卒業論文

Processingによるアプリケーションの研究

経営情報学部　経営情報学科

BG15012 伊藤恭兵

目次.

1.研究目的・研究背景

2.Processingとは

2-1.Processingの歴史

2-2.Processingの特徴

3.環境構築

　3-1.ダウンロード方法と起動方法

　3-2.基本操作

　　　3-2-1.画面詳細

　　　3-2-2.プログラムのデバック

4.ペイントアプリ作成

4-1.今回使用した関数や変数、型の一覧

4-1-1.使用した型

4-1-2.使用した変数

4-1-3.使用した関数

4-2.カラーパレットの実装

4-3.戻る、進む機能の実装

4-4.消しゴム、ペンの太さ変更機能の実装

4-5.保存機能の実装

4-6.キャンバスの新規作成機能の実装

4-7.タブアイコンの作成と描画

5.Processing.jsの使用

6.今後の課題

参考文献

付録

謝辞

1. 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、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イラスト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イラスト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/お絵かきの森)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、卒業論文を書いく。

2.Processingとは

2-1.Processingの歴史

　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。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.Processingの特徴

・開発環境の使いやすさ

Processingは、Windows、MacOS、iOS、Linux上で動かすことができ、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。

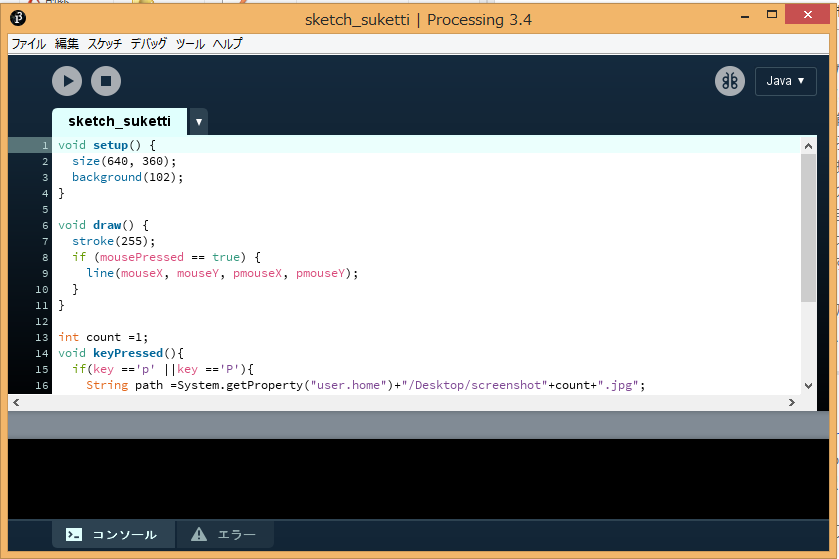


図1.Processingの統合開発環境

・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)

・ビジュアル表現が得意

　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。)

・周辺機器との連動が可能

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が存在し、情報のシェアが可能

　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2

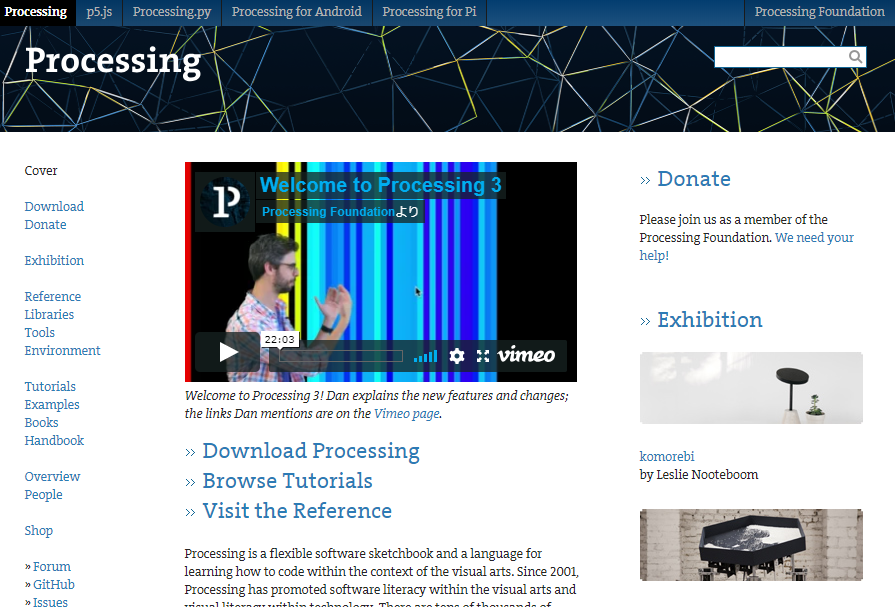


図2.Processingの公式サイト

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」をクリックする。

　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3.Processing公式サイトのダウンロード画面

図4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4.解凍後のフォルダー内

すべて展開終了後図5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5.ファイル展開後のフォルダー内

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.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。

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。

コメントアウトによる機能の一時的な停止

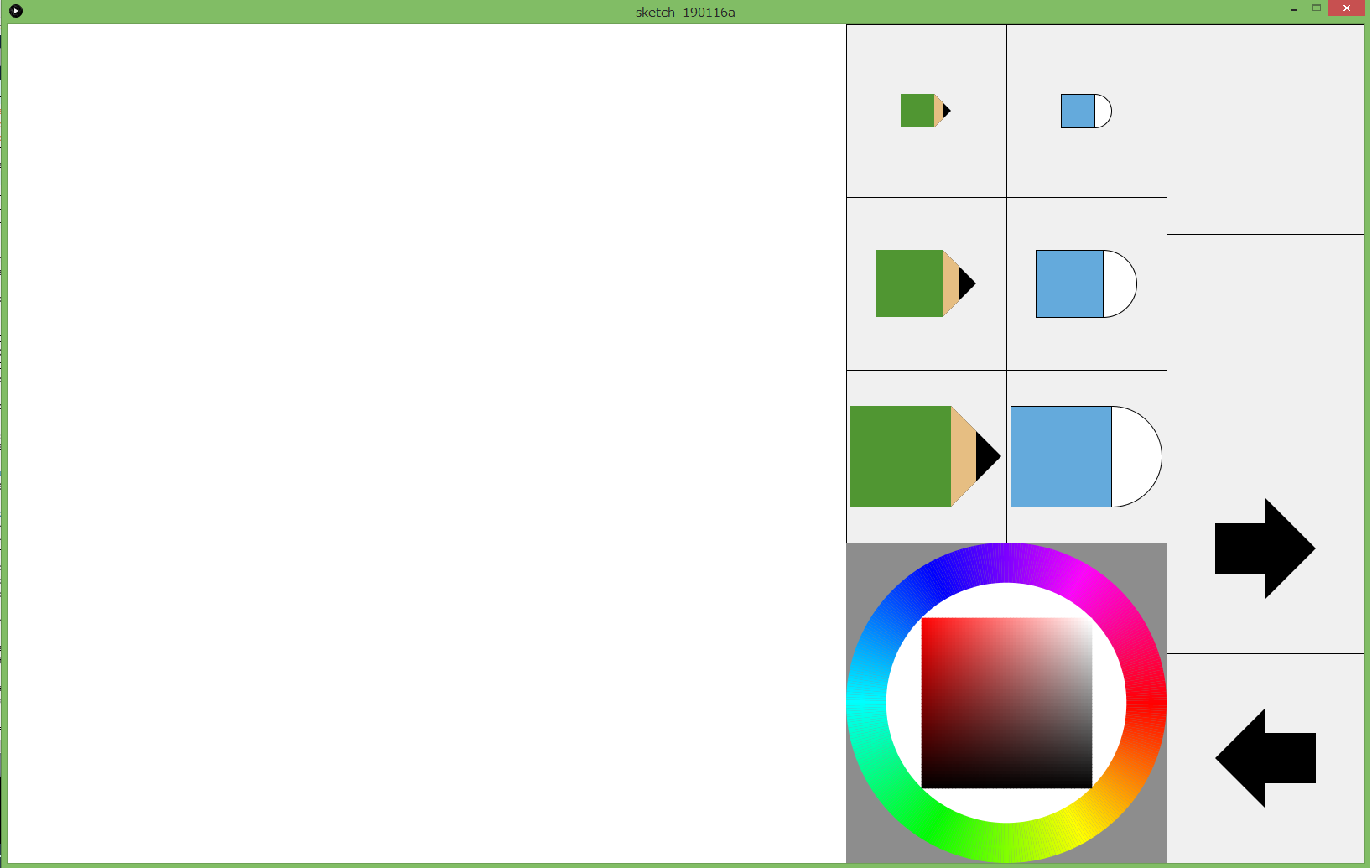
エラーを探していくときに、怪しいと思う部分を無効化する方法である。具体的にどのようにするかというと自分で書いたプログラムの怪しいと思う部分の先頭に//と付けその部分にエラーとの関連性があるのかを見極めていく。このように、その部分がプログラムのエラーに関係あるのかというテストをするデバッグをコメントアウトデバッグという。ここをはずせばうまく動くというプログラムの弱点部分を絞って見つけ出せればあとはその弱点を重点的に調べられる。行をコメントアウトしてプログラムが動くようになる又は別のエラーが出たら、今コメントアウトした行に何か原因がある。それでも動かなければ、別の行をコメントアウトしてみる。

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.ペイントアプリ作成

今回作成したペイントアプリの完成図は図6である。以下で詳細を説明する。



領域8

領域7

Palletウィンドウ

領域3

Canvasウィンドウ

領域2

領域1

領域6

領域5

領域4

　　　　　　　　　　　　　　　図6.完成図

4-1.今回使用した関数や変数、型の一覧

4-1-1.使用した型

今回使用したデータ型は以下の5個である。

|  |  |
| --- | --- |
| int | 整数値を表すデータの型 |
| float | 小数値を表すデータの型 |
| boolean | 真偽値を表すデータの型 |
| color | 色を表すデータの型 |
| PImage | 画像を表すデータの型 |
| string | 文字列を表すでデータ型 |

4-1-2.使用した変数

今回使用した変数は以下の60個である。

そのうち既存の変数は以下の9個である

|  |  |
| --- | --- |
| width | int型でウィンドウの横幅(x座標) |
| height | int型でウィンドウの高さ(y座標) |
| mouseX | float型で現在のフレームのマウスカーソルのx座標 |
| mouseY | float型で現在のフレームのマウスカーソルのy座標 |
| pmouseX | float型で前のフレームでのマウスカーソルのx座標 |
| pmouseY | float型で前のフレームでのマウスカーソルのy座標 |
| mousePressed | boolean型でマウスボタンが押されていればtrue、押されていなければfalse |
| key | string型でキーの文字を表す変数 |
| keyCode | int型でキーを数字で表す変数 |

自作した変数は以下の51個である。

|  |  |
| --- | --- |
| golden\_ratio | float型でウィンドウサイズを設定する上で使用する変数(今回は1.618) |
| size\_height | ウィンドウの高さを指定する変数 |
| lv01\_golden | int型でウィンドウサイズを設定する上で使用する変数(今回は618) |
| lv02\_golden | int型でウィンドウサイズを設定する上で使用する変数(今回は382) |
| lv03\_golden | int型でウィンドウサイズを設定する上で使用する変数(今回は236) |
| gg、ff、tt | 引数として使うPGraphics型の変数 |
| circle\_ratio | float型でカラーパレットを設定する上で使用する変数(今回は0.75) |
| selectColor | color型で選択した色を格納する変数(初期値は0, 0, 0) |
| squareOneSide | int型でpalletウィンドウの一辺をあ表す変数 |
| large\_r | float型でカラーパレットでの色相を選択する部分(ドーナツ型部分)の外側の円の半径 |
| small\_r | float型でカラーパレットでの色相を選択する部分(ドーナツ型部分)の内側の円の半径 |
| on\_rad | float型で1ラジアンの値を代入する変数 |
| r01 | 引数として使用するfloat型の変数(主にlarge\_rを代入して使用) |
| r02 | 引数として使用するfloat型の変数(主にsmall\_rを代入して使用) |
| color\_hue | 引数として使用するfloat型の変数(主にselectColorの色相を代入して使用) |
| separate | int型でカラーパレット(正方形部分)を縦と横を何個に分けるか代入する変数(今回は50を代入) |
| separate\_squareOneSide | float型でカラーパレット(正方形部分)を分割した際の１辺の長さを取る変数 |
| start\_separate\_coordinate | float型でカラーパレット(正方形部分)を分割した際に出来る図形を書くうえで座標指定するときに使う長さ |
| change\_saturation | float型でカラーパレット(正方形部分)を分割し彩度の変化具合を調整する変数 |
| change\_brightness | float型でカラーパレット(正方形部分)を分割し明度の変化具合を調整する変数 |
| gg\_pallet\_center\_x | float型でカラーパレット(ドーナツ型部分)での座標情報から色情報を取り出す際に、その座標を感知するのに使用する変数 |
| gg\_pallet\_center\_y | float型でカラーパレット(ドーナツ型部分)での座標情報から色情報を取り出す際に、その座標を感知するのに使用する変数 |
| gg\_pallet\_halfSquareOneSide | float型でカラーパレット(正方形部分)での座標情報から色情報を取り出す際に、その座標を指定するのに使用する変数 |
| stepImg | PImage型の配列 |
| stepImgNumber | int型で配列の個数を何個作るかを決めるのに使用する変数(保存する画像数、今回は15) |
| backCount | int型で戻ることのできる回数を数える変数 |
| forwardCount | int型で進むことのできる回数を数える変数 |
| stepImgCount | int型で取り出す画像の配列番号を指定する変数 |
| changeCanvas | boolean型でキャンバス内が変化したかを判断する変数 |
| pen\_size | float型でペンの太さを指定する変数 |
| ers\_size | float型で消しゴムの太さを指定する変数 |
| ers\_change | boolean型で消しゴムを使用できる状態かを判断する変数 |
| downCtrl | boolean型でCtrlを押しているかを判断する変数 |
| downShift | boolean型でShiftを押しているかを判断する変数 |
| downA | boolean型でaを押しているかを判断する変数 |
| dayY | int型で西暦を代入する変数 |
| dayM | int型で月を代入する変数 |
| dayD | int型で日付を代入する変数 |
| timeH | int型で時間を代入する変数 |
| timeM | int型で分を代入する変数 |
| timeS | int型で秒を代入する変数 |
| dayString | String型で西暦、月、日付をまとめる変数 |
| timeString | String型で時間、分、秒をまとめる半数 |
| separate\_count | int型でtab01、tab02、tab03がそれぞれ何個に分けるか指定する変数 |
| arrowSize | int型で戻る進む機能を表すアイコン(矢印アイコン)の大きさを指定する変数 |
| arrowStart\_x | float型で戻る進む機能を表すアイコン(矢印アイコン)の位置を決定するx座標 |
| arrowStart\_y | float型で戻る進む機能を表すアイコン(矢印アイコン)の位置を決定するy座標 |
| center\_x | 引数として使用するfloat型の鉛筆のアイコンと消しゴムのアイコンの中心を表すx座標 |
| center\_y | 引数として使用する鉛筆のアイコンと消しゴムのアイコンの中心を表すy座標 |
| pencilSize | 引数として使用するint型で鉛筆アイコンの大きさを指定する変数 |
| eraserSize | 引数として使用するint型で消しゴムアイコンの大きさを指定する変数 |

4-1-3.使用した関数

今回使用した関数は以下の56個である。

そのうち既存の関数は38個である。

|  |  |
| --- | --- |
| setup() | このファイルが開かれた時に一度実行する関数 |
| draw() | 1フレームごとに繰り返し実行する関数 |
| colorMode() | 色のモードや、色の段階を指定する関数 |
| color() | 色データを作成する関数 |
| createGraphics() | PGraphics型のウィンドウを作成する関数 |
| if() | 条件を指定し、動きを実行させる関数 |
| for() | 繰り返す回数を指定し、動きを実行させる関数 |
| background() | 背景色を指定する関数 |
| stroke() | ペンの色を指定する関数 |
| noStroke() | ペンの描画をしない関数 |
| strokeWeight() | ペンの太さを指定する関数 |
| fill() | 指定した色で図形の中を塗りつぶす関数 |
| rect() | 指定した座標からx軸、y軸に、指定した長さ分の線を書き長方形を描写する関数 |
| image() | 指定したイメージを指定した座標に表示する関数 |
| get() | 指定した範囲のデータをPImage型で取得する関数 |
| beginDraw() | PGraphics型のデータに変更を加える上での始まりを表す関数 |
| endDraw() | PGraphics型のデータに変更を加える上での終わりを表す関数 |
| pushMatrix() | 原点座標を一時保存する関数 |
| translate() | 原点座標を変更する関数 |
| arc() | おうぎ形を書く関数 |
| ellipse() | 円、楕円を書く関数 |
| triangle() | 三角形を書く関数 |
| quad() | 四角形を書く関数 |
| line() | 線を書く関数 |
| mouseDragged() | ボタンが押された状態でマウスが移動したときに呼び出される関数 |
| mousePressed() | マウスボタンが押されると呼び出される関数 |
| mouseReleased() | マウスボタンが離れたときに呼び出される関数 |
| mouseClicked() | マウスボタンが押されてから離されたときに呼び出される関数 |
| println() | ()内に書かれた内容を統合開発環境のコンソールに出力して改行する関数 |
| keyPressed() | キーを押した時に実行する関数 |
| keyReleased() | キーを離した時に実行する関数 |
| year() | 現在の西暦を取得する関数 |
| month() | 現在の月を取得する関数 |
| day() | 現在の日付を取得する関数 |
| hour() | 現在の時間を取得する関数 |
| minute() | 現在の分を取得する関数 |
| second() | 現在の秒を取得する関数 |
| nf() | 指定した桁数で表現する関数 |

自作した関数は以下の18個である。

|  |  |
| --- | --- |
| makeWindow() | ウィンドウサイズの設定をする関数 |
| makeWindow\_setup02() | 背景色を作成する関数 |
| makeColorpallet() | カラーパレットを作成する関数 |
| makeColorpallet\_draw01() | カラーパレットでの色相を選択する部分(今回だとドーナツ型部分)を作成する関数 |
| makeColorpallet\_draw02() | カラーパレットでの明度、彩度を選択する部分(今回だと正方形部分)を作成する関数 |
| makeWindow\_show() | 自作したウィンドウを指定の場所に描画する関数 |
| make\_stepImg\_setup() | PImage型の配列に初期canvasウィンドウを代入する関数 |
| save\_stepImg() | PImage型の配列に変更のあったcanvasウィンドウを代入する関数 |
| undo() | 戻る機能を実行する関数 |
| redo() | 進む機能を実行する関数 |
| stepImgCount\_add() | PImage型の配列を参照するときに使う配列番号を1足す関数 |
| stepImgCount\_sub() | PImage型の配列を参照するときに使う配列番号を1引く関数 |
| show\_stepImg() | PImage型の配列から配列番号を使って、該当する画像をcanvasウィンドウに描画させる関数 |
| maketab01() | 鉛筆のアイコンを大きさを変えながらtab01ウィンドウに描画する関数 |
| maketab02() | 消しゴムのアイコンを大きさを変えながらtab02ウィンドウに描画する関数 |
| maketab03() | 戻る進む機能を表すアイコン(矢印アイコン)をtab03に描画する関数 |
| pencill() | 鉛筆のアイコンを作る関数 |
| eraser() | 消しゴムのアイコンを作る関数 |

4-2.カラーパレットの実装

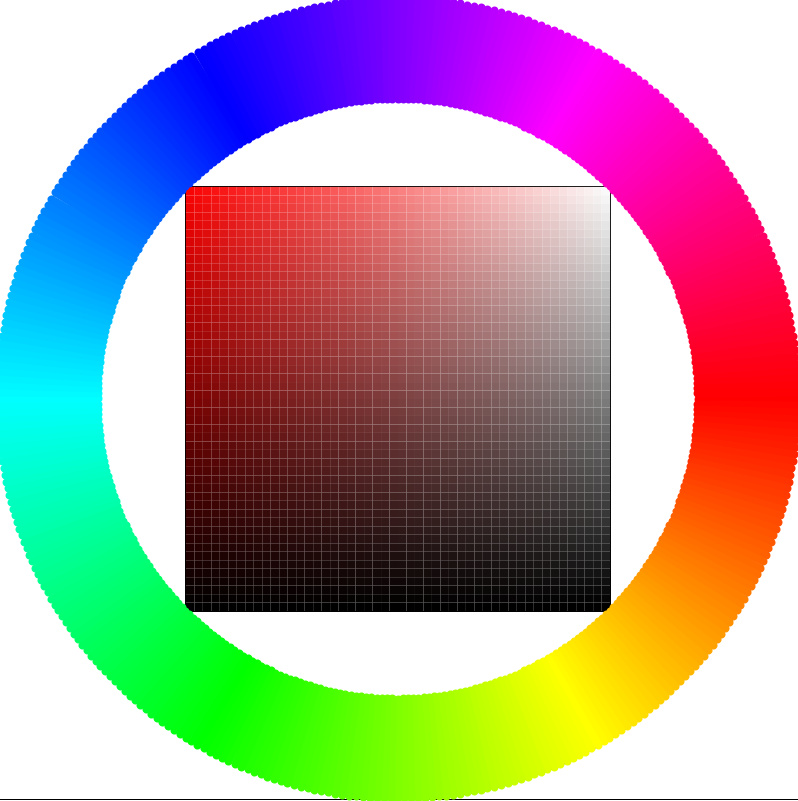


　　 図7.カラーパレットのイメージ図

|  |
| --- |
| PGraphics pallet, tab01, tab02, tab03; |

palletというウィンドウをPgraphics型で宣言する。

canvasウィンドウは今回のイラストアプリでのイラストを描く部分(紙)であり、palletウィンドウは色を選択することのできるカラーパレットの部分である。

|  |
| --- |
| final float golden\_ratio = 1.618;  final float circle\_ratio = 0.75;  int size\_height = 1000;  int lv01\_golden = round(size\_height\*(golden\_ratio-1));  int lv02\_golden = round(lv01\_golden\*(golden\_ratio-1));  int lv03\_golden = round(lv02\_golden\*(golden\_ratio-1)); |

ここでは、それぞれのウィンドウを作成する上で必要となってくる比の値を代入している。今回は黄金比をもとにウィンドウの大きさを決定した。circle\_ratioはカラーパレット(ドーナツ型部分)の外側の円と内側の円の比として使う。

lv01\_golden=618、lv02\_golden=382、lv03\_golden=236の値をとる。

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

color selectColorは、型がcolor、変数がselectColorであり、複数の関数で値を使えるようにしたいのでここで記述する。

|  |
| --- |
| void setup(){  size(1618, 1000);    colorMode(HSB, 360, 100, 100);  makeWindow();  makeWindow\_setup02(color(0, 0, 100));  makeColorpallet(pallet);  (省略)  makeWindow\_show();  (省略)  } |

size()全体のウィンドウサイズを横1618×縦1000で作成している。

colorMode()で色の指定をするときに使う値をRGBからHSBに変更している。カラーパレットで色を取る際にHSBのほうが直観的に色を選択できるため、変更した。

|  |
| --- |
| void makeWindow(){  pallet = createGraphics(lv02\_golden, lv02\_golden);  (省略)  } |

makeWindow()という自作関数ではpalletウィンドウを382×382で取るように指定した。

|  |
| --- |
| void makeWindow\_setup02(color base\_color){  noStroke();  fill(base\_color);  rect(0, 0, height, height);  } |

makeWindow\_setup02()という自作関数では背景色を選択する関数を作っている。makeWindow\_setup02()を実行する場合は、color base\_colorに当たる部分にはcolor(0, 0, 100)という色情報を入れることによりcanvasウィンドウの背景色を白に設定することができる。

|  |
| --- |
| void makeColorpallet(PGraphics gg){  int squareOneSide = gg.height;  float large\_r = squareOneSide/2;  float small\_r = (large\_r)\*circle\_ratio;    gg.beginDraw();  gg.colorMode(HSB, 360, 100, 100);  gg.pushMatrix();  gg.translate(large\_r, large\_r);  makeColorpallet\_draw01(gg, large\_r, small\_r);  makeColorpallet\_draw02(gg, small\_r, 0);  gg.endDraw();  } |

makeColorpallet()という自作関数ではカラーパレットの色相を変化させる部分(ドーナツ型部分)と、彩度・明度を変化させる部分(正方形部分)についてのプログラムを行っている。

int squareOneSideはpalletウィンドウの1辺の長さ、float large\_rはドーナツ型部分の外側の円の半径、float small\_rはドーナツ型部分の内側の円の半径の値を示している。

beginDraw()とはPGraphics型の変数gg(makeColorpallet()を実行するときはpalletウィンドウが代入)に変更を加えるときに実行する関数である。

実際にどのような変更を加えるかというと、カラーモード、ウィンドウの左上の原点座標を一時保存、原点座標をpalletウィンドウの中心に変更、makeColorpallet\_draw01()で色相を変化させる部分の図形について(ドーナツ型部分)、makeColorpallet\_draw02()で彩度・明度を変化させる部分の図形について(正方形部分)書かれている。

では具体的にmakeColorpallet\_draw01()とmakeColorpallet\_draw02()を見ていく。

|  |
| --- |
| 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);  gg.arc(0, 0, r01\*2, r01\*2, i\*on\_rad, (i+1)\*on\_rad);  }  gg.fill(0, 0, 100);  gg.ellipse(0, 0, r02\*2, r02\*2);  } |

　makeColorpallet\_draw01()という自作関数はドーナツ型部分の図形がどのように書かれているのかがプログラムされている。

どのように書かれているかというと、まずは中心角が1度のおうぎ形を作成し、ある色相で塗りつぶす。それを360回繰り返し、360種類の色を表現したカラーパレット(ドーナツ型部分)を作成した。

では詳細を見ていこう。

まずはfloat on\_radで中心角の大きさの変数を作成する。今回は上でも書いた通り360分割したいため中心角の数字が1になるようTWO\_PI/360と置かれている。

noStroke()では、rect(-r01, -r01, r01\*2, r01\*2)で書いている四角形(palletウィンドウと同じ大きさ)の枠線を消している。

|  |
| --- |
| gg.fill(200);  gg.rect(-r01, -r01, r01\*2, r01\*2); |

ここでは、ドーナツ型部分の後ろの余白(背景)を灰色で塗りつぶしている。

noStroke()では今後書くおうぎ形の枠線を消し、色選択での見かけ上での境界線をなくしている。

|  |
| --- |
| for(int i = 0; i < 360; i++){  gg.fill(i, 100, 100);  gg.arc(0, 0, r01\*2, r01\*2, i\*on\_rad, (i+1)\*on\_rad);  } |

ここでは0～1度、1～2度……359～360といったように1度ずつ、かつ色相を1つずつずらし描画している。

|  |
| --- |
| gg.fill(0, 0, 100);  gg.ellipse(0, 0, r02\*2, r02\*2); |

ここでは、中心に白で塗りつぶした円を描画することにより見かけ上カラーパレットがドーナツ型になる。

　次にmakeColorpallet\_draw02()を見ていく。

|  |
| --- |
| void makeColorpallet\_draw02(PGraphics gg, float r02, float color\_hue){  int separate = 50;  float separate\_squareOneSide = r02\*(1/sqrt(2))\*2/separate;  float start\_separate\_coordinate = -r02\*(1/sqrt(2));  gg.noStroke();  for(int i = 0; i < separate; i++){  for(int j = 0; j < separate; j++){  float change\_saturation = 100\*(i+1)/separate;  float change\_brightness = 100\*(j+0)/separate;  gg.fill(color\_hue, 100-change\_saturation, 100-change\_brightness);  gg.rect(start\_separate\_coordinate + separate\_squareOneSide\*i, start\_separate\_coordinate + separate\_squareOneSide\*j, separate\_squareOneSide + 1, separate\_squareOneSide + 1);  }  }  } |

void makeColorpallet\_draw02()という自作関数は正方形部分の図形がどのように書かれているのかがプログラムされている。

まずint separateではカラーパレット(正方形部分)をどれだけの個数で区切るのかを表している。今回は50×50にしてあるが下の関数がこれを基準に作成しているのでint separate = 5;に変えれば5×5といったように変更が可能である。

float separate\_squareOneSideではカラーパレット(正方形部分)をint separateで入れた数分分け、その分けて出来た小さい正方形の1辺の長さを示している。またstart\_separate\_coordinateでは、カラーパレット(正方形部分)の1辺の半分の長さを示している。

この二つの値は、int separateの数字によって分けられて出来た小さい正方形を配置する座標情報(gg.rect(start\_separate\_coordinate + 省略)で使用する。

どのように使われているかというと、for文を使いカラーパレットの正方形部分を左上から順に描画している。(今回だと縦1列50個の箱を50列分)

書く場所を1つずらすとともにfill()ではそれに対応する色を1つずつ変化させ、彩度・明度の違いを作っている。

|  |
| --- |
| void makeWindow\_show(){  image(pallet, height, lv01\_golden);  (省略)  } |

makeWindow\_show()の自作関数では今作成したpalletウィンドウをどこに配置するのかを設定している。

今回だとpalletは(1000,618)から配置され描画されている。

ここまででキャンバス、カラーパレットの描画・配置は完了したので、ここからはどのような座標で色を選択できるのか、その選んだ色で書くことができるのかという範囲指定をしていこうと思う。

|  |
| --- |
| void draw(){  float gg\_pallet\_center\_x = height + lv02\_golden/2;  float gg\_pallet\_center\_y = lv01\_golden + lv02\_golden/2;  float gg\_pallet\_halfSquareOneSide = (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\_halfSquareOneSide < x  && x < gg\_pallet\_center\_x + gg\_pallet\_halfSquareOneSide  && gg\_pallet\_center\_y - gg\_pallet\_halfSquareOneSide < y  && y < gg\_pallet\_center\_y + gg\_pallet\_halfSquareOneSide){  selectColor = get(round(x), round(y));  }    // カラーパレットの彩度と明度  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();  pallet.translate(lv02\_golden/2, lv02\_golden/2);  makeColorpallet\_draw02(pallet, circle\_ratio\*lv02\_golden/2, hue(selectColor));  pallet.endDraw();  image(pallet, height, lv01\_golden);  }  }  } |

floatの4つでは今回の範囲指定に使用する値を設定している。

float gg\_pallet\_center\_xはパレットウィンドウのx座標を、float gg\_pallet\_center\_yはパレットウィンドウのy座標を、gg\_pallet\_halfSquareOneSideはカラーパレット(正方形部分)の1辺の半分の値を示している。

|  |
| --- |
| if (gg\_pallet\_center\_x - gg\_pallet\_halfSquareOneSide < x  && x < gg\_pallet\_center\_x + gg\_pallet\_halfSquareOneSide  && gg\_pallet\_center\_y - gg\_pallet\_halfSquareOneSide < y  && y < gg\_pallet\_center\_y + gg\_pallet\_halfSquareOneSide) |

ここではマウスの座標がカラーパレット(正方形部分)に入っている場合の色の取り方を示している。条件式を見ていくと、x座標はカラーパレット(正方形部分)の左側の辺から右側の辺までを指定しており、y座標はカラーパレット(正方形部分)の上側の辺から下側の辺までを指定している。

|  |
| --- |
| 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) |

ここではマウスの座標がカラーパレット(ドーナツ型部分)に入っている場合の色の取り方を示している。今回は色相を選ぶ際の図形がドーナツ型ということもあり範囲指定を、中心からの距離間で指定するようにしている。条件式を見ていくと、今マウスを置いている座標からカラーパレット(ドーナツ型部分)の中心までの距離よりも内円の半径のほうが小さく、かつ外円の半径のほうが大きい場合に色情報を取ることができるようにプログラムされている。

また選択した色をカラーパレット(正方形部分)に反映させたいため、選択した色の色相をmakeColorpallet\_draw02()に渡し描画させる。

4-3.戻る、進む機能の実装

戻る機能は Ctrl + zを、進むではCtrl+Shift+zを入力した場合に実行するようなプログラムを作成していく。

|  |
| --- |
| PImage[] stepImg;  int stepImgNumber = 15;  int backCount = 0, forwardCount = 0;  int stepImgCount;  boolean changeCanvas = false; |

まずPImage型の配列を宣言している。名前はstepImgとする。

int型の変数stepImgNumberを置いており15としているのは、今回のプログラムでは15個の配列を作成するという意味の元使用する。

backCount、forwardCountでは戻れる回数、進める回数をカウントする変数である。

stepImgCountは実際にどの配列番号の画像を抽出するかを指定するために用いる変数である。

changeCanvasでは、キャンバスに変化が見られたかを考える変数なのでboolean型にしてある。

|  |
| --- |
| stepImg = new PImage[stepImgNumber];  make\_stepImg\_setup();  (省略)  void make\_stepImg\_setup(){  for (int i=0; i<stepImgNumber; i++){  stepImg[i] = get(0, 0, height, height);  } |

stepImgに配列の個数を宣言し、配列に初期イメージ(キャンバス白)を代入している。

|  |
| --- |
| void save\_stepImg(){  backCount = min(backCount+1, stepImgNumber-1);  forwardCount = 0;  stepImgCount\_add();  stepImg[stepImgCount] = get(0, 0, height, height);  }  (省略)  void stepImgCount\_add(){  stepImgCount = (stepImgCount + 1) % stepImgNumber;  } |

キャンバスに変更が見られた場合に実行される関数である。

具体的のどのような動きをするかというと配列に画像を一時保存する。

詳細は戻ることができる回数を+1、勧めむことができる回数を0にする。minを使用しているのはbackCountが14を超えた数値を取らないようにするためである。

stepImgCount\_add()関数は配列番号を1つ増やす動きをしている。配列番号を1つ増やさないと、キャンバスにて変更が加わった場合に上書き保存されてしまうからだ。

さっき設定したstepImgCountを使用し、変更が加わったキャンバスデータを配列に一時保存している。

|  |
| --- |
| void undo(){  if (0 < backCount){  backCount--;  forwardCount++;  stepImgCount\_sub();  show\_stepImg();  }  }  (省略)  void stepImgCount\_sub(){  stepImgCount = (stepImgCount - 1 + stepImgNumber) % stepImgNumber;  } |

自作関数undo()では、戻る上で使用する関数である。

詳細を見ていくとif文の条件より、戻ることの出来る回数(backCount)が0回よりも多いときに使用可能であり、1つ戻る動作なのでbackCountの値を1つ減らしforwardCountの値を1つ増やす。

その後、stepImgCount\_sub()より配列の参照番号を-1し、その配列番号に保存されているイメージをキャンバスに貼りつける。

|  |
| --- |
| void redo(){  if (0 < forwardCount){  backCount++;  forwardCount--;  stepImgCount\_add();  show\_stepImg();  }  }  (省力)  void stepImgCount\_add(){  stepImgCount = (stepImgCount + 1) % stepImgNumber;  } |

自作関数redo()では、進む上で使用する関数である。

詳細を見ていくとif文の条件より、進むことの出来る回数(forwardCount)が0回よりも多いときに使用可能であり、1つ進む動作なのでforwardCountの値を1つ減らしbackCountの値を1つ増やす。

その後、stepImgCount\_add()より配列の参照番号を+1し、その配列番号に保存されているイメージをキャンバスに貼りつける。

stepImgCount\_add()、stepImgCount\_sub()で計算式に%を使用しているのは、stepImgNumber(今回だと15)の値を超えないようにするためである。

|  |
| --- |
| void mouseClicked(){  float x = mouseX, y = mouseY;  (省略)  else if(height+lv02\_golden < x && x < height+lv01\_golden  && height\*3/4 < y && y < height\*4/4){  undo();  }  else if(height+lv02\_golden < x && x < height+lv01\_golden  && height\*2/4 < y && y < height\*3/4){  redo();  }  } |

undo、redo機能をウィンドウ内でのボタンとして動作させるためのプログラムである。

if文内の条件式よりundoは領域8を、redoは領域7をマウスでクリックすれば使用できるようになっている。

|  |
| --- |
| void mousePressed(){  if (0 < mouseX && mouseX < height && 0 < mouseY && mouseY < height){  changeCanvas = true;  }  }  void mouseDragged(){  if (0 < mouseX && mouseX < height && 0 < mouseY && mouseY < height){  changeCanvas = true;  }  }  void mouseReleased(){  if (changeCanvas){  save\_stepImg();  changeCanvas = false;  }  } |

boolean型のchangeCanvasがどのようなときにtrueになり、キャンバスの変更を確認するか、またどこからどこまでの動きを1つの動きとするかを示している。基本はmousePressed()で事足りるが、キャンバス外から押したままキャンバス内に入ってきた場合に対応できないためmouseDragged()でキャンバス外からの動きにも対応できるようにした。

そしてmouseReleased()でchangeCanvasがtrueからfalseに変わるタイミングでキャンバス情報として保存している。保存し終えたあとは、また次のタイミングでほぞんでき保存できるようchangeCanvasをfalseにしている。

4-4.消しゴム、ペンの太さ変更機能の実装

|  |
| --- |
| float pen\_size = 5; //5 20 50  float ers\_size = 3; //3 10 50 |

まずペンと消しゴムの太さをfloat型の変数としておく。

|  |
| --- |
| boolean ers\_change = false; |

ers\_changeをboolean型の変数としておく。これは後のペンで書くモードか消しゴムで消すモードかを判定するのに使用する。

|  |
| --- |
| void mouseClicked(){  float x = mouseX, y = mouseY;  if ( height < x && x < height+lv02\_golden/2  && lv01\_golden\*0/3 < y && y < lv01\_golden\*1/3){  ers\_change = false;  pen\_size = 0.5\*10;  println(pen\_size, ers\_size, ers\_change);  }else if(height < x && x < height+lv02\_golden/2  && lv01\_golden\*1/3 < y && y < lv01\_golden\*2/3){  ers\_change = false;  pen\_size = 2\*10;  println(pen\_size, ers\_size, ers\_change);  }else if(height < x && x < height+lv02\_golden/2  && lv01\_golden\*2/3 < y && y < lv01\_golden\*3/3){  ers\_change = false;  pen\_size = 5\*10;  println(pen\_size, ers\_size, ers\_change);  }  else if(height+lv02\_golden/2 < x && x < height+lv02\_golden  && lv01\_golden\*0/3 < y && y < lv01\_golden\*1/3){  ers\_change = true;  ers\_size = 0.3\*10;  println(pen\_size, ers\_size, ers\_change);  }  else if(height+lv02\_golden/2 < x && x < height+lv02\_golden  && lv01\_golden\*1/3 < y && y < lv01\_golden\*2/3){  ers\_change = true;  ers\_size = 1\*10;  println(pen\_size, ers\_size, ers\_change);  }  else if(height+lv02\_golden/2 < x && x < height+lv02\_golden  && lv01\_golden\*2/3 < y && y < lv01\_golden\*3/3){  ers\_change = true;  ers\_size = 5\*10;  println(pen\_size, ers\_size, ers\_change);  } |

どの領域をクリックすればペンor消しゴムのモードになるのかを設定している。

領域1をクリックしたときの機能を使用できる部分はers\_change = falseなのでペン機能を使うことを示しておりペンの太さは5である。

領域2をクリックしたときの機能を使用できる部分はers\_change = falseなのでペン機能を使うことを示しておりペンの太さは20である。

領域3をクリックしたときの機能を使用できる部分はers\_change = falseなのでペン機能を使うことを示しておりペンの太さは50である。

領域4をクリックしたときの機能を使用できる部分はers\_change = trueなので消しゴム機能を使うことを示しており消しゴムの太さは１である。

領域4をクリックしたときの機能を使用できる部分はers\_change = trueなので消しゴム機能を使うことを示しており消しゴムの太さは10である。

領域4をクリックしたときの機能を使用できる部分はers\_change = trueなので消しゴム機能を使うことを示しており消しゴムの太さは50である。

|  |
| --- |
|  |

上で作成した消しゴム機能もしくはペン機能をキャンバス部分にのみかけるよう条件付けされている。 まずif文の条件付けで現在のマウス座標と、1フレーム前のマウス座標がキャンバス内にあることを設定する。 そののちers\_changeがtrueなら白(キャンバス背景色と同じ色)で、falseならペン機能でラインを引くようにプログラムしている。

4-5.保存機能の実装

4-6.キャンバスの新規作成機能の実装

4-7.タブアイコンの作成と描画

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

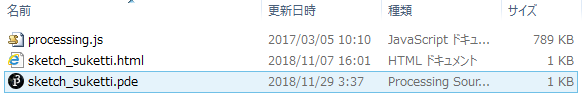
　Processing.jsは、拡張子が.pdeであるProcessingのような実行ファイルとは違い、HTMLに読み込む.jsであるJavaScriptファイルになっている。

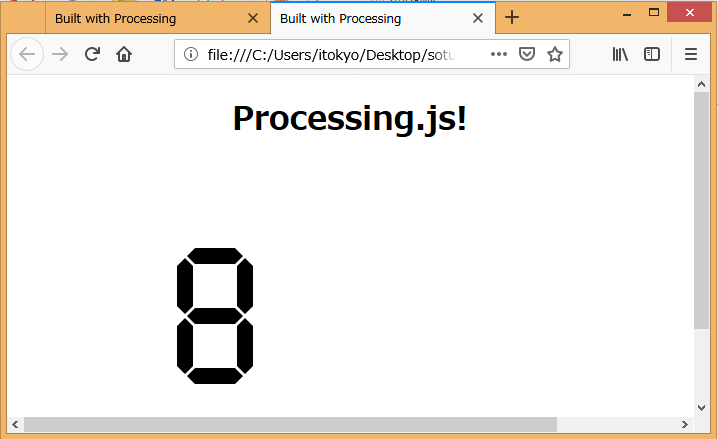
Processing.jsはProcessingの公式サイトのダウンロードページ(http://processingjs.org/download/) からダウンロードすることが可能である。

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.今後の研究課題

参考文献

謝辞

付録

今回作成したプログラム

|  |
| --- |
| PGraphics pallet, tab01, tab02, tab03;  final float golden\_ratio = 1.618;  final float circle\_ratio = 0.75;  color selectColor = color(0, 0, 0);  color eraserColor = color(255, 255, 255);  int size\_height = 1000;  int lv01\_golden = round(size\_height\*(golden\_ratio-1));  int lv02\_golden = round(lv01\_golden\*(golden\_ratio-1));  int lv03\_golden = round(lv02\_golden\*(golden\_ratio-1));    float pen\_size = 5;  float ers\_size = 3;    boolean ers\_change = false;    PImage[] stepImg;  int stepImgNumber = 15;  int backCount = 0, forwardCount = 0;  int stepImgCount;  boolean changeCanvas = false;    boolean downCtrl = false, downShift = false, downA = false;    int dayY, dayM, dayD, timeH, timeM, timeS;  String dayString, timeString;    void setup(){  size(1618, 1000);    colorMode(HSB, 360, 100, 100);  makeWindow();  makeWindow\_setup02(color(0, 0, 100));  makeColorpallet(pallet);  maketab01(tab01);  maketab02(tab02);  maketab03(tab03);    makeWindow\_show();    stepImg = new PImage[stepImgNumber];  make\_stepImg\_setup();  }  void makeWindow(){  pallet = createGraphics(lv02\_golden, lv02\_golden);  tab01 = createGraphics(lv02\_golden/2, lv01\_golden);  tab02 = createGraphics(lv02\_golden/2, lv01\_golden);  tab03 = createGraphics(lv03\_golden, height);  }  void makeWindow\_setup02(color base\_color){  noStroke();  fill(base\_color);  rect(0, 0, height, height);  }    void makeColorpallet(PGraphics gg){  int squareOneSide = gg.height;  float large\_r = squareOneSide/2;  float small\_r = (large\_r)\*circle\_ratio;    gg.beginDraw();  gg.colorMode(HSB, 360, 100, 100);  gg.pushMatrix();  gg.translate(large\_r, large\_r);  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);  gg.arc(0, 0, r01\*2, r01\*2, i\*on\_rad, (i+1)\*on\_rad);  }  gg.fill(0, 0, 100);  gg.ellipse(0, 0, r02\*2, r02\*2);  }  void makeColorpallet\_draw02(PGraphics gg, float r02, float color\_hue){  int separate = 50;  float separate\_squareOneSide = r02\*(1/sqrt(2))\*2/separate;  float start\_separate\_coordinate = -r02\*(1/sqrt(2));    gg.noStroke();  for(int i = 0; i < separate; i++){  for(int j = 0; j < separate; j++){  float change\_saturation = 100\*(i+1)/separate;  float change\_brightness = 100\*(j+0)/separate;  gg.fill(color\_hue, 100-change\_saturation, 100-change\_brightness);  gg.rect(start\_separate\_coordinate + separate\_squareOneSide\*i, start\_separate\_coordinate + separate\_squareOneSide\*j, separate\_squareOneSide + 1, separate\_squareOneSide + 1);  }  }  }    void makeWindow\_show(){  image(pallet, height, lv01\_golden);  image(tab01, height, 0);  image(tab02, height+lv02\_golden/2, 0);  image(tab03, height+lv02\_golden, 0);  }    void maketab01(PGraphics tt){  int separate\_count = floor(tt.height / tt.width);    tt.beginDraw();  tt.translate(0, tt.height\*0/separate\_count);  tt.stroke(1);  tt.fill(240);  tt.rect(0, 0, tt.width, tt.height\*1/separate\_count);  pencil(tt, tt.width/2, tt.height/6, 10);    tt.translate(0, tt.height\*1/separate\_count);  tt.stroke(1);  tt.fill(240);  tt.rect(0, 0, tt.width, tt.height\*1/separate\_count);  pencil(tt, tt.width/2, tt.height/6, 20);    tt.translate(0, tt.height\*1/separate\_count);  tt.stroke(1);  tt.fill(240);  tt.rect(0, 0, tt.width, tt.height\*1/separate\_count);  pencil(tt, tt.width/2, tt.height/6, 30);  tt.endDraw();    }  void maketab02(PGraphics tt){  int separate\_count = floor(tt.height / tt.width);    tt.beginDraw();  tt.translate(0, tt.height\*0/separate\_count);  tt.stroke(1);  tt.fill(240);  tt.rect(0, 0, tt.width, tt.height\*1/separate\_count);  eraser(tt, tt.width/2, tt.height/6, 10);    tt.translate(0, tt.height\*1/separate\_count);  tt.stroke(1);  tt.fill(240);  tt.rect(0, 0, tt.width, tt.height\*1/separate\_count);  eraser(tt, tt.width/2, tt.height/6, 20);    tt.translate(0, tt.height\*1/separate\_count);  tt.stroke(1);  tt.fill(240);  tt.rect(0, 0, tt.width, tt.height\*1/separate\_count);  eraser(tt, tt.width/2, tt.height/6, 30);  tt.endDraw();  }  void maketab03(PGraphics tt){  int separate\_count = floor(tt.height / tt.width);  int arrowSize = 15;  float arrowStart\_x = tt.width/2 - 4\*arrowSize;  float arrowStart\_y = tt.height/8;    tt.beginDraw();  tt.translate(0, tt.height\*0/separate\_count);  tt.stroke(1);  tt.fill(240);  tt.rect(0, 0, tt.width, tt.height\*1/separate\_count);    tt.translate(0, tt.height\*1/separate\_count);  tt.stroke(1);  tt.fill(240);  tt.rect(0, 0, tt.width, tt.height\*1/separate\_count);    tt.translate(0, tt.height\*1/separate\_count);  tt.stroke(1);  tt.fill(240);  tt.rect(0, 0, tt.width, tt.height\*1/separate\_count);  tt.noStroke();  tt.fill(0);  tt.triangle(arrowStart\_x + 4\*arrowSize, arrowStart\_y + 4\*arrowSize, arrowStart\_x + 8\*arrowSize, arrowStart\_y, arrowStart\_x + 4\*arrowSize, arrowStart\_y - 4\*arrowSize);  tt.fill(0);  tt.quad(arrowStart\_x, arrowStart\_y + 2\*arrowSize, arrowStart\_x + 4\*arrowSize, arrowStart\_y + 2\*arrowSize, arrowStart\_x + 4\*arrowSize, arrowStart\_y - 2\*arrowSize, arrowStart\_x, arrowStart\_y - 2\*arrowSize);    tt.translate(0, tt.height\*1/separate\_count);  tt.stroke(1);  tt.fill(240);  tt.rect(0, 0, tt.width, tt.height\*1/separate\_count);  tt.noStroke();  tt.fill(0);  tt.triangle(arrowStart\_x, arrowStart\_y, arrowStart\_x + 4\*arrowSize, arrowStart\_y + 4\*arrowSize, arrowStart\_x + 4\*arrowSize, arrowStart\_y - 4\*arrowSize);  tt.fill(0);  tt.quad(arrowStart\_x + 4\*arrowSize, arrowStart\_y + 2\*arrowSize, arrowStart\_x + 8\*arrowSize, arrowStart\_y + 2\*arrowSize, arrowStart\_x + 8\*arrowSize, arrowStart\_y - 2\*arrowSize, arrowStart\_x + 4\*arrowSize, arrowStart\_y - 2\*arrowSize);    tt.endDraw();    }    void pencil(PGraphics tt, float center\_x, float center\_y, float pencilSize){  tt.noStroke();  tt.fill(0);  tt.triangle(center\_x + pencilSize, center\_y - 2\*pencilSize, center\_x + pencilSize, center\_y + 2\*pencilSize, center\_x + 3\*pencilSize, center\_y);  tt.fill(230, 190, 130);  tt.quad(center\_x + pencilSize, center\_y - 2\*pencilSize, ((center\_x + pencilSize)+(center\_x + 3\*pencilSize))/2, ((center\_y - 2\*pencilSize)+(center\_y))/2, ((center\_x + pencilSize)+(center\_x + 3\*pencilSize))/2, ((center\_y + 2\*pencilSize)+(center\_y))/2, center\_x + pencilSize, center\_y + 2\*pencilSize);  tt.fill(80,150,50);  tt.rect(center\_x - 3\*pencilSize, center\_y - 2\*pencilSize, 4\*pencilSize, 4\*pencilSize);  }  void eraser(PGraphics tt, float center\_x, float center\_y, float eraserSize){  tt.fill(255);  tt.ellipse(center\_x + eraserSize, center\_y, 4\*eraserSize, 4\*eraserSize);  tt.fill(100,170,220);  tt.rect(center\_x - 3\*eraserSize, center\_y - 2\*eraserSize, 4\*eraserSize, 4\*eraserSize);  }    void make\_stepImg\_setup(){  for (int i=0; i<stepImgNumber; i++){  stepImg[i] = get(0, 0, height, height);  }  }  void save\_stepImg(){  backCount = min(backCount+1, stepImgNumber-1);  forwardCount = 0;  stepImgCount\_add();  stepImg[stepImgCount] = get(0, 0, height, height);  }  void undo(){  if (0 < backCount){  backCount--;  forwardCount++;  stepImgCount\_sub();  show\_stepImg();  }  }  void redo(){  if (0 < forwardCount){  backCount++;  forwardCount--;  stepImgCount\_add();  show\_stepImg();  }  }  void stepImgCount\_add(){  stepImgCount = (stepImgCount + 1) % stepImgNumber;  }  void stepImgCount\_sub(){  stepImgCount = (stepImgCount - 1 + stepImgNumber) % stepImgNumber;  }  void show\_stepImg(){  image(stepImg[stepImgCount], 0, 0);  }    void mouseClicked(){  float x = mouseX, y = mouseY;  if ( height < x && x < height+lv02\_golden/2  && lv01\_golden\*0/3 < y && y < lv01\_golden\*1/3){  ers\_change = false;  pen\_size = 0.5\*10;  println(pen\_size, ers\_size, ers\_change);  }else if(height < x && x < height+lv02\_golden/2  && lv01\_golden\*1/3 < y && y < lv01\_golden\*2/3){  ers\_change = false;  pen\_size = 2\*10;  println(pen\_size, ers\_size, ers\_change);  }else if(height < x && x < height+lv02\_golden/2  && lv01\_golden\*2/3 < y && y < lv01\_golden\*3/3){  ers\_change = false;  pen\_size = 5\*10;  println(pen\_size, ers\_size, ers\_change);  }  else if(height+lv02\_golden/2 < x && x < height+lv02\_golden  && lv01\_golden\*0/3 < y && y < lv01\_golden\*1/3){  ers\_change = true;  ers\_size = 0.3\*10;  println(pen\_size, ers\_size, ers\_change);  }  else if(height+lv02\_golden/2 < x && x < height+lv02\_golden  && lv01\_golden\*1/3 < y && y < lv01\_golden\*2/3){  ers\_change = true;  ers\_size = 1\*10;  println(pen\_size, ers\_size, ers\_change);  }  else if(height+lv02\_golden/2 < x && x < height+lv02\_golden  && lv01\_golden\*2/3 < y && y < lv01\_golden\*3/3){  ers\_change = true;  ers\_size = 5\*10;  println(pen\_size, ers\_size, ers\_change);  }  else if(height+lv02\_golden < x && x < height+lv01\_golden  && height\*3/4 < y && y < height\*4/4){  undo();  }  else if(height+lv02\_golden < x && x < height+lv01\_golden  && height\*2/4 < y && y < height\*3/4){  redo();  }  }  void mousePressed(){  if (0 < mouseX && mouseX < height && 0 < mouseY && mouseY < height){  changeCanvas = true;  }  }  void mouseDragged(){  if (0 < mouseX && mouseX < height && 0 < mouseY && mouseY < height){  changeCanvas = true;  }  }  void mouseReleased(){  if (changeCanvas){  save\_stepImg();  changeCanvas = false;  }  }    void draw(){  float gg\_pallet\_center\_x = height + lv02\_golden/2;  float gg\_pallet\_center\_y = lv01\_golden + lv02\_golden/2;  float gg\_pallet\_halfSquareOneSide = (1/sqrt(2))\*circle\_ratio\*lv02\_golden/2;    if (mousePressed){  float x = mouseX, y = mouseY;      if (gg\_pallet\_center\_x - gg\_pallet\_halfSquareOneSide < x  && x < gg\_pallet\_center\_x + gg\_pallet\_halfSquareOneSide  && gg\_pallet\_center\_y - gg\_pallet\_halfSquareOneSide < y  && y < gg\_pallet\_center\_y + gg\_pallet\_halfSquareOneSide){  selectColor = get(round(x), round(y));  }      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();  pallet.translate(lv02\_golden/2, lv02\_golden/2);  makeColorpallet\_draw02(pallet, circle\_ratio\*lv02\_golden/2, hue(selectColor));  pallet.endDraw();  image(pallet, height, lv01\_golden);  }    if (0 < mouseX && mouseX < height && 0 < mouseY && mouseY < height  && 0 < pmouseX && pmouseX < height && 0 < pmouseY && pmouseY < height){  if (ers\_change){  stroke(eraserColor);  strokeWeight(ers\_size);  }else{  stroke(selectColor);  strokeWeight(pen\_size);  }  line(mouseX, mouseY, pmouseX, pmouseY);  }  image(tab01, height, 0);  image(tab02, height+lv02\_golden/2, 0);  image(tab03, height+lv02\_golden, 0);  }  }    void keyPressed(){  if (keyCode == 'A'){  downA = true;  return;  }  if (key == CODED){  if (keyCode == CONTROL){  downCtrl = true;  }  if (keyCode == SHIFT){  downShift = true;  }  return;  }  if (downCtrl){  if (keyCode == 'Z'){  if (downShift){  redo();  }else{  undo();  }  }  if (keyCode == 'S' && !downShift){  dayY = year();  dayM = month();  dayD = day();  timeH = hour();  timeM = minute();  timeS = second();  dayString = nf(dayY, 4) + nf(dayM, 2) + nf(dayD, 2);  timeString = nf(timeH, 2) + nf(timeM, 2) + "-" + nf(timeS, 2);  get(0, 0, height, height).save("image\_" + dayString + "\_" + timeString +".png");  println("保存しました");  }  if (downShift && downA){  if (keyCode == 'D'){  noStroke();  fill(color(0, 0, 100));  rect(0, 0, height, height);  make\_stepImg\_setup();  backCount = 0;  forwardCount = 0;  stepImgCount = 0;  changeCanvas = false;  println("すべて削除");  }  }  return;  }  }  void keyReleased() {  if (key == CODED) {  if (keyCode == CONTROL){  downCtrl = false;  }  if (keyCode == SHIFT){  downShift = false;  }  }  if (keyCode == 'A'){  downA = false;  }  } |