アートプログラミング

美馬　義亮
木村　健一
柳　英克

アートプログラミングとは何？

- 絵を作成するためのスクリプト
  - 動作は乱数でパラメタライズするため非決定的
  - 同じスクリプトで同系統の絵を発生させる
- なるべくプログラミングを感じさせないものとしたい
  - うるさくないプログラミング
  - 共有できるプログラム資産やデータ

どこがリソースコンシャスか

- リソースとしてのプログラミングスキル
  - 学ばずにできるプログラミング
- リソースとしてのプログラム資産
  - プログラムをデータのように扱う
  - 過去のプログラムをしゃぶり尽くす
- リソースとしての入力デバイス
  - キーボードの要らないプログラミングにも挑戦してみたい

復習：ThinkingSketchとは

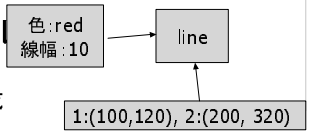
- デザイン生成のためのツール
  - 「部品」を用意する
  - 「部品」はプリミティブ、トレース（なぞり）など
  - 部品を乱数とルールにもとづいて再配置
- コマンドライン、GUIの同時利用可の二系統のUI
- 高速で何パターンものデザインを生成
  - 良い生成ルールとはなにかをみつける
  - 大量のパターンからよいものを拾い出す
    - 計算量、データ量に関しては富豪的

現在http://www.sketch.jp/ で公開されている

ThinkingSketch上の作業の流れ

- 種データの生成
  - 描画バッファでデータを記述
  - 既存ファイルからの読込も可能
- 配置手続きの呼び出し
  - 配置手続きはコマンドラインから
    - コマンドは行単位で与える
  - 複数行をならべて逐次実行
    - ファイルにあらかじめ記述
- 生成されたデータを評価

データのモデル



- オブジェクトの構成
  - 複数の制御点
    - Point型を要素とするVector
  - 属性
    - 色、線幅、透明度など
  - 拡張属性
    - イメージオブジェクトの場合対応するイメージファイル名
    - テキストオブジェクトの場合はフォント情報、テキストデータ
  - データの型
    - 抽象クラス DrawObject のサブクラスのクラス名

オブジェクト生成の手続き

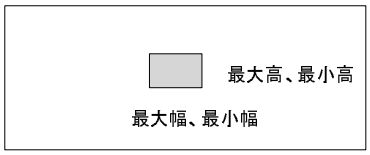
- オブジェクトのタイプと制御点情報を付与
  - line 100 120 200 320
  - image 200 300 400 600
- 属性はカレントの設定値を利用
  - color red
  - alpha 0.5
  - 選択されたオブジェクトに対しても変更適用
- 拡張属性も同様
  - text Hello World

配置ルール

- Javaレベルで記述されたライブラリ
  - 部品庫からのオブジェクトの選択
  - 制御点パラメータの変更
  - カレントの属性値の変更
  - 配置のためのパラメータをもつ
- 動的にリンク、利用可能
- リファレンス的な実装が提供されている
  - pollock, stella, imai, matisseなどと呼ばれるルール

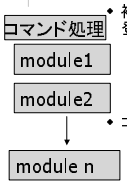
配置ルールの例

- matisse
  - オブジェクトのサイズを乱数により変化
    - サイズの上限、下限を設定可能
  - 表示位置を乱数により決定



配置ルールの拡張

- コマンド（解釈のためのJavaClass）の登録
  - addCmd <classname>
- コマンドインタプリタの構造
  - コマンド行の解釈
    - 複数のコマンド実行モジュールが、コマンド行の内容とともに、登録順に事項される
      - 同一コマンドが複数の実行モジュールで実行される可能性あり
      - 継続実行可能、あるいは実行中断の指示が各モジュールの実行後に返る
      - ただし、必要なAPIは未公開⇒いずれ公開(?)
    - コマンドが実行されたことは呼び出し側に伝えられる



Scriptの一括実行

- 関数呼び出しに対応
  - 引数の明示的な渡しはない
- exec <filename>
  (あるいは！<filename>)
  ファイル中のテキストを一行ずつ解釈

言語としてのSketchScript

- 制御構造
  - くり返し
    - くり返し回数の設定
    - くり返しを表す演算子
  - 条件実行　実行モジュールに依存
- エラー処理
  - 理解できない処理は無視する
- データ構造
  - なし、実行モジュールが勝手に処理

くり返しの指定

- 多重のレベルでくり返しを指定可能
  - repeat 10 20 30
- 多重度を演算子で指定
  - *stella
    - 10回のstella実行
  - **stella
    - 200(10×20)回のstella実行
- 明示的なブロック構造は行わない

ブロック化　1

- 同一のレベルをまとめて実行
  - *stella
    - 10回のstella実行
  - **stella
    - 200(10×20)回のstella実行
- 明示的なブロック構造は行わない，同一レベルのくり返しはまとめておこなう
  - *stella
  - **stella
  - *pollock

ブロック化　2

- 明示的なブロック構造は行わない，1行ごとに実行したい場合
  - +stella
  - ++stella
  - +pollock

条件による実行

- 配置ルールの一部
  - 既存オブジェクトの上に配置する/しないの指定
    - オブジェクトに重ねたい場合(mode10)
    - オブジェクトに重ならないようにしたい場合(mode01)
    - 下のオブジェクトに関係ない配置の場合(mode11)
  - modeを指定した後
    - stellaを実行した場合stellaの実行中に上記の解釈が行われる
    - 実行順序の制御が全体で行われるわけではない

プログラムの再利用

- 「化学変化」を起こしうるプログラミングパラダイム
  - エラー耐性の強いプログラミングパラダイム
    - プログラムを行ごとにシャッフルしてもプログラムとして意味をもつ
- 「化学変化」の効用
  - 表現の多様化
- 共有する「アートゲノム」の積極的利用

プログラムの変換

- シャッフル
  - 行単位での入れ替え
- 変異
  - パラメータの置き換え
    - 数字パラメータの変更
  - プリミティブ名の置き換え
- 結合
  - 二つのプログラムの結合

シャッフルの例

color red	oval 200 220 300 340
line 10 20 40 50	color red
color blue	oval 250 270 320 340
rect 320 340 400 450 →	line 10 20 40 50
oval 200 220 300 340	color blue
color green	color green
oval 250 270 320 340	rect 320 340 400 450

パラメータの変更

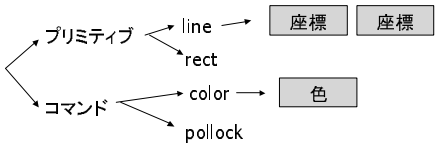
- 数値をもつパラメータを変化させる
  - 数の加減乗除
    - 揺らぎをあたえる
    - 上限、下限の値などはできれば守る
- プリミティブの変更
  - プリミティブ表にあるものを変更
    - eg. line ⇒ rect など

考慮点

- 画面構成に大きな影響を与えるコマンド群
  - clear, reset など
    - 画面上のオブジェクトを消す命令
  - next, prev, visitなど
    - ページ間の移動
  - これらのコマンドはコメントアウト
    - 利用者が編集時に生かす
    - ゴミ出し用ソフト

キーボードレスプログラミング

- コマンドとパラメータを木構造化して整理
  - 順序化することが容易
  - コマンドを矢印キーだけで順次選択可能
  - パラメータは制約を満たすように生成



まとめ:
言語(?)デザインで考慮した点

- エンドユーザの作業が進みやすいことが肝心(既存のプログラミング言語のスタイルには固執しないこと)
- 変数、関数呼び出し、条件分岐などの概念の使用は避ける(でも、くり返しは重要)
- エラーが起こってもうるさく追求しない(警告しても、対応できないのなら、うるさく言わない)
- 発見的プログラミングが可能なこと(偶然の恵みを最大限に利用するという考え方)