# ビジュアルプログラミング環境を用いたアルゴリズム学習支援

伊　藤　栄　一　郎

## 1. はじめに

プログラムの骨格となるアルゴリズムを学ぶことは、効率的にプログラミングを行うために必要不可欠である。最近では、アルゴリズムを学んだり考案することが、論理的な思考力を養い、問題解決力を培うとも言われている。(1)

初学者は、文章や数式、疑似言語やプログラミング言語など、何らかの形で記述されたものを通じてアルゴリズムを学ぶが、これらの記述法には一長一短がある。

そこで本稿では、アルゴリズムの記述にビジュアルプログラミング環境を用いる方法を提案する。Scratch と呼ばれる子ども向けのプログラミング環境を改良し、初学者がアルゴリズムの作成や動作確認できるような支援環境を構築した。

## 2. アルゴリズムとアルゴリズム教育

### 2.1. アルゴリズムとは

一般的に、アルゴリズムとは、問題の解法のうち平易な方法で解を導く手順のことを指す。数学における解の定義とアルゴリズムとの違いは、解の定義が問題に対する解の関係や性質を示すのに対して、アルゴリズムが解を求める具体的な手順だという点である。(2)

古典的なアルゴリズムとして、2 整数の最大公約数を求める「ユークリッドの互除法」がある。最大公約数は小学校の算数で学ぶ基本的な概念であるが、その解法として 2 数に共通する約数を見つけ、2 数を割りながら最大公約数を見つける方法が説明されている。(3)

一方、ユークリッドの互除法では以下の補題を原理としている。(4)

m と n の最大公約数 gcd（m, n）は、n と m mod n の最大公約数 gcd(n, m mod n) に等しい。

この補題から最大公約数を求める関数 gcd (m, n) を図 1 のように定義できる。

**図 1：関数 gcd の定義（文献 11 の p73 より引用）**

$$\gcd(m,n)=\begin{cases} m & n=0 \text{ のとき} \\ \gcd(n, m \bmod n) & \text{それ以外のとき} \end{cases}$$

この定義に従って 323 と 204 の最大公約数を求める例を示す。

gcd (323, 204) = gcd (204, 119) = gcd (119, 85) = gcd (85, 34) = gcd (34, 17) = gcd (17, 0) = 17

結果として、剰余の計算と数値の比較により、323 と 204 の最大公約数が 17 であることがわかる。

小学校で学ぶ「共通する約数を見つける」方法でこの例題を解く場合は、323 と 204 に共通する約数を見つけるために素数を順に試す必要があり、最終的に 17 を見つけるのは自明ではない。ユークリッドの互除法ならば、剰余の計算を伴うものの少ない手間で効率よく解を求めることができる。

コンピュータのプロセッサが実行可能なのは

データ転送のほか四則演算や論理演算などごく限られた計算命令にすぎないため、限られた計算命令を適切に組み合わせて問題を正しく解決するプログラムを作成する必要がある。そのために、プログラムの骨格としてアルゴリズムが求められ、アルゴリズムをもとにプログラミング言語文法に従ってプログラムが作成される。ユークリッドの互除法のように単純な計算をもとにしたアルゴリズムを用いれば、そうでない場合に比べて効率的なプログラムを作成できる。

### 2.2. アルゴリズムの学習に関する研究

アルゴリズムを学ぶことは「既知のアルゴリズムについて学ぶ」「問題から適切なアルゴリズムを考案する」の2面があると考えられる。前者はユークリッドの互除法のように既に考案されたアルゴリズムを学んでプログラミングの場面で活用することである。既知のアルゴリズムを骨格とすれば、効率的にプログラムを作成できる。この面については情報教育の草創期から多く実践されている。[5]

後者はアルゴリズムを学ぶことで論理的思考力を養い、問題解決力を身につけることである。既知のアルゴリズムに適切なものがない場合、問題を理解し、限られた計算手段を適切に組み合わせて問題を解決する。このようなことが論理的思考力や問題解決力を高める役にたつと考えられている。[6][1]

初学者にとって、既知のアルゴリズムを覚えたりアルゴリズムを考案する際、アルゴリズムがどのように記述されているかは重要である。理解しやすい形でアルゴリズムを記述する方法については、さまざまな研究が行われている。いくつかの例を挙げると、唐澤は情報基礎教育向けの表記方法として疑似言語を提案した。[7] 河村らはPADを用いてアルゴリズムを記述する支援環境を開発した。[8]

### 2.3. 本稿の対象とするアルゴリズム

アルゴリズムといってもさまざまな問題を解く、多くのアルゴリズムが存在する。本稿では、整列アルゴリズム、探索アルゴリズムといった基本的なアルゴリズムを対象とする。

## 3. アルゴリズムの記述方法

既知のアルゴリズムはさまざまな形で表現されている。ここでは、一般的に用いられているアルゴリズム記述の方法について述べる。

### 3.1. 数式や文章によるもの

ユークリッドの互除法の説明における関数gcdの定義のように、数式としてアルゴリズムを記述する方法がある。関数gcdの定義のように再帰関数的な記述は、数式による記述がシンプルでわかりやすい。一方で、複雑な定義になると正しく数式を読み解くのが難しくなり、アルゴリズムの理解を妨げる。

文章によるアルゴリズムの記述についても、簡単なアルゴリズムであれば記述は容易で理解しやすいが、数行にわたって記述すれば読み解くのは容易ではない。

### 3.2. 疑似言語によるもの

文章による記述と後述するプログラミング言語による記述との中間的な方法として、疑似言語による記述がある。一定の文法規則を定めた上で、英語や日本語などの自然言語を用いるものや、プログラミング言語に似た表現に図形や記号を加えて直感的に表すものなど様々な形式がある。前者の例として大学入試センター試験の情報関係基礎で用いられているDNCLが挙げられる[9]。後者の例としては情報処理推進機構による基本情報技術者試験がある。[10]

例として、プログラミング言語に近い形式で記述されたアルゴリズムを図2に示す。C言語

に近い文法規則を持つが、数学的な記号が用いられている。[11]

図 2：選択法のアルゴリズム
（文献 11 の p36 より引用）

```
selection-sort( ){
    for (i = 1; i ≤ n − 1; i = i + 1){
        min ← i;
        for (j = i + 1; j ≤ n; j = j + 1){
            if (A[j] < A[min]){
                min ← j;
            }
        }
        swap(A[i], A[min])²;
    }
}
```

このアルゴリズムは、データの配列を順に走査して、最小の要素を配列の先頭の要素と交換する。次は配列の 2 番目以降を走査し、最小の要素を 2 番目の要素と入れ替える。以下同様の手順を続けていくことにより昇順のデータを得る手順が示されている。

アルゴリズムの学習支援のために、疑似言語をコンピュータ上で動かすような実装が存在する。中村らによるプログラミング環境 PEN は、DNCL を拡張したものでコンピュータ画面上に入力した DNCL によるプログラムを実行したり、作成を支援する機能を持っている。[12]

### 3.3. プログラミング言語によるもの

技術的な書籍では典型的なプログラミング言語やそのサブセットを用いてアルゴリズムを表現することが多い。プログラミング言語に習熟しているプログラマにとっては、厳密的で理解しやすい。実装上の詳細を省いているが、コンピュータ上でアルゴリズムの動作確認を行うことができる。[13][14]

### 3.4. アルゴリズム記述方法の評価基準

初学者がアルゴリズムを学習するのに適したアルゴリズム記述方法を検討するにあたり、それらを評価する際の基準として、読解の容易さ、記述の容易さ、アルゴリズムの動作確認の容易さを挙げたい。

読解の容易さとは、既にあるアルゴリズムを表現したときの読みやすさのことである。具体的には繰り返しや条件分岐の違いが明確かどうか、それらの範囲が明確かどうか、処理の記述が直感的で統一性があるかどうかである。

記述の容易さとは、自分でアルゴリズムを記述する際の手間の少なさのことである。また、記述したアルゴリズムの修正も容易でなければならない。

アルゴリズムの動作確認の容易さとは、自分で記述したアルゴリズムの動作を手軽に確認できることである。アルゴリズムの動作に加え、アルゴリズムが対象とするデータ自体の扱いやすさも重要である。

### 3.5. アルゴリズム記述方法の比較

アルゴリズムの記述方法によって、アルゴリズムの読解や記述にどのような影響をもたらすのだろうか。

文章による記述は、簡易なアルゴリズムならば初学者にとって理解しやすいが、複雑なアルゴリズムでは、数式や文章の高い読解能力が求められる。記述が容易でも厳密に表現しようとすると数式が増えて理解しにくくなり、平易に表現すると細かい部分の解釈に幅が生じる可能性がある。

疑似言語により表現されたアルゴリズムは、字下げなどを用いた構造的な表記により、複雑なアルゴリズムでも理解しやすい。PEN のような支援環境を用いれば、疑似言語によるアルゴリズムを実際に試すこともできる。しかし、文法規則が定められていればプログラミング言

語で記述するのと同様に、文法の理解が必要となり入力ミスなどのエラーが発生してしまう。

プログラミング言語をアルゴリズムの記述に用いるのは、アルゴリズムを厳密に定義できたり実際に動作させたりする点で良い。しかし、学習者にプログラミング言語文法や意味について習熟していることが求められる。初学者にはアルゴリズム学習に加えてプログラミング言語学習の負担を強いることになる。

## 4．ビジュアルプログラミング環境を用いたアルゴリズムの記述と読解

アルゴリズムの記述にはさまざまな方法があり、それぞれ一長一短を持つことがわかった。簡単に記述できて動作の仕組みを理解しやすい方法があれば、初学者にとってアルゴリズムの理解が容易になるだろう。

本稿では、初学者が既知のアルゴリズムについて学ぶ際の効果的な手法として、ビジュアルプログラミング環境を用いる方法を提案する。具体的には整列や探索などの基本的アルゴリズムの記述や実験を支援するために拡張したScratchを用いた。

### 4.1.Scratch

ビジュアルプログラミング環境とは、主にマウスやタブレットなどを用いて図形的な要素を操作することでプログラムを作成する環境のことである。Scratchは、2007年にマサチューセッツ工科大学のメディア・ラボにあるライフロング・キンダーガーテン・グループにより開発されたビジュアルプログラミング環境で、ブロックを組み合わせたプログラムを作ることができる。（図3）各国語に翻訳されており、2014年9月時点で400万人の登録ユーザーがいる。10歳以下から60代まで幅広い層のユーザーがいるが、10代前半の利用者が多く、学校などでの活用例もある。[15]

Scratchはインタラクティブなプログラムを開発するために、画像やアニメーション、音楽等を操作する処理が取り入れられている。基本的なデータ構造として「変数」「リスト」が、制御構造としては「条件分岐」「繰り返し」があり、典型的な手続き型言語のプログラムをブロックを組み合わせて作成することができる。

### 4.2.Scratchをアルゴリズム教育で用いる際の問題点

Scratchは初学者向けの優れたプログラミング環境であるが、一般的なアルゴリズム教育で用いるのに向いている点とそうでない点がある。以下でそれぞれ整理する。

Scratchが向いている点として主なものを挙げる。

⑴文法エラーが発生しない。

ブロックを組み合わせるタイプのビジュアルプログラミング環境は、文法的に無意味なプログラムは、ブロックとして組み合わせられないため、文法エラーが発生しない。ブロックの組み合わせ方が適切な言語文法を示唆するので、正しい書き方を経験的に学ぶことができる。

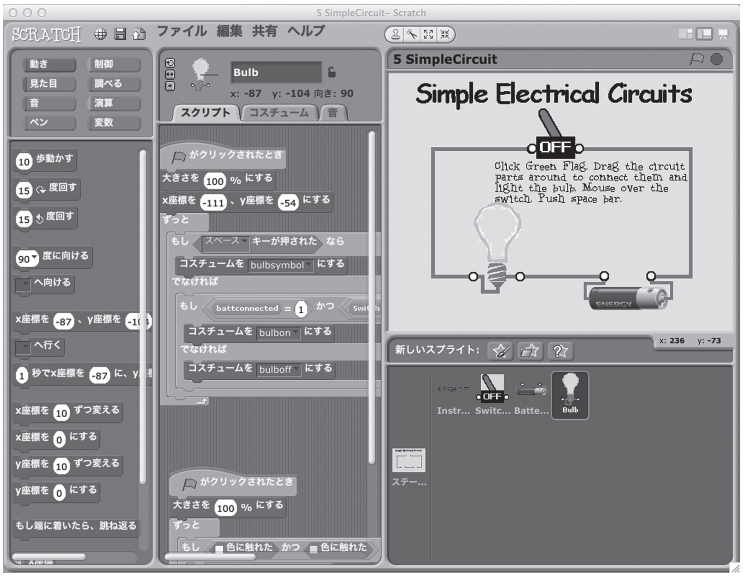
⑵データが可視化されている

データの名前や内容が画面上に表示される。データの変化を把握することはアルゴリズムの理解に重要であるが、Scratchではデータをリアルタイムに確認できるため、アルゴリズムの理解に役立つ。

⑶試行錯誤しやすい

典型的なアルゴリズム教育では、文章で記述されたものをプログラムとして作成することが求められるが、自由にブロックを組み合わせたり、作成したものを部分的に実行させて機能を

図 3：Scratch の画面




確認できるので試行錯誤が容易であり、アルゴリズムの構成的な理解につながる。

一方で、Scratch をアルゴリズム教育に利用する場合に、不便に感じられる点を挙げる。

⑴繰り返しの制御構造が貧弱である

典型的なプログラミング言語には、一定の範囲の繰り返し（for ループ）や、指定された条件における繰り返し（while ループ）、指定された条件に達するまでの繰り返し（until ループ）といった制御構造を持っており、アルゴリズム教育においてもこれらの制御構造を用いて説明されることが多い。しかし、Scratch には無限回の繰り返し、指定された回数の繰り返し、指定された条件に達するまでの繰り返しがあるものの、いわゆる for ループや while ループにあたるものがない。（図 4）

⑵リストの操作の貧弱である

Scratch は可変長のリストを持っており、数値や文字列、真理値を格納できる。リストを操作する機能としては、データの追加および挿入、データの削除および変更、データの参照、リスト長の取得、存在判定がある。（図 5）必要最小限の機能は備えているが、リストを頻繁に操作する場合、記述が複雑になってしまう。

図 4：Scratch の制御構造ブロック

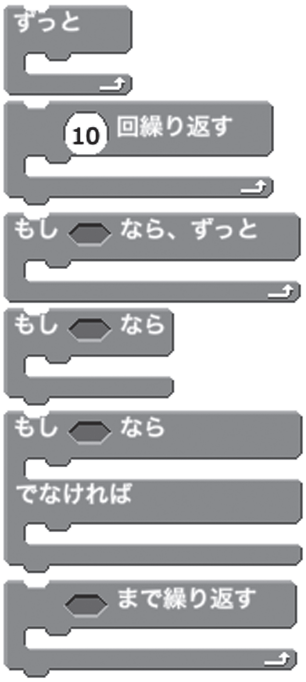
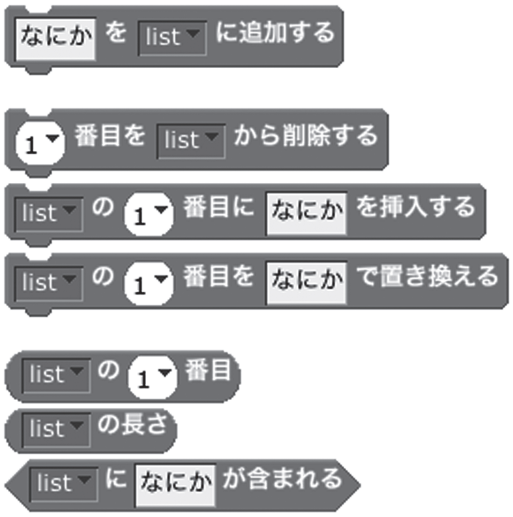


図 5：Scratch のリスト操作ブロック




⑶簡単なスクリプトでも多数のブロックが必要となる

多数のブロックが必要になると、操作が煩雑になり、作成したスクリプトが画面に収まらなくなる。アルゴリズムの本質とは異なる部分の記述が増えることで一覧性が悪くなり、結果として理解を妨げてしまう。

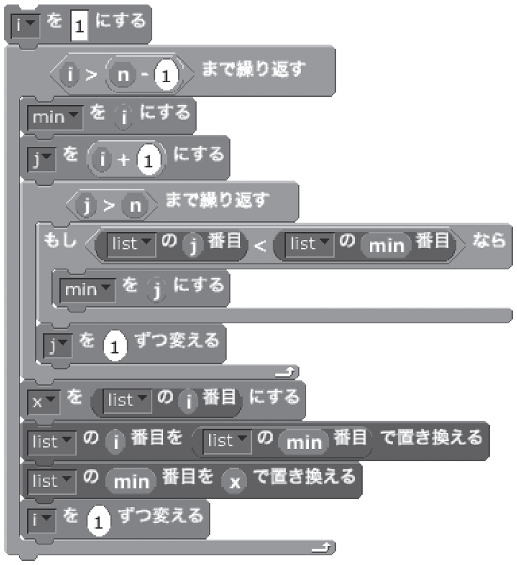
上記の問題点を具体的に示すために、図 2 の選択法アルゴリズムをオリジナルの Scartch で記述した例を図 6 に示す。

図 2 の選択法アルゴリズムでは、Scratch にはない for ループを until ループを用いて表すために、繰り返し判定の論理を逆にしている。また、リストの要素を交換するのに一時変数 x を用いて代入を用いて表現している。結果として元のアルゴリズムの本体部分が 9 行で表現されているのに対して Scratch では 36 個のブロック（制御構造に限れば 12 個のブロック）を必要とする。

### 4.3.Scratch に対する改良

筆者はこれらの問題を回避するために Scratch を拡張することにした。Scratch はソースコードが公開されており、MIT ライセンスのもとで改造や配布が認められている。公開されている Scratch v1.4 のソースである Based on Scratch に新たなブロックを追加することで改良した。追加した新たなブロックは、リスト要素の交換、リストの生成、リストのシャッフル、while ループの 4 つである。

図 6：オリジナルの Scratch で記述した選択法アルゴリズム




リスト要素の交換ブロックは、先ほどのアルゴリズムにもあるリスト内のデータの入れ替えを行うものである。整列アルゴリズムなどでは頻繁に用いられており、一時変数を用いることはアルゴリズムの本質には関わらないため追加することにした。オリジナル版では i 番目と j 番目のデータを入れ替えるために計 10 個のブロックを必要とするが（図 7 左）、新たなブロックの導入により 3 個のブロックで表現でき

図 7：リスト要素の交換

（左：オリジナル版　右：改良版）

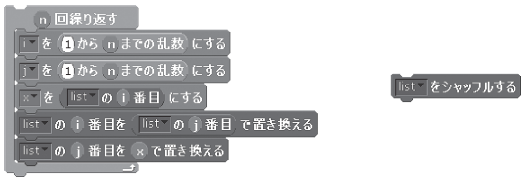
る。（図 7 右）

アルゴリズムの動作を確認するにはリストにデータを追加する必要があるため、与えられた範囲の数値をリストに加えるリストの生成ブロックを追加した。例えば、1 から n までの数をリストに追加するスクリプトは 6 個のブロックが必要なのに対して、図 8 のように 2 個のブロックへ短縮できる。

**図 8：1 から n までの数をリストに加えるスクリプト（左：オリジナル版　右：改良版）**


リストのシャッフルはリスト内のデータをランダムに並べ替えるものである。リストのデータを小さい順に整列するアルゴリズムを試す際、あらかじめランダムな並びのデータが必要となる。オリジナルの Scratch では図 9 左のように 18 個のブロックを必要としたが、図 9 右のように新しい 1 個のブロックのみでよい。

**図 9：リストのシャッフル**
**（左：オリジナル版　右：改良版）**




その他、アルゴリズムで頻繁に用いられる while ループを追加した。

### 4.4. 改良に対する効果

4 つのブロックの追加によってアルゴリズムの記述をかなり減らすことができる。典型的なアルゴリズムの記述でのブロック数の違いを表 2 に示す。

**表 2：典型的なアルゴリズムにおいて用いたブロックの個数**

| アルゴリズム | 逐次探索 | 二分探索 |
|---|---|---|
| オリジナル版 | 14 | 33 |
| 改良版 | 15 | 34 |

| アルゴリズム | 選択法 | バブルソート |
|---|---|---|
| オリジナル版 | 34 | 31 |
| 改良版 | 28 | 22 |

逐次探索と二分探索で改良版のブロック数が増えている理由は、元にしたアルゴリズムが while ループを用いており、繰り返し判定条件で論理を反転した until ループの方が while ループよりも少ないブロック数で記述できたためである。

今回 Scratch に加えた変更は、4 つのブロックを加えただけであるが、必要なブロック数を減らすことで、初学者のアルゴリズムの読解に対する負担を軽減できると考えられる。

## 5. 授業における活用

筆者が担当している経営情報学部の 2 年次配当科目である「情報処理概論」の第 7 回から第 9 回までの 3 回において、探索や整列のアルゴリズムを学ぶために、ブロックを追加した Scratch（以下、拡張版 Scratch）を用いた演習を行った。

従来の授業では、文章や疑似言語によるアルゴリズムを板書や配布プリントに記述して講義を行っていた。一方的な説明になりやすく学生の反応も悪かった。

今年度実施した授業では、改良版 Scratch を

用いたスクリプトプログラムで基本的なアルゴリズムを記述した。学生は自分のノート PC に拡張版 Scratch をインストールし、ブロックを組み合わせてアルゴリズムを作成する。

例えばリスト内の特定のデータを探索するアルゴリズムの場合、まずリストの内容を順に表示するスクリプト（図 10）を提示する。次に、データが特定の値になったら停止させるようにスクリプトを改良させる。こうして出来上がったものは「逐次探索（線形探索）」と呼ばれる基本的な探索アルゴリズム（図 11）である。

**図 10：リスト内容を表示する**

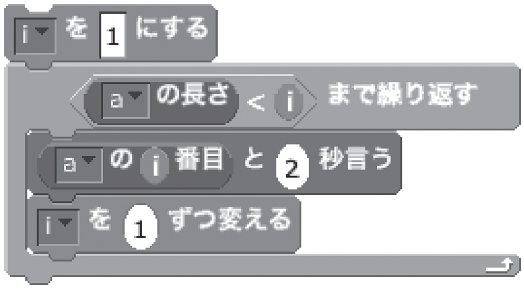
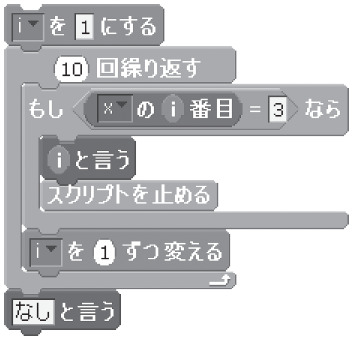



**図 11：逐次探索のアルゴリズム**




整列アルゴリズムでは、シャッフルされたリストに対して、最小値をリストの先頭に移動させるスクリプト（図 12）を提示する。その後、2 番目、3 番目の値を移動させるようにスクリプトを拡張させていく。最終的に「選択法」と呼ばれる整列アルゴリズムを作成する。（図 13）

いずれの授業でも、学生が基本的な構造を学んだ後、少しずつ拡張することで構成的にアルゴリズムを理解する授業内容になった。

**図 12：最小値を見つけるアルゴリズム**

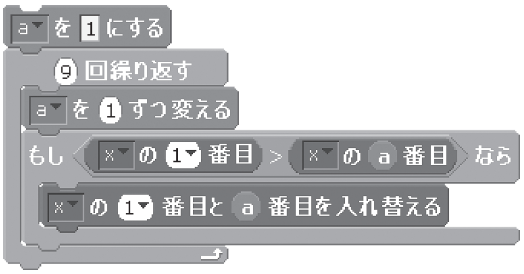
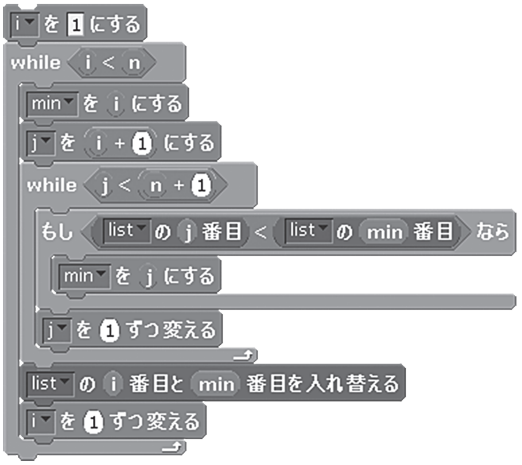



**図 13：選択法のアルゴリズム**




## 6. おわりに

本稿では基本的なアルゴリズムの学習の支援のために、ビジュアルプログラミング環境を用いる方法を提案した。拡張版 Scratch を用いる利点は、文法を覚えるといった煩雑さなしにアルゴリズムを組み立て、実際にその動作を試せるということである。部分的な機能を個別に確かめたり、それらを組み合わせて全体的な動作をさせたりと試行錯誤が簡単である。授業では「テキストを写したら終わり」から、「動くまで試行錯誤する」ことへの変化が学生のアルゴリ

ズムの理解を促しているように見られた。貸与ノート PC を用いて自宅でも試行錯誤できるので、授業で作成したスクリプトを自宅で改良するような課題を提示することができる。

現時点では改良版 Scratch を用いた場合とそうでない場合の理解度に関する評価を行っていないため、今後は学生の評価を行った上で更なる改善をはかっていきたい。

## 参考文献


1）山本樹他：“アルゴリズム的思考による論理的な文章作成力養成のための一検討”、教育システム情報学会研究報告、Vol.25, No.2（2010 年 7 月）

2）ジェラルド・サスマン他：“計算機プログラムの構造と解釈　第二版”、（株）ピアソン・エデュケーション（2000 年 2 月）

3）文部科学省：“新学習指導要領・生きる力　高等学校学習指導要領”

4）浅野哲夫：“アルゴリズム・サイエンス : 入口からの超入門”、共立出版株式会社（2006 年 10 月）

5）浅野考平他：“アルゴリズム教育再考”、情報処理学会研究報告（2014 年 6 月）

6）飯田周作他：“アルゴリズム的思考法の教育”、情報処理学会研究報告（2008 年 2 月）

7）唐澤博：“情報基礎教育のためのアルゴリズム記述形式の提案”、情報処理学会研究報告「コンピュータと教育」（1995 年 7 月）

8）河村一樹：“文科系向けプログラミング教育支援システム JPADet の設計と開発”、情報処理学会研究報告「コンピュータと教育」（2001 年 12 月）

9）情報処理推進機構：“基本情報技術者試験 : 平成 26 年度春期試験問題冊子”

10）西田知博他、“大学入試センター試験とプログラミング言語”、情報処理 Vol.50 No.10（2009 年 10 月）

11）広瀬貞樹：“あるごりずむ”、（株）近代科学社（2006 年 11 月）

12）中村亮太他：“プログラミング入門教育用学習環境 PEN”、情報処理学会研究報告（2005 年 10 月）

13）Geore T. Heineman 他：“アルゴリズム・クイックリファレンス”、（株）オライリー・ジャパン（2010 年 4 月）

14）奥村晴彦：“C 言語によるアルゴリズム事典”、（株）技術評論社（1991 年 2 月）

15）“Scratch―想像、プログラム、共有”、http://scratch.mit.edu/