Implementing Node-Link Interface into a Block-Based Visual Programming Language

Ryo Suzuki(✉), Takuto Takahashi, Kenta Masuda, and Ikuro Choh

Waseda University, 3-4-1 Ookubo, Shinjuku-Ku, Tokyo 169-8555, Japan
reputeless@gmail.com

Abstract. We developed a novel node-link style interface that can be introduced into a block-based visual programming language as an alternative representation of named variables. By using our new interface, the programmer no longer needs to decide the name of a variable. Tracking the data flow in the program can be easily achieved. Since keyboard typing is not required, the coding is expected to be more accessible to children and persons with disabilities, and it is also suitable for touch operations on mobile phones and tablets.

In our system, as the number of variables increases, the intersections of the links increase, which makes the appearance complicated. To avoid this problem, we implemented improvements in the design, such as emphasizing the focused link list, and making the curves of the links consistent.

Keywords: Visual programming language · Blocks-based programming
User interface

1 Introduction

1.1 Visual Programming Languages

A visual programming language is a programming language that expresses data, flow, and logic by correlation of visual elements such as graphics, icons, and texts. The visual programming language contrasts with text programming consisting of plain text. Many visual programming languages are developed for beginners of programming. In such languages, instead of typing the source code with a keyboard, code is edited by placing elements of the graphical user interface using a mouse or touch. By those design, the effort required for learning coding is made smaller than that of text programming. In programming education for young people, such as computer classes in elementary schools and events like Hour of Code [1], visual programming languages are widely used.

1.2 Block-Based Visual Programming Languages

Visual programming languages that express flows and scopes by aligned blocks are called block-based visual programming languages. A block expresses one unit of programming elements such as variable definition and function call, and the execution order is defined by the direction in which the blocks are connected. Generally ordered from top to bottom, from left to right. The blocks are basically rectangular, and they often have irregularities like a piece of a puzzle, and they visually represent connectable blocks.

1.3 Representation of a Variable

Visual programming languages are often used as introductions to text programming languages. Structures such as loops and branches seen in Scratch [2] and Blockly [3] correspond to those of text programming languages such as C and Java. The concept of variables is also commonly used in these languages.

In this paper, we investigate existing methods for representing variables in visual programming languages and propose node-link interface as a new method.

2 Previous Approach

2.1 Block-Based Visual Programming Languages

Block-based visual programming languages such as Scratch (Fig. 1) and Blockly have been developed for programming education for novices, and they are widely used in introductory computer science classes [4]. This type of interface makes better productivity, makes less syntax errors, and provides better learnability compared to text programming. However, even in such visual programming languages, the concept of variables is still expressed in text form and the advantages of expressive visual programming languages are not utilized. The programmer always needs to read the code carefully to understand the name of the variable, where it is defined, where the value is modified, and where it is used as an argument of the function. Furthermore, naming variables takes time, and causes problems when the entire program is translated into a language of another country for localization.

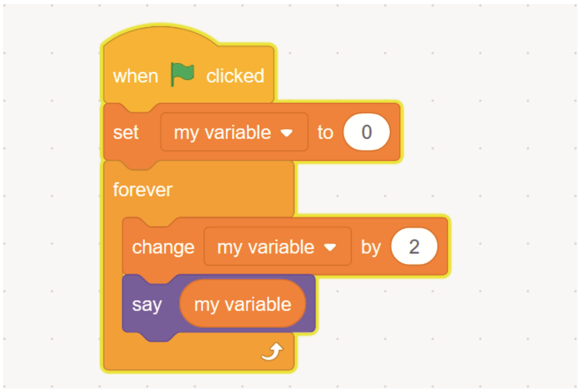

Fig. 1. An example of block-based visual programming language. (Scratch 3.0)

2.2 Node-Based Visual Programming Languages

Node-based interface is also famous option for visual programming languages. In the node-based interface as shown in Fig. 2, a program is created with the directed flow of data between operation and operation in a graph. The programmer connects the output node of the block corresponding to the return value of a function or a variable to an input node of the block to reuse the value. Since a programmer can place the blocks everywhere, space efficiency is poor compared to aligned blocks. In general, variables are not required to be named.

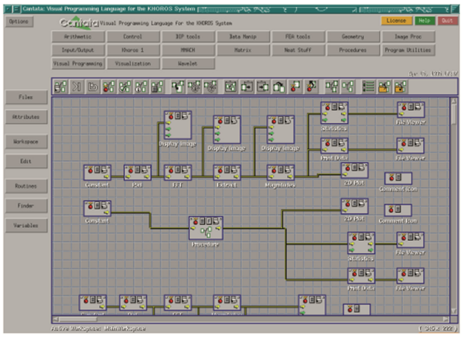

Fig. 2. A classic node-based visual programming language.

3 Node-Link Interface Overview

We developed a new visual programming language "Enrect" [5] that introduces a node-based interface into a block-based visual programming language. In Enrect, flows and scopes are expressed by aligned blocks as well as a block-based visual programming language. Variables are represented by nodes and link curves as shown in Fig. 3.

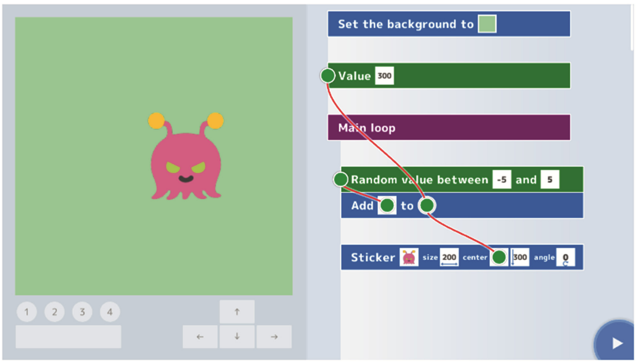
Set the background to
Value 300
Main loop
Random value between -5 and 5
Add to
Sticker size 200 center 300 angle 0
1
2
3
4

Fig. 3. Node-link interface is used instead of named variables in our visual programming language.

The node is displayed at the left end of the block corresponding to a variable, and the user extends the curve of the link by dragging it and connects it to the parameter node (white box) of the argument on the function block. The connected box can be used as a new node that represents the same variable, and a new link can be created by dragging from the node. When a user drags outside the scope of a variable, it is displayed as an invalid operation. To remove the link from the argument box, the user grabs the node and releases it on the box. The order of the nodes and links to be visualized is updated each time when reconnection or block rearrangement event has occurred. The link always starts from the variable block, and the nodes are connected in the order in which they appear in the code. When a variable is statically typed, connectable boxes that accept the value are highlighted while dragging. This is one of the useful interface that cannot be achieved with a text-based IDE. Our code structure using the node-link instead of named variables is essentially the same as the ordinary block-based language, thus it can easily be converted to a text code as a traditional block-based language can.

The purpose of our node-link interface is to achieve the following usability during the coding with a visual programming language.

- Keyboard is not required
- Works on small mobile screen
- Can be operated with one finger
- Can translate source code into other languages (e.g. from English into Japanese).

3.1 Node

As shown in Fig. 4, a node is an element which is used as a starting point of a link that expresses a data flow. Colored circular or square nodes are attached to the left side of variable blocks or function blocks with return values, and white circular or square nodes are inside function blocks that accept arguments. Especially, we call the latter as a parameter node. By starting the drag operation with the mouse or touch from the node, the programmer can create a link and connect it to another node.

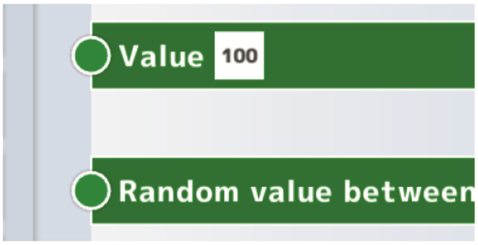

Fig. 4. Variable node is attached to a left side of a variable block or a function block that has a return value.

3.2 Parameter Node

The parameter node is an element expressing the argument of the function as a node. As shown in Fig. 5, a programmer can write a value directly instead of using a variable. The parameter node accepts a specific type of input, dependent on the value type. Programmer cannot connect a node when they don't match.

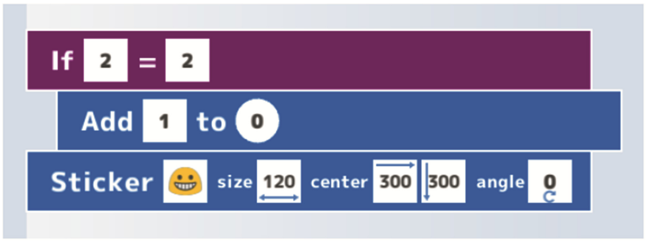

Fig. 5. A block usually has one or more parameter nodes. They are represented as a white box or circle inside the block.

3.3 Link Curve

The curve connecting the nodes is a link. As shown in Fig. 6, it is usually drawn as a cubic Bezier curve connecting the nodes. At the present Enrect specification, there are a maximum of two links connected to one node, but there is also a possibility that it will be increased due to implementation of branch representation in the future.

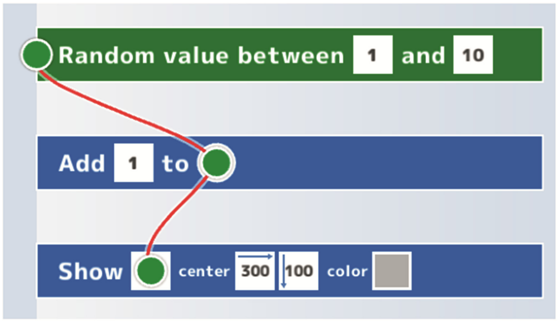

Fig. 6. Nodes are connected by a link curve.

4 Design Features

Enrect has several unique interface features for improving usability of programming. Large font size makes touch operation accurate. The input annotation provides additional information for the user to grasp the specification of the function. In-app software keyboard provides visual consistency and prevents source code from being covered. The shape of a parameter node changes depending on whether the argument may be modified or not. Allowing blank lines improves the expressiveness of source code.

4.1 Font Size

Enrect uses larger fonts compared to other visual programming languages (Table 1). It was designed to make it easier to operate with touch. This size is helpful for a teacher looking at the student's tablet screen in the classroom.

Table 1. Height of the minimum block in block-based visual programming languages.

Scratch 3.0	Blockly	Enrect
33px	24px	58px

It is planned to implement semantic zooming which supports adaptive scaling like online map service.

4.2 Input Annotation

As shown in Fig. 7, in some parameter nodes that receive numerical values, small symbols indicating the use of the numerical values are displayed as input annotations. In the parameter node corresponding to the width, arrows in both directions indicate that it is the thickness of the line. In the parameter node corresponding to the angle, the arrow rotating clockwise indicates the rotation and its direction visually to the programmer. Color and emoji are represented by image indexes and RGB values internally, but they are displayed visually in the source code.

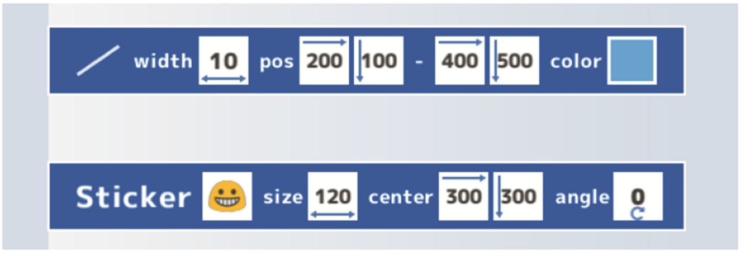

Fig. 7. Some parameter nodes have input annotations that represent their usages.

4.3 Anonymous Variable

By using our interface, variables in the source code are not required to be named by the programmer. Thus, programmer can save time and effort to devise the name of a variable and input it. The source code of Enrect can be translated into the language of each country by switching the system language setting. Since the variables in the source code do not have names, it is accessible for people who don't understand English. For example, a Japanese student and an American student can share their code with no barrier.

4.4 Software Keyboard

Although it is no longer necessary to input the name of a variable, the keyboard is still needed to input the numerical value and the text to be handled by the program. A general problem while coding a visual programming language on a tablet PC is that a software keyboard obscures the editor as shown in Fig. 8. We solved the problem partially by implementing an in-app input window in the editor (Fig. 9). Node link interface is used in the window for consistency.

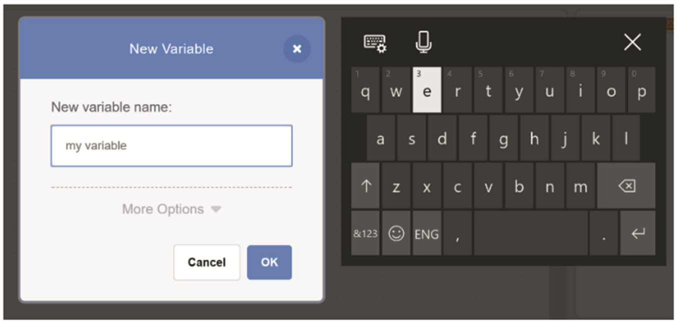

Fig. 8. A text input window and a system software keyboard may hide the source code editor.

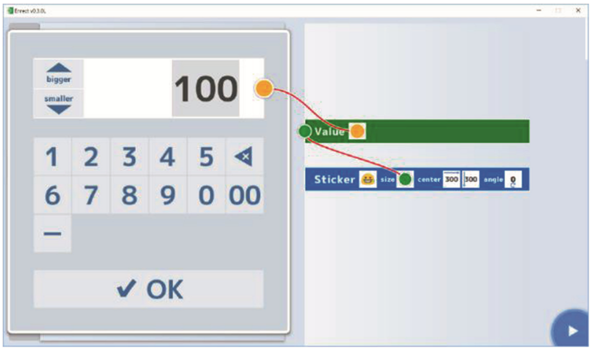

Fig. 9. In-app text input window connected to a parameter block with node-link interface.

4.5 Constant Value

Blocks that represent constants are implemented to make Enrect close to practical text programming languages. Constants cannot be connected to parameter nodes which perform destructive operation. To express it visually, the parameter node which modifies input has circular shape and an immutable variable block has a square node. Then a programmer can connect variable nodes (circle or square) to square parameter nodes, but only circle variable nodes can be connected to circle parameter nodes. This is a visualization of the lvalue/rvalue in C++ showed in the following code (Fig. 10).

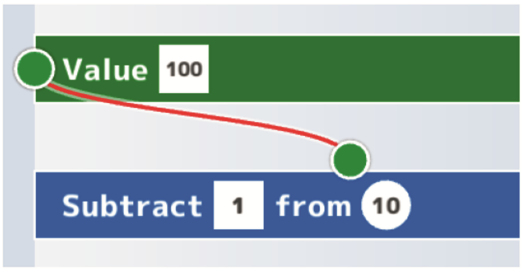

Fig. 10. A circle parameter node only accepts mutable variable.

```
void Add(int& x, int y);

Add(10, 5); // error. `10` is not a lvalue
```

4.6 Blank Line

When the link curve is complicated, and the readability drops, a programmer can eliminate complexity by inserting blank lines between blocks. This also has a secondary effect on the ease of programming. In text programming, a programmer can write blank lines in the code. A blank line is sometimes used to indicate that the contents of the function are not implemented or to express boundaries of processing units. Block-based visual languages generally do not allow elements corresponding to empty lines in consecutive blocks, but Enrect allows blank lines.

4.7 Creating a New Node

A new link cannot be inserted between connected nodes in an ordinary node-based language with one action, but Enrect can create new link for a variable from any nodes as shown in Fig. 11. It can improve usability while editing a long source code.

4.8 Avoiding Confliction

We found that link curves can be complicated when many variables are used in single source code. Then we improved the design of the link curves. The currently selected variable and its link curves are emphasized with bold and highlight visual effect. Each variable has unique color for its link curve. Random offset is added to the curve parameter to avoid overlapping. Curves are rendered consistently and smoothly through the all nodes instead of calculating the curve parameters on each link.

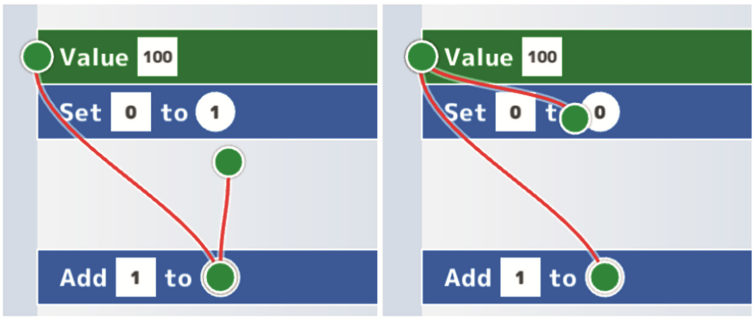

Fig. 11. A new link can be created from any active nodes.

5 Evaluation

5.1 Programming Effort

To investigate how much our programming language reduces the user's effort compared to the existing block-based visual programming language, we measure the minimum number of clicks and drags, and key touches required to implement the sample program. We used FizzBuzz [6] which is a famous programming practice as the sample program. The pseudocode is as follows.

```
int i = 1;

for(;;)
{
  if(i%3 == 0)
    Print("Fizz");
  if(i%5 == 0)
    Print("Buzz");
  if(i%3 != 0 && i%5 != 0)
    Print(i);
  ++i;
}
```

As shown in Table 2, Enrect user can implement the program with less actions.

Table 2. Action counts while coding FizzBuzz.

Language	Click	Drag	Key touch	Sum
Scratch 3.0	16	25	18	59
Blockly	24	24	22	70
Enrect	34	20	0	54

5.2 Understandability

To verify that our interface is comprehensible to beginners of programming, we conducted a experiment. The experiment was conducted for 10 students aged 11 to 12 years. They all use Enrect for the first time. In the experiment, we first presented guidance of Enrect for about 5 min, then let them develop freely for 40 min and collected operation log data. As shown in Table 3, except for one student, all students succeeded in creating multiple variables and links. The number of variable blocks created was 5.8 on average, and the link was created 23.3 times on average.

Table 3. The numbers of variables and links created during the test.

User ID	Variables created	Links created
#1	6	34
#2	4	16
#3	7	28
#4	12	29
#5	5	18
#6	4	21
#7	9	38
#8	4	12
#9	6	34
#10	1	3
Avg.	5.8	23.3

6 Future Work

6.1 Data Flow Animation

In complicated programs, it is difficult to infer the flow of data and the change in state from the source code. By visualizing data moving on link curves between nodes with animation, it is possible to implement a visual debugger that assists development.

6.2 Tagging

When using multiple variables, the meaning of each value is not explicit without its name. It can be improved by providing several types of variable blocks with roughly labeled as "count", "time", etc., or by allowing the user to attach pre-defined tags to variable blocks.

7 Conclusion

We showed our novel node-link style interface that can be introduced into a block-based visual programming language. By using our interface, variables are not required to be named, data flow in the program can be displayed clearly, and a programmer can make a simple program with less effort. Our experiment showed our system is almost comprehensible by novice programmers.

References

1. Hour of Code. https://code.org/learn. Accessed 1 Jan 2018
2. SCRATCH. https://scratch.mit.edu/. Accessed 1 Jan 2018
3. Blockly. https://developers.google.com/blockly/. Accessed 1 Jan 2018
4. Weintrop, D., Wilensky, U.: To block or not to block, that is the question: students' perceptions of blocks-based programming. In: Proceedings of the 14th International Conference on Interaction Design and Children. ACM (2015)
5. Enrect. https://enrect.org. Accessed 1 Jan 2018
6. Using FizzBuzz to Find Developers who Grok Coding. https://imranontech.com/2007/01/24/using-fizzbuzz-to-find-developers-who-grok-coding/. Accessed 1 Jan 2018