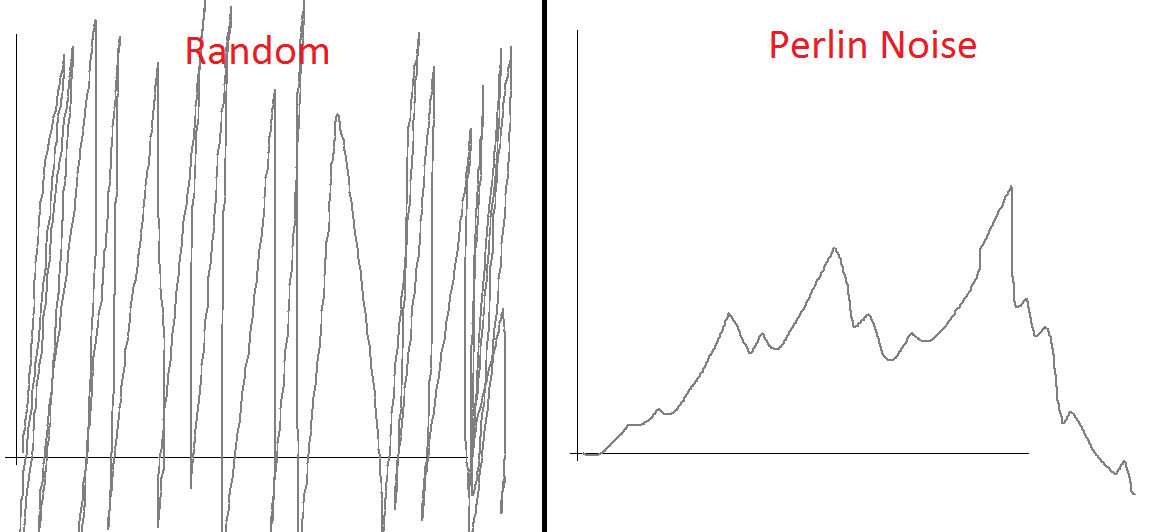
**Perlin Noise**

**Introduction**

Perlin noise created by Ken Perlin, is a type of gradient noise often used to generate computer graphics, and procedural generation. Perlin noise can also be seen as a “smooth” number generator, where numbers feel less random, and more natural. In this document, I will mostly focus on the usage of Perlin noise on procedural content generation.

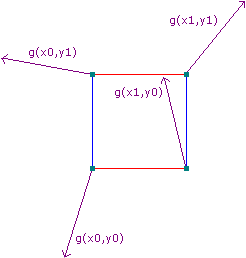
(graph draw from paint)

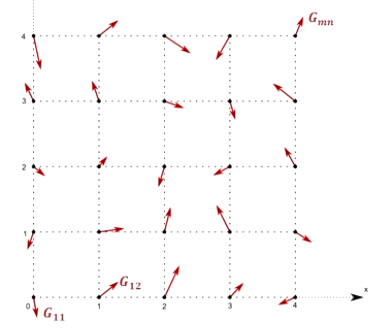
**Algorithm:**

There are three main components to this algorithm: Grid definition, Dot product, Interpolation.

**Grid definition:**

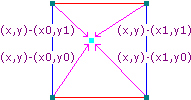
Make a 2 dimensional-grid (up to n dimension), and at each point on the grid, assign a random gradient vector of unit length.

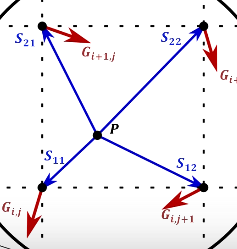




**Dot product:**

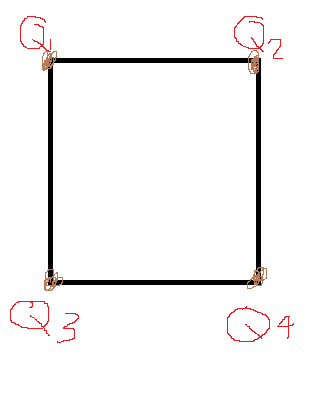
We then take the dot product of the gradient vector and distance vector. The distance vector is determined by an arbitrary point on each square section to each point on that square.



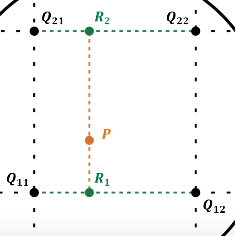


Dot (G . S )= Gx \* Sx + Gy \* Sy

**Interpolation:**



After we calculate the dot product of all vectors and obtains the values, we then interpolate pair of these values to get the desired value that we need. There are many interpolate function, and the one Ken Perlin used in his improved algorithm was: s(t) = 6t5-15t4+10t3 (2 dimension). The reason why the new interpolation function was chosen over the old one [s(t) = 3t3-2t3] because in the second derivative of the old function t = 0, or t = 1 does not equal 0 which caused some problems when using Noise-displaced surface (which described in his revised paper).

R1 = Lerp(Q1, Q2, Px) 

R2 = Lerp(Q3, Q4,Px)

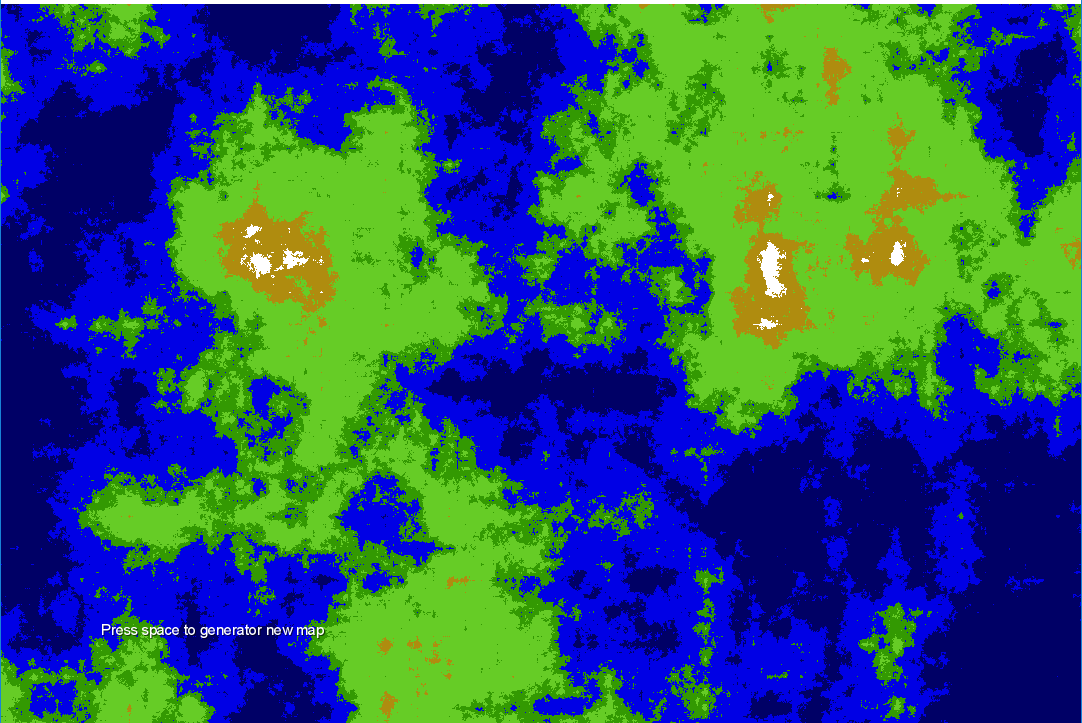
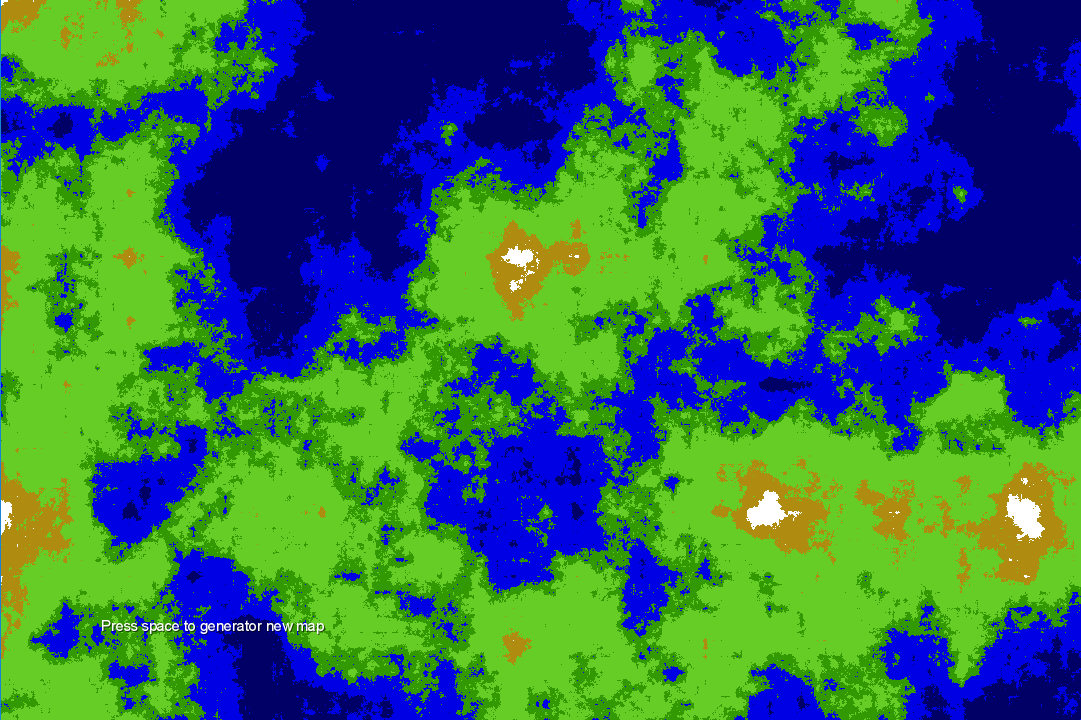
Average = Lerp(R1, R2,Py)

\*Lerp() use s(t) = 6t5-15t4+10t3

After this, we now need to do the same thing for other square in the grid, then we will have all the necessary, nitty gritty of Perlin noise.

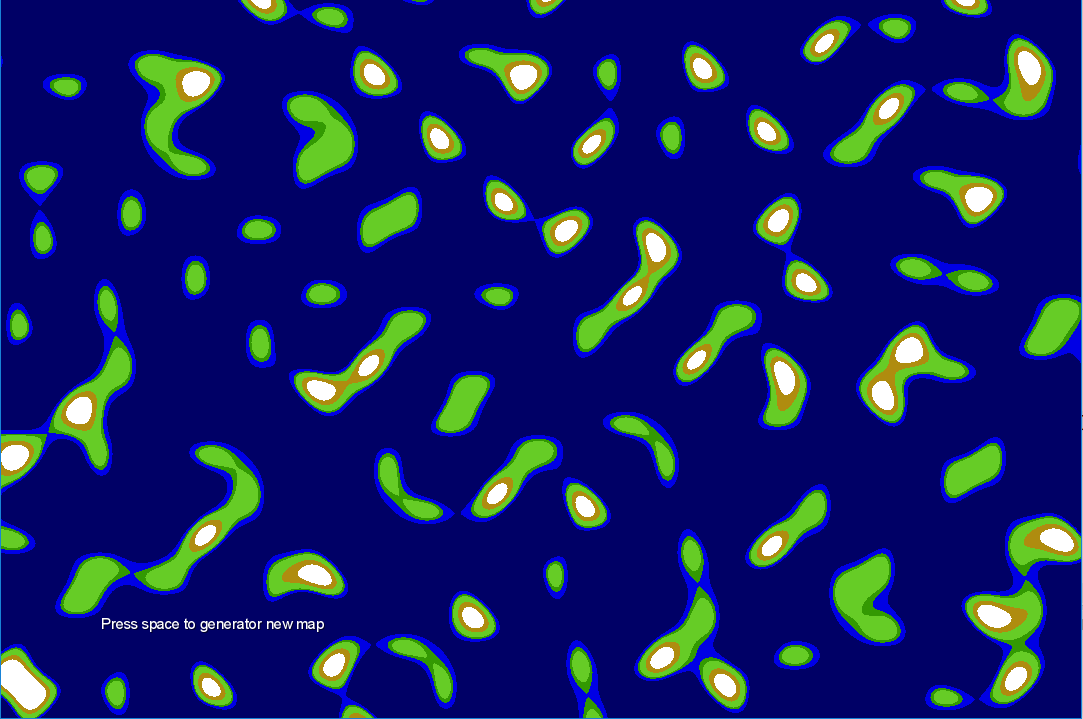
**Demonstration:**

I was not able to implements the code by myself. However, there are plenty of implementation and variations of this algorithm on the internet, and with that I was able to put together a very simple 2d map generator. With the values generated by Perlin Noise algorithms, I mapped those values to a specific pre-defined color and painted on the screen, which result to:



\*Generated by a variation of Perlin Noise

This was generated by a different algorithm called Simplex Noise. I might have implemented this wrong, because the result look really weird.



I will included the codes + the demo

**References:**

<http://mrl.nyu.edu/~perlin/paper445.pdf>

<https://en.wikipedia.org/wiki/Perlin_noise>

<http://flafla2.github.io/2014/08/09/perlinnoise.html>

<https://www.youtube.com/watch?v=Or19ilef4wE>

simplex noise: <http://staffwww.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf>