Custom Blocks in StarLogo Nova

A Template-Based Approach to Abstraction for Improved Ease of Use and Expressive Power

Hyeonsu Kang
Scheller Teacher Education Program
Massachusetts Institute of Technology
Cambridge, MA USA
hyeonsu@mit.edu

David Wu
Dept. of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA USA
dwu@mit.edu

Daniel Wendel
Scheller Teacher Education Program
Massachusetts Institute of Technology
Cambridge, MA USA
djwendel@mit.edu

ABSTRACT

Many blocks-based programming environments allow the creation of custom blocks, either as the primary means of creating procedural abstractions, or in addition to a procedure/function definition feature. Custom blocks in these systems are intended to be used like procedures, and creating a custom block has similar conceptual challenges; abstracting away local context, and choosing and using parameters, can be difficult for novice programmers. Additionally, we have identified certain types of custom blocks, dealing with scope and context, that cannot be created using a procedure-based abstraction. To address the difficulties inherent to the processes of abstraction and parameterization, and to expand the expressive power of custom blocks for the creation of domain-specific language extensions, we have designed a method of creating custom blocks that is based on a "template" or "macro" metaphor rather than a procedure metaphor. We present this design for consideration and discussion.

KEYWORDS

Custom blocks, Blocks-based programming, Abstraction, Computer Science education

Introduction

Many blocks-based programming environments provide mechanisms for creating custom blocks, either as a metaphor for procedural abstractions (e.g. Scratch and Snap!) or as a way to create entire domain-specific languages (e.g. ViMaP). Alternatively or additionally, some environments provide a mechanism for working with procedures more explicitly, as with StarLogo Nova's "procedure" blocks, App Inventor's "to ___, do" blocks, or Snap!'s "ring" operator.

While mechanisms for creating procedural abstractions are common, there seem to be two separate motivating cases behind the designs. The first use case, which motivates Snap! and similar tools, is of users learning about and creating their own abstractions as they grow in their programming ability. In this case, users develop their own abstractions out of language primitives, and then use those new abstractions to build successive layers.

The second use case comes from the opposite direction: in tools like ViMaP, the primary intended use is by expert curriculum designers creating the abstraction layers for students, as a way of more closely aligning the programming language to other content (e.g. science or math class). The underlying layers of abstraction may then be "peeled back" progressively, eventually arriving at primitives as the students grow in their computational thinking skill.

Regardless of the motivating use case, the approaches above share a similar requirement that the designers of the procedural abstractions think about parameterization from a "top-down" perspective. That is, the designer must first name or otherwise specify the parameters, and then incorporate those parameters into the "body" or implementation of that procedure. However, in a paper studying the infrequent use of procedures in App Inventor projects, it was hypothesized that this degree of pre-planning or "top-down" design could be one of the factors contributing to the limited use of procedures[1].

StarLogo Nova aims to enable both of the use cases mentioned above with its design for custom blocks, but attempts to alleviate some of the difficulty associated with working with parameterized procedures by using a different metaphor. StarLogo Nova's approach begins with the implementation of the block, and does not require parameters to be predetermined or treated as abstract entities.

Template-based metaphor

To the best of our knowledge, the other well-known blocks-based programming environments for education use a procedure metaphor for custom blocks. Or, conversely, they might say that their custom blocks are a metaphor for procedures. The conceptual benefits of this approach are obvious as a stepping-stone toward procedural abstractions. However, procedural abstractions have some properties that are not immediately intuitive to new programmers, and may lead to less-obvious drawbacks of the procedural approach to custom blocks.

One such subtle and confusing property of procedures is that a procedure body is (typically, in languages where context is not explicitly passed) evaluated in the context of its declaration rather than the context of its call. For example, many programming languages allow procedures to refer to variables declared outside of the procedure, if those variables are in scope where the procedure is defined. However, a procedure that refers to a variable that is only in scope where the procedure will be called will not compile in many languages, and will not execute in other languages where it might compile, because a procedure's body is bound to the declaration's scope rather than to the calling scope.

The implication of this property is: if a user has a correctly-executing program, they can remove a bit of code from that program, place it inside a procedure, then call that procedure from the original location of that code, and the program will fail! The explanation of a "procedure" as a "named set of instructions to do something" breaks down at this point, and suddenly the user is thrown into a CS-laden discussion about context, frames, and scope.

StarLogo Nova has tentatively taken a different approach, which is currently available to test on our beta server (beta.slnova.org). In our approach, custom blocks are treated as "templates", essentially C-style parameterized macros, rather than procedures. Users can think of them as being drop-in replacements for the code that they represent. So, whereas the example above might fail with the procedural abstraction, it works fine with the template/macro abstraction. In fact, certain operations, such as declaring a variable, cannot be implemented as custom blocks with a procedural abstraction even if blocks themselves are first-class data types, because they are executed in the procedure's frame. However, with a template/macro metaphor, a custom block could declare a variable for use by following blocks without any trouble. This enables us to maintain an important invariant: if a user creates a custom block by encapsulating a section of code, and uses the block in place of that original code, program execution will not be affected.

Bottom-up parameterization

Another of StarLogo Nova's departures from other tools lies in the way users parameterize the custom blocks. In other tools, the user adds parameters (or, "sockets") to the custom block first, and then incorporates those parameters into the body of the procedure, typically through the use of "parameter getter" blocks. This approach is somewhat "top-down", as the parameters must be first declared, and then incorporated into the body.

StarLogo Nova, however, takes more of a "bottom up" approach: the user first writes the body, then selects which of the parameters of the body's blocks should be promoted the label of the resulting custom block, and, finally, determines the wording and layout of the block. While this limits the use of each parameter to one location in the body, it provides some benefits:

1. Parameters do not need to be named or otherwise referred to as their own abstractions.
2. Default values can be specified merely by inserting a value into that parameter's location in the body. This can include inserting blocks, causing the custom block to come with pre-connected default blocks.
3. Promoting existing block arguments from the implementation body to the custom block label allows the use of StarLogo Nova's context-aware menu arguments (see Fig. 1) on custom blocks, which, because custom blocks depend on the context of the call location rather than the declaration, means that users can create custom blocks that show different options in menu arguments depending on the context in which they are used.

This work is still in the early stages, but Figure 1 shows a first attempt at a UI for choosing which arguments from the underlying implementation to promote to the label of the custom block.

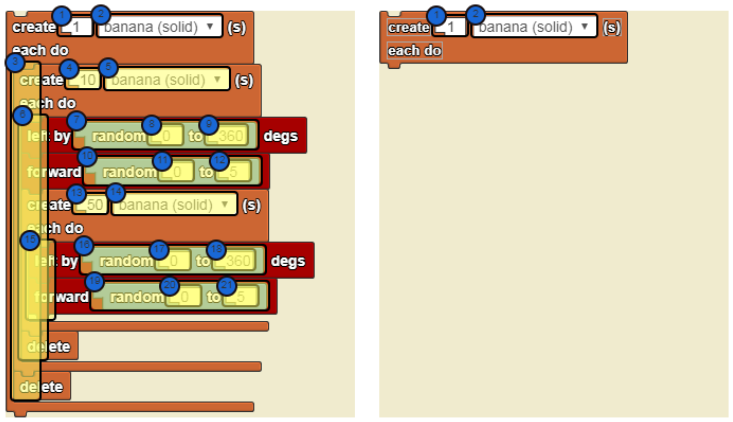


Figure 1. Users create a parameterized custom block by choosing block arguments to promote to the custom block, dragging them from the implementation on the left and dropping them onto the custom block on the right.

Conclusion

Despite the benefits of the prevalent design of custom blocks, the top-down parameterization and procedure-style scoping rules may cause confusion and limit the expressive power of custom blocks. We have explored a different approach for StarLogo Nova that aims to address these potential issues, and we hope to discuss the implications of this approach with the BLOCKS+ community.

REFERENCES

[1] I. Li, F. Turbak and E. Mustafaraj, "Calls of the wild: Exploring procedural abstraction in app inventor," 2017 IEEE Blocks and Beyond Workshop (B&B), Raleigh, NC, 2017, pp. 79-86. doi: 10.1109/BLOCKS.2017.8120417.
[2] Sengupta, P. & Wright, M, 2010, ViMaP: Virtual Multi-Agent-based Programming Language, [Computer Software], Vanderbilt University
[3] Scratch reference
[4] Snap! Reference
[5] App Inventor reference