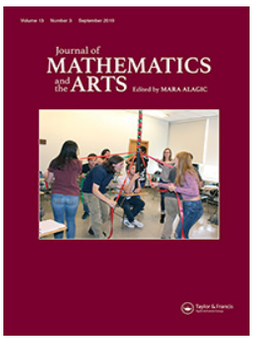
A program for Victory Boogie Woogie



Journal of Mathematics and the Arts

ISSN: 1751-3472 (Print) 1751-3480 (Online) Journal homepage: https://www.tandfonline.com/loi/tmaa20

A program for Victory Boogie Woogie

Loe M. G. Feijs

To cite this article: Loe M. G. Feijs (2019) A program for Victory Boogie Woogie, Journal of Mathematics and the Arts, 13:3, 261-285, DOI: 10.1080/17513472.2018.1555687

To link to this article: https://doi.org/10.1080/17513472.2018.1555687

Published online: 26 Mar 2019.

JOURNAL OF MATHEMATICS AND THE ARTS
2019, VOL. 13, NO. 3, 261–285
https://doi.org/10.1080/17513472.2018.1555687

Taylor & Francis
Taylor & Francis Group

A program for Victory Boogie Woogie

Loe M. G. Feijs

Department of Industrial Design, Eindhoven University of Technology, Eindhoven, Netherlands

ABSTRACT

The paintings of Piet Mondrian have been attractive for programmers and researchers writing programs that produce compositions resembling the originals. In earlier work published in Leonardo, a generic framework was proposed, which turned out flexible and could be adapted to generate many different Mondrian types. Until recently, there were hardly any attempts to target Mondrian's Victory Boogie Woogie, which is more difficult than most of the earlier Mondrian types. This article describes an extension of the Leonardo approach by including Victory Boogie Woogie. This work poses different challenges because of its uniqueness, its complexity and the fact that it is unfinished. In the extension, three principles for modelling and programming are used: (1) working with cells, (2) nesting, and (3) object-orientation. The program entered and won a Dutch national competition on programming Victory Boogie Woogie in 2013.

ARTICLE HISTORY
Received 24 February 2017
Accepted 24 November 2018

KEYWORDS
Mondrian; Victory Boogie Woogie; algorithmic art; generative art; composite pattern; De Stijl; object-oriented programming

1. Introduction

Piet Mondrian (1872–1944) was a Dutch painter who worked in The Netherlands, in Paris, in London and finally in New York. He is famous for his abstract and non-figurative paintings. He was born as Piet Mondriaan in The Netherlands in 1872 and was trained as a painter in Amsterdam and in a tradition known as the 'Haagse School'. His early works were mostly traditional landscapes and from 1908 there was an influence of symbolism and theosophy. From 1910 to 1920 the topics got restricted to windmills, coast lines, trees, church towers and flowers. While experimenting, Mondrian made the windmills, coast lines, trees and towers more abstract, first in a cubist tradition, later in his own unique style. Related experiments were done by other Dutch artists such as Theo Van Doesburg and Bart Van der Leck; together with other artists and designers they formed a movement called 'De Stijl'. Mondrian adopted a restricted set of colours, sometimes soft pastels, eventually only red, yellow and blue, next to black, white and sometimes gray. He moved to Paris and changed his name to Mondrian and in 1938 he moved to London. Between 1918 and 1940 he made many paintings of the kind for which he became famous, characterized by being non-figurative and having only horizontal and vertical black lines, and colour planes in primary colours. A few typical examples are shown in Figure 1.

CONTACT Loe M. G. Feijs l.m.g.feijs@tue.nl Department of Industrial Design, Eindhoven University of Technology, PO Box 513, Laplacebuilding 1.78 5600MB, Eindhoven, Netherlands

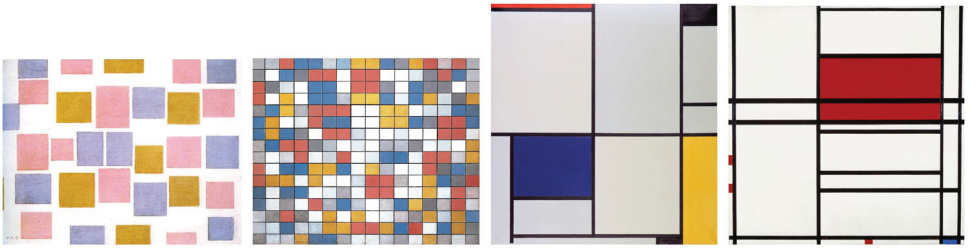


Figure 1. Typical works of Mondrian: Composition number 3 with colour planes (1917), Checkerboard with bright colours (1919), Tableau I (1921), and Composition No. 4 with red and blue (1938–1942).

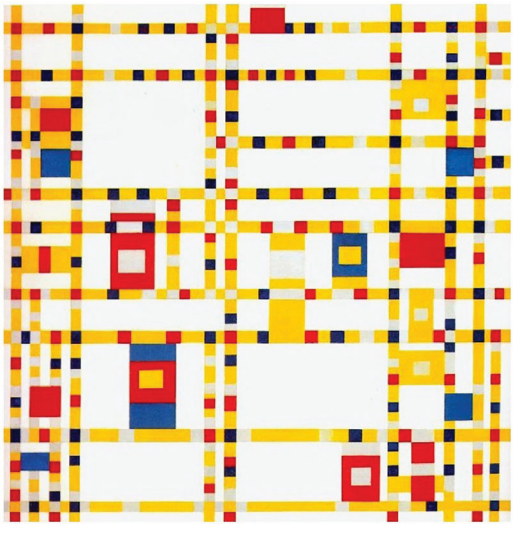


Figure 2. Piet Mondrian, Broadway Boogie Woogie, 127 cm × 127 cm, 1942–1943, current location Museum of Modern Art, New York (public domain).

From 1942 onward, after Mondrian arrived in New York and when he was already in his 70s, his work took yet another new direction: the compositions got coloured lines, new plastic tape was used for experimenting and the compositions became much more complex and lively, see Broadway Boogie Woogie (Figure 2). In an abstract way, Broadway Boogie Woogie and Victory Boogie Woogie (Figure 3) seem to represent the busy, lively and energetic atmosphere of New York. The last work, Victory Boogie Woogie, is fascinating and it gives rise to many questions, as it was unfinished when Mondrian died. More information on Mondrian can be found in several excellent books [3,15,23] or in the complete catalog of his work [16]. For more information on Victory Boogie Woogie we refer to the book by van Bommel *et al.* [4].

Mondrian, in his own writings, argued in various ways that art should always move on, searching for a pure beauty. The paintings do not depict anything from the real world. For example in 1937 he wrote a long article in Circle [20], from which we take a short quote:

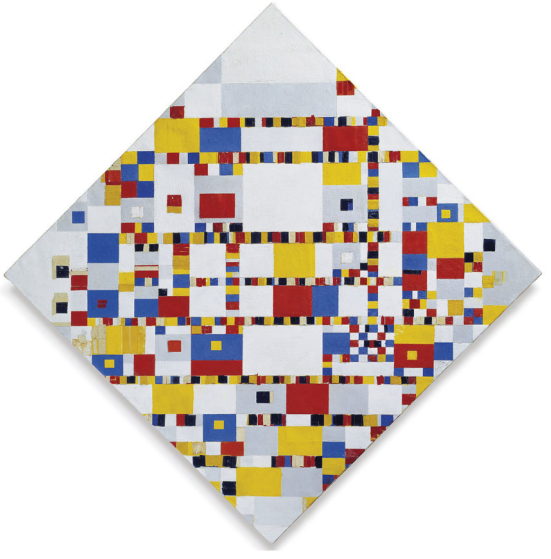


Figure 3. Piet Mondrian, Victory Boogie Woogie, 127 cm × 127 cm, 1944, current location: Gemeentemuseum The Hague (public domain).

> Precisely by its existence non-figurative art shows that 'art' continues always on its true road. It shows that 'art' is not the expression of the appearance of reality such as we see it, nor of the life which we live, but that it is the expression of true reality and true life ... indefinable but realizable through the plastic.

Although popular wisdom sees the grid of lines in Broadway Boogie Woogie (Figure 2) as a representation of the Manhattan street grid, it is unlikely that Mondrian would resort to such direct figuration. This also is the opinion of H. Janssen, co-author of the book on Victory Boogie Woogie [4], as he told me in 2013. Blotkamp [3] sees Mondrian's work as the systematic and sustained evolution towards abstraction. This direction means eliminating real-life interpretation and even eliminating options for object recognition and Gestalt effects. In this view, it is also unlikely that the lines in the Boogie Woogies represent streets.

There is a long tradition of synthesizing Mondrian-like compositions automatically, which was started in the 1960s by A. Michael Noll [18]. An overview is in Section 6. The work described in this article fits in that tradition. Programming Mondrian-like compositions is interesting to me, as I have great appreciation for both the aesthetics of Mondrian's work and the perseverance of Mondrian as a person. Moreover, I live in The Netherlands, Mondrian's first country, and know how to program. Combining these interests, I undertook a first curiosity-driven project in the period 1993–2003. The result was a single program that could produce different Mondrian composition types [7,8]. In that work I claimed that programming is a way to study Mondrian's work, to develop vocabulary, to see more, and understand it better. However, that program stopped before the Boogie Woogies, which are different and more complex, offering challenges that this method could not match.

The main topic of the present article is a new computer program, developed in 2013 and aimed at Victory Boogie Woogie (1944), Mondrian's last, unfinished work. This program is a tool for studying Victory Boogie Woogie and is also meant as a tribute and a

mark of honor to both Victory Boogie Woogie and Mondrian. The program builds on the experience and the principles developed in the project of 1993–2003, such as working with cells, see Section 2. It is coded in Processing [22] instead of the now outdated Turbo Pascal and is specific for Victory Boogie Woogie. Processing is an open source project by Casey Reas and Benjamin Fry. The language is essentially Java but packaged in an environment that makes the language accessible and attractive for creative work. The trigger for writing the new program was a *Call For Code* issued by Setup and Gemeentemuseum Den Haag in March 2013, communicated via www.setup.nl/events/2013/call-elegante-algoritmes. More information about the call for code is in Section 5. Whereas the focus of this article is on the technical aspects of the program, the project cannot be entirely separated from my personal narrative, where continuity with the earlier work [7,8] plays a role, next to enthusiasm for Mondrian, and a quest for elegance. The quest for elegance is motivated by my long-standing interest in code structure [10], name-space structure [6] and also by the demands of the *Call for Code* competition.

The process of conceiving, coding and evaluating the program described in this article began already with my previous project, and built upon many visits to museums and many books about Mondrian, such as the complete catalog [16]. The Pascal program of that previous project could synthesize many Mondrian composition types as produced by Mondrian in the years 1916–1942, after his figurative years and stopping just before the Boogie Woogies. The first step for this new work was to re-implement the program's essentials in a modern language and test it by generating the type of compositions Mondrian made in 1917. After that, over a sequence of iterations, I modified the program to generate compositions first in the style of Broadway Boogie Woogie, and finally in the style of Victory Boogie Woogie.

Sections 2–4 describe the development of the program. In Section 5 the work is evaluated and Section 6 is about related work.

2. Working with cells

When approaching Mondrian algorithmically, the first ideas that come to mind are top-down decompositions of the canvas. One such idea is to take a regular grid and perturb it. This idea finds support in works such as Lozenge with Gray Lines (1918), Composition: Light Colour Planes with Grey Lines (1919) and of course the checkerboards such as Composition with Grid 9: Checkerboard Composition with Bright Colours (1919), but after that Mondrian abandoned the grids. Blotkamp [3, p. 126] suggests that Mondrian responded to critique by Van Doesburg who had written that these works were without composition because all the rectangles had the same size. A. Michael Noll used a perturbed grid, but that was only for emulating a Composition with Lines (1917). Given my goal of reproducing Mondrian's later works, I knew that I also had to abandon regular grids (Figure 4).

A second idea is to recursively split the plane and its sub-planes. An example thus obtained is the leftmost image in Figure 5. However, this approach is not convenient to generate the typical lines which cross other lines before stopping, like the middle vertical line which stops at the lower horizontal line in the real Mondrian composition of Figure 5 (right).

The line arrangement needs more freedom than offered by recursive splitting. However, pure bottom-up approaches based on drawing random rectangles will not work either: if

Figure 4. Examples of Mondrian compositions based on of regular grids. Lozenge with Gray Lines, 1918 (left), Composition: Light Colour Planes with Grey Lines, 1919 (middle), Composition with Grid 9: Checkerboard Composition with Bright Colours, 1919 (right).

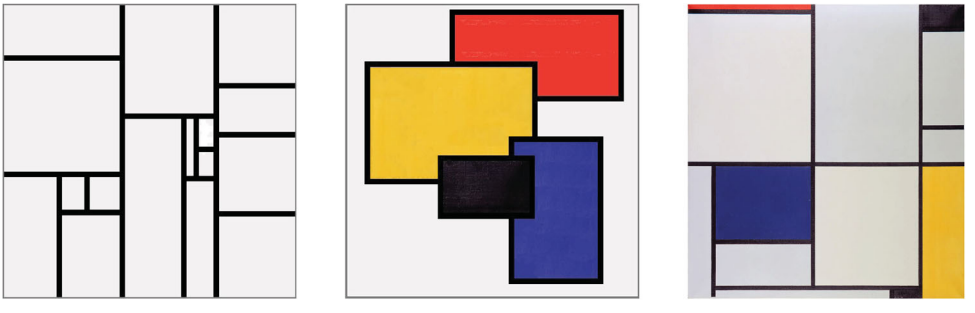


Figure 5. Two examples of obvious non-Mondrian compositions (left, centre) and a typical real Mondrian, Tableau I, 1921 (right).

there is too much freedom in the relative positioning of rectangles, they will overlap, as in Figure 5 (centre). Mondrian usually avoided such overlaps because they cause an undesired foreground-background Gestalt.

My previous approach [7,8] was to let planes and lines compete for space on the canvas and let them align with a process somewhat resembling cell growth. Blotkamp [3, p. 232] mentions about the painting New York (1941–1942) that there is 'a wide spectrum of forces that, pushing and pulling, attracting and repelling, succeed in keeping one another in balance and in their place'. It is clear that if the lines are ready, the rectangles between them could be flooded with colours. But I have decided not to treat planes and lines as different object types, but to work with a single unified type, which I call 'cell'. One main reason is that eventually Mondrian himself decided that the effect of lines should be absorbed by the coloured planes:

> Only now I become conscious that my work in black, white and little color planes has been merely 'drawing' in oil color. In drawing, the lines are the principal means of expressions . . . In painting, however, the lines are absorbed by the color planes; but the limitations of the planes show themselves as lines and conserve their great value. (Blotkamp's translation [3, p. 240])

This is what Mondrian wrote in May 1943, so I used the wisdom of hindsight. Already in paintings in the 1930s the difference between lines and planes was blurring.

Cells grow from initial seeds and interact with their neighbours to find their final positions. This is a generalization of the colour-flooding of rectangles. The analogy with cell growth in nature holds in certain aspects, such as the fact that my cells have a genetic profile determining their expansion. But in nature, the entire organism grows because its cells multiply and expand. For most compositions, the canvas never grows; instead the cells expand in empty space until they meet their neighbors. I have to admit that this was not exactly how Mondrian worked: in the early years he used 'doorbeelding' (see Section 3); later he shifted lines experimentally after a top-down design. My choice for working with a unified cell type uses the insight, also explained by Blotkamp [3], that gradually the difference between lines and planes would disappear. One of many examples discussed by Blotkamp is Composition with Yellow lines (1933) where it is impossible to decide whether the yellow lines are actually lines or planes [3, p. 229]. At the same time, working with cells is a practical framework for experimentation [7,8]. The algorithm operates in a manner that does not resemble Mondrian's way of working but is intended to produce compositions that do resemble Mondrian's.

Now the programming technicalities will be addressed. A *cell* is a two-dimensional coloured rectangle with some additional numerical and Boolean properties to regulate the cell's growth behaviour. Each cell is axis-aligned: its sides are horizontal and vertical. The cells grow from small kernels until they collide, and in this way, eventually, reach their final locations and extents. The growth occurs in steps. In one step, each cell attempts to expand, backtracking if it turns out that the expansion would cause any pair of cells to overlap. Cells may grow only horizontally (for example, cells that will become horizontal lines), only vertically, or both. Some cells grow until they reach the border of the canvas, others maintain a small distance ϵ. Some cells grow from time zero onwards, others begin only when a step counter reaches a particular value, its *activation* age. Ideally, as much knowledge about Mondrian as possible is coded as cell instance variables and methods, and as little as possible is coded in ad-hoc solutions for specific cell types or specific cells.

Initially, the locations of the cells are chosen uniformly at random within the canvas, though of course other probability distributions could be used instead, depending on Mondrian's composition type: for example for the 'Peripheral' type Mondrian made around 1922 the probability is high near the edges of the canvas and low in the centre [7]. Taylor [27] argues that Mondrian's lines are not random and finds a higher prevalence of lines close to the edge of the canvas. Every time the program makes its random choices, a different composition appears. Some of these results are 'better' than others, and assessing them is part of the process of continuous adaptation. The advantage of this model and the programs based on it is that one cell growth engine can be used to produce a wide variety of composition types. Next, more details of the code are given. The reader who wishes to skip the code can jump to Section 3.

We first explain how to produce two types of compositions, called 1917 Composition and 1938 Composition. The program has one important class called Cell. As explained by Bruce,

> Classes are extensible templates for creating objects, providing initial values for instance variables and the bodies for methods. All objects generated from the same class share the same methods, but contain separate copies of the instance variables. New objects can be created from a class by applying the **new** operator to the name of the class. [5, p. 18]

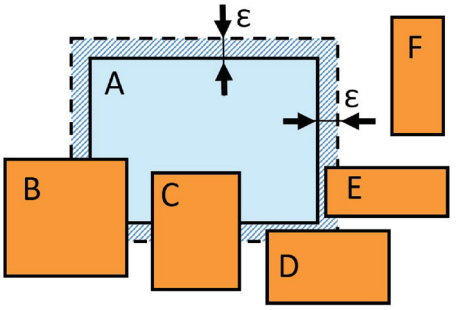


Figure 6. Possible configurations of cells. A^ϵ will denote A together with the extended area around it. In this configuration A overlaps with B and C, A^ϵ overlaps with B,C,D and E, whereas $A^\epsilon \cap F = \emptyset$.

```
class Cell
 1  int xMin, xMax, yMin, yMax
 2  colour clr
 3  boolean hor ← false
 4  boolean vert ← false
 5  boolean stops ← false
 6  float ratio ← −1
 7  int ϵ ← 0
 8  int age ← 0
 9  int activation ← 0
10  int midlifetrigger ← maxint
```

The cell has minimal and maximal x and y coordinates. Each cell has a colour and three booleans that regulate growth: if *hor* is true, the cell is allowed to grow horizontally, if *vert* is true, it is allowed to grow vertically. If *stops* is true, it will stop when bumping into another cell, otherwise it will cross-over or cross through (in the terminology of Andrzejewski *et al.* [1] a line that stops is a *non-spanning line*). Next, *ratio* is used to force the cell to have a certain height/width ratio but the special value -1 means that no ratio constraint applies. Then ϵ defines a band around the cell (see Figure 6). Finally *age*, *activation* and *midlifetrigger* control the precise timing of the growth, which is important when several types of cells compete.

The methods of Cell analyse the state of the cell, mostly in relation to its neighbours. Overlap is always checked with the extra distance band of ϵ. The most important methods of Cell are:

- hasOverlap() provides a collision detection mechanism that also detects *almost*-collisions. Let a and b be Cells, with associated rectangles A and B. Then a.hasOverlap(b,ϵ) is true if and only if A^ϵ intersects B. We extend this definition to allow a collection as cells as the argument, in which case the method determines whether any cell in the collection intersects A^ϵ.
- checkRatio() determines whether a cell is within some tolerance of its desired aspect ratio. Small cells are exempt from this test. The code listing is given below.

- GROW() is the heart of my program. The code listing is given below. In essence it is conditional backtracking for the expansion in each of the four directions. Only the horizontal case is included; the vertical case is similar and can be easily added by the reader.
- DRAW() simply draws the cell's rectangle using its designated fill colour.
- TRIGGER() is an empty method. Some cells perform special actions upon reaching their midlife, but the precise nature of this action will vary with the cell's type. Subclasses are therefore allowed to override TRIGGER.

We give code listings for CHECKRATIO() and GROW().

```
CELL.CHECKRATIO()
1  dx ← max(0.01, xMax − xMin)
2  dy ← yMax − yMin
3  return ratio < 0
4         or (dx < 10 and dy < 10)
5         or (0.8 × ratio) ≤ dy/dx ≤ (1.2 × ratio)
```

```
CELL.GROW(others)
 1  if age > activation
 2    then if hor
 3           then xMin ← xMin − δ
 4                fails ← this ⊈ canvas
 5                      or not CHECKRATIO()
 6                      or not HASOVERLAP(others, ϵ)
 7                if stops and fails
 8                  then xMin ← xMin + δ
 9                xMax ← xMax + δ
10                fails ← this ⊈ canvas
11                      or not CHECKRATIO()
12                      or not HASOVERLAP(others, ϵ)
13                if stops and fails
14                  then xMax ← xMax − δ
15         if vert
16           then yMin ← yMin − δ
17                // Continue as in the horizontal case
18  if age ++ == midlifetrigger
19    then TRIGGER()
```

This concludes the generic class CELL. The cells are not used in their pure form, but they are the base class which can be extended to have specific field values, additional fields, and additional methods.

The first example of such an extension is the class SQUARE. It is meant for generating compositions resembling the 1917 coloured rectangles compositions painted by Mondrian, such as the first painting of Figure 1.

SQUARE is based on specific colours and specific values for *hor*, *vert*, *stops*, *ratio* and ϵ. The idea is to make cells which are more or less square, in equal proportions in the colours, called PINK, GOLD and SKY. The class SQUARE extends CELL.

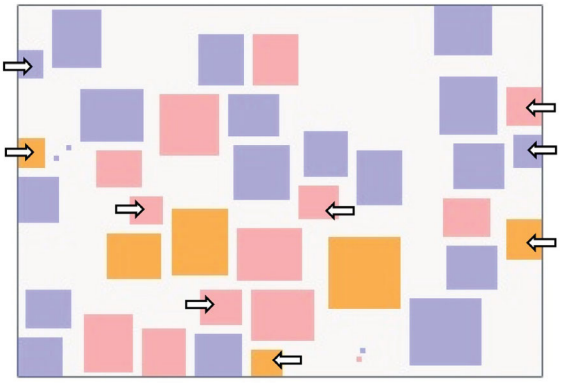


Figure 7. Intermediate composition, the snapshot being taken just before the point where all cells would execute TRIGGER(). There are four cells which did not develop at all, and hence will be pruned. Moreover, there are nine others which are also not large enough and which are to be pruned too (indicated by arrows).

```
class SQUARE(CELL)
 1  colour PINK ← colour (255, 175, 175)
 2  colour GOLD ← colour (255, 175, 75)
 3  colour SKY ← colour (170, 170, 255)

SQUARE()
 1  type ← "Square"
 2  setxy(positiononcanvas, uniform)
 3  clr ← CHOOSERANDOM(PINK, GOLD, SKY)
 4  hor ← true
 5  vert ← true
 6  stops ← true
 7  ratio ← 1.0
 8  ϵ ← 5
 9  midlifetrigger ← 20

SQUARE.TINY()
 1  return xMax − xMin < 20
 2         and yMax − yMin < 20

SQUARE.TRIGGER()
 1  if TINY()
 2    then cells.REMOVE(this)
```

The purpose of the TRIGGER() method, called at age *midlifetrigger*, is to eliminate cells that are too small. This is shown in Figure 7, an intermediate phase of a composition; the snapshot is taken when all cells have age *midlifetrigger* − 1.

Four cells in Figure 7 will never grow beyond their original 3×3 pixel size (they were conceived in a HASOVERLAP position). Another nine cells managed to develop somewhat,

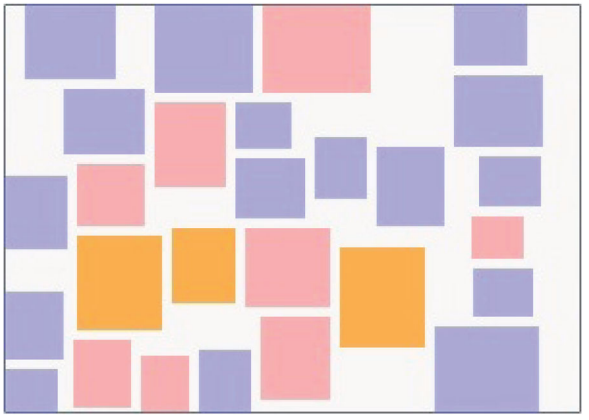


Figure 8. Generated composition emulating the visual style of Mondrian's compositions from ca. 1917.

but do not get large enough. They are considered tiny if they are less than 20 pixels in both dimensions. Once these are pruned, the other 27 cells continue to expand further into space which has come available, as can be seen by comparing to the final Figure 8.

The top-level program is given next. It initializes a global *cells* variable. In Processing [22], unlike C, there is no `main()` but there are two callback functions called SETUP(), which is done once, and DRAW(), which is called once per frame after SETUP(), at a rate of about 60 frames per second.

```
SETUP(1917)
1  size (297, 210)
2  cells ← new CELL[40]
3  for i ∈ cells
4     do cells[i] ← new SQUARE()

DRAW()
1  for i ∈ cells
2     do cells[i].GROW(cells)
3        cells[i].DRAW()
```

This completes the discussion of the generator for Figure 8, which could also generate an infinity of similar compositions. Note that the generic cell engine is coded in the class CELL and the specific details of the 1917 compositions are in the extension SQUARE.

It would be interesting to control the spatial distribution of colours, perhaps avoiding the effect in Figure 8 where the blues dominate the left and right sides of the composition, while the oranges run across the middle. It would be easiest to check the balance *after* the growth algorithm has done its work. If the balance could be checked algorithmically, one could search for a minimal colour-change intervention to restore balance. The work of Locher *et al.* [17] would be a good starting point, I leave it as a possibility for future research.

The same generic cell engine can also be used for other composition types. The composition of Figure 9 is obtained by deploying three derived classes of cells: HLINE, VLINE and

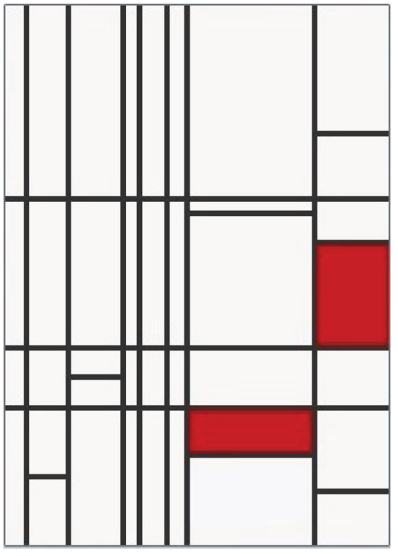


Figure 9. Generated composition emulating the visual style of Mondrian's compositions from ca. 1938.

Atom. HLine represents horizontal lines, which are black and have *hor* and *vert* set to **true** and **false**, respectively; VLine is defined analogously. Another difference between HLine and VLine is that vertical lines never stop, coded as *stops* ← **false**, whereas two-thirds of the horizontal lines will stop upon collision (*stops* ← **random**(3) < 2).

In essence, this is how 15 different Mondrian composition types were coded in the project of 1993–2003 [7,8]. The above program fragments are taken from a partial 2013 remake, simplified and polished for readability. This 2013 remake was taken as a basis for undertaking the challenge to program Victory Boogie Woogie.

3. Interpreting Victory Boogie Woogie

Before discussing the modelling and the coding, it is necessary to come to an interpretation of Victory Boogie Woogie. Rather than looking only at the features of Victory Boogie Woogie (for example lack of obvious overall structure, greater overall complexity, and the presence of many small rectangles), I try to understand the work as a next logical step in Mondrian's development. Blotkamp explains Mondrian's development in his book *Destruction as art* [3]. The term *destruction* should not be taken too literally: it does not mean to take away elements such as lines, planes or colours from a composition. It refers instead of Mondrian's desire to remove the figurative interpretation of the work. We should explain one hard-to-translate Dutch term first: *Doorbeelding* is the original Dutch term used by artists of the De Stijl movement for their specific process of continued abstraction and geometrization. The term means, roughly 'continued imaging' or perhaps 'shaping-through'. Mondrian did not merely pursue doorbeelding to the point of abstraction, beyond which there is really no tree, church, windmill, or coastline to be seen; after circa 1920, the goal was to limit the recognition of objects such as individual rectangles or lines. In short, Mondrian did not want any Gestalt. *Gestalt* is a German word, essential in modern perception theory [2] but hard to translate. The words 'form' or 'shape' come close.

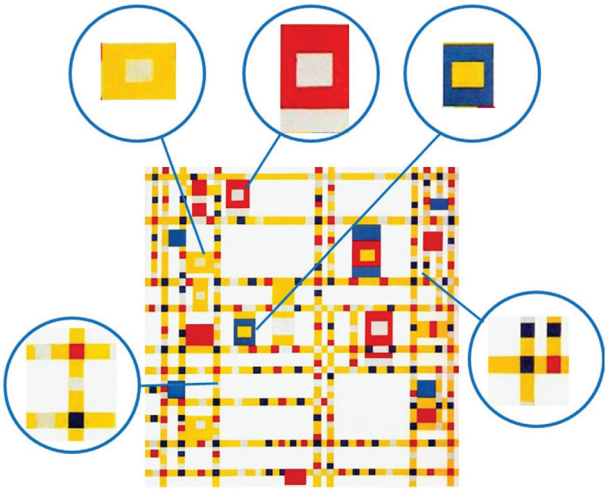


Figure 10. Mini-canvases and small rhythmic/intersection rectangles in Broadway Boogie Woogie.

It means the whole nature of something, as perceived. The distinction between foreground and background is also a Gestalt effect. The false observation that coloured patches are floating in front of a background is avoided by Mondrian's introduction of black lines. The object-nature of rectangles is weakened by solutions such as having the rectangles lie partly on the edge of the canvas. It is also weakened by ambiguity in adjacent same-coloured rectangles. The object-nature of lines is weakened by ambiguities such as doubled lines. The difference between lines and rectangles disappears at times almost entirely by their unification into coloured lines in the late 1930s. The invention of the Boogie Woogies involves one final unification step, in which the difference between the *canvas* and the *objects on the canvas* disappears: they belong to the same typology of objects now, witnessed by the mini canvases appearing inside (see Figure 10).

In the Boogie Woogies, nesting appears for the first time: inside the canvas there are mini-canvasses, which are indicated by the circled regions in Figure 10. A 'mini' is a mini-canvas, a small painting in the great work. Therefore I adopt nesting as a key theme in this project. In object-oriented programming, this nesting corresponds to the well-known Composite pattern. As described by Gamma *et al.* 'Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly' [11]. It can also be viewed as a limited type of recursion in the data structures, elegantly reflected by the way functions call themselves to process the objects.

Let us adopt the hypothesis that a notion 'Boogie Woogie type' does exist and perhaps even a notion 'Victory Boogie Woogie type'. In Mondrian's previous works such types are well recognizable [7,8], but the new hypothesis is riskier: there are only two Boogie Woogies in total and there is only *one* Victory Boogie Woogie, which is very complex and unfinished (there are even small pieces of tape on it). It is literally 'work in progress', as Jaffé [15] puts it. We suffer from ignorance regarding the generalized Victory Boogie Woogie type, unlike Mondrian's earlier work, for which one can have a good idea of the 'colored rectangles type' of 1917 or the 'ladder type' of the late 1930s. Furthermore, we can only speculate on

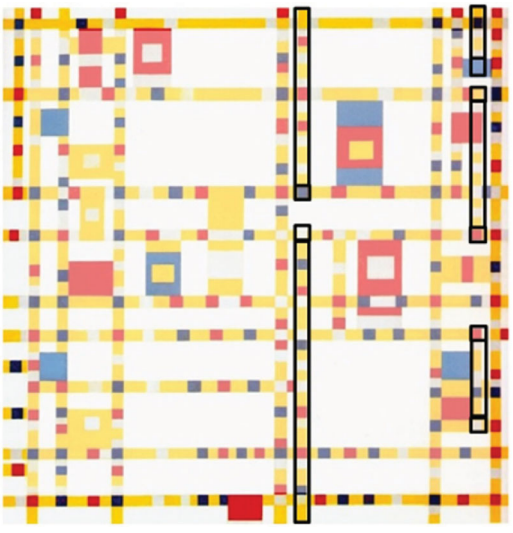


Figure 11. Twin and triplet verticals in Broadway Boogie Woogie.

Mondrian's vision for the completed work. There is research on how the work would appear without the pieces of tape [4, Chapter 6], which I studied before the present project. And there is information about the process of Mondrian at work on Victory Boogie Woogie: we know that he experimented and changed the painting over and over again.

There is also inspiration to be found in earlier work by Mondrian. The first task was programming the structure of the yellow line grid, for example, so that Broadway Boogie Woogie-like compositions could be made. The program for Victory Boogie Woogie creates a grid that is Broadway-inspired. It is clear that Mondrian explored certain innovations in Broadway Boogie Woogie which he re-used later in Victory Boogie Woogie. This is the case for the yellow lines with the small rectangles thereon, often praised as 'rhythmic planes'. The yellow lines are not new: they are already in the Lozenge Composition with Yellow Lines(1933), but the rhythmic rectangles are new indeed. Also, the invention of the mini-canvases is new; they are positioned as bridges connecting the yellow lines.

Another interesting detail is given by the small rectangles, each of which is positioned on the intersection of a horizontal and a vertical line. In Victory Boogie Woogie there are similar rectangles, but the situation is not very clear, and therefore it was decided to stay closer to the Broadway inspiration. Even if a vertical crosses a horizontal and is stopped there, then there is an intersection between the hypothetically extended vertical line and the horizontal line. Sometimes there are stray intersection rectangles at quite a distance. Figure 10 shows Broadway Boogie Woogie with its yellow line grid and the small rhythmic rectangles and intersection rectangles. Three of the mini-canvases and some of the small rhythmic/intersection rectangles are highlighted.

Another Broadway inspiration demands attention: the twin and triplet verticals. Several interpretations are possible: for example that a twin line once was one single line which is now interrupted, or alternatively, that the two lines simply happen to be aligned with respect to each other, see Figure 11. Later in the design and implementation, the second option was used.

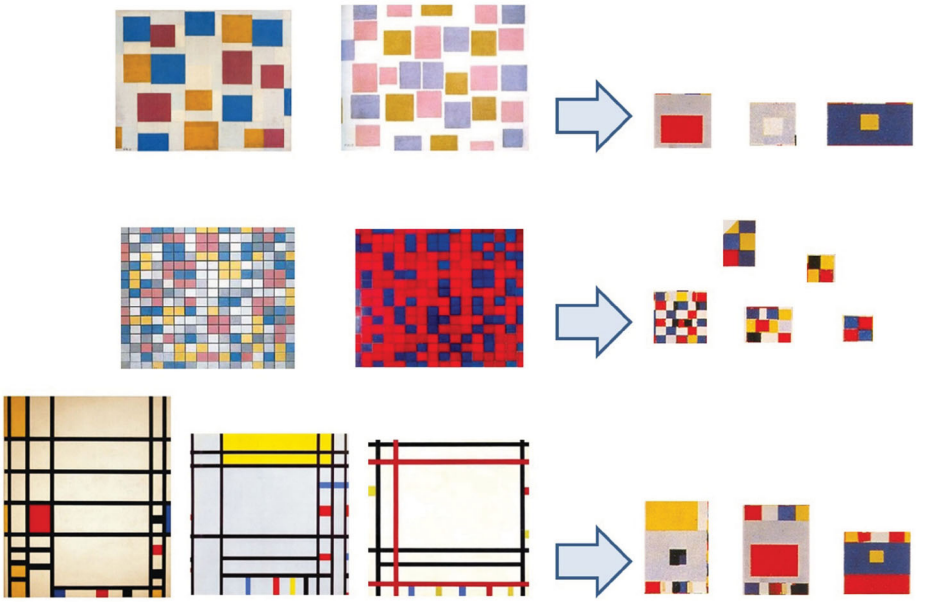


Figure 12. Victory Boogie Woogie as a retrospective: elements in Victory Boogie Woogie interpreted as a revival of earlier compositions.

In Victory Boogie Woogie, one can see not only many more mini-canvases, 'minis' for short, but also more *varied* types of minis, compared to Broadway Boogie Woogie. Perhaps those minis are inspired by earlier types of Mondrian's compositions, such as the 1917 'coloured rectangles' type, the 'checkerboard' type of 1918 and 1919, and the 'square' type, by which I mean Place de la Concorde 1938–1943, Trafalgar Square in 1939 and New York 1941. The background colour of the horizontal lines with rhythmic rectangles *inside* the minis is more white than yellow. These three types are depicted in the three rows of Figure 12.

The minis can be interpreted as a retrospective of these types from 1917 to 1941. They can be seen as a revival of some solutions that were problematic in the early days and thus were discontinued, at least in their pure form. There exist only two 'checkerboard' compositions, for example. It is an attractive idea that Mondrian, at last, found a home for them in Victory Boogie Woogie.

4. Modelling and coding Boogie Woogies

The framework of object-oriented modelling and programming is suitable to sort out and organize the properties, differences and similarities between Victory Boogie Woogie elements. It is also suitable to construct specific elements, to analyse them, to clean them up if necessary, and to display them. One unifying object type is chosen again: Cell. This choice is combined with a further exploration of the cell model deployed in the project of 1993–2003 [7,8] and described in detail in Section 2.

The main innovation is nesting. For Mondrian that began with Broadway Boogie Woogie in 1943, notably with the invention of the mini-canvases. In the new model, each cell

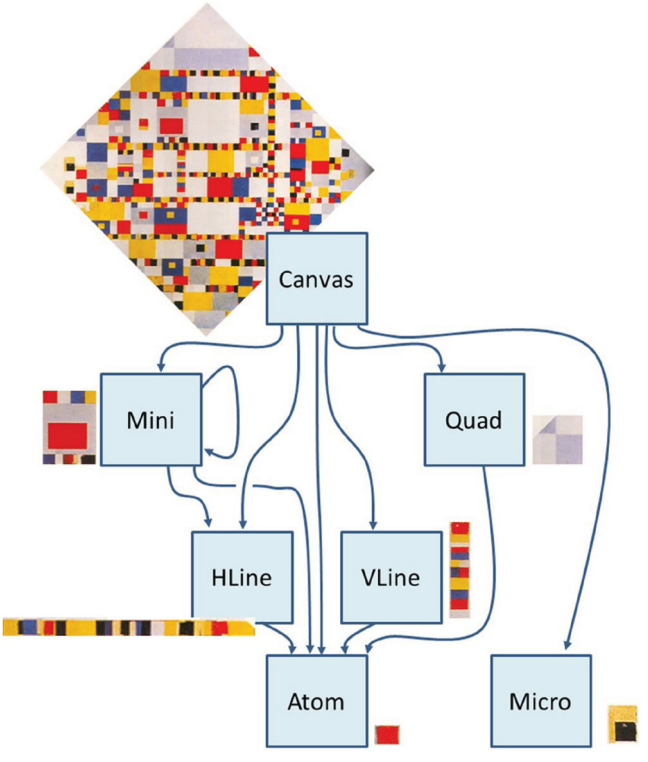


Figure 13. Cell containment.

has a 'parent', which is either the canvas or the larger cell in which this cell is contained. Each cell also has a special attribute consisting of a *set of* cells, each of which again has a colour, a location and several growth characteristics. And possibly some of those cells again have a set of cells themselves. Important operations on a cell are programmed in a recursive manner, including growth, overlap detection, drawing, and counting the number of cells. The canvas itself is one large cell. The Composite pattern [11] is both practical and theoretically appropriate for the model.

The set of all cells is represented through a Cell base class. Specialized cell types can then be represented as subclasses of Cell, yielding the classes Canvas, HLine (horizontal line), VLine (vertical line), Mini, Atom, Micro, and Quad. As shown in Figure 13, many of these cell types are composites: their instances may contain other cells recursively. Even a Mini may contain another smaller Mini. Atom is the base of this hierarchy: an Atom has only a colour and position.

For correct handling of the Broadway-style concept of intersection rectangles, which occur on the intersection of an HLine and a VLine, certain cells are equipped with *two* parents: a parent and a 'co-parent'.

Two additional classes are named above: Micro and Quad. A Micro is a small cell which has one subcell. Two Micros can be seen in the leftmost corner of Victory Boogie Woogie, Figure 3 (yellowish white colour, black subcell). In my implementation, the subcell, which is either black or gray, occupies two-thirds of the Micro in both x and y-direction.

A Quad (short for quadruple) is a configuration of four white or gray rectangles sharing one corner. There is a Quad in the upper corner of Victory Boogie Woogie, Figure 3. I also deploy a slightly misaligned Quad, using yellow, to fill the lower corner of the canvas. The implementations of MICRO and QUAD are, admittedly, a bit ad-hoc.

Extensive use is made of the well-known generate-and-test approach: as a very simple example, the whole square area, including the parts outside of the lozenge that makes up Victory Boogie Woogie's tilted canvas, is sprayed with line kernels and mini kernels and then those that happen to be outside of the lozenge, or are positioned in a corner to be kept empty, are removed again. After experimentation, I believe this is the best compromise regarding readability.

As discussed in Section 2, every Cell has an age when it reaches midlife, at which point a polymorphic TRIGGER method is invoked. Different Cell subclasses react differently: for example the clean-up operation within an HLINE or VLINE cell, must wait until the line has sufficiently matured by itself. After that, indeed, those inner atoms, the 'rhythmic rectangles', that have failed to become large enough, are eliminated. Similarly, a mini should not begin its recursive development unless the mini itself is more or less mature.

Finally a few technical points of interest. The rhythmic rectangles are implemented as Atoms that are contained within the HLINE and VLINE cells. They have an $\epsilon > 0$ so that, while growing, they keep some minimal distance to their neighbors, implemented by the ϵ mechanism discussed before.

The nesting is implemented through the SETUP function of MINI, which is mediated by the TRIGGER mechanism. The nesting of the real minis in Mondrian's Victory Boogie

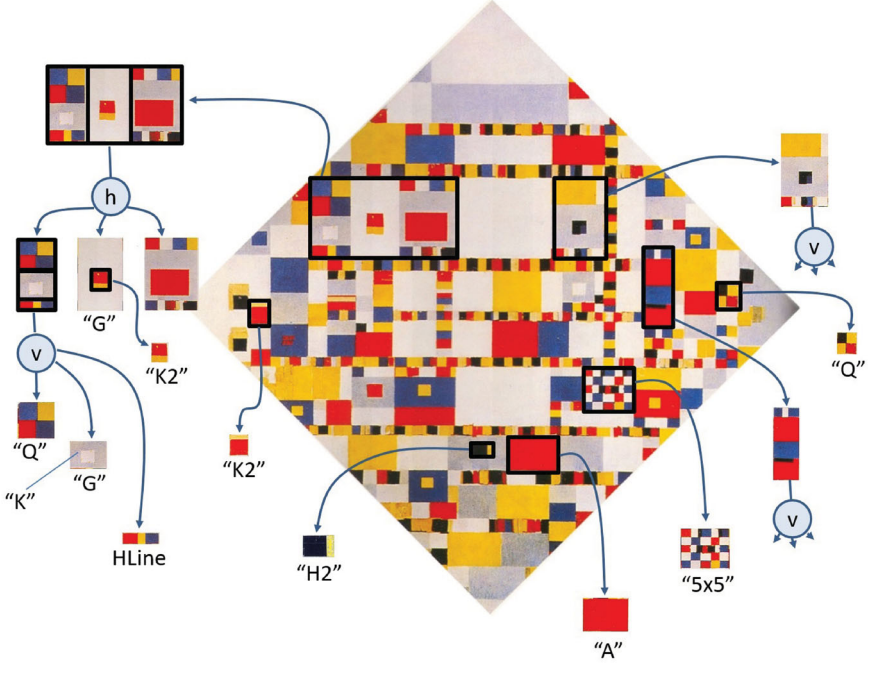


Figure 14. Nesting of the real minis in Mondrian's Victory Boogie Woogie. The decomposition operations are labelled 'h' for horizontal and 'v' for vertical decomposition.

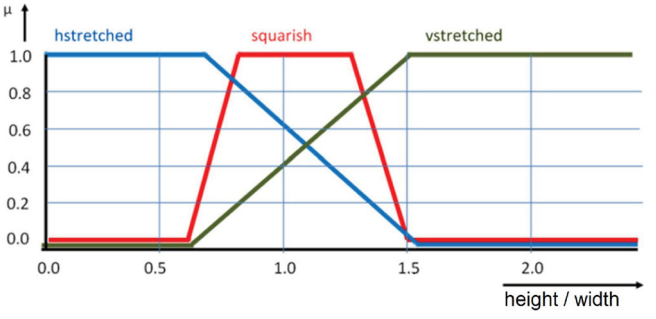


Figure 15. Fuzzy logic predicates related to aspect ratio.

Woogie was studied first. Although all kinds of ambiguities arise, the analysis of Figure 14 provided guidance. The decision of how to further decompose a given Mini is taken on the basis of three data: its size, its aspect ratio, and its nesting depth. Future work could probably take additional contextual information into account.

These splitting decisions are made based on fuzzy logic [28]. A Cell's size is broken down into four fuzzy predicates: Large, Medium, Small, and Tiny. Each predicate returns a continuous truth value between 0 (false) and 1 (true). Aspect ratio is broken into predicates Hstr (horizontally stretched), Vstr (vertically stretched) and Sqr (squarish); the graphs for these fuzzy predicates are visualized in Figure 15. Nesting depth is expressed via a Rec (recursible) predicate, which transitions from true to false in the neighbourhood of depth 2. Any continuous truth value a may be used to perform an action with probability a via a function P(a), which simply checks whether a number chosen uniformly at random from [0, 1] is less than a.

Working with these fuzzy logic predicates, the splitting process for a Mini might begin as follows:

```
Mini.setup()
  1  if P(rec()) and P(hstr()) and P(tiny())
  2    then setupH2()
  3  if P(rec()) and P(hstr()) and P(small())
  4    then setupG()
  5  if P(rec()) and P(hstr()) and P(medium())
  6    then setupHN()
  7  if P(rec()) and P(hstr()) and P(large())
  8    then setupHN()
  9  if P(vstr()) and P(tiny())
 10    then setupA()
 11  if P(vstr()) and P(small())
 12    then setupG()
 13  / / More cases
```

Here, Mini.setupH2 gives a horizontally oriented composite mini with two atoms. Mini.setupHN gives a horizontally oriented composite mini, based on the (h) split in Figure 14. It has N sub-minis, where N is either 3, 4, or 5, depending on the cell's aspect

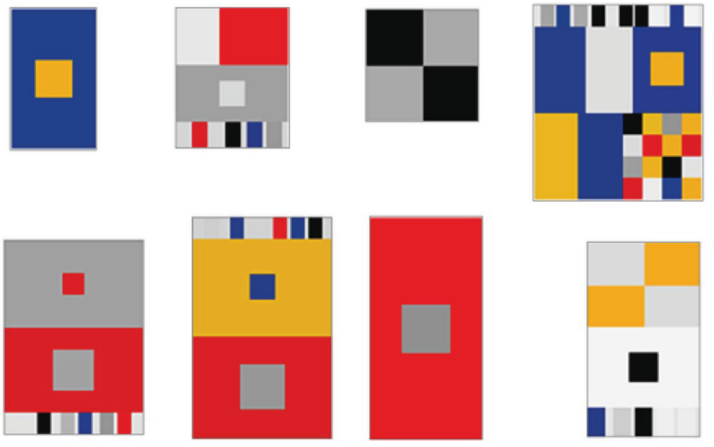


Figure 16. Minis constructed by nesting based on fuzzy-logic decision rules.

ratio. For setupG, the goose-eye mini, and setupA, the atom mini, see the insets labelled 'G' and 'A' in Figure 14. Note that the options in the code above are not mutually exclusive: multiple splits may be applied to the same Mini. Figure 16 shows some of the minis thus obtained.

There are 21 rules, leaving room for lots of experimentation. Also, the mini typology itself is still an interesting playground. The above Mini.setup deploys auxiliary functions coding the lower levels of fuzzy decision making.

A few typical examples of the millions of possible output of the program are presented in Figure 19. All code is written in version 2.0 of Processing [22]. The whole program is 1366 lines of code and available online at github.com/Elegant-Setup/LMG-Feijs.

5. Evaluation

Before showing the final Victory Boogie Woogies I give an impression of my iterative process. How did I know when I was on the right track? How did I know what to adjust? To address these questions, three outputs of intermediate versions of the Broadway Boogie Woogie emulator are presented in Figure 17. For most such images, the deficiencies are clear. Certain aspects are so wrong that their repair takes priority and in this way the process is self-propelling.

In BBW4 in Figure 17 (left) it is obvious what is wrong: (1) the mini-canvasses are not implemented yet, and (2) some of the intersection planes are not well-aligned with one or both co-parents. In BBW8 in Figure 17 (centre) other problems are dominant: (1) the big blue mini-canvas is too large and the way it occupies most of four rectangles is ugly and not Mondrian-like, (2) the yellow line which ends by bumping into the red mini-canvas is not Mondrian-like, and (3) the goose-eyes are not centred. In BBW10 there are several acceptable mini-canvasses, but Mondrian would have been unlikely to paint the two big minis in the same rectangle.

There are several lessons I learned in this process. First, the naive enthusiasm about how great and how Mondrian-like successive results will be is a good motivator to get started and to keep on going. Secondly, during the process, in which I mixed coding with

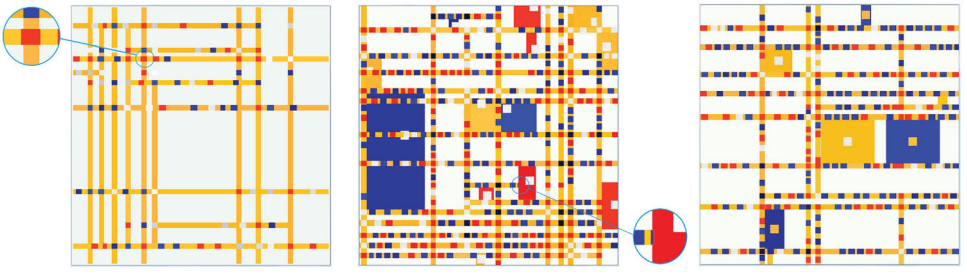


Figure 17. Three examples of outputs of experimental Broadway Boogie Woogie programs (intermediate versions BBW4, BBW8, and BBW10).

studying resources on Mondrian, the understanding of Mondrian and intuition of what is and what isn't Mondrian-like developed as well. Paradoxically, this combination of study and implementation brought the limitations of my work into sharper focus, causing the goal of a perfect Mondrian reproduction to recede even as I pursued it. It is an active means of studying Mondrian.

Turning to Victory Boogie Woogie, Figure 19 at the end of the article shows three generated compositions, so the reader can judge whether he or she considers them Victory Boogie Woogie-like.

Next a number of technical and elegance-of-coding aspects can be considered. This part of the evaluation is personal, also in the sense that my personal ambition is used as an evaluation criterion. The first ambition was to see whether the principles of working with the cells of the project of 1993–2003 [7,8] could be extended. The second ambition was to write elegant code where the unification principles (lines, planes, minis, canvas → cells) would matter for the code.

Regarding the first ambition, I am moderately positive. The principles of working with the cells, first explored in the project of 1993–2003 [7,8] seem to be still applicable: instead of having a top-down decomposition imposed upon everything, the lines and the other cells interactively guide each other towards their locations and their extents. It is also an interesting experience to see this in real-time. For example the horizontal and vertical yellow lines collide and stop each other, the rhythmic rectangles contained in them dribble to find their place, the minis flood into empty space and form bridges between the yellow lines, and the quadruples grow outwards while pushing against each other. On the self-critical side: the way some of the minis appear is not satisfactory yet. Sometimes they go on in a rather uninteresting horizontal-vertical decomposition scheme. Also, the left and right corners of the composition, with cells called 'micros' are not always acceptable. Mondrian was always looking for subtle balance, as discussed by Overbeeke and Locher [17], but the code embodies no knowledge about balance. The lack is compensated by generating a few more lines and cells so the output as a whole is 'busier' than Victory Boogie Woogie. As a final critical remark, in the modelling and the coding of what happens in the corners, *quadruples* and *micros* are still too ad-hoc.

Regarding the second ambition, I am positive in the sense that I applied a number of widely advocated principles: object-orientation was used systematically throughout the entire program, exploiting encapsulation, inheritance, and polymorphism. The Processing language offers good support for that. The classes are not just clusters of code fragments,

they are mainly a model of what is going on in Victory Boogie Woogie, at least in the interpretation of Section 3, namely unification of constructive elements. Also, several of the patterns described by Gamma, Helm, Johnson, and Vlissides [11] can be recognized in the code such as the *Composite* pattern, i.e. cells in cells, and the *Flyweight* pattern, i.e. colours are coded as objects with a string label, but they are shared to prevent each pixel having its own colour object. Next there is the *Template* pattern: for example, GROW is an abstract algorithm whose details are provided elsewhere and are activated in an age-dependent manner. Finally, there is the *State* pattern as state components HOR, VERT and STOPS determine behaviour. The random generator, partially embedded in fuzzy logic, implements the idea that there is a large space of possibilities, whereas at the same time there is a lot of structure and typology.

Programming with cells and nesting is an elegant way to look at Victory Boogie Woogie, to develop vocabulary, to look better, to see more, and to ask new questions. The vocabulary includes a number of words, the most important being: mini, micro, quad, atom, epsilon, parent, co-parent, twin and stops. In the context of the full program, they have a very precise meaning. There are many more variable names and function names that have a very precise meaning too, but it is hard to explain them in this article without adding more technicalities. The reader is refered to the full code and its comment on github.com/Elegant-Setup/LMG-Feijs. The code embodies the insights gained.

I consider the code as unfinished, a work in progress, which is consistent with the fact that Mondrian was always trying to improve on his paintings, and with the vision that coding is the twenty-first century craft.

As a third type of evaluation, we describe the *Call for Code* competition organized in early 2013 by Setup and Gemeentemuseum Den Haag. Setup is a media laboratory in Utrecht (The Netherlands). Gemeentemuseum The Hague is an art museum in The Hague, in The Netherlands, and it holds the world's largest Mondrian collection, including Victory Boogie Woogie. The challenge was 'to re-create Victory Boogie Woogie on a screen'. The inclusion criterion was 'being able to create rectangles on a screen' and no further restrictions applied. International applicants were allowed and many entered. From the call:

> It is a kind of demoscene competition where beauty comes first. The work doesn't need to be a 1-to-1 reproduction of the painting. The jury will look for quality and creativity of the underlying algorithms. The beauty of the code comes first, not the least number of signs, although that will be considered too.

The quote is translated from Dutch from www.setup.nl/events/2013/elegante-algoritmes. From the title of the site it was clear that elegance would be important and there was a vision that programming is the craft of the twenty-first century.

The jury members were Ionica Smeets, freelance science writer and since 2015 also Professor of Science Communication at Leiden University, Eric van Tuijn, representing Gemeentemuseum The Hague, Jan Baptist Bedaux Jeroen Clausman and Arthur Veen, being the three members of Compos 68, a group pioneering computer art as early as 1968–1969 (http://monoskop.org/Compos_68). The program entered the *Call For Code* competition on 1 March 2013. Besides the code and a `README` file I submitted a full paper in Dutch describing the approach. The present article is partially based on this prior paper, written in Dutch [9].

Figure 18. Event at Gemeentemuseum The Hague.

There were 32 other submissions, developed in Matlab, Processing, Javascript, Python, Java, Common Lisp, Shell, Perl, Microsoft tag, R, scratch, C#, and PHP. There were roughly speaking three approaches: (1) algorithmic approaches based on a model of the structure inside Victory Boogie Woogie, (2) data compression approaches mimicking Victory Boogie Woogie in a compact code scheme, (3) artistic and humorous approaches not trying to come close to Victory Boogie Woogie in a literal way, but for example, embedding a portrait of Mondrian in a lozenge-based fashion, or letting a structured and coloured representation of the code itself appear as a composition. Three selected candidates were invited to appear on Dutch National Television in the late-night show of Pauw and Witteman in Season 7, Episode 13, where Ionica Smeets explained the notion of *algorithm*.

The next day the program described in this article won the competition, the prizes being an honorable mention and appearing of the work in the Gemeentemuseum The Hague, and a large printed canvas. Figure 18 is a photograph taken at the event. The entire competition was an explosion of creativity and a tribute to Mondrian. All details can be found on Github: github.com/Elegant-Setup/.

6. Related work

Mondrian's work has been the subject of many past computer programs; I do not aim at completeness in this section. Several interesting projects are discussed.

The pioneering work of A. Michael Noll should be mentioned first. Noll [18] let the computer generate black and white drawings resembling the 1917 'Composition With Lines'. The degree of randomness could be varied. In psychological experiments with 100 test subjects, Noll claimed that 'only 28 percent of the subjects were able to identify correctly the computer-generated picture, while 59 percent of them preferred the computer-generated picture'. I did not undertake such experiments and my claims are different: that programming is a tool that forces one to have a different and closer look to the real Mondrians, and to develop vocabulary.

Hiroshi Kawano was a pioneer too; he created for example Work no.7. Artificial Mondrian, 1966 (see dada.compart-bremen.de/item/agent/234).

Figure 19. Three Victory Boogie Woogie-like composition generated by the computer program.

De Silva Garza and Lores [24] describe a program to generate Mondrian compositions using an evolutionary algorithm. The program is based on a number of rules to check whether a composition satisfies rules about colours, number of horizontal and vertical lines, and adjacency of certain types of coloured fields. The works generated seem to fit in Mondrian's 1935–1940 period, although the authors are not very specific about the fact that Mondrian's own work evolved significantly over the years. The work of De Silva Garza and Lores differs from the present project in two ways: first, I focus specifically on Victory Boogie Woogie, not the earlier 1935–1940 works. Second, I tried to build various principles into the construction process, not leaving all evaluation to survival rules, although the fuzzy decision process for the nested minis is closer to their approach. De Silva Garza and Lores also use a neural network for the evaluation [25], and therefore need training data. It is a disadvantage of their approach that one needs to choose carefully the parameters of the

evolutionary process. In the present project, there is no need not worry about population management.

Van Hemert and Eiben [13] describe an interactive evolutionary generator. The user has to grade some individuals before the next generation is produced. There is a chromosome for colour and a chromosome for composition. Van Hemert and Eiben also use recursion but they use it to split the main area into sub-areas (which is different from my way of making sub-rectangles, where the process of cell alignment is started inside an existing rectangle). In fact, some of my early attempts in 1990 were based on a similar genetic idea: the values of $x, y, hor, vert, stops, ratio, \epsilon$ and *age* would form a composition's DNA, next to the number of cells for various cell-types. But neither the aesthetics nor the Mondrian-resemblance would converge easily and the approach was rejected as the human, in this case me, had to emulate the ecological niche over and over again.

Similarly, Andrzejewski *et al.* [1] use feature vectors to train classifiers and generators. They use real Mondrians as training data, mostly the 1930–1940 compositions. No attempt is made to analyse or generate Boogie Woogies: 'We did not include his few works that contained diagonal lines or were on diamond canvases'.

Zhang *et al.* [29] argue that digital art is more expressive than traditional visual art and creates opportunities for computational aesthetics, which is quite distinct from my claims which are that programming is a tool for better understanding and for developing vocabulary. Their blueprint for generative art is to have an arrangement of primitive shapes, which forms an initial aesthetics, and then disrupt it with randomized perturbations. The present article's program does not follow this blueprint in the sense that the initial cell seeds hardly have an initial aesthetics, but the aesthetics emerge after the collisions. But for the filling of open areas between the yellow lines, my program does seem to follow their blueprint, if we take the gradual and recursive refinement of the minis as a perturbation mechanism. Zhang *et al.* [29] also define three levels of complexity: full human participation is level 1, no human participation is level 2 or 3. There is a fourth level of 'minimal' human participation. My program for Victory Boogie Woogie should be classified as level 3: the means of computer support is 'knowledge-based heuristic rules or patterns encoding given styles or domain knowledge'.

Mirjam Haring [12] describes a Malevich generator which is influenced by my own projects of 1993–2003 [7,8]. Haring ran an experiment to test whether people could distinguish between computer generated and real Malevich compositions, which usually they could. It turned out however that:

> people with higher knowledge of art are worse in identifying Malevich's paintings. But this could be caused by the small number of people who rated their art knowledge as 5, which is also illustrated by the larger standard deviation. Another explanation for these results could be that the people with a lot of knowledge of art, who were all in the age group above 45, have less knowledge of computers, therefore being less able to spot the computer characteristics.

There are many online interactive Mondrian generators, for example 'Mondrimat' by Stephen Linhart (www.stephen.com/mondrimat/), based on repeated splitting of the area and filling colours, and Automondrian by Bruce Ediger (http://www.stratigery.com/automondrian.php) which lets the user choose a number of parameters such as density, whiteness, splittyness, buffer size, max area fill and max aspect ratio to fill.

To the best of my knowledge, there have been very few attempts to generate Victory Boogie Woogie by computer before 2013. John F. Simon Jr. created ComplexCity in 2000,

inspired by Broadway Boogie Woogie, although not very literal. ComplexCity is dynamic [26]: 'everything moves, controlled by a computer program: cars drive, stop for red light, elevators move up and down'.

Of the results of the *Call for Code*, programs by Kiri Nichol and Edwin Jacobs were also nominated in the top 3 by the jury. Kiri Nichol's generator [21], called Mondify, makes a fine-grained grid first, using irregular distances. There are rules restricting adjacency of colours, e.g. white and ivory. Then two coarser grids are added, which later will give rise to the larger blocks and also are used for the 'streets'. The results are certainly Victory Boogie Woogie-like.

Edwin Jacobs' generator [14], called MVBW−1, inserts rectangles from a table of templates, which is obtained by an earlier process of manual adjustment. There is a tree structure to maintain a hierarchical organization of the canvas, which is refined by splitting. In a certain phase of the program, items are removed, based on adjacency rules. This hierarchical tree structure seems similar to the `parent` hierarchy of the present project (although the projects were developed simultaneously and independently). Also, Jacobs' space partitioning scheme to efficiently find neighbouring rectangles is very similar to my neighboring structure.

Nichol's work fits in the tradition of A. Michael Noll in the sense that grids are used as a starting point then are perturbed later, whereas the hierarchy of Jacobs gives priority to the idea of nesting (similar to the approach of the present article). Both Nichol and Jacobs are more careful regarding adjacency than the present work and looking back, this could be used to improve my on work.

Acknowledgments

I received very encouraging emails after the Leonardo publication [7] from emeritus professor A. Michael Noll, pioneer of the 1960s [18,19]. Special thanks go to the organizers of the *Call for Code* and to the people at TU/e who inspired and supported me during this project: Marina Toeters, Ellen Konijnenberg, Lidia Ritzema, Andy Van Eggelen, Sabine van Gent, Tom Jeltes, Bart van Overbeeke, Jun Hu and Matthias Rauterberg.

Disclosure statement

No potential conflict of interest was reported by the author.

References

[1] D. Andrzejewski, D. Stork, X. Zhu, and R. Spronk, *Inferring compositional style in the neoplastic paintings of Piet Mondrian by machine learning*, in *Computer Vision and Image Analysis of Art*, D. G. Stork, J. Coddington, and A. Bentkowska-Kafel, eds., Proceedings of SPIE-IS&T Electronic Imaging, SPIE Vol. 7531, Article 0G, 2010.
[2] R. Arnheim, *Art and Visual Perception*, Univ. of California Press, Berkeley, CA, 1997.
[3] C. Blotkamp, *Mondrian: The Art of Destruction*, Reaktion Books, London, 1994.
[4] M. van Bommel, H. Janssen, and R. Spronk, *Inside Out Victory Boogie Woogie, a Material Study of Mondrian's Masterpiece*, Amsterdam University Press, Amsterdam, 2012.
[5] K.B. Bruce, *Foundations of Object-oriented Languages: Types and Semantics*, MIT Press, Cambridge, MA, 2002.
[6] L. Feijs, *Mechanisms for naming, an algebraic approach with an application to Java*, Sci. Comput. Program. 39(2–3) (2001 March), pp. 149–188.

[7] L. Feijs, *Divisions of the plane by computer: Another way of looking at Mondrian's nonfigurative compositions*, Leonardo 37(3) (2004), pp. 217–222.
[8] L. Feijs, *Il linguaggio di Mondrian: Ricerche algoritmiche e assiomatiche*, in *Matematica e cultura 2006*, M. Emmer, ed., Springer Verlag, Berlin, 2006, pp. 181–194.
[9] L. Feijs, Een programma voor Victory Boogie Woogie, github.com/Elegant-Setup/LMG-Feijs/, file "Een Programma voor Victory Boogie Woogie.pdf" (filed April 3, 2013).
[10] L. Feijs and Y. Qian, *Component algebra*, Sci. Comput. Program. 42(2–3) (2002), pp. 173–228.
[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Boston, MA, 1995.
[12] M. Haring, M*alevich, computational simulation of an abstract artistic style. Bachelor thesis bachelor kunstmatige intelligentie*, University of Amsterdam, faculty of science, science park 904, 1098XH Amsterdam. Available at staff.fnwi.uva.nl/b.bredeweg/pdf/BSc/20102011/Haring.pdf.
[13] J. van Hemert and G. Eiben, *Mondrian art by evolution*, in *Proceedings of Dutch/Belgian Conference on Artificial Intelligence*, E. Postma and M. Gyssens, eds., (BNAIC'99), 1999, pp. 291–292.
[14] E. Jacobs, *MVBW−1*. (2013). Available at github.com/Elegant-Setup/Edwin-Jakobs/blob/master/README.md.
[15] H. Jaffé, *Piet Mondrian* (masters of art), Harry N. Abrams, Inc., 1985.
[16] J. Joosten and R. Welsh, *Mondrian: Catalogue Raisonné*, Harry N. Abrams Inc., New York, NY, 1996.
[17] P. Locher, K. Overbeeke, and P.-J. Stappers, *Spatial balance of color triads in the abstract art of Piet Mondrian*, Perception 34(2) (2016), pp. 169–189.
[18] A. Michael Noll, *Human or machine, a subjective comparison of Piet Mondrian's composition with lines and a computer-generated picture*, Psychol. Rec. 16(1) (1966), pp. 1–10. Reprinted in James Hogg, ed., Psychology and the Visual Arts (London: Penguin Books, 1969) pp. 302–314.
[19] A. Michael Noll, *The beginnings of computer art in the united states: A memoir*, Leonardo 27(1) (1994), pp.39–44.
[20] P. Mondrian, *Plastic Art & Pure Plastic Art (1937)*, in *Circle*, J.L. Martin, B. Nicholson, and N. Gabo, eds., Faber & Faber, London, 1937, pp. 41–56. Available at arthistoryproject.com/artists/piet-mondrian/plastic-art-and-pure-plastic-art-part-2/.
[21] K. Nichol, *Mondify*, 2013. Available at github.com/Elegant-Setup/kiri-nichol/blob/master/knichol_mondify_beschrijving.pdf.
[22] C. Reas and B. Fry, *Processing, A Programming Handbook for Visual Designers and Artists*, MIT Press, Boston, MA, 2014.
[23] G. Schufreider, *Mondrian's Opening: The Space of Painting*, lecture delivered in the Department of Fine Arts, Harvard University, April 1997. Available at www.focusing.org/apm_papers/schuf.html.
[24] A.G. De Silva Garza and A.Z. Lores, *Automating evolutionary art in the style of Mondrian*, in *GECCO 2004*, LNCS 3103, K. Deb *et al.*, eds., Springer-Verlag, Berlin, 2004, pp. 394–395.
[25] A.G. De Silva Garza and A.Z. Lores, *Case-based Art*, in *Case-Based Reasoning Research and Development*, ICCBR 2005. Lecture Notes in Computer Science, vol 3620, H. Muñoz-Ávila and F. Ricci, eds., Springer Berlin, Heidelberg, 2005.
[26] J. Simon, Interview with John F. Simon Jr., *Artificial.dk* November 2003. Available at http://www.artificial.dk/articles/simon.htm.
[27] R.P. Taylor, *Mondrian and nature: Recent scientific investigations*, in *Chaos and complexity in the arts and architecture*, Sala, N., ed., Nova Science Publishers Inc., New York, NY, pp. 25–37.
[28] L.A. Zadeh, *Fuzzy logic, neural networks, and soft computing*, Commun. ACM. 37(3) (1994), pp. 77–85.
[29] K. Zhang, S. Harrell, and X. Ji, *Computational aesthetics: On the complexity of computer-generated paintings*, Leonardo 45(3), pp 243–248. Available at muse.jhu.edu/journals/leonardo/v045/45.3.zhang.pdf.