アートプログラミング

美馬 義亮　　木村 健一　　柳 英克*

概 要

昨年、芸術のためのプログラミングということで絵を半自動で生成するためのツール ThinkingSketch に関する報告を行った。これまで我々は、このプロジェクトを実行するにあたり、システムのコンセプトや実装自体、あるいは ThinkingSketch を用いた芸術教育について時間を割いてきたが、これらの作業が一段落を迎え、アート作品を定義するためのプログラミング言語について考察をはじめた。どんなユーザでもプログラムを生成する「一本指プログラミング」の可能性をあわせもつ、お手軽プログラミングという意味でリソースコンシャスなプログラミング、「アートプログラミング」の考え方について、そのアイデアと言語設計の途中経過について紹介する。

1　はじめに

ThinkingSketch [1] は、ユーザが素描の訓練を十分うけていない場合でも、（作画ルールやその適用順序を指定することにより）ある程度自分の意志を反映するように画風をコントロールしながら、半自動で（乱数で生み出された位置情報に従って）高速に絵を作成することができるというシステムである。このツールはすでにインターネット上からダウンロードし利用することが可能である [2]。このツールを使うことにより生成するルールをある程度制御しながら、大量の絵を描くことにより、アートセンスを強化させることが可能である。

絵画教育の支援が本システム開発の発端ではあるが、将来にわたる実用的な応用としては、クリエータが自己の絵画の作風を形成するための実験プラットフォームとして用いたり、テイストの同じデザインを大量に作成するなどの用途も考慮している。このシステム上の描画要素の作成、ならびに画像作成に対しては、(メニュー選択を含む) ポインティング・デバイスによる直接操作が定義されている。また、これらの操作に対応したテキストによるコマンドの実行が可能である。

これらのテキストによる複数のコマンドは、同一順序で再実行するためのインタプリタを備えており、操作マクロプログラミングが可能である。ThinkingSketch 上で一定のスタイルに従った「絵」を発生させるためのこれらのコマンドの集まりを、「アートゲノム」と呼び、さらに、このような記述様式をプログラミング言語として `SketchScript` と呼ぶことにした。`SketchScript` はプログラミングの経験を持たないユーザに対しても ThinkingSketch が提供できる機能の多くを自然な形で提供できることを目標として設計を行った。ここでは現状における言語 `SketchScript` の設計におけるコンセプトに関する報告を行う。

*公立はこだて未来大学 システム情報科学部 情報アーキテクチャ学科

2 背景と目的

2.1 要件

上で述べたように、SketchScript の設計にあたっては、それまでにプログラミング言語を記述したことのないユーザであっても、簡単にそれらを利用できることを目標としている。また、この言語で記述した「アートゲノム」はインターネット上で公開したり、自由に結合して新しい効果を生むことを想定した。いまのところは、テキストエディタ程度の利用が前提なのだが将来は指一本（ポインティングデバイスとボタン操作あるいはカーソルキーとエンターキー）でプログラムの生成を行うことを可能にしたいとも考える。これらの要求をレベルごとに分けて記述すると以下のようになる。

1. 基本的な要求としては、
    - ポインティングデバイスで候補を選択する程度に、具体的な絞込みを行う操作により言語の生成、記述が可能なこと。
    - くり返しの操作を、自然な形で表現できること。
    - 表現や基本概念が簡単で、できるだけ多くのユーザ層に理解が可能になること。
2. ThinkingSketch というソフトウエアがもつ特殊な要求として、
    - プログラムのソースコードの並べ替えや、混在を可能にすること。すなわち、プログラムの結合や順序の変更などが自由な形で行えること。
    - プリミティブや配置規則などを拡張することが可能なこと。

以上の要求に応えるものとして、言語 SketchScript の設計を行うことにした。

2.2 関連研究

広いユーザ層にわたるプログラミングを可能にする言語となることを設計された言語としては LEGO が代表的なものであろう。二次元空間に図形を描くためのプログラムを、身体的な動作のメタファでとらえ、タートルの動きになぞらえて、「ペンの上下」、「前進」と「方向転換」という単純な操作によって実現している。また、シンタックスなどは比較的簡単である。最近現れた類似のアプローチとしては、DBN[3] が存在する。ここで設計されているプログラムは、汎用言語というより、限られた空間での描画を中心的なターゲットとして設計されたものである。そのデザインの単純さによって言語の理解・習得を容易にしていることやターゲットの世界の単純さに比較して、それのもつ表現力の大きさがこのシステムの存在価値を高めている。

これらの言語は、小学生、あるいはデザイナーなどのいわばコンピュータ利用の非専門家の利用を意識して扱いやすい設計になるような努力をしている。しかしながら、（アセンブリプログラミングを抽象化することによって与えられた）既存のプログラミング言語で用いられる、条件分岐、変数、ブロック構造や関数呼び出しの概念の理解が要求される。小学生の多くが利用実績をもつシステムとはいえ、これらの概念の理解を前提としないと、くり返し構造を含むプログラムを書くことが困難であるという点は改善したい点である。

SketchScript においては、変数や条件分岐のなどを明示的に与えない形でくり返しを実現できる、あるいは制御構文を意識しないで記述できる、「リソースコンシャスな」設計思想をもつプログラミング言語の試みを行なうことにした。

3 システムの前提と言語の設計方針

3.1 ThinkingSketch のデータモデル

ThinkingSketch はオブジェクトベースの図形編集プログラムであり、Java の 2 次元のグラフィックライブラリを用いて表示されている。基本となるデータ構造は個々の図形の表示に必要な情報を保持できるものでなければならない。このグラフィックライブラリ上で描画時に指定すべき色、線幅、透明度などの属性をオブジェクトが持っている[1]。

また、個々のオブジェクトは、図形の形を決めるパラメータとして、1 つ以上のコントロールポイントを持つ。これらのコントロールポイントによって表示される位置や大きさ、形態が変化する。コントロールポイントの数は矩形や楕円のようにオブジェクトのプリミティブの種類ごとにその個数が決まっているものと、折れ線のようにプリミティブの種類が決まってもその個数が不定のものがある。

図形の実態は複数のコントロールポイントの集合と図形のタイプ、および図形の属性値である。個々のコントロールポイントは順序をもった x 座標、y 座標をもつ単なる点の集まりである。

- 図形タイプ
  - 図形タイプ依存の属性（タイプフェィス、文字列やイメージ）が付加的な情報として加わる場合がある。
- 図形属性
  - 色や線幅、透明度など
- コントロールポイント
  - 点の並び

これらの属性値が決定されると、表示可能なオブジェクトになる。

[1]特別な例としては、テキスト、イメージなどのグラフィックオブジェクトにおいては、オブジェクトのクラスに依存した属性として、テキストの文字列やフォントの種類を決定するための属性あるいは対応するイメージのファイル名などの属性が存在する。

3.2 環境の設定

ThinkingSketch は、デフォルトの属性として一組の図形属性を保持していおり、これがオブジェクト生成時の属性テンプレートとして使われる。文字列やイメージのように拡張された属性を必要とするものに対してもこれらのテンプレートを設定し、オブジェクト生成時にはそれらの属性のコピー（あるいはそれらの参照）がオブジェクトに割り当てられる。

3.3 オブジェクトの生成

オブジェクトの生成については、GUI で行われる手順と同等の指定を行う。システムはプロトタイプとなるオブジェクト属性の組（カレント属性）を 1 つ保持しており、ユーザはカラーパレットやツールボックスで現状の値を確認したり、再設定することが可能である。このプロトタイプとなる（色や線幅などの）オブジェクトの属性値は言語から見る方法は提供せず、ただ設定のみが可能になっている。

オブジェクトを生成するときは、オブジェクトのプリミティブ名とコントロールポイント（の並び）を与える。属性に関してはシステムが保持しているプロトタイプを用いてオブジェクトが生成される。

3.4 オブジェクトの生成例

プログラムの中で、次に生成されるオブジェクトとして特定の色を指定したいときは、オブジェクト生成直前にカレントの色属性を設定する。たとえば、青い矩形を枠として書いた後、その中に

カレントディレクトリの「photo.jpg」という名のJPG フォーマットのイメージオブジェクトを表示するためには以下のような指定を行う。

```
// デフォルトの色属性を青 ('blue') にする
color blue
// 青い矩形が生成される
rect 80 180 180 260
// デフォルト属性としてのイメージを指定
loadimage photo.jpg
// image の作成とコントロールポイントの付与
image 100 200 160 240
```

3.5 操作コマンド

操作系のコマンドは、ThinkingSketch 上で定義されたオブジェクトを再配置するための手段を指定するためのものである。オブジェクトの再配置とは、部品庫と呼んでいる場所からパーツとなるオブジェクトを取り出し、コマンドが持つ戦略により配置を行うというものである。

コマンド体系（あるいは言語）の設計の方針として、以下の二点を特徴としている。

- できるだけプログラミング言語にまつわる変数や制御構造といった概念を言語の前面に出さないようにする。たとえば、1 文の記述は実行の単位である 1 行である。ブロック構造に似たものが定義できたとしてもこの原則は守っている。
- 解釈ができない場合には善意の判断を行い、実行が中断されるようなことがないものとした。すなわち、どのようなプログラムを与えてもエラーによって実行を停止することがない。

上記の 2 点の特徴をもたせることにより、複数の文からなるアートゲノムを無作為で結合あるいは、順序を変更してもプログラムの実行結果が得られることになる。

3.6 くり返しの制御

オブジェクトの配置は、絵作りの中でくり返しの要素をもたせることにより、さらに複雑な絵を生成することが可能である。

[2]

絵の配置規則は Java で記述し、これらの配置規則を（場合によっては動的）に読み込み、拡張コマンドとして呼び出すことにより実行される。これらのコマンドは以下の例では'stella'、'matisse'、'pollock'、'imai'、などとして呼び出されているものであり、これらの自動再配置によって、絵柄が生成されているように見せるためのものである。一度の呼び出しで、一つのオブジェクトをコピーし、決められた戦略で図形のサイズと位置や属性の変更や決定を行う機能をもっている。

現在のところ制御構造にあたるものはくり返しのみが必要であると考え、最初にくり返しの数を指定し、配置戦略の演算子として'*' をつけることによりくり返しの段数を指定するものとしている。('*' がないものは 1 回だけの実行。）'*' が同数以上ついている行が隣あっていると、自動的にブロック化されるものとしている。現在のところくり返しのループの中で制御変数を得るような機構は定義していない。

```
repeat 2 3 4
***stella
matisse
*pollock
**imai
*pollock
```

上記の例では、最初の宣言は'*' が 1 つついている場合はくり返しが 2 回、2 つついている場合

[2]絵の配置規則を SketchScript で記述し、配置位置を詳細に決定する方法はとっていない。実行速度を期待できないと考え、積極的に SketchScript の記述力を拡大することはあまり検討していない。

はくり返しが (2 × 3 で)6 回、3 つついている場合は (2 × 3 × 4 で)24 回という宣言となる。上の例では、

stella コマンドが 24 回実行
matisse コマンドが 1 回実行
（pollock 1 回、imai 3 回, pollock 1 回という順に実行）が 2 回実行

というくり返し操作が行われることになる。

このような記述は構文を単純にするという効果を持つ。ただし、多重のループを記述する場合に、コマンドライン入力を入力するたびに、一行ごとの文の解釈を逐次実行しようとすると難しい問題が生じる。ループが閉じるのは最後の（より厳密に言えば、同一レベルの）'*' で始まる行が終わった場所である。しかし、ループがいつ閉じるかは次の行の入力が終わるまでインタプリタにはわからないわけだから、実行のタイミングにはずれが生じてしまう。

というわけで、逐次実行とセマンティクスが変わらないくり返し、指定用のプレフィックス'+' の採用を検討している。

```
repeat 2 3 4
+++stella
matisse
+pollock
++imai
+pollock
```

最初の例と同様に、最初の宣言は'+' が 1 つついている場合はくり返しが 2 回、2 つついている場合はくり返しが 6 回、3 つついている場合は 24 回となり、

stella コマンドが 24 回実行
matisse コマンドが 1 回実行
pollock 2 回実行
imai 6 回実行
pollock 2 回実行

というくり返し操作が行われるわけである。実際上は）二つのオペレータ'*' と'+' の解釈を使い分けるような仕掛けを用意するとよいのかもしれない。

3.7 関数呼び出し

ThinkingSketch 上では Java の拡張で配置ルールを導入できるようになっており、新しい名前をもった操作の追加は、ユーザにとってそれほど困難な概念ではないと考える。

したがって、関数呼び出しに相当する概念を用意している。ここでは、ファイル単位で保存された一連のコマンド列を呼び出し、そのファイルにあるプログラムを順次実行を行うことができる。明示的なパラメータの引渡しは行っていない。

このような構造については、行単位でのプログラムの並べ替えなどに対応することを考えると言語の設計をこれ以上変更を行える可能性は大きいものではないが、環境を変化させることによりより柔軟な対応が可能になる可能性もあると考える。

4 現状と今後の課題

あんまり、努力せず、しかしある程度の自動化を記述するためのプログラミングスタイルを探っている。限定された世界とはいえ、（プログラム言語というほど限定されたものでもないと思うので...）プログラミングシステムの設計というテーマのなかでも大きなトピックに大胆に挑戦しているのではないかと思いつつ今後の見通しについて述べる。

現在、ThinkingSketch 上の操作は、操作履歴が保存されているのでその履歴をファイルに落として、プログラムとして再実行することが可能であり、ある意味「お手軽な」プログラミング環境を提供している。また、正直いうと、くり返しの実装は一重ループに相当する機能だけである。さ

らに、くり返しや関数呼び出しのような言語の拡張めいたことが実現できるようになるとより柔軟性に富んだ環境になり、本システムで生成させる絵柄にも影響をあたえることになると考える。今後、いろいろな拡張をやってみると、さらに面白いことがみつかってしまうかもしれない。

言語のさらなる、拡張を考えた場合に、現在の課題を列挙すると以下のようになる。

- 条件分岐、変数の概念を自然な形で表現できること。
- 算術、文字列演算のサポート。

一般的なプログラミング言語が備えている、条件分岐、関数呼び出し、変数などに対応する操作が記述できない。条件分岐については、オブジェクトの再配置動作を修飾するような機構を取り入れることにより実現が可能であると考えており検討を続けている。変数の概念は本質的な問題でもあるが何らかの解決は見出せると楽観的に考えている、言語のモデルを複雑にすることの無いような解決策を求めて慎重に検討を続けている。

5 謝辞

この研究におけるソフトウエア ThinkingSketch の開発に対しては、情報処理振興協会の未踏ソフトウエア創造事業として平成 12 年度、13 年度にわたり支援を受けた。合わせて本事業のプロジェクトマネージャとしてかかわっていただいた電気通信大学 竹内郁雄氏には，研究の方向性に関してアドバイスをいただいた。

参考文献

[1] 芸術分野における内省ツール, 美馬　義亮, 木村　健一, 第 18 回日本認知科学会発表論文集, pp.246-247, 2001

[2] http://www.sketch.jp/

[3] John Maeda MAEDA@MEDIA, 前田ジョン, デジタローグ, 2000