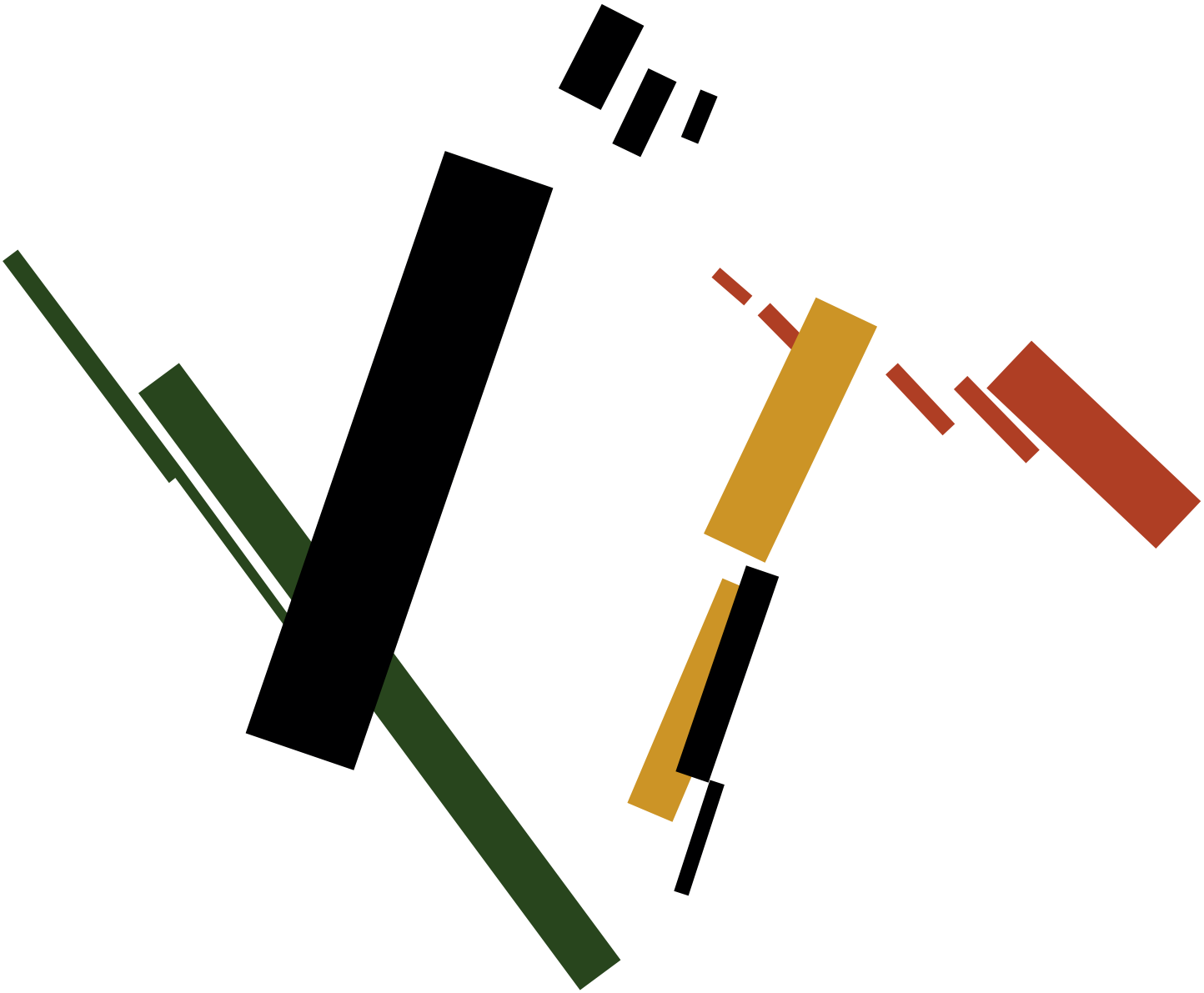


computational simulation of an abstract artistic style

malevich

bachelor thesis 6ec bachelor kunstmatige intelligentie
university of amsterdam, faculty of science, science park 904, 1098 xh amsterdam

mirjam e. haring
5743001

supervisor
prof. dr. ir. r.j.h. scha
institute for logic language and computation

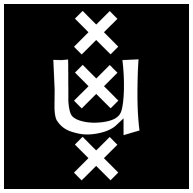
Computational Simulation of an Abstract Artistic Style: Malevich

Mirjam E. Haring
5743001

Bachelor thesis
Credits: 6 EC

Bachelor Opleiding Kunstmatige Intelligentie

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam



Supervisor
Prof. dr. ir. R.J.H. Scha

Institute for Language, Logic and Computation
Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

June 24, 2011

Abstract

This paper describes how abstract art can be approached in an algorithmic way, by constructing a formal language that characterizes the art style. Malevich's Suprematism is the art movement that is focused on in this paper. By analysing Malevich's paintings, a formal language is constructed that can generate similar images. The formal language describes the individual objects and their joint composition. The formal language is used to make a generative computer program that is able to construct and output new images. To evaluate the results of the program, these outputted images are compared to Malevich's paintings.

Keywords: *Formal language, Malevich, Generative Program, Computer Art*

Contents

1 Introduction

On Friday 11^{th} of February 2011, Prof. dr. ir. R. Scha gave a lecture about artificial art. He demonstrated how art can be made by computers but that not all computer generated art is 'good'. But what makes an art generator a good one? Scha explained how different approaches of generation can influence their outcome, and demonstrated this with examples of Mondrian generators. Loe Feijs for example uses a formal language to generate objects, which influence each other, making the painting grow in an organic way. Simple random generators on the other hand, make paintings that try to resemble Mondrian, but they fail miserably; the composition seems off and does not match up to the images Feijs produces.

This paper describes how abstract art can be approached in an algorithmic way, by constructing a formal language that characterizes the artistic style. Malevich's Suprematism [1] is the art style that is focused on in this paper. By analysing Malevich's paintings, a formal language is constructed that can generate similar images. The formal language describes the individual objects and their joint composition. The formal language is used to make a generative computer program that is able to construct and output new images. To evaluate the results of the program, these outputted images are compared to Malevich's paintings.

2 Research Question

Is it possible to capture Malevich's Suprematism in a formal language and use this language to construct new compositions? The answer to the first question will be needed in order to address the second question. The goal is to make a generative program that uses the formal language and outputs new images that resemble Suprematist paintings of Malevich.

The word *composition* will be used to indicate the total image (or painting) and the relations between the objects in it. Analysis of Malevich's paintings will be needed in order to find a formal language. By creating duplicates of his paintings, it will help discover which operations can be done on shapes in the image, like rotation, scaling and colouring.

Extracting the relations concerning the composition of the paintings is necessary in order to form a formal language. When these relations are successfully transcribed to a formal language, implementing them in a prototype version of a generative program can be done. This prototype version will show if the found rules are correct or not, and if new rules should be added to improve the results. By adding the new rules, the final formal language will be constructed and implemented in the generative program.

3 Literature Study

3.1 Malevich's Suprematism

Kazimir Severinovich Malevich was born in Ukrain in 1879. He studied at the Moscow School of Painting, Sculpture and Architecture from 1904 to 1910. In 1910 he began to use more geometric shapes in his paintings and around 1915 he started to define his style as Suprematism. He found that reality could not be captured right by paintings, and that art itself should not try to resemble reality and should therefore be non-figurative.

As Malevich states in his *Manifesto of Suprematism*:

> The 'thing' (the nose, the eye, etc.) is raised to the criterion by which an artistic (pictorial) representation is judged and thus the singular opinion of the public that art is not creative but imitative is clearly expressed.

Malevich's critique of an aesthetic imitation of reality suggested different ways of representing "reality" - for example by cancelling out all known objects and creating new ones, simple and unique. These objects could convey the spiritual meaning he was striving to capture and thus represent his ideas of pure art - art not driven by the reality of the object world but by the artist and his ideas. [2]

3.2 Computer Art

The earliest computer art dates back to 1960 when Desmond Paul Henry invented the Henry Drawing Machine. The main component of the Henry drawing machine was the bombsight computer (a mechanical analogue computer). Each drawing took between two hours to two days to complete. Henry's drawing machines were quite unlike the conventional computers of the 1960s since they could not be pre-programmed nor store information. Instead his machines relied upon the chance relationship in the arrangement of each machine's mechanical components. [3]

Since then, more drawing computers have been developed, the best known being AARON by Harold Cohen.[4] AARON is a program designed to investigate the cognitive principles underlying visual representation. It has been under continuous development for fifteen years (starting in 1973) and it is now able autonomously to make "freehand" drawings of people in garden-like settings. Two types of knowledge are needed for AARON: object-specific knowledge of how people are constructed and how they move, together with morphological knowledge of plant growth: and procedural knowledge of representational strategy. AARON demonstrates that, given appropriate interaction between domain knowledge and knowledge of representational strategy, relatively rich representations may result from sparse information.

Other computer art pioneers were Frieder Nake, Georg Nees and Michael Noll. Dr. A. Michael Noll was one of the earliest to use digital computers in the

visual arts and stereoscopic animation. Noll used a digital computer and microfilm plotter to produce a semi-random picture similar in composition to Piet Mondrian's painting "Composition With Lines." Noll varied the degree of randomness, which ranged from a grid-based placement of varying-length lines to a completely random placement. Reproductions of both pictures were then presented to 100 people whose tasks were to identify the computer picture and to indicate which picture they preferred. Only 28% of the participants were able to correctly identify the computer-generated picture, while 59% of the participants preferred the computer-generated picture. [5]

Loe Feijs used a different approach to construct Mondrian art. He talks about a formal language for constructing the paintings. He says the language consists of two parts: genotype and phenotype. [6] These terms, that are borrowed from biology, describe how objects are formed. The genotype describes what kind of object we are dealing with (i.e. a black line), the phenotype describes how this object is formed (i.e. when two lines meet, one stops and the other one goes on). The phenotype is dependent of the surrounding objects with which it interacts. Feijs started with constructing the grid through the interaction of its elements, instead of Noll, who started from a grid, deviating from it by adding randomness.

The method Feijs talks about will be used for this project: finding a formal language to describe an art style. The part of Noll's approach of adding randomness will also be used.

4 Method

4.1 Using Python Tkinter

Tkinter is a package in Python that can be used to construct Canvas Widgets, on which the constructed images can be placed. Tkinter is used to duplicate some of Malevich's paintings to see which operations are needed to make an image. Some examples can be found in the Appendix (10.3) For example: what operations are needed to place two red rectangles on a canvas with the same rotation? This immediately gave some issues, due to Tkinter's inability to rotate rectangles. (see 10.2)

By duplicating Malevich's paintings, relations between objects in their composition are discovered, like their tendency to be parallel or perpendicular, and grouped objects usually having similar sizes. The discovered rules form the beginning of the formal language.

4.2 Constructing a formal language

The formal language will be used to describe Malevich's paintings. Malevich Suprematist paintings range from simple (containing one object) to very complex (see figure 1). The formal language that will be described will be applicable to a subset of Malevich's Suprematism, consisting of paintings that are not too complex and only contain circles and rectangles. For the following analysis a subset of 10 paintings was used (see section 10.4, seven paintings were also used

in the survey).

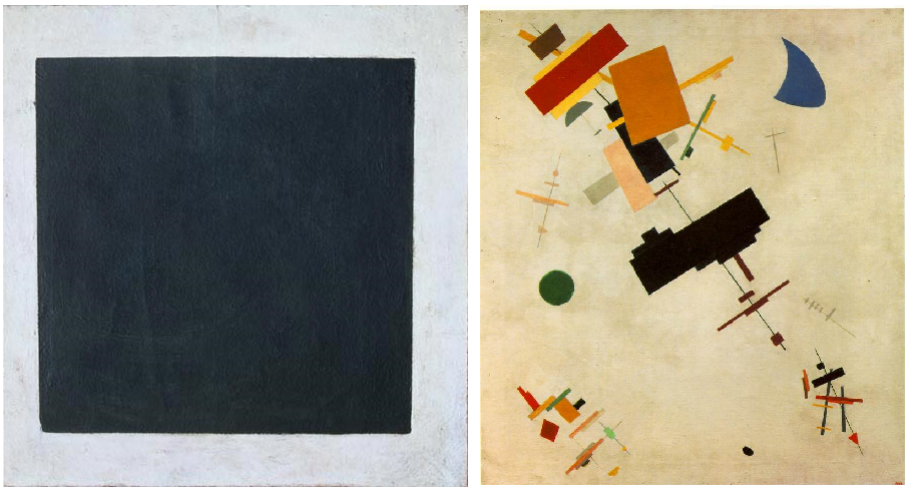


Figure 1: Malevich's Suprematist paintings (simple and complex)

The formal language will be used to describe and construct images in the following way:

$$image = << shape,\ position,\ colour,\ rotation >, < ... >, ..., < ... >>$$
$$position = < x1, y1, x2, y2 >$$

Each object on the canvas has a shape (rectangle or oval), a position (defined by four coordinates), a colour and a rotation. The subset is used for analysis to extract rules that are applicable to the composition. The set of paintings that is used only contain rectangles and ovals. Malevich also made Suprematist paintings that contain triangles, but it was chosen not to use these in this project.

To validate the new found rules they were implemented in a generative program. A random number of objects is placed on a canvas and the rules are used to construct their composition. But the outputted compositions of the program were nothing near the compositions of Malevich. The rules had to be improved and new rules had to be constructed. By using the implemented program, it is easier to see when new found rules are correct, because they immediately improve the results. It also visualizes where there is room for improvement and what kind of rules should be added (i.e. when all elements are placed in the right bottom corner, a rule should be added to enable a more balanced placement).

The first version of the formal language is shown in figure 2. It is visualised as a finite State Automaton, where the states represent the rules and the numbers represent the probabilities for transition.

The first step is to create an object that can be either a rectangle or an oval. A random size, position, colour and rotation (in radians) is assigned and the next

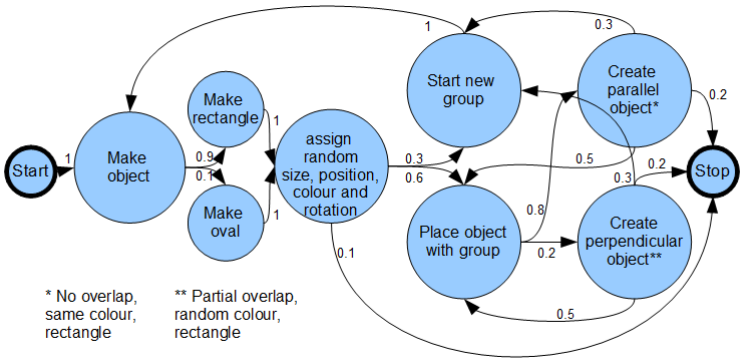


Figure 2: Finite State Automaton of first version of formal language

step is selected. This can either be to place an object near the previous one, or to create a new group by placing a new random object. When an object is placed in a group, it is usually parallel, but can also be perpendicular. When it is perpendicular it tends to overlap an item in the group. When it is parallel, it is usually separated by the other object with a small white space (so no overlap). Now a new object can be placed with the group, or randomly on the canvas. Results of this rule system can be found in the Appendix (section 10.1).

The initial probabilities are rough estimates based on the analysed paintings. A statistic analysis was done to derive better estimates of probabilities.

4.2.1 Probabilities of Formal Language

For acquiring better estimates of the probabilities used in the formal language the selected set of Malevich's paintings is used. Each of the 10 paintings was described in terms of the Finite State Automaton. Through saving which branches were taken to construct the painting, better probabilities were acquired. Figure 3 shows the counts per branch from which the improved probabilities were calculated. With these improved probabilities, further tuning of the language was possible.

4.2.2 Improvements to the formal language

Objects tended to be clustered on one point on the canvas, while Malevich's compositions are more balanced. To prevent this clustering the canvas was divided into nine positions (a 3x3 grid). Whenever a new object is placed on the canvas (so not when an object is placed in a group), it is placed in one of these areas. The list of positions pops the area and places the new object in that position. The remaining items in the position list represent the unused areas, which will be used when a new object is placed.

Another rule which had to be added was that objects in groups are always rectangles, so when an oval is placed on the canvas, the next step is always to start a new group.

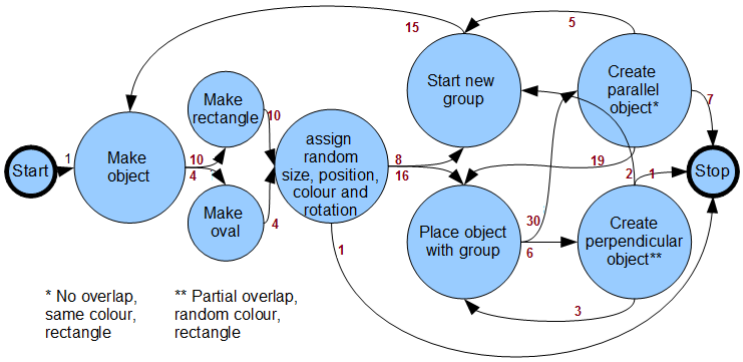


Figure 3: FSA with counts per branch after running 10 paintings

After all the objects are placed on the canvas, they are rotated accordingly to their rotation value. Sometimes this causes the objects to rotate (partially) off the canvas. In Malevich's paintings the canvas is always the right size to fit the composition. For the program this means that the canvas has to grow according to the objects in it, so it will always be the right size for the composition.

4.3 Complete formal language

The formal language described below specifies how each object gets assigned its value and how the image is constructed. It is a formal definition of the FSA described in section 4.2 with additional rules described in the section above. It can be used to describe and also generate images.

The position of an object is specified with four coordinates. $x1$ And $y1$ represent the top left corner (before rotation) and the $x2$ and $y2$ coordinates represent the right bottom corner. The positions 0 to 8 represent a 3x3 grid on the canvas. They can be popped from the position list when a new group is created, it is then no longer present in the position list. The popped position number corresponds to an area on the canvas where the new object will be placed. The function $place\_with\_group(y, x)$ places object y with group x. When parallel is True, the object will be placed parallel to x, separated by a small distance of 10 pixels. It can be placed above or below in horizontal or vertical direction. When parallel is False, the object is placed perpendicular to the group ($rotation - \pi$), having a partial overlap. { } Is used to identify a collection, [ ] is used to identify a range.

$words = \{rectangle,\ oval\}$

$canvas =< 400, 600 >$
$colours = \{black, yellow, green, blue, red, brown\}$
$positions = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$
$position =< x1, y1, x2, y2 >$
$parallel = \{True, False\}$

- $\forall x : shape(x) \in words$

- $\forall coord\ in\ position(x) \in canvas$
- $\forall x : rotation(x) \in [0, 2\pi]$
- $if\ start\_new\_group(x) \rightarrow position(x) = positions.pop()$
- $if\ place\_with\_group(y, x) \wedge shape(x) = rectangle \wedge parallel = True \rightarrow$
  $colour(y) = colour(x) \wedge$
  $rotation(y) = rotation(x) \wedge$
  $shape(y) = rectangle \wedge$
  $position(y) =< x1, y1, x2, y2 >, position(x) =< u1, v1, u2, v2 > \wedge$
  $x1 \in \{u1, u2 + 10\} \wedge$
  $y1 \in \{v1, v2 + 10\} \wedge$
  $x2 = [x1, size\ x] \wedge$
  $y2 = [y1, size\ x]$
- $if\ place\_with\_group(y, x) \wedge shape(x) = rectangle \wedge parallel = False \rightarrow$
  $colour(y) \in colours \wedge$
  $rotation(y) = rotation(x) - \pi \wedge$
  $shape(y) = rectangle \wedge$
  $position(y) =< x1, y1, x2, y2 >, position(x) =< u1, v1, u2, v2 > \wedge$
  $(x1, y1) \in position(x) \wedge$
  $x1 = [u1, u2] \wedge$
  $y1 = [v1, v2] \wedge$
  $x2 = [x1, size\ x] \wedge$
  $y2 = [y1, size\ x]$
- Stop.

5 Implementation

The implementation process was partially parallel to the construction of the formal language, because the language had to be tested and validated while developing.

The program consists of two main parts. One part containing the formal language (using Python), one part containing the construction of the image (using Python module Tkinter). The formal language module outputs a list of lists, where the nested lists represent the separate objects with their attributes (shape, position, colour and rotation). The construction module takes this list as input, calculates the coordinates of objects after rotation and places the objects on a canvas.

5.1 Transcribing formal language to Python

For each rule in the language a rule in Python was written. [and] in python are used to describe a list. Squares is the position list described in section 4.3, square represents a position on the 3x3 grid on the canvas. Objects is the list which represents the composition, object represents one object in the composition.

- Rule 0(objects, canvas, squares): return objects
  determine grid number of new group. It calls rule 1 and rule 2.
- Rule 1(): return shape
  determine shape of object; rectangle or oval.
- Rule 2(canvas, shape, square): return [shape, position, colour, rotation]
  determine rotation and colour of new object. It calls rule 6.
- Rule 3(objects, canvas, squares): return objects
  places object with group. Calls rule 4 or rule 5.
- Rule 4(object, canvas): return [shape, position, colour, rotation]
  places object parallel to group.
- Rule 5(object, canvas): return [shape, position, colour, rotation]
  places object perpendicular to group.
- Rule 6(square, canvas): return position
  determine exact position of object based on square.
- Rule Stop.

Which rules are called is based on a random number representing a probability. The probabilities are according to the ones found in section 4.2.1. When the Stop rule is reached, the objects are passed on to the construction module, where the objects will be placed on a Canvas Widget.

5.2 Constructing the image

The image will be constructed using Python module Tkinter. The objectlist representing the image has the following syntax:

$$objects = [[shape,\ position,\ colour,\ rotation], [...], ..., [...]]$$
$$position = [x1,y1,x2,y2]$$

Before the objects are placed, they have to be rotated. But due to Tkinter's inability to rotate rectangles, the rectangles have to be transformed to polygons (see section 10.2). To rotate a polygon, the rotation angle has to be transformed to a complex number representing that angle, by multiplying each coordinate to the complex number, the new coordinates can be found.

When all the objects are placed on the canvas a final check is done to see if any objects have rotated outside the canvas. If this is the case, the canvas will be altered to fit the composition. Another solution was to reduce the objects that rotated off the canvas, but this would influence the composition, therefore the other method is used.

6 Results

6.1 Survey

A survey was designed to evaluate the results of the program. 10 Randomly selected images made by the program were used, and 7 images of Malevich. The photos of the paintings of Malevich were cleaned up to resemble the computer generated output, because this would otherwise influence the choice of the participants. Figure 4 shows such an alteration.

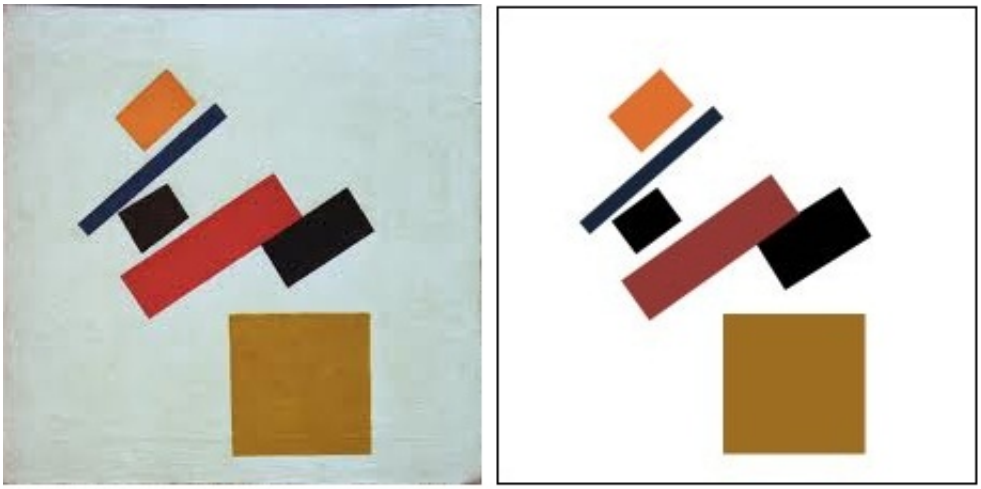


Figure 4: Malevich painting and its altered version used in the survey

Participants were asked to rate their knowledge of art, ranging from 1 (none) to 5 (expert). They were shown an image and asked if it was designed by Malevich or a computer. The second question was if they liked the image (range 1 to 5). A total of 17 images were shown. The randomly selected images of the program and the images of Malevich that were used are shown in the Appendix (10.4).

A total of 54 completed surveys were used for the following analysis. The average knowledge of art of the participants was 2.89.

The average value of appreciation of the images was 2.56 (on a scale from 1 to 5). The average value of appreciation for the images of Malevich was 2.89, for the computer images it was 2.33. This means though there is only a small difference, the images of Malevich are on average more likeable then the computer generated ones. The image with the highest value was by Malevich (3.24). The highest score for a computer generated image was 3.06, which is higher than the average of images by Malevich, which shows it is possible for the computer program to make likeable compositions. Table 1 shows the appreciation values for Malevich and Computer generated paintings according to the knowledge of art of the participants.

Average knowledge of art of Participants	Appreciative value of Malevich	Appreciative value of Computer	Average Appreciative value
2.89	2.89 ($\sigma = 0.23$)	2.33 ($\sigma = 0.79$)	2.56 ($\sigma = 0.43$)
5	2.10 ($\sigma = 1$)	2.03 ($\sigma = 1$)	2.06 ($\sigma = 1$)

Table 1: Appreciation Values and their σ (Standard deviation)

This table shows that people with a higher knowledge of art rate the images lower, but the difference between the computer generated and Malevich is smaller. The standard deviation in the group of experts is higher because the group consists of less participants, making the results more fluctuating.

On average, people could identify 64.46% correctly, when choosing if an image was computer generated or by Malevich. This is significantly higher then chance ($\alpha = 0.05$, 50% being chance level), which means that people are able to distinguish between computer made and Malevich. This also means that there is room for improvement in the program.

Average knowlegde of art	Computer as Malevich	Malevich as Computer	Correct
2.89	22.5% ($\sigma = 18.75\%$)	45.0% ($\sigma = 15.86\%$)	64.5% ($\sigma = 20.54\%$)
5	43.3% ($\sigma = 35.32\%$)	71.4% ($\sigma = 30.17\%$)	45.1% ($\sigma = 35.08\%$)

Table 2: Identification of images with their σ (Standard deviation)

Table 2 shows how many times computer generated images were mistaken for Malevich, and vice versa. The art experts show worse results than the average group, but they have a much larger standard deviation due to the small number of participants in the group of experts.

Some of the computer generated images that were used in the survey were easy to identify (100% correct and 90.3% correct), but others were difficult (only 35.5% correct). In order to improve the program, evaluation of these easy-to-identify images must be done. This will show why these images don't resemble Malevich, and what must be done to prevent the program from constructing these.

The computer generated image that was correctly identified by 100% of the participants is shown in figure 5. It is easy to see this is not an image made by Malevich due to the compactness of the composition. There are too many objects overlapping and they are all located in the center of the canvas. A rule was added to prevent this clustering (described in section 4.3), but clearly it does not work sufficiently. The rule could be altered (by changing the size of the

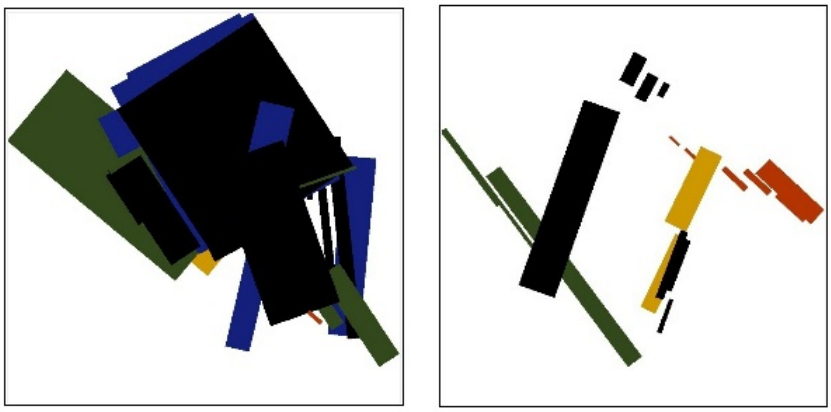


Figure 5: Left: 100% identified as computer generated. Right: 35.5% identified as computer generated

3x3 grid), or an additional rule could be added to prevent overlapping, which is also a cause of this clustering.

The computer generated image that was the hardest to identify is shown in figure 5 (this was also the computer generated image with the highest appreciative value). This image has a more balanced composition and there is only a small portion of overlap. There are clear distinct groups, made visible by their different rotation and colour. These are all characteristics of Malevich's paintings that are captured in this image.

7 Evaluation

When I started this project I thought it would definitely be possible to create a formal language and a generative program, but I did not know if they would be strong enough to generate images that could be mistaken for Malevich. The survey shows there are indeed generated images that satisfy this condition.

The results of the survey show that people are able to distinguish between computer generated and Malevich. The computer generated images showed signs of aliasing, which could help identify them. This means the results of the survey can be viewed more positively towards the program. Because even with this computer characteristic people still made mistakes identifying the computer generated image. I wanted to improve the quality of the output of the program by removing the aliasing on skewed lines, but I was unable to because Tkinter does not support anti-aliasing.

Another thing the survey shows: there is room for improvement in the generative program. Some of the images it generates are clearly computer made, while others are hard to identify and often mistaken for Malevich. Adding rules to prevent these first ones from being made will definitely improve the program.

The selected set of paintings used in this project only contain rectangles and circles. But Malevich also made paintings with triangles. Because the set of paintings containing triangles was so small, I chose not to use them and focus on the larger set of images containing only rectangles and circles. Adding the triangles to the formal language would improve the result, because more of Malevich Suprematismic paintings could then be described.

I am pleased with the generative program itself, because it generates images within seconds. The only thing I would like it to have is a Graphical User Interface (GUI). That way, users can press a button and a new 'Malevich' will appear on their screen.

Overall I am satisfied with the results, even though there is still room for improvement.

8 Conclusion

In section 3.2 is described how Feijs used a formal language to capture Mondrian art. The images constructed with this formal language were hard to identify as computer made. To see if it was just Mondrian art that could be captured in a formal language, or if there are other artistic styles that can be described this way, I tried to find a formal language for Malevich's Suprematism.

The first part of the research question is: is it possible to capture Malevich's Suprematism in a formal language? The constructed formal language (described in section 4.3) can be used to describe all Malevich's Suprematismic paintings that were selected. Therefore the answer to the first question is: yes, it is possible to capture (a subset of) Malevich's Suprematism in a formal language.

The second part of the research question is: can this language be used to make new compositions? In other words: is the formal language strong enough to have generative strength? The results from the survey show that some generated images are not Malevich-like and easy to identify as computer made, and other images are not recognized as computer made and often mistaken as Malevich. Therefore the answer to the second question is: yes, the language can be used to make new compositions, but only a part of the constructed compositions resemble Malevich's style. Adding additional rules can help prevent the non-Malevich-like compositions from being constructed.

9 Discussion

The constructed formal language can be used to describe Malevich's paintings. With statistical analysis the probabilities for the language were determined. To see if this language was complete, a program that interprets the language constructs new images. If the language is complete (all rules that are needed are present) and the implementation of the program works correct, then the con-

structed images should resemble Malevich's style. The results from the survey show that although not all images are good, there are some that are indeed Malevich's style. From this I must conclude that the language is not yet complete and additional rules should be added.

But does every generated image has to be good? When an artist makes a painting, he often starts over and he selects which paintings he finds good enough to exhibit. So the program is actually more artist-like when not all generated images are good. But then the program should also be able to decide which images are good and which are not, which could be accomplished with a learning algorithm and an annotated dataset containing images and their appreciative values. But that is a project in itself.

Table 2 shows the results of identification of images. It is surprising to see 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 of 45, have less knowledge of computers, therefore being less able to spot the computer characteristics.

Finding the differences between the Malevich-like compositions and the others is needed in order to develop new rules that can be added to the formal language to improve the generative program. The image shown in figure 5 was identified as computer made by 100% of the participants, and it clearly shows signs of clustering. Therefore one thing these additional rules will be concerned with is preventing clustering and overlapping of objects on the canvas.

But is this clustering the only reason why some images look computer made? Or are there other factors involved? For example, the colouring of different objects. The only rule in the formal language concerning colouring describes how objects in the same group should have the same colour. There are no rules concerning colouring of neighbouring groups, the first object in a group gets assigned a random colour, which is not what Malevich does.

A solution for the colouring problem could be to do a statistical analysis on Malevich's paintings to see how the colouring is done. For example, almost every painting has a black object in it. Besides the probabilities for different colours, relations between the use of different colours must be found too. For example, when there is a black object, at least two other colours are used on different objects.

Overall, the improvements that could be made to the formal language and the program are: anti-aliasing, a GUI, adding triangles, adding additional rules to prevent clustering and adding a colouring method.

References

[1] Kasimir Malevich. *Die Gegenstandslose Welt.* Bauhaus-Bucher, Band 11. Munich: Albert Langen, 1927.

[2] Y. Ziv. Parallels between suprematism and the abstract, vector-based motion graphics of flash.

[3] D.P. Henry. Art and technology. Bulletin of the Philosophy of Science Group, Newman Association, No. 53, 1964.

[4] Harold Cohen. How to draw three people in a botanical garden. pages 846–855, 1988.

[5] A.M. Noll. Human or machine: A subjective comparison of piet mondrian's" composition with lines"(1917) and a computer-generated picture. *The Psychological Record*, 1966.

[6] L. Feijs. Divisions of the plane by computer: another way of looking at mondrian's nonfigurative compositions. *Leonardo*, 37(3):217–222, 2004.

[7] Tkinter tricks: Using complex numbers to rotate canvas items. http://effbot.org/zone/tkinter-complex-canvas.htm.

10 Appendix

Figure 6: Suprematism with Eight Red Rectangles. 1915

10.1 Results of first program

Figure 7 shows the images made by the program which implements the first version of the formal language.

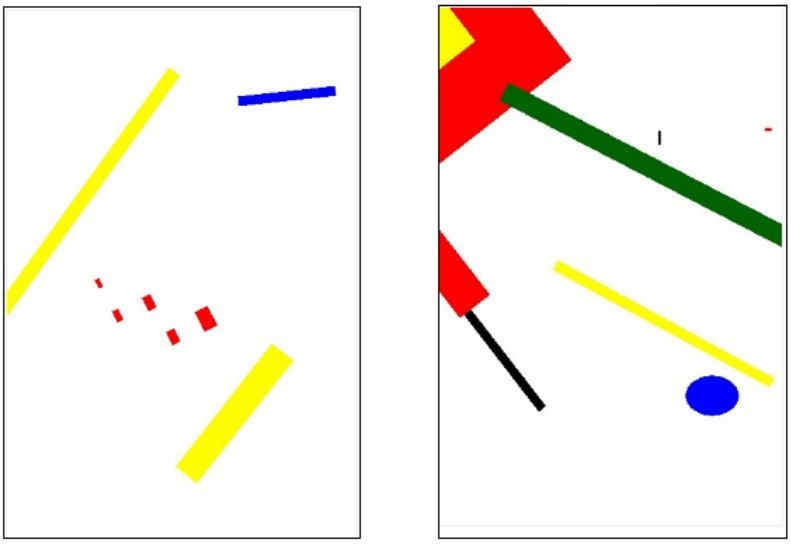


Figure 7: Results of the first version of the generative program

10.2 Tkinter

Tkinter is unable to rotate rectangles. Rectangles are defined by four coordinates, representing the top left and bottom right corners, which are placed on the canvas. In order to rotate this rectangle, for each of the four corners a new position must be calculated, i.e. the position of the corners after rotating. This is done by multiplying each coordinate with a complex number representing the angle in radians. But four corners (eight coordinates) are needed (instead of two corners), so the rectangle must first be transformed to a polygon. [7] To solve this issue I created a *rectangeToPolygon* and a *rotatePolygon* method.

10.3 Duplicates

Figure 8 contains paintings of Malevich on the left side and its 'duplicate' on the right side. Rotation is not used because a separate module had to be written to rotate rectangles (which was written after the duplication phase).

10.4 Images used in the survey

Figures 9 and 10 show the images that are made by the program and were used in the survey. Figure 11 shows the images by Malevich that were used in the survey. They were slightly altered (cleaned up) to resemble the computer generated output.

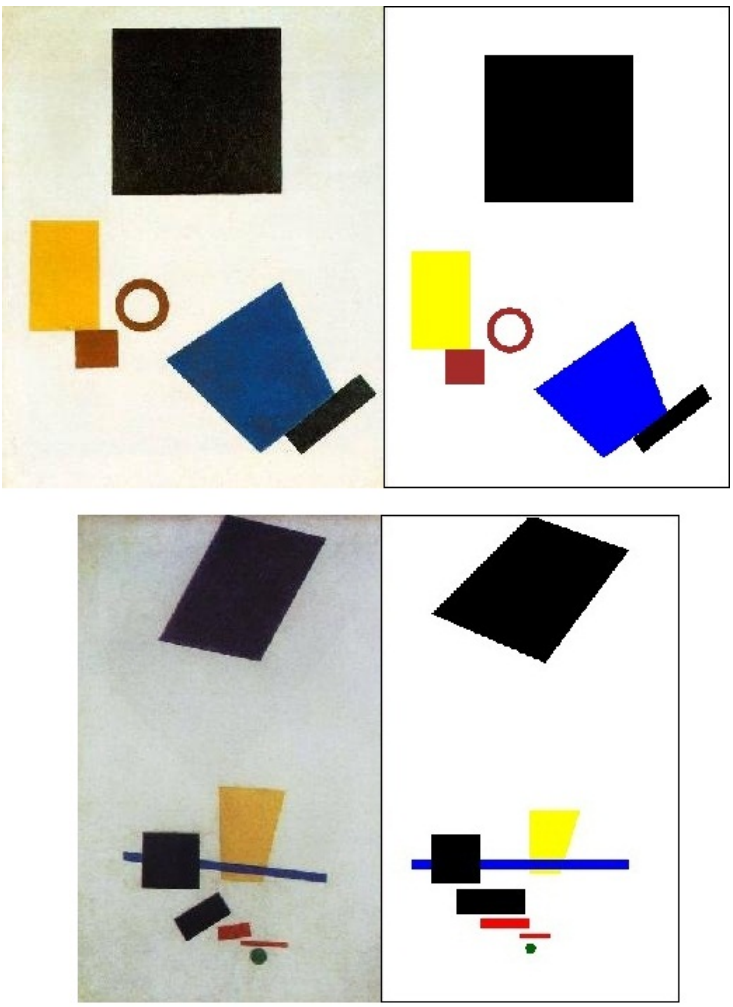


Figure 8: Duplicates of Malevich's paintings

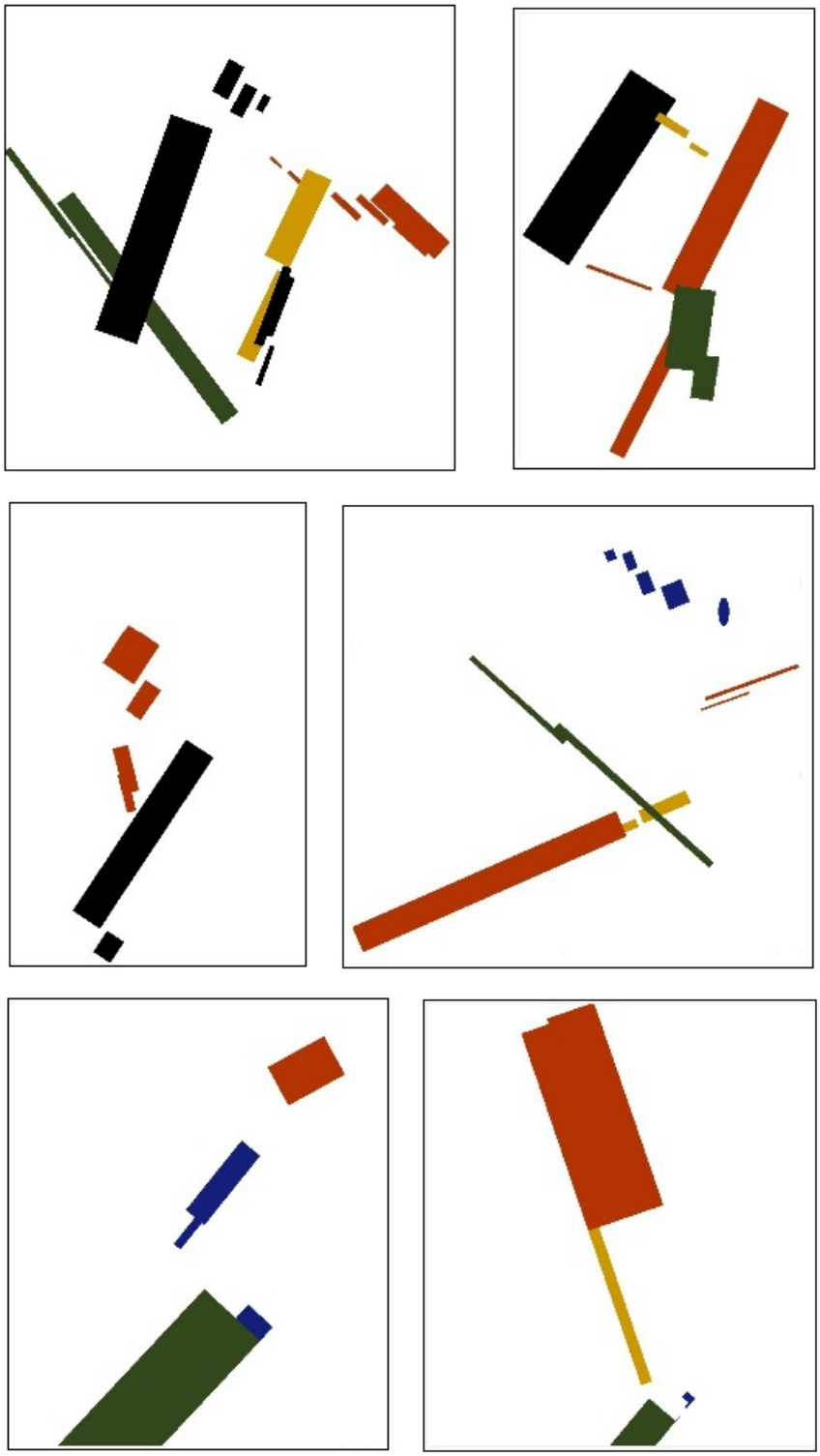


Figure 9: Images by the computer program used in the survey

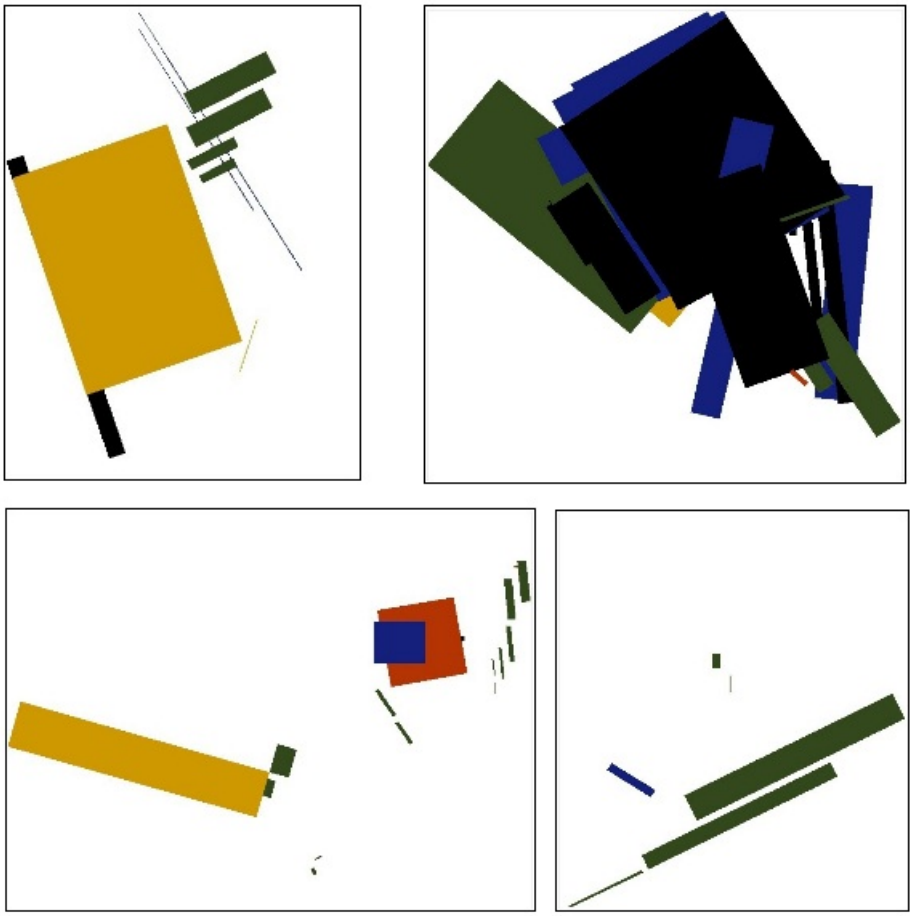


Figure 10: Images by the computer program used in the survey

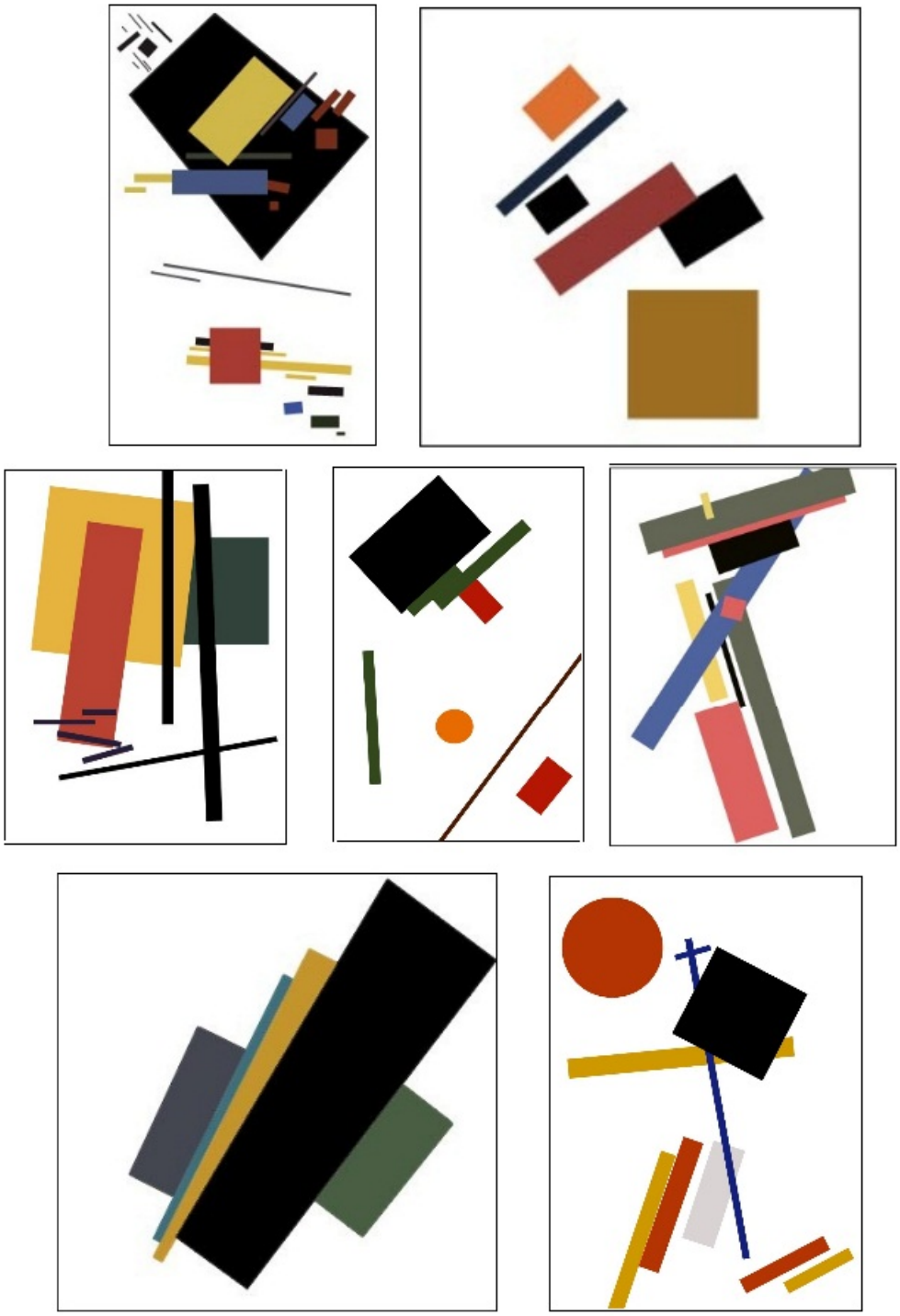


Figure 11: Images by Malevich used in the survey