ブロック型プログラミングの作業効率を向上するユーザインタフェース

A User Interface for Improving the Efficiency of Block-Based Programming

稲山　陽太
Yota Inayama
法政大学情報科学部ディジタルメディア学科
E-mail: yota.inayama.5w@stu.hosei.ac.jp

Abstract

Block-based programming languages are paid attention to as a method of teaching programming because they allow users to write programs visually with blocks and also because they do not cause syntax errors. Since the users can write programs only with drag-and-drop and do not need to have difficult knowledge, block-based languages are suitable for beginners of programming. However, they often require the users to take out blocks from block lists by dragging and then to put them on workspaces by dropping. Such a user interface is inefficient because the users always need to do drag-and-drop. Even though there are many block-based languages, they have similar user interfaces. This paper proposes a user interface that improves the efficiency of block-based programming. It provides three new functions: (1) the semi-automatic connection of blocks without drag-and-drop; (2) a pie menu for the selection of a category of blocks; (3) the focus+context visualization of blocks that dynamically changes their sizes. This user interface was developed as a web application that runs on a web browser. This paper shows the results of experiments on the evaluation of the user interface.

1. はじめに

近年プログラミングの教育法としてビジュアルプログラミングの一種であるブロック型プログラミングが注目されている．ブロック型プログラミングはプログラムをテキストで記述するのではなく画面上でブロックを組み立てることで視覚的にプログラムを記述する点や，文法上正しい場合にのみブロックを組み合わせられる設計から，プログラミング未経験者やPCの操作に慣れていない年少者でも手軽にプログラミングの基礎を学習できる．

ブロック型プログラミングのユーザインタフェース(UI)は，ブロック一覧からブロックを取り出してドラッグ＆ドロップでワークスペース上にブロックを組み立てるというものが一般的になっている．しかし実際にプログラムを記述すると，ブロックの移動先が決まっているにもかかわらず毎回ドラッグ＆ドロップする必要がありマウスカーソルの移動が多くなるなど非効率的に感じる場面が多い．また，近年様々な種類のブロック型プログラミング言語が開発されているが，UI の研究はほとんど行われていない．

本論文では，従来のブロック型プログラミングにおけるプログラムの記述の効率化を目的として，以下の 3 つの新しい機能を備えた UI を提案する．

- ブロックの半自動入力．選択したブロックを自動でワークスペース上のブロックに接続する．
- パイメニュー．カテゴリ切り替えをパイメニューによって行う．
- フォーカス＋コンテキスト．マウスカーソルの位置に応じてブロック一覧上のブロックの大きさを動的に変更する．

また，提案したそれぞれの UI に対してプログラムの記述にかかる時間を計測し，従来の UI と比較することで提案手法が正しく機能しているかの評価を行う．さらに実験の結果に対して考察を行う．

2. 関連研究

Scratch [1]は 2006 年に MIT メディアラボが開発したブロック型プログラミング言語である．子供を対象にしたプログラミング教育を目的に設計されており，世界中で使用されている．プログラムの記述は GUI を通して視覚的に行われ，キャラクターや背景に対してブロックを組み合わせて動作をプログラムすることでゲームやアニメーションを製作することができる．Scratch にはユーザが快適にプログラムを記述するための機能がある．例えば，ワークスペース上のブロックを複製する機能がある．この機能によって一度設置したブロックを再び一覧から見つけ出す必要がなくなる他，一連のブロックの塊をコピー＆ペーストすることで入力の手間を省くことができる．Scratch について様々な研究がなされているが，子供プログラミング学習に与える影響などの内容がほとんどであり，UI について研究されることはほとんどない．

MOONBlock[1]は秋葉原リサーチセンターで開発されたブロック型プログラミング言語である．Scratch と類似した UI の構成であるが，通常時はブロック一覧が表示されておらず，ブロックのカテゴリを選択するとブロック一

[1] http://moonblock.jp/

Supervisor: Prof. Hiroshi Hosobe

覧が表示されるため，ワークスペースを広く使用することができる．ブロックのカテゴリ一覧は画面上に全て表示されておらず，カテゴリ一覧を全て確認するためには左右にドラッグする必要がある．

プログラミン[2]は文部科学省が開発したブロック型プログラミング言語で UI の設計に Scratch を参考にしている．この環境ではブロックのカテゴリ一覧が存在せず全てのブロックが画面上に表示されている．ブロック一覧上のブロックの機能がテキストではなくアイコンで視覚的に表現されているので，ブロックが全て表示されていてもかさばることがなく視認性も高い．

3. ブロック型プログラミング

3.1. 従来の UI

Scratch の UI はブロック型プログラミングで代表的なものである．主に 3 つの要素で UI が構成されている．1 つ目はブロックのカテゴリ一覧である．多数存在するブロックをカテゴリで分類することで，ユーザが目的のブロックを見つけやすくしている．2 つ目はブロック一覧である．選択されているカテゴリに分類されたブロックが全て表示されている．3 つ目はワークスペースである．ワークスペースはプログラムを記述する場所である．ブロック一覧からドラッグでブロックを移動させて，ドロップでワークスペース上に設置し，ブロック同士を組み合わせる．

3.2. 問題点

従来の UI には 3 つの問題点がある．

- カテゴリの切り替えが多い点．プログラムを記述する際には様々な種類のブロックを不規則に使用するためカテゴリの切り替えが頻繁に行われる．カテゴリの切り替えのために，画面所定個所のカテゴリ一覧までマウスカーソルを移動させ，必要なカテゴリを選択しなければならない．
- ドラッグ＆ドロップが多い点．プログラムの記述は全てドラッグ＆ドロップで行われるため，ブロック一覧からワークスペースの間で絶えずマウスカーソルを動かす必要が生じる．プログラムの規模が大きくなりワークスペースを広く使うようになるにつれてマウスカーソルの移動範囲も広くなる．ドラッグ＆ドロップは対象を自由な場所に移動させる際には有効だが，ブロック型プログラミングの場合，ワークスペース上でのブロックの位置はプログラムには影響がないので，ドラッグ＆ドロップが有用に働く場面は少ない．また，ブロックに接続する際には移動先はブロックの下に限られているのでドラッグ＆ドロップである必要性はないと言える．ブロックを接続する際もブロックを接続すると認識する有効範囲が狭いため誤入力の可能性を高める．
- 必要なブロックが見つけにくい点．ブロックはカテゴリで分類されているが，同一カテゴリ内でもブロックの数は多く，形状やテキストの似たブロックが多いため，ブロックごとの見分けがつきにくく，目的のブロックを見つけるのに時間がかかる．

これらの問題点を裏付ける UI 設計における普遍的な 2 つの法則がある．1 つ目は Fitts の法則 [2]である．Fitts の法則は 1954 年に Paul M. Fitts によって発見された原理で，マウスなどの入力装置を使用してモニタ上の特定の点を指すまでにかかる時間を予測することができる．予測には以下の式が用いられる．

$$T = a + b\log_2\left(1 + \frac{D}{W}\right)$$

ただし，Dはターゲットまでの距離，Wはターゲットの大きさ，aはポインタの移動開始時間と停止時間，bはポインタの速度である．この式は，対象までの距離が長く，対象の大きさが小さい程，カーソルの移動に時間がかかることを示している．この法則によって，ドラッグ＆ドロップでブロックを組み立てることやカテゴリの切り替えには時間がかかることが分かる．

2 つ目は Hick の法則 [3]である．Hick の法則は 1885 年から J. Merkel によって研究されてきた原理で，複数の選択肢の中から適切なものを選び操作するまでにかかる時間を予測することができる．予測には以下の式が用いられる

$$T = a + b\log_2(n + 1)$$

ただし，aは意思決定を除く時間，bは平均的意思決定時間，nは選択肢の数である．この式は，選択肢が多い程，時間が必要なことを示している．この法則によって，カテゴリ選択やブロック一覧から目的のブロックを見つけ出すのに時間がかかることが分かる．

4. 提案手法

本研究では，ブロック型プログラミングにおけるプログラムの記述の効率化を目的として，ブロックの半自動入力，パイメニュー，フォーカス＋コンテキストの 3 つの新しい機能を備えた新しい UI を提案する．

4.1. ブロック設置の半自動化

1 つ目の機能は，選択されたブロックをプログラムへ自動的に挿入するものである．この機能によって，ブロック設置のためのドラッグ＆ドロップの回数を減らし，時間短縮を図る．また，ブロックのドロップ時の設置ミスを減らす．

ユーザはワークスペース上に設置されたブロックをクリックすることで接続対象を選択する．選択されたブロックは図 1 のように点滅によって視覚的に表現される．接続対象の選択後，ブロック一覧上のブロックをクリックすると，接続対象のブロックに自動で接続される．ブロックの入力を連続して行えるようにするため，ブロックの接続後に接続対象は自動的に更新される．文法上接続不可能なブロックは選択出来ないので，文法エラーを

[2] http://www.mext.go.jp/programin/

回避できるという利点が損われることはない．ユーザは接続対象のブロックを再度クリックすることで，接続対象の状態を解除できる．

図 1　ブロックの半自動設置

4.2. パイメニュー

2 つ目の機能は，ブロック一覧のカテゴリ切り替えるためのパイメニュー [4]である．パイメニューは円形メニューであり，中心から各項目までの距離が全て等しく短いため通常のリスト型のメニューより速い選択が可能となる．また，項目を角度で判断するため選択する際のミスも少なくなる．

ユーザが画面上を右クリックすることでマウスカーソルを中心に図 2 のようにパイメニューが展開される．パイメニューの項目は本言語におけるブロックの各カテゴリと対応しており，項目を選択することでカテゴリの切り替えを行うことができる．また従来の UI ではカテゴリの切り替えのたびにクリックをする必要があったが，パイメニューでは項目上にマウスカーソルをホバーすることでカテゴリが切り替わるので，カテゴリをまたいで目的のブロックを探すことができる．

図 2　パイメニュー

4.3. フォーカス＋コンテキスト

3 つ目の機能として，ブロック一覧の視認性を高くするためにフォーカス＋コンテキスト [5]を用いる．フォーカス＋コンテキストはデータの詳細な情報とデータ全体の概要を同時に表示することで，重要なデータとデータ全体の構造がどのように関係するのかを確認できるようにする技術である．

ユーザがブロック一覧上でマウスカーソルを動かすと，その位置に応じて図 3 のようにブロックのサイズが動的に変更されるようにする．これによって，ユーザはブロック一覧の全体を見つつ，カーソル付近のブロックの内容を容易に認識できる．また，カーソル付近のブロックが大きくなるため，ブロックの選択も容易になる．

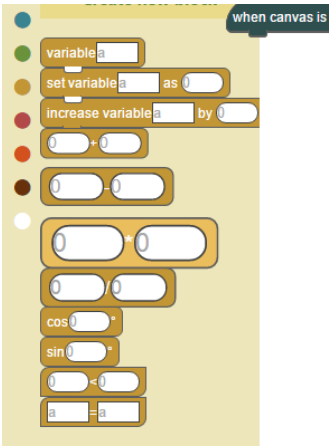

図 3　フォーカス＋コンテキスト

5. 実装

本研究は，栗原らが開発した Processing のコードに対応したブロック型プログラミング言語 [6]の UI を拡張する．本言語はブラウザ上で動作する Web アプリケーションであり，HTML，JavaScript，CSS で記述されている．

本アプリケーションの UI は，ブロック一覧，ブロックのカテゴリ一覧，ワークスペースで構成されており，ワークスペース上でブロックを組み合わせてプログラムを記述する．ブロックのカテゴリは全部で 6 種類ありカテゴリごとにブロックが色分けされている．

6. 実験

実装したそれぞれの UI と全ての UI を組み合わせた場合に対して著者自身による評価実験を行った．実際に動作するプログラムの記述にかかる時間を計測し，従来の UI との比較を行う．この実験の目的は実際の環境で提案した UI が正しく機能するかを確認することである．それぞれの実験は 5 回ずつ行っており結果は平均の値を用いる．実際に動作するプログラムとして Processing 公式サイトの例題からブロック数ごとに選んで使用した．

実験の結果は図 4 のようになった．ただし，「半自動化」はブロック設置の半自動化を，「F+C」はフォーカス＋コンテキストを，「全種類」は全ての UI を組み合わせた場合を表す．ブロックが 2 個の場合，半自動と全種類は 3.0 秒，それ以外が 4.0 秒となり，全ての手法においてほとんど差が見られなかった．ブロック数が 4 個以下の場合，カテゴリ切り替えが行われなかったためパイメニューの効果が得られなかった．ブロックが 20 個の場合，

従来の手法では 58.3 秒，半自動では 31.8 秒と約半分の時間となり．全種類では 25.1 秒と半分以下となった．パイメニューでは 48.6 秒と 15%程度の減少にしかならなかった．F+C では 54.2 秒とほとんど差が見られなかった．

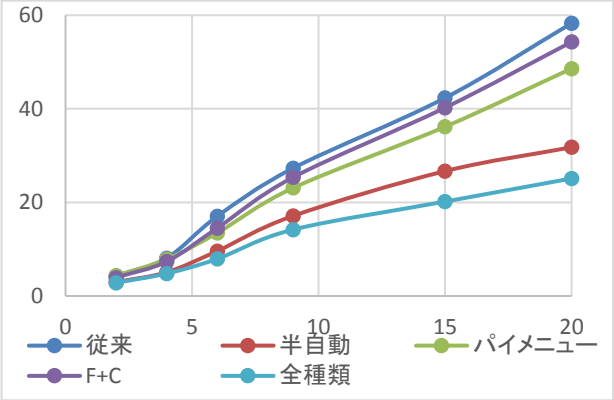

図 4　実験結果

減少率をまとめた結果は以下の図 5 のようになった．ブロックの数が一定以上の場合，プログラムの内容に関わらず一定の割合で時間が短縮できていることが分かる．

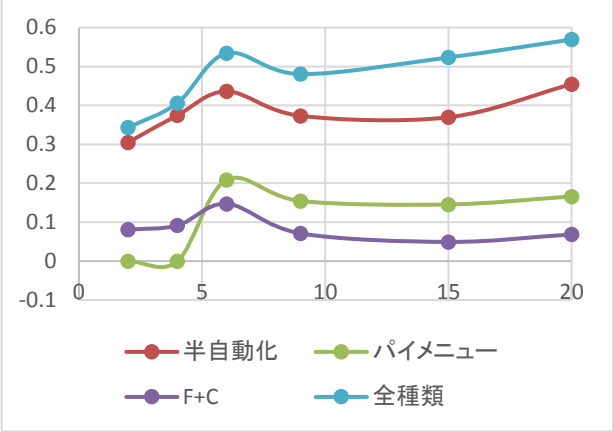

図 5　減少率

7.　考察

7.1. ブロック設置の半自動化

前述の通り，本手法によってプログラムの記述時間が大幅に削減された．プログラムの規模が一定より大きくなると効果が顕著に表れた．これはドラッグ＆ドロップと比較してブロックの入力速度がプログラムの規模に影響されないことが理由と考える．しかし，ブロック数が少ない場合には効果が少なかった．接続対象の選択に時間を要することが原因と思われる．

今回の研究ではブロックの接続のみの半自動化であったため，ブロックへの値の入力のためにワークスペース上のブロックまでマウスカーソルを動かす必要がある．選択中のブロックに値の入力を行うための専用ダイアログ等を用意することでより効率化を期待できる．

7.2. パイメニュー

実験によりパイメニューを用いたカテゴリの切り替えが予想と比べてあまり効果ないことが分かった．本研究の環境ではカテゴリ一覧は画面左端に縦一列で並んでいることが原因であると考えられる．Fitts の法則によると，ディスプレイの隅や端に配置された要素はカーソルが動かないため操作しやすいとされている． Scratch 等のカテゴリ一覧が画面の端に配置されていない環境ではパイメニューの効果が期待できる．

7.3. フォーカス＋コンテキスト

フォーカス＋コンテキストの機能では効果がほとんど見られなかった．これはユーザが環境に慣れてくるにつれてブロックの配置を暗記することで，機能を活用することなくブロックを発見できるようになることが原因と思われる．

ブロックを大きく表示することでブロックの視認性が高くなりブロックの発見が高速化されると予想したが，大きく表示されたブロック自体の視認性は高いものの，動的に大きさが変化しているブロックはテキストが読みにくかった．このため，マウスカーソルを動かしながらブロックを探すことが困難であった．

8. おわりに

本論文ではブロック型プログラミング言語において，プログラム記述におけるブロックの入力効率の向上を目的とした UI を提案した．提案した UI について評価を行った結果，ブロック設置の半自動化が最も効率化が図れていることが示された．フォーカス＋コンテキストはほとんど効果が見られなかったので，ブロックの視認性を高める方法については今後も検討する必要がある．

文　献

[1] M. Resnick, "Scratch: Programming for All," *Communications of the ACM*, vol. 52, no. 11, pp. 60-67, 2009.

[2] I. S. MacKenzie, "Fitts' Law as a Research and Design Tool in Human-Computer Interaction," *Human-Computer Interaction*, vol. 7, pp. 91-139, 1992.

[3] L. Rosati, "How to Design Interfaces for Choice: Hick-Hyman Law and Classification for Information Architecture," *Proc. International UDC Seminar*, pp. 125-138, 2013.

[4] D. Hopkins, "The Design and Implementation of Pie Menus," *Dr. Dobb's Journal*, vol. 16, no. 12, pp. 16-26, 1991.

[5] S. Card, J. Mackinlay and B. Shneiderman, Readings in Information Visualization: Using Vision to Think, 1999.

[6] A. Kurihara, A. Sasaki, K. Wakita and H. Hosobe, "A Programming Environment for Visual Block-Based Domain-Specific Languages," *Procedia Computer Science (Proc. SCSE)*, vol. 62, pp. 287-296, 2015.