卒業論文

～によるProcessingの卒業研究

中部大学経営情報学部経営情報学科

学籍番号:BG15012

伊藤恭兵

目次.

1.はじめに(研究目的・研究背景)

2.Processingとは

2-1.歴史

2-2.特徴

3.準備

　3-1.ダウンロードから起動まで

　3-2.基本操作

　　　3-2-1.画面詳細

　　　3-2-2.命令入力

　　　3-2-3.プログラムのデバック

4.お絵かきアプリ作成

4-1.描画関係

4-2.カラーパレット

　　4-2-1.カラーパレットの原画

　　4-2-2.カラーパレットの組み込み

4-3.undo・redo

5.Processing.jsの使用

6.今後の課題

7.参考文献

1. はじめに(研究目的・研究背景)

　近年では、IT技術が私たちの生活には欠かせないものとなっている。2020年からプログラミングの義務教育導入が検討されていること(http://www.mext.go.jp/component/a\_menu/education/micro\_detail/\_\_icsFiles/afieldfile/2018/03/30/1375607\_01.pdf )、買い物での会計が電子化されていることなどから実際の生活でも感じ取ることが出来る。今あげた例だけでもITに関する重要性が増していることがわかる。なので、まずは私の身近にあるアプリケーションについて学習しようと考えた。アプリケーションといっても様々で、SNS、Webブラウザ、データベース、メディアプレイヤー等あるが、まずは通信技術を使わずかつ作りもイメージしやすいお絵かきアプリでの機能がどのような仕組みになっており、プログラムされているのか・作らなければならないのかを学習することに決めた。本論文では、プログラミングの入り口と考えているので、将来的には今回作成したお絵かきアプリと通信技術を用いたお絵かきの森(複数人でリアルタイムにお絵かきをすることが可能でありお題に沿ったものを書き上げていくことを主としたゲーム。https://dic.pixiv.net/a/%E3%81%8A%E7%B5%B5%E3%81%8B%E3%81%8D%E3%81%AE%E6%A3%AE )のようなアプリケーションも作っていきたいと考えている。 では実際に何を利用して学習し、プログラムを書いていこうかと考えたときProcessingが目にとまった。 　理由としては、Processingを使う利点である「直観的に欠ける点」や「開発環境の構築がほかのプログラミングソフトに比べてコンパクトである点」、「すぐにビジュアル化できる点」などがあり、私のような初心者が始めるにはうってつけだと思ったからである。

では、このprocessingを用いて当初の目的であったアプリや機能の作成を自分のような初心者にもわかるような説明書を書いていこうという方向性の元、卒業論文を書いていこうと思う。

2.Processingとは

2-1.歴史

　Processingは、Design By Numbersというプログラムに起源がありMITメディアラボの教育用に作成されたのが始まりである。 　Design By Numberとは、アーティストやデザイナーを対象に作られたプログラムであり、プログラムによるデザインの考え方を伝える教材として作られた。このDesign By Numberは約10個の命令を覚えることで、プログラミングを用いたデザインの概念を大まかに知ることが出来る。また、Javaアプレットで出来ているため一般的なWebブラウザから使用することが可能である。 　しかし、このDesign By Numberはデザインすることの出来る範囲が100ピクセルの正方形であったり、色が白黒で100段階の濃さの違いでしか表現できないという制限があった。プログラミングの概念を大まかに知ることは可能だが表現の方法に限界があるため、その先の作品制作や仕事で使うとなると機能不足であった。 　そこで改良され登場したのがProcessingである。当時のMITのラボに所属していたCasey ReasとBen Fryにより、このProcessingが開発された。 　ProcessingはDesign By Numberとは違い、問題点であった色の種類がフルカラーで使用可能になり、表現できるサイズは100ピクセルの正方形よりも大きい範囲で指定できるようになった。他にも3次元のグラフィックスなど表現の幅が格段に変化していた。そのためある程度の技術を身に付けられれば実際の製品レベルのものを作ることが可能である。 　Processingは2001年の秋にMITのラボの授業で、インタラクションデザイン・インタラクティブアートを模索するためのツールとして使われた。初期のProcessingは、MITの研究者や授業参加者だけの少人数で使用されており、そのコミュニティーの中で機能追加やバグ修正等が行われてきた。 　このように様々な改良がなされ、2002年8月3日にαバージョンの公開がされた。Processingの公開後は、美術系の大学や工業系の学校でプログラミング教育の環境として多くの人に使われていた。 　その後Processingはβ版の公開がなされ、2008年11月にver.1.0をリリースした。2018年現在ではver.3.4となっている。 　またProcessingはオープンソースのソフトウェアであるため、ソースコードが公開されている。(オープンソース‥‥ソフトウェアの設計図に相当するソースコードを誰でも閲覧出来、そのソフトウェアの改良と再配布をライセンスの範囲内で行えるもののこと) 　そのため開発側が新機能の実装や改良が出来なくとも、他の人が開発することにより機能を追加していくことが可能である。このようにオープンソースになっていることにより外部の開発者による改良が可能なため、Processingは今現在もなお広がりを見せている。 引用…Built with Processing デザイン/アートのためのプログラミング入門　田中孝太郎・前川峻志

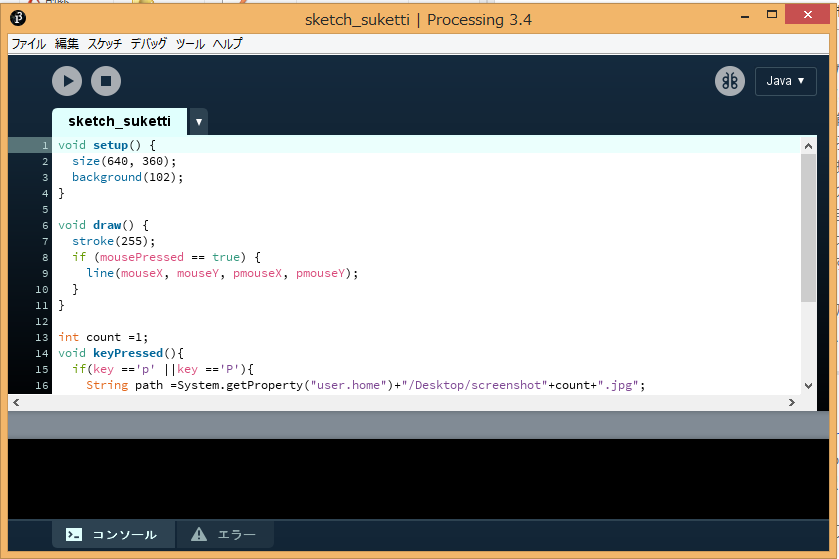
2-2.特徴

・開発環境の使いやすさ

Windows、Mac　OS 、Linux上で動かすことが出来、かつオープンソースであるプログラミング言語である。

　また機能面において、Processingの開発環境がとてもシンプルな画面構成になっており見やすく、初心者にも分かりやすい。

　プログラムを画面上に入力し、左上部にあるRunボタンを押せばウィンドウが新しく開き実行結果をすぐに確認することが可能である。入力ミス等で実行されなかったとしても、どの部分が間違っているかの指定や内容が報告される。



Processingの開発画面

・命令の直感性

　Processingはjavaをベースに作成されたプログラミング言語なので命令の書き方が似ている。

　また、processingの命令は目的の動作を英単語に置き換えたものが多いため覚えやすく、イメージしやすい。(例：直線を引くための関数…line、円を描くための関数…ellipse)

・ビジュアル表現

　Processingでは色の透明度を指定する命令があるため，ほかのプログラミング言語に比べ動きを視覚的に感じられるように描画しやすい。(例：影を付ける。動かしたい物の通った後を徐々に消していく。)

・周辺機器との連動

書いた命令を実行する際にマウスやキーボードに限らず、マイクやカメラ、ペン等様々な入力機械での実行・入力が可能である。

・外部コミュニティーの存在

　Processingには公式サイトに限らず、外部にも様々な関連コミュニティーが存在しており、自分の作品をWeb上に投稿し共有することが可能である。

　実際にどのようなものがあるのかというと、作品公開の場としてOpenProcessingというProcessingのスケッチを公開する投稿サイトが存在する。

このOpenProcessingには多くのスケッチが公開されており、その多くの作品のソースコードが公開されている。他にも公開するだけではなく他者の作品に対してお気に入り登録が出来、その作者をフォローし他の作品を閲覧可能である。またそのお気に入りのスケッチ作成者に対してコメントも送れるため他者や自分自身のプログラミング能力向上になるであろう。

OpenProcessing…https://www.openprocessing.org/

　ほかにも、Processingのスケッチ公開の場として写真共有サービスのFlickrや、動画を投稿できるサービスを持ったVimeoなどのWebサービスを使用することが出来る。

これらの多くはProcessingやProcessing.orgというタグが使用されているため、そこから様々な種類のスケッチが見つけられる。

Flickr…https://www.flickr.com/groups/processing/

Vimeo…https://vimeo.com/channels/processing

3.準備

3-1.ダウンロードから起動まで

まず、webブラウザにてprocessingの公式サイトである「<https://processing.org/>」

を検索する。…図1

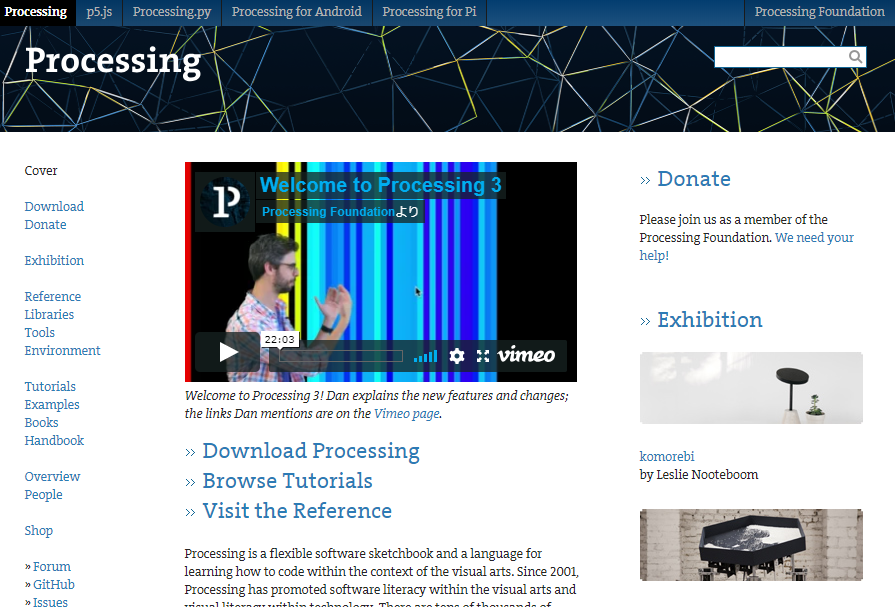


図1

検索出来図1の画面が表示されたら画面中央にある「Download　Processing」をクリック。

その後図2のような画面が表示されるので、自分のpcのosにあったものを選択し、ダウンロードを開始する。



図2

図3のようにpcのフォルダーからダウンロードした圧縮ファイル解凍する。解凍しフォルダーを開けるようになったら、その中にあるアプリケーションファイルである「processing」を選択し「すべて展開」を選択する。



図3

すべて展開終了後図4のように、さっき触っていたアプリケーションファイルの「processing」のアイコンが変化し、クリックすることにより起動が可能になる。



図4

3-2.基本操作

3-2-1画面詳細

<ファイル>

・新規( Ctrl + N )…現在の日付をもとに"sketch\_#######"という名前のファイルを作成する。

・開く( Ctrl + O )…自分のPC内から.pdeのファイルを選択しProcessingでコードを読み込む。 ・最近開いたファイル…直近で開いたファイル名が表示され選択することでコードを読み込む。

・スケッチブック( Ctrl + Shift + K )…ローカルコンピュータまたはネットワーク、スケッチブックのどこからでもスケッチを開くことが出来る。

・サンプル( Ctrl + Shift + O )…すでに１つの形として出来上がっている実行済みコードを読み込むことが出来る。

・閉じる( Ctrl + W )…現在のウィンドウを閉じる。

・保存( Ctrl + Shift + S )…現在のウィンドウで作られているものを上書き保存する。

・名前を付けて保存( Ctrl + Shift + S )…新規に名前を付け、指定した場所に保存する。

・アプリケーションとしてエクスポート( Ctrl + Shift + E )…プログラムしたものを.pdeではなく.exeとして保存する。ProcessingがインストールされていないPCでも実行が可能となる。

・ページ設定( Ctrl + Shift + P )…印刷するうえでのサイズ等を変更できる。

・印刷( Ctrl + P )…印刷の実行。 ・設定( Ctrl + . )…処理の仕組みを変更できる。

・終了( Ctrl + Q )…すべてのウィンドウを閉じる。

<編集>

・元に戻す( Ctrl + Z )…最後に入力したコマンドまたは最後に入力したエントリを元に戻す。

・やり直し( Ctrl + Y )…最後の元に戻すコマンドの操作を元に戻す。このオプションは、[元に戻す]操作が既に行われている場合にのみ使用できる。

・切り取り( Ctrl + X )…選択したテキストを削除し、クリップボードにコピーする。

・コピー( Ctrl + C )…選択したテキストをクリップボードにコピーする。

・Htmlとしてコピー( Ctrl + V )…HTMLのテキストの一部としてコピーされる。

・貼り付け( Ctrl + V )…クリップボードの内容を指定した場所に張り付ける。

・すべて選択( Ctrl + A )…現在書かれているテキストをすべて指定する。

・自動フォーマット( Ctrl + T )…プログラムのインデントを整形する。

・コメント/アンコメント( Ctrl + / )…指定した場所をコメントアウト、もしくはすでにコメントアウトされている場合はコメントアウトしていない状態にする。

・→インデントを増やす( Ctrl + [ )…指定された先頭部分に空白を増やす。

・←インデントを減らす( Ctrl + ] )…指定された先頭部分の空白を減らす。

・検索( Ctrl + F )…テキストエディタで開いているファイル内のテキスト文字列の出現箇所を検索し、それを別のテキストに置き換える。

・次を探す( ctrl + G )…キストエディタで開いているファイル内のテキスト文字列の次の出現箇所を検索する。

3-2-2.命令入力

3-2-3.プログラムのデバック

デバッグとは、エラーやバグを直してプログラムを正常に動くよう修正することである。

Processingではデバッグを効率的に行うために以下のようなポイントに意識すると良い。

1…エラー内容によるエラーの発見

Processingではエラーが発生すると、エラー内容が画面下部のメッセージ行に表示されると同時にエラーになったプログラムの部分にカーソルが移動し黄色で示される。基本的にこの情報の付近でエラーがあることが多い。またProcessingのエディタではプログラムが正しいと文字色が変化したり、()にカーソルを移動させると対応した()の位置が表示される機能があるので、ミスをしたタイミングで視覚的に間違いを発見するヒントになる。

2…コメントアウトする方法

エラーを探していくときに、怪しいと思う部分を無効化する方法である。具体的にどのようにするかというと自分で書いたプログラムの怪しいと思う部分の先頭に//と付けその部分にエラーとの関連性があるのかを見極めていく。このように、その部分がプログラムのエラーに関係あるのかというテストをするデバッグをコメントアウトデバッグという。ここをはずせばうまく動くというプログラムの弱点部分を絞って見つけ出せればあとはその弱点を重点的に調べられる。行をコメントアウトしてプログラムが動くようになる又は別のエラーが出たら、今コメントアウトした行に何か原因がある。それでも動かなければ、別の行をコメントアウトしてみる。

3…数値の変更

プログラムに使っている数値を、一時的に大きくしたり小さくしたりして様子を見てみるという方法である。数値を変更して動作をチェックすれば、その数値が原因で異常を起こしているのか、それとも数字を処理しているプログラムに原因があるのかが分かる。また、他の場所の処理の結果を使うような部分を絶対に動く数値に変更し実行してみるとその処理の前か後にバグがあるかが分かる。

4.お絵かきアプリ作成

4-1.描画関係

4-2.カラーパレット

4-2-1.カラーパレットの原画

int squareOneSide = 800;

float r = squareOneSide\*3/4;

boolean controlDown = false;

boolean shiftDown = false;

void setup() {

size(1000, 1000);

background(255);

colorMode(HSB, 360, 100, 100);

colorPallet();

colorPalletDraw();

}

void draw() {

/\*

if (mousePressed){

strokeWeight(10);

line(mouseX, mouseY, pmouseX, pmouseY);

\*/

if(mousePressed){

color c = get(mouseX, mouseY);

println("color", hue(c), saturation(c), brightness(c));

// println("zahyou", mouseX, mouseY);

}

}

void colorPallet(){

translate(width/2, height/2);

rect(- squareOneSide/2, - squareOneSide/2, squareOneSide, squareOneSide);

ellipse(0, 0, squareOneSide, squareOneSide);

// ellipse(0, 0, r, r);

rect(- r\*(1/sqrt(2))/2, - r\*(1/sqrt(2))/2, r\*(1/sqrt(2)), r\*(1/sqrt(2)));

colorPallet02();

}

void colorPalletDraw(){

for(int i = 0; i < 360; i++){

strokeWeight(8);

float rad = TWO\_PI/360;

stroke(getColor(i, 100, 100));

line(cos(rad\*i)\*r/2, sin(rad\*i)\*r/2, cos(rad\*i)\*squareOneSide/2, sin(rad\*i)\*squareOneSide/2);

}

}

color getColor(int i, int j, int k){

return color(i, j, k);

}

void colorPallet02(){

int s = 50; // 何個に分けるか

float x = r\*(1/sqrt(2))/s; // 1マスの辺

float y = -r\*(1/sqrt(2))/2; // スタートの座標

noStroke();

for(int i = 0; i < s; i++){

for(int j = 0; j < s; j++){

int aa = 100\*(i+1)/s;

int bb = 100\*(j+1)/s;

fill(getColor(0, abs(100-aa), abs(100-bb)));

rect(y+x\*i, y+x\*j, x, x); // 縦書いて横に伸ばす

// println("S\_S", i, "B\_B", j);

// println("S", 100\*(i+1)/s, "B", 100\*(j+1)/s);

}

}

}

void keyPressed() {

if (key=='s') {

saveFrame("image####.png");

}

}

説明…

4-2-2.カラーパレットの組み込み

//5つのウィンドウを宣言

PGraphics canvas, pallet;

// 宣言してから代入　finalこの変数の値を今後変えない

final float golden\_ratio = 1.618;

//円の半径の関係性

final float circle\_ratio = 0.75;

//いま抽出している色（初期色）

color selectColor = color(0, 0, 0);

//それぞれの辺の長さをとるため、親を変えると連動する

int size\_height = 1000; //仕方ない

int lv01\_golden = round(size\_height\*(golden\_ratio-1)); // 618

int lv02\_golden = round(lv01\_golden\*(golden\_ratio-1)); // 382

int lv03\_golden = round(lv02\_golden\*(golden\_ratio-1)); // 236

//

void setup(){

size(1618, 1000);

//HSB、何パターンつくるか、彩度、明度

colorMode(HSB, 360, 100, 100);

//

makeWindow();

//背景色白(0,0,100)、キャンバスウィンドウの背景色を

makeWindow\_setup01(canvas, color(0, 0, 100));

//makecolorpallet palletウィンドウにカラーパレットを作成する

makeColorpallet(pallet);

makeWindow\_show();

}

//ウィンドサイズの設定

void makeWindow(){

canvas = createGraphics(height, height);

pallet = createGraphics(lv02\_golden, lv02\_golden);

}

//背景色を作成する関数

void makeWindow\_setup01(PGraphics ff, color base\_color){

ff.beginDraw();

ff.colorMode(HSB, 360, 100, 100);

ff.background(base\_color);

ff.endDraw();

}

void makeColorpallet(PGraphics gg){

//せんたくした図形の一辺の値をとる

int squareOneSide = gg.height;

//大きいほうの円の半径の長さ指定

float large\_r = squareOneSide/2;

//小さいほうの円の半径の値を設定

float small\_r = (large\_r)\*circle\_ratio;

gg.beginDraw();

gg.colorMode(HSB, 360, 100, 100);

//(0,0)で座標を指定

gg.pushMatrix();

//中心座標(0,0)をウィンドウの中心に来るように設定

gg.translate(large\_r, large\_r);

//draw01 外のカラーパレット、外の背景　draw2　中のカラーパレット　PGraphics ウィンドウを表す型

makeColorpallet\_draw01(gg, large\_r, small\_r);

makeColorpallet\_draw02(gg, small\_r, 0);

gg.endDraw();

}

void makeColorpallet\_draw01(PGraphics gg, float r01, float r02){

float on\_rad = TWO\_PI/360;

gg.noStroke();

gg.fill(200);

gg.rect(-r01, -r01, r01\*2, r01\*2);

gg.noStroke();

for(int i = 0; i < 360; i++){

gg.fill(i, 100, 100);

//おうぎ形arc(中心座標x,y,直径,直径,何度,～何度)

gg.arc(0, 0, r01\*2, r01\*2, i\*on\_rad, (i+1)\*on\_rad);

}

//中心背景塗りつぶし

gg.fill(0, 0, 100);

gg.ellipse(0, 0, r02\*2, r02\*2);

}

//color\_hue 選択されている色の色彩変更

void makeColorpallet\_draw02(PGraphics gg, float r02, float color\_hue){

//中の色彩のしきりを50\*50で表現

int separate = 50;

//分けた1マスの1辺の長さを出している

float separate\_squareOneSide = r02\*(1/sqrt(2))\*2/separate;

float start\_separate\_coordinate = -r02\*(1/sqrt(2)); // スタートの座標

gg.noStroke();

//i=横 彩度　j=縦 明度

for(int i = 0; i < separate; i++){

for(int j = 0; j < separate; j++){

//0～100の組み合わせ表現を半分のサイズにしている　100種類を50種類にしているから

float change\_saturation = 100\*(i+1)/separate;

float change\_brightness = 100\*(j+0)/separate;

//分けた図形のなかをfillで指定した条件で塗りつぶすHSB

gg.fill(color\_hue, 100-change\_saturation, 100-change\_brightness);

//分けた小さい正方形の座標をずらしながら書いていく　　+1した理由…枠なしで書いているため隙間が出来る

gg.rect(start\_separate\_coordinate + separate\_squareOneSide\*i, start\_separate\_coordinate + separate\_squareOneSide\*j, separate\_squareOneSide + 1, separate\_squareOneSide + 1); // 縦書いて横に伸ばす

}

}

}

//ウィンドウの配置する場所を指定

void makeWindow\_show(){

image(canvas, 0, 0);

image(pallet, height, lv01\_golden);

}

void draw(){

float gg\_pallet\_center\_x = height + lv02\_golden/2;

float gg\_pallet\_center\_y = lv01\_golden + lv02\_golden/2;

//1/sqrt(2))\*circle\_ratio\*lv02\_golden/2 pallet内部の正方形の一辺の半分の長さ

float gg\_pallet\_squareOneSide = (1/sqrt(2))\*circle\_ratio\*lv02\_golden/2;

if (mousePressed){

float x = mouseX, y = mouseY;

// カラーパレットの色相

if (gg\_pallet\_center\_x - gg\_pallet\_squareOneSide < x

&& x < gg\_pallet\_center\_x + gg\_pallet\_squareOneSide

&& gg\_pallet\_center\_y - gg\_pallet\_squareOneSide < y

&& y < gg\_pallet\_center\_y + gg\_pallet\_squareOneSide){

//getそこのデータの色情報を持ってくる　round 四捨五入　マウスで座標を取ると小数もすうちとして取ってしまうため

selectColor = get(round(x), round(y));

}

// カラーパレットの彩度と明度　dist 座標から座標の距離　円の領域指定

if (circle\_ratio\*lv02\_golden/2 < dist(x, y, gg\_pallet\_center\_x, gg\_pallet\_center\_y)

&& dist(x, y, gg\_pallet\_center\_x, gg\_pallet\_center\_y) < lv02\_golden/2)

{selectColor = get(round(x), round(y));

pallet.beginDraw();

//中心座標(0,0)をウィンドウの中心に来るように設定

pallet.translate(lv02\_golden/2, lv02\_golden/2);

//(どのウィンドウに書くかを指定,数値を上の関数と同じにするため,取った色情報の色彩)

makeColorpallet\_draw02(pallet, circle\_ratio\*lv02\_golden/2, hue(selectColor));

pallet.endDraw();

image(pallet, height, lv01\_golden);

}

// キャンバスだけ書けるように条件付け

if (x < height && y <height){

strokeWeight(10);

stroke(selectColor);

line(mouseX, mouseY, pmouseX, pmouseY);

}

}

}

説明…本文にかいてしまったのでまた抽出してまとめる…

4-3.

5.Processing.jsの使用

Processing.jsとは

　Webブラウザ上でProcessingのプログラムを解釈し、グラフィックを描画することの出来るJavaScriptで作られたProcessing互換ライブラリである。 Processing.jsはグラフィックの描画にCanvasを導入しているという特徴がある。FirefoxやGoogle Chrome、Safari、OperaなどはこのCanvas要素の機能を持っているためWebブラウザで表示が可能である。しかしIEではこのCanvas要素が標準装備されていないため、IE用のCanvas互換ライブラリをしなければProcessing.jsのグラフィックを表示することが出来ないので注意が必要になる。

Processing.jsのダウンロード

　Processing.jsは、拡張子が.pdeであるProcessingのような実行ファイルとは違い、HTMLに読み込む.jsであるJavaScriptファイルになっている。

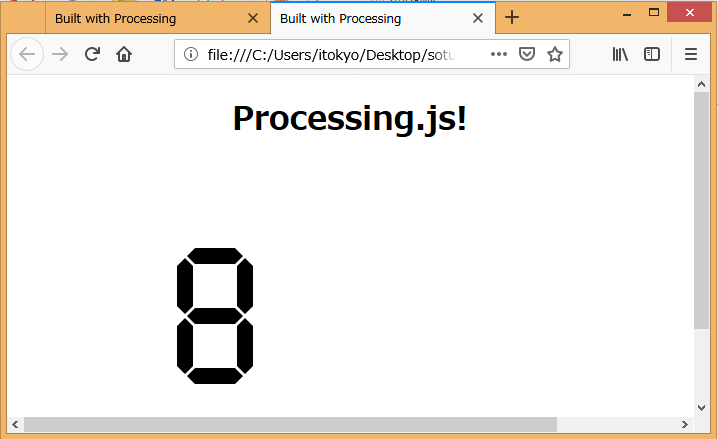
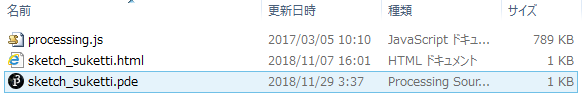
Processing.jsはProcessingの公式サイトのダウンロードページ(http://processingjs.org/download/) からダウンロードすることが可能である。



Processing.jsの使い方

　Processing.jsでは、実際にWeb上で表示させたいHTMLファイルの中にProcessingのプログラムを書かずに、ファイルを読み込み実行することが可能である。上記のようにProcessingファイルをHTML内で読み込ませるためには、Processing.jsを読み込んだHTML内のcanvasタグにdata-srcという属性でファイルのパスを書く。そうすることでcanvasタグのdata-src属性の情報からプログラムファイルを読み込み実行し、Web上に作成したProcessingのファイルを表示することが出来る。

　指定した後、Processing.jsと作成した.pdeファイル、HTMLファイルを同じフォルダーに収納しHLMLファイルを開けばWeb上に作成したProcessingの作品が表示される。



6.今後の課題

7.参考文献