# Een Programma voor Victory Boogie Woogie

Loe Feijs
l.m.g.feijs@tue.nl

Technische Universiteit Eindhoven

## Inleiding

Het programma dat ik geschreven heb is *voor* Victory Boogie Woogie in de zin dat het een programma is voor het bestuderen van de Victory Boogie Woogie. Het is ook bedoeld als een eerbetoon voor de Victory Boogie Woogie en voor Mondriaan.

Aanleiding voor het schrijven van het programma is de *Call For Code* uitgeschreven door Setup en Gemeentemuseum Den Haag voor 7 Maart 2013. Voor mijzelf zijn er twee extra redenen. Ten eerste mijn belangstelling voor het gebied “Creative Programming”, die hoort bij mijn werk. Ten tweede mijn project van eind jaren ‘90 dat heeft geleid tot één programma waarvan de output is geïnspireerd op een aantal *verschillende* compositietypen. Over dat programma heb ik nadien gepubliceerd in Leonardo en in Matematica e Cultura. Mijn programma van toen stopt vóór de Boogie Woogies, die ik destijds als té complex, voor mij althans, beschouwde.

Nu in 2013 heb ik een nieuw programma geschreven, dat mijn oude principes uitbreidt. De nieuwe code is in Processing (Java) in plaats van het achterhaalde Turbopascal en is specifiek voor Victory Boogie Woogie - hoewel de ingrediënten ervan ook voor eerdere compositietypen gebruikt zouden kunnen worden. Mijn claim is dat het programmeren een manier is om naar Mondriaan’s werk te kijken, er vocabulaire voor te ontwikkelen, méér te zien en het beter te begrijpen. De code belichaamt zo nieuwe kennis. Of zoals Michael A. Noll mij recent schreef: “Art, design, mathematics, computers – all are interrelated today.”



## Interpretatie

Alvorens de modellering en de code te bespreken geef ik eerst aan hoe ik persoonlijk de Victory Boogie Woogie probeer te interpreteren, en waarom.

Om te beginnen spreekt het door Carel Blotkamp uitgelegde principe “Destructie als kunst” mij erg aan. Het komt er ongeveer op neer dat Mondriaan eerst van de figuratieve de interpretatie van het werk af wilde. Niet alleen wordt de eerdere “doorbeelding” voortgezet tot en voorbij het ultieme einde waarbij er echt geen boom, kerk, molen, of kustlijn meer te zien is. Daarna is de kunst om ook het herkennen van objecten zoals individuele vlakken of lijnen te beperken. Geen Gestalt. Het “zweven” van vlakjes voor een achtergrond wordt vermeden door de zwarte lijnen. De object-natuur van vlakken wordt verzwakt door oplossingen zoals vlakken die deels op de rand van het canvas liggen en door ambiguïteit bij aangrenzende gelijkgekleurde vlakken. De objectnatuur van lijnen wordt verzwakt door ambiguïteiten zoals verdubbelde lijnen. Het verschil tussen lijnen en vlakken verdwijnt op sommige momenten bijna helemaal door hun unificatie in de gekleurde lijnen eind jaren 30. De Boogiewoogies voegen daaraan een laatste stap toe: de unificatie van het type van het canvas zelf met de objecttypen óp het canvas. Ik omarm dat idee en neem *unificatie* mee als thema.

Natuurlijk ligt de zaak niet echt zo simpel, bijvoorbeeld ook Blotkamp schrijft over de Victory Boogie Woogie in zijn boek: "Het is alsof Mondriaan zich er niet meer om bekommerde dat het ene

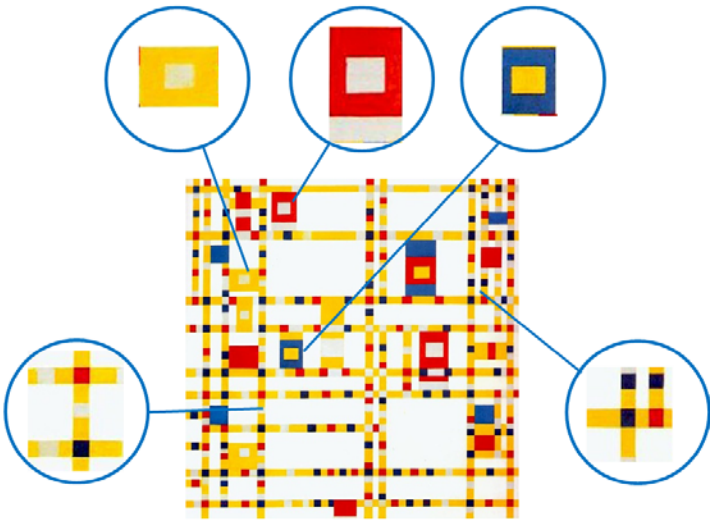
kleurvlakje bovenop het andere lijkt te liggen en dat de witte vlakken zich soms naar voren dringen ten opzicht van de kleur [ ]". Maar programmeren is keuzes maken, althans voorlopige keuzes.

Bij de stap naar de Boogiewoogies is er voor het eerst recursiviteit: in het canvas zitten mini-canvassen. Ik denk dat de tijd rijp was voor recursiviteit. Het Droste cacao blik, met wat nu het Droste-effect heet, was er al sinds circa 1900. Eind jaren 30 en begin jaren 40 werkte Alonzo Church en zijn medewerkers de theorie van lambda calculus en recursieve functies uit – in Princeton, een uurtje rijden van New York (hoewel ik geen aanwijzingen heb dat Mondriaan en Church elkaar kenden). Ik neem *recursiviteit* mee als interessant thema.

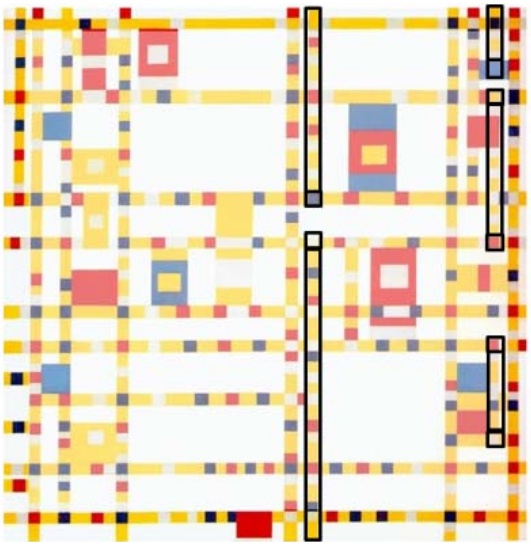
Mijn hypothese is dat er zoiets zou kunnen bestaan als Boogiewoogies en misschien zelfs *het type* van de Victory Boogie Woogie. Bij eerdere Mondriaanwerken zijn die typen wel te herkennen, maar nu is de hypothese riskanter: er zijn maar twee Boogiewoogies en er is maar een Victory Boogie Woogie en die is zeer complex en on-af (letterlijk "work in progress", zoals Jaffé schrijft). Ik heb dus last van onwetendheid wat het algemene idee is (terwijl ik wel een algemeen idee kan hebben van de vlakjestypen uit 1917 of van de “ladder”-typen van eind jaren 30 ). En niemand weet hoe het werk zou zijn geworden. Natuurlijk heb ik gekeken naar het onderzoek, bijvoorbeeld hoe het werk zou zijn zonder de plakband en het papier (Hoofdstuk zes in het boek van Van Bommel, Janssen en Spronk ). En gegevens over het proces van Mondriaan zelf bij het werk aan de Victory Boogie Woogie, bijvoorbeeld uit dat zelfde boek en de Catalogue Raisonné.

Zelf kies ik ook ervoor te steunen op inspiratie die in eerder Mondriaanwerk te vinden is. De structuur van het gele-lijnenrooster heb ik éérst zo uitgewerkt dat ik Broadway Boogie Woogie-achtige composities kon maken. Mijn programma maakt een rooster dat Broadway-inspiratie heeft. Het is wel zeker dat Mondriaan in Broadway Boogie Woogie innovaties beproefd heeft waarvan sommige in de Victory Boogie Woogie terugkeren. In elk geval zijn dat de gele lijnen met daarop vlakjes, vaak geroemd als "ritmisch". De gele lijnen zijn niet nieuw (ze zijn er al in de compositie met gele lijnen uit 1933), de ritmevlakjes wél. En ook nieuw is de vinding van de mini-canvasjes die als brugjes tussen de gele lijnen liggen.

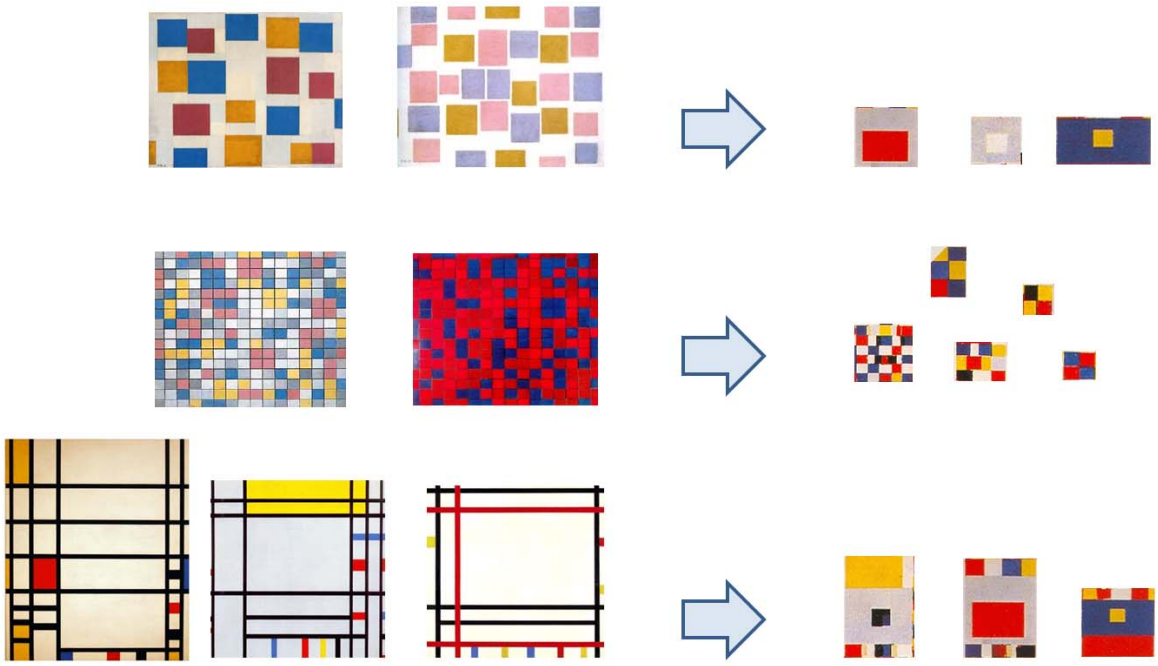
Subtieler maar voor mij wel interessant zijn de vlakjes die precies op de intersectie van een horizontaal en een verticaal liggen, die voorkómen dat er een diepte-werking ontstaat. In Victory Boogie Woogie gaat dat weer anders, mij nog niet zo duidelijk, en ik kies ervoor dichter bij de Broadway inspiratie te blijven. Eigenlijk zijn het niet alleen de intersecties, ook als een verticaal tegen een horizontaal gekomen (gebotst) is en daar gestopt, dan nog is er een intersectievlakje tussen de (hypothetisch) verlengde verticaal en de horizontaal. Soms zijn er zelfs een heel eind verder nog verdwaalde intersectievlakjes. De volgende figuur toont de Broadway Boogie Woogie met het gele-lijnenrooster en de ritmevlakjes. Drie van de mini's en enkele van de intersectievlakjes zijn uitvergroot.



Er is nog een Broadway-inspiratie die mijn aandacht trok: de twee- en drieling verticalen. Meerdere interpretaties zijn mogelijk, dat een tweeling ooit één lijn was die nu onderbroken is, of dat het twee lijnen zijn die in elkaars verlengde liggen (zie figuur). Later in mijn model en implementatie werk ik met de tweede optie.



In de Victory Boogie Woogie zie ik niet alleen veel méér mini's maar ook van verschillende soorten. Die mini's zijn misschien geïnspireerd op eerdere typen van Mondriaan's werk zoals de 1917 vlakjes, de dambord-achtigen uit 1918 en 1919 en de plein-achtigen (Place de la Concorde 1938-1943, Trafalgar Square 1939, New York 1941). De ondergrond van de horizontale lijnen (met ritmevlakjes) in de mini's zie ik als meer wit dan geel. Onderstaande figuur geeft het idee.



Ik interpreteer de mini's als een retrospectief van deze typen uit 1917-1941. Ikzelf meen ook te zien dat sommige van de oplossingen die hier terugkeren eigenlijk destijds niet onproblematisch waren, en ook in pure vorm niet lang voortgezet zijn (slechts twee damborden-achtige werken,

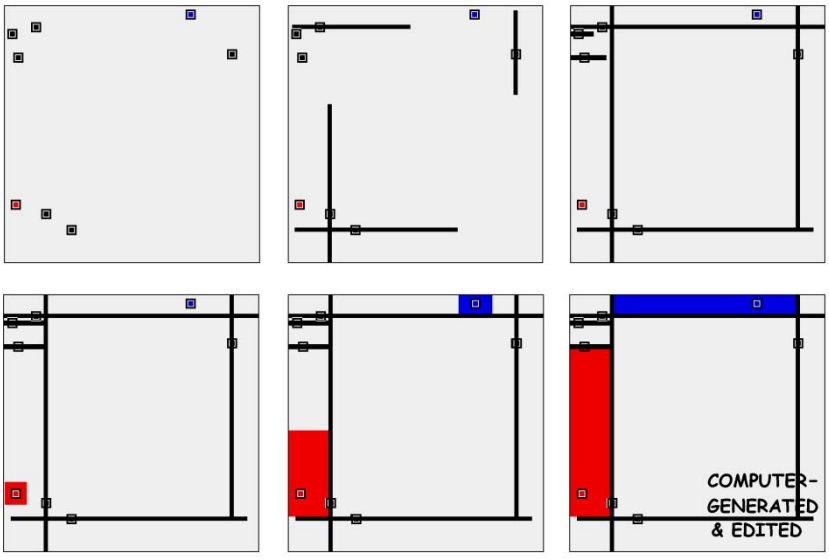
bijvoorbeeld). In die zin vind ik het een mooie gedachte dat Mondriaan er alsnog een héél mooie plek voor gevonden heeft: in de Victory Boogiewoogie.

## Modellering

Het kader van object-georiënteerd modelleren en programmeren is geschikt om de eigenschappen, verschillen en overeenkomsten tussen object-typen in kaart te brengen en ook om specifieke objecten te construeren, analyseren, zo-nodig op te ruimen en weer te geven. Ik kies één unificerend object-type, dat van de *cel*.

Ik combineer die keuze met het verder beproeven van een groeimodel. Een cel is dan een twee-dimensionale gekleurde rechthoek met nog enkele extra Boolese en numerieke eigenschappen die groei-gedrag regelen. De cellen groeien vanuit kleine kernen totdat ze tegen elkaar aanbotsen en zo al groeiend hun uiteindelijke ligging en uitgebreidheid bereiken. De groei gebeurt in stappen. In één stap maakt elke cel een groeistap, die bestaat uit een tentatieve uitbreiding, gevolgd door een "backtracking" als blijkt dat die stap een grensoverschrijding zou veroorzaken. Sommige cellen groeien horizontaal (bijvoorbeeld de cellen die horizontale lijnen worden), andere verticaal, weer andere horizontaal én verticaal. Sommige houden een beetje afstand (epsilon), anderen gaan tot de grens. Sommigen groeien vanaf tijdstip nul, anderen pas als een stappenteller een bepaalde waarde bereikt. Het is de kunst zoveel mogelijk kennis over Mondriaan op te hangen aan de cellen, en zo min mogelijk ad-hoc modellering te doen voor specifieke celtypen of specifieke cellen.

De initiële ligging van de cellen kan gemodelleerd worden met kansverdelingen, zodat een semi-random generator gebruikt kan om specifieke keuzes te maken binnen die kansverdelingen. Elke keer als alle keuzes random gemaakt worden, ontstaat een andere compositie. Sommige van die resultaten zijn "beter" dan andere, en het beoordelen daarvan is onderdeel van mijn proces van continue aanpassing. Het aardige van dit model en de erop gebaseerde programma's is dat één celgroeimachine gebruikt kan worden om een grote variatie van compositietypen te produceren. Een destijds gepubliceerd voorbeeld is onderstaande celgroei en output van mijn oude Turbopascal programma. De output hoort bij een eind jaren 30-type compositie.

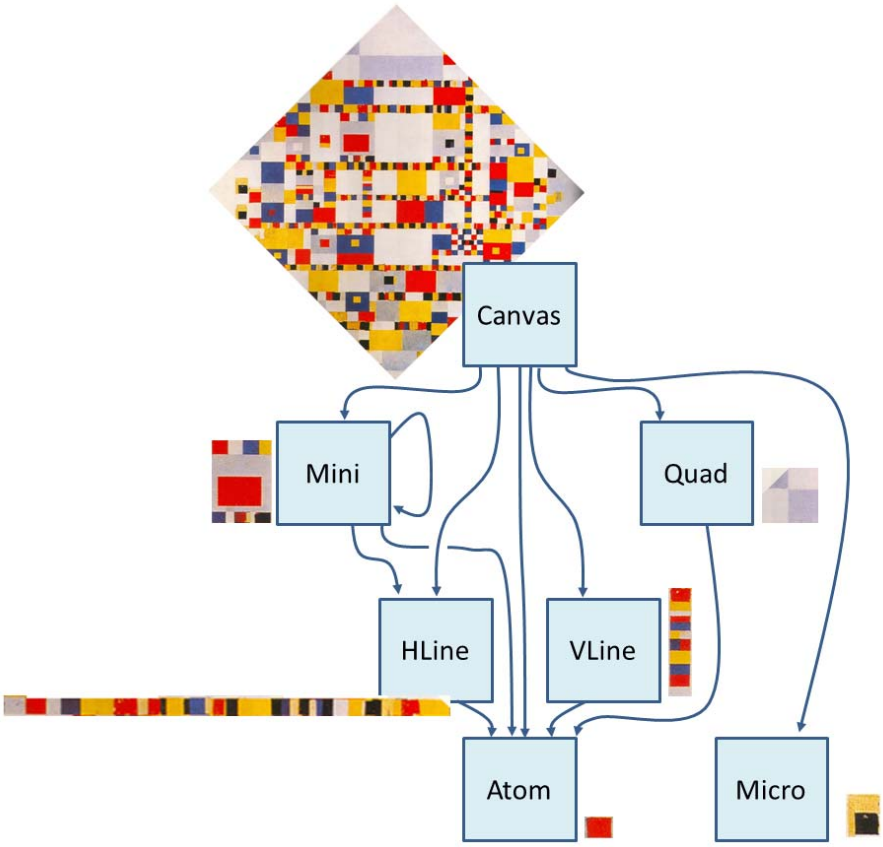



Nu komt de *recursiviteit* erbij. Voor Mondriaan kwam die voorzichtig met Broadway Boogie Woogie in 1943, met name met de vinding van de mini-canvasjes die als brugjes tussen de gele lijnen liggen. In mijn nieuwe model zit het als volgt: elke cel heeft een *ouder*, op het canvas de grotere cel waarin deze cel bevat is. Elke cel heeft als eigenschap een verzameling van cellen, die op hun beurt

ook weer kleur, ligging en groei-eigenschappen hebben. En eventueel zelf ook weer een verzameling van cellen. Belangrijke bewerkingen op een cel zijn veelal recursief (laten groeien, tekenen, tellen van het aantal cellen). Het canvas zelf is een (grote) cel. Ik vind het een goed idee om die recursiviteit diep in mijn hele model in te bouwen.

De verzameling van alle cellen en hun bewerkingen vormt een klasse. De principes van object-oriëntatie kunnen nu worden benut: inkapseling, vererving, en polymorfisme. Vererving (inheritance) is geschikt om speciale gevallen te beschrijven in subklassen (van de celklasse). Canvas, horizontale lijn, verticale lijn, mini en atoom vormen zulke subklassen. Een *mini* is een mini-canvas, een klein schilderijtje binnen het grote werk. Een *atoom* is een cel die geen inwendige structuur heeft anders dan zijn kleur en ligging. Er zijn nog twee klassen: *quadrupels* en *micro's* die, toegegeven, een beetje ad-hoc zijn.

De hiërarchie qua vererving is zeer overzichtelijk: Canvas, HLine, VLine, Mini, Atom, Quad en Micro erven van Cel. De hiërarchie qua "containment" van de elementen van deze klassen is te zien in de volgende figuur. Dit is hoe de ene concrete cel de andere bevat (en omgekeerd, hoe de ouder-relaties liggen), niet het erven van eigenschappen. Merk op dat mini's recursief kunnen zijn: in een mini zit mogelijk weer een kleinere mini.

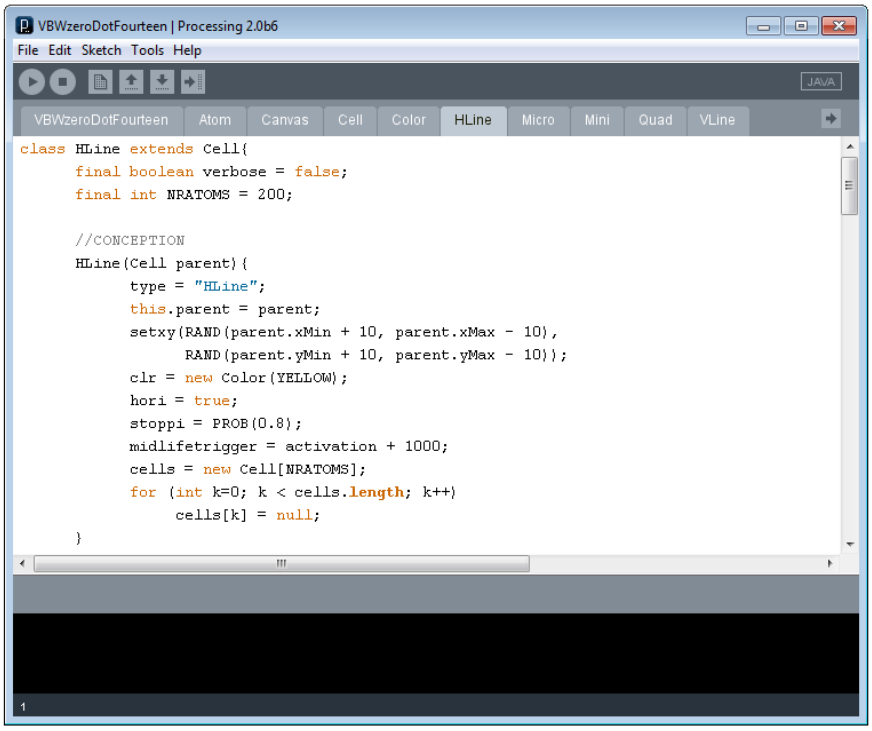



Om goed om te kunnen gaan met de Broadway-achtige oplossing voor de intersectievlakjes tussen HLine en VLIne hebben sommige cellen twee ouders: parent en co-parent (met een knipoog naar het eigentijdse begrip co-ouderschap).

## Code

De code is geschreven in Processing, een open source project van Casey Reas and Benjamin Fry. Ik gebruik versie 2.0. De taal is in essentie Java, maar Processing is een omgeving die de taal laagdrempelig en aantrekkelijk maakt voor creatief werk. Er zijn enkele technische probleempjes,

zoals het niet goed werken van `getClass`, `private`, en `public`, maar daar valt omheen te werken zonder grote nadelen. Mijn functies en variabelen hebben Engelstalige namen en dito commentaar. Een indruk van de code in de context van de Processing omgeving is te zien in de figuur. Elke `Class` staat in een eigen bestand, en in de omgeving staan de verschillende celklassen keurig naast elkaar in tabbladen.




Er zijn tien klassen: naast `Cell` en de zeven klassen die ervan erven zijn er nog `Color` en `VBW` (het hoofdprogramma, een subklasse van `PApplet`). De typische code-opbouw van een `Class` is als volgt:

```
//CONCEPTION
  Class, insert, setxy, copy, ..
//COMPOSITION
  setup
//ANALYSIS
  boxed, bumped, rated, ..
//GROWTH
  Grow, ..
//APOPTOSIS
  exit, purge, ..
//TRIGGERED ACTION
  trigger
//PRESENTATION
  draw, count, rontgen, tell, ..
```

Enkele woorden per sectie:

CONCEPTION is het deel met de celspecifieke eigenschappen, afgeleide functies voor coördinaatbepalingen, constructoren en aanverwante zaken.

COMPOSITION gaat over het plaatsen van onderdelen. Hier speelt de semi-random generator een grote rol en hier wordt een bovengrens vastgelegd aan het aantal atomen op een lijn, het aantal mini's op een canvas enzovoort. Replay van de random generator met dezelfde `seed` is

een globale optie. De voornaamste functie heet `setup` (naar analogie van de ingebouwde `setup` van Processing).

ANALYSIS gaat vooral over botsing-detectie en "containment" in al hun varianten. Dit is essentieel voor de celgroei. In de generieke `Cell` is dit een zeer uitgebreide, technische en ietwat tricky collectie methoden.

GROWTH bevat de methode `grow`, het hart van de celgroei-machine. Hier wordt gerekend, maar er is niet gestreefd naar uiterste efficiëntie. De botsingsdetectie is voorzien van een algoritme dat de verzameling buren berekent die niet buiten een veilige afstand liggen, bijvoorbeeld 100 pixels, waarbij dus om de 50 groeistappen de buurverzameling her-berekend moet worden (een Cell groeit mogelijk één pixel in elke richting per stap). Sorteren van de atomen binnen een `HLine` en `VLine` ligt voor de hand maar dat doe ik niet omdat het tégen de idee van een generieke celgroei-machine in gaat. Maar zelfs met de oorspronkelijk eenvoudigere kwadratische botsingsdetectie is een moderne PC snel genoeg om in enkele tientallen seconden de compositie te laten groeien. Een verfijnder algoritme met bijvoorbeeld quadtrees ligt voor de hand maar het zou een kanon zijn om op een mug te schieten. Het tekenen kost nu al ongeveer evenveel tijd als het rekenen; de groei is nu mooi op het scherm te volgen. Dat is een voordeel. Er wordt trouwens ook geen gebruik gemaakt van externe bibliotheken, wat het installatiegemak hopelijk ten goede komt.

APOPTOSIS betekent geprogrammeerde celdood (ook essentieel bij groei in de biologie, inclusief die van onszelf). Ik maak ruim gebruik van de welbekende genereer-en-test aanpak: bijvoorbeeld het hele te-grote vierkant wordt vol-gestrooid met lijnkernen en minikernen en daarna worden degenen die buiten de grote ruit liggen, of die in een hoek liggen die ik vrij wil houden, weer opgeruimd. Na veel geëxperimenteer meen ik dat dit qua leesbaarheid nog het beste compromis is. De methode `exit` werkt voor cellen en heet zo naar analogie met de `exit` van Processing.

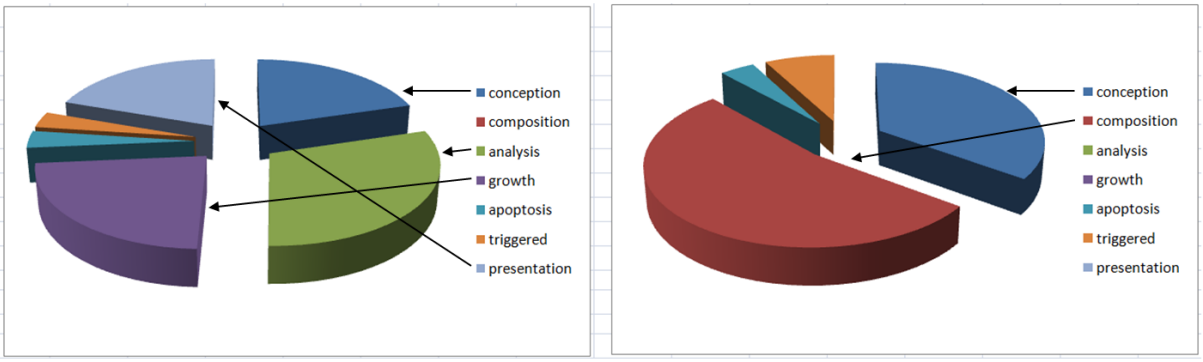
TRIGGERED ACTION gaat over acties die pas uitgevoerd kunnen worden na een vooraf bepaald aantal stappen. Bijvoorbeeld opruimacties binnen een `HLine` of `VLine`, die moeten wachten totdat de lijn in kwestie zelf volgroeid is. Daarna kunnen inderdaad degenen van de inwendige atomen (beoogde "ritmische vlakjes") die qua groei mislukt zijn opgeruimd worden. En voor een mini begint de recursieve invulling pas als de mini zelf enigszins volgroeid is. Vanuit de groei-machine komt dan een call-back, die `trigger` heet. Voor verschillende klassen doet deze polymorfe trigger telkens iets anders: opruimen ingeval van `Canvas`, `HLine` en `VLine`, en een `setup` ingeval van `Mini`. Voor diverse variaties verwijs ik naar de code (de `split` in `Atom` en de `shrapnel` in Quad klinken gevaarlijker dan ze zijn). De `trigger` komt ten tijde van stap nummer `midlifetrigger`, een numerieke cel-eigenschap (met een knipoog naar de menselijke midlife-crisis).

PRESENTATION gaat over het weergeven van resultaten. Allereerst op het scherm met de methode `draw` (naar analogie van de ingebouwde `draw` van Processing). En ook diverse varianten die nuttig zijn bij fout-zoeken en bij studie (twee niet helder te scheiden activiteiten). Een eenvoudig code voorbeeld uit `Cell` laat de recursiviteit mooi zien:

```
int count(){
    int n = 1;
    if  (cells != null)
         for (int i = 0; i < cells.length; i++)
              if (cells[i] != null)
                  n += cells[i].count();
    return n;
}
```

De volgende twee taartpuntdiagrammen geven een indruk hoe de code verdeeld is over de verschillende secties. Het linker-diagram is voor `Cell`, het rechter voor `HLine`. Bij `Cell` springen de complexe botsings- en containment detectie (ANALYSIS), de groeimachine (GROWTH), en de

generieke tekenmethoden (PRESENTATION) eruit. Bij `HLine`, die qua aantal regels overigens veel kleiner is, zit het meeste werk in de feitelijke compositie.



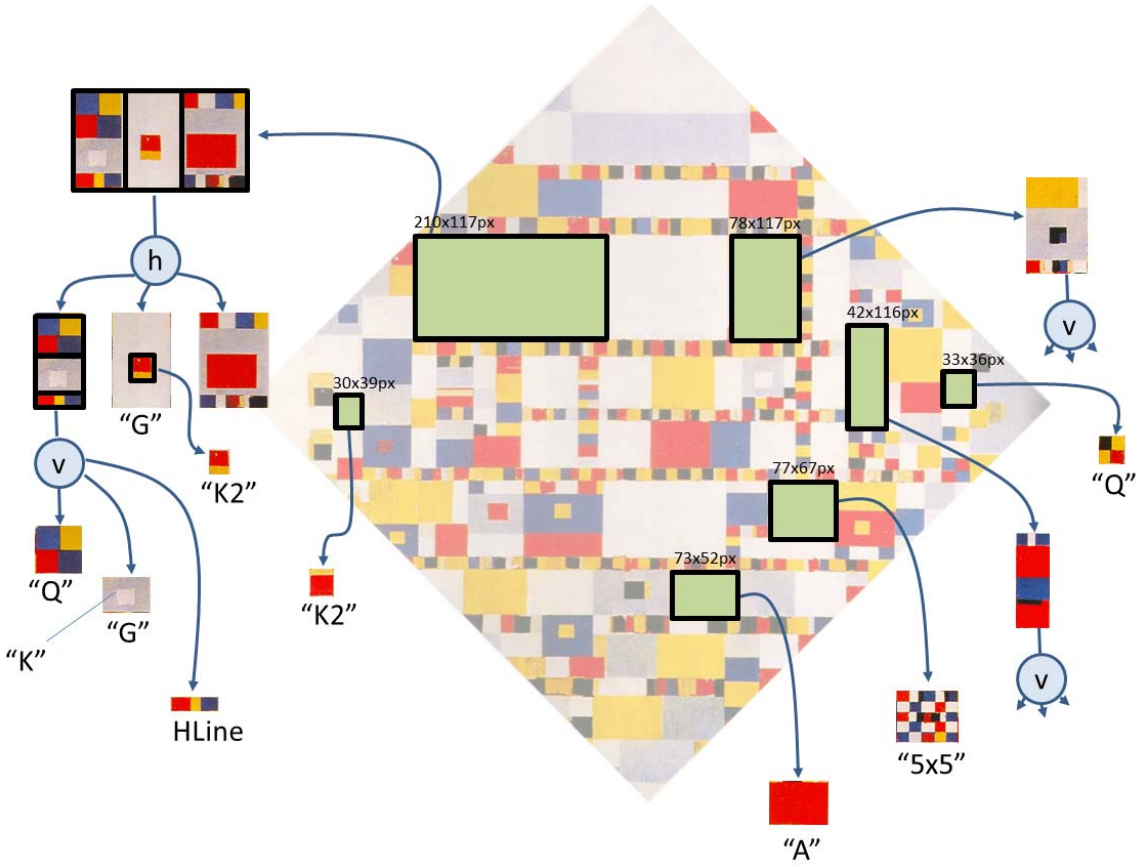

Na dit overzicht nog enkele technisch interessante punten. De ritmevlakjes zijn atomen die geplaatst worden binnen de `HLine` en `VLine` cellen. Ze hebben een `epsilon` die groter is dan nul zodat ze bij de celgroei dus afstand houden tot hun buren. Ze vinden hun plaats op een manier die lijkt op soldaten die opgesteld worden in een gelid en die het commando "naar réchts - ríchten" uitvoeren, waarbij ze een arm verder strekken en/of zich verplaatsen door snel te *dribbelen* totdat de gestrekte arm de nevenman raakt.

Over inkapseling (encapsulatie): het grote voordeel is dat meerdimensionale array's, lange namen en naam-prefixschema's overbodig zijn. Zie bijvoorbeeld de eerder gegeven `count()` waarbij alles werkt op het object `this`, dat impliciet in de context zit. Verder ben ik pragmatisch met de inkapseling omgegaan door soms direct met de coördinaten van een cel te werken zonder aparte `set_` en `get_` methodes.

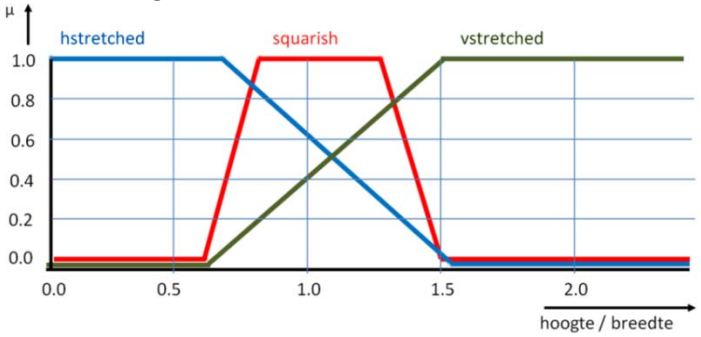
Polymorfisme is over de hele linie toegepast. Terwijl vererving bijna automatisch methoden oplevert met dezelfde namen die veelal ook hetzelfde doen, maakt polymorfisme het mogelijk om bewerkingen die qua uitvoering echt anders zijn en/of zelfs verschillende parametertypen hebben, tóch hetzelfde te noemen mits qua semantiek verdedigbaar. Daarom zijn ook namen als `setup`, `draw` en `exit` uit Processing behouden. Een overzicht is te zien in onderstaande figuur. De vetgedrukte namen zijn van een `Class`. De oranje methoden zitten al in Processing (en verschijnen oranje en vetgedrukt in de editor). Inspringen geeft vererving aan. Onderstreping betekent "override". Schaduwtinten zijn vererfde methoden die niet buiten de eigen klasse gebruikt worden.

| **VBW0.7** | **setup** | **draw** | **exit** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Cell** | | draw | exit | count | rontgen | tell | insert | copy | boxed | bumped | grow | trigger | setxy | |
| **Atom** | | draw | exit | count | rontgen | tell | insert | copy | **boxed** | bumped | grow | trigger | setxy | |
| **Canvas** | setup | draw | exit | count | rontgen | tell | **insert** | copy | boxed | bumped | grow | trigger | setxy | $purge_{123}$ |
| **Hline** | setup | draw | exit | count | rontgen | tell | **insert** | copy | boxed | bumped | grow | trigger | setxy | $purge_{123}$ |
| **Mini** | setup | draw | exit | count | rontgen | tell | **insert** | copy | boxed | bumped | grow | trigger | setxy | |
| **Quad** | setup | draw | **exit** | count | rontgen | tell | **insert** | copy | boxed | bumped | grow | trigger | setxy | |
| **Vline** | setup | draw | exit | count | rontgen | tell | **insert** | **copy** | boxed | bumped | grow | trigger | setxy | $purge_{123}$ |
| **Micro** | setup | draw | exit | count | rontgen | tell | **insert** | copy | boxed | bumped | grow | trigger | setxy | |
| **Color** | | | | | | tell | | | | | | | | |

De recursiviteit wordt geïmplementeerd via de `setup` van `Mini` (met het `trigger` mechanisme ertussen). Ter oriëntatie heb ik gekeken naar de recursiviteit van de echte mini's in de Victory Boogie Woogie. Hoewel ik moet erkennen dat hier meteen ambiguïteiten opduiken, geeft de volgende figuur toch enig houvast. De keuze hoe een gegeven mini verder te de-componeren neem ik op basis van drie gegevens: oppervlakte, hoogte-breedte verhouding, en nesting-diepte. Eigenlijk zou extra context-informatie wel nodig zijn, maar dat is vooralsnog eruit gelaten. De getallen in de figuur zijn in pixels (1 pixel = 2mm "echt"). Dat is handig voor een PC scherm van 1920 x 1080 pixels, en net bruikbaar op een 1600 x 900 laptop.



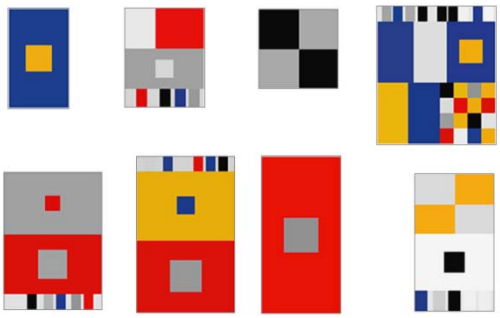

De genoemde keuze wordt gemaakt met *fuzzy logic*. De predicaten `large`, `medium`, `small` en `tiny` testen de grootte van een cel. Er zijn verschillende predicaten, één voor elke eigenschap. De tests leveren een waarheidswaarde, die niet traditioneel `false` of `true` is, maar een getal tussen 0 en 1. De waarheidswaarde is een soort kans, iets kan *een beetje* waar zijn. In traditionele logica is dat onzin maar in de door Zadeh uitgedachte fuzzy logic gaat dat prima. In de literatuur wordt zo'n waarde soms aangeduid als μ (membership). Een voorbeeld is te zien voor drie van de predicaten van de hoogte-breedte verhouding.




In feite zijn er vier: `hstretched`, `vstretched`, `squarish` (bij zeer ruwe benadering vierkant) en `square` (bijna zuiver vierkant). Tenslotte is er de test op nesting-diepte, `recursible`, die een vage grens trekt rondom 2. De nesting-diepte van het canvas is hierbij op 0 gesteld. Werkend met deze fuzzy-logic predicaten zien typische beslissingsregels er (ongeveer) als volgt uit:

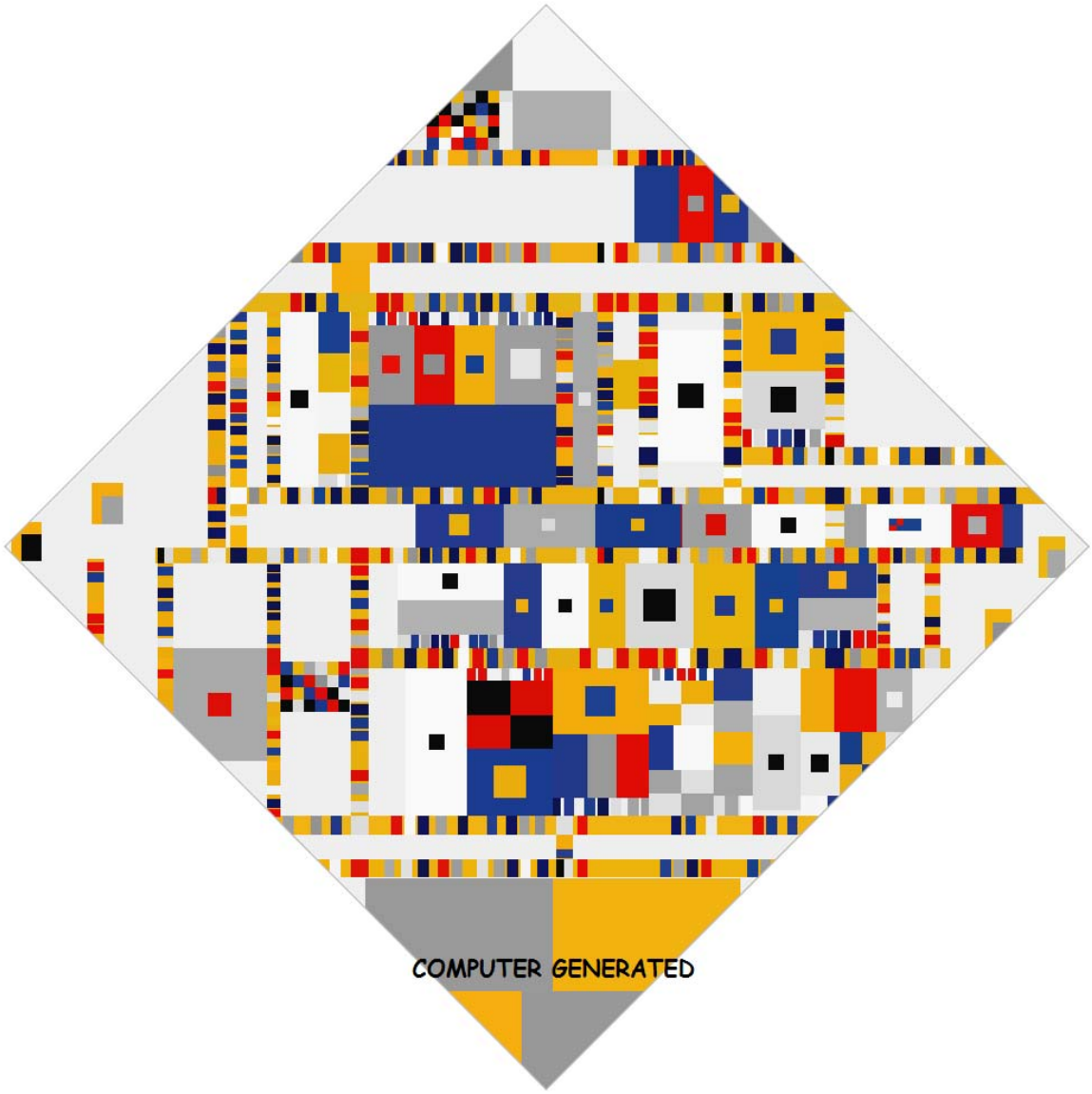
```
if (P(recursible()) && P(hstretched()) && P(tiny()))   setupH2();
if (P(recursible()) && P(hstretched()) && P(small()))  setupG();
if (P(recursible()) && P(hstretched()) && P(medium())) setupHN();
if (P(recursible()) && P(hstretched()) && P(large()))  setupHN();
if (                   P(vstretched()) && P(tiny()))   setupA();
if (                   P(vstretched()) && P(small()))  setupG();
```

Enkele voorbeelden van zo verkregen  mini's zijn hier te zien:



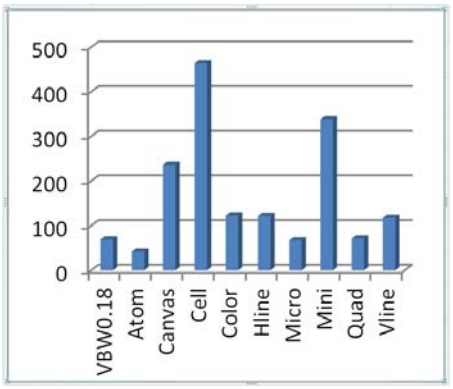
De regels hebben geen `else`, want het ene predicaat sluit het andere niet persé 100% uit. Een rechthoek kan een beetje horizontaal-uitgerekt zijn en tegelijk ook een beetje vierkant-achtig.  Op deze manier is er een twintigtal regels (rules), waarmee nog veel te experimenteren is. Ook de typologie is nog interessant om mee te spelen. Bijvoorbeeld K2 is nog niet geïmplementeerd en de Micro cellen zouden met de G variant  gecombineerd kunnen worden.

Een typisch resultaat van de code, één van miljoenen mogelijke outputs, is onderstaande afbeelding:



## Reflectie

Tenslotte enkele gegevens over de omvang van de code en een korte reflectie. Het geheel bestaat uit circa 1300 regels code. Het staafdiagram toont de regels code per klasse, inclusief commentaar. De klasse `Cell` is de grootste, en dat is goed: `Cell` bevat de technisch ingewikkelde generieke code van de groei-machine. Mijn op Blotkamp's destructietheorie geïnspireerde keuze voor unificatie betekent dat alles een cel is en dat de verschillen tussen celtypen minimaal zouden moeten zijn. Met andere woorden: eigenlijk hoort zoveel mogelijk van de code in `Cell` thuis. De klasse `Mini` is nogal groot, maar dat is verklaarbaar vanuit de centrale rol van recursiviteit. Idealiter zouden ook `Canvas`, `HLine`, `VLine`, `Micro` en `Quad` elk slechts enkele tientallen regels zijn.




Ik ben blij dat ik systematisch en ten volle gebruik heb kunnen maken van *object-orientatie*: inkapseling, vererving, en polymorfisme. De taal bood mij ondersteuning. De klassen zijn niet zomaar clusters van stukjes code, ze zijn vooral een model van wat er in de Victory Boogie Woogie aan de hand is, althans in mijn voorlopige interpretatie, namelijk *unificatie* van constructive elementen. Alle respect voor de ontwerpers van Java en Processing.

De random generator, soms ingebed in fuzzy logic, geeft vorm aan de idee dat er een grote ruimte van mogelijkheden is waar uit gekozen wordt, terwijl er wel degelijk structuur is en typologie.

Ook het principe van *celgroei* lijkt nog toepasbaar: in plaats van een topdown decompositie zoeken de lijnen en andere cellen zelf hun ligging en uitgebreidheid in interactie. Bijvoorbeeld de horizontale en verticale gele lijnen die op elkaar botsen, de atomen erbinnen die al dribbelend hun plaats vinden, de mini's die een brug vormen tussen de gele lijnen of soms een ruimte er tussen opvullen, en de quadrupels die naar buiten groeien terwijl hun cellen zich tegen elkaar afzetten. In de natuur komen vlakverdelingen middels celgroei trouwens ook voor, maar zonder de beperking tot horizontale en verticale lijnen ontstaan daar eerder Voronoi diagrammen (recent toegepast in het Drapely-o-lightment project met Marina Toeters van `by-wire.net`, Utrecht).

Tenslotte meen ik dat de *recursiviteit* op een aantal plaatsen in de code goed uitpakt. Omdat het dé nieuwe vinding is van Mondriaan in de Boogiewoogies heb ik die ten volle toegepast, bijvoorbeeld in de structuur van `Cell`, in de definitie van `grow`, en in de `setup` van `Mini`.

Waar ik minder tevreden over ben is hoe sommige mini's uitpakken. Soms gaan ze op in een nogal oninteressant horizontaal-verticaal decompositieschema. Ook de linker- en rechterhoeken met micro's zijn soms niet oké. Mondriaan zocht naar subtiele balans, dat kan de code niet. De meeste outputs zijn net iets drukker dan de Victory Boogie Woogie. Ik zie de code dan ook als on-af, work in progress.

Toch meen ik alvast te mogen concluderen dat het programmeren met celgroei en recursiviteit een manier is om naar de Victory Boogie Woogie te kijken, er vocabulaire voor te ontwikkelen en zo meer te zien, anders te begrijpen en nieuwe vragen te stellen. De code belichaamt de verworven inzichten.

Eindhoven, Februari 2013.