# Vooraf

Ik heb mijn hoofd gepijnigd over hoe je een Victory Boogie Woogie-achtige graphic kan genereren, zonder van te voren exact de positie, kleur en grootte van elk onderdeel te specificeren. Ik heb getracht het originele schilderij te analyseren en geprobeerd om 'regels' te ontdekken. Op basis van deze gevonden regels heb ik gezocht naar een mogelijkheid om deze regels te vervatten in een algoritme.

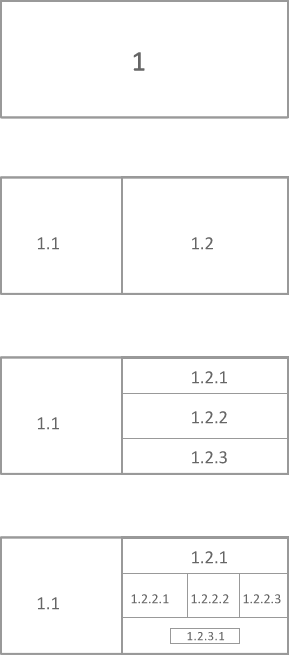
Uiteindelijk ben ik tot de conclusie gekomen dat er een complex algoritme moet worden ontwikkeld om aan alle regels te kunnen voldoen. En het is zelfs de vraag of het überhaupt mogelijk is om de geest van het schilderij in regels te kunnen vangen. Daarom heb ik er voor gekozen om een selectie te maken van de meest in het oog springende regels en een zo eenvoudig mogelijk algoritme te schrijven dat aan deze beperkte set van regels voldoet.

# Uitleg van het algoritme "Chop 'n Punch"[[1]](#footnote-1)

## Tegels

De visualisatie wordt opgebouwd uit een groot aantal tegels (tiles) van verschillend formaat. Een tegel wordt geïnitialiseerd met positie (x en y) en dimensies (hoogte x breedte). Daarna wordt aan de hand van regels en het toeval bepaald hoe een tegel 'evolueert'.

De onderstaande figuur toont een mogelijke ontwikkeling van een tegel in vier opeenvolgende fases. Alle tegels zijn in eerste instantie niet meer dan een rechthoek (zie de bovenste rechthoek met het getal 1). Vervolgens wordt de rechthoek in de volgende fase opgedeeld in twee nieuwe tegels (tegels 1.1 en 1.2). Deze nieuwe tegels kunnen op hun beurt weer worden opgedeeld in nieuwe tegels. Horizontale en verticale opdelingen wisselen elkaar steeds af.



Het opdelen van een tegel in verticale of horizontale delen wordt in het programma aangeduid met 'chop'. De opdeling met referentie 1.2.3.1 is een ander soort opdeling en is in het programma te vinden onder de functie 'punch'. Omdat 1.2.3.1 een tegel is, kan ook dit element zich weer verder opdelen volgens chop of punch.

Het toeval bepaalt:

- het aantal kleinere tegels waarin een tegel wordt opgedeeld

- hoe groot elke tegel is

- of er een chop, punch of helemaal geen opdeling plaatsvindt

- wat de kleur wordt van elke tegel.

## Kleuren

Elke tegel krijgt een willekeurige kleur. Er zijn drie kleurenpaletten gedefinieerd. De grootte van de tegel bepaald uit welk palet wordt gekozen.

Een kleurenpalet kan worden ingesteld met een aantal kleuren en een verdeling van deze kleuren. Hiermee wordt bepaald dat sommige kleuren vaker worden gekozen dan andere kleuren.

De functie initColors() initialiseert 3 verschillende kleurenpaletten. Voorbeeld van het kleurenpalet voor kleine tegels:

c\_small = new ColorRandomizer();

c\_small.addColor(#AEB49D, 1); // dark grey

c\_small.addColor(#E0CB00, 2); // yellow

c\_small.addColor(#C6A400, 2); // light brown

c\_small.addColor(#B71603, 2); // red

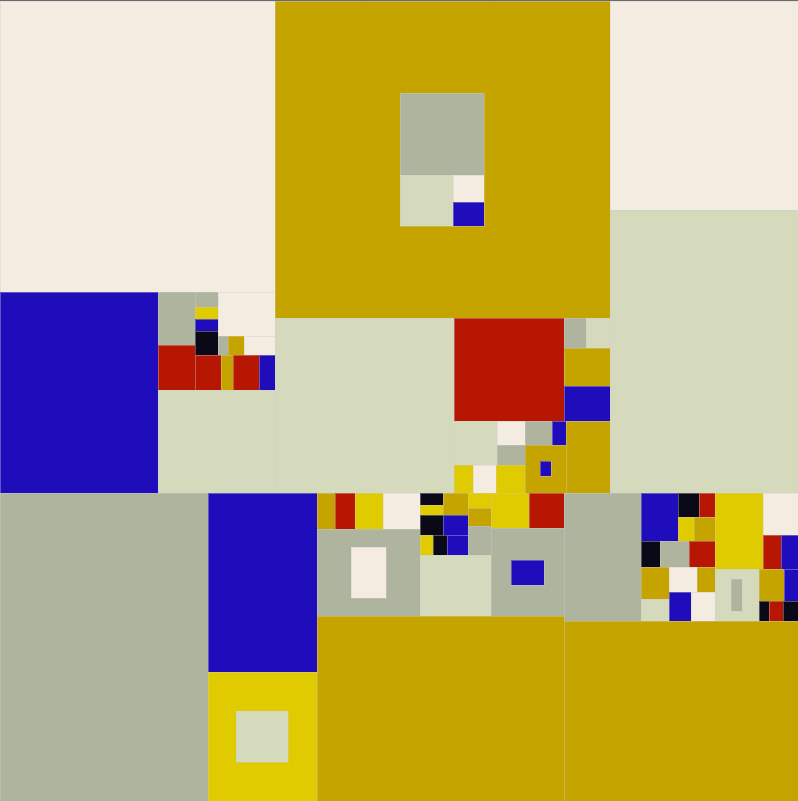
c\_small.addColor(#1F0CBA, 2); // blue

c\_small.addColor(#090918, 4); // black

Het kleurenpalet voor kleine tegels bevat 6 verschillende kleuren. De kleur zwart komt 4 keer zoveel voor als donker grijs en twee keer zoveel als geel.

## Layout

De eenvoudigste methode om de visualisatie te renderen is door uit te gaan van een enkele tegel ter grootte van het scherm en daar het Chop 'n Punch algoritme op toe te passen. Onderstaand plaatje is het voorbeeld van een resultaat dat daarmee is behaald:

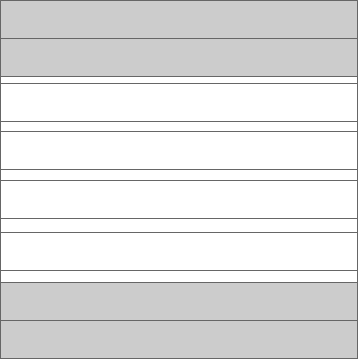


Dit gaat al in de goede richting, maar wat mist zijn de smalle, spoorlijnachtige structuren van het origineel zoals hieronder te zien:



Om dergelijke lijnen te genereren wordt er niet uitgegaan van een enkele 'start-tegel', maar van meerdere start-tegels, waarvan er enkele lang en smal zijn.

Onderstaande layout is de eerste verbetering ten opzichte van het renderen met een enkele starttegel.

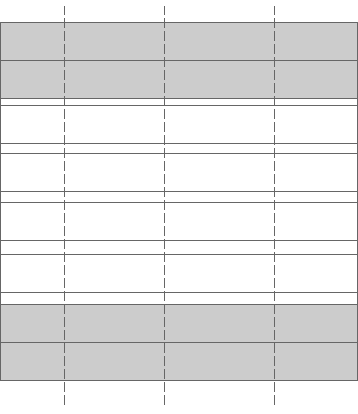


Bovenstaande layout bestaat uit 13 horizontaal georienteerde tegels die het gehele vierkant bedekken. De twee grijze tegels aan de bovenkant en de twee grijze tegels aan de onderkant vormen een spoorloze zone om rust in het uiteindelijke resultaat te bereiken. In het midden wisselen smalle en brede tegels elkaar telkens af.

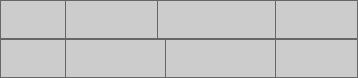
In werkelijkheid zijn deze buitenste rijen niet grijs. De kleur grijs is alleen gebruikt voor de uitleg in dit document.

Toepassing van bovenstaande startlayout resulteert echter in visualisaties waarbij er veel structuur is in horizontale richting en heel weinig structuur in verticale richting. De reden daarvoor is dat tegels altijd in min of meer willekeurige groottes worden opgedeeld. De kans dat er dan een verticale structuur ontstaat is daardoor klein.

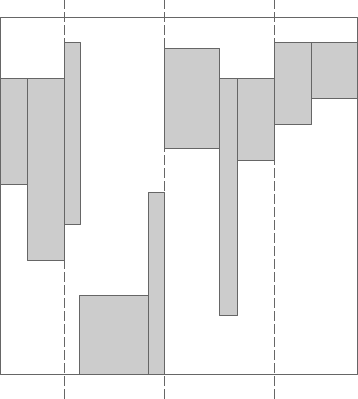
Uiteindelijk is er voor gekozen om de tegels uit bovenstaand plaatje altijd in vier kolommen op te delen. De positie van de stippellijnen in onderstaand figuur vormen een leidraad voor de positionering van de tegelgrenzen. De uiteindelijke tegelgrenzen liggen precies op een stippelijn, of wijken 12 pixels af van de positie van een stippellijn. Daardoor ligt een aantal tegelgrenzen wel precies boven elkaar en een aantal niet. Er ontstaat een verticale structuur die op onverwachte plekken wordt onderbroken.



Hieronder staat een voorbeeld van 8 starttegels voor de bovenste twee tegelrijen. Een aantal verticale grenzen ligt boven elkaar, maar een aantal niet.

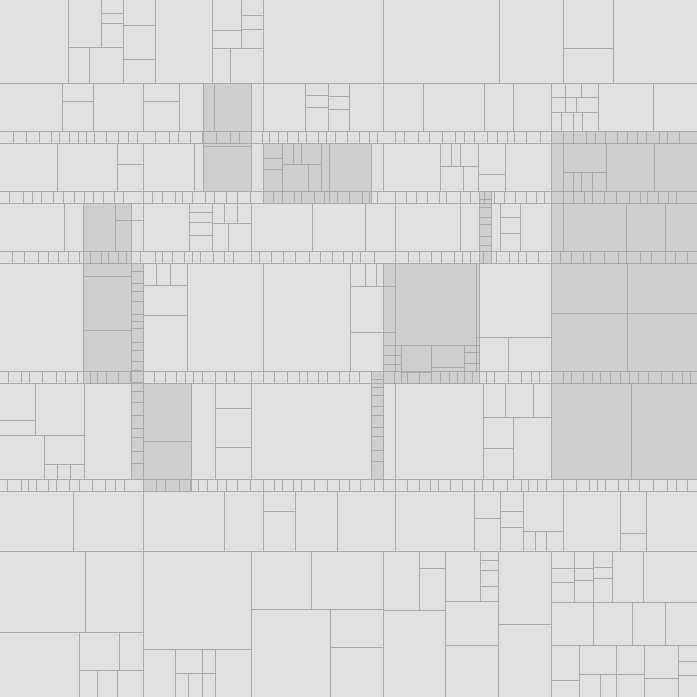


Om nog meer verticale structuur te geven, worden er verticaal georienteerde starttegels toegevoegd die boven op de horizontaal georienteerde starttegels liggen.



Dit is de uiteindelijke layout geworden: een basis van horizontaal georienteerde tegels die het vierkant volledig bedekken en daaroverheen een aantal verticaal georienteerde vierkanten.

Onderstaand figuur is gegenereerd door de tegels de kleur grijs te geven. De lichtgrijze tegels komen voort uit de horizontaal georienteerde tegels, de donkergrijze tegels zijn de verticaal georienteerde tegels die daar bovenop liggen.



Het preciese aantal lijnen en hun posities, het aantal (start)tegels, hun positie, afmeting en kleur, het gebruik van chop of punch wordt door de computer bepaald door veelvuldig gebruik (15 keer) van de random functie binnen Processing. Elk gegenereerd beeld is daarom (waarschijnlijk) uniek.

# Code

De applicatie is gemaakt met processing: http://processing.org/

Na installatie kan de meegeleverde code worden uitgevoerd door te kiezen voor File/Open.

De code mbt hoofdstuk 1.1 is te vinden in het tabblad 'Tile'.

De code mbt hoofdstuk 1.2 is te vinden in het tabblad 'Colors'.

De code mbt hoofdstuk 1.3 is te vinden in het tabblad 'BoogieWoogie'.

In hoofdstuk 1 staat de code in hoofdlijnen beschreven. Uiteindelijk zijn er nog bepaalde verfijningen toegepast. De code is voorzien van commentaar om ook inzicht te geven in deze verfijningen.

# Screenshots

Elke keer wanner er met de muis op de visualisatie wordt geklikt, wordt een nieuwe graphic gegenereerd. Van elke graphic wordt een afbeelding opgeslagen in het mapje 'screenshots'. Dit mapje wordt automatisch aangemaakt.

Het algoritme bevat een groot aantal parameters om het uiteindelijke resultaat af te stellen. De instellingen zijn vastgelegd voor het genereren van een beeld ter grootte van 700 x 700 pixels. In principe kan elk formaat worden ingesteld via de size() functie van Processing, zolang hoogte en breedte gelijk zijn. Als een groter formaat nodig is, bijvoorbeeld voor printen, kan dit dus eenvoudig worden verkregen. Het kan wel zo zijn dat een extreme afwijking een minder mooi beeld oplevert.

# Online versie

De online-versie van de applicatie is te vinden via:

http://tjoadesign.nl/mondriaan/

1. 'Hakken en plakken' mag ook. [↑](#footnote-ref-1)