Programmeerwerk is altijd een (poging tot een) vereenvoudigde weergave van de werkelijkheid. Om de beperking van een vereenvoudigd model compleet te omarmen ben ik begonnen door mezelf regels te geven. Startpunt is dan ook geworden: ik gebruik alleen vierkanten.

Om deze vierkanten gestructureerd te kunnen gebruiken werk ik met een vierkant grid. In het begin van de code staat hoeveel eenheden het grid lang en breed is. Gevolgd door de grote van deze eenheden in pixels, de vergrotingsfactor.

```
int grid = 60;
int f = 10;
```

Vervolgens maak ik een array van de gebruikte kleuren, en stel ik vast dat ik geen omlijningen wil.

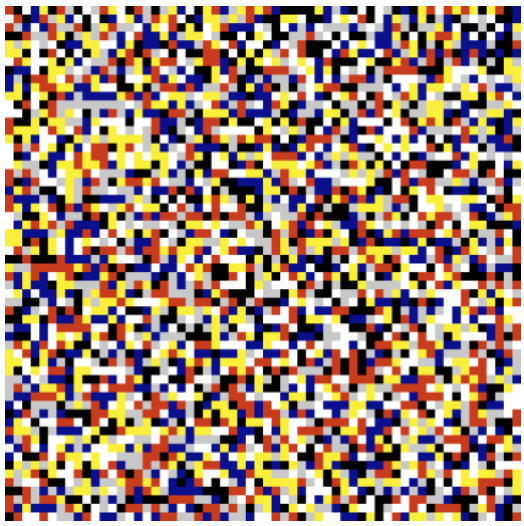
```
color[] c = new color[6];
c[0] = color(0);
c[1] = color(200);
c[2] = color(255);
c[3] = color(250,240,60);
c[4] = color(200,60,30);
c[5] = color(10,15,140);
noStroke();
```

Tijd om daad werkelijk iets te gaan maken. Ik creëer een klein beeld met de afmetingen van het grid, laad de pixels, en loop elke pixels langs om deze te vullen met een willekeurige kleur. Dit doe ik door een getal van 0 tot 6 te laten genereren, en dit element uit de array als kleur te gebruiken.

```
PImage img = createImage(grid,grid, RGB);
img.loadPixels();
for(int i=0; i<grid*grid; i++){
  img.pixels[i] = c[int(random(6))];
}
```

De afbeelding wordt geupdate. Op dit punt wordt het interessant om een venster te hebben, zodat zichtbaar is wat we doen. Het script zou korter kunnen door dit over te slaan, maar dat maakt het geheel zo saai. Het venster geven we de maten van het grid, vermenigvuldigd met de vergrotingsfactor. De gekleurde pixels plaatsen we vergroot in dit venster.

```
size(grid*f, grid*f);
image(img, 0, 0, grid*f, grid*f);
```



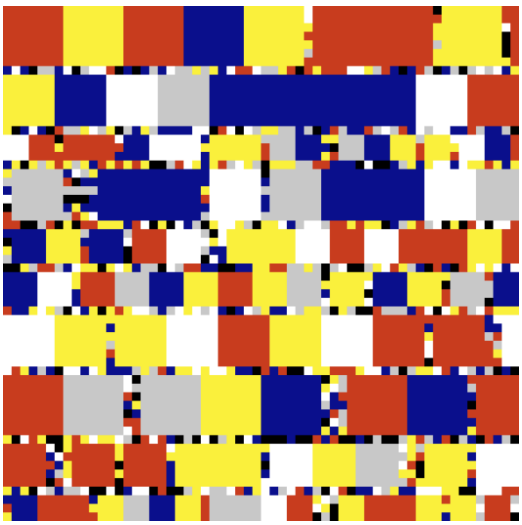
Het resultaat is een kleurrijke ruis. Deze ruis moet de kleinste blokken gaan vormen, die van de horizontale en verticale lijnen. Je zou kunnen zegen dat in deze opbouw de ruis de achtergrond vormt die wordt onderbroken door grotere kleurvlakken. En die vlakken zijn de volgende stap.

We gaan het beeld van boven tot onder vullen met vlakken. Om dit bij te houden hebben we een teller nodig; y. Zo lang y binnen de grenzen van het venster valt moet het volgende stuk script blijven draaien. Binnen dit blok vullen we eerst van links naar rechts, dit houden we bij met x. Hiervoor geldt hetzelfde, zolang dit binnen het venster valt zal een blok code blijven uitvoeren. Dit vullen van links naar rechts doen we met blokken van hetzelfde formaat. Voor elke rij wordt willekeurig bepaald hoe groot deze zijn en dit wordt opgeslagen in s.

```
int y = 0;
while(y<grid*f){
  int x = 0;
  int s = int(random(3,8))*f;
  while (x<grid*f){
```

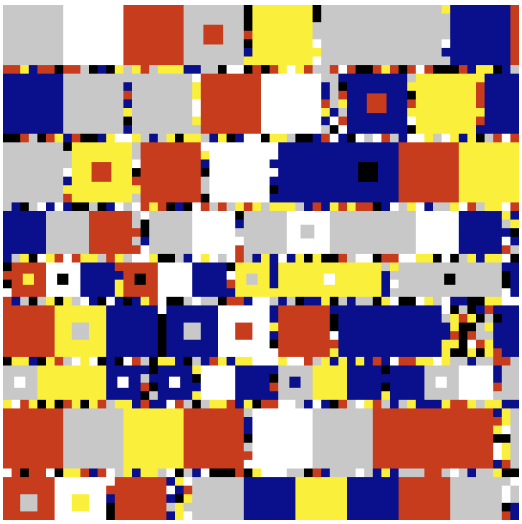
Om verticale lijnen te creëren slaan we soms een stukje over tussen de blokken. Dit wordt wederom bepaald door de random-functie. Als het getal groot genoeg is wordt een blok van willekeurige kleur getekend (maar niet zwart). Als het niet groot genoeg is wordt de teller x een beetje verhoogd, en beginnen we overnieuw. Hier had ik ook x direct kunnen laten verhogen, zonder de if-statement opnieuw te laten toetsen, maar dan was er geen kans geweest op dubbele lijnen, die in het werk wel zitten.

```
if(random(10)>=3){
    fill(c[int(random(1,6))]);
    rect(x, y, s, s);
    x+=s;
   }
   else { x+=f; }
```



We hebben nu lijnen, zowel horizontaal als verticaal, en gekleurde vlakken tussen deze lijnen. In het origineel zijn echter ook vakken waar weer een klein vakje in zit, ongeveer een negende van het grotere vak. Om dit effect toe te voegen komt er een conditie bij die willekeurig vakjes toevoegt in willekeurige kleuren.

```
 if(random(10)>=3){
   fill(c[int(random(1,6))]);
   rect(x, y, s, s);
   if(random(10)>=7){
     fill(c[int(random(0,6))]);
     rect(x+(s/3), y+(s/3), (s/3), (s/3));
   }
   x+=s;
  }
  else { x+=f; }
```



Het patroon is compleet. Of tenminste, compleet genoeg. In het origineel zitten ook nog dubbele vakken

en grote vakken gevuld met grotere vakken dan een negende. Maar het vereenvoudigde model moet eenvoudig blijven, en hier trek ik graag de grens.

Alleen nog even de teller afsluiten, en dan komt het afrondende deel.

```
  }
 y+=s+f;
}
```

Omdat ik mijzelf de beperkingen van vierkanten had opgelegd kwam ik hier op een leuk punt. Hoe geven we die karakteristieke ruitvorm? Het makkelijkste is vier witte driehoeken over het tot nu toe gegeneerde plaatsen. Maar een driehoek is geen vierkant. Een vierkant kan wel een driehoek lijken, als je er maar een kwart van ziet. De oplossing is dan ook, vier vierkanten op de hoeken van het venster, die een alles een slag gedraaid zijn. Daarvoor groeten we eerst onze vriend pythagoras om te kijken hoe groot die vierkanten moeten zijn om precies te passen.

```
fill(255);
float r = sqrt(2*sq(grid*f/2));
```

Vervolgens stel ik de rotatie in, en verschuif ik de vierkanten.

```
rotate(QUARTER_PI);
translate(-r/2,-r/2);
```

De gemakzuchtige oplossing is om nu vier keer de locaties van de vierkanten in te voeren. Maar wat is daar de lol nu van?

```
rect(0,0,r,r);
rect(r,r,r,r);
rect(2*r,0,r,r);
rect(r,-r,r,r);
```

Er zit een ritme in. Kijk bijvoorbeeld naar de tweede waarde. Die is achtereenvolgens 0, r, 0, -r. Dat is net 0, 1, 0, -1. En dat doet weer denken aan een cosinus! Tijd voor een algoritmische plaatsing van vier vierkanten dus!

```
for(int i = 0; i<4; i++){
  rect( (cos(i*HALF_PI)+1)*r, cos((i+1)*HALF_PI)*r, r, r);
}
```

Nu hoeft het beeld alleen nog maar opgeslagen te worden als een praktische tif, zonder lelijke JPG-pixelblokjes.

```
save(year()+"_"+month()+"_"+day()+"_"+hour()+"_"+minute()+"_"+second()+"_grid"+grid+
".tif");
```

En klaar is Erik!

Zo hebben we een originele kopie van Victory Boogie Woogie waar de kracht zit in creatief programmeerwerk vóór de creatie, en het oog van de kijker achteraf, om de mooiste er uit te pikken.