = Z
    im.set_data(M)


Z = np.random.randint(0, 2, (300, 600))
```

RUN SAVE RESET

output.mp4

Output:

The output of this code would look like the following if you download the file. The animation below will help you visualize the game of life:



Solve this Quiz!

1 What is the purpose of `numpy.argwhere(a)` ?

- A) Find the indices of array elements that are non-zero, grouped by element.
- B) Find the indices of array elements that are zero, grouped by element.

COMPLETED 0% 1 of 2

Now that you have learned about numpy approach for uniform vectorization, let's move to another coding example in the next lesson!