Piet 班　活動報告書

立命館コンピュータクラブ
2019 年度後期プロジェクト活動

2020 年 2 月 10 日

山口　流星 [*1]
青木　雅典 [*2]
伊藤　聡子 [*3]
坪倉　奏太 [*4]
渥美　柾彦 [*5]

概要

本班は難解プログラミング言語の一種である Piet の理論を習得し，自らの力でプログラミングを行い，ソースコードを解読出来るようになることを目標とした．なお，本プロジェクトでは開発環境として Pidet と Pietron を使用している．

[*1]理工学部 数理科学科 三回生
[*2]理工学部 電子情報工学科 三回生
[*3]情報理工学部 情報理工学科 知能情報コース 三回生
[*4]情報理工学部 情報理工学科 実世界情報コース 二回生
[*5]理工学部 電子情報工学科 三回生

目 次

1　Piet の理論

文責: 山口　流星

1.1　難解言語

難解言語とは，読解が意図的に困難な言語のことを指す．Piet のソースコードは可読性が著しく低いことから，これも難解言語の一種である．

1.2　Piet の歴史

Piet は，2002 年 4 月[*6]には David Morgan-Mar(以下，DM 氏) により発案されていた難解プログラミング言語である．Piet はソースコードがドット絵であり，Piet の命名には抽象絵画のパイオニアである Piet Mondrian が由来とされる．なお，DM 氏はこの難解プログラミング言語を Mondrian と命名したかったようであるが，Mondrian というプログラミング言語が当時既に存在していたため，Piet と名付けたとされる．

1.3　コーデル

コーデルとは，ドットの最小単位を指し，1 コーデルのピクセルサイズを指定することが出来る．

1.4　キャンバス

キャンバスとは，プログラムを記述するためのフィールドである．キャンバスのサイズには単位をコーデルとして幅と高さを持つ．なお，キャンバスの端は黒ブロックと同様の働きを持つ．

1.5　カラーブロック

カラーブロックは Piet のメインユニットであり，白・黒以外の隣接している同色コーデルの塊を指す．ただし，斜めのみへの隣接を含まない．また，白と黒のカラーブロックは全て 1 コーデルで独立している．

1.6　プログラムポインター

プログラムポインター (以下，PP) は，プログラムが実行するドット上を動く点のことであり，これが移動することによって命令が実行される．なお，動作の開始は必ずキャンバスサイズの左上からである．PP の移動には 2 種類の方向が伴い，ダイレクションポインター (以下，DP) とコーデルチューザー (以下，CC) が存在する．

1.7　ダイレクトポインター

DP はプログラムの移動方向であり，初期は右方向を持つ．CC を切り替えても移動に失敗するとき，時計回りに 90 度回転する．

[*6]Web Archive を遡ったところ，これが最古の記録であった．

1.8 コーデルチューザー

CC は複数の移動候補を絞るための方向である．初期は左方向を持つ．移動に失敗するまでは CC は変更されないが，一度移動が失敗するとき CC を右方向に切り替える．ただし 2 回連続で移動に失敗するとき，前述の通り DP が切り替わるが，CC は切り替わらないため注意する．

1.9 プログラムの実行

PP の移動は，DP 方向から見て最も遠い別のカラーコーデルの端を探す．この場合，移動候補が複数あることもある．次に候補のうち，DP 方向から見て最も CC 方向へ PP が進む．この時，CC 方向には右もしくは左方向が入る．もし移動に失敗するときは，CC 方向を切り替えて再び移動を試みるが，それでも移動に失敗するときは DP 方向を時計回りに 90 度回転する．これらを繰り返して，終了条件を満たすとプログラムは終了する．

1.10 プログラムの正常終了

命令の詳細は後述するが，wait 命令を 8 回連続で実行するか，幾つかの白コーデル上における slide 命令の無限ループが発生すると，プログラムはそれ以上の実行に意味がなくなり，プログラムが終了する．

1.11 色に関して

Piet では赤，黄，緑，水，青，紫の 6 色に対して 3 パターンの明度があり，それらに白と黒を加えた合計 20 色で構成される．また，白と黒は特別な役割を持っており，18 色とは区別する．

1.11.1 黒ブロック

移動候補先に黒ブロックが存在する場合，必ず移動が失敗する．終了や移動の制御に非常に重要である．

1.11.2 白ブロック

PP が移動する際，他のカラーブロックの端かキャンバスの行き止まりまでの間に白ブロックが存在する場合，DP 方向の直線方向に移動し続ける．また，白ブロックは 1 コーデルのカラーブロックとして独立している．さらに，白ブロックが関与する重要な命令も存在する．

1.12 スタック

スタックとは，メモリのような役割を果たすものである．整数値 (符号付き 32bit) のみ扱うことが出来る．データは先入れ後出し構造になっており，データを入れることを push, データを取りだすことを pop という．なお，push と pop については命令にも同様の単語が定義されているが別の意味であるため注意する必要がある．しかし，本報告書内では命令に関しては全て○○命令と記述することにしているため，今回に関しては特段気にする必要はない．

1.13 命令

命令は色の差分によって意味を持つ．この時，元にいたカラーブロックの明度と色から，次に移動したカラーブロックの明度と色の差分に対して実行される命令が決まっている．ただし，命令が実行される場合でも，命令の条件を満たしていない場合はその命令は無視する点に注意する必要がある．なお，明度は明るい，標準，暗いをローテーションしており，暗いの次は明るいで繋がっている．同様に色は赤，黄，緑，水，青，紫でローテーションしており，紫の次は赤で繋がっている．以下で全ての命令を記述しておく．

1.13.1 命令一覧

色の差分によって生じる命令に関しては，(色差，明度差) と条件を記述する．また，命令の紹介であるため，この項に限り命令の名称のみを羅列する．

- push: (0,1) 差分の命令．移動前にいたカラーブロックの最大コーデルサイズを整数値に変換して push する．
- pop: (0,2) 差分の命令．スタックから 1 つ pop して破棄する．
- add: (1,0) 差分の命令．スタックから 2 つ pop して加算した整数値を push する．
- subtract: (1,1) 差分の命令．スタックから 2 つ pop して 2 つ目に pop した値から 1 つ目に pop した値を減算した整数値を push する．
- multiply: (1,2) 差分の命令．スタックから 2 つ pop して掛算した整数値を push する．
- divide: (2,0) 差分の命令．スタックから 2 つ pop して 2 つ目に pop した値から 1 つ目に pop した値を切り捨て除算した整数値を push する．
- mod: (2,1) 差分の命令．スタックから 2 つ pop して 2 つ目に pop した値から 1 つ目に pop した値を割り，その剰余を push する．
- input(n): (4,2) 差分の命令．入力スペースの整数値を 1 つ受け取り push する．
- input(c): (5,0) 差分の命令．入力スペースの文字を受け取り，文字コード Unicode に従って 10 進数の整数値に変換したものを push する．
- output(n): (5,1) 差分の命令．スタックから 1 つ pop して出力スペースに出力する．
- output(c): (5,2) 差分の命令．スタックから 1 つ pop して整数値を文字コード Unicode に従って文字に変換したものを出力スペースに出力する．
- duplicate: (4,0) 差分の命令．スタックから 1 つ pop してその値を 2 つ push する．
- greater: (3,0) 差分の命令．スタックから 2 つ pop して 2 つ目に pop した値が 1 つ目に pop した値以上ならば 1，そうでなければ 0 を push する．
- pointer: (3,1) 差分の命令．スタックから 1 つ pop して，その値だけ DP 方向を時計回りに回転する．pop した値が負の場合は反時計回りに回転する．
- switch: (4,2) 差分の命令．スタックから 1 つ pop して，その値だけ CC 方向を切り替える．
- not: (2,2) 差分の命令．スタックから 1 つ pop して，その値が 0 ならば 1, それ以外の整数値は 0 を push する．
- roll: (4,1) 差分の命令．スタックから 2 つ pop して，2 つ目に pop した値で roll の対象範囲となるスタック内の個数を決定して，1 つ目の値分だけ回転させる．
- noop: 白ブロックからカラーブロックへ移動する命令．この時，何も命令が起こらない．
- slide: 移動先が白ブロックとなるときの命令．この時，PP は移動しながらも黒ブロックないしキャンバス端に到達するため，移動に失敗して CC 方向ないし DP 方向が変更される．
- wait: PP がカラーブロック上にあるが移動に失敗したときの命令．移動に失敗しているため，CC 方向ないし DP 方向が変更される．

2 成果物

文責: 山口　流星

ここからは本活動を通じて制作したソースコードを紹介する．

2.1 Hello World

Hello World の作り方は何種類か方法があるが，今回は最も簡単な input(c) 命令と output(c) 命令を用いて制作した．Hello World 制作に合わせてスタックの理解を並行して行ったため，入力を [dlroW olleH] としたが，実際は input(c) 命令と output(c) 命令を交互に行うことでも同様の結果が得られる．(画像 5 参照) 真の Piet 使いであれば，標準入力を使わない方法こそ Hello World であるかもしれないが，今回はここまで踏み込まない．以下，班員の Hello World である．

図 1: 青木 (Hello World)

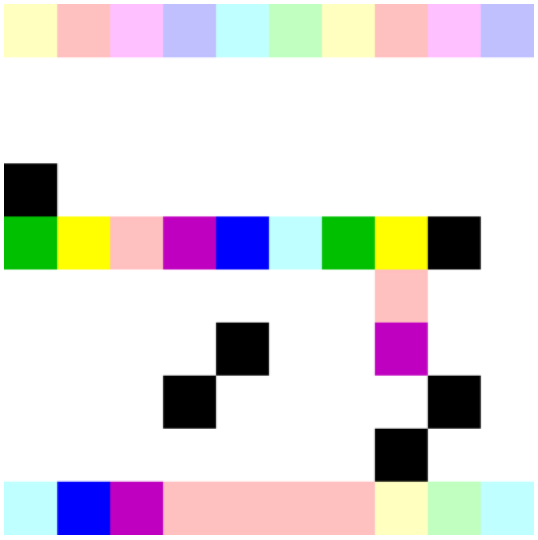

図 2: 伊藤 (Hello World!)

図 3: 坪倉 (Hello World)

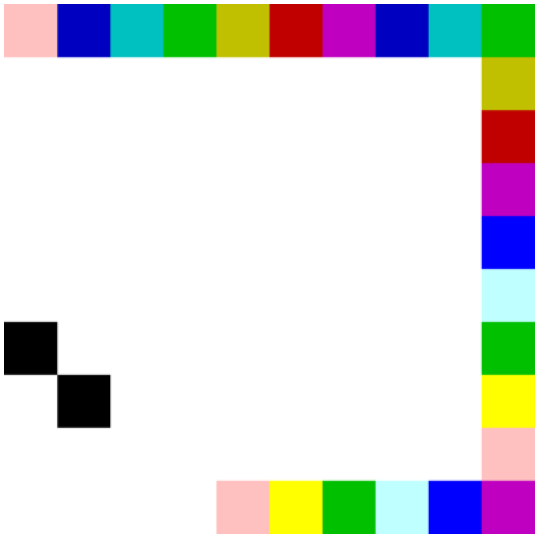

図 4: 渥美 (Hello World)

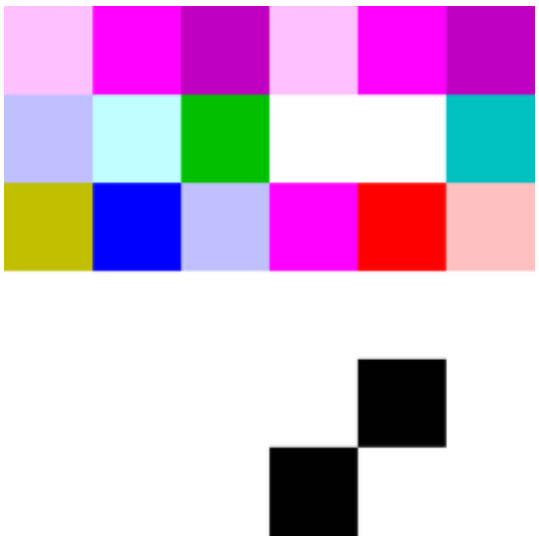

図 5: 山口 (交互 Hello World!)

2.2 階乗

Piet の基本を押さえるという意味において，階乗のプログラムは最適である．今回のプロジェクトではこのコードを自分の力で描くことが出来れば十分だと考えている．階乗のプログラムは，初心者の壁となるループと roll 命令を扱わなければならない．本プロジェクトでもそれらの習得には苦労し，描き上げるまでにかなりの時間を要した．以下，班員の階乗のプログラムである．画像 6 は 2 以上の整数値に対応，画像 7 から 9 は 0 以上の整数値に対応している．なお，画像 10 についてはそれに加えて，13 以上の階乗の値はオーバーフローしてしまうため，計算しない処理を含めてある．

図 6: 渥美 (2 以上)

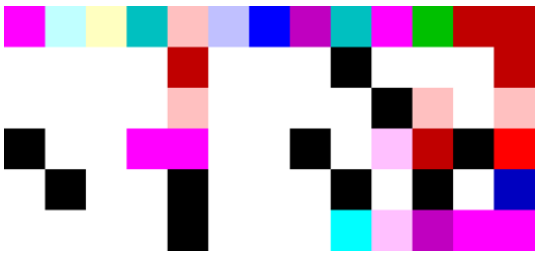

図 7: 青木 (0 以上)

図 8: 坪倉 (0 以上)

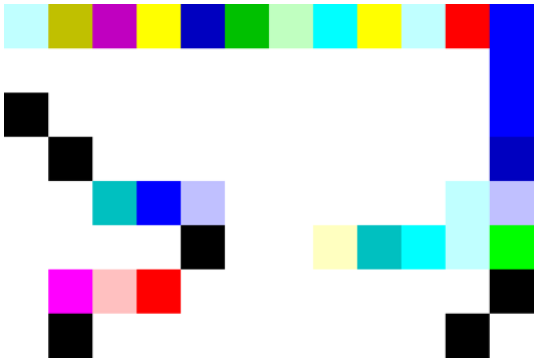

図 9: 伊藤 (0 以上)

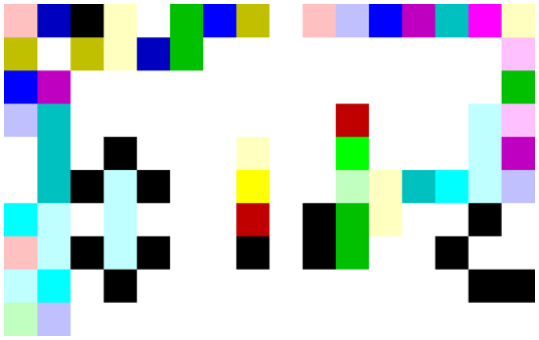

図 10: 山口 (0 以上 & 13 以上処理)

2.3 各自制作したもの

上記以外で各自任意に制作したプログラムについて載せておく.

2.3.1 $a^b(modM)$ を求めるプログラム

文責: 山口　流星

M, a, b を標準入力として受け取り，$a^b(modM)$ を返すプログラム．$b = 1$ と $b > 1$ で処理を分けて実装し，$b > 1$ は mod の逐次処理を行うことで計算している.

図 11: $a^b(modM)$ を求めるプログラム

2.3.2　任意個の標準入力 (整数値) を受け取るプログラム

文責: 山口　流星

標準入力の整数値がいくつであってもこれ一つで解決してしまう関数のようなものを制作した．入力スペースにて，空白を読み取るとループから抜ける仕組みになっているため，連続して input(n) 命令を行いたい場合は改行で整数値を区切れば良い．

図 12: 任意個の標準入力 (整数値) を受け取るプログラム

3　刺繍

Piet はドット絵がソースコードとなっており，１つの作品にかなりの制作時間を要する．しかしそれら制作物である画像をただ紙に印刷するだけでは惜しい．そのため，今回は刺繍糸を用いたクロスステッチに挑戦し，ソースコードを納めることにした．それにあたり，各班員は自由なプログラムを１つ制作し，クロスステッチを行った．以下，各班員の成果を報告するとともに，活動の振り返りも同時に行う．

3.1　何もしないもん

文責: 山口　流星

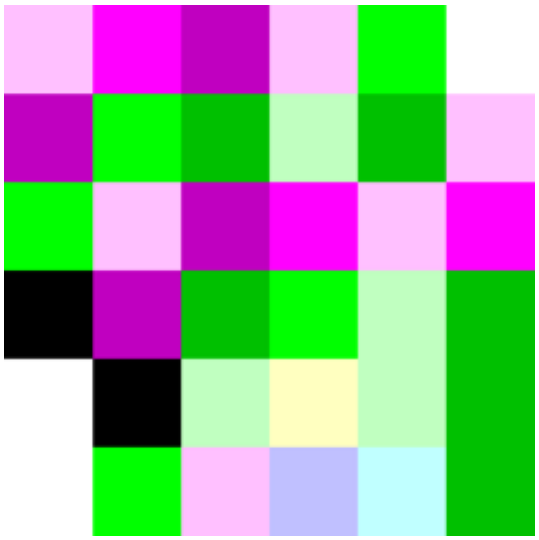

図 13: 山口 (ソースコード)

図 14: 山口 (刺繍画像)

3.1.1　コード概要

標準入力を必要とせず，push 命令を用いて整数値を生成し，pointer 命令や switch 命令など制御命令を用いて進行方向を変更するただそれだけのプログラムである．もちろん実行は正常に終了し，その時スタックも空になるため，非常に美しい辺鄙なソースコードに仕上がったと自負している．なお，このプログラムの実行はプログラムポインタが動作していることをただ眺めるだけになるため，これを実行する暇があるならば他に時間を使う方が良いであろう．

3.1.2　制作ポイント

刺繍に使用するコードであるため，キャンバスサイズは 6 × 6 に限定した．正方形のキャンバスサイズを使用する際，いかに余白を少なくするかどうかが腕の見せ所であり，ここに大きく躓いた．概ねプログラムポインタの動きを想像しながら，まずは上 3 段を隙間なく埋めながらも下段に上手くつながるよう調整した．そして 4 段目右下は 3 コーデルのカラーブロックを使用することにより，コーデルチューザーを切り替えることで分岐を生成することに成功した．

3.1.3　感想・展望

今回，約 2 年ぶりの Piet コーディングであったが，当時の知見を活かしながら様々なプログラムを制作し，プロジェクト内で共有することが出来た．Piet は理論として習得することはそれほど多くはないが，その単純さゆえにコーディングは困難を極めた．今回は 2 重ループなど高度な実装するまでは及ばなかったが，いずれ機会があれば挑戦してみたいと考えている．

3.2　やがて戻ってくる

文責: 坪倉　奏太

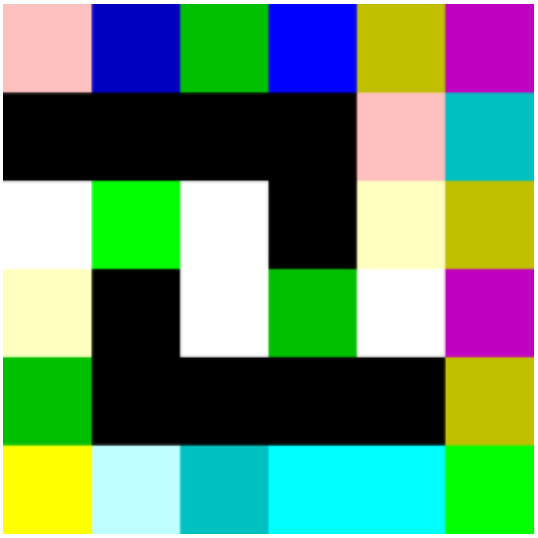

図 15: 坪倉 (ソースコード)

図 16: 坪倉 (刺繍画像)

3.2.1　コード概要

ある整数 n を標準入力で受け取った後，それをメモリ上で 4 回複製，Piet における積演算，和演算を実行したのちメモリの一番上の二番目の整数を入れ替え，差演算，商演算を実行，外部出力する．なお，0 が入力された場合そのまま 0 を出力する

3.2.2　制作ポイント

やや対称性のある色配置，roll 命令の位置

3.2.3　感想・展望

Piet のプログラミングで学べたこととして 1 番にあげられることは,「自分のやりたいことをどうコンピュータがわかるように言い直すか」という姿勢である．Piet の基本命令は数が少なく，またそのどれもが低レイヤのものであるので，例えば今回活動で実装した階乗計算でもその処理の中身を一つ一つ，丁寧に分解しそれを Piet の処理に落とし込む必要があった．学部，及び趣味のプログラミングでも，これほどまでに自分の実装したいことを一つ一つ詳しく分析してコードに落とし込む，という経験はしたことがなく，実際にはコンピュータがどう動作しているのかなどを考察する上で大変参考になった．また，Piet のプログラミングにおいて詰まる箇所というのは知識不足からというよりも自分で組みわせた基本命令群が認識とは違う動きをした，などのアルゴリズム面に原因がある場合が多く，コーディングする際にかなりの時間を要した点が印象に残っている．

また，今後の展望として「ある処理を Piet 上で定義する時，最低限必要とされるスペースはどれくらいなのか」を調べるなどと言ったことが挙げられる．

3.3　割り算 (あまり付き)

文責: 伊藤　聡子

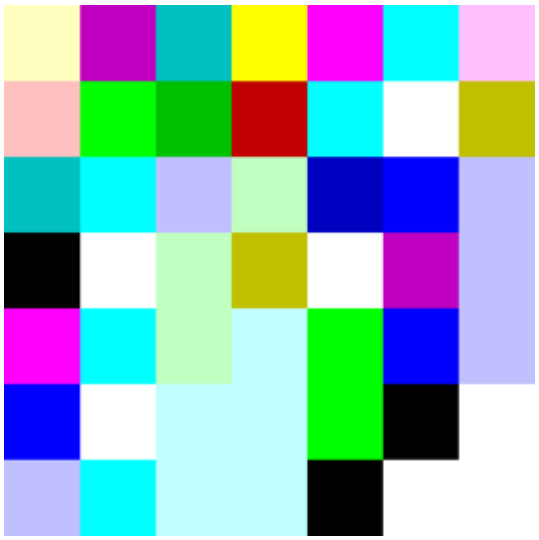

図 17: 伊藤 (ソースコード)

図 18: 伊藤 (刺繍画像)

3.3.1　コード概要

事前に入力した値の 1 つ目を 2 つ目で割り，その整数解とあまりを出力する．

3.3.2　制作ポイント

整数解とあまりの 2 つを出力する間に，ドットを 2 つ出力することにより，より「あまり感」を出した．なお，このドットは外部から入力したものではなく，コード内部でドットを表す「46」を四則演算を組み合わせて用意したものである．

3.3.3　感想・展望

いつまでも piet に慣れることができなかったため，一から十までプロジェクトリーダーに助けてもらいながら作った．彼には頭が上がるまい．これを作っても未だに Piet に慣れることはできないようだ．さすが難読言語．高級言語に甘えて生きてきた人間には厳しいものがある．だがこれも修行．今後はより自分の力でコードが描けるように精進していく所存である．

3.4　ポーランド記法で加減乗算

文責: 青木　雅典

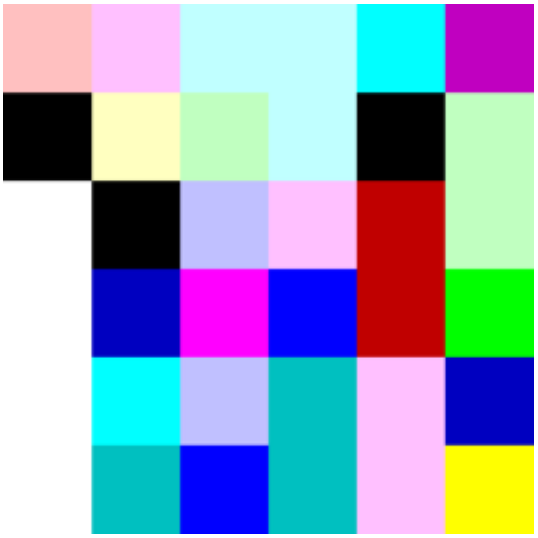

図 19: 青木 (ソースコード)

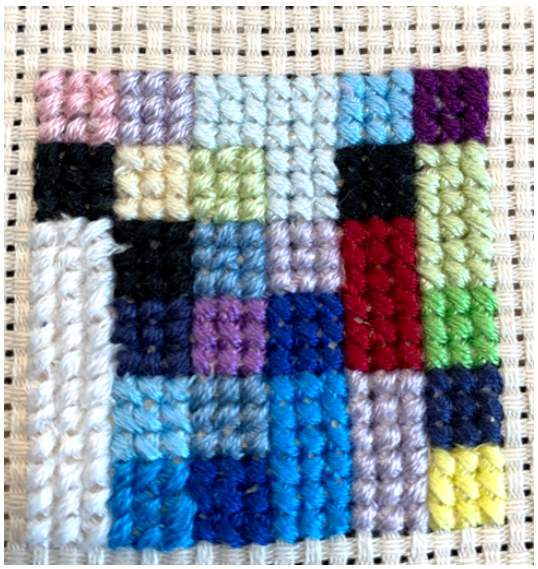

図 20: 青木 (刺繍画像)

3.4.1　コード概要

標準入力に 1 つの演算子と 2 つの被演算子を入力すると，演算結果を標準出力に吐き出すプログラムである．例えば「+ 5 3」とすれば「8」が出力され,「* 5 3」とすれば「15」が出力される．標準入力を先頭から順に受け取るため，プログラムはまず演算子を判定する．次に被演算子となる 2 数を標準入力からスタックへ取り込んだ後，指定された演算を行い，結果を標準出力へと吐き出す．

3.4.2　制作ポイント

プログラム上で演算子を判定することは，それぞれの演算が実行される経路を PP が辿るよう導くことを意味する．この複雑な処理を 6×6 の正方形という非常にコンパクトなキャンバスに収めるため，剰余と pointer 命令，switch 命令を活用して PP を導く．まず，演算子は実際には Unicode 値で入力される.「+」が「43」,「-」が「45」,「*」が「42」となり，まずこれを 3 で割った剰余を考えると,「+」のみが判別できる．次に，2 で割った剰余を考えると「-」と「*」の分岐が可能となる．演算子を決定した後は，それぞれ標準入力から数値を 2 つとり，それらの演算を行った後プログラム左下の領域へ持っていくと終了となる．

3.4.3　感想・展望

難解プログラミング言語と呼ばれる Piet だが，実際にプログラムを書くと非常に論理的な動作と，職人芸に近しい要素を体感することができた．特に，以前関西情報系学生団体交流会で体験した時と比べ，roll 命令などの細やかな部分についても触れたため，実現したい処理とキャンバス上の色羅列がどの程度かけ離れているかを痛感しながら入力する，文字列ベースとは異なる感覚のプログラミングができた．いつかは処理系実装などができれば，と夢見ている．

4　全体総括と考察

文責: 山口　流星

単純であるほど奥が深いということに関して誰もが共感できる節があると思われるが，Piet もその一つに当たる．班員からはその奥深さに頭を抱える一面もありつつ，なおも楽しんで活動している様子が伺えた．班員らが

多忙な時期であることから活動時間の確保が難しくもあったが，目的としていたコーディングを行う力についてはある程度身に付いたと考えられる．また，難解という単語に怯むことなく，興味を持ったことに挑戦する姿勢を大きく評価したい．

4.1 Piet の考察

文責: 山口　流星

今回の活動では，Piet をより書きやすくするためのテクニックについて考える機会を幾度か設けた．その結果と経緯についてまとめる．

4.1.1 開発環境

Piet には公式の開発環境が存在しないため，開発環境を選定することから始まる．しかし，開発環境によっては公式の仕様と異なる可能性があり，また公式の仕様の揺れが固まった時期も定かではない．そのため今回は比較的新しくリリースされ，かつ非常に扱いやすい統合開発環境 Pidet を選定した．ただし，Pidet は Windows のみで扱えることに注意する必要がある．また，開発者は Pidet は obsolete と発言しており，後継の Pietron を推奨している点についても留意したい．Pietron は MacOS，Linux もサポートしているが，Pidet と比較して機能がまだ十分でない点を鑑みて，今回は Pidet を主軸の開発環境とし，Windows を保持していない班員は Pietron を使用した．

4.1.2 プログラムの終了形

初心者がコーディングする上で最初の壁となるのは終了条件を満たす形であった．正常な終了条件は，wait 命令を 8 回連続で実行するか，幾つかの白コーデル上における slide 命令の無限ループのどちらかでしかない．ここから次の 3 種類の終了形が便利であることを結論付けた．また，キャンバスサイズに余裕がある際は slide 命令で終了させることが極めて簡単である．

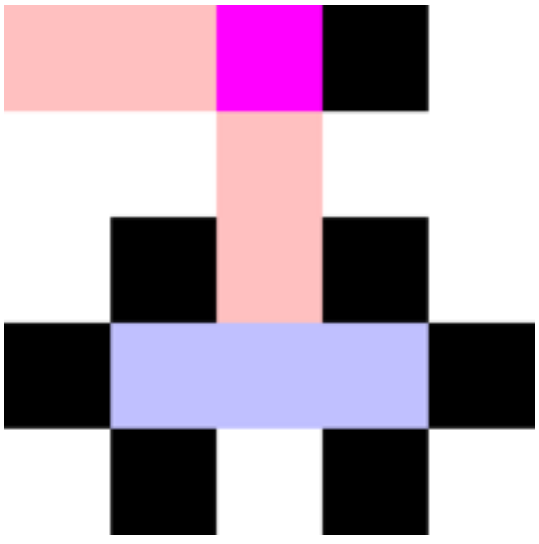

図 21: wait 命令の終了形 1

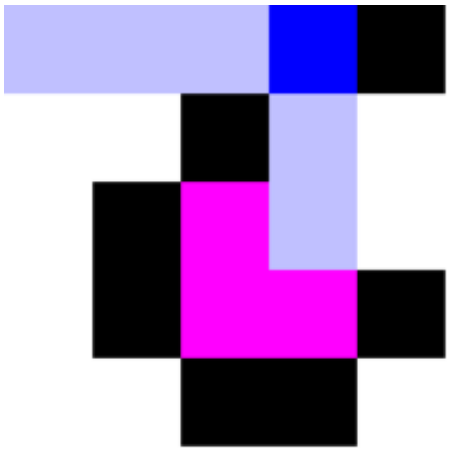

図 22: wait 命令の終了形 2

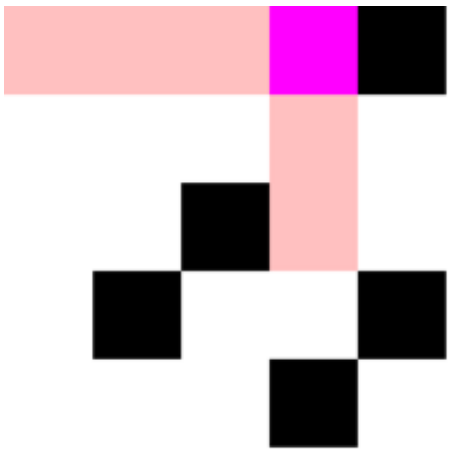

図 23: slide 命令の終了形

4.1.3　キャンバス端の処理

キャンバスの端は wait 命令として処理されるが，端に到達する 1 つ前で 1 コーデルの push 命令を行い，その後 pointer 命令を用いて DP を変更する書き方がある．この方法を用いると，CC を変更せずに DP を変更することが出来るほか，キャンバス端の処理が開発環境に依存しないことからも安全である．ただし，DP を変更する度にスペースが余分に必要になるため，キャンバスサイズが実行したい処理に対して適切であるかどうかをよく判断することが重要である．

4.1.4　ループの書き方

基本的にループは not 命令を用いることで記述するが，not 命令を 2 回続けて使用することでもループを記述することが出来る．not 命令を 1 回使用する場合，ループを抜けるときに pointer 命令によって DP 方向は 90 度回転するため，ループ処理がコンパクトである場合はソースコードの収まりが良くなりやすい．一方で not 命令を 2 回使用する場合，ループを抜けるときに pointer 命令によって DP 方向が変化しないため，終了形を見据えたコーディングが行いやすい．結果として，どちらが書きやすいかは描き手次第だと考えられるが，初心者の方がコーディングするのであれば，プログラムの終了が容易な not 命令を 2 回使用する形が良いのではないかと考える．

5　展望

文責: 山口　流星

今後，Piet をサークル内活動で触れる機会があるかどうかは定かでないが，今回制作したクロスステッチないし Piet のソースコードをスマートフォンなどで読み取り実行できる環境を制作することが望まれる．

参考文献

[1] 「Piet」 最終更新:2018/09/27 URL: http://www.dangermouse.net/esoteric/piet.html

[2] 「Piet Program Gallery」最終更新: 2019/11/20 URL: http://www.dangermouse.net/esoteric/piet/samples.html

[3] 「dnek/Pidet: IDE for Piet. - GitHub」 URL: https://github.com/dnek/Pidet/releases

[4] 「Piet のエディタを作った話」 URL: https://www.slideshare.net/KMC_JP/piet-46068527

[5] 「dnek/pietron: Cross-platform IDE for Piet. - GitHub」 URL: https://github.com/dnek/pietron/releases

[6] 「ドット絵でプログラミング!難解言語『Piet』勉強会」 作者: base64 URL: https://www.slideshare.net/KMC_JP/piet-80098546