Block Shelves for Visual Programming Languages

Sheng-yi Hsu  
National Chiao Tung University  
Hsinchu, Taiwan  
syhsu@cs.nctu.edu.tw

Yuan-fu Lou  
National Chiao Tung University  
Hsinchu, Taiwan  
yflou@cs.nctu.edu.tw

Chuen-tsai Sun  
National Chiao Tung University  
Hsinchu, Taiwan  
ctsun@cs.nctu.edu.tw

ABSTRACT
The blocks editor, such as the editor in Scratch [13], is widely applied for visual programming languages (VPL) nowadays. Despite it’s friendly for non-programmers, it exists three main limitations while displaying block codes: (1) the readability, (2) the program structure, and (3) the re-use. To cope with these issues, we introduce a novel formatting tool, block shelves, into the editor for organizing blocks. A user could utilize shelves to constitute a user-defined structure for the VPL projects. Based on the experiment results, block shelves improves the block code navigating and searching significantly. Besides, for achieving code re-use, users could use shelf export/import to share/re-use their block codes between projects in the file format of eXtensible Markup Language (xml.) All functions were demonstrated on MIT App inventor 2 [12], while all modifications were made in Google Blockly.

Author Keywords
Human-computer interaction; Computer education; Visual programming languages; IDE; Usability

INTRODUCTION
Visual programming languages (VPLs) are widely applied for reducing the programming barriers for non-programmers nowadays. Scratch [13], MIT App Inventor [12], Alice [4], etc., help the program learners significantly. Besides, many software tools provide VPL as an alternative programming environment for non-programmers, such as Orange [5] for data mining, Modkit [10] and Ardublock for Arduino programming, LabVIEW [7] and Quite Universal Circuit Simulator (Qucs) [2] for circuit simulation, etc. Unlike text editors, VPL is intuitive and easy for non-programmers to program by drag-and-drop. Users first drag a block from a toolbox categorized with the block functions, and put the block in the blocks editor. With the concept of event-first programming [15], the orders or locations are irrelevant to the compiled result of a project/program.

However, according to Okerlund and Turbak [11], a user has a fifty-fifty chance to deal with a project consisting more than 30 blocks while using MIT App Inventor. Undoubtedly, the more functions a user needs, the more blocks will be shown in the editor. From the perspective of personal usage, keeping tracks on all blocks becomes an issue when the blocks are accumulated to an extent. Meanwhile, in order to maintain a project, reminding oneself about all meanings of blocks are also difficult after a period of time; another remaining issue would be the tracing/understanding of blocks after opening others’ projects for referencing or adapting. Another issues may cause problems when, for instance, a lecturer in a workshop/class needs the students to find a certain block in the blocks editor, or when the user want to share his/her blocks with others. In short, VPL effectively lower the barrier for program learning and developing, but unfortunately not for reading, tracing, and maintaining.

To address these challenges, we divide issues arisen from the current blocks editor into 3 categories: (1) the readability, (2) the program structure, and (3) the re-use of block codes. In order to cope with the issues, we introduce a novel formatting tool, block shelves, into the editor for organizing blocks in VPL projects. A block shelf in the editor is used for classifying blocks. Users could arrange the shelves from program functions, block usages, to the orders in an execution process, etc. Besides, every user could re-arrange the blocks various times without any deconstructions until finding the most suitable classification method. For more detailed explanations, the rest of the paper is organized as follow: background and related works, design overview, quantitative evaluation, and conclusion.

BACKGROUND AND RELATED WORKS
Approaches in improving the code readability and the code usability have been researched for decades, but the techniques designed and discussed are mostly for the text-based programs, such as indentation [3], coding style [9], variable naming [14], modulation [6], and applying text context to improve the productivity [8]. However, more and more projects today are block-based. The current blocks editor design provides 3 functions: block commenting, block collapsing and sorting by block category, and block duplicating to manage the readability issues, the structure, and the re-use of block codes. The details of the functions are explained in the following.

- **Block commenting:** Adding comments to blocks is the most common method in the blocks editor to improve the code readability. Not only does it help others to understand your programs but it is also useful to keep one’s own programs on...
track. Nevertheless, the current blocks editor doesn’t provide a comment search function, so a user still needs to check blocks individually.

- **Block collapsing and sorting by block category**: Block collapsing is originally designed for keeping the screen real-estate small; the collapsed blocks could be sorted with the block category. After sorting by block category, the editor would line up the collapsed blocks and show in a horizontal or vertical way. Though block collapsing and sorting by block category makes the block layout succinct and clean, the block layout still lacks a structure or organized system for users to trace the codes.

- **Block duplicating**: Block duplicating is an operation for users to copy and paste a group of selected blocks. However, this operation is only for single project at the moment; therefore, if a user wants to adapt other projects, he/she has to “imitate” (re-build) the blocks from the referenced project.

### DESIGN OVERVIEW

The design of block shelves provides a user-defined structure for shelving blocks in VPL projects. We chose MIT App Inventor 2 (AI2) to demonstrate the block shelves. In MIT AI2, the blocks editor is a Javascript-based Google Blockly combined with Google App Engine. Hence all modifications for demonstrating shelves are made on Google Blockly.

Adding blocks to a shelf is same with using other block functions in AI2. The menu is shown in figure 1. A user could use the right mouse key to add a group of blocks to a shelf in a project. The added shelf is shown in the shelf toolbox, ShelfBox (figure 2). The main function designs for shelves are shelf visibility, shelve minimization/maximization, shelf activation/deactivation, shelf duplication, and shelf exportation/importation. The details of each design are described in the following.

- **Shelf visibility (_enabled)**: Users could utilize this function to hide or show the blocks in the selected shelf in the blocks editor. This design not only saves the screen space efficiently but controls the numbers of blocks shown in the block editor.

- **Shelf minimize/maximize (_enabled)**: This function is adapted from block collapsing/expanding. By collapsing or expanding all blocks in selected shelves, a user controls the numbers of blocks shown in the block editor effectively. However unlike shelf visibility, one still sees a minimized block in the editor after applying this function.

- **Shelf activate/deactivate (_enabled)**: Shelf activation/deactivation is used for activating or deactivating all block functions in the selected shelves. Unlike shelf minimize/maximize, this design is used for tracing codes by checking the block functions.

- **Layer duplicate**: This design is similar to block duplicating; the only difference is that this function duplicates all blocks in a selected shelf, which is useful when creating similar programs or testing functions without changing the original programs.
QUANTITATIVE EVALUATION

Our main focus are code-reading and -navigating; therefore, we first define the twin activities of reading and navigating as code understanding [1], using block searching time and block reading time to measure the differences between the block editors with and without block shelves. In our research framework, the sample was divided into one experimental (App inventor 2 with block shelves) and one control group (original App inventor 2). This evaluation investigates the following hypotheses:

1. whether block shelves improve code understanding,
2. whether block shelves improve code re-use, and
3. whether block shelves improves the searching time of locating a certain block.

Experimental Design and Procedure

A total of 60 graduate and undergraduate students (mean age 23, max age 38, min age 18) were recruited from National Chiao Tung University or National Tsing Hua University. They never use VPL but ever took the courses of computer introduction and programming. All participants are randomly divided into either experimental or control group, 30 students for each group. The experiment includes 3 stages: (1) tutorial, (2) pre-test, and (3) post-test. Both groups joined the tutorial stage and pre-test. In the post-test stage, the experimental and control groups are separated: the experimental group took a introductory tutorial for block shelves before using App inventor 2 with shelves, while the control group used the original App inventor 2. Meanwhile, all trials are conducted with the Chrome browser and recorded with oCam v23.0 (video capture software.) The operating system is Windows 7, and the spec of computers for trials is Intel Core i5 2400 @ 3.10GHz (CPU) with 4GB of ram, a 22” monitor running at 1920x1080@60Hz, and a NVIDIA GeForce GT 630.

In the tutorial stage (40 minutes), we introduced the Android app front-end design and the block code design. The tutorial project includes 197 blocks and 6 different screens to show users how to add a gadget to the mobile phone interface and use block codes to control the gadgets on the interface, including how to add a button, a canvas, a ball, a timer, a graph, and a text tag. Besides, the pre-test adopted calculator (figure 3), which is designed for both groups to test whether abilities between the two groups are identical; In the post-test, Pusheen the cat (figure 4) is adopted for testing whether the shelves improves the user performance. Pusheen the cat is a memory game, which requires a player to remember the location of each number. Every time a player needs to pick two buttons to press and a player earns two points when the sum of the two numbers of the selected buttons is 17. To complete a game a player has to achieve 16 points, and the number in the lower-right corner of the screen shows how long a player used. The tasks in each project are listed in the table 1.

In Calculator (fig. 3), task 1 and task 2 ask users to add the functions of division and number 9 by referencing other arithmetic operations and numbers. These two tasks are designed for checking the conditions of code re-use in the original blocks editor.
Table 1. Tasks in Calculator and Pusheen The Cat

| Task | Calculator (Pre-test) | Pusheen The Cat (Post-test) |
|------|-----------------------|-----------------------------|
| 1.   | Add the function of button ⟨/⟩ | Add the function of button ⟨item 7⟩ |
| 2.   | Add button ⟨9⟩ | Correct the response when choosing two identical photos |
| 3.   | Correct the function of button ⟨+⟩ | Add the display of a cat’s photo after hitting ⟨item 5⟩ |
| 4.   | Correct the function of button ⟨5⟩ | Correct the timer reset after hitting button ⟨Restart⟩ |
| 5.   | Correct the function of button ⟨=⟩ | Correct the text alert when game is over or finished |

The rest tasks (task 3, 4, and 5) are used for observing the code understanding (code reading and code navigating) because users need to locate the target block and then to correct the target block in these three tasks. On the other hand, the block shelves used by the experiment group in Pusheen The Cat (fig. 4), are constructed first with the buttons, as shown in figure 2. In Task 1 and task 3, users are asked to add/correct block codes by referencing other items. Through the two tasks we want to see if block shelves are helpful for code re-use. Task 2, task 4, and task 5 are used for checking whether layers help code understanding since users are asked to locate the target block to fix.

Results and Analysis

We measured every participant’s reading and searching time while doing Calculator and Pusheen The Cat. The searching time refers to the time a participant used for locating the target block after reading a task, and the reading time indicates the time a participant spent to find the bug after locating the target block. The averages and the standard deviations of the pre-test and the post-test are shown in figure 5 and figure 6.

We applied Levene test to check if the variances of the two groups are equivalent. In the pre-test, Calculator (table 1), we first check the searching abilities between the experimental group and the control group: the differences between the two groups are not significant ($P(T ≤ t) = 0.750 > .05$) based on the assumption of equal variance in t-test ($P(F ≤ f) = 0.118 > .05$). This test result also implies the abilities between the two groups were similar in the searching abilities. Further we checked the reading abilities between the two groups. The result of t-test assuming equal variance ($P(F ≤ f) = 0.306 > .05$) shows that the reading abilities between the two groups are also similar ($P(T ≤ t) = 0.974 > .05$). Combining the results mentioned above we concluded that the reading and searching abilities of block codes between the two groups are similar.
After confirming the ability levels between the two groups, we proceeded to the post-test. For the searching time, we found that the experimental group used significantly less time than the control group (\(P(T \leq t) = 4.727E-19 < .05\)) based on \(t\)-test assumed unequal variance (\(P(F\mid f=6.897E-07\;|\;0.05\)). However, the two groups were not significantly different (\(P(T \leq t) = 0.826 > .05\)) in the time spent on reading according to \(t\)-test assumed equal variance (\(P(F \leq f) = 0.062 > .05\)).

The post-test results showed that the block shelves help users navigate and locate the target blocks efficiently. However, as for the reading of block codes, the introduce of block shelves doesn’t improve significantly because the reading time is highly correlated with the extent of a user’s familiarity of block codes. All participants in our experiment were their first time to use the block editor. Hence the block shelves provides users an formatting tool for improving navigating, searching, and tracing in block-based projects.

CONCLUSION

We introduce block shelves, a novel formatting tool, for organizing the block-based projects. The experiment results suggest that the block editor with block shelves helps users navigate and search block codes, which improves the searching time significantly. By adopting shelves users could organize the block codes with their own arrangements. Besides, we also designed five functions for shelves: (1) shelf visibility, (2) shelf minimize/maximize, (3) shelf activate/deactivate, (4) shelf duplication, and (5) shelf export/import. The first four functions are adapted from the original block functions, and the last function is specially designed for cross-project sharing/re-use. Combining the above points, block shelves provides a useful shelving system for the current VPL projects, breaking through the conventional structure and offers a brand new interface for users.

REFERENCES

1. Andrew Bragdon, Robert Zeleznik, Steven P Reiss, Suman Karumuri, William Cheung, Joshua Kaplan, Christopher Coleman, Ferdi Adeputra, and Joseph J LaViola Jr. 2010. Code bubbles: a working set-based interface for code understanding and maintenance. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, 2503–2512.

2. M. E. Brinson and S. Jahn. 2009. Qucs: A GPL software package for circuit simulation, compact device modelling and circuit macromodelling from DC to RF and beyond. International Journal of Numerical Modelling: Electronic Networks, Devices and Fields 22, 4 (2009), 297–319. DOI: http://dx.doi.org/10.1002/jnm.702

3. Mitchell H. Clifton. 1978. A Technique for Making Structured Programs More Readable. SIGPLAN Not. 13, 4 (April 1978), 58–63. DOI: http://dx.doi.org/10.1145/953411.953415

4. Stephen Cooper, Wanda Dann, and Randy Pausch. 2000. Alice: a 3-D tool for introductory programming concepts. In Journal of Computing Sciences in Colleges, Vol. 15. Consortium for Computing Sciences in Colleges, 107–116.

5. Janez Demšar, Tomaz Curk, Aleš Erjavec, Črt Gorup, Tomaz Hočevar, Mitar Milutinovič, Martin Možina, Matija Polajnar, Marko Toplak, Anže Starič, Miha Štajdohar, Lan Umek, Lan Žagar, Jure Žbontar, Marinka Žitnik, and Blaž Zupan. 2013. Orange: Data Mining Toolbox in Python. Journal of Machine Learning Research 14 (2013), 2349–2353. http://jmlr.org/papers/v14/demsar13a.html

6. Stephen G Eick, Todd L. Graves, Alan F. Karr, J. Steve Marron, and Audris Mockus. 2001. Does code decay? assessing the evidence from change management data. Software Engineering, IEEE Transactions on 27, 1 (2001), 1–12.

7. Chance Elliott, Vipin Vijayakumar, Wesley Zink, and Richard Hansen. 2007. National instruments LabVIEW: a programming environment for laboratory automation and measurement. Journal of the Association for Laboratory Automation 12, 1 (2007), 17–24.

8. Mik Kersten and Gail C. Murphy. 2006. Using Task Context to Improve Programmer Productivity. In Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT ’06/FSE-14). ACM, New York, NY, USA, 1–11. DOI: http://dx.doi.org/10.1145/1181775.1181777

9. Richard J. Miara, Joyce A. Musselman, Juan A. Navarro, and Ben Shneiderman. 1983. Program Indentation and Comprehensibility. Commun. ACM 26, 11 (Nov. 1983), 861–867. DOI: http://dx.doi.org/10.1145/182.358437

10. Amon Millner and Edward Baafi. 2011. Modkit: blending and extending approachable platforms for creating computer programs and interactive objects. In Proceedings of the 10th International Conference on Interaction Design and Children. ACM, 250–253.

11. Johanna Okerlund and Franklyn Turbak. 2013. A Preliminary Analysis of App Inventor Blocks Programs. Poster presented at Visual Languages and Human Centric Computing (VLHCC), Sept (2013), 15–19.

12. Shaileen Crawford Pokress and José Juan Dominguez Veiga. 2013. MIT App Inventor: Enabling personal mobile computing. arXiv preprint arXiv:1310.2830 (2013).
13. Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and others. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (2009), 60–67.

14. Barbee E. Teasley. 1994. The Effects of Naming Style and Expertise on Program Comprehension. *Int. J. Hum.-Comput. Stud.* 40, 5 (May 1994), 757–770. DOI: [http://dx.doi.org/10.1006/ijhc.1994.1036](http://dx.doi.org/10.1006/ijhc.1994.1036)

15. Franklyn Turbak, Mark Sherman, Fred Martin, David Wolber, and Shaileen Crawford Pokress. 2014. Events-first Programming in APP Inventor. *J. Comput. Sci. Coll.* 29, 6 (June 2014), 81–89. [http://dl.acm.org/citation.cfm?id=2602724.2602739](http://dl.acm.org/citation.cfm?id=2602724.2602739)