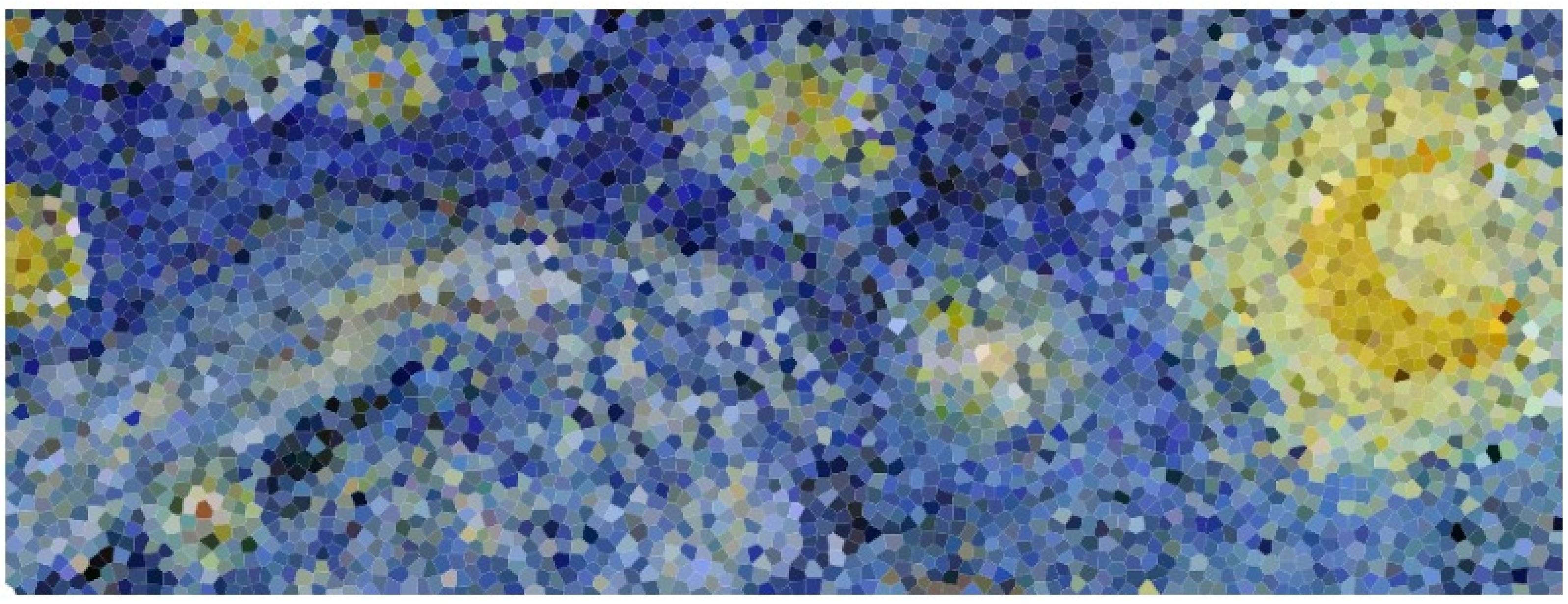
Coding Example: Blue Noise Sampling

The case study discussed in this lesson is based on a study related to the Starry Night painting called "Blue Noise Sampling". Let's learn what is it about!

Problem Description

> *Blue noise* refers to sample sets that have random and yet uniform distributions with absence of any spectral bias. Such noise is very useful in a variety of graphics applications like rendering, dithering, stippling, etc.

Many different methods have been proposed to achieve such noise, but the most simple is certainly the **DART method**.



Detail of "The Starry Night", Vincent van Gogh, 1889. The detail has been resampled using voronoi cells whose centers are a blue noise sample.

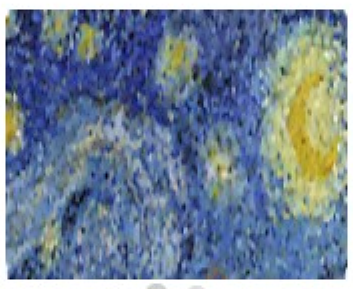
Here's one of the implementations that generates the blue noise:

main.py

voronoi.py

```python
# -----------------------------------------------------------------------------
# From Numpy to Python
# Copyright (2017) Nicolas P. Rougier - BSD license
# More information at https://github.com/rougier/numpy-book
# -----------------------------------------------------------------------------
import numpy as np

def poisson_disk_sample(width=1.0, height=1.0, radius=0.025, k=30):
    # References: Fast Poisson Disk Sampling in Arbitrary Dimensions
    #             Robert Bridson, SIGGRAPH, 2007
    def squared_distance(p0, p1):
        return (p0[0]-p1[0])**2 + (p0[1]-p1[1])**2

    def random_point_around(p, k=1):
        # WARNING: This is not uniform around p but we can live with it
        R = np.random.uniform(radius, 2*radius, k)
        T = np.random.uniform(0, 2*np.pi, k)
        P = np.empty((k, 2))
        P[:, 0] = p[0]+R*np.sin(T)
        P[:, 1] = p[1]+R*np.cos(T)
        return P

    def in_limits(p):
        return 0 <= p[0] < width and 0 <= p[1] < height

    def neighborhood(shape, index, n=2):
        row, col = index
        row0, row1 = max(row-n, 0), min(row+n+1, shape[0])
```

RUN SAVE RESET



DART method

The DART method is one of the earliest and simplest methods. It works by sequentially drawing uniform random points and only accepting those that lie at a minimum distance from every previous accepted sample. This sequential method is therefore extremely slow because each new candidate needs to be tested against previous accepted candidates. The more points you accept, the slower the method is.

The next lesson will cover both, Pythonic and NumPy approach to do the sampling!