Shelves: A User-Defined Block Management Tool for Visual Programming Languages

Sheng-Yi Hsu, Yuan-Fu Lou, Shing-Yun Jung, Chuen-Tsai Sun

Shelves: A User-defined Block Management Tool for Visual Programming Languages

Sheng-Yi Hus[1,*], Yuan-Fu Lou[1], Shing-Yun Jung[1], Chuen-Tsai Sun[1]

[1]National Chiao Tung Univeristy, Taiwan (R.O.C)
{syhsu, yflou, syjung, ctsun}@cs.nctu.edu.tw

Abstract. Block editors such as the one used in Scratch are now found in many visual programming languages (VPLs). While considered user-friendly for non-programmers or program learners, they have at least three important display limitations: readability, program structure, and re-use. To address these issues we have developed block shelves, a formatting and organizing tool in support of user-defined VPL structures. Usability experiment results indicate that block shelves can significantly enhance block code navigation and searches, as well as project structure clarification. In the interest of improving project collaboration and code re-use, users can utilize the extensible markup language file format to export/import shelves, and thereby share block codes between projects. Features designed for shelves and the experiment findings are value for course design in project-based learning and future block editor interface improvements.

Keywords: human-computer interaction, computer education, visual programming languages, code usability, project-based learning.

1 Introduction

As tools for reducing barriers for non-programmers or for program teaching for children, visual programming languages (VPLs) are currently in widespread use (Fig. 1). Scratch [17], MIT App Inventor [16], and Alice [5] are three examples of VPLs that have been created to help learners create their own programs. The list of software tools that contain VPLs as alternate programming environments include Orange [6] for data mining, Modkit [14], and LabVIEW [9] and Quite Universal Circuit Simulator [3] for circuit simulation, among others. Compared to text editors, VPLs are considered much more intuitive and user-friendly due to their drag-and-drop feature. A user first drags a block from a toolbox, which is normally categorized with the block functions, and then drop the block in the block editor. The event-first programming characteristic makes block location irrelevant to compiled project/program results [19].

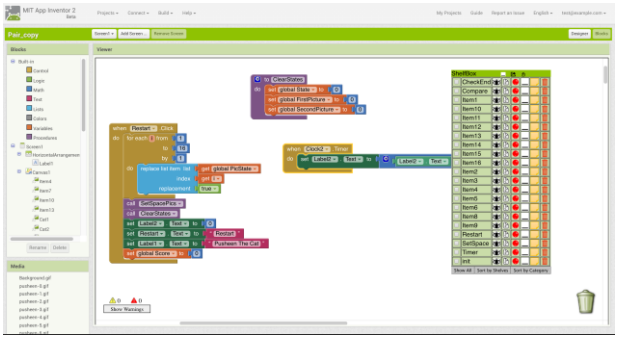


Fig. 1. Screenshots of block editor in MIT App Inventor with block shelves.

However, Weintrop and Wilensky [21] pointed out that the recognition that longer blocks-based programs can be difficult to manage. Meanwhile according to Okerlund and Turbak [15], more than 50% of all MIT App Inventor projects consist of more than 30 blocks, which indicates that a user of MIT App Inventor has a more than 50-50 chance of referencing a project having 30+ blocks. In short, the higher the number of functions, the greater the number of blocks shown in the editor, the larger the canvas size, and the greater the potential for degraded block code readability. Thus, the users of many block editors must make tradeoffs between block code readability and canvas size. Another problem is the lack of block search functions in any of the currently available block editors—this despite the importance of search and navigation support tools for professional developers, who spend at least one-third of their time on redundant but necessary navigation between code fragments [10]. Clearly there is a need to keep track of all blocks during a project and to ensure that each block is easily identifiable. A related issue involves the tracing or reading of blocks when opening projects managed by others for review or adaptation, or when workshop/class instructors want their students to locate specific blocks in their block editors. In sum, VPLs may be very effective tools for lowering barriers to program learning and development, but they are less useful in terms of reading, tracing, and maintenance.

Block editor issues can be categorized as readability, program structure, or block code reuse problems. To address all three issues we have developed a block tagging, formatting, and organizing tool that we call block shelves. The basic idea is that users can arrange block codes on shelves that are labeled as “function,” “usage,” “execution process order,” and so on. Users can re-arrange blocks at any time without deconstruction until they identify the most suitable classification method. Our goal in this paper is to provide a detailed explanation of our proposed tool using a review of related works, a design overview, a quantitative evaluation, and a conclusion.

2 Related Works

Until recently, most approaches to improving code readability and usability have been created in support of text-based programs. Some of the most common involve indentation [4], coding style [13], variable naming [17], modulation [8], and the application

of context to improve productivity [10]. Most current block editors provide 4 primary block functions for managing project complexity [16-17, 20]. However, more and more projects are block-based, and therefore entail four primary block functions:

1. Commenting. This is considered the most common method for improving code readability. Not only do comments help others understand the program creator's ideas, they are also useful for keeping track of one's own projects. However, none of the block editors currently in use have comment search functions, thus forcing users to check blocks individually.

2. Block collapsing. Originally created as a method for keeping screen real estate small, this feature also helps to reduce block code complexity while navigating. After applying this feature, each group of connected blocks is displayed as a single block on a canvas, thereby reducing the overall number of blocks that are shown.

3. Sort by category. After sorting, blocks can be lined up according to category. However, even though block collapsing and sorting can make block layout simple and clean, it still lacks structure or a system for code tracing.

4. Block duplicating. This operation allows users to copy and paste individual or multiple selected blocks. However, this operation is limited to individual projects. Users who want to use the same blocks in other projects must re-build them from the beginning.

In addition to providing block functions, some VPLs also provide a conceptual structure for project managements. The methods used for conceptual structure design can be divided into two categories:

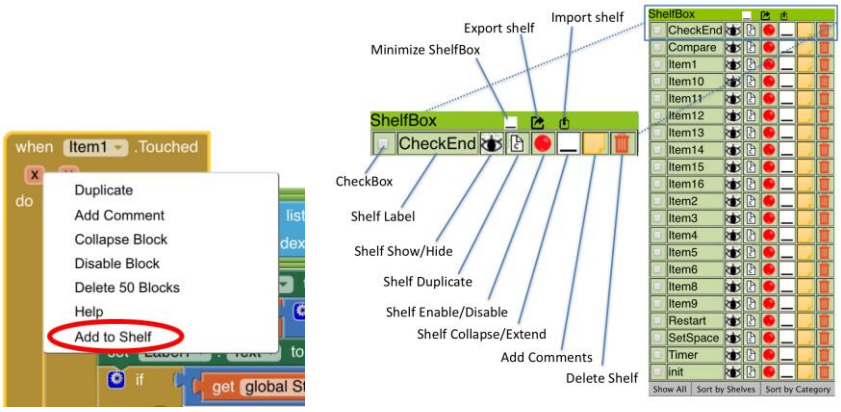
1. Design a strict coding style. This method requires users to follow certain rules to program, which improves the readability and the structure, but not block re-use, in block editors. In Pencil Code, blocks are only allowed to connect vertically, which is clear, easy, straightforward for first-time programmers to use [2].

2. Design "containers" to organize blocks. Most VPL adapt this method to design editors; Scratch [17], Stencyl [12], Gameblox [7] are well-known examples for this type of design. In such editor environment, containers can be designed according to objects (Scratch), behaviors (Stencyl), or user-customized purposes (Gameblox). Taking Scratch as an example, each sprite is a container for anchoring blocks designed for the sprite.

Introducing a conceptual structure for managing a project is indeed helpful for readability. However, the current methods don't provide extra functions for making good use of the conceptual structure; for example, users can't enable or disable all actions designed in some container once. Therefore, besides designing block shelves as containers to organize blocks, we also design a toolbox, shelfBox, for showing shelves and providing extra shelf functions to utilize the shelves defined by users.

3 Design Overview

We chose MIT App Inventor 2 (AI2), which combines the Javascript-based Google Blockly with the Google App Engine, to demonstrate our proposed tool. Accordingly, all modifications for shelf demonstration were created using Google Blockly.



(a) **(b)**

Fig. 2. (a) Menu for adding a shelf, and (b) An example of the ShelfBox toolbox. In Fig. 2b, Left: detailed interface descriptions. Right: 21 block shelves used for the posttest involving Pusheen the Cat.

As shown in Fig. 2a, adding blocks to a shelf is similar to other AI2 block-related processes. Right mouse clicks are used to add blocks to a shelf within a project. The primary functions designed for each shelf are shown on the left hand side of Fig. 2b. Added shelves are shown on the right hand side. There are seven primary shelf functions:

1. Show/hide (👁/👁‍🗨). This function allows users to hide or show blocks placed on a selected shelf. In addition to saving screen space, this feature controls the number of blocks shown in the block editor. Take Figure 1 bottom as an example: the editor would merely show the connected block group of <Restart> in the canvas, that is, the groups of <Clock2> and <to ClearStates> are hiding from the canvas, when the show button in the shelf of "Restart" is clicked.

2. Collapse/expand (🗕/🗖). This feature allows users to control the number of blocks shown in a block editor. Applying this feature, the blocks in the same shelf would collapsed into one single block. Unlike in the visibility function, users can still see each collapsed block in the editor.

3. Enable/disable (🔴/🟢). Since "breakpoint" isn't implemented in current editors, this feature is used to enable or disable all block codes on a selected shelf. Instead of enabling/disabling blocks one-by-one, this feature allows users to enable/disable entire shelves, thus making functional checking/debugging more efficient. Unlike shelf collapse/expand, this function traces codes by checking block functions.

4. Duplicate (🗎). This function duplicates all blocks on a selected shelf, which is useful when creating similar programs or testing block codes without wanting to change the originals.

5. Comment (🗒). This feature allows users to add detail to their shelf titles/descriptions as displayed on the shelfbox.

6. Delete (). This feature deletes existing shelves without affecting blocks in the editor. However, users need to be careful when using this function due to the lack of a mechanism for recovering deleted shelves.

7. Shelf export/import (/). This feature supports block sharing between projects. Users can export/import selected shelves using the extensible markup language (xml) file format. This function is useful for co-designers, who can work in separate locations and send their contributions to a central organizer.

4 Experiment Design and Procedure

To measure code reading and navigation, we used block search and reading time as measures of differences between block editors with and without block shelves [1]. We tested whether block shelves improved block code navigation, improved block code reading, and/or helped users understand project structure in terms of implementing VPL project functions. Our primary goal was not to determine whether the proposed block shelves system improved bug-fixing efficiency, but whether it improved block code understanding. For each task, participants were given full information as well as hints to help them understand task requirements. At various debugging stages we measured the time that each participant required to locate target blocks after reading a task, and reading time required to locate bugs after locating a target block. Debugging in the current block editor necessitates locating target blocks before replacing, updating, and adding blocks or values. Task requirements were analyzed as one of two types: requiring one or more changes to a single existing block, or duplicating existing block codes before changing corresponding blocks to prevent users from creating their own block codes during the experiment.

A total of 60 graduate and undergraduate students were recruited from National Chiao Tung and National Tsing Hua Universities in Taiwan (mean age 23, maximum age 38, minimum age 18). None of the study participants had ever used a VPL, but all had taken computer introduction and programming courses. Participants were randomly divided into experimental (MIT App Inventor 2 with block shelves) and control groups (original MIT App Inventor 2) of equal size.

The experiment consisted of three stages: tutorial, pre-test, and post-test. During the 40-minute tutorial stage we introduced Android front-end design and block code design apps. The tutorial project consisted of 197 blocks and 6 screens to show users how to add a gadget to the mobile phone interface, and how to use block codes to control them (e.g., how to add buttons, canvases, balls, timers, graphs, and text tags). All participants took the pretest after the tutorial course. The 250-block calculator was added to the pretest to determine whether the two groups' coding abilities were identical. To minimize the effect of imprinting, the number of blocks for the posttest project was increased by a factor of 4.2 (1,045 blocks). Pusheen the Cat, a pelmanism game that requires players to remember the locations of icons, was used to determine whether the block shelves improved user performance. All trials were conducted using the Windows 7 version of the Chrome browser, and recorded with oCam v23.0

video capture software. Meanwhile all participants were given 10 minutes to practice using their respective apps on Xiaomi Redmi 1 mobile phones.

Table 1. Pretest and posttest tasks.

Task	*Calculator (pretest)*	*Pusheen the Cat (posttest)*
		
1	**Add a division function.** Hint: Add block code for "div" by referencing "mul" or "sub."	**Add the item 7 function.** Hint: Add the "item7" block code by referencing the block codes of other items, except for "item5."
2	**Add a 9 function.** Hint: Add block code for "num9" by referencing block codes for other numbers, except for "num9" or "num5."	**Correct condition for checking if two selected icons are identical.** Hint: Find the "compare" block code and correct the condition (FirstPicture + SecondPicture ==17) in the if-statement.
3	**Correct the + function.** Hint: Find the "add" block code and modify the function by referencing the block codes of other operations.	**Correct the "cat5" icon display after selecting item5.** Hint: Find the "item5" block code and modify the display function by referencing other items.
4	**Correct the 5 function.** Hint: Find the "num5" block code and modify the function by referencing the block codes of other numbers.	**Correct the timer displayed in the bottom-right corner after hitting Restart.** Hint: Find the "Restart" block code and set the "Label2" variable to 0.
5	**Correct the = function.** Hint: Find the "equ" block code and set result.text to get the variable "global result."	**Correct the text alert for the end of a game.** Hint: Find the "CheckEnd" block code and set the "Label1" variable to "win."

All participants didn't require to start from scratch but to debug from a base of block codes in both pretest and posttest. The task descriptions used in the pretest and in the posttest are listed in Table 1 respectively. For pretest tasks 1 and 2, participants were asked to add block codes for division and the number 9 by respectively referencing existing arithmetic operations and numbers. For tasks 3, 4 and 5, they were required to locate and fix bugs in a target block. For the posttest, block shelves used by the Pusheen group were constructed prior to the experiment (shelf labels are shown in Fig. 2b). Also for tasks 1 and 3, participants were asked to add and correct block

codes by referencing other items; for tasks 2, 4 and 5 they were instructed to locate and repair a target block.

5 Results and Analysis

All search and reading times while performing calculations and Pusheen the Cat tasks were measured and listed in the table 2. Pretest search and reading time data were used to compare the programming abilities of control and experimental group participants. No significant differences between the two groups were observed for search time (control group M=207.53, SD=62.21; experimental group M=212.2, SD=49.78; t_{58}=-0.321, p=.750), reading ability (control group M=317.67, SD=17.12; experimental group M=316.83, SD=18.83; t_{58}=.033, p=.974), or completion time (control group M=525.2, SD=14.40; experimental group M=529.03, SD=20.82; t_{58}=-1.41, p=.888). Next, we used completion time differences between pretests and posttests for all participants to measure improved performance associated with block shelves. The data indicate a significant difference ($F_{1,58}$=53.757, p=.000 <.05), with mean completion time for the experimental group (M = -9.567, SD = 27.89) significantly better than for the control group (M=285.2, SD=28.95). Combined, these results suggest that the experimental group participants benefited from the block shelves feature.

Table 2. Descriptive statistics (M ± SD) for search time and reading time.

Study Group	Task1	Task2	Task3	Task4	Task5
		Calculator (Pretest)			
		Search Time (Sec.)			
Control	84.67±43.61	27.73±16.50	24.50±13.12	20.13±12.89	50.50±28.14
Experimental	71.20±38.97	33.07±15.51	31.07±14.99	22.43±13.27	54.43±34.05
		Reading Time (Sec.)			
Control	60.47±49.01	21.97±14.21	12.37±16.28	7.53±7.05	215.33±98.52
Experimental	68.93±59.73	15.87±14.78	17.77±22.33	7.87±9.89	206.40±98.10
		Pusheen the Cat (Posttest)			
		Search Time (Sec.)			
Control	61.57±33.10	124.13±59.72	44.27±26.47	55.97±39.87	64.87±51.51
Experimental	14.23±6.73	20.73±20.74	10.90±6.65	11.33±7.61	11.57±6.31
		Reading Time (Sec.)			
Control	140.93±64.80	135.30±73.73	68.07±60.63	73.57±49.13	41.73±28.99
Experimental	116.73±57.80	141.43±60.15	56.60±25.30	89.97±49.18	45.97±25.55

Search and reading time differences were also respectively used to determine whether the block shelves helped users with block code navigation and reading tasks. Our data indicate a significant improvement in block code navigation ($t_{52.33}$=14.159, p=.000<.05), with the experimental group (M=-143.43, SD = 62.64) outperforming the control group (M=143.27, SD=88.17). However, no significant improvement was found for block code reading time (experimental group, M=133.87, SD=151.02; control group M=141.93, SD=151.04; t_{58}=.207, p=.837). Reading time is associated with

the time used to locate a program bug after identifying a target block, but debugging ability is strongly correlated with a user's familiarity with block code grammar. Since none of the study participants had ever used a block editor or read block code prior to our experiment, it is understandable that no difference was noted between the two groups.

The data also confirm a decrease in VPL project readability as the number of blocks increased. As stated in the Experimental Design section of this paper, the Pusheen the Cat block number was four times greater than the number for the calculator, thus reducing the potential for imprinting. Control group participants spent significantly more time on code navigation during the posttest compared to the pretest (t_{29}=-8.9, p=.000<.05). Both control (t_{29}=-5.147, p=.000<.05) and experimental group participants (t_{29}=-4.855, p=.000<.05) used significantly more reading time during the posttest compared to the pretest. Even though project complexity increased for the posttest, experimental (block shelf) group members used significantly less search time finding the blocks they wanted to reference and/or change (t_{29}=12.542, p=.000<.05).

For the posttest, all participants had 10 minutes to use the app before dealing with the assigned tasks. That practice period, plus experience from the tutorial and pretest, meant that every participant had at least some ability using the app components and mechanisms (e.g., buttons and basic functions). For posttest task 1, participants were told to reference other items to reconstruct <item 7>. We intentionally deleted all blocks programmed for <item 7> as a means of testing whether the block shelves helped users understand the VPL project structure. Further, experimental group participants dealt with a shelfbox from which <item 7> had been deleted (Fig. 2b). Still, average search time for the experimental group participants for task 1 was 4 times faster than for the control group, suggesting that the shelves shown in the shelfbox were sufficient for the experimental group users to quickly understand what was required of them. A more detailed experiment is needed to determine just how much the experimental group participants understood, as well as to determine how much a similar control group might improve with greater time spent practicing. Our data indicate significant improvement for control group users between tasks 1 / 2 and tasks 3 / 5.

6 Conclusion

The experiment data indicates that the proposed block shelves tool[1] significantly improved project readability, that the shelfbox feature provided users with fast global views of projects, and that visible shelfbox features allowed users to select shelves for checking, which improved their understanding of block code navigation and VPL project structure. Once users understood how blocks were organized on block shelves, they could easily understand which blocks they wanted to check/add, even though the shelves in question may not have been listed in the shelfbox. The results suggest that code block understanding increases, even for large projects, when users organize block shelves with clear titles and comments.

[1] Source code can be downloaded at https://github.com/syhsu/appinventor-shelves.

References

1. Bragdon, A., Zeleznik, R., P Reiss, S., Karumuri, S., Cheung, W., Kaplan, J., Coleman, C., Adeputra, F., LaViola, Jr., J. J.: 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 (2010)
2. Bau, D., Bau, D. A., Dawson, M., Pickens, C. S.: Pencil code: block code for a text world. In Proceedings of the 14th International Conference on Interaction Design and Children (IDC '15). ACM, New York, NY, USA, 445-448 (2015)
3. Brinson, M. E., Jahn, S.: 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, 297–319 (2009)
4. Clifton, M. H.: A Technique for Making Structured Programs More Readable. SIGPLAN Not. 13, 4 , 58–63 (1978)
5. Cooper, S., Dann, W., Pausch, R.: 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 (2000)
6. Demšar, J., Curk, T., Erjavec, A., Gorup, Č., Hočevar, T., Milutinovič, M., Možina, M., Polajnar, M., Toplak, M., Starič, A., Štajdohar, M., Umek, L., Žagar, L., Žbontar, J., Žitnik M., Zupan, B.: Orange: Data Mining Toolbox in Python. Journal of Machine Learning Research 14, 2349–2353 (2013)
7. Du, E.: Gameblox Flexidor: adding flexibility to blocks based programming environments. Master Thesis. Massachusetts Institute of Technology (2015)
8. Eick, S. G., Graves, T. L., Karr, A. F., Marron, J. S., Mockus, A.: Does code decay? assessing the evidence from change management data. Software Engineering, IEEE Transactions on 27, 1, 1–12 (2001)
9. Elliott, C., Vijayakumar, V., Zink, W., Hansen, R.: National instruments LabVIEW: a programming environment for laboratory automation and measurement. Journal of the Association for Laboratory Automation 12, 1, 17–24 (2007)
10. Kersten, M., Murphy, G. C.: 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 (2006)
11. Ko, A. J., Myers, B., Coblenz, M. J., Aung, H. H.: An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. Software Engineering, IEEE Transactions on 32, 12, 971–987 (2006)
12. Liu, J., Lin, C., Wilson, J., Hemmenway, D., Hasson, E., Barnett, Z., Xu, Y.: Making games a "snap" with Stencyl: a summer computing workshop for K-12 teachers. In Proceedings of the 45th ACM technical symposium on Computer science education (SIGCSE '14). ACM, New York, NY, USA, 169-174 (2014)
13. Miara, R. J., Musselman, J. A., Navarro, J. A., Shneiderman, B.: Program Indentation and Comprehensibility. Commun. ACM 26, 11, 861–867 (1983)
14. Millner A., Baafi, E.: 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 (2011)
15. Okerlund, J., Turbak, F.: A Preliminary Analysis of App Inventor Blocks Programs. Poster presented at Visual Languages and Human Centric Computing (VLHCC), 15–19 (2013)
16. Pokress, S. C., Veiga, J. J. D.: MIT App Inventor: Enabling personal mobile ´ computing. arXiv preprint arXiv:1310.2830 (2013)

17. Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., Kafai, Y.: Scratch: programming for all. Commun. ACM 52, 11, 60–67 (2009)
18. Teasley, B. E.: The Effects of Naming Style and Expertise on Program Comprehension. Int. J. Hum.-Comput. Stud. 40, 5, 757–770. (1994)
19. Turbak, F., Sherman, M., Martin, F., Wolber, D., Pokress, S.C.: Events-first Programming in APP Inventor. J. Comput. Sci. Coll. 29, 6, 81–89 (2014)
20. Roque, R. V.: OpenBlocks: an extendable framework for graphical block programming systems. Master Thesis. Massachusetts Institute of Technology (2007)
21. 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 (IDC '15). ACM, New York, NY, USA, 199-208 (2015)