Vol.2014-CE-124 No.9
2014/3/14

OpenBlocks を用いたプログラミング学習用ソフトウェアの開発

主原佑記† 赤井昭仁† 中村亮太† 松浦敏雄†

一般情報教育の一環として、プログラミングを経験することの重要性が認識されるようになり、初等中等教育においてもプログラミング教育が取り入れられつつある。Scratch は初学者向けのプログラミング入門用ソフトウェアとして注目されており各地で利用されている。Scratch は、画面上でブロックを組み合わせることでプログラムを作成でき(ブロック言語)、構文エラーが起こりにくいという特徴を有している。しかし、Scratch では扱えるブロックが 100 種以上もあり、初学者がそのすべてを一度に見せられると混乱を招く恐れがある。

本研究では、Scratch の GUI 部分のみを抽出した OpenBlocks と呼ばれる GUI ライブラリを用いて、初学者に分かりやすいプログラミング学習環境を開発した。Scratch と同様にブロック言語をベースとし、プログラムの実行・デバッグ環境を構築し、学習者の混乱を避けるため、段階的に学べる仕組みを導入した。さらに、ブロックで構成したプログラムを C や Java などの既存の言語に変換する機能も実装し、既存言語の学習の導入としても利用できるようにした。

Development of a Programming Environment for Novices using OpenBlocks

Yuki SHUHARA† Akihito AKAI†
Ryota NAKAMURA† Toshio MATSUURA†

Programming experience has come to be recognized as an important part of general information education. As a result, even in elementary and secondary education, programming has been adopted. Scratch is noted as a user-friendly programming environment for beginners, and it is used in many schools. Scratch has a feature of combining the blocks on the screen to create programs. By using this, syntax errors are unlikely to occur. However, there are 100 or more blocks, so beginners are confused by all the blocks shown at once.

In this study we have developed an easily understandable learning environment for beginners using OpenBlocks which is a GUI library extracted from Scratch. As with Scratch, it implements a block programming language. We also developed an environment for software execution and debugging, and we introduced a mechanism for learning step-by-step to avoid confusion by learners. In addition, programs written with blocks are converted into existing languages such as C, Java, etc. Therefore, learners are able to use this software as they begin to study these coventional programming languages.

1. はじめに

近年、クリエイティブな道具としてのプログラミングが注目されており、初等中等教育で取り入れられつつある。ここでのプログラミング教育は、専門家の養成を目的としたものではなく、情報化社会を生き抜くために誰もが有するべき力を養うことを目指すものである。

一方、平成 25 年に出された閣議決定「世界最先端 IT 国家創造宣言」[(1)]では、初等中等教育の段階からプログラミングを行うべきであると記されているが、これは今後の IT 社会の発展を担う IT 専門家の養成を意図したものと思われる。

前者の目的のために作られたツールとして Scratch[(2)(3)]が注目されている。Scratch はプログラミングを学ぶだけではなく、作文や絵を描くこと同じような感覚で、創造的な活動をおこなうための道具として、多くの小中学生を中心に利用されている。Scratch は、ブロックを組み合わせることでプログラムを作成でき、構文エラーが起こりにくいという特徴がある。しかし、Scratch では扱えるブロックの種類が 100 以上あり、初学者がその全てを一度に見せられると混乱しかねず、何らかの誘導が必要と思われる。

また Scratch を IT 専門家の養成のための入門用として用いることを想定した場合、C や Java などの既存言語との差が大きいため、これらの入門用としては適切であるかどうかは疑問である。

本研究では、Scratch の GUI 部分のみを抽出した、OpenBlocks[(4)]を利用して、初学者により理解し易いプログラミング学習環境を開発した。Scratch のメリットを引き継ぎつつ、学習者の負担を軽減するために、段階的に学べる仕組みを導入した。これにより利用者の学習に必要なブロックを、学習の段階に応じて表示できるようにした。さらに、実行環境とデバッグ機能も実装しており、ブロックで組み立てたプログラムを 1 ステップずつ実行することが可能であり、実行中の変数や制御の流れを逐次確認することもできる。また、専門家養成のための入門用として用いることも考慮し、画面上で組み立てたプログラムを C や Java に変換できるようにしている。

† 大阪市立大学, Osaka City University

ⓒ 2014 Information Processing Society of Japan

2. 関連研究

2.1 Scratch

Scratch は、Squeak Etoys[5]をベースに開発された初学者向けプログラミング環境である。Scratch では、画面上のブロックを組み立てることでプログラムを構築でき、構文エラーが起こりにくいという特徴があるため、初学者でも簡単にプログラミングが可能となっている。一方、画面に表示されるブロックの種類が多く、初学者にとっては混乱を招く要因となっている。

2.2 OpenBlocks

OpenBlocks は Scratch の GUI 部分を抽出した、プログラミング環境を構築するためのライブラリである。 Scratch のような実行環境は備わっていないが、オープンソースとして公開されている。

BlockEditor[6]は、OpenBlocks を拡張して、Java の入門教育用として開発されたプログラミング学習環境である。ブロックで組み立てたプログラムを Java 言語に変換できる。また、いくつかのブロックを折畳んで 1 つのブロックのようにできる抽象化ブロック機能が実装されている。そして、ブロックの表現も英語表記から日本語表記に変更されている。

3. 提案するソフトウェア

3.1 提案するソフトウェアの概要

本研究で提案する初学者向けプログラミング学習環境 **oPEN** (Openblocks based Programming Environment for Novices) の外観を図 3.1 に示す。

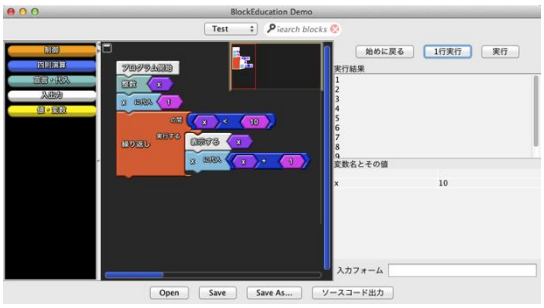


図 3.1 oPEN の画面表示例

oPEN は、OpenBlocks をベースに、初学者によりやさしいプログラミング学習環境を目指して開発した。図 3.1 の画面左側はワークスペースであり、ここでブロックを組み立ててプログラミングを行う。画面の右側はプログラムの実行結果や、変数の値を表示する画面である。

oPEN を授業で用いる際に、教員が所定の定義ファイルを作成することで、授業の目的に応じてカスタマイズできる。この機能を用いて、学習段階に応じて必要なブロックを段階的に追加することで（「ステージ分け」と呼ぶ）、初学者への負担を軽減できる。

3.2 各機能の概要

oPEN の主な機能を以下に示す。

3.2.1 実行とデバッグ

「実行とデバッグ」機能として、組み立てたプログラムをその場で一括実行したり、一ステップずつ実行させることもできる。ステップ実行では、実行中の変数の値を確認することができ、また、図 3.2 のように実行中のブロックを強調表示(図 3.2 の黄色く囲われたブロック)するため、プログラムの制御の流れを追うこともできる。

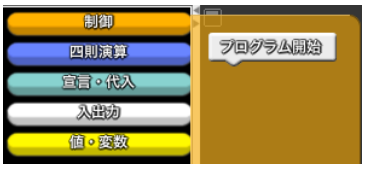
図 3.2 実行中のブロック（黄色の枠）

プログラムが不完全な場合、例えば必要なブロックが接続されていなかったり、未定義の変数がプログラム中で利用されていたりした場合、実行時にエラーとなり、図 3.3 のようにエラーが発生したブロックを赤色の点滅で強調し、より詳細なエラー内容を実行結果欄で使用者に通知する。

図 3.3 実行時のエラー

3.2.2 ステージ分け

「ステージ分け」は、プログラムの学習の段階に応じて、必要なブロックだけを表示させる機能である。ステージの分け方は所定の xml ファイルに記述することで任意に設定できる。ステージはカテゴリー分けするメニューボタン(図 3.4(a)の左側)と、カテゴリーに属するブロック群(図 3.4(b)～(e))からなる。



(a) 制御

(b) 四則演算

(c) 宣言・代入

(d) 入出力

(e) 値・変数

図 3.4 カテゴリーメニューと各ブロック

ⓒ 2014 Information Processing Society of Japan

3.2.3 既存言語への変換

「既存言語への変換」機能として、組み立てたブロックをCやJavaなどの既存言語に変換する機能を用意している。この機能を利用することで、既存言語の学習の導入としても oPEN を利用することができる。
現状では Java と PEN(xDNCL)[8]へ変換が可能であるが、所定のファイルを追加することで、他の言語に対応することも可能である。

4. oPEN のカスタマイズ

4.1 各種の設定ファイル

ここでは oPEN をカスタマイズするための個々のブロックやステージ等の各種の設定ファイルについて述べる。

```
本体
 ├Language
 |      ├PEN.xml              //PEN 用の言語変換ルール
 ├Stage
 |      ├PEN                  //ステージの総称フォルダ
 |          ├ステージ 1.xml   //ステージ 1 のファイル
 |          ├ステージ 2.xml   //ステージ 2 のファイル
 ├Resources
 |      ├blockInfo.xml        //ブロック情報ファイル
 |      ├blockInfo.dtd
 |      ├startUp.properties   //起動時の設定ファイル
```

図 4.1 各種設定ファイルの構成

OpenBlocks ではブロックの定義や、ブロックの表示は 1 つの XML ファイルに記述されているが、本研究では図 4.1 のようにいくつかのファイルに分けてカスタマイズを容易にした。

図 4.1 の Language フォルダには「言語変換」機能を利用するための言語変換ルールを記述した XML ファイルを配置する。Stage フォルダには、ステージの総称を表すフォルダを置き、そのフォルダの下に各ステージを記述した XML ファイルを配置する。Resources フォルダには、個々のブロックの定義を記述した blockInfo.xml や、起動時の設定を記述した startUp.properties を配置する。

4.2 ステージの記述

図 3.4 のステージは、図 4.2 の XML ファイルの記述によりに実現している。

```
 1: <?xml version="1.0" encoding="UTF-8" standalone="no"?>
 2: <root>
 3: <Output>
 4:   <Language>JAVA.xml</Language>
 5:   <FileNameExtention>.java</FileNameExtention>
 6:   <OutputButton>ON</OutputButton>
 7: </Output>
 8:
 9: <BlockDrawerSets>
10: <BlockDrawerSet drawer-draggable="no" location="southwest"
11:       name="factory" type="stack" window-per-drawer="no">
12:  <BlockDrawer name="制御" type="factory" button-color="255 173 0">
13:       <BlockGenusMember>start</BlockGenusMember>
14:  </BlockDrawer>
15:  <BlockDrawer name="四則演算" type="factory" button-color="102 129 255">
16:       <BlockGenusMember>sum</BlockGenusMember>
17:       <BlockGenusMember>difference</BlockGenusMember>
18:       <BlockGenusMember>quotient</BlockGenusMember>
19:       <BlockGenusMember>product</BlockGenusMember>
20:  </BlockDrawer>
      ・
      ・(省略)
      ・
30: </BlockDrawerSet>
31: </BlockDrawerSets>
32: </root>
```

図 4.2 ステージの記述

図 4.2 の<BlockDrawer>タグがメニューボタンの記述であり、属性「name」がボタンに表示される名称を表しており、属性「color」でボタンの色を指定している。の<BlockGenusMember>タグはカテゴリー内に表示するブロック名を記述する。なお、ブロック名はブロック定義ファイル(blockInfo.xml)で定義されている。

図 4.2 の 4 行目、<Language>タグ内には、このステージ内で「言語変換」機能を利用した際の、言語変換ルールを記述した XML ファイル(図 4.1 の Language フォルダ内)を指定する。「言語変換」機能を利用しない場合、ここに「NULL」と記述することで oPEN 上に「言語変換」のボタンを表示しない。また、「FREE」と記述することで、oPEN 上で言語変換ルールを選択することもできる。この場合は Language フォルダ下にある言語変換ルールから出力する言語を起動時に選択できる。5 行目の<FileNameExtention>のタグは、言語変換後に出力されるファイルに付けられる拡張子を指定している 6 行目の<OutputButton>のタグは「言語変換」機能を実行するボタンの表示(ON)/非表示(OFF)を切り替える。

4.3 起動時の設定ファイルについて

oPEN の各種の設定は、startUp.properties に記述する。このファイルは oPEN の起動時に参照され、ブロックの情報が書かれた XML ファイルの所在や、どのステージから始めるかなどを記述している。

＊ブロック情報の参照先

個々のブロックの情報が記されたファイルの参照先は次のように記述する。

```
select_BlockAllData = resources/blockInfo.xml
```

＊ステージフォルダの参照先

利用するステージファイルが配置されているフォルダを次のように指定する。なお、ここで指定できるのは、図 4.1

ⓒ 2014 Information Processing Society of Japan

の Stage フォルダ下にあるフォルダ名に限られる。

```
select_BlockDrawerList_Folder = PEN
```

ここで指定したフォルダ下にあるステージファイルが、3.4 節のステージ切り替え時に選択できるようになる。この例では、oPEN の利用者は Stage/PEN フォルダ内にあるステージであれば、ステージ切り替え機能で切り替えることができる。

＊開始ステージの設定

前項「ステージフォルダの参照先」で指定したフォルダ（この場合は PEN フォルダ）下にあるステージファイルを選んだ状態で oPEN を起動できる。

例えば、stage/PEN/Stage1.xml を開始ステージとして設定するには、次のようにファイル名を記述する。

```
select_BlockDrawerList = Stage1.xml
```

また、「FREE」と記述した場合、oPEN 起動時にポップアップメニューの選択画面が表示され、利用者がどのステージから始めるかを選ぶことができる。

```
select_BlockDrawerList = FREE
```

4.4 新たなブロックの作成

新たなブロックを定義方法を示す。例えば、図 4.4 のブロックを定義するには、図 4.3 に示す情報を、ブロック情報の XML ファイル(blockInfo.xml)へ追記する。

```
1:<BlockGenus name="NewBlock" kind="command" initlabel="新たなブロック"
2:                                           color="126 206 244">
3:  <description>
4:       <text>新しく定義したブロックです</text>
5:  </description>
6:  <BlockConnectors>
7:       <BlockConnector label="数値コネクタ"
8:            connector-kind="socket" connector-type="number">
9:       </BlockConnector>
10:      <BlockConnector label="文字コネクタ"
11:           connector-kind="socket" connector-type="string">
12:      </BlockConnector>
13: </BlockConnectors>
14: <LangSpecProperties>
15:      <LangSpecProperty key="vm-cmd-name" value="NewBlock">
16:      </LangSpecProperty>
17:      <LangSpecProperty key="stack-type" value="breed">
18:      </LangSpecProperty>
19:      </LangSpecProperties>
20: </BlockGenus>
```

図 4.3 ブロック情報の記述

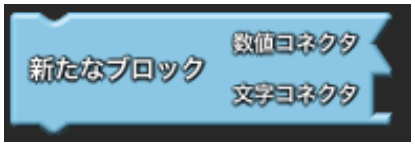


図 4.4 生成されるブロック

＊　ブロックの属性情報（1～2 行目）

図 4.3 の<BlockGenus>タグの属性「name」は、ブロック名であり、これはブロック定義ファイル(blockInfo.xml)で定義されている名前でなければならない。次に属性「kind」は、ブロックの役割を表すもので、下記のいずれかを指定する。属性「initlabel」の値は、ブロックに表示されるラベルの初期値である。

command: プログラムの制御に関するブロックで、自身が何らかのアクションを起こすブロック。
function: プログラム中の数式や、他の式の呼び出しなどのような、自身はデータを持たないが他のブロックを参照することで値を持つブロック。
data: プログラム中の定数や文字列などの、自身が何らかの値を持つブロック。

＊説明書き (3～5 行目)

図 4.3 の<description>タグは、マウスカーソルがブロック上にあるときに表示される説明書きである(図 4.5)。

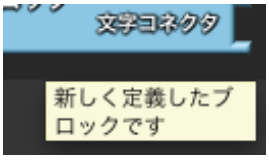


図 4.5 説明表示

＊ブロックのコネクター (6～13 行目)

図 4.3 の<BlockConnectors>タグは、ブロックが持つコネクターの種類等が記述されている。

図 4.3 の 7 行目と 10 行目にある<BlockConnector>タグの数だけ、ブロックはコネクターを保持する。図 4.3 の場合、該当するタグは 2 つあるので、生成されるブロックは 2 つのコネクターを有する。<BlockConnector>タグの属性「label」は、そのコネクターに表示する文字列を表す。

属性「connector-kind」は、コネクターが凹型なのか、凸型なのかを決める。凹型(図 4.6 左側)の場合は、connector-kind の値は”socket”である。または、凸型(図 4.6 右側)の場合は、connector-kind の値は　“plug”である。

図 4.6 コネクターの形（左が凹型、右が凸型）

属性「connector-type」は、コネクターの種類を表す。コネクターの種類の主なタイプは数値(number)、文字列(string)、論理(boolean)のいずれかである。

4.5 新たな言語変換ルールへの対応

「言語変換」機能において、新たに変換する言語を追加したい場合の言語変換ルールの定義方法を示す。図 4.7 はその言語変換ルールの XML ファイルの内容である。

ⓒ 2014 Information Processing Society of Japan

```
1: <BlockCode name="NewBlock">
2:    <CodeText>
3:       @新しく定義したブロックです。 @_br
4:       @コネクター１の値は「 @_val
5:       @」で、コネクター２の値は「 @_val
6:       @」です。
7:    </CodeText>
8:   <PreText></PreText>
10: </BlockCode>
```

図 4.7 言語変換ルール xml

次の図 4.8 は図 4.7 の言語変換ルールの出力結果である。

図 4.8 ブロックと出力結果

図 4.7 の 2 行目～6 行目の<CodeText>タグ内は、言語変換の際に出力されるコードを記述する箇所である。「@」は、文字と特殊タグ(表 1)との区切りを表す記号で、言語変換の際には出力されない。特殊タグはアンダースコアから始まるタグで、言語変換の際に動的に何らかの値に置き換わる。特殊タグに指定されていない文字列はそのまま出力される。このタグ内では半角スペース、改行、タブは全て無視して読み込むため、これらが出力するコードの中で必要な場合は、表 1 のタグで指定する。

表 1 <CodeText>タグ内で扱える特殊タグ一覧

特殊タグ	仕様
_val	コネクターに接続された値を取得する。
_label	ブロックのラベルを取得する。
_preval	コネクターに接続された値を、行の始めにインデントを付けて取得する。
_br	任意の箇所で改行する。
_t	タブキーを挿入する
_space	半角スペースを挿入する。

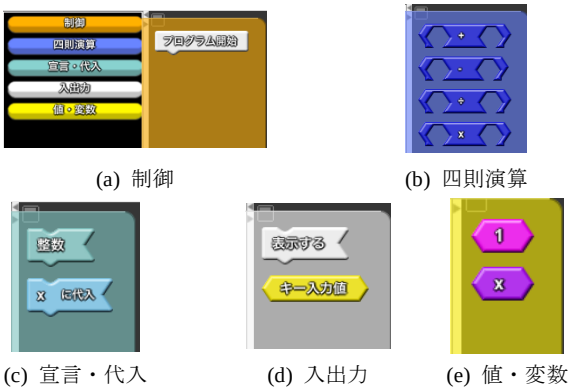
図 4.7 の 8 行目の<PreText>タグ内には、インデント挿入処理が必要な場合は、インデントとして出力される文字または特殊タグを記述する。

5. oPEN 利用例

ここでは oPEN を利用した、コースウェアの例を示す。

5.1 ステージ 1. 逐次処理を学ぶ

ステージ 1 （図 5.1）では、プログラムの基本である逐次処理と変数について学ぶと共に oPEN の操作方法に慣れることを目標とする。



(a) 制御　(b) 四則演算　(c) 宣言・代入　(d) 入出力　(e) 値・変数

図 5.1 ステージ 1 で使用するブロック

図 5.1(a)の左側はカテゴリー分けされたメニューボタンが並んでいる。制御ボタンをクリックすると(a)の右側半分が表示され、四則演算ボタンをクリックすると(b)が表示される（以下同様）。

図 5.1(c)は変数の宣言と、変数への代入を行うブロックが表示されている。「x に代入」ブロックの変数名（x）は、ブロックをワークスペースに配置後に任意の名前に変更できる。図 5.1(e)では、任意の定数を表すための定数ブロック（「１」）と、変数を参照するための変数ブロック（「x」）が表示される。

逐次処理のステージでは全部で 11 種類のブロックを使用してプログラムの組み立てを行う。これら 11 種類のみを登場させるステージを構成することで、ブロックの種類を減らし、導入時の学生への負担を軽減した。

このステージでの例題、練習問題の例を以下に示す。

[例題 1-1] １＋３の結果を表示する（図 5.2）。

図 5.2 例題 1-1

[例題 1-2] キーボードから入力した値を３倍にして表示する（図 5.3）。

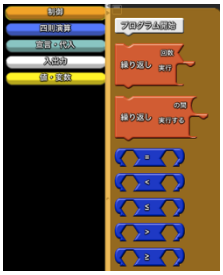
図 5.3 例題 1-2

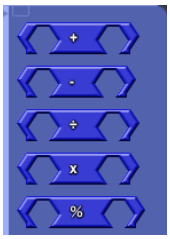
[練習問題 1-1] 変数 x に１＋２＋３の計算結果を代入し、それを出力する。

5.2 ステージ 2. 繰り返しを学ぶ

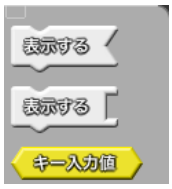
ステージ 2（図 5.4）では、繰り返し処理について学ぶ。ステージ 1 で扱った練習問題 1-1 のような問題を繰り返し処理として解決できることを学ぶ。

ⓒ 2014 Information Processing Society of Japan


(a) 制御


(b) 四則演算

(c) 宣言・代入



(d) 入出力

(e) 値・変数

図 5.4 ステージ 2 で使用するブロック

ステージ 2 で追加されたブロックは(a)の「制御」に「繰り返し」ブロックが 2 種類、「比較演算」ブロックが 5 種類、(b)の「四則演算」に「剰余算(%)」ブロック、(d)の「入出力」に「表示する(文字列用)」ブロック、(e)の「値・変数」に「テキスト」ブロックとなり、ブロックは全部で 21 種類となる。

[例題 2-1] 指定した回数だけ変数 x に 1 を加算することを繰り返し、繰返す度にその計算結果を表示する（図 5.5)。

図 5.5 例題 2-1

[例題 2-2] 1 から順に 1 ずつ増やしながら、入力された値に達するまで順に数を出力する (図 5.6)。

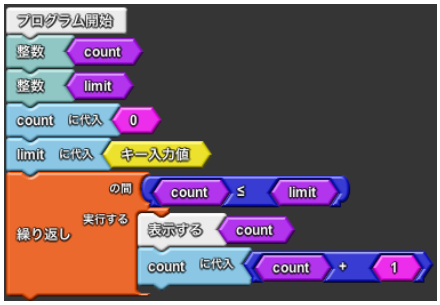
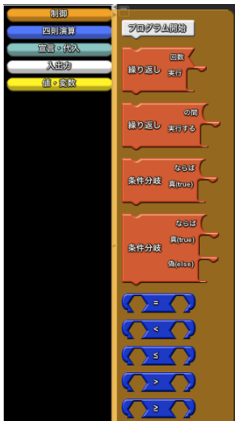
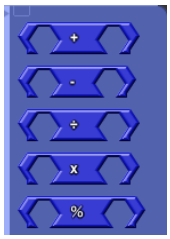


図 5.6 例題 2-2

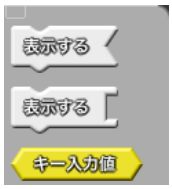
5.3 ステージ 3. 条件分岐を学ぶ

ステージ 3（図 5.7）では、条件分岐について学ぶ。


(a) 制御


(b) 四則演算

(c) 宣言・代入



(d) 入出力

(e) 値・変数

図 5.7 ステージ 3 で使用するブロック

図 5.7 の(a)の「制御」カテゴリー以外は、繰り返し処理のステージと同じである。(a)の「制御」カテゴリーは、ステージ 2 に加えて、条件分岐用のブロック 2 種類が追加され、ブロックは全部で 23 種類となっている。

[例題 3-1] キーボードから入力された値が偶数なら、入力値を 2 倍にして出力し、奇数ならそのまま出力する(図 5.8)。

図 5.8 例題 3-1

[例題 3-2] 0 から始め、入力された値までの偶数のみの合計値を出力する(図 5.9)。

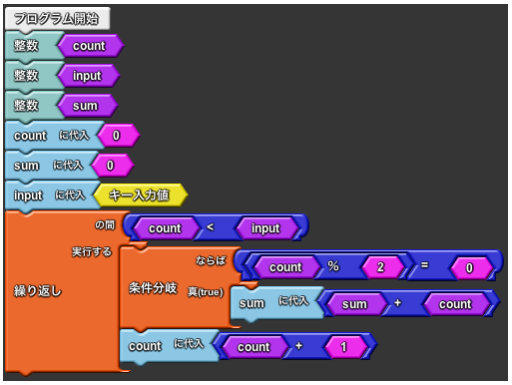


図 5.9 例題 3-2

ⓒ 2014 Information Processing Society of Japan

6. おわりに

OpenBlocks を用いて初学者向けプログラミング学習環境 oPEN を開発した。oPEN には「ステージ分け」「実行とデバッグ」「言語変換」の 3 つの機能があり、それぞれ授業や目的に応じたカスタマイズが可能なので、利用者の学習への負担を軽減できる。

今後の課題としては、ステージカスタマイズの際の XML ファイルの書き換えを GUI で定義できるツールを開発し、学習者だけではなく oPEN を利用する教員の負担の軽減をはかりたい。また「言語変換」機能を拡張し、言語変換ルールに記載されている文法で、既存言語(C や Java)からブロックへの逆変換機能を実現したい。また、実際のに oPEN を使った授業で評価を行い、そこで得られた結果を元にさらに改良を加えたい。

参考文献

1) 世界最先端 IT 国家創造宣言について, http://www.kantei.go.jp/jp/singi/it2/kettei/pdf/20130614/siryou1.pdf (2014/1/20 アクセス).

2) Mitchel Resnick, John Maloney, Andres Monroyhernadez, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Braian Silverman, Yasmin Kafai: “Scratch: Programming for All”, CACM, Vol.52, pp. 60-67, (2009).

3) Scratch - Imagine, Program, Share – MIT, http://scratch.mit.edu/ (2014/1/20 アクセス).

4) Ricarose Vallarta Roque – “OpenBlocks: An Extendable Framework for Graphical Block Programming Systems”, Electrical Engineering and Computer Sciences - Master's degree, (2007).

5) Alan Kay: “Squeak Etoys, Children & Learning”, VPRI Research Note RN-2005-001 (2005).

6)保井 元, 松澤 芳昭, 酒井 三四郎: ”ブロックエディタ方式によるプログラミング構造化教育支援システム”, IPSJ Vol.2012-CE-113 No.11 (2012).

7) Ardublock – a graphical programming language for Arduino, http://blog.ardublock.com/ (2014/1/20 アクセス).

8) ArduBlock プロジェクト日本語トップページ, http://sourceforge.jp/projects/sfnet_ardublock/ (2014/1/20 アクセス).

9) 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”, 情処論誌, Vol.48 No.8, pp. 2736-2747 (2007).

ⓒ 2014 Information Processing Society of Japan