Vol.2013-CE-119 No.2
2013/3/15

ビジュアル型言語とテキスト記述型言語の併用による プログラミング入門教育の試みと成果

松澤 芳昭[1]　酒井 三四郎[1]

概要：近年，ビジュアル型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が，C や Java などのテキスト記述型言語へのシームレスな移行が考慮されていないという問題がある．本研究では，筆者らが CE113 で提案したビジュアル記述型言語とテキスト記述型言語の併用開発環境「BlockEditor」について，Java 言語習得を目的とし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．本実験では，15 週の全ての課題解答過程において，学習者がビジュアル言語 (BlockEditor) とテキスト言語 (Java) を任意に選択できる環境が与えられた．採取したシステム記録，および質問紙調査の結果から，プログラミングの学習が進行するにつれて，BlockEditor から Java へ徐々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ビジュアル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 言語習得の足場かけとなることが示された．

A Block Editing System for Seamless Migration to Java in Introductory Programming Education

Matsuzawa Yoshiaki[1]　Sanshiro Sakai[1]

Abstract: In the past decade, improvements to the environment of an introductory programming education by block-based 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 convert block language and Java both ways. We conducted an empirical study of this system in an introductory programming course for one hundred and ten university students. When students were given opportunities to select their 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 age.

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[*1]である．ビジュアルプログラミング言語では，主としてマウスによる操作によってプログラム開発を行う事ができ，テキスト記述型言語で生じる文法エラーが発生しないので，初学者にとって使いやすいプログラミング言語と考えられている．

[1] 静岡大学情報学部
Faculty of Informatics, Shizuoka University

[*1] プログラム要素をテキストで指定するのではなく，グラフィカルに操作することでプログラムを作成する方式のプログラミング言語

ビジュアルプログラミング言語は初学者にとって「プログラミング」を学ぶためには有用なシステムであり，入門段階で成功をおさめている [3][4]．しかしながら，その後のプログラミングの発展に寄与したとされているデータは提示されていない．実際の現場では，学習者の中に将来的にテキスト記述型言語を取得する必要のある者も混在している．将来的にテキスト記述型言語を必要とする学習者にとって，テキスト型言語への発展が見えにくいシステムを利用することは，入門学習への動機付けをそぐ理由となる．ビジュアルプログラミング言語で学習した内容がテキスト記述型言語に転移されないとしたら本末転倒である．

そこで本研究では，ビジュアルプログラミング言語からテキスト記述型言語へのシームレスな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 [5] を利用し開発を行った．本システムの特徴はビジュアルプログラミング言語とテキスト記述型言語の相互変換ができることである．アルゴリズムの基本構造などのプログラミングの基礎知識を学習し理解したあと，テキスト記述型言語によるプログラムの実装へと移行することができる．

本論文は全 6 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

本研究では，Squeak や Scratch のように，システムによって用意されたブロック型の命令を組み立てる事によってプログラムを実装できるビジュアルプログラミング言語のことをブロック型言語と呼ぶ．本章では，ブロック型言語の開発と教育に関する先行研究について述べる．

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][7] や，ブロック型言語の開発 [7][8] が行われている．ブロック型言語はテキスト記述型言語と違い文法エラーが無いので，プログラミング初学者にとって使いやすい言語となっている．一方でドリトル [9] や PEN[10] のように、テキスト記述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，プログラミング初学者にとってブロック型言語と直接比較しているデータは無く，これらの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記述型言語への移行を支援する研究として，Erik の研究 [11] がある．この研究で Erik は JubJub と呼ばれるシステムに，ブロック型言語で実装したプログラムを Java に変換する機能を追加しシステム開発を行っている．JubJub はビジュアルプログラミング言語のもつ目標に焦点を当てて設計されたシステムである．Erik は「テキスト記述型言語への移行を支援する」という目標に注目し，ブロック型言語によるプログラムを実装した後，ブロック型言語で実装したプログラムを Java のプログラムで表現するプログラムの開発している．しかし，このシステムはブロック型言語から Java への移行が一方通行であり，Java からブロック型言語への変換ができない．

Brian 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 [12]．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．Brian らは構造的プログラミングの教科書として著名な SICP(Structure and Interpretation of Computer Programs)[13] を引用し構造的化サポートの重要性を述べている．これは我々の動機とほぼ同様である．しかしながら，関数の作成インタフェイスは洗練されておらず，評価もなされていないことが課題である．

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 [14][15] 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とし，プログラミング言語は C 言語を対象としている．C 言語から PAD に変換するとき，プログラムの文がそのまま PAD のラベルになり，PAD のプログラムに変換される．さらに，石田ら研究 [14] の問題点を分析し，その問題点を解決するとともに，永原ら研究 [15] では学習支援のための新たな機能を追加している．アルゴリズムを作成する時，まず問題を大局的に捉え，徐々に詳細化していく事を支援する機能を提案している．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．C 言語ではコメントとして出力される．しかし，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づらいという問題がある．また，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．

岡田ら [16] は日本語プログラミング環境「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 で同じ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た．しかし，この授業は「ことだま on Squeak」を用いたプログラミング学習から Java によるプログラミング学習へ，シームレスな移行になっていない．

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 はプログラミングの構造化教育支援とブロック型言語からテキスト記述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 ブロック型言語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れる．

機能 2 ブロック型言語と 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：ユーザがブロック型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，Java によるプログラミング教育では，初学者がコンパイルエラーに惑わされ，プログラミングに対して苦手意識を持ってしまうという仮説に基づくものである．これは一般的なビジュアルプログラミング言語の利点であるが，我々がブロック型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の基本構造とプログラムの構造化を学習し，プログラミングに対する苦手意識が徐々に解消され，Java でのプログラムの実装へと移行するという仮説に基づくものである．

機能 3 は，ユーザが抽象化ブロックを利用してプログラムの実装を行うことで，プログラムの構造を考慮しながら実装を行い，その手段を確立するという仮説に基づくものである．プログラムを部分的にまとめ，まとめたプログラムの目的を正確に把握出来るようになることを目的に支援を行う．

3.2 機能 1:ブロック型言語によるプログラム記述機能

3.2.1 インタフェイス

開発した「BlockEditor」の外観を**図 1** に示す．図 1 の左下の画像が BlockEditor を使用している様子である．BlockEditor はエディタの右下の作業ウィンドウでプログラムを作る．プログラムを作るときは，左のウィンドウにある各カテゴリからブロックを取り出し，作業ウィンドウでブロックを組み立てる．Java に変換するときは，エディタの上部にある「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．「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 ブロックの設計

BlockEditor の基礎部分は OpenBlocks フレームワーク [5] を利用して開発を行った．ブロック設計の際に考慮した点は 2 点ある．第一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全て英語であった．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」[16] を参考にしてブロックの日本語化を行った．第二は Java への変換を考慮したことである．特に以下の変数，繰返し文，分岐文の 3 点について考慮している．

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，標準形式に統一して変換するように設計した．Java の繰返し文は for 文と while 文の二種類がある．BlockEditor の繰返し命令のブロックを Java に変換したとき，Java では while 文に変換されるように設計した．Java の分岐文は if 文，if-else 文，switch 文の三種類がある．BlockEditor の分岐命令のブロックを Java に変換したとき，Java では if-else 文に変換されるように設計した．

このように設計した理由として以下の 2 点がある．第一の理由として，本システムの対象者は Java の習得を目指す目的とするユーザを対象としている．そのため，BlockEditor から Java に移行した時，違和感なく Java を利用できるように BlockEditor を設計している．第二の理由として，ブロック型言語の利点としてコンパイルエラーが起きないという事が挙げられる．そのため，主に変数の型に言える事であるが，BlockEditor から Java に変換した時，構文エラーが起きるとブロック型言語の利点が活かされず本末転倒である．よって，Java に変換した時，構文エラーが起きないことを優先し設計した．

3.3 機能 2：ブロック型言語と Java の相互変換機能

3.3.1 ブロック型言語から Java への変換

BlockEditor では，組み立てられたブロック型言語を Java のソースコードに変換する．この変換システムによって，BlockEditor 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

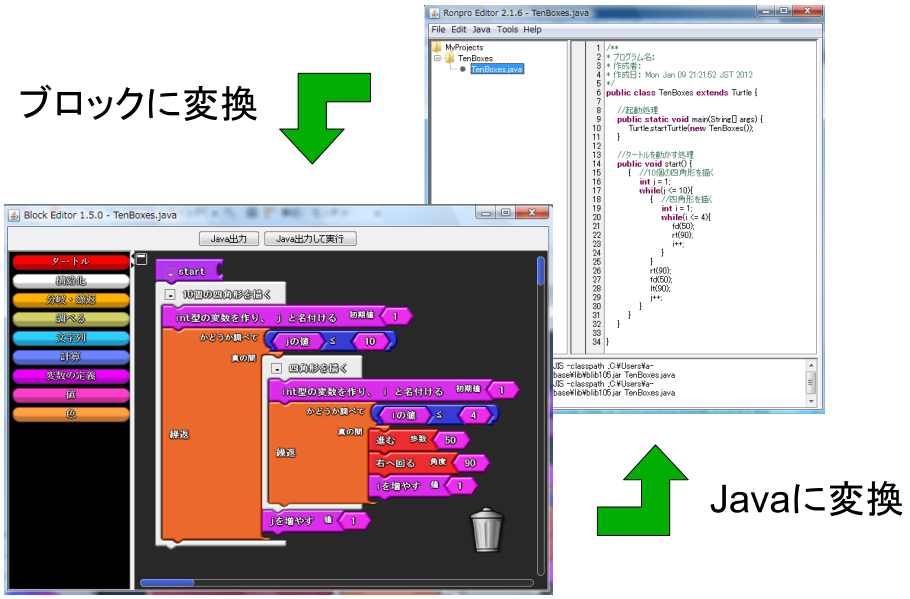

図 1 BlockEditor のインタフェイス
Fig. 1 The interface of BlockEditor.

```
1  public class TenBoxes extends Turtle {
2
3      public void start() {
4          { //10 個の四角形を描く
5              int j = 1;
6              while(j <= 10){
7                  { //四角形を描く
8                      int i = 1;
9                      while(i <= 4){
10                         fd(50);
11                         rt(90);
12                         i++;
13                     }
14                 }
15                 rt(90);
16                 fd(50);
17                 lt(90);
18                 j++;
19             }
20         }
21     }
22
23 }
```

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

式を変換することで，Java のソースコードを出力する．

例として「10 個の四角を描くプログラム」の BlockEditor の変換例を示す．**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 で実装されたプログラムを変換したものである．図 1 の BlockEditor で実装されたプログラム中に使われているブロックで，「繰返」ブロックは while 文に変換される．プログラミング教育カリキュラムの前半は Java で実装されたタートルグラフィックス [17] を採用している．タートルグラフィックスで実行するために，Java において (1)「右へ回る」ブロックは rt，(2)「進む」ブロックは fd と変換される．

3.3.2 Java からブロック型言語への変換

Java のソースコードを BlockEditor のプログラムに変換する．この変換システムによって，Java で構築したプログラムが BlockEditor ではどのようなプログラムに実装されるのかがユーザに提示される．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 からブロック型言語に変換する際，文法エラーが存在した状態では変換が出来ない．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と考えたための設計である．

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，ブロック型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」を開発した．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出来る．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ブロックの中に構築したプログラムの目的を記述することができる

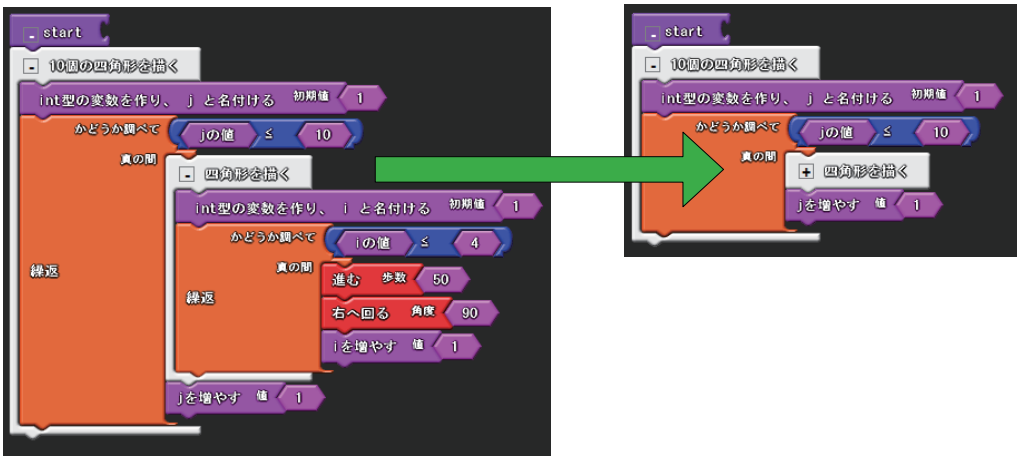

図 3 抽象化ブロックの折りたたみ機能
Fig. 3 The demonstration of collapsing "abstraction block".

3.4.2 折りたたみ機能

抽象化ブロックは，折りたたみ中に構築したブロックを隠すことが出来る．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，抽象化ブロックを折りたたむことが出来る．図 3 は抽象化ブロックの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「四角形を描く」の目的の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他の命令ブロック同様にこの折りたたみ機能によって，プログラム抽象化の基本である，「複数のものを一つのものとしてまとめ，名前をつけて一つのものとして扱う」[13] という機能が扱えるようになる．これはビジュアルプログラミングにおいて，一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．

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 ソースへの展開は，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目的記述として反映される．

4. 実験方法

4.1 実施環境

著者所属学部で 2012 年度後期に行われた，文科系の学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 Squeak によるビジュアルプログラミング体験を 4 週間受講済である．

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

「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．

習得目標の言語は Java 言語であるが，学習者には BlockEditor も併用できる環境が与えられた．教材で提供する例題プログラムにおいては，ブロックバージョンと Java バージョンの両方が用意された．毎週 180 分の授業時間の内，60 分ほど実施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 言語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，問題毎に 3 通りの解き方が指示された．それらは，

- B(Block 問題) : BlockEditor のみを使って解かなけれ

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	-	-	-

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良く，途中で変更しても良い問題

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（勿論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 BlockEditor 専用問題を数問用意した．Java 言語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 言語での習得を必須ではなくし，ANY 問題のみを用意した．

4.2 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 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．

$$Rb = \frac{Tb}{Tb + Tj} \quad (1)$$

Rb を課題と受講者の組み合わせ毎に求め，それらを単純平均することにより，受講者毎の Rb と課題毎の 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．

4.3 質問紙調査

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）
- クォーター毎 BlockEditor の選択状況
- 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 BlockEditor 利用率について，課題の登場順に並替え，推移グラフを作成した．その結果を**図 4**，および**図 5** に示す．両推移グラフにおける BlockEditor 利用率は同様のデータであり，棒グラフについては．図 4 が平均行数，図 5 が平均作業時間を示している．

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，BlockEditor から Java への自然移行だけではなく，内容の変化もその一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伴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

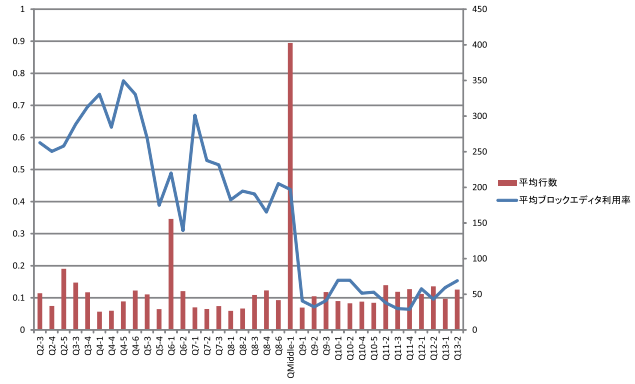

図 4 BlockEditor 利用率と各課題の平均行数の遷移
Fig. 4 Chart for the rate of working with BlockEditor and lines of code for each assignment.

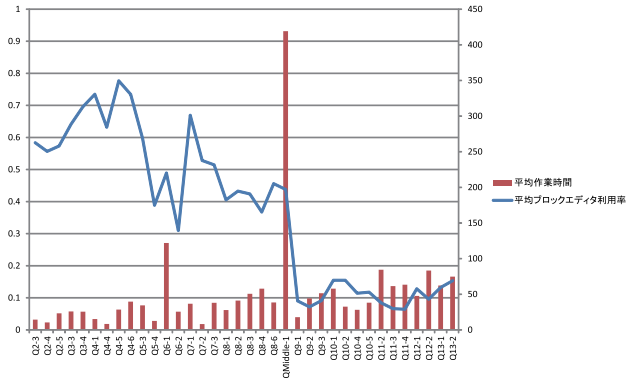

図 5 BlockEditor 利用率と各課題の平均作業時間の遷移
Fig. 5 Chart for the rate of working with BlockEditor and total work time for each assignment.

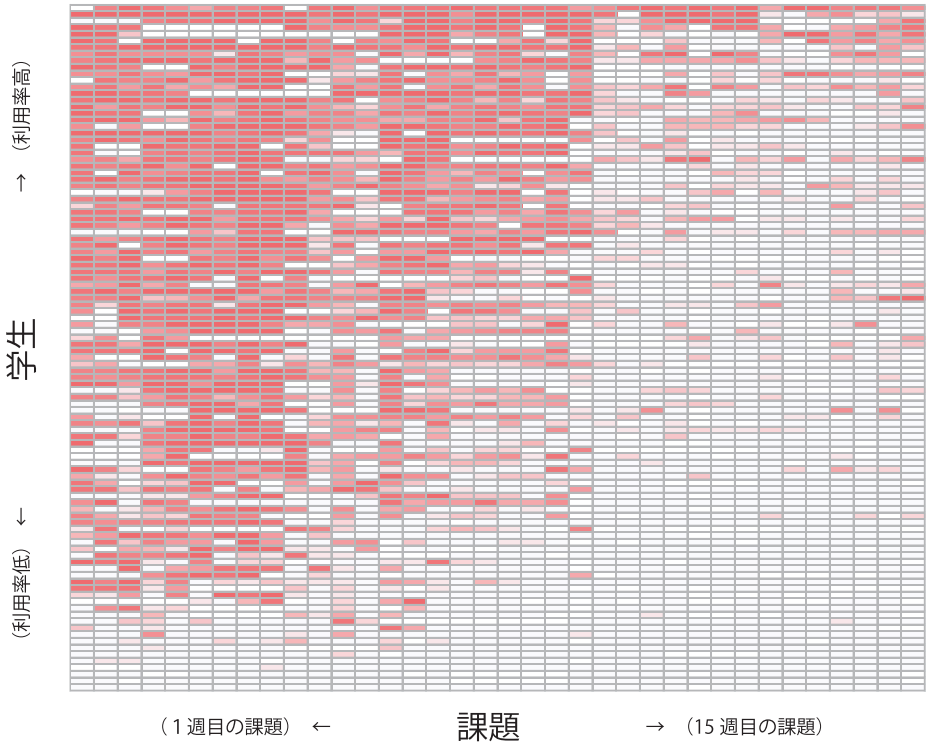

図 6 学生毎の BlockEditor 利用状況と推移
Fig. 6 Tiles representation of rate of working with BlockEditor for each student.

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，学生毎の BlockEditor 利用状況について図示を試みた．その結果を**図 6** に示す．図 6 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[*2]．濃色が利用率が高く，淡色は利用率が低いことを表現する．つまり，横一列の帯を左から右へ順に追っていくと学生一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．

図 6 について分析を行う．利用率が低い 10%程度の学生は，最初から BlockEditor を全く利用していない．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，その後も全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．

[*2] 未提出も淡色で示している．

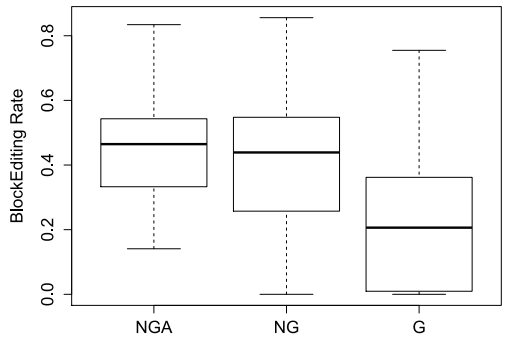

図 8 プログラミングの意識と BlockEditor 利用率
Fig. 8 Difference of the rate of working with BlockEditor by self evaluation of programming skill levels.

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）毎にその利用程度を質問した．その結果を**図 7** 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 6 等のデータ一致している．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

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 (全く苦手) の 3 段階の回答が得られた．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 それぞれの組み合わせにたいして，分散の大きさが等質とみなせなかったので，ウェルチの法による t 検定を行った．その結果，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，使用時間にして概ね 2 倍ほどの有意差があるといえる．

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．

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 割程度といったところである．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．概念の難易度が異なるため，ブロック形状，仕様の問題と結論するのは早計であるが，中盤以降の利用率の低下は使用性の低下も一つの要因として考えられることを示している．

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%)*

5.3.2 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．想定された 3 つの使い方，「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 で書きにくい所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所を Java で書いた」について利用者の多くが経験しいるとの結果であり，想定ど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．逆に「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」と考える学生も 2 割ほどおり，否定的な意見の学生も一定数いることが分かった．

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．筆者らの仮説では，ビジュアル方式の大きなの利点として「コンパイルエラーが出ない」ことを考えていたので，制御構造のわかりやすさを支持するこの結果は多少意外性がある．

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析し，BlockEditor の効果を質的に表現しているデータを 4 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，BlockEditor 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．プログラミングになれ，構造を理解した後は Java でも苦労が無くなり，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，BlockEditor 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 D のコメントからは，Java への移行時にはコンパイルエラーへの対処が問題であり，BlockEditor 改良によるコンパイルエラー改善支援の必要性を示している．

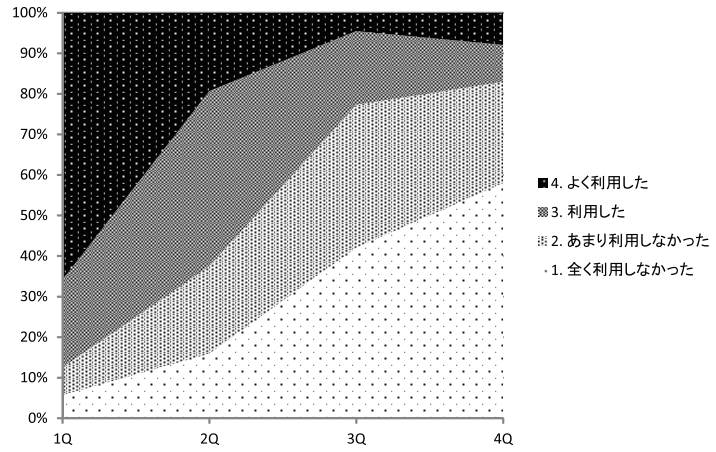

図 7 BlockEditor 利用状況
Fig. 7 Rate of working with BlockEditor by questionnaire.

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.

	全く同意できない	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%)*

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 BlockEditor 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 言語習得の足場かけ (Scaffolding) として機能していることの一つの証拠となろう．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一つの前提となるが，学習者に BlockEditor 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

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全く経験したことが無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事が出来ましたが，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 を理解する上でわからないときに 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	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．

が「混ぜ書き」のために BlockEditor 環境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 の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 の使用性も一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，理解進行に伴って，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．メソッド移行の BlockEditor 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．

学生毎の BlockEditor 利用状況について分析（図 6）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．全く 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 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 [18] 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一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通りであり，個人の能力差が大きいと言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オンデマンドで用意することが出来ることを BlockEditor の利点として主張したい．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」ことは無く，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 BlockEditor 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出来る．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広く対応できるであろう．

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慣れや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全く理解できず，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．今後も言語間比較等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となるだろう．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記述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，および個人毎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ビジュアル型言語の選択率が 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 言語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] Dan Ingalls, Ted Kaehlei, John Maloney, Scott wallace and Alan Kay: Back to the Future: The Story of Squeak, A Practical Smalltalk Writtern in Itself `http://www.squeak.org`, *Proc. of ACM OOPSLA' 97*, p. 318 (1997).

[2] Scratch Team Lifelong Kindergarten Group MIT Media Lab: Scratch -imagine.program.share- `http://scratch.mit.edu/`.

[3] 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).

[4] 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).

[5] Ricarose Vallarta Roque: OpenBlocks: An Extendable Framework for Graphical Block Programming Systems, *Master thesis at MIT* (2007).

[6] 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).

[7] 須曽野仁志，木谷康司，下村 勉：Logo プログラミングを主体とした中学校「情報基礎」の実践，日本教育情報学会 10 回年会，pp. 106–107 (1994).

[8] Joey C.Y. Cheung, Grace Ngai, Stephen C.F. Chan and Winnie W.Y. Lau: *Filling the gap in programming instruction: a text-enhanced graphical programming environment for junior high students*, SIGCSE' 09 Proceedings of the 40th ACM technical symposium on Computer science education, New York, NY, USA (2009).

[9] 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).

[10] 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).

[11] Erik Pasternak: Visual Programming Pedagogies and Integrating Current Visual Programming Language Features, Master's thesis, Carnegie Mellon University Robotics Institute Master's Degree (2009).

[12] Brian harvey and Jens Monig: Bringing "No Ceiling" to Scratch: Can One Language Serve Kids and Computer Scientists?, *Constructionism 2010, Paris* (2010).

[13] Abelson, H., Sussman, G. J., Julie Sussman(和田英一訳)：Structure and Interpretation of Computer Programs – 2nd ed.(邦題：計算機プログラムの構造と解釈), MIT Press (1996).

[14] 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).

[15] 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).

[16] 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).

[17] Papert, S.: *Mindstorms: children, computers, and powerful ideas*, Basic Books, Inc., New York, NY, USA (1980).

[18] 中西 渉：PenFlowchart の開発，情報処理学会研究報告 CE113(13)，pp. 1–6 (2012).