ThinkingSketch Cookbook

Version 2.00

ThinkingSketch Unit

www.sketch.jp

2002年8月27日

図 1: ThinkingSketch Unit Members

はじめに

ThinkingSketch は人間が絵をかくことを支援するソフトウエアです。その一番の特徴は、コンピュータに図形を発生させることです。

ThinkingSketch には、ただ単に、絵をコンピュータに勝手に発生させるのではありません。描き出す図形の内容に対して人間が注文をつけることにより、「自分で描きたい図形に、より近いもの」を生成することができる仕組みが備わっています。

この「User's Guide」は ThinkingSketch をはじめて使う方々に、ThinkingSketch の基本的な機能を紹介するためのものです。

さあこれから、みなさんを ThinikingSketch の世界へご案内しましょう。

ThinkingSketch Unit 一同

第1章　準備しよう

ThinkingSketch を利用するためには Java 言語で書いたプログラムを実行できる環境が必要です。一番簡単なのは、JRE という Java の実行環境をインストールすることです[1]。ほかにも JDK をインストールすることでもできます[2]。

あなたの環境にまだ、Java がインストールされていないのであれば、java.sun.com というウエッブサイトから最新の JRE または JDK をダウンロードし、指示にしたがってインストールを行ってください。

Java をインストールするために必要なデータのサイズは数メガバイトあります。インターネットからダウンロードされる際はそのことに注意してください。また、JRE は雑誌の付録として提供されていることもあります。

Java の実行が可能になったら、ThinkingSketch Unit が提供する CD-ROM もしくは `http://www.sketch.jp/` から ThinkingSketch をダウンロードしてインストールしてください。

そのあと本書の指示にしたがって ThinkingSketch を実行してください[3]。

[1] この環境はプログラムは作れませんが Java の実行が可能です。
[2] この環境は少し大きくなりますが、Java のプログラム作成と実行がともに可能です。
[3] run、ts といったコマンドを実行することによって ThinkingSketch は実行されるようになっているはずです。

第2章　試してみよう

2.1　スタートアップ

パッケージの解説したがってThinkingSketchを実行すると、コンピュータのスクリーンは全面がThinkingSketchのウィンドウになります。

ウインドウの下の部分に細長い、長方形の部分があります。ここにキーボードからテキストを入力します。

キーボードから、文字列を入力してエンターキーを押すとコマンドが実行されます。

図 2.1: 最初の画面

2.2　モンドリアンをまねてみる

20 世紀前半に活躍したオランダの画家、ピエト・モンドリアン (1872-1944) をご存知ですか?ThinkingSketch を最初に使うにあたって、ひとまずモンドリアンが描いたような抽象的なパターンを生成してみましょう。

ウィンドウ下の長方形の部分[1]にマウスポインタを移動し、キーボードから、

```
mondrian
```

という文字列を入力してみましょう。エンターキーを押すと、長方形で構成された絵が画面に現れます。

[1] ここをコマンド入力エリアと呼びます。

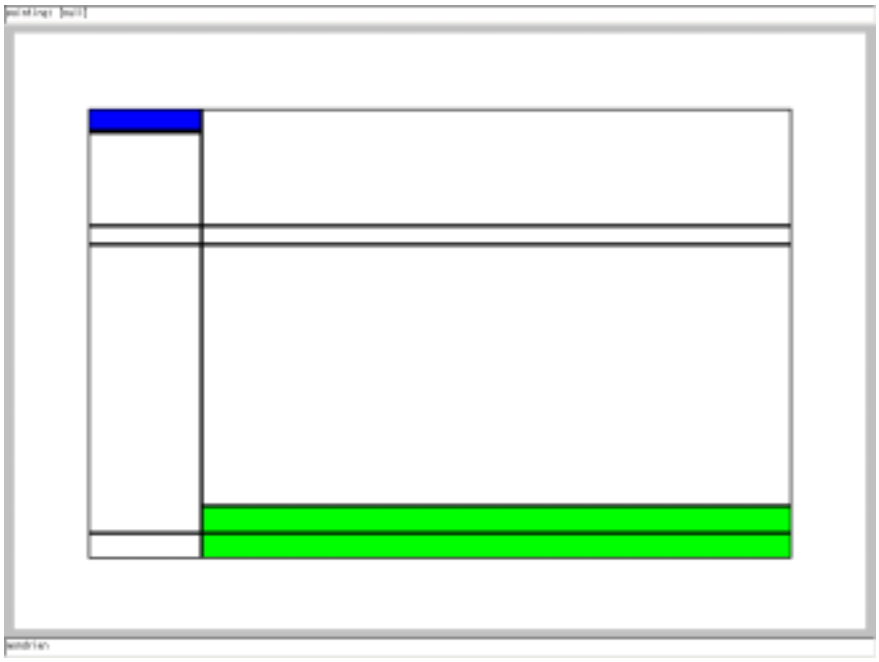

図 2.2: モンドリアンの作品に似させてつくってみた図

2.3　mondrian コマンド

図形が出たら、マウスカーソルでコマンド入力エリアを選択した状態であることを確認し、続けてエンターキーを何回か押してみてください[2]。

長方形が書き加えられ、画面が少しずつ変化してゆくのがわかりますか？

ここでは、mondrian というコマンドが実行され、赤や青, 緑などの色のついた長方形が書き加えられます[3]。

さらに短い mond というコマンド[4]を用いると、長方形とともに、メッシュが現れます。ThinkingStech では、この縦横の線を 2 本ずつ選ぶことで、色をつける長方形の大きさや場所を決めているのです。

[2] ThinkingSketch にはヒストリという機能があります。一度コマンドを実行していたなら、文字を再入力せずにコマンド入力を行う領域に上矢印（↑）や下矢印（↓）キーを用いて過去の入力履歴を呼び出してエンターキーを押すことにより再実行を行うことができます。

[3] mondrian コマンドは ThinkingSketch 独自のプラグイン機能で、実現されています。画家 Piet Mondrian は緑を使わなかったようですが、そういうことにこだわらず、ここでは緑色の長方形も表示するようなプログラムにしました。

[4] mond は mondrian コマンドの説明のために用意されたコマンドです。

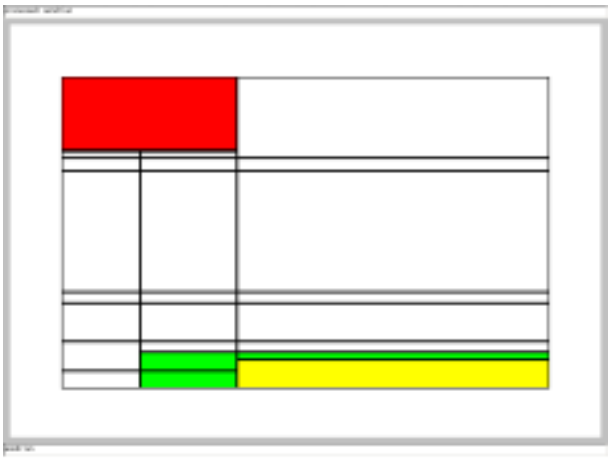

図 2.3: さらに描き加える

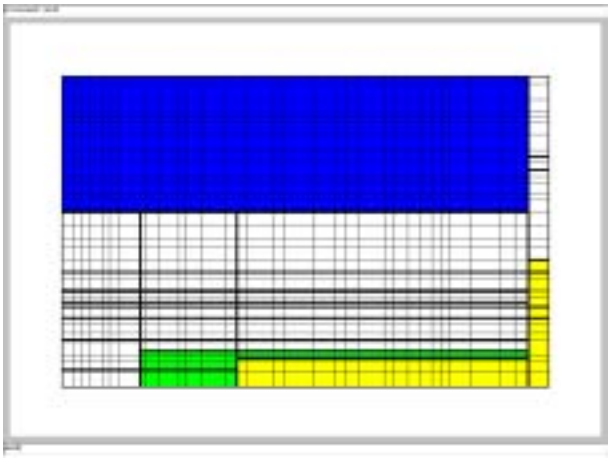

図 2.4: 補助線（メッシュ）を描く

2.4　画面のクリア

一度描いた画面をクリアしてみましょう。

```
clear
```

というコマンドをキーボードから入力します。画面が初期の状態に戻ります。

図 2.5: 画面のクリア直後

2.5　部品の読み込み

ThinkingStetch では、絵を描くのに利用する部品をオブジェクトとして保管することができます。

コマンド入力エリアに以下のようなコマンドを入力し、

```
import samples/birdfish.drw
```

を実行してみましょう。

この操作では、あらかじめ作成されファイルに保管されていたオブジェクト（絵の部品）が ThinkingSketch に読み込まれます[5]。

[5] ThinkingSketch が読み込めないファイルであった場合には単に無視され、ThinkingSketch には何もおこりません。

図 2.6: samples/birdfish.drw を読み込んだところ

2.6　部品庫への保存

画面上にいくつかのオブジェクトが存在することを確認してください。ファイルを読み込んだあとであれば、画面上にオブジェクトが見えます。

コマンド

```
storage
```

を実行します。このコマンドの実行で画面から図形は見えなくなります。

この操作で画面に表示されていたいくつかの図形が、個々の部品として「部品庫」に保管されます。

図 2.7: 部品庫に部品を移動

2.7　コマンド *stella の使用

部品庫においた部品を画面に配置することができます。

```
*stella
```

を実行してみましょう。

さきほど読み込まれた図形が大きさを変えて、ひとつ画面上に現れます。*stella コマンドは、部品庫にあった部品をランダムに選択した二本の垂直な線、二本の水平な線で囲まれた長方形の領域に収まるように変形して配置します。

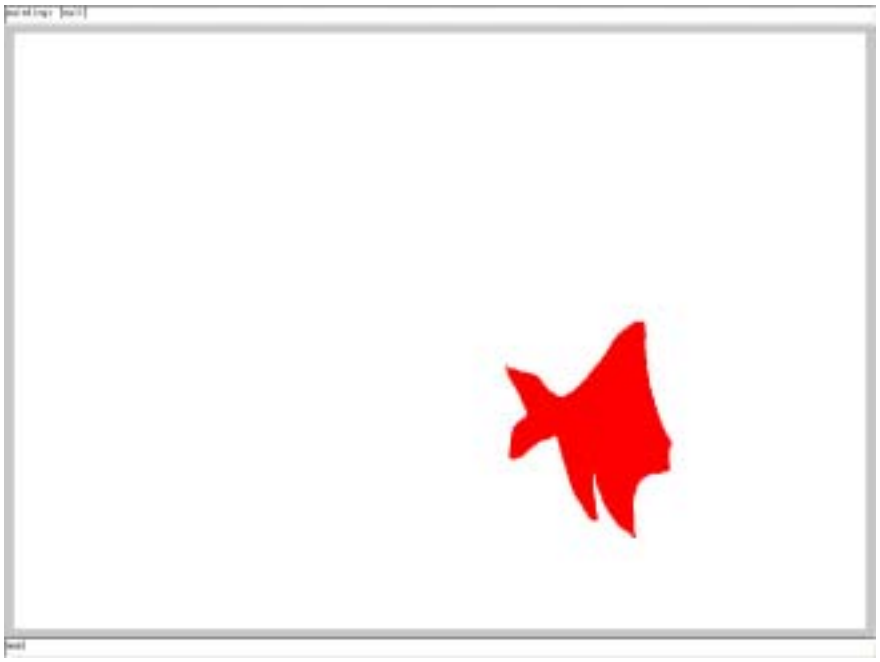

図 2.8: *stella コマンドで部品庫から部品を移動

2.8　部品庫から複数のオブジェクトをコピーする

* で始まる名前をもつコマンドは、一度の操作で部品庫から複数のオブジェクトを移動させるように指定できます。一度に何個のオブジェクトを移動させるかを repeat のパラメータとして指定します。たとえば、

```
repeat 5
```

を実行すると、それ以降 *stella で生成する図形は一度に 5 個作られることになります。

図 2.9: repeat のパラメータを設定

図 2.10: 再度 *stella を実行する

2.9　その他のコマンド、*pollock の使用

部品庫においた部品を画面に配置することができます。これに関連したコマンドはいくつかあります。

```
*stella
*pollock
*imai
*mattise
```

などです。コマンドをクリア (clear) を実行しながら、順に実行してみましょう。その違いがわかるはずです。

はじめに紹介した*stella はオブジェクトの高さや幅に変化がありました。*pollock は、オブジェクトの位置だけが乱数に基づいて配置され、サイズの変化はありません。

図 2.11: *stella の実行例

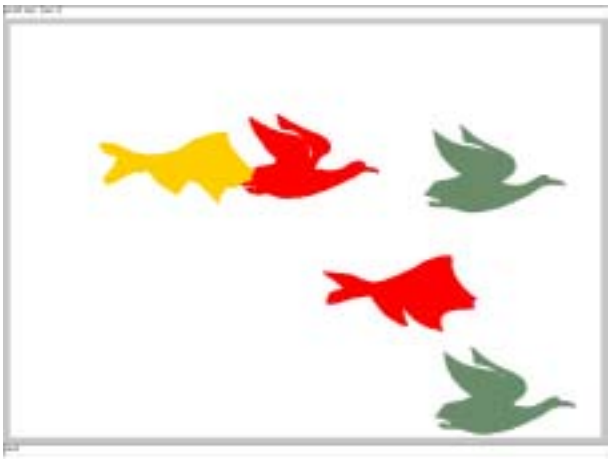

図 2.12: *pollock の実行例

2.10　*imai, *matisse の使用

```
*imai

*matisse
```

を実行してみました (図 2.13, 2.14) 。*imai は*stella よりさらに大胆な配置を行うようにみえます。これは変換する矩形の一辺を、枠の一辺上に配置するためです。*matisse は、配置する先の矩形の大きさを限定します。そのため派手さは感じられませんが繊細な印象を与えます。

図 2.13: *imai の実行例

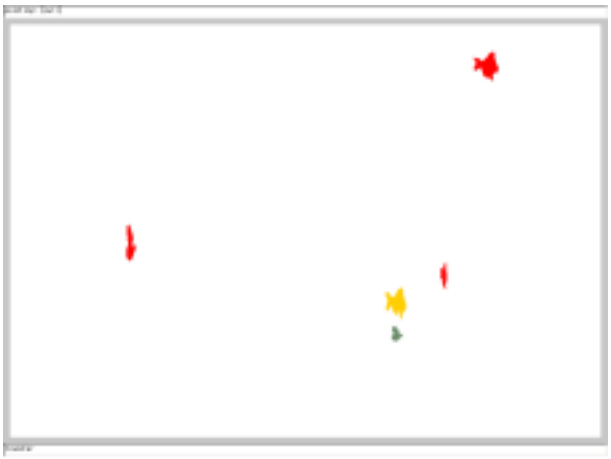

図 2.14: *matisse の実行例

2.11　*matisse の繰り返し使用

```
*matisse
```

を何度か実行してみました。さびしい感じの画面が、デザインされたパターンに変化していきます。

コマンド実行の結果は乱数によって変化するので、繰り返し使用や repeat の変数を変えることにより、画面が変化していくことを確認してみましょう。

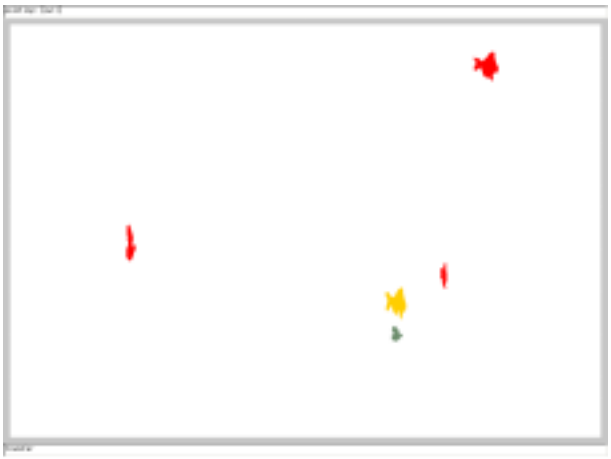

図 2.15: *matisse の一度だけの実行例

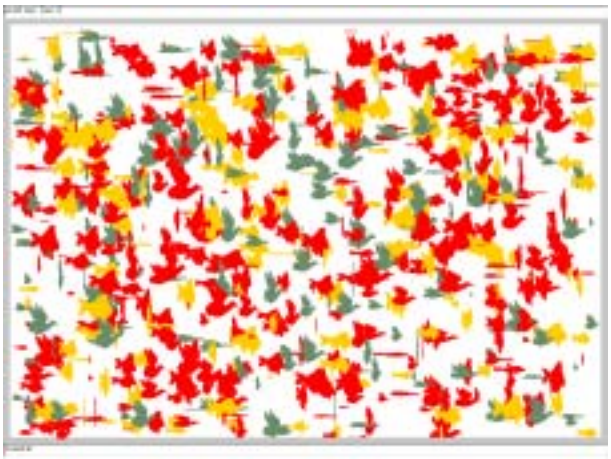

図 2.16: *matisse を繰り返し実行した例

2.12　mask の使用

これまで紹介したコマンドは画面全体にオブジェクトを配置していました。

オブジェクトを画面全体ではなく、特定の矩形に対して配置することが可能です。

```
mask 100 200 300 400
```

を*matisse 実行前に実行すると、2 点 (100, 200)、(300, 400) で囲まれた矩形の中にオブジェクトを配置します。

図 2.17: mask と*matisse の実行例

2.13　画面と部品庫のクリア

一度描いた画面と部品庫を同時に初期化します。

`reset`

というコマンドをキーボードから入力します。画面が初期の状態に戻ります[6]。

[6] reset コマンドに対し、clear コマンドは画面の初期化のみを行い、部品庫は変化しません。

図 2.18: `reset` で画面と部品庫を空にする

第3章　部品を作ろう

3.1　プリミティブ

前章では、あらかじめ作った部品を配置しました。

今度は、オリジナル部品を作ってみましょう。

まずは、

```
line 100 200 300 250
```

を実行してみましょう。端点の座標が(100, 200)、(300, 250)の直線からなる部品が画面に現れます。実は、部品をつくるのは、コマンドラインでの入力だけが唯一の方法ではありません。マウスのようなポインティングデバイスを使うことも可能です。右ボタンを押すとメニューが表示されます。

図 3.1: 直線をひく

図 3.2: ポップアップメニューからプリミティブを選択

3.2　壁紙の読み込み

壁紙として画像ファイルを読み込んで、ウィンドウの背面に表示する機能が提供されます。この画像をなぞり、図形を作り出すことができます[1]。まずは、

```
loadimage samples/photo.jpg
```

を実行してみましょう。画面上は何も変化しませんが、次に、

```
wall
```

を実行することにより、画面上にファイルから読み込まれた写真が現れます。

[1] 画像としては JPG, GIF の形式のファイルのみが読み込めます。読み込みが出来ない場合は上記の二つの形式に変更してください。

図 3.3: JPG イメージを読み込む

図 3.4: 壁紙として表示

3.3　トレースする

トレースに適したプリミティブはDrawCurvです。

```
curv
```

をキー入力するか、ポップアップメニューからcurvを選択します。次に、壁紙をなぞることで自由な形のプリミティブを作成することが可能です。

```
wall clear
```

を行うと壁紙が見えなくなるので、どのようなオブジェクトが出来たのかを確認することができます。

図 3.5: トレースの実行

図 3.6: 壁紙を除くと作り出した部品が見える

3.4　色をつける

オブジェクトに色をつけることができます。

```
color red
```

color コマンドを実行することにより、色を指定することができます[2]。色を指定したあとで、描くオブジェクトは指定の色になります。また、カラーパレットを使うことも可能です。ポップアップメニューの「color-palette」を選択することにより、標準のカラーパレットを表示することが可能です。

[2] black, blue, cyan, darkGray, green, lightGray, magenta, orange, pink, white, red, yellow の各色が指定できます。三原色の RGB を 255 段階の明るさで指定することも可能で、たとえば color 255 0 0 とすると赤が選択できます。

図 3.7: 色の変更

図 3.8: カラーパレット

3.5　壁紙からパレットを作る

壁紙から色を抽出してパレットを作ることが可能です。ポップアップメニューの「update-palette」を選択したあと、ポップアップメニューの「image-palette」を選択することにより、標準のカラーパレットを表示することが可能です。

この機能を用いて、古典の絵画をスキャナやデジカメから読み込み、巨匠の色使いを追体験することもできます。

図 3.9: image-palette の更新

図 3.10: 壁紙から作ったパレット

3.6　色以外の属性

fill 属性を指定することにより、オブジェクトの中を塗りつぶすか枠だけにするかの決定が行えます。

```
fill fill
```

または、

```
fill outline
```

のように指定します。

図 3.11: fill 属性の図形と outline 属性の図形

第4章　マクロを使おう

4.1　マクロとは

ThinkingSketch ではテキストファイルに書いたコマンドをまとめて実行する機能をマクロと呼びます。

こうしておくと1行ずつコマンドを入力する手間を省くことができます。自分の気に入った手順 (コマンド列) が決まったら、マクロにしておくと便利です。

コマンド行で実行したいコマンドをテキストエディタでまとめて書いて、ファイルに SAVE します一つのコマンドは1行にまとめて書きます。[1]。

マクロは、「メモ帳」など好みのテキストエディタで作成しておきます。右にあるのはこれから実行しようとする「マクロ howa1.mac」の内容をメモ帳でみたものです。

[1] マクロはどんな拡張子をつけても実行可能になっていますが、マクロであることがわかるように拡張子を“ .mac”とすることをお勧めします。

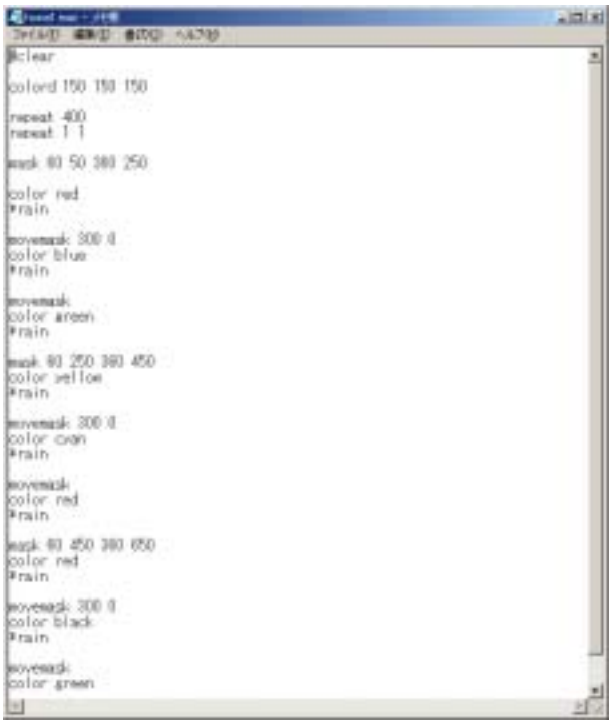

図 4.1: howa1.mac マクロをメモ帳で見る

4.2　マクロ howa1.mac

右にあるのが今回実行するマクロの全体です[2]。空行をも含めて、全体で 40 行あります。これから、このマクロについての解説を行います。

[2] プログラムの中の一行の先頭にある数字はマクロにかかれているものではなく、解説のために加えたものです。

```
 1: colord 150 150 150
 2:
 3: repeat 400
 4: mirror 1 1
 5:
 6: mask 60 50 360 250
 7: color red
 8: *rain
 9:
10: movemask 300 0
11: color blue
12: *rain
13:
14: movemask
15: color green
16: *rain
17:
18: mask 60 250 360 450
19: color yellow
20: *rain
21:
22: movemask 300 0
23: color cyan
24: *rain
25:
26: movemask
27: color red
28: *rain
29:
30: mask 60 450 360 650
31: color red
32: *rain
33:
34: movemask 300 0
35: color black
36: *rain
37:
38: movemask
39: color green
40: *rain
```

マクロ "samples/howa1.mac"

1 行目の colord は、繰り返しオブジェクトを生成したり、部品個からオブジェクトをコピーしたりする場合、色彩に変化を与えるためのものです[3]。

赤、緑、青の加色混合の 3 原色に対応する 3 つの数字を 0 から 255 までの値で指定します。数字が多くなるほど変化のばらつきが大きくなります。

colord <赤の変化量> <緑の変化量> <青の変化量>

3 行目は repeat により一度に生成するオブジェクトの数を 400 個に指定しています[4]。

4 行目は初期化手続きの一種です[5]。

[3] これらは、乱数によって色調に関連したパラメータに変化を与えることによって実現されます。

[4] 指定されているパラメータが 1 つであることに注意してください。

[5] 多くの場合は必要ありませんが、安定した動作のために使用しています。簡略化のために、ここでは説明を省略します。

```
1: colord 150 150 150
2:
3: repeat 400
4: mirror 1 1
5:
```

マクロ "samples/howa1.mac line 1-5"

6行目のmaskは、*stella、*pollock、*imai、*matisse、などで部品庫から部品をコピーする際にコピー先の大きさを矩形領域[6] の指定によって指定ができます。

```
mask <左> <上> <右> <下>
```

7行目のcolorは8行目の*rainコマンドで生成する図形の色を指定しています。redを指定することにより、赤を中心に描画を行うという指定となります。

8行目の*rainはデモンストレーション用のプラグインに含まれているコマンドで、左上方向から右下方向に直線を引きます。

10行目のmovemaskは現在指定されているマスクの矩形領域を平行移動させるためのものです。

```
movemask <左右方向の移動> <上下方向の移動>
```

11行目、12行目は今度は青系の色で直線を描くことを指定します。

14行目のmovemaskも現在指定されているマスクの矩形領域を平行移動させるためのものです[7]。

```
movemask
```

このあと、15行目、16行目は今度は緑系の色で直線を描くことを指定します。

[6] 矩形というのは「なが四角」ということです。

[7] パラメータとして移動量の指定がない場合は前回指定したパラメータがそのまま使われます。

```
 6: mask 60 50 360 250
 7: color red
 8: *rain
 9:
10: movemask 300 0
11: color blue
12: *rain
13:
14: movemask
15: color green
16: *rain
17:
```

マクロ "samples/howa1.mac line 6-17"

18-28 行目は表示する位置が異なりますが、6-17 行目のくり返しです。

これらの部分では 3 つの矩形領域への描画を行います。

```
18: mask 60 250 360 450
19: color yellow
20: *rain
21:
22: movemask 300 0
23: color cyan
24: *rain
25:
26: movemask
27: color red
28: *rain
```

マクロ "samples/howa1.mac line 18-28"

マクロの実行は exec または!コマンドを用います。コマンドの後にファイル名を書きます。

```
! samples/howa1.mac
```

右ページはコマンドの実行例です。

画面の中に矩形領域を設定し、中心となる色を指定します。描画時には色彩を変化させながら、400 本ずつ線を引くという作業を繰り返し 9 つの領域を塗りつぶします。

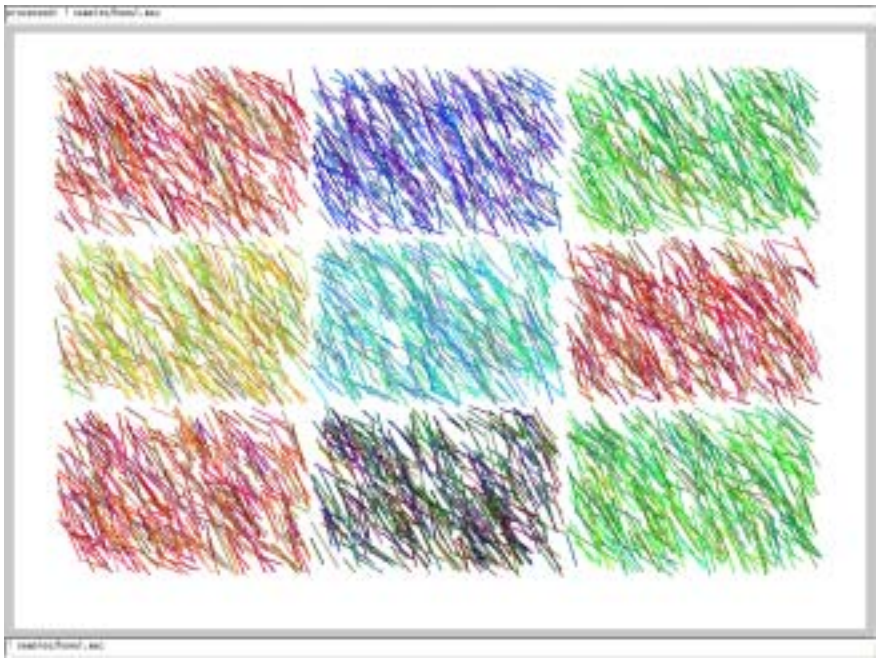

図 4.2: マクロ samples/howa1.mac の実行例

マクロの実行前に linewidth コマンドを用いて、描画される線の太さを変更します。

```
linewidth 10
```

この指定により、今後描かれる線の太さが 10 となります。

図 4.3: 線幅を変更する

再度、同一のマクロを実行してみます。

```
! samples/howa1.mac
```

実行結果が異なり、類似の図柄ながら画面の雰囲気も大きく変わるのがわかります。

このように場合によってはコマンドとマクロを交互に使って新しい図柄の検討を行うことが可能です。

このほかにも *rain を *boxer に変更したり、*bubble に変更したらどうなるかを調べてみると生成される図柄が変化します。

図 4.4: マクロ samples/howa1.mac の実行

4.3　マクロ igo.mac

乱数による配置をおこなっていても、規則性を感じさせる図形を生成することは可能です。

今回はその例の一つ samples/igo.mac を紹介します。このプログラムは 16 行で記述されています。

```
 1: color black
 2:
 3: wmax 50
 4: wmin 50
 5: hmax 50
 6: hmin 50
 7:
 8: gridw 50
 9: gridh 50
10:
11: repeat 25
12:
13: fill fill
14: *bubble
15: fill outline
16: *bubble
```

マクロ "samples/igo.mac"

1 行目は、色の指定です。黒を指定しています。

wmax、wmin、は*bubble [8] で生成されるオブジェクトの横幅の最大値、最小値を指定するためのコマンドです。最大、最小を 50 と指定しますので生成されるオブジェクトの幅は 50 となります。

hmax、hmin、は同様にオブジェクトの高さを指定することができます。オブジェクトの高さは 50 と指定したことになります。

gridw、gridx、でオブジェクトの配置をおこなう格子の大きさを指定することができます。ここでは x 軸方向の格子、y 軸方向の格子の大きさを 50 とします。

[8] *matisse、*rain、*boxer コマンドにも wmax、wmin、は影響を与えます。これは hmax, hmin, gridw, gridh についても同様です。

```
 1: color black
 2:
 3: wmax 50
 4: wmin 50
 5: hmax 50
 6: hmin 50
 7:
 8: gridw 50
 9: gridh 50
10:
```

マクロ "samples/igo.mac line 1-10"

*bubble は repeat で指定した数だけ、楕円を生成します。14 行目と 16 行目で実行が行われます。

11 行目では一度に生成するオブジェクトの数を 25 個とします。

13 行目、15 行目の fill コマンドはそれぞれ図形の中を塗りつぶす、塗りつぶさないという指定変更を行います。

このプログラムが実行されたあとは 50 個のオブジェクトが存在します。

```
11: repeat 25
12:
13: fill fill
14: *bubble
15: fill outline
16: *bubble
```

マクロ "samples/igo.mac" line 11-16

マクロを実行してみます。

```
! samples/igo.mac
```

右の図のように、一種のデザイン画が表示されました。

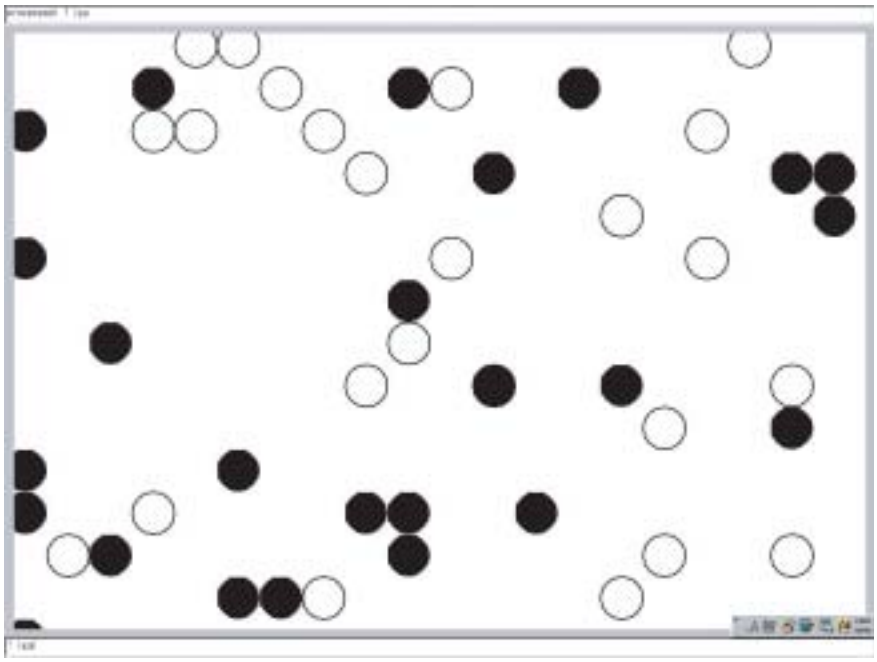

図 4.5: マクロ sampels/igo.mac の実行

4.4　マクロ yanagi_style.mac

　一定のテイストをイメージしてマクロを作った例を示します。

右の作品は、ThinkingSketch Unit のメンバーが過去に作成したものです。

図 4.6: ThinkingSketch Unit メンバーの作品

前ページの作品のイメージを参考にマクロを作成しました。

このマクロは samples/yanagi_style.mac として実行可能です。右の図は、作成した yanagi_style.mac の実行結果の例です。

図 4.7: yanagi_style.mac の実行結果

マクロの内容は右のとおりです。

第 1 行目から第 5 行目までは、画面に表示すべき部品を設定しています。

ここでは、もとのイメージをスキャンして JPG 形式としたファイル yanagi_style.jpg を読み込み、colormap コマンドによってこのイメージを構成している色彩からなる矩形を生成します。こうやって生成した図形は storage の実行によって部品庫に移されます。

7 行目は部品を 50 個コピーすることを指定し、8 行目ではマスク（対象となる描画領域）の指定を行います。9 行目以降はマスクをずらしては *stella コマンドの実行をおこなうという作業を繰り返しています。

```
 1: // select color
 2: loadimage samples/yanagi_style.jpg
 3: colormap
 4:
 5: storage
 6: //
 7: repeat 50
 8: mask 100  50 150 660
 9: *stella
10: movemask 60 0
11: *stella
12: movemask
13: *stella
14: movemask
15: *stella
16: movemask
17: *stella
18: movemask
19: *stella
20: movemask
21: *stella
22: movemask
23: *stella
24: movemask
25: *stella
26: movemask
27: *stella
28: movemask
29: *stella
30: movemask
31: *stella
32: movemask
33: *stella
```

マクロ "samples/yanagi_style.mac"

この段階ではまだ、最終形とは言えません。

これから先は好みに合わせてプログラムを変化させ、さらに、画面に深みをつけていくことになります。

また、終盤には画面を手で編集して微調整をおこなうことも有効です。

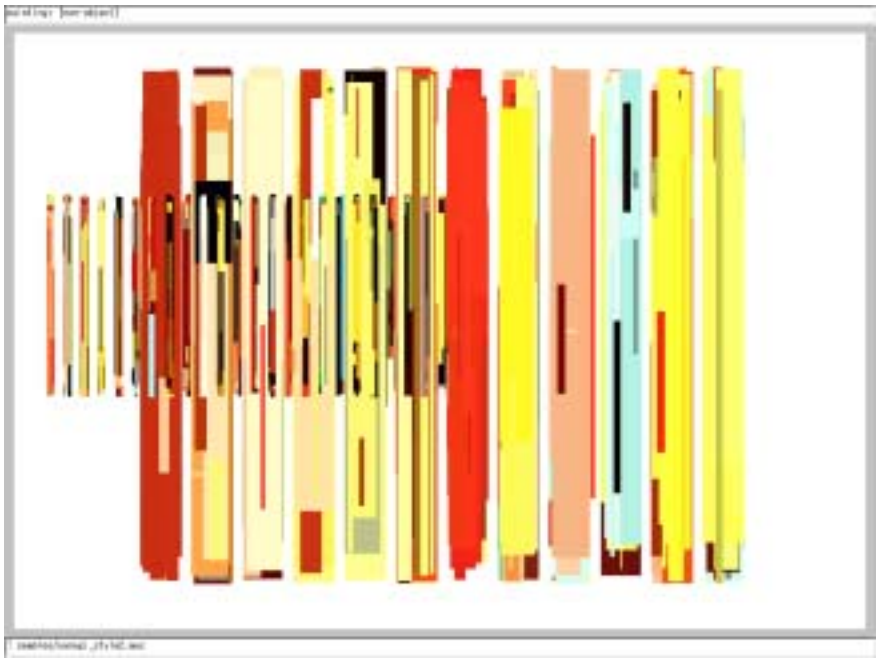

図 4.8: さらに変化をつけた画面

第5章　よいこの作品

ここでは、いくつか作品例を紹介します。これらの作品を参考にしながら、新たな作品作りを試みてみてください。

右の作品は、東京都在住の T さんが ThinkingSketch をはじめて用いた頃のものです。

単純ですが、力強いオブジェクトが画面に広がっています。

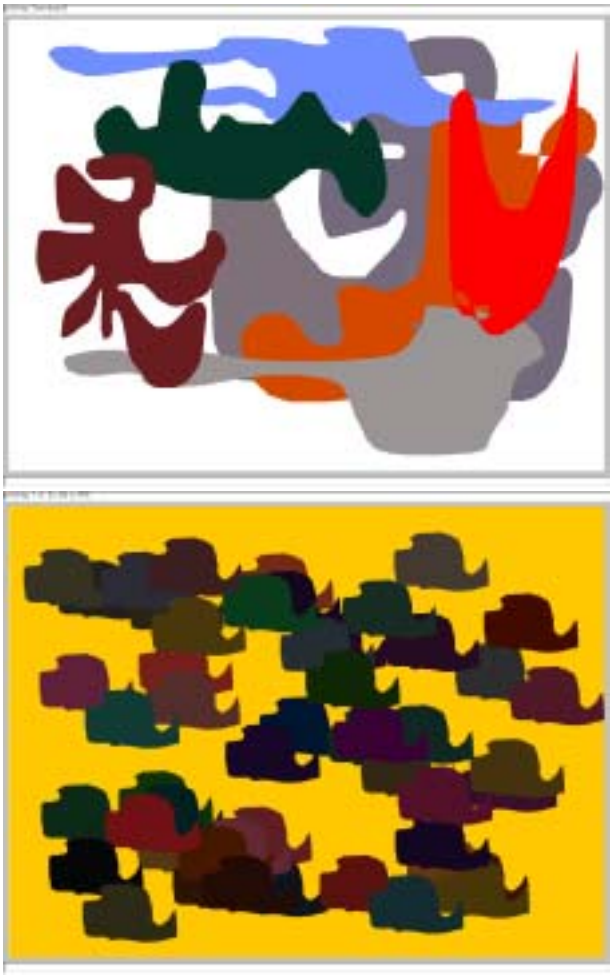

図 5.1: T さんの作品たち

次の作品は、「赤い森」です。作者は同じ T さん。昔、キャンバスにむかって描いた絵をイメージして ThinkingSketch を利用しました。

図 5.2: T さんの作品「赤い森」

右の作品は、matisse の絵をイメージした別の作者のものです。matisse の絵をイメージしたものです。マチス風の部品を作ったあと、*pollock コマンドで配置し、微調整をおこなったものです。

図 5.3: マチス風

春の景色をイメージしました。*imai と*pollock を使っています[1]。

[1] ここでは、一様にオブジェクトを配置するのではなく、一定の場所に配置を行うためのパラメータの指定を行っています。

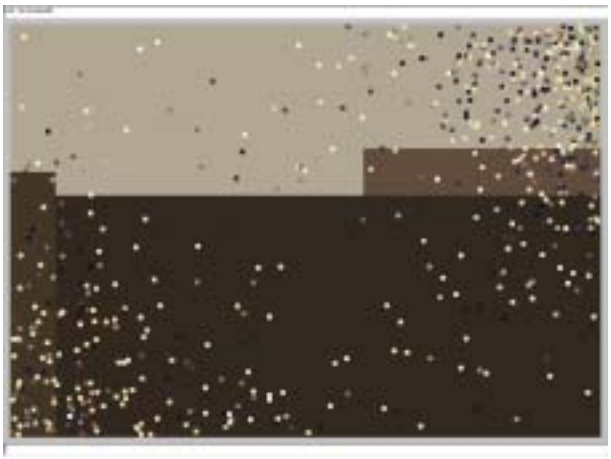

図 5.4: 春の景色

Update History

2002 年 4 月 30 日：repeat コマンドを変更。repeat コマンドが一つ引数を持つときと二つ引数を持つときで異なる意味を持たせていたが、あらたに mirror というコマンドをつくり `repeat <x-dup> <y-dup>` という機能を `mirror <x-dup> <y-dup>` という機能として実現した。

2002 年 8 月 26 日：Version 2.0 Alpha へとバージョンアップを行った。今回の大きな変更点は (1) 複数のページにデータを置けるようになり、互いのページを部品庫として指定できるようにしたこと。(2) 半透明のオブジェクトを指定できるようにしたこと。ページ間の移り変わりを行うときに、デゾルビングの効果をつけたこと。の 2 つの機能拡張を行ったことにある。