Vol.2012-CE-116 No.17
2012/10/14

Processing 上で古典的コンソールアプリケーションも開発可能な Crowbar+Tomahawk の紹介

白井　達也[†1]

プログラミング言語 Processing 上で古典的なコンソール対話型アプリケーションの開発を可能とするフレームワーク Crowbar を開発した．Processing はグラフィックスを用いてプログラミング技法を習得することを目指して開発された入門者向けのプログラミング言語であるためテキスト入出力は苦手だが，Crowbar を用いれば旧来の BASIC や C 言語で開発したコンソール入出力のみの単純な数値解析プログラムを比較的容易に移植可能である．Crowbar にはマルチビューポート対応の多機能なグラフィックスライブラリ Tomahawk も搭載した．本稿では Crowbar および Tomahawk の機能と動作原理を解説する．Crowbar + Tomahawk を用いれば低学年から高学年まで一つのプログラミング言語で情報教育が行える．さらに，プログラミングから遠ざかっていた教員にグラフィカルな e ラーニングコンテンツ開発の手段を提供する．

Introduce of Crowbar + Tomahawk Which Enables the Creation of Applications by Classical Interactive Console Programming Style on Programming Environment Processing

TATSUYA SHIRAI[†1]

I have developed a software framework named as Crowbar which enables the creation of applications by the classical interactive console programming style on programming environment ‘Processing’. Crowbar makes it possible to port a hoard of applications which use only console input and output, such as program for numerical analysis coded by FORTRAN, BASIC, C language and so on, to programming environment 'Processing' easily. Crowbar also includes high-functional graphics library Tomahawk which can treat multi-viewport feature. In this paper, I had explained concerning functions and mechanism of Crowbar + Tomahawk. By using Crowbar and Tomahawk, we can teach the information education by single programming language from the lower class to the higher class in school. Furthermore It provides a development environment for creating graphical e-learning contents for teachers who lost interest in programming.

1. はじめに

高等学校学生指導要領「情報」では「アルゴリズムとプログラム」で，アルゴリズムの基礎，プログラミングの基礎，数値計算の基礎，データの型と構造，アルゴリズム応用の５項目を教えることを定めている[1]. 使用するプログラミング言語は「情報システム実習」で使用する言語も含めて各校で学校や生徒の実態に応じて選定して構わず，“この科目のねらいがプログラム言語の規則の習得ではなく，論理的な思考力を身に付けることにある”とも注記している．平成２４年度より中学校技術の新指導要領においても「プログラムによる計測と制御」が必履修に指定され，大学入試センターの入試科目「情報関係基礎」では独自の手順記述言語 DNCL を用いた問題が出題されるなど，一般校においても情報教育における実践的なプログラミング演習の必要性が高まっている．高校生以下の学生にプログラミングの初歩を学ばせるための教育用プログラミング言語として，DNCL に準拠した xDNCL 言語の実行環境 PEN（Programming Environment for Novice）[2]やタートル（亀）と呼ぶカーソルを操作して図形を描く機能を中心としてオブジェクト指向によるプログラミングを比較的容易に学習できるドリトル[3]などが開発され，徐々に利用が広がっている．PEN やドリトルは，初学者が数値計算とアルゴリズムの基礎を集中して習得できるように日本語による制御文を採用するなどの工夫が施されている．工業系に特化した工業高校，高専などでは，教育用言語ではなく，FORTRAN, BASIC, Pascal, C / C++, Java, Ruby などの実用的なプログラミング言語を導入時に用いる傾向が強い．メカトロ教育の基礎を学ぶためにアセンブリ言語や LEGO マインドストームを低学年時から併用する場合も多い．雛型を用意して本格的な Windows アプリケーションを作成する学校もあるが，プログラミング初心者にとってイベント駆動型アプリケーションの思考スタイルは単純な数値解析プログラムを開発するには飛躍が大き過ぎ，アルゴリズムの学習よりもアプリケーション開発スキルの習得に多くの時間と手間を要する問題がある．大学の教養課程や一般高校では，研究上のデータ解析の基本的なスキルの習得や就職後の実益を狙って Excel および VBA（Visual Basic for Applications）マクロでプログラミング実習を行う事例も多い．Web プログラミングへの発展を睨み，実行結果が Web ブラウザで直ぐ確認できる JavaScript や，実行環境は Internet Explorer に限定されるが Visual Basic Script を用いる事例もある．

†1 鈴鹿工業高等専門学校　機械工学科
Suzuka National College of Technology, Mechanical Eng. Dept.

Vol.2012-CE-116 No.17
2012/10/14

ユーザインタフェースに GUI（Graphical User Interface）を用いるコンピュータ端末が大半となった近年，図１に示すようなコンソール内で実行する古典的なプログラム開発環境は多くの学生にとって馴染みが薄いものとなってしまった. しかし ウィンドウやダイアログボックスを開いたり，グラフィックスを描画することが中心の GUI アプリケーションの開発に比べて約束事が少ないことから，Windows / Linux 環境上でもコマンドプロンプト上で標準入出力を用いるプログラムの作成スタイルを堅持する学校も多い.

以上のようにコンピュータ環境の劇的な変化に合わせて演習に用いることができるプログラミング言語と開発スタイルの選択肢が年々，幅広くなっていく一方で，各言語には一長一短があるため教育現場ではどのプログラミング言語を選択するべきか判断に悩んでいる面もある．鈴鹿工業高等専門学校ではコマンドライン版 C 言語や統合開発環境を持つ N88 互換 BASIC for Windows,（仮称）十進 BASIC をプログラミングの導入用に用いてきたが，２０１２年度より Processing を用いることとした. Processing はグラフィックスの描画を通してプログラミング技法を学ぶことができる入門者向きのプログラミング言語である．近年，多数の入門書が出版され，多くの教育機関で用いられるようになってきた[4]．工学系の基礎知識が不足する文系専攻学生によるメディアアート作品の制作[5]やオープンソースハードウェアプラットホーム Arduino と組み合わせたシステム開発[6]，C 言語と文法の類似性が高い利点を生かして C 言語習得のための橋渡しとして用いる事例[7]など，幅広い分野で導入事例が報告されている．入門者向けでありながら，より実用的なプログラミング言語習得への橋渡しも可能な Processing だが，数値解析の基礎であるデータの入出力がコンソール経由では行なえない致命的な弱点がある．筆者は，BASIC や C 言語におけるキーボードによる入力と単純な文字列の出力による古典的なコンソール対話型プログラムと同様のシンプルなプログラミング作法で Processing のアプリケーションを作成可能とするフレームワーク Crowbar + Tomahawk を開発した．本発表では Crowbar + Tomahawk の特徴，機能，動作原理を説明する．

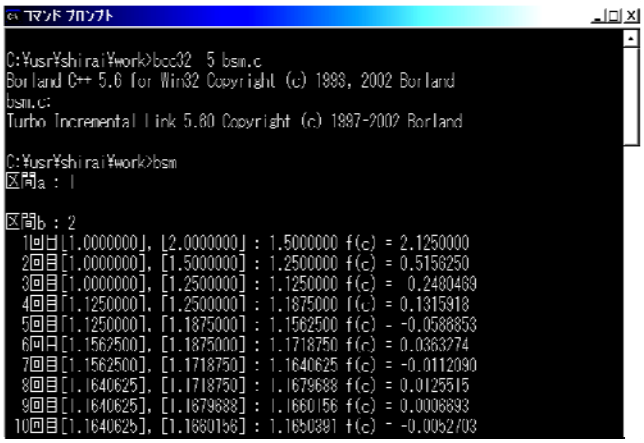


図１．コンソール対話型プログラミング環境

2. Processing 用フレームワーク Crowbar + Tomahawk

2.1 Processing とは

Processing は Casey Reas, Ben Fry が 2001 年に発案し, 2008 年 11 月末に安定版 Processing1.0 が公開された比較的新しいプログラミング言語である．Java を基盤とするオブジェクト指向言語で，グラフィックス描画機能に重点を置くことでプログラムの実行結果を即座に視覚的に得られる点が優れている．動作環境は Linux, Mac, Windows である．Processing の開発環境はインストール作業が不要である．たとえば USB メモリに Processing のシステム一式をコピーすれば，動作条件を満たすコンピュータに装着するだけで即座にプログラミングを開始できる．Processing には図２左に示すスケッチブックと呼ばれる統合開発環境が用意されている．スケッチブックのテキストエディタ部にソースコードを記述し実行すると図２右に示す実行ウィンドウが一つだけ開く．このウィンドウ上にグラフィックスを描画し，マウスやキーボードからの入力をトリガとするイベント駆動型のプログラムを作成可能である．Processing のソースコードはスケッチと呼ばれる．スケッチのプロジェクトはフォルダ単位で管理され，フォルダ内にある拡張子 pde のテキストファイルがソースファイルである．スケッチはワンクリックでフォルダごと ZIP 書庫形式にアーカイブできる．完成したスケッチは Linux, Max, Windows 用の実行形式（実体はローダと Java アプレット）でのエクスポートが可能である．JAR（Java Archive）形式でのエクスポートも可能なため, 完成したスケッチを Web サイトにアップロードして配布することも可能である．Android SDK をインストールすれば Android アプリケーションとしてもエクスポート可能である．Processing のスケッチを Web ブラウザ上で実行するための JavaScript ライブラリ Processing.js も開発された．OpenGL に対応した三次元グラフィックス機能や動画ファイルにエフェクトを施すことができるなど単体での多機能さに加えて，前述のようなさまざまなデジタルデバイス上で制作物が実行できるポータビリティの高さは，単なる入門者用言語の域を超えている．

Processing の文法は単純化された Java 言語であり，C 言語を習得した者であれば容易に習得できる．静的変数や列

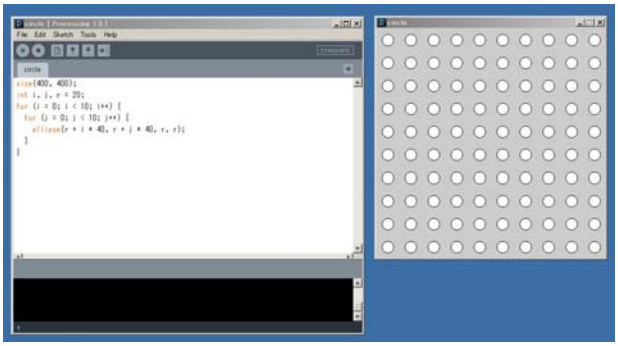


図２．Processing の動作画面

挙型変数が利用できない制約はあるが，Processing は Java 上に実装されているため Pure Java 表記により Java の機能を通して同等の機能の実現が可能である．Java 同等の自動ガーベージコレクション機能を備えるため不要な配列やクラスのインスタンスを明示的に破棄する必要はなく，メモリリークの心配もない．浮動小数点型が３２ビット単精度であるため科学演算には強くないが，文字列型変数や配列もオブジェクトとして定義されているため自然とオブジェクト指向プログラミングの概念が身に付く．

キーボードの打鍵やマウスクリックなどの入力を検知する機能を標準で備えているため，比較的容易にゲームやデジタルアート作品などのインタラクティブなアプリケーションを開発可能である．一方，GUI アプリケーションの開発を前提としているため，第１章でも述べたように古典的なコンソール対話型のアプリケーションを作成することは全く考慮されていない．具体的な仕様上の問題点は大きく分けて二つ，キーボードから文字列を入力する機能が無いことと，折り返しやスクロールアップに対応した文字列表示機能が無いことである．Processing の実行ウィンドウではキーボードの打鍵を検知できるが，これはキーボードのキーと一対一対応した一文字分の文字情報しか Processing 標準の機能では取得できないという意味である．C 言語における scanf()や gets()に相当する標準入力から文字列の入力を待ち受ける関数は存在しないし，かな漢字変換機能を介した日本語文字列の入力も不可能である．Processing にも文字列を実行ウィンドウ上に表示する関数 text()は存在するが，実行ウィンドウ上の表示位置の *x,y* 座標を指定してグラフィックス要素として描画するものである．実行ウィンドウ右端に溢れ出る文字列は画面に表示されないので，数値計算の実行結果などを表示する際には常に実行ウィンドウから文字列が溢れ出てしまわないかチェックしなくてはならない．

2.2 Crowbar + Tomahawk とは

Crowbar は Processing 上でキーボードによる文字列入力と折り返しやスクロールに対応した文字列出力が可能なコンソール対話型のアプリケーションを開発可能とするフレームワークである．Tomahawk は Crowbar にマルチレイヤー対応のビューポート機能を追加するグラフィックスライブラリである．Crowbar + Tomahawk は以下の５個のソースファイルで構成される．

1. (スケッチ名).pde　：ユーザが記述する雛型
2. optionCode.pde　：ユーザが記述する雛型
3. crowbarClass.pde　：Crowbar 関係のクラス定義
4. tomahawkClass.pde　：Tomahawk 関係のクラス定義
5. functions.pde　：汎用性の高い関数群

3,4,5 のソースファイルはフレームワーク本体なので利用者は一切手を入れる必要が無い．1,2 のソースファイルには最低限必要な関数の雛型が定義されており，利用者は作成するプログラム固有の処理をこの雛型内に記述する．

2.3 Crowbar の機能

2.3.1 プロセス制御

図２の実行結果を得る Processing のソースコードを図３に示す．この例では実行ウィンドウのサイズを 400 ピクセル×400 ピクセルに変更し，40 ピクセル間隔で直径 20 ピクセルの円を縦横２０個表示する．図３(a)に示す単純な書式は Processing の最も基礎的な形式である．グラフィックスを用いたり自作の関数を呼び出すことも可能であるため初学時には最適だが，キーボードやマウスからのイベントを受信したり動きのあるグラフィックス表現を実現するには Processing 標準の書式である図３(b)の表記が好ましい．Processing には setup()と draw()の二つの関数が C 言語における main()同様に予約関数として用意されており，利用者は必要に応じて関数内に処理を記述する．もし setup()が記述されているならば，スケッチ実行後，広域変数の宣言を処理した後に setup()が実行される．その後，draw()が記述されているならば stop()あるいは noLoop()が実行されるまで draw()が繰り返し実行される．もし予約関数 keyPressed()が記述されているならば，draw()実行中のキーボード打鍵イベント発生時に Processing により自動的に呼び出される．押されたキーの文字コードは予約変数 key あるいは keyCode により取得可能である．一般的にはイベント駆動型の考えに基づき keyPressed()中に押されたキーに応じた処理を記述するが，boolean 型の予約変数 keyPressed で打鍵の有無を判断して draw()中で処理を記述しても構わない．いずれの方法にしてもキーの打鍵を検知できるのは draw()

```
size(400, 400);
int i, j, r = 20;
for (i = 0; i < 10; i++) {
  for (j = 0; j < 10; j++) {
    ellipse(r + i * 40, r + j * 40, r, r);
  }
}
```

(a) 基礎的な表記法

```
int i, j, r;

void setup() {
  i = j = 0;
  r = 20;
  size(400, 400);
}

void draw() {
  ellipse(r + i * 40, r + j * 40, r, r);
  if (++i >= 10) {
    if (++j >= 10) noLoop();
    i = 0;
  }
}
```

(b) Processing 本来の表記法

図３．Processing のプログラム例

実行時の 1 サイクルの間に 1 キーのみである．したがって setup()や draw()中で二文字以上の連続した文字列入力は原理的に不可能である．根本的にこの問題を解決するために Crowbar では，Processing のもつ setup(), draw(), keyPressed()等の動作の仕組みを利用した独自のプロセス管理を行っている．プロセスの状態遷移の大筋は以下の通りである．

1. Crowbar 自体の初期化　：initCrowbar()
2. 動作環境の設定　　：Options()
3. パラメータ等の宣言　　：Setup()
4. プログラムコメントの表示
5. 宣言されたパラメータのキーボードからの入力
6. 再入力か実行かの確認（再入力希望時は 5 へ）
7. メインプログラム実行前処理　：preMain()
8. メインプログラムの実行　　：Main()
9. ループ処理の実行　　：userDraw()
10. メインプログラム実行後処理　：postMain()
11. 再実行確認（再実行希望ならば 5 へ）
12. 終了処理

setup()内で 1 を，draw()内で状態遷移に基づいて 2 から 12 を順に実行する．setup()と draw()を Crowbar が利用してしまうため利用者は使えない．対応する関数として Main()と userDraw()の二つを用意した．Main()だけを用いれば古典的な単純実行のプログラム，userDraw()も用いればイベント駆動型の Processing らしいループ実行するプログラムを開発できる．他の予約関数，たとえば keyPressed()に対応する userKeyPressed()などの関数の雛型も optionCode.pde 内に用意してあるので必要に応じて処理を記述すれば良い．

2.3.2 キーボードからのテキスト入力機能

利用者がスケッチ実行時にキーボードから入力したいパラメータは Crowbar で用意した関数の雛型の一つ，Setup()にまとめて宣言する．Setup()は setup()と名前は似ているが別物である．パラメータ宣言用の命令として，数値入力用の define()，文字列入力用の defineStr()，多肢選択入力用の defineSel()の三種類を用意した．define()と defineStr()の書式は以下の通りである．

```
define(String msg, float initValue).label(String name)
defineStr(String msg, String initValue) .label(String name)
```

msg はパラメータ入力要求時に画面に表示するメッセージ，initValue は初期値（省略可），name は入力されたパラメータを呼び出すためのラベル名である．defineSel()の説明は本稿では割愛する．以下のように宣言すると，図 4 に示すように利用者の作成した Main()実行前に Crowbar がキーボード入力を要求する．

```
void Setup() {
  crowbar.define("区間ａ", 1.0).label("xa");
  crowbar.define("区間ｂ", 2.0).label("xb");
}
```

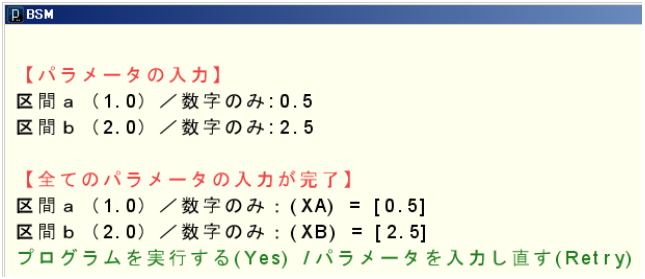


図４．パラメータ入力

define()で宣言したパラメータは数値入力用の 0 から 9 および記号以外のキー入力は無視される．defineStr()ではキー入力の制限は無いが，Processing の仕様上，日本語文字列の入力は不可能である．キー入力された文字は一文字単位でエコーバックされ，バックスペースキーにより一文字単位で削除可能である．初期値をそのまま採用する場合は文字列をキー入力しないでエンターキーのみを押せば良い．TAB キーを押せば，以降の初期値が設定されているパラメータは自動的に初期値が入力される．C 言語の scanf()や gets()との違いは，Main()等の任意のタイミングで文字列の入力が可能なのではなく，必要なパラメータは全て Setup()内で宣言し，Main()実行前に一括して入力しなくてはならない点である．全パラメータの入力が完了したら，パラメータを再入力するか，Main()等を実行するかを Crowbar が確認してくる．実行を選択すると preMain(), Main(), … の順に利用者が記述したコードが実行される．

入力済みのパラメータ値を Main()等で取得するには以下の関数を用いる．

- float　getFloat(String name)
- int　　getInt(String name)
- String getStr(String name)

name は Setup()内で define()系命令を用いてパラメータを宣言した際のラベル名である．Main()等の実行前にキーボードで一括入力したパラメータはラベル名と紐付けされてメモリ中に格納されており，Main()等の任意のタイミングでラベル名を介して参照できる．パラメータ宣言時の数値か文字列かの指定はキーボード入力の制限が目的であり，実際にはどちらの宣言でも文字列型でデータはメモリ中に格納されている．getFloat(), getInt()を用いると文字列型データを浮動小数点数，整数に変換して返し，getStr()を用いると文字列型のまま無変換で返す．以下に例を示す．

```
void Main() {
  float a, b;
  a = crowbar.getFloat("xa");
  b = crowbar.getFloat("xb");
```

なお，パラメータは define()系命令による宣言時に自動的に Crowbar 内部で 0 から始まる整数のシリアルナンバーが割り振られる．パラメータ値の読み出しはシリアルナンバーを直接指定しても行えるが，Crowbar 内部にカウンタを用

意しているので以下のようにラベル指定なしに宣言順に読み出すことも可能である．この場合，宣言時の.label(name)も記述を省略できる

```
a = crowbar.getFloat();
b = crowbar.getFloat();
```

前述したようにMain()等の自由なタイミングでキーボードからのパラメータの入力を状況に応じて行なうことはProcessing の動作原理上，不可能である．たとえばソーティングの演習を行う際に，まず入力されるデータ数を入力した後に，指定されたデータ数分のデータ入力を繰り返し処理により行うというような動作である．この点に留意すれば比較的容易に過去の FORTRAN, BASIC, C 言語などのソフトウェア資産を Processing に移植可能である．

2.3.3 実行ウィンドウへのコンソール型テキスト出力機能

Crowbar では実行ウィンドウ上にコンソール端末風のテキスト出力が可能である．実行ウィンドウ右端から溢れ出す文字列を表示しようとすると自動的に折り返しが生じ，実行ウィンドウ最下端まで文字列が表示された後は自動的にスクロールアップする．以下にテキスト出力関係の代表的な命令を示す．

- `write(String str)` ：**文字列出力**
- `writeln(String str)` ：**文字列出力後に改行**
- `newline()` ：**改行**
- `textColor(int col)` ：**以降に出力する文字色の変更**
- `clrscr()` ：**全画面消去**
- `locate(float x, float y)` ： **カーソル位置を移動**

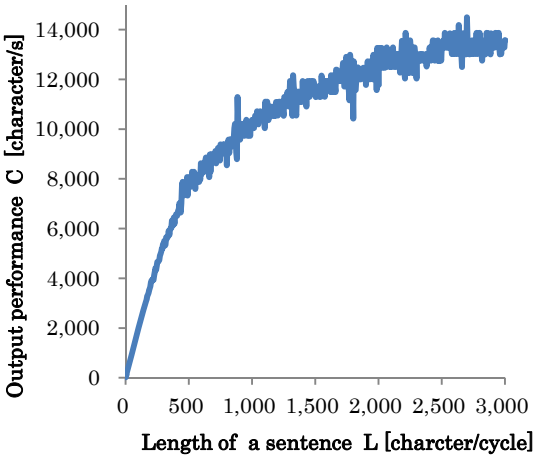
crowbar.writeln(“Hello world!”); のように用いる．write()は文字列を現在のカーソル位置に表示するだけだが，writeln()は文字列を表示した後にカーソル位置を自動的に次行の行頭に移動する．カーソル位置は実行ウィンドウ左上隅を(0.0, 0.0)とする浮動小数点数の座標である．グラフィックス描画のピクセル座標系とは異なる Crowbar 独自の座標系である．*x* 座標値の単位は文字数であり，全角文字幅を 1.0 文字，半角英数文字幅を 0.5 文字とする．ただしプロポーショナルフォントによる自然な表示にも対応しており，その場合は表示する半角英数文字毎に文字幅は異なる．*y* 座標の単位は行数であり，1 行分の高さが 1.0 である．write()/writeln()で文字列を出力すれば自動的に *x* 座標値は更新され，改行されれば *y* 座標値が自動的に 1.0 加算される．newline()は文字列を出力しないで単純に改行を行う．textColor()は１６進整数表現による色指定により，以降に表示される文字色を変更し，戻り値として変更前の文字色を返す．引数を指定しなかった場合は文字色の変更は行われず，現在の文字色を返す．clrscr()は実行ウィンドウを消去し，カーソル位置を(0.0, 0.0)に移動する．カーソル位置はlocate()により自由に移動可能である．clrscr()と locate()をuserDraw()中などで組み合わせて用いることでリアルタイムに値の変化するカウンタなどを実装可能である．

Crowbar は自前のテキストバッファを用意している．テキストバッファのデータ構造は可変長の文字列バッファと文字色などの文字属性を要素に持つ片方向リストである．リストの一ノードは明示的あるいは折り返しにより自動改行されるまでの一行分の文字列データを保持する．利用者が write()/writeln()で文字列を出力すると現ノードの文字列バッファに文字列を追加する．明示的な改行や折り返しが発生すると次ノードを生成して現ノードに連結する．テキストバッファに追記された文字列を実行ウィンドウへ一文字単位で出力しながらカーソル位置の *x* 座標を文字幅に応じて更新し，これから描画する文字が実行ウィンドウ右端からはみ出すと判断したら自動的に折り返して次行の行頭から続きを描画する．文字列出力毎にカーソル位置は自動更新されるので利用者は座標値を意識することなく文字列出力と改行を組み合わせた古典的なコンソール出力のスタイルで文字列を実行ウィンドウに順次出力できる．カーソル位置が最下端に達したら画面を消去して所定の行数だけ自動的にスクロールアップして再描画する．自動スクロールアップの送り量は設定により変更可能である．自動スクロールアップ以外に，Main()等の動作終了後の再実行か終了かの選択待ち状態で，カーソルキーおよびスクロールキーの上下により一行単位あるいは半ページ単位でのスクロールアップ／ダウンも可能である．

Crowbar のテキスト出力の速度計測結果を図５に示す．図５(a)に示すように半角 60 文字幅で 15 行の実行ウィンドウを用意し，合計一万文字の半角数字を write()命令で一文字ずつ描画した際の表示速度の測定結果のグラフが図５(b)である．横軸は userDraw()の一サイクルに出力する文字数 *L*，縦軸は一万文字の表示に要した時間より計算した１秒あたりの表示文字数 *C* である．実行ウィンドウ右端で折り返しが発生し，実行ウィンドウ下端に達すると自動的に（本実験では１行ずつ）スクロールアップする．WindowsXP/64（Core i7, 2.7GHz, 6GB）のパーソナルコンピュータ上で，*L* = 1[文字]の時に 506.8[s]を要し，*C* = 約 19.7[文字/s]，*L* = 2, 3[文字]でそれぞれ *C* = 約 39.5[文字/s]，約 59.3[文字/s]，*L* = 430 近辺までほぼ線形に *C* が増加し，*L* = 2,500 以降で *C* は約 13,000[文字/s]前後で飽和する．以上の結果より，１秒あたり約 216 行の表示と 200 行程度の行単位のスクロールアップを行う性能を持つことが分かる．

Crowbar によるテキストバッファを用いたテキスト出力も Processing の text()による出力同様にピクセル出力であるためマウスで領域選択して文字情報としてコピーすることはできない．Crowbar はテキストファイルへのログ出力の機能を持つ．図６に示すように，Crowbar が出力するシステム出力も Main()等で write()/writeln()を用いて出力した文字列も全て記録されるので，数値計算結果を Excel などの表計算ソフトウェアで可視化処理可能である．なお，ログ

(a) 実行ウィンドウ（60 文字×15 行）



(b) 計測結果

図５　テキスト出力性能試験

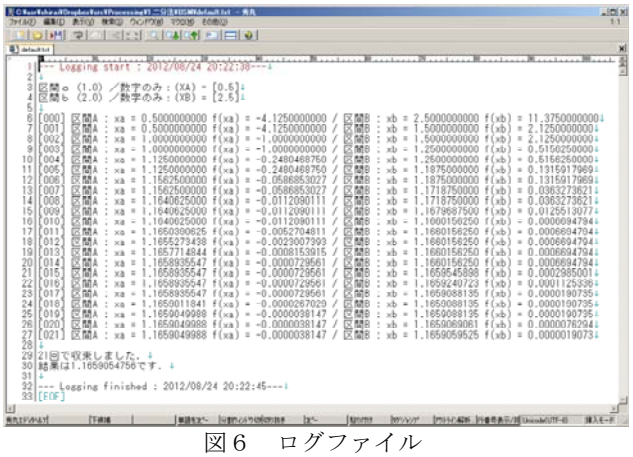


図６　ログファイル

ファイルへのマルチバイト文字列の出力はUTF-8形式なので，前処理として文字コード変換を行わずにログファイルを Excel 等に読み込ませると日本語文字列が文字化けする場合がある．

2.4 Crowbar の仕組み

2.3.1 節で説明した Crowbar のプロセス管理の制御機構を図 7 に示す．Crowbar により呼び出される initCrowbar(), Options(), Setup(), preMain(), Main(), userDraw(), postMain() 等の関数内に利用者はアプリケーション毎に異なる動作を記述する．

initCrowbar(), Options(),Setup()は Crowbar 実行後に setup() 内で一度だけ順番に実行される．initCrowbar()には Crowbar の実体である crowbarClass のインスタンス crowbar を生成するためのコードが以下のように記述されている．

```
crowbarClass crow;
void initCrowbar() {
  crow = startCrowbar.generate(50, 30, 2);
}
```

startCrowbar.generate()はインスタンス crowbar を生成するランチャーである．三個の引数の内の最初の二個の引数で実行ウィンドウのサイズを文字数，行数で指定する．第三引数は後述するグラフィックスライブラリ Tomahawk の動作モードである．Tomahawk を使用しない場合は 0 あるいは省略，Tomahawk を使用する場合は 1 か 2 を指定する．Tomahawk 動作モード 1 と 2 の違いは 2.6 節で述べる．startCrowbar.generate()の戻り値は crowbar のアドレスなので，上記例のように crow などの自由なインスタンス名にシャローコピーして構わない．Options()と Setup()には実行順以外に本質的な違いは無い．いずれも Crowbar と Tomahawk の設定，パラメータの宣言，ビューポートの宣言と設定（後述）を記述するための関数である．たとえば Options()には課題プログラムの雛型を与える教員が初期設定を記述し，学生が Options()よりも後に実行される Setup()内で与えられた設定を変更するといった使い分けを想定している．

以上の初期設定を実行した後，Crowbar は draw()内に実装した制御機構に従って状態遷移を行いながら以下の処理をループ毎に順次行う．Setup()でパラメータが宣言されていたならば，まず msg を実行ウィンドウにテキスト出力し，次いで draw()をループ実行しながらキーボードから入力されるキーを一文字ずつバッファリングする．エンターキーが押されたら現在のパラメータの入力が完了である．続いて他にもパラメータが宣言されていたのであれば上記動作を繰り返す．全パラメータの入力が完了したらプログラム実行かパラメータ再入力かの選択を利用者に問う．実行を選択したら，preMain(), Main(), userDraw(), postMain()の順に

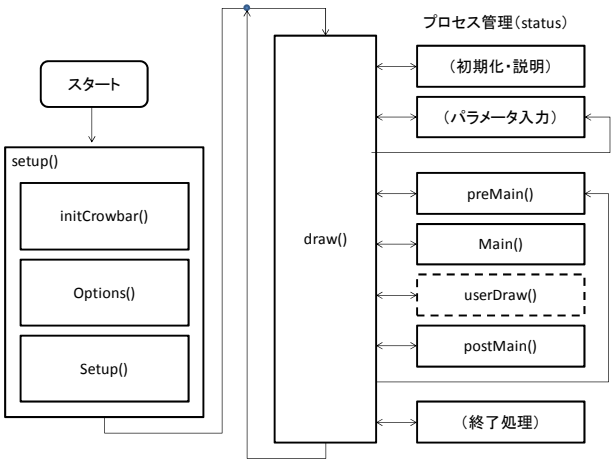


図７　フレームワーク Crowbar の動作

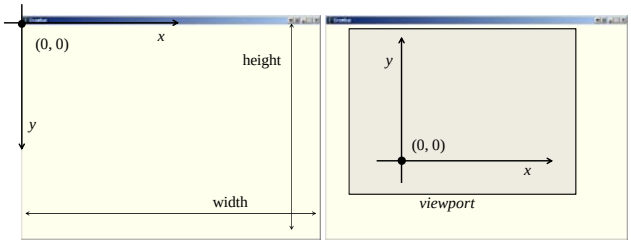
ユーザのコードを実行する. preMain(), postMain()は Main(), userDraw()実行前後に行ないたい前処理と後処理を記述するために用意した関数で，optionCodes.pde 内に雛型が存在する．前処理や後処理が必要無ければ関数内にコードを記述する必要はない．Main()は C 言語の main()に相当する関数であり，setup()同様に一度だけ実行する．Main()終了までの間に crowbar.nonStop()が実行されたならば userDraw()を draw()同様にループ実行する. Processing の draw()と異なり userDraw()の宣言は省略できないので, userDraw()を実行するかしないか（デフォルト）は利用者が明示的に指示する必要がある．userDraw()を実行しないか，userDraw()が crowbar.stop()によって終了した後に postMain()を実行する．全てのコードの実行が終了したら，パラメータ入力からやり直すか，Crowbar を終了するかを利用者に確認する．

2.5 Tomahawk の機能

Tomahawk の最大の特徴は容易に複数のビューポートを実行ウィンドウ内に作成して合成出力可能な点である．図８に Tomahawk を利用した移動ロボットのシミュレータの例を示す．実行ウィンドウを三領域にビューポートで分割している．上と右下のビューポート内のロボット画像は同一のワールド座標値と描画コマンドを用いて描画している．右下のビューポートに示すように，ロボットが移動や回転を行っても，視点の回転や中心座標の移動機能によりロボットを常時上向きかつ中心に表示するような視点のトラッキングも可能である．

一般的なグラフィックデバイス同様に，Processing の実行ウィンドウの座標系も図９(a)に示すように左上角が原点(0, 0)，右方向と下方向がそれぞれ *x* 座標軸と *y* 座標軸の正，各軸座標値が浮動小数点数のピクセル座標系である．

図８　Tomahawk によるアプリケーション例

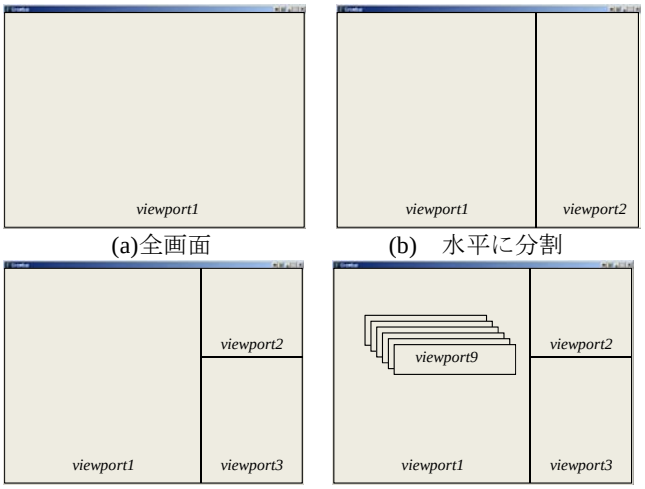


(a)　ピクセル座標系　　(b)　ワールド座標系

図９　二つの座標系

座標値は浮動小数点数で指定可能だが，translate()や scale()といった関数を用いない限りは実質的に整数と同等の扱いである．Tomahawk では確保したビューポート内に world()命令を用いてワールド座標系を自由に設定可能である．ワールド座標系は図９(b)に示すように右方向が *x* 座標軸の正なのはピクセル座標系と同じだが，上方向を *y* 座標軸の正とする浮動小数点数の座標系である．ビューポート内の座標原点の位置やピクセル比を変更可能なので，利用者は座標変換を行なわずに数値計算結果に基づいて単純に点をプロットしたりラインを描画したりできる．ピクセル比とはワールド座標系における長さ１がピクセル座標系の何ピクセルに相当するのかを表わす比率である．現段階では *x* 軸方向も *y* 軸方向も同一のピクセル比だが，将来的には各軸の物理量が異なる場合などにも対応できるように各軸でピクセル比を異なる値に設定できるように拡張する予定である．ビューポート内への描画の際の座標値はピクセル座標系でもワールド座標系でも指定可能である．たとえばラインを描画するならば，ワールド座標系では line()命令，ピクセル座標系で_line()命令を用いる．

もし Processing で図８の例のように実行ウィンドウを複数の矩形領域に分割したいならば，指定領域からはみ出す部分はクリッピングする必要がある．三角形，四角形と言った直線のみで構成される図形要素ならば独自の関数を作成してクリッピングすることもできるが，円や文字をクリッピングすることは Processing では困難である．Tomahawk は自由なサイズの矩形領域（ビューポート）を実行ウィンドウ上に作成できる機能を実装したことでこの問題を解決した．各ビューポートに対して Processing が持つ大半の描画命令が利用可能かつ矩形領域からはみ出した描画オブジェクトは正しくクリッピングされる．ビューポートの作成方法を四種類用意した．全画面サイズ，縦横幅指定，分割，複製である．図１０に例を示す．全画面サイズのビューポートは図１０(a)のように実行ウィンドウ全体を一枚のビューポートで覆うように作成される．ビューポートの分割



(a)全画面　(b)　水平に分割

(c)　垂直に分割　(d)　複製

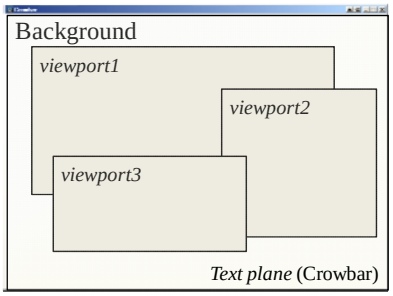
図１０　ビューポートの作成法の例

は既存のビューポートを図１０(b), (c)のように水平方向あるいは垂直方向に分割することで新しくビューポートを作成する．複製は図１０(d)のように一つのビューポートを作成後,それと全く同じ属性をもつビューポートを作成する．作成したビューポートの表示／非表示は viewVisible(), viewUnvisible()命令でいつでも自由に個別に変更できる．図８のスケッチのビューポートの確保と設定は以下の通りである．

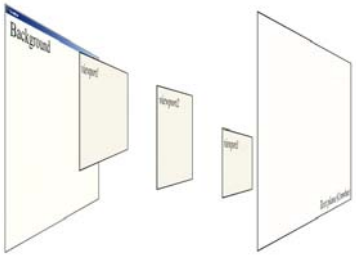
```
#01 void Setup() {
#02     crow.createView("Oval").viewBgColor(#eeeeee);
#03     crow.splitViewV("Text", 0.45).viewBgColor(#ffffff);
#04     crow.cv.prepareFont().setTextSize(14);
#05     crow.splitViewH("DMC", 0.6).viewBgColor(#eeeeee).
            viewPivotCenter();
#06     crow.getView("Oval").world(0.0, 0.0, 12200.0, 2000.0).
            worldOrigin();
#07     crow.getView("DMC").worldY(-200.0, 200.0).worldOrigin();
#08     crow.cloneView("Oval", "Trajectory").
        setTransparentView(true);
#09 }
```

２行目は，createView(“Oval”)で”Oval”というラベル名で全画面サイズのビューポートを作成し，viewBgColor(#eeeeee)で背景色を灰色に設定している．３～４行目は”Oval”を0.55:0.45 の比率で上下に分割し，比率 0.45 の大きさの下のビューポートを”Text”というラベル名で新規作成し，背景色を白に設定，テキスト表示用のフォントを準備し，文字サイズを 14 ピクセルに設定している．５行目は”Text”を0.4:0.6 の比率で左右に分割し，右側のビューポートを”DMC”というラベル名で新規作成し，背景色を灰色に，回転や拡大縮小の基準点となる視点をビューポート中央に設定している．６～７行目は”Oval”と”DMC”のワールド座標系をそれぞれ設定している．８行目は”Oval”を複製した”Trajectory”というビューポートを新規作成し，背景を透明に設定している．”Oval”はワールド座標系が設定済みなので”Trajectory”も同じ位置に同じワールド座標系を持って最上面に配置される．背景が透明なので線などを描けば”Oval”に重ね書きされる．”DMC”はロボットの移動をアニメーション表示するために userDraw()がループするたびに消去して書き直すが”Trajectory”に影響はなく，ロボットの移動軌跡を描き続けることができる．

ビューポートは図８の例のようにタイル状に平面内に配置するだけではなく，昨今のウィンドウアプリケーション同様に図１１(a)に示すように重ねて配置可能である．各ビューポートの上下関係の入れ替え，ビューポートの位置の移動，表示／非表示の変更，背景色の不透明度の変更，枠線の太さ／色／非表示の変更をアプリケーション実行中にダイナミックに変更可能である．マウスクリックした際のアクティブなビューポート番号とそのビューポート内にお



(a) 画面構成例



(b) レイヤーの合成順（Tomahawk 動作モード２）

図１１ Tomahawk モード２時のレイヤー構成

けるピクセル座標値を取得できるなど，一般的なウィンドウアプリケーションと同等の機能を備えている．

2.6 Tomahawk の仕組み

Tomahawk では複数のウィンドウ風のビューポートを作成して合成するための仕組みとして Processing が持つ描画バッファ専用レンダラーPGraphics クラスを用いている．PGraphics のインスタンスは複数作成可能であり,それぞれが各ビューポートに対応する．Processing のほぼ全ての描画関数は PGraphics の描画バッファに対して利用可能だが，PGraphics 経由の描画結果は実行ウィンドウに全く反映されない．実行ウィンドウに描画バッファの内容を反映するには blend()関数を用いている．blend()は画像ファイルを実行ウィンドウの任意の位置に合成する関数として用いられることが大半だが PGraphics の描画バッファも実行ウィンドウに合成可能である．Tomahawk では draw()のサイクル終了時に，図１１(b)に示すよう実行ウィンドウに対して”深さ” の深いビューポートから順に blend()で描画バッファを重ね合わせ合成し，重なりのあるビューポートを実現している．なお，ビューポートを合成される背景の実行ウィンドウはビューポートとは独立しているので，Processing の描画コマンドによって自由に描画できる．Tomahawk 動作モード１の時は背景に Crowbar のテキスト出力も重ね書きされるためビューポートによってテキスト出力は隠れてしまうが，Tomahawk 動作モード２の場合は図１１(b)に示すように背景とは独立した Crowbar によるテキスト出力専用の全画面サイズかつ背景が透明なテキスト表示プレーンを持ち，最上位に表示するため常に表示が隠れることはない．ただ現バージョンの Processing1.5.1 ではレンダラーの制約により，Tomahawk 動作モード１の方がテキストの出力結果が奇麗かつ表示速度が速い．

3. 情報教育課程における Crowbar + Tomahawk の位置付け

Windows, Mac OS, Linux など GUI により操作可能なコンピュータに加えて，iOS, Android などタッチ方式採用の OS が増えるなどコンピュータ端末の環境が大きく変化する過渡期であっても情報教育の核が「データ構造とアルゴリズム」の理解と習得であることに変わりは無い．たとえば図１２中(A)のようにデータ型の理解と数値計算の基礎から情報教育を始め，アルゴリズムを学んだ後にコンピュータグラフィックス（以下，ＣＧ）による表現手法の習得へ進む課程も，同図中(B)のようにＣＧを入り口として学生のモチベーションを維持しつつアルゴリズムの習得へ導く課程も間違いではない．コンピュータ環境，教員の方針，学校の特性の違いを考慮して適した課程を構築するべきである．

課程(A)は，Processing 上で Crowbar を用いることでデータ型の理解と基礎的な数値計算，さらにアルゴリズムの習得までカバーし，その後に Processing の描画機能を利用してグラフィカルなアプリケーションを学ぶ流れが考えられる．課程(B)は，入門書などを参考に Processing のみでプログラミング能力の基礎と発展まで学ぶ．その後に別の言語に乗り換えるのではなく Crowbar でアルゴリズムを本格的に学ぶ流れが考えられる．Tomahawk は命令数が多く，座標系やビューポートの概念を理解する必要があるためＣＧの基礎を先に学ぶことが望ましく，導入教育には適さない．課程(A),(B)などで Processing + Crowbar によるプログラミングの基礎知識を修得した後に卒業研究などで実験結果や数値解析結果の可視化に利用するのに適している．以上のように Processing, Crowbar, Tomahawk を組み合わせることで，プログラミングの導入教育から応用まで一つの言語で対応可能である．

過去にプログラミング言語を修得して活用していたもののコンピュータ環境の大幅な変化により開発環境が失われてプログラミングから遠ざかっている教員は多い．FORTRAN，BASIC，コンソール版 C 言語のソースリストを死蔵しているだろう．多忙な業務の片手間で Windows や iOS，Android 向けアプリケーションの開発環境を整備して学習し直す時間的余裕は無い．Processing ならば比較的容易に短時間で開発環境を整えられるので，Crowbar + Tomahawk を習得すれば死蔵しているプログラムを移植し，専門教育に役立つ教材を開発できる．開発したプログラムは Java の JAR 形式で，e ラーニングシステム上で提供して学生の学習を助けることができる．

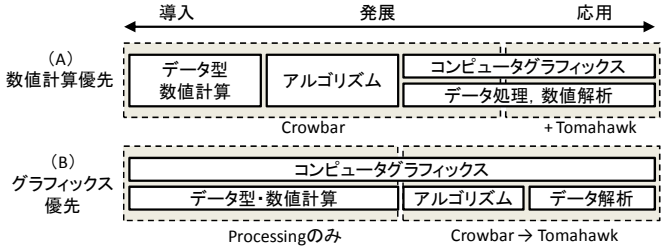


図１２　情報教育課程の例

4. おわりに

一般校から大学まで，さまざまなタイミングでプログラミングの初歩を学ぶ機会がある．各学校や学生の性質に合わせて様々なプログラミング言語が初学者用言語として選択可能である．その中でも GUI 中心のコンピュータ環境下，旧来のコンソール対話型環境で数値計算から始めるプログラミングスタイルとグラフィックスから始めるプログラミングスタイルを Processing のみで実現可能とするフレームワーク Crowbar を開発した．Processing は入門者用というだけではなく，デジタルコンテンツの作成など幅広い用途に本格的に活用できる高い能力も備えている．しかし高度な描画能力を生かすには複雑なプログラミングの知識と経験が必要である．その助けとなるように，ビューポートを用いた高度なグラフィックスアプリケーションを作成可能なグラフィックスライブラリ Tomahawk も開発した．本稿では Crowbar, Tomahawk の特徴，機能，動作原理を説明した．Crowbar と Tomahawk をどのタイミングで教育課程に組み込むべきか一例を示すと共に，プログラミングから遠ざかっていた教員にも e ラーニングコンテンツを作成し，授業で活用できることを示した．

謝辞　Crowbar + Tomahawk をオープンソースプロジェクトとして開発および公開する場を無償で提供し，管理してくれている SourceForge.JP の運営の皆さんおよび OSDN（株）に感謝いたします[9].

参考文献

1) 文部科学省：“高等学校学習指導要領解説　情報編”, 第６節 アルゴリズムとプログラム，pp.66-69，平成 22 年１月.
2) 西田知博，原田章，中村亮太，宮本友介，松浦敏雄：“初学者用プログラミン環境 PEN の実装と評価”，情報処理学会論文誌，Vol.48, No.8, pp.2736-2747 (2007-08).
3) 兼宗 進，久野 靖：“ドリトルで学ぶプログラミング（第２版）”，イーテキスト研究所(2011).
4) 菊池 誠："プログラミング，何をどう教えているか：Processing によるプログラミング教育"，情報処理，Vol. 52, No. 2, pp.213-215, 2011.
5) 森崎巧一，比気千秋，大海悠太，橘口宏衛，橋本　誠，上村　眞，田丸直幸："センサーと Processing を利用した情報デザイン教育のための教材の開発"，大妻女子大学紀要，社会情報系，社会情報学研究，Vol. 19, p. 93-100, 2010.
6) 田村景明："Processing と Arduino の連携による”ものづくり”教育"，平成２４年度全国高専教育フォーラム, PO-A04, 2012.
7) 尋木信一："Processing を教材とした C 言語学習",平成２４年度全国高専教育フォーラム, PO-A11, 2012.
8) Casey Reas, Benjamin Fry（船田 巧訳）："Processing をはじめよう". オライリー・ジャパン，2011.
9) http://sourceforge.jp/projects/crowbar/