情報処理学会論文誌 Vol.55 No.1 57–71 (Jan. 2014)

ビジュアル-Java相互変換によるシームレスな言語移行を指向したプログラミング学習環境の提案と評価

松澤 芳昭[1,a)] 保井 元[2] 杉浦 学[3] 酒井 三四郎[1]

受付日 2013年3月18日, 採録日 2013年10月9日

概要：近年，Squeak や Scratch をはじめとするビジュアルプログラミング言語による入門教育の実践が広く行われている．しかしながら，既存の環境は C や Java などのテキスト型言語への移行が考慮されていないため，学習の発展性に課題がある．本研究では，OpenBlocks フレームワークを利用して，ブロック型ビジュアル言語とテキスト型言語（Java）の相互変換ができるプログラミング教育環境「BlockEditor」を開発した．文科系大学生向けプログラミング入門授業の学生約 100 名に対して実証実験を行った．15 週の授業で課した 36 題の課題解答過程において，学習者に 2 つの言語を任意に選択できる環境を与え，言語選択率を測定した．その結果，プログラミング学習の進行に沿ってブロック言語から Java へ緩やかに移行していくこと，およびそのタイミングには個人差があることが定量的に示された．プログラミングに苦手意識を持つ学生ほどブロック型言語の選択率が高く，言語の相互変換環境が言語の交ぜ書きを促進し，Java プログラム構築能力習得の足場かけとなることが示された．

キーワード：ビジュアルプログラミング，Java，プログラミング教育，言語移行，足場かけ

Seamless Language Migration in Introductory Programming Education through Mutual Language Translation between Visual and Java

Yoshiaki Matsuzawa[1,a)] Hajime Yasui[2] Manabu Sugiura[3] Sanshiro Sakai[1]

Received: March 18, 2013, Accepted: October 9, 2013

Abstract: In the past decade, improvements to the environment of introductory programming education by visual programming language have been made by Squeak, Scratch, etc. However, there is still a problem for migration to text-based programming languages such as C and Java. Hence, using the OpenBlocks framework proposed by MIT, we developed a system named BlockEditor, which has functions to translate block language and Java both ways. We conducted an empirical study of this system in an introductory programming course for approximately one hundred university students. When students were given opportunities to select the language to solve their programming assignments, we traced their selection by tracking working time with BlockEditor or Java for each individual student. As a result, we succeeded in illustrating the nature of the seamless migration from block language to Java, and found there is great diversity for timing and speed of the migration by each individual. Additionally, we found the selection rate of the block language by students with low self-evaluation for their skills was significantly higher than students with high self-evaluation. The BlockEditor could scaffold them by promoting mixed writing with block language and Java in their migration phase.

Keywords: visual programming, Java, programming educationn, language migration, scaffolding

[1] 静岡大学情報学部
Faculty of Informatics, Shizuoka University, Hamamatsu, Shizuoka 432–8011, Japan
[2] 静岡大学大学院情報学研究科
Graduate School of Informatics, Shizuoka University, Hamamatsu, Shizuoka 432–8011, Japan
[3] 慶應義塾大学環境情報学部
Faculty of Environmental Information, Keio University, Fujisawa, Kanagawa 252–8520, Japan
[a)] matsuzawa@inf.shizuoka.ac.jp

1. はじめに

プログラミングの教育法に関する研究，およびその支援教育システムの開発がすすめられている．その中でも注目を集めているのは，Squeak [1] や Scratch [2] などのビジュアルプログラミング言語によるアプローチである．ビジュアルプログラミング言語とは，プログラムをテキスト記述によって作成する（以下，「テキスト型言語」と称す）のではなく，GUI（Graphical User Interface）よって作成する方式の言語を指すもので，以下に説明するブロック型のほかに，ラダー言語やフローチャート図などの様々な方式が含まれる．

本研究が対象とするのは Squeak Etoys や Scratch が採用している Building-block approach [3]（以下，本論文では「ブロック型」と称す）によるビジュアルプログラミング言語である．ブロック型言語では，プログラム言語の構成要素（命令，分岐，変数など）を表現する「ブロック*1」をジグソーパズルのように組み合わせることでプログラムを作成する [3]．命令やデータ型の種類によって形状が異なり，文法上正しい位置にのみブロックどうしがはめられるように設計されているため，文法エラーを回避することが可能である．

文科系，理科系の学習者を問わず，プログラミング入門教育の主目的は，特定のプログラミング言語の習得ではなく，「アルゴリズム構築能力」すなわち「自分ができる何らかの仕事を他人や機械が実行可能なように，十分詳細に，完全に，その手続きを記述すること [5]」の習得であると，我々は考えている．プログラミング言語の習得そのものはこの目的の手段であり，重要なのは言語からは独立するプログラム構成要素の概念について理解することである．このような目的のプログラミング教育では，アルゴリズム構築に集中するために，ブロック型言語を使用することが有効であると考えられている [6]．

ブロック型言語によって学習者にプログラム構成要素の概念形成がなされていれば，その後に学習する言語への転移も容易となるはずである．しかしながら，ブロック型言語によるプログラミング体験が，それ以後の学習者のプログラミング能力の発展に寄与したとされる明示的なデータは示されていない．兼宗らは中学生向けのプログラミング教育言語としてテキスト型を選択した理由の 1 つとして「他のプログラミング言語へ発展させやすい」ことをあげている [7]．ブロック型言語による学習者が実際にテキスト記述型言語へ移行するには，文法エラーへの対処について学習する支援環境が必要である．「高校生や大学生レベルではタイルなどの GUI 部品を組み立てるだけのプログラミングでは不十分である」という主張もある [8]．この主張の根拠は不明瞭であるが，両言語の見た目が異なることで関係性を見いだしにくいことが学習者の転移を阻害したり，学習への動機付けをそいだりする理由となる可能性は考えられる．

そこで本研究では，ブロック型言語からテキスト型言語（Java）への移行をサポートするシステムを提案する．この目的を達成するシステムについて，MIT（Massachusetts Institute of Technology）で開発された OpenBlocks フレームワーク [9] を利用し開発を行った．本システムの特徴はブロック型言語と Java の相互変換ができることである．ブロック型言語は学習者がアルゴリズム構築とプログラム構成要素を理解するための足場かけ（Scaffolding [10]）として利用し，その後，Java によるプログラミングへ，学習者が任意のタイミングで，小規模の部分から全体へとシームレスに移行することができるよう設計を行った．

本論文は全 7 章からなる．2 章で先行研究のレビューを行う．3 章で本研究で提案・開発するシステムの設計を述べる．4 章で本システムの評価実験の方法，5 章ではその結果について述べる．6 章で結果を吟味して，本研究の有効範囲について議論する．7 章はまとめである．

2. 先行研究

教育現場でブロック型言語を用いたプログラミング教育 [6], [11], [12] や，ブロック型言語の開発 [1], [2], [13], [14], [15] が行われている．ブロック型言語はテキスト型言語と異なり文法エラーを回避できることから，プログラミング初学者にとって使いやすい言語と考えられている．一方でドリトル [16] や PEN [8]，Nigari [17] のように，テキスト型言語のプログラム入門環境も提案されている．文法エラーを起こりにくくしたり，入力支援をしたりすることによってエラーの問題を回避して，テキスト型言語のプロセスが入門で身につくことが長所である．その一方で，ブロック型言語を用いた教育と比較しているデータはなく，精緻な評価はこの分野全体の課題である．

ブロック型言語からテキスト型言語への移行を支援する研究として，Pasternak の研究 [18] がある．この研究では JubJub と呼ばれるシステムに，ブロック型言語で作られたプログラムを Java に変換する機能が提案されている．しかしながら，このシステムはブロック型言語から Java への移行が単方向のみであり，Java からブロック型言語への変換ができない．Google Blockly [15] も Javascript や Python などの言語に変換可能であるが，逆変換ができないのは同様である．Warth らは，TileScript と呼ばれるブロック型言語と Javascript の相互変換システムを試作している [19] が，相互変換の技術的検証にとどまっており，教育現場で利用可能なブロック編集システムの提案と評価は行われていない．

岡田ら [20] は日本語ブロック型ビジュアルプログラミン

*1 Squeak コミュニティでは「タイル」と呼ばれている [4].

グ環境「ことだま on Squeak」を用いて，プログラミング入門教育を行ったあと，Java で同様の内容を行うという授業を実施している．岡田らは，最初に論理的に考える力を養った後，プログラムを作成する力を養うという授業を目指している．しかしながら，この授業ではブロック型言語と Java のフェーズが完全に分離しており，シームレスな移行になっていない．ブロック型言語の効果についての評価も未達である．

Harvey らは Scratch を拡張し，関数（メソッド）を作ることができる BYOB（Build Your Own Block）機能を提案している [21]．この目的は，初学者用言語である Scratch を利用して初学者から習熟者まで幅広い層をサポートすることである．Harvey らは，著名な構造的プログラミングの教科書（SICP：Structure and Interpretation of Computer Programs [22]）を引用し，構造化サポートの重要性を述べている．しかしながら，テキスト型言語への移行は主目的ではなく，より高度な構造化プログラミングまでブロック型言語で支援しようとする試みであるため，本研究の目的とは異なる．

PAD（Problem Analysis Diagram）と C 言語を併用した学習支援システムを開発した石田らの研究 [23] がある．この研究では，PAD と C 言語のソースコードを相互変換できるようにしている．支援対象はプログラミング初学者で，プログラミング言語は C 言語である．永原らの研究 [24] では PAD 上で日本語を利用した記述を可能とし，その PAD と C 言語の相互変換も行えるようにしている．日本語部は C 言語ではコメントとして出力されるが，コメントが表示されるだけであるのでプログラムのまとまりを把握しにくいという問題がある．これらの研究は評価実験を行っていないため，有用な機能なのか検証できていない．

3. システムの設計

3.1 「BlockEditor」と機能概要

本研究ではプログラミング初学者を対象としたプログラミング教育支援システム「BlockEditor」を提案する．BlockEditor はブロック型言語からテキスト型言語へのシームレスな移行を支援する．BlockEditor の支援対象として想定している学習者は，初めてプログラミングを学習する者（初学者）である．本システムは大まかに以下の 3 つの機能を持つ．

機能 1 「ブロック言語[*2]」によるプログラム記述機能：ユーザがブロック型言語を用いてプログラムを作ることができる．

機能 2 ブロック言語と Java の相互変換機能：ユーザが任意のタイミングで，Java プログラムを用いてプログラムを作ることができる．

機能 3 抽象化ブロック機能：ユーザがブロック言語を利用して部分プログラムをまとめることができる．

機能 1 は，アルゴリズム構築の学習に集中するためのビジュアル言語の基本機能である．我々がブロック言語を設計する際に，この利点を失わないことを考慮している．

機能 2 は，ユーザがアルゴリズム構築を学習した後，理解進行にともなって，徐々に Java でプログラムを記述できるようにする足場かけの機能である．言語が相互変換できることによって，プログラムをブロック言語と Java の両方を行き来して書く「交ぜ書き」[*3]が促進されるという仮説がある．

機能 3 は，ブロック言語の欠点である，プログラムが一定規模（20 行）以上になった際に全体の見通しが悪くなることを防ぐ目的がある．なお，プログラム構造化の教育も目的であるが．本論文では言語移行に焦点を絞るため，この目的での評価は行わない．

3.2 機能 1：ブロック言語によるプログラム記述機能

3.2.1 インタフェース

開発した「BlockEditor」の外観を**図 1** に示す．図 1 の左下のウインドウが BlockEditor である．BlockEditor では中央の作業区画でプログラムを作る．プログラムを作るときは，左の区画にある各カテゴリからブロックを取り出し，作業区画でブロックを組み立てる．Java に変換するときは，ウインドウの上部にある「Java 出力」というボタンを押すと，BlockEditor で組み立てたプログラムが Java に変換され，初学者向けに開発された Java 開発環境（以下，Java エディタ）に出力される．図 1 の右上の画像が Java エディタである．BlockEditor 上で「Java 出力して実行」を押すと，Java が出力され，組み立てたプログラムが実行される．

3.2.2 ブロック言語の設計

BlockEditor の基礎部分は OpenBlocks フレームワーク [9] を利用して開発を行った．設計全般で考慮したのはブロックラベルの日本語化である．OpenBlocks 標準のブロックラベルはすべて英語であった．日本人にとってプログラムが読解しやすくなるようにするため，「ことだま on Squeak」[20] を参考にしてブロックの日本語化を行った．

変数については，Java のデータ型をそれぞれブロックとした．Scratch や Squeak などの子供向け教育用プログラミング環境のブロック型言語は，変数の型を気にすることなくプログラムの実装を行うようにするため，数値型，文字列型の変数のみ用意するという配慮がなされている．しかしながら，BlockEditor では Java 変換の必要性を優先するため，Java のデータ型の変数をそれぞれの型のブロック

[*2] 本論文では我々の開発したブロック型言語を「ブロック言語」と呼び，一般的な「ブロック型言語」と区別する．

[*3] 本論文では，最終的なプログラム形態で言語が交ざっているという意味ではなく，記述過程でブロック言語，Java の両方を行き来して書かれたことを「交ぜ書き」と呼ぶ．

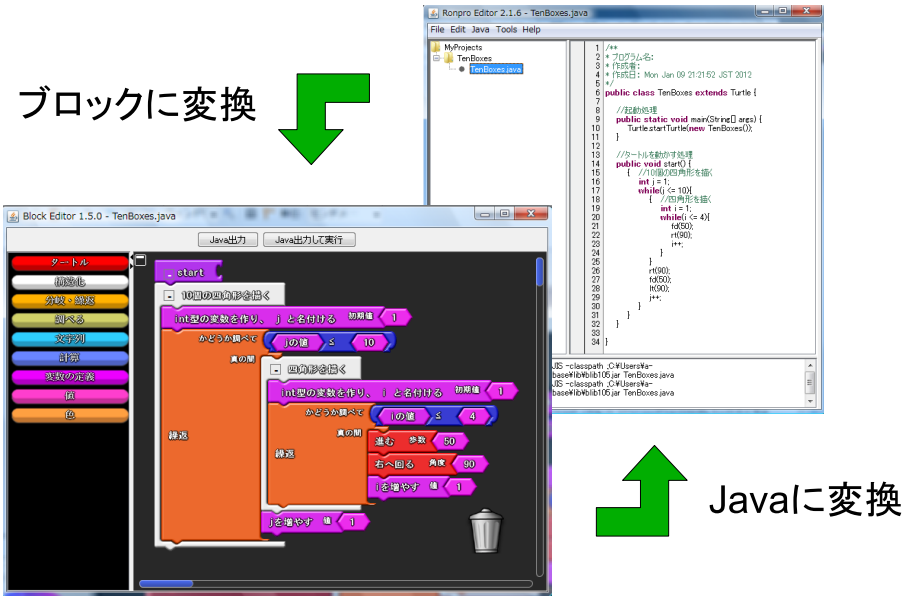


図 1 BlockEditor のインタフェース
Fig. 1 The interface of BlockEditor.

としている．現在は int 型，double 型，boolean 型，String 型が利用できる．

繰返し文，分岐文については，統一形式に変換しブロックとして扱う．たとえば，Java の繰返し文は for 文と while 文の 2 種類があり，分岐文は if 文，if-else 文，switch 文の 3 種類がある．BlockEditor では for 文や if 文を書き戻した際も問題なくブロックに変換し，while 文や if-else 文に統一した形で変換し，ブロックとして扱われる．ただし，統一形式に変換されたブロックは変換された形式で Java 出力される．したがって，たとえば for 文をブロックに変換し，Java に書き戻すと同等の while 文として出力される．

メソッドの定義と呼び出しは OpenBlocks 標準と同様の仕様である．オブジェクト生成と呼び出しに関しては，OpenBlocks に当該機能の実装がないため，メソッド呼び出しと制御文タイルを参考に追加実装した．メソッドとオブジェクト関係のブロック形状を示す例として，西暦和暦変換プログラムと集合データ構造のプログラムサンプルを付録に付す．なお，インスタンスメソッドやクラスの定義に関しては現在対応していない．

このように設計した理由として以下の 2 点がある．第 1 の理由として，本システムの対象者は Java によるプログラム構築能力の習得を目指す学習者である．そのため，ブロック言語から Java に移行したとき，違和感なく Java を利用できるように設計している．第 2 の理由として，ブロック型言語の利点として文法エラーが起きないということがあげられる．ブロック言語から Java に変換したとき，文法エラーが起きるとブロック型言語の利点が活かされない．特に型の問題に関しては，Java に変換した際に文法エラーが起きないことを優先した結果の設計である．

3.3 機能 2：ブロック言語と Java の相互変換機能

3.3.1 ブロック言語から Java への変換

BlockEditor は，組み立てられたブロック言語を Java に変換する．この変換システムによって，ブロック言語で構築したプログラムが Java ではどのように記述されるかがユーザに提示される．内部的に，OpenBlock フレームワークで組み立てられたプログラムは XML 形式で出力される．BlockEditor ではこの出力された XML 形式を変換することで，Java ソースコードを出力する．

例として「タートルグラフィックスで 10 個の四角を描くプログラム」の BlockEditor の変換例を示す．我々のプログラミング入門教育カリキュラムでは，Java のライブラリとして実装したタートルグラフィックス [25] を利用している．**図 2** に示された Java のソースコードは，図 1 で示されている BlockEditor で作られたプログラムを変換したものである．タートルグラフィックスのライブラリの命令は，(1)「右へ回る」ブロックは Java の「rt」命令，(2)「進む」ブロックは Java の「fd」命令へと変換されている．「繰返」ブロックは while 文に変換されている．

BlockEditor が出力するコードには，for 文や if 文が統一形式になってしまう，計算式の冗長な括弧が追加される，後に述べる抽象化ブロックのコメント出力方法が特定の形式になるなど，若干の「クセ」はある．ただし，付録の例に示したように，特に可読性に問題がある Java コードではないと我々は評価している．BlockEditor は Java 言語文法自体の習得が目的ではなく，プログラム手段としての Java 習得を支援するもので，Java の専門教育のために利用することは現在は考えていない．このような理由で，入門教育修了後に学習を発展させるための基礎レベルの Java として，適当なソースが出力されていると我々は評価している．

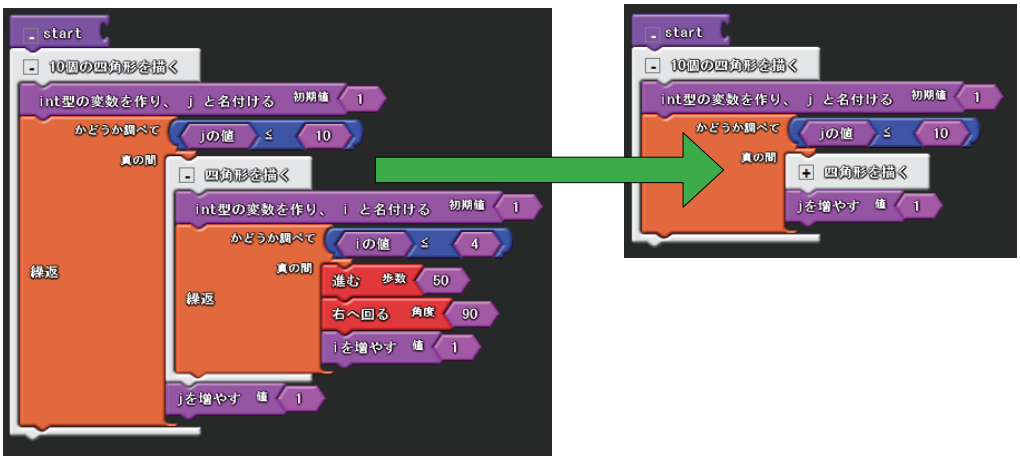


図 3 抽象化ブロックの折りたたみ機能
Fig. 3 The demonstration of collapsing “abstraction block”.

```
1  public class TenBoxes extends Turtle {
2    public void start() {
3      { //10 個の四角形を描く
4        int j = 1;
5        while(j <= 10){
6          { //四角形を描く
7            int i = 1;
8            while(i <= 4){
9              fd(50);
10             rt(90);
11             i++;
12           }
13         }
14         rt(90);
15         fd(50);
16         lt(90);
17         j++;
18       }
19     }
20   }
21 }
```

図 2 変換された Java ソース例
Fig. 2 The converted Java source code.

3.3.2 Java からブロック型言語への変換

BlockEditor は，Java をブロック言語に変換する．この変換システムによって，Java でしたプログラムがブロック言語として BlockEditor 上で構築され，ユーザが編集できるようになる．内部的には，Java のソースコードを抽象構文木に変換し，そこから OpenBlock 形式の XML 形式に変換している．

Java からブロック言語に変換する際，文法エラーが存在した状態では変換ができない．エラーを無視してブロックに変換することも可能であるが，初学者にとって，無視された部分を特定するのが難しいことを考慮した設計である．

3.4 機能 3：抽象化ブロック機能

3.4.1 抽象化ブロック

BlockEditor では，OpenBlocks フレームワークを拡張し，ブロック言語においてプログラムを構造的なまとまりごとに分けることができるブロック「抽象化ブロック」を利用できるようにした．抽象化ブロックの例として，**図 3** のプログラムの「10 個の四角形を描く」，「四角形を描く」の 2 カ所では抽象化ブロックが使用されている．抽象化ブロックの中に他の命令ブロックを構築することができる．抽象化ブロックもその対象となるので，入れ子が可能である．ブロック上のバーに自然言語でコメントを記述することができる．

3.4.2 折りたたみ機能

抽象化ブロックは，その中に構築したブロックを隠し，折りたたむことができる．抽象化ブロックの中に構築した命令ブロックが多くなった場合や，段階的にプログラムを構築するために抽象化ブロックを入れ子にすることになったときなどに利用できる．図 3 は折りたたみ機能を利用したときの様子である．図 3 のプログラムでは，「四角形を描く」というコメントが付けられた抽象化ブロックを折りたたんでいる．

折りたたまれたブロックは 1 つの命令のようにブロック構築ができるので，プログラム抽象化の基本である，「複数のものを 1 つのものとしてまとめ，名前をつけて 1 つのものとして扱う」[22] という機能が扱えるようになる．本機能はブロック言語において，1 つのブロックの大きさがテキストと比較して大きく，プログラムが一定規模（20 行）以上になった際に全体の見通しが悪くなることを防ぐ役割も果たす．

3.4.3 抽象化ブロックと Java の相互変換

抽象化ブロックの Java ソースへの変換は，Java のブロック機能とコメント機能を利用している．Java ブロックの中にそれぞれの構造のソースコードを記述し，そのブロッ

クの始まりの括弧の右隣に構造の目的をコメントで記述する．図 2 では，図 3 のプログラムの「10 個の四角形を描く」,「四角形を描く」ブロックを変換した例を示している．

このブロックとコメントの構造が維持されることによって，Java から抽象化ブロックへの変換が可能である．Java で変更したコメントは，抽象化ブロック上のコメントとして反映される．

4. 実験方法

4.1 実験目的と仮説

実験目的は，提案する BlockEditor の利用によってブロック言語から Java へシームレスな移行が可能になるかを評価することである．「実際の教育現場で，ブロック言語，Java を学生が任意に選択できるようにしたら，学生はどのように言語を選択するのか」というのが本研究の Research Question で，その仮説は以下の 2 つである．

仮説 1 最初はブロック言語を選択し，理解進行にともなって，交ぜ書き状態を経て，最終的に Java を選択するようになる．

仮説 2 移行のタイミングには個人差がある．苦手意識を持つ学生ほど，ブロック言語を選択する．

4.2 実施環境

著者所属学部で 2012 年度後期に行われた，文科系の 1 年生を対象にした授業「プログラミング」の全編で BlockEditor の使用実験を行った．実施環境の詳細を**表 1** に示す．この講義は 110 名の学生が受講しており，教員 2 名と 6 名のティーチングアシスタントが配置されている．初学者向けの入門科目であるが，全員前期の「コンピュータ入門」科目で「ことだま on Squeak」によるビジュアルプログラミング体験を 4 週間受講済みである．

1 章で述べたように，プログラミング入門教育の主目的である「基礎的なプログラムの動作を理解し，自力でアルゴリズムを構築して，プログラミング言語を用いてそれを十分に，正確に記述できるようになること」が本授業における学習目標である．ただし受講する学生の 2 年生からの進路は大きく 2 つに分かれている．その 1 つは IS（Information Systems）コースであり，発展的な Java プログラミングの単位が必修として要求される．もう 1 つは ID（Information Design）という文科系のコースである．ID へ進路をとる学生にとって当科目は教養としてプログラミングを学習するという位置づけであり，その後一生プログラミングの経験を持たない可能性も高い．このような学生たちに対して，苦手意識を植え付けることがないように留意しながら，後に発展可能なプログラミング能力を授けるのが，本授業の教授目標である．

「プログラミング」の全体のカリキュラムについて，各授業回の内容をまとめて**表 2** に示す．第 1 回から第 4 回の授業ではタートルグラフィックスを利用した図形描画，第 5 回と第 6 回の授業でアニメーション作品の制作，その後中間課題として個人で自由作品を作成する．後半は，メソッドおよび集合データ構造とアルゴリズムについての基礎について学習する内容である．最終回はグループによるミニ・プロジェクトの発表会である．

本授業で用いられる基本的な言語は Java であり，学習者にはブロック言語も併用できる環境が与えられた．ただし，「プログラミング言語として Java を用いるが，多くのプログラミング言語で共通に用いられる基本的な考え方を学習するのが目標である」ことを学生に示した．Java が習得できるとよいが，アルゴリズム構築ができるようになるのであれば最後までブロック言語でも問題はないという方針も学生に示した．教材で提供する例題プログラムについては，ブロック言語バージョンと Java バージョンの両方が用意された．毎週 180 分の授業時間のうち，60 分ほど実施

表 1 実験の実施環境
Table 1 The environment of the experimental study.

対象学部	情報学部社会学科（文科系）
対象授業	プログラミング（全 15 回）
前提条件	コンピュータ入門で Squeak プログラミングを 4 週間体験
受講者数と学年	110 名（1 年：108 名 2 年：2 名）
授業時間数	90 分 × 2 コマ × 15 週
指導体制	教員 2 名，アシスタント 6 名

表 2 「プログラミング」授業のカリキュラムと課題数
Table 2 The curriculum of the introductory programming course.

授業回	内容	B	A	J
第 1 回	環境設定	-	-	-
第 2 回	タートルグラフィックスの基本的な命令	1	3	1
第 3 回	変数，条件分岐	1	2	2
第 4 回	繰返し，ブロックの入れ子	1	4	2
第 5 回	アニメーション作品の制作	1	2	2
第 6 回	アニメーション作品の制作	0	2	0
中間課題	自由作品	-	1	-
第 7 回	コンソール入出力プログラム（割勘計算など）	0	3	0
第 8 回	コンソール入出力プログラム（BMI 計算など）	0	5	1
第 9 回	メソッド（引数）	0	3	1
第 10 回	メソッド（戻り値）	0	4	2
第 11 回	再帰メソッド	0	3	1
第 12 回	集合データ構造	0	2	0
第 13 回	並替えアルゴリズム	0	2	0
第 14 回	辞書アルゴリズム	0	0	0
第 15 回	最終作品の発表会	-	-	-

される講義時間においては，アルゴリズムはブロック言語を中心に説明され，Java は適宜紹介するにとどめられた．

学習者は，毎週課される課題について，問題ごとに 3 通りの解き方が指示された．それらは，

- B（Block 問題）：ブロック言語（BlockEditor）のみを使って解かなければならない問題
- J（Java 問題）：Java のみを使って解かなければならない問題
- A（ANY 問題）：どちらの言語で解いてもよく，途中で変更してもよい問題

の 3 種である．各回に課された問題数と種類をまとめて前掲の表 2 に示した．なお，課される課題はすべて，基本的にはスクラッチから解く問題である（もちろん似た例題はテキストに存在する）．カリキュラム初期は制御構造の概念理解を強調し，BlockEditor を使う機会を強制的に作るために Block 問題を数問用意した．Java への移行を促進するために Java 問題は当初から継続して用意した．後半はアルゴリズムの難易度が高くなるため，Java での習得を必須ではなくし，ANY 問題のみを用意した．

4.3 使用するデータと採取方法

エディタに付属の操作記録保存機能を用いてプログラミングの過程の記録を行い，2 つの言語の使用時間を計算した．このとき計算機上での操作が 5 分以上記録されない場合には作業時間にカウントしない．このように取得された以下の指標について，

Tb ブロック言語（BlockEditor）使用時間

Tj Java 使用時間

BlockEditor 利用率（Rb）を以下の式により求めた．

$$Rb = \frac{Tb}{Tb + Tj} \tag{1}$$

Rb を課題と受講者の組合せごとに求め，それらを単純平均することにより，受講者ごとの Rb と課題ごとの Rb を求めた．

使用したデータは，各回の課題のすべての ANY 問題，および中間課題についての 108 名分のデータ（未提出分を除く）である．最終課題はグループ課題のため対象としていない．

4.4 質問紙調査

初回にプログラミングについての意識調査を実施し，96 件の有効回答を得た．分析に利用したのは，以下の設問である．

- プログラミングに対する意識（不安の有無）

第 14 回の授業終了後に BlockEditor に関する質問紙調査を実施し，88 件の有効回答を得た．分析に利用したのは，以下の 7 つの設問である．

- プログラミングに対する意識（得意・苦手）
- カリキュラムの 4 半期（約 4 週間）ごとの BlockEditor 選択状況
- どのように BlockEditor を利用したか
- BlockEditor を使うことによる Java 習得への影響について
- BlockEditor の使用性
- BlockEditor の良い点
- 自由記述欄

5. 結果

5.1 BlockEditor 利用率

各課題について計算された平均 BlockEditor 利用率について，課題の登場順に並べ替え，その推移グラフを作成した．その結果を**図 4** に示す．青色の折れ線グラフが BlockEditor 利用率である．各課題の規模を考慮して考察できるように，平均の行数，および作業時間をそれぞれ黄色，赤の棒グラフで付記してある．測定単位は，平均の行数については行，作業時間については分である．

BlockEditor 利用率の平均値は，序盤 0.6 付近から始まり，中盤で 0.4 付近を推移し，最終的には 0.1 程度まで下落している．途中，Q6-1，Q6-2 で大幅な下落が見られるが，配列が未習の段階でアニメーションを作るため，たくさんの if 文コピーを作る必要のある回であり，ソース行数が長く入れ子の深度が深くなるのが原因と考えられる．第 6 回で落ち込んだ利用率は，7，8 回で学習内容がコンソールアプリケーションとなり，実数型やキャストの内容が含まれると，再上昇している．中間課題（QMiddle）のソースコード量は突出しているが，BlockEditor 利用率はそれ以前と同様の水準を保っている．中間課題以降の，メソッドの学習（Q9-1 以降）では大幅な利用率の下落が見られる．それ以前の傾きと比較して急激のため，ブロック言語から Java への自然移行だけではなく，内容の変化もその 1 つの要因であると考えられる．

カリキュラム全般を通して 1 課題あたりのソースコード量は一定に保たれており，平均作業時間は授業進行にともなって徐々に増加する傾向が見られ，これは難易度の増加を示していると考えられる．しかしながら，BlockEditor 利用率には関係が見られず，時間経過による自然移行，および内容による利用率の変化の要因が大きいように思われる．

各学生の移行の状況，および個人差の分析を目的として，学生ごとの BlockEditor 利用状況について図示を試みた．その結果を**図 5** に示す．図 5 においては，x 軸の要素が各課題を表現し，y 軸は学生を表現している．各格子要素の濃淡はその区画（課題 × 学生）の BlockEditor 利用率を表現する[*4]．濃色が利用率が高く，淡色は利用率が低いこ

[*4] 未提出も淡色で示している．

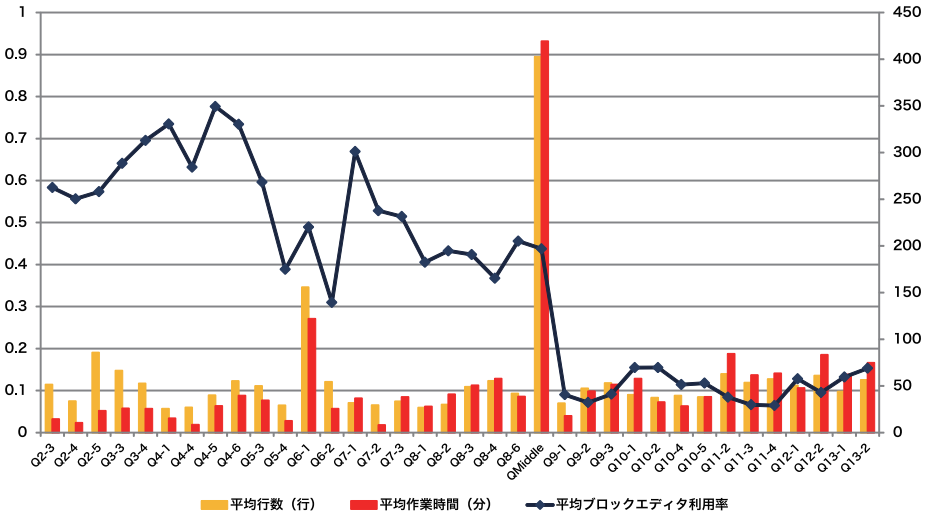


図 4 各課題の BlockEditor 利用率，作業時間，および行数（いずれも平均値）の遷移
Fig. 4 Chart for the rate of working with BlockEditor, total work time, and lines of code for each assignment.

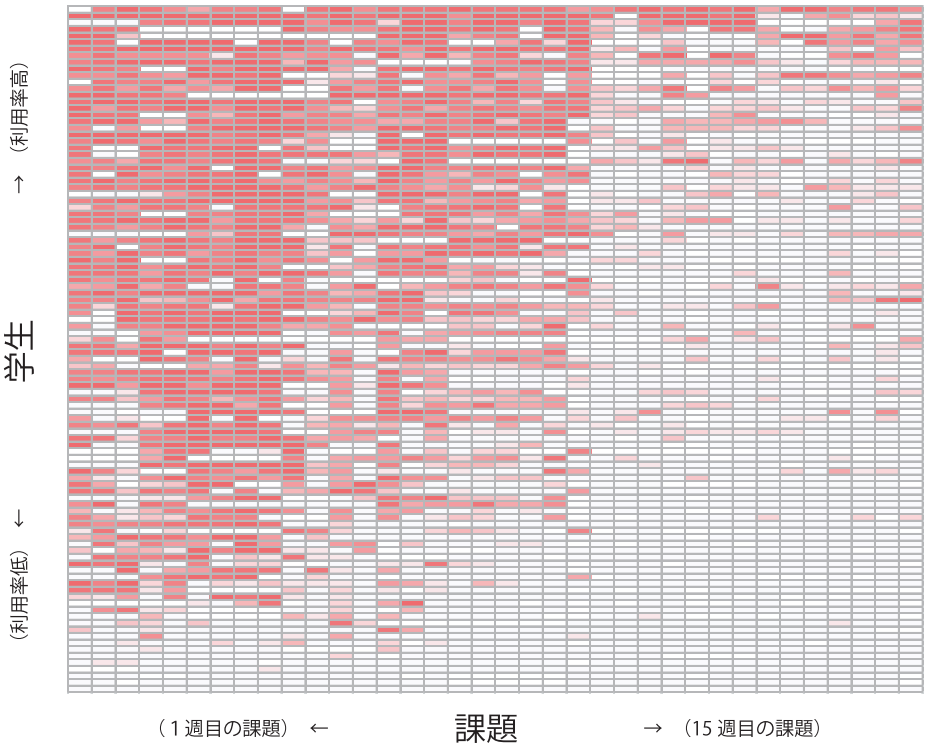


図 5 学生ごとの BlockEditor 利用状況と推移
Fig. 5 Tiles representation of rate of working with BlockEditor for each student.

とを表現する．つまり，横 1 列の帯を左から右へ順に追っていくと学生 1 人の利用率の推移が分かるようになっている．y 軸はカリキュラムを通して利用率が高い学生順に整列してある．

利用率が低い 10%程度の学生は，最初から BlockEditor をまったく利用していない．逆に最後まで BlockEditor を主として利用している学生が 10%程度いる．残りの 80%は，全体的に緩やかに濃から淡へ模様が遷移している．平均の推移グラフと同様に中間課題からメソッドの学習についての急激な利用率の低下が観察されるが，その後もまったく利用されていないわけではなく，部分的に利用され，その数が徐々に低下している．個人単位で観察しても利用率は徐々に低下し，クラス単位での BlockEditor 利用の人数比も徐々に低下し，その時期の個人差が大きいことも観察できる．

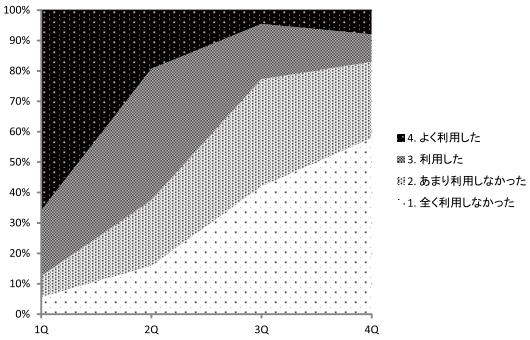


図 6 BlockEditor 利用状況
Fig. 6 Rate of working with BlockEditor by questionnaire.

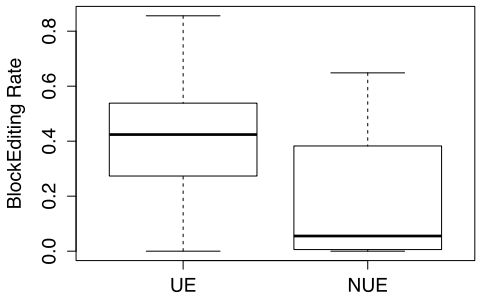


図 7 プログラミングの意識と BlockEditor 利用率（事前調査）
Fig. 7 Difference of the rate of working with BlockEditor by self evaluation of programming skill levels.

5.2 質問紙調査結果

5.2.1 BlockEditor の利用状況

BlockEditor の利用状況について，カリキュラムの 4 半期（約 4 週間，図中では 1Q，2Q と表記）ごとにその利用程度を質問した．その結果を**図 6** に示す．徐々に利用率が低下していく傾向を示しており，実際の作業時間から計測した図 5 のデータと一致している．作業時間データの信頼性を支持する結果である．

5.3 プログラミングの意識と BlockEditor 利用率

授業事前に実施したプログラミングの意識調査において，プログラミングの授業に不安があるかどうかを 6 段階で尋ね，6 段階の回答（まったく不安はない，不安はない，どちらかといえば不安はない，どちらかといえば不安がある，不安がある，とても不安である）が得られた．これを NUE（不安がないグループ）と UE（不安のあるグループ）に分類し，BlockEditor 利用率の比較を行った．その結果を**図 7** に示す．

基本統計量は NUE（n = 10，sd = 0.24，avg = 0.19），UE（n = 84，sd = 0.19，avg = 0.41），であった．分散の大きさが等質と見なせなかったので，ウェルチの法による t 検定を行った．その結果，NUE と UE の平均の差は有意であった（両側検定：$t(10.41) = 2.75$，$p < .05$），したがって，事前に不安を持っている学生が BlockEditor を使用する傾向にあることがいえる．

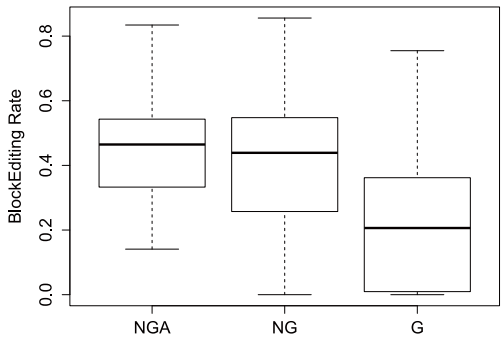


図 8 プログラミングの意識と BlockEditor 利用率（事後調査）
Fig. 8 Difference of the rate of working with BlockEditor by self evaluation of programming skill levels.

授業事後に実施した質問紙調査においてプログラミングの意識を 4 段階で尋ね，G（どちらかといえば得意），NG（どちらかといえば苦手），NGA（まったく苦手）の 3 段階の回答が得られた[*5]．この 3 群について，BlockEditor 利用率の比較を行った．その結果を**図 8** に示す．

基本統計量は G（n = 11，sd = 0.25，avg = 0.25），NG（n = 47，sd = 0.20，avg = 0.41），NGA（n = 28，sd = 0.15，avg = 0.44）であった．G と NG，G と NGA それぞれの組合せに対して，ウェルチの法による t 検定を行った．その結果，G と NG の平均の差は有意傾向であった（両側検定：$t(13.29) = 2.04$，$.05 < p < .10$）．また G と NGA の平均の差は有意であった（両側検定：$t(13.15) = 2.39$，$p < .05$）．したがって，プログラミングに苦手意識がある学生ほど，BlockEditor を選択する傾向が強く，使用時間にしておおむね 2 倍ほどの有意差があるといえる．

以上のように，事前の不安意識や，事後の苦手意識のある学生が BlockEditor をより選択するという仮説は支持された．全体的に苦手意識を持つ学生が多く，その数を減らすことはできなかったが，授業実施者（2 名）の観察では「難しいけれども面白い」と学生に授業が評価されていると考えているため，この分布は妥当な学生の反応と考えている．事前の不安意識と，事後の得意・不得意の意識の相関は r = .44，$p < .01$（スピアマン順位相関係数）であり，弱い相関があった．BlockEditor は任意に利用できるので，BlockEditor の導入が学習者の苦手意識を増加させるとは考えにくい．したがって，苦手意識を持つ学生がプログラミングをよりしやすい環境を求めて BlockEditor を選択し，一定の効果が得られたものと我々は評価している．

5.3.1 BlockEditor の使用性について

BlockEditor の使用性について，6 つの要素ごとに質問した．その結果を**表 3** に示す．「利用していない」と回答した学生は回答数から除外したため，各項目によって回答数が異なっている．百分率は，回答数を分母として計算している．

[*5] 「得意」という選択肢を選んだ学生はいなかった．

表 3 BlockEditor の使用性
Table 3 Results of usability evaluation.

	回答数	とても使いにくい	使いにくい	使いやすい	とても使いやすい
制御構造	80	1 (1%)	6 (8%)	61 (76%)	12 (15%)
構造化ブロック	80	0 (0%)	9 (11%)	57 (71%)	14 (18%)
変数	81	2 (2%)	23 (28%)	44 (54%)	12 (15%)
オブジェクト	73	5 (7%)	21 (29%)	42 (58%)	5 (7%)
メソッド	53	6 (11%)	24 (45%)	22 (42%)	1 (2%)
集合データ構造	47	10 (21%)	18 (38%)	17 (36%)	2 (4%)

表 4 どのように BlockEditor を利用したか
Table 4 Usage of the BlockEditor.

	しなかった	そうしたことがある	よくそうした
BlockEditor だけを利用して，課題を解いた	17 (19%)	64 (73%)	7 (8%)
基本的には Java で書いて，Java で書きにくい所をブロックで書いた	19 (22%)	54 (61%)	15 (17%)
基本的にはブロックで書いて，ブロックで書きにくい所を Java で書いた	27 (31%)	50 (57%)	11 (13%)

表 5 BlockEditor を使うことによる Java 習得への影響
Table 5 Effect of the BlockEditor experiences for developing Java skills.

	まったく同意できない	同意できない	どちらともいえない	同意できる	非常に同意できる
BlockEditor を使うことで Java 構文の理解が深まると思う	1 (1%)	15 (17%)	23 (26%)	43 (49%)	6 (7%)
BlockEditor を使うことで Java 構文の理解ができなくなってしまうと思う	9 (10%)	20 (23%)	42 (48%)	15 (17%)	2 (2%)

制御構造，構造化ブロックについては 8 割以上が「使いやすい」と評価は高く，次に変数，オブジェクトが 6 割程度である．メソッド，集合データ構造に関しては，4 割程度に落ち込んでいる．この理由としては，概念の難易度，ブロック形状，利用者数の問題が考えられるが，学習者の利用している姿の観察結果も考慮して，大きな問題はなかったと我々は評価している．

5.3.2 どのように BlockEditor を利用したか

BlockEditor の利用方法についての質問結果を**表 4** に示す．「BlockEditor だけを利用して，課題を解いた」，「基本的には Java で書いて，Java で書きにくいところをブロックで書いた」，「基本的にはブロックで書いて，ブロックで書きにくいところを Java で書いた」について利用者の多くが経験しているとの結果であり，交ぜ書きも含めて仮説のとおりに利用されていることが確認できた．

5.3.3 BlockEditor を使うことによる Java 習得への影響について

BlockEditor を使うことによる Java 習得への影響についての質問結果を**表 5** に示す．「BlockEditor を使うことで Java 構文の理解が深まると思う」は半数程度の支持率であった．「BlockEditor を使うことで Java 構文の理解ができなくなってしまうと思う」に関しては「どちらともいえない」が半数であり，5 割の学生が「BlockEditor は便利だが，Java ができるようになるには頼りすぎてはいけない」と考えていると分析できる．否定的な意見が 2 割ほどいるが，これまでの経験から否定的な意見の学生が一定数いることは自然で，本システムでは学習方法の嗜好の多様性に対応できる環境であると評価している．

表 6 BlockEditor の良い点
Table 6 Advantages of the BlockEditor.

制御構造が分かりやすい	62 (72%)
コンパイルエラーが出ない	21 (24%)
表記が日本語である	60 (70%)
ブロックがたためる	32 (37%)

5.3.4 BlockEditor の良い点

BlockEditor の良い点と考えられる項目について，複数回答可で質問実施した．結果を**表 6** に示す．我々は，ビジュアル方式の一番の利点は「コンパイル（文法）エラーが出ない」であると考えていたが，「制御構造の分かりやすさ」や「日本語である」ことの方が初学者には重要であることが分かった．「ブロックがたためる」ことも一定の支持を得ており，BlockEditor の 1 つの特徴である抽象化ブロックも利便性が評価された．

5.3.5 自由記述欄のコメント

定量データを補強する目的で自由記述欄のコメントを分

表 7 自由記述欄のコメント
Table 7 Students' comments.

受講者	コメント
A	BlockEditor，Java に慣れるまでは使いやすく慣れるとどんどん使いにくくなっていったのでプログラミング初心者にとっていいエディタだと感じました．
B	私はプログラミングをまったく経験したことがなかったのでプログラムの構造や組み方を理解するのに BlockEditor が使いやすかったです．最初の頃は BlockEditor の方が早くプログラムを書くことができましたが，メソッドあたりから直接 Java で書いた方がやりやすくなった感じがします．
C	Java を理解する上で分からないときに BlockEditor を使用しました．動作が重くなってしまったり，Java で見たときに自分が書いたものが少し変わってしまったりしていたので途中からは BlockEditor の使用をやめました．ですが，詰まったときに見直すために今でもたまに覗いたりしています．
D	いきなり Java だとかまえてしまうが，ブロックだと日本語で分かりやすいので，プログラミングに対する恐怖や不安や拒否反応を抑えることができた．
E	アルゴリズムをより理解しやすくなるのではないかと思う．
F	BlockEditor を使うことで Java のプログラミングの組み立て方は理解できるとは思いますが，どの動作をするときにどういう記述をするかなどは少し覚えにくくなってしまうのではないかと感じました．
G	BlockEditor を使うことによって Java の構文はとてもよく理解できました．しかし，BlockEditor ばかり使っていると，Java でプログラムを書くときに簡単なところでエラーをしてしまうことが多かったです．
H	Java でコンパイルに失敗したときに BlockEditor 見て直したいけれど，コンパイルに成功していないと直すことができないので残念だなと思いました．

析し，BlockEditor の効果を質的に表現しているデータを 8 件抽出した．抜粋したものを**表 7** に示す．

受講者 A，B のコメントは，ブロック言語から Java に移行する理由について，プログラミングの理解進行と関係が深いことを示している典型例と考えられる．プログラミングに慣れ，構造を理解した後は Java でも苦労がなくなり，書きやすく感じていくことが推察される．受講者 C のコメントは，ブロック言語から Java への移行が，急に行われるのではなく，交ぜ書きの状態を通って緩やかに進行していくモデルの妥当性を補強している．受講者 C は動作が重くなることや出力された Java コードの問題についても指摘している．特に動作が重くなることの問題は多くの学習者が指摘しているため，ソフトウェア自体の改良も今後必要である．受講者 D のコメントからは不安意識のある学生に有効であるという結果を支持するものであり，受講者 E のコメントは本提案の目的であるアルゴリズム構築に集中できることの利点が表現されているものと解釈できる．受講者 F のコメントの後半は Java を覚えることに関する問題を指摘しているが，それは言語教育の問題であり，前半で Java プログラミングの組み立て方の理解に有効であったと述べていることから，授業の目的に沿った効果が得られていると解釈できる．受講者 G，H のコメントからは，Java への移行時には文法エラーへの対処が問題であり，BlockEditor 改良による文法エラー改善支援の必要性を示している．

6. 考察

本章では，提案した BlockEditor が，当初の目標であったブロック型言語から Java への移行支援として有効であるか，について，本実験で明らかになった範囲を考察する．

5 章で示したすべてのデータは，学生の選択はブロック言語から Java へ，緩やかに移行していく結果を示している．BlockEditor の利用率の低下模様は，BlockEditor が Java 習得の足場かけ（Scaffolding）として機能していることの 1 つの証拠となろう．この結論を得るためには，学習者が自身の現時点での学習に最適な環境を選択できるメタ認知能力が備わっていることが 1 つの前提となるが，学習者に Block 問題，Java 問題といった形で双方の経験をさせていること，および，対象が大学生であることから一定の選択能力は担保されていると考えた．教師が方法を教示したわけではないのにもかかわらず，多くの学生が「交ぜ書き」のために BlockEditor を利用していることも，学生自身が理解していない個所を意識している証拠であり，この結論を強く支持していると考えた．

BlockEditor 利用率は対象プログラムの長さや作業時間との関連は小さく，経験量や個人差が大きいように思われる．この結論を揺るがす脅威としては，中盤以降メソッドの内容に進行した際の BlockEditor 利用率の下落が速かったことがある．BlockEditor の使用性を問う質問で，前半の内容と比較して相対的にメソッドや集合データ構造のスコアが低かったことから，BlockEditor の使用性も 1 つの要因と考えられる．しかしながら，他の質問紙調査結果は，学生たちが中盤以降特別に BlockEditor が使いにくくなったというわけではなく，理解進行にともなって，あるいは，最終的な Java 習得の目標に向かって自律的に Java を選択するようになっていったことを支持している．メソッド以降のブロック言語仕様の改善によって，さらに緩やかな移行カーブが得られる可能性があるが，大まかには経験量によって移行するという流れは一定の普遍性を持つ，と結論できるデータが示されたと我々は考えている．

学生ごとの BlockEditor 利用状況について分析（図 5）では，移行のタイミングの個人差が大きいことが分かった．まったく BlockEditor を必要としない学生から，最後まで頼る学生まで半年以上の差がある．BlockEditor の Java 習得への効果を問う質問の回答は賛否両論という結果で

あった．中西 [26] も PEN 環境においてとテキスト形式とフローチャートの双方で教育を実施し，賛否両論であるという報告をしている．しかしながら，それは問うタイミングの問題が 1 つの大きな要因である可能性が高く，経験量や修業レベルにより結果が変化することが予想される．プログラミングの自己評価が低い学生の BlockEditor 利用率が高く，自己評価が高い学生の利用率が低いという結果は我々の仮説のとおりであり，個人差が大きいといわれるプログラミング教育に対して，能力に見合った環境を用意できることを BlockEditor の利点として主張する．テキスト記述形式，ビジュアルプログラミング形式の「どちらか一方が教育現場で利用しやすい」ことはなく，学習者が選択できる環境がプログラミング教育の裾野を広げ，教育環境を豊かにすることが期待できる．

加えて，本論文で提案する環境は多くの教育現場で運用可能である．本実験で教員はブロック言語を中心に講義を進め，Java に触れることは控えた．したがって，講義にかかるコストも現場で運用可能レベルと考えることができる．Squeak や Scratch などのビジュアルプログラミングだけではプログラミングの教育にならないと主張するような教員の教育目標や嗜好にも対応できる．ただし，本データは学生の意識（自己評価）と利用率の関連を述べているものであり，実際の能力との関連ではない．BlockEditor は学習者が学習の方法を選択できることを主眼として設計されており，能力が低い学生に対して教師が BlockEditor を推薦するという使い方は考えていない．

本研究の限界は，BlockEditor の利用率の分析にとどまっていることである．質問紙調査でその理由がプログラミングの慣れや本授業の主目標であるアルゴリズム構築の理解と関連性があることは示唆されるものの，BlockEditor が理解度の向上に寄与した直接的なデータは得られていない．特に，これまでのテキスト型言語ではまったく理解できず，プログラミングが嫌になってしまう学生が何人救えたか，というのが我々の関心あるテーマである．本論文で述べた授業の担当者 2 名は，前年度に BlockEditor の有無以外はほぼ同様の環境（Java のみ）の授業を行った経験を持つ．主観データではあるが，年度比較して本年度は BlockEditor が利用できたために，学生がアルゴリズム構築に集中している多くの姿を観察することができたと評価している．課題の提出率や中間課題，最終課題の評価点は同等かそれ以上であったことも確認している．このようなデータに対して客観的な裏付けがなされるために，今後も言語間比較などの精緻な初学者のプログラミングプロセス分析研究が必要である．その際，本研究で提案した環境を利用することによって，教育研究者が教育効果を確認しながら研究データの取得が可能になることが期待される．

7. まとめ

本論文では，MIT で開発された OpenBlocks システムを利用して開発した「BlockEditor」の提案を行った．ビジュアルプログラミング言語からテキスト型言語へのシームレスな移行の支援を目的としてブロック型言語と Java の相互変換機能の提案を行った．文科系大学生向けプログラミング入門教育全編での使用実験を行った．採取したシステム記録からクラス全体の BlockEditor 利用率が初期 0.6 から中盤 0.4，後期に 0.1 とプログラミングの学習が進行するにつれて低下していくこと，および個人ごとの観察においても徐々に低下していくこと，およびそのタイミングに個人差があることが定量的に示された．プログラミングに苦手意識を持つ学生ほどブロック言語の選択率が 2 倍ほど高いこと，言語の相互変換環境が言語の交ぜ書きを促進することも示された．以上のデータを考察し，BlockEditor が Java によるプログラム構築能力習得の足場かけとして有用に機能し，多くの教育現場で活用可能であることを主張した．

参考文献

[1] Ingalls, D., Kaehler, T., Maloney, J., Wallace, S. and Kay, A.: Back to the Future: The Story of Squeak, A Practical Smalltalk Writtern in Itself available from ⟨http://www.squeak.org⟩, *Proc. ACM OOPSLA'97*, p.318 (1997).

[2] Scratch Team Lifelong Kindergarten Group MIT Media Lab: Scratch – imagine, program, share – available from ⟨http://scratch.mit.edu/⟩.

[3] Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B. and Resnick, M.: Scratch: A Sneak Preview, *Proc. 2nd International Conference on Creating Connecting and Collaborating through Computing 2004*, pp.104–109 (2004).

[4] Kay, A.: Squeak Etoys Authoring & Media, *Viewpoints Research Institute Research Note* (online) (2005), available from ⟨http://www.squeakland.org/resources/articles/article.jsp?id=1008⟩.

[5] UNESCO: ICT Curriculum for School/Program of Teacher Development (2002), available from ⟨http://unesdoc.unesco.org/images/0012/001295/129538e.pdf⟩.

[6] 杉浦 学，松澤芳昭，岡田 健，大岩 元：アルゴリズム構築能力育成の導入教育：実作業による概念理解に基づくアルゴリズム構築体験とその効果，情報処理学会論文誌，Vol.49, No.10, pp.3409–3427 (2008).

[7] 兼宗 進，中谷多哉子，御手洗理英，福井眞吾，久野 靖：初中等教育におけるオブジェクト指向プログラミングの実践と評価，情報処理学会論文誌：プログラミング，Vol.44, No.SIG 13 (PRO 18), pp.58–71 (2003).

[8] 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グ学習環境 PEN の実装と評価，情報処理学会論文誌，Vol.48, No.8, pp.2736–2747 (2007).

[9] Roque, R.V.: OpenBlocks: An Extendable Framework for Graphical Block Programming Systems, Master thesis at MIT (2007).

[10] Wood, D., Bruner, J.S. and Ross, G.: The Role of Tu-

toring in Problem Solving, *Journal of Child Psychology and Psychiatry*, Vol.17, No.2, pp.89–100 (1976).
[11] 森 秀樹，杉澤 学，張 海，前迫孝憲：Scratch を用いた小学校プログラミング授業の実践：小学生を対象としたプログラミング教育の再考，日本教育工学会論文誌，Vol.34, No.4, pp.387–394 (2011).
[12] 伊藤一成：Scratch を用いた授業実践報告，情報処理，Vol.52, No.1, pp.111–113 (2011).
[13] Cooper, S., Dann, W. and Pausch, R.: Teaching Objects-first in Introductory Computer Science, *Proc. 34th SIGCSE Technical Symposium on Computer Science Education, SIGCSE '03*, pp.191–195, ACM (2003).
[14] Cheung, J.C., Ngai, G., Chan, S.C. and Lau, W.W.: *Filling the Gap in Programming Instruction: A Text-enhanced Graphical Programming Environment for Junior High Students, SIGCSE '09, Proc. 40th ACM Technical Symposium on Computer Science Education*, New York, NY, USA (2009).
[15] Google Inc: Blockly: A Visual Programming Editor, available from ⟨http://code.google.com/p/blockly/⟩ (accessed 2013-03-17).
[16] 兼宗 進，御手洗理英，中谷多哉子，福井眞吾，久野 靖：学校教育用オブジェクト指向言語「ドリトル」の設計と実装，情報処理学会論文誌：プログラミング，Vol.42, No.SIG11 (PRO12), pp.78–90 (2001).
[17] 長 慎也，甲斐宗徳，川合 晶，日野孝昭，前島真一，筧捷彦：プログラミング環境 Nigari—初学者が Java を習うまでの案内役，情報処理学会論文誌：プログラミング，Vol.45, No.SIG09, pp.25–46 (2004).
[18] Pasternak, E.: Visual Programming Pedagogies and Integrating Current Visual Programming Language Features, Master's Thesis, Carnegie Mellon University Robotics Institute Master's Degree (2009).
[19] Warth, A., Yamamiya, T., Ohshima, Y. and Scott, W.: Toward a More Scalable End-user scripting Language, *Proc. 2nd International Conference on Creating Connecting and Collaborating through Computing 2008*, pp.172–178 (2008).
[20] 岡田 健，杉浦 学，松澤芳昭，大岩 元：日本語プログラミングを用いた論理思考とプログラミングの教育，情報処理学会研究報告．コンピュータと教育研究会報，pp.123–128 (2000).
[21] Harvey, B. and Monig, J.: Bringing "No Ceiling" to Scratch: Can One Language Serve Kids and Computer Scientists?, *Constructionism 2010*, Paris (2010).
[22] Abelson, H., Sussman, G.J. and Sussman, J.: *Structure and Interpretation of Computer Programs – 2nd ed.*, MIT Press (1996). (和田英一（訳）：計算機プログラムの構造と解釈).
[23] 石田真樹，桑田正行：C プログラミングの学習支援に関する研究–PAD エディタを用いたアルゴリズム学習支援システムの構築，コンピュータと教育，pp.41–48 (2000).
[24] 永原工策，桑田正行：PAD エディタを用いた C プログラミング学習支援システム構築に関する研究，情報処理学会研究報告，pp.17–24 (2004).
[25] Papert, S.: *Mindstorms: children, computers, and powerful ideas*, Basic Books, Inc., New York, NY, USA (1980).
[26] 中西 渉：PenFlowchart の開発，情報処理学会研究報告 CE113 (13)，pp.1–6 (2012).

付 録

A.1 BlockEditor プログラムサンプル

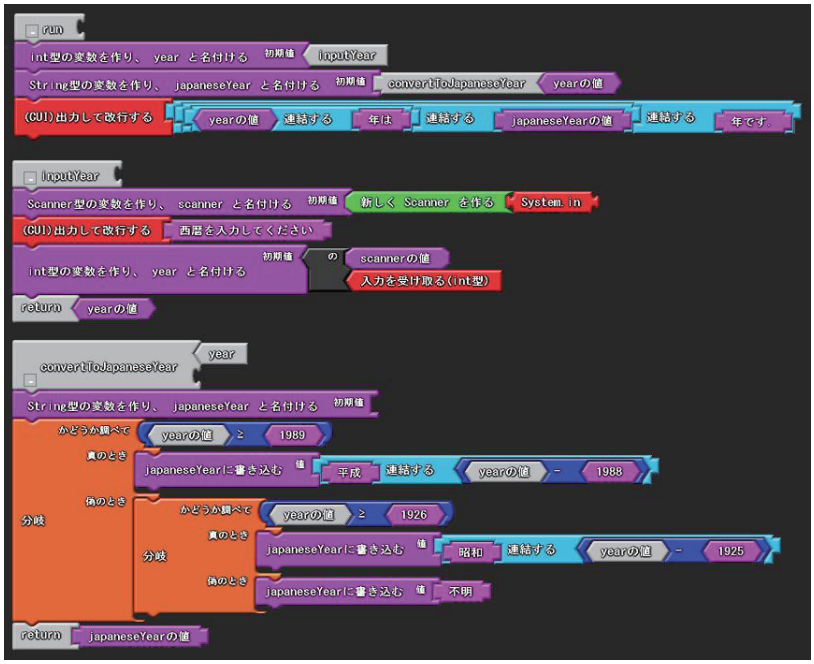


図 A·1 西暦和暦変換のプログラムサンプル（Block）
Fig. A·1 Sample program converting European to Japanese year (Block).

```
1  import java.util.Scanner;
2
3  //西暦和暦変換プログラム
4  public class ConvertYearApplication {
5    public static void main(String[] args) {
6      ConvertYearApplication main = new ConvertYearApplication();    main.run();
7    }
8
9    // 西暦を和暦に変換する
10   void run() {
11     int year = inputYear();
12     String japaneseYear = convertToJapaneseYear(year);
13     System.out.println(year + "年は" + japaneseYear + "年です.");
14   }
15
16   // 西暦を入力する
17   int inputYear() {
18     Scanner scanner = new Scanner(System.in);
19     System.out.println("西暦を入力してください");
20     int year = scanner.nextInt();
21     return year;
22   }
23
24   // 西暦を和暦に変換する
25   String convertToJapaneseYear(int year) {
26     String japaneseYear;
27     if (year >= 1989) {      japaneseYear = "平成" + (year - 1988);    }
28     else if (year >= 1926) {      japaneseYear = "昭和" + (year - 1925);    }
29     else {      japaneseYear = "不明";    }
30     return japaneseYear;
31   }
32 }
```

図 A·2 西暦和暦変換のプログラムサンプル（Java）

Fig. A·2 Sample program converting European to Japanese year (Java).

図 A·3 集合（リスト）のプログラムサンプル（Block）

Fig. A·3 Sample program using collection (Block).

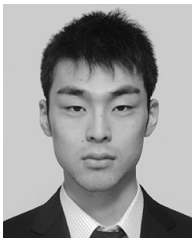
```
1  //集合（リスト）のサンプルプログラム
2  public class ListSample extends Turtle {
3    public static void main(String[] args) {
4      Turtle.startTurtle(new ListSample());
5    }
6
7    //10 枚のカードをリストに追加する
8    public void start() {
9      ListTurtle<CardTurtle> cards = new ListTurtle<CardTurtle>();
10     {//カードを作り，追加する
11       int i = 0;
12       while(i < 10){
13         cards.addLast(new CardTurtle(i * 10));
14         i++;
15       }
16     }
17   }
18 }
```

図 A·4 集合（リスト）のプログラムサンプル（Java）

Fig. A·4 Sample program using collection (Java).

松澤 芳昭 （正会員）

2000 年慶應義塾大学環境情報学部卒業．2002 年慶應義塾大学大学院政策・メディア研究科修士課程修了．2007 年同後期博士課程単位取得退学，2008 年より静岡大学情報学部特任助教．現在，同学部助教．博士（政策・メディア）．オブジェクト指向技術を応用したソフトウエアの設計と開発，情報教育，情報システム技術者育成の研究に従事．日本教育工学会，情報システム学会，教育システム情報学会各会員．



保井 元 （正会員）

2010 年静岡大学情報学部卒業．2012 年静岡大学大学院情報学研究科修士課程修了．現在，スズキ教育ソフト（株）在職中．在学中はプログラミング学習の教育手法および支援環境の研究に従事．

杉浦 学 （正会員）

2003 年慶應義塾大学環境情報学部卒業．2005 年慶應義塾大学大学院政策・メディア研究科修士課程修了．2010 年慶應義塾大学大学院政策・メディア研究科後期博士課程修了．博士（政策・メディア）．2006 年より慶應義塾大学環境情報学部非常勤講師．2013 年より山梨英和大学人間文化学部専任講師，津田塾大学女性研究者支援センター客員研究員．情報教育，教育学習支援情報システムの研究に従事．IEEE Computer Society，日本教育工学会各会員．

酒井 三四郎 （正会員）

1984 年静岡大学大学院電子科学研究科博士後期課程修了．学習院大学，新潟産業大学，静岡大学工学部を経て，1998 年静岡大学情報学部助教授．現在，同学部教授．工学博士．ソフトウェア開発支援環境，プログラミング学習支援環境，遠隔学習，協調学習に関する研究・開発に従事．電子情報通信学会，教育システム情報学会各会員．