Toward an Efficient User Interface for Block-Based Visual Programming

Yota Inayama Hiroshi Hosobe
Faculty of Computer and Information Sciences
Hosei University
Tokyo, Japan
hosobe@acm.org

***Abstract*—Block-based visual programming (BVP) is becoming popular as a basis of programming education. It allows beginners to visually construct programs without suffering from syntax errors. However, a typical user interface for BVP is inefficient partly because the users need to perform many drag-and-drop operations to put blocks on a program, and also partly because they need to find necessary blocks from many choices. To improve the efficiency of constructing programs in a BVP system, we propose a user interface that introduces three new features: (1) the semiautomatic addition of blocks; (2) the use of a pie menu to change categories of blocks; (3) the focus+context visualization of blocks in a category. We implemented a prototype BVP system with the new user interface.**

***Index Terms*—visual programming, block, user interface**

I. Introduction

We propose a user interface that improves the efficiency of block-based visual programming (BVP). It introduces the following three new features:

1) the semiautomatic addition of blocks;
2) the use of a pie menu to change categories of blocks;
3) the focus+context visualization of blocks in a category.

We implemented a prototype BVP system with the new user interface. Our showpiece is the demonstration of this prototype system.

II. Problems with Existing User Interfaces

The user interface of Scratch [5] is a representative of those for BVP. Such user interfaces consist mainly of three components, i.e., a set of categories of blocks, a set of blocks in the currently selected category, and a workspace for programming. Users of such interfaces suffer from the following three problems:

- They need to frequently change categories of blocks;
- They need to perform many drag-and-drop operations to construct programs;
- It is often hard for them to find necessary blocks because there are several categories that contain many blocks.

We can explain these problems by using two well-known principles for user interface design. The first principle is Fitts' law [4]. It is able to predict the time length that a user needs to point at a target on a display with a pointing device such as a mouse. It uses the following formula to predict the time length:

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

where D is the distance to the target, W is the size of the target, and a and b are constants that are determined experimentally. Intuitively, this law indicates that, the longer the distance to the target is, or the smaller the size of the target is, the longer time the user needs to point at the target. We can see from this law that users of BVP need time to construct programs with drag-and-drop operations and also to change categories of blocks.

The second principle is Hick's law [8]. It is able to predict the time length that a user needs to select an appropriate item from multiple choices. It uses the following formula to predict the time length:

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

where n is the number of choices, and a and b are constants that are determined experimentally. Intuitively, this law indicates that, the more choices there are, the longer time the user needs to make a decision. We can see from this law that users of BVP need time to select a category of blocks and also to select a necessary block from a category of blocks.

III. Our User Interface

To improve the efficiency of constructing programs in a BVP system, we propose a user interface that introduces three new features. The first feature is to enable the user to semiautomatically add a selected block to the visual program. It reduces the time by decreasing the number of the drag-and-drop operations for positioning blocks. In addition, it reduces the mistakes that the user makes to position blocks when the user drops them. In our user interface, the user first selects an exiting block on the workspace to indicate that a new block should be added immediately under the selected block. Then the selected block becomes blinking (Figure 1(a)). After this, the user can perform the semiautomatic addition of a new block (Figure 1(b)) just by clicking it on a category of blocks. To enable the successive addition of blocks, such a selected block is automatically updated like a cursor moving in a text editor. The user can also deselect such a block by clicking it.

978-1-5386-4235-1/18/$31.00 ©2018 IEEE

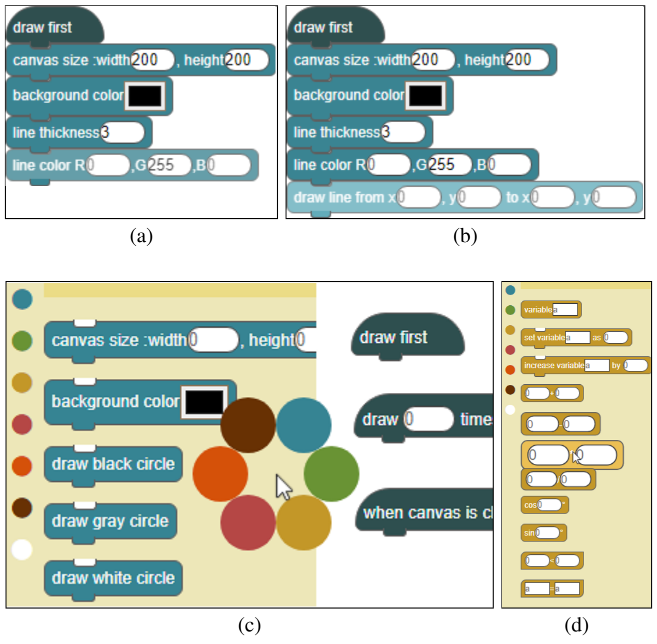


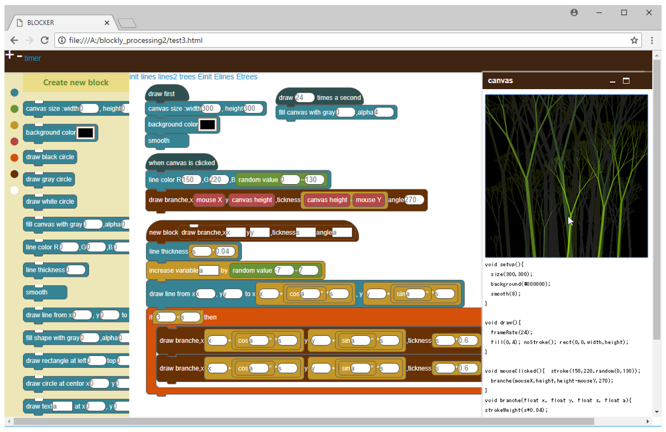
Fig. 1. Our user interface for block-based visual programming.

The second feature is to enable the user to use a pie menu [1] to change categories of blocks. A pie menu is a circular menu where the distance from the center to each menu item is equal and short, which allows the user to more quickly select an item than when using an ordinary linear menu. In addition, it reduces the mistakes that the user makes because the user distinguishes menu items by angles. In our user interface, the user pops up a pie menu around a mouse pointer by pressing the right mouse button (Figure 1(c)). The items in the pie menu correspond to the categories of blocks, and the user can change categories by clicking menu items. In addition, while users of exiting interfaces need to click menu items to see the contents of categories, our user interface allows the user to see the content of a different category only by hovering over a menu item, which immediately shows the corresponding category.

The third feature is to enable the user to use the focus+context visualization [3] of blocks. Focus+context visualization simultaneously shows a particular detail and the overview of given information to enable the understanding of the relationship between the important part and the entire structure of the information. In our user interface, the user can change his/her focus by moving the mouse pointer over blocks in a category (Figure 1(d)). This allows the user to more easily recognize blocks around the mouse pointer while viewing the entire category of blocks at the same time. Also, it eases the user to select a block since blocks around the mouse pointer become larger.

IV. Implementation

We implemented a prototype BVP system adopting the user interface that we proposed in the previous section. For this purpose, we extended Kurihara et al.'s BVP system [2], which generates programs written in Processing [6]. This system is a Web application written in HTML, JavaScript, and CSS that runs on a Web browser by using the Processing.js [7] library. The user interface of our prototype system consists of three typical components, i.e., a set of categories of blocks, a set of blocks in the currently selected category, and a workspace for programming (Figure 2). There are six categories of blocks that are painted with different colors.



```
void setup(){
  size(300,300);
  background(#000000);
  smooth(8);
}

void draw(){
  frameRate(24);
  fill(0,4); noStroke(); rect(0,0,width,height);
}

void mouseClicked(){  stroke(150,220,random(0,130));
  branche(mouseX,height,height-mouseY,270);
}
void branche(float x, float y, float s, float a){
strokeWeight(s*0.04);
```

Fig. 2. Our prototype system.

V. Conclusions and Future Work

We proposed a user interface for BVP that introduced three new features. We also implemented a prototype BVP system with the proposed user interface. Our future work includes the experimental evaluation of the performance of the proposed user interface by comparing it with a typical user interface for BVP. Another future direction is to further explore possible features, for example, for enabling users to efficiently entering values in blocks.

Acknowledgement

This work was partly supported by JSPS KAKENHI Grant Number JP17H01726.

References

[1] D. Hopkins. The design and implementation of pie menus. *Dr. Dobb's J.*, 16(12):16–26, 1991.
[2] A. Kurihara, A. Sasaki, K. Wakita, and H. Hosobe. A programming environment for visual block-based domain-specific languages. In *Proc. SCSE*, volume 62 of *Procedia CS*, pages 287–296, 2015.
[3] J. Lamping and R. Rao. The hyperbolic browser: A focus+context technique for visualizing large hierarchies. *J. Visual Lang. Comput.*, 7(1):33–55, 1996.
[4] I. S. MacKenzie. Fitts' law as a research and design tool in human-computer interaction. *Human-Comput. Interact.*, 7:91–139, 1992.
[5] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond. The Scratch programming language and environment. *ACM Trans. Comput. Educ.*, 10(4):16:1–15, 2010.
[6] C. Reas and B. Fry. Processing: Programming for the media arts. *AI Soc.*, 20(4):526–538, 2006.
[7] J. Resig. Processing.js, 2008. https://johnresig.com/blog/processingjs/
[8] L. Rosati. How to design interfaces for choice: Hick-Hyman law and classification for information architecture. In *Classification & Visualization: Interfaces to Knowledge*, pages 121–134, 2013.