Integrating Droplet into Applab – Improving The Usability of a Blocks-Based Text Editor

David Anthony Bau
Phillips Exeter Academy
Exeter, New Hampshire 03833
Email: dbau@exeter.edu

***Abstract*—Droplet is a new programming editor that allows dual-mode editing in blocks and text for any text program. This paper presents observations and improvements to Droplet based on integrating Droplet into Applab, Code.org's JavaScript sandbox learning environment. Droplet's unique interactions with both text and blocks create several unusual problems and opportunities for improvement.**

I. Introduction

This paper describes the development of a series of usability improvements for a Droplet block editor. Droplet provides a visual editing mode similar to Scratch [8], Alice [**?**], and Blockly [9]. However, Droplet is unique because it works as a text editor, and provides a block interface on top of parsed text code. Initially, Droplet provided the following features:

- Blocks based on parsed text, allowing lossless conversion between blocks and text.
- A palette of short prewritten code fragments, represented as blocks.
- An editor supporting drag-and-drop assembly and editing of the blocks in a program.

Because Droplet is a text editor, many features of other block languages were initially unimplemented. For example, Weintrop [4] found that a key benefit of block languages is that the two-dimensional surface allows bottom-up assembly of code. However, because of the linear nature of text code, the first version of Droplet did not support it.

In this work, the author recognized four major usability principles:

1) Two dimensional code assembly, as proposed by Weintrop [4]. The challenge is to relate this to the underlying text code.
2) User control and freedom through undo. In the context of block editing, two forms of undo are important. Both will be described here.
3) Recognition versus recall. Although the block palette is an aid for remembering commands, an aid is needed for remembering text socket values, in a way that is compatible with text coding.
4) Error messages and warnings. Droplet can edit any text program and therefore can create programs with errors. To help users with errors, Droplet can borrow ideas from traditional text programming editors.

The work was done in the context of integrating Droplet's Javascript mode into Code.org's Applab environment.

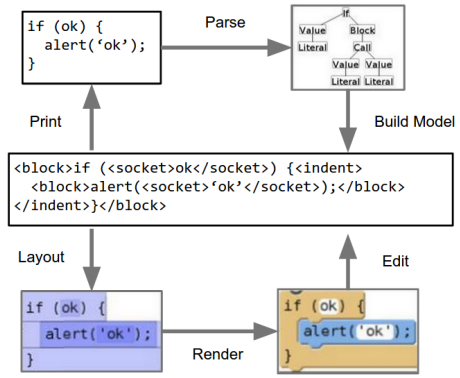

Fig. 1. Droplet's Lifecycle for a JavaScript Program

II. Background

A Droplet document is a text file which Droplet annotates with markup for editing purposes. Figure 1 shows the lifecycle of a Droplet program in JavaScript. When the user opens a file, the language adapter for JavaScript runs the code through a standard JavaScript parser. It uses the resulting syntax tree to annotate the text stream with tokens like "blockStart" or "socketEnd" to indicate where block entities should be rendered. The adapter also attaches metadata to the blocks about color, shape, and droppability rules. Droplet then lays out and renders the resulting stream. When the user saves or runs the file, the markup is discarded to recover text code.

Droplet's layout algorithm always places text nodes in the same rows as the text appeared in the source. This allows Droplet to achieve a smooth animation between blocks and text.

III. Relating Floating Stacks to Text Code

A study by Weintrop [4] found that Scratch's two-dimensional editing surface was beneficial to students. It allowed students to try out different ways of performing the same task, and to compose programs in a "non-linear" way. Maloney et al. [10] refer to this as "tinkerability," and say that it supports "a bottom-up approach to writing scripts where small chunks of code are assembled and tested, then combined into larger units." This layout, however, is unfriendly to the linear nature of text code. Droplet needed a way to support assembling and viewing code side-by-side while retaining a one-to-one relationship to text code.

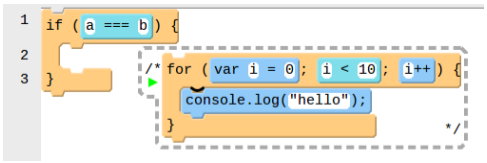

Fig. 2. An Example of Droplet's Floating Block Graphics

Both Scratch and Blockly support floating blocks. Blockly runs all floating code in top-left to bottom-right order, while each Scratch block stack is associated with an event handler and runs whenever the attached event is fired. Scratch also runs a stack when it is double-clicked.

Droplet now allows floating blocks to be assembled, but surrounds them with a dotted line and the language's block comment symbol to indicate that they are not going to be run (fig. 2). The surrounding dotted line can be grabbed to move the entire stack or replace it into the main document to "uncomment" it. This allows students to assemble blocks in a nonlinear order, and borrows metaphors from text to make it clear that the stacks will not be run. Dropet can also display a "play" button (as seen in figure 2) and emit an event to support running individual stacks.

In the future, Droplet may represent these blocks in the code by inserting them as comments. This would allow Droplet show an animation between the floating stacks in text mode and in block mode.

IV. User Control and Freedom

Especially in untyped languages like JavaScript, it is easy to accidentally drop Droplet blocks into the wrong socket. However, because Droplet is text-based, sockets often contain important information like long strings. Other major block languages do not have this problem because sockets with information like variable names or strings are not usually drop targets for other blocks.

Neither Scratch nor Blockly implements a full undo stack. Blockly may not need it because most constants that could be long, sensitive work are not also drop targets for other blocks. In Blockly, typing socket, which is round, is different from a droppable socket, which looks like a puzzle piece, so work is not lost when an accidental drop occurs. In Scratch, sockets that contain strings are also not drop targets. Scratch supports one level of "undelete" but not undo.

A. Remembering Old Socket Values

Droplet now remembers the old values of a socket when a block is dropped into it, so that when the block is dragged back out the socket is repopulated with the old value (fig. 3). This allows students to correct an accidental drop by simply continuing to move the block to its intended location.

Because Droplet frequently reparses blocks, attaching the remembered value data directly to the socket is not possible. Instead, Droplet maintains a map from socket locations to remembered values. However, the method by which locations should be serialized is unclear. The structure of a Droplet document can drastically change when Droplet reparses blocks, but the text of the Droplet document remains unchanged. This suggests that a locations should be based on text offsets.

B. Full Support for Undo and Redo

Undo stacks are important to usability, and are included in Neilsen's widely-recognized user interface heuristics [7]. However, Droplet faced two obstacles in implementing full undo stacks for Droplet. First, because of Droplet's text-based affordances, Droplet had a large number of types of mutations, which were difficult to track and maintain. Second, Droplet did not have a way to unambiguously serialize the locations at which operations were happening.

Droplet mutations can be reduced to combinations of inserts and deletes. Locations, however, present a difficulty. A text-based location, like the remembered socket mechanism might use, is ambiguous when blocks or sockets are adjacent without intervening text. Because the undo stack would track reparses, the structure of the document when the location is retrieved is would be identical to that when it was serialized. This suggest that locations should be based on token offsets.

C. Resolving the Location Dilemma

To permit both socket value restoration and a full undo stack, Droplet uses two location models and converts between them as necessary. To assist this, Droplet has a third type of fundamental mutation: replace. A replace operation is used only for reparsing, and requires that the text content of the replaced section does not change. Droplet stores all locations as token-based offsets. When a replace operation occurs, Droplet converts any locations inside the replaced section that need to be persisted to text-based locations and converts back afterward. This allows Droplet to preserve socket locations across most reparses, but for the primary location model to be unambiguous.

V. Error Prevention and Recovery

A. Breakpoints and Line Annotations

Applab had existing support for live errors and warnings and debugging breakpoints in text mode. Because Droplet blocks have a one-to-one relationship with text code, adding breakpoint and live line-annotation support to Droplet could easily take advantage of Applab's existing debugging infrastructure. Line breakpoints and live annotations are a part of most major professional development environments, including Applab's text mode and Eclipse [3]. A study by Murphy [6] found that over 70% of Eclipse users use breakpoints. In 1986 Baecker [5] proposed "Metatext" or annotations as one of the five main principles of program visualization.

This is a unique opportunity for Droplet, because it has a one-to-one relationship to text code. Neither Scratch nor Blockly have support for line annotations or line breakpoints.

Droplet now supports breakpoints and annotations in the gutter the same way major text editors do (fig. 4). Droplet mimics Ace editor's API to allow Applab and other embedders to easily convert their existing debugging infrastructure from Ace editor to Droplet.

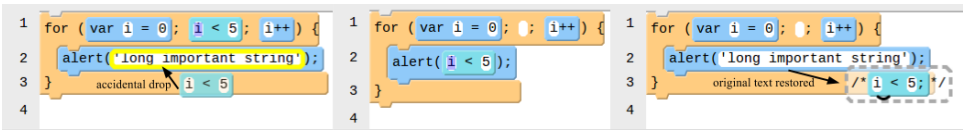

Fig. 3. An Example of Droplet Restoring Old Socket Values

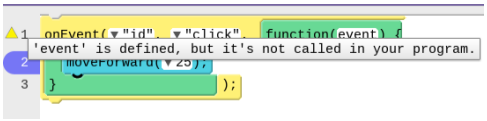

Fig. 4. An Example of Droplet Gutter Decorations in Applab

Fig. 5. Droplet's New Behavior on Syntax Errors

B. Handling Syntax Errors

Droplet allows users to type free-form text into sockets, which it will reparse on-the-fly and turn into blocks. This helps give students the experience of writing text without switching fully to text mode. However, it also means that users can create syntax errors by typing into sockets, unlike in other major block languages. Scratch and Blockly will only allow valid inputs in text areas. Droplet will now outline the violating input when a syntax error is created, and supports error annotations to help users identify the error (fig. 5).

VI. Recognition Rather than Recall: Dropdowns in a Text-Based Editor

Because all Droplet blocks are generated from text, Droplet did not have good support for dropdowns from sockets, which other major block languages do. Dropdowns, like autocomplete in text code, help students remember what parameters are valid, in accordinance with Neilsen's heuristic of recognition vs. recall [7].

Both Scratch and Blockly implement dropdowns for their text inputs. Both have special selectors for colors, allowing users to use a color picker or to "eyedrop" existing pixels on the screen.

Droplet added new configuration to allow the embedding application layer to specify dropdowns. Embedders may specify dropdowns by function name and argument position in JavaScript and CoffeeScript mode – for instance, in Figure 6, the "fd" function has a dropdown specified at argument 0. Dropdowns can be dynamically generated – in Figure 6, a list of element ids is generated using information taken from Applab's WYSIWYG HTML Design Mode.

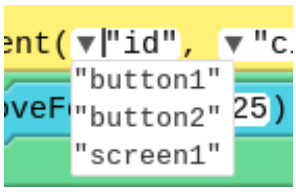

Fig. 6. An Example of Droplet Dropdowns in Applab

VII. Acknowledgements

The author would like to thank Code.org for their support of this work, and Sarah Filman at Code.org and David Bau at Pencilcode for their advice.

References

[1] Bau, D. A. Droplet, A Blocks-Based Editor for Text Code. Journal of Computer Science in Colleges. 30, 6 (June 2015).

[2] Code.org. http://code.org

[3] Mars Eclipse. http://eclipse.org

[4] Weintrop, D. and Wilensky, U. To Block or Not To Block, That is the Question: Students' Perceptions of Block-based Programming. IDC '15 proceedings (June 2015).

[5] Baecker, R. and Marcus, A. Design Principles for the Enhanced Presentation of Computer Program Source Text. CHI '86 proceedings (April 1986).

[6] Murphy, G. Kersten, M. and Findlater, L. How Are Java Software Developers Using the Eclipse IDE? IEEE Software (July/August 2006) 72-82.

[7] Nielsen, J. (1994). Heuristic evaluation. In Nielsen, J., and Mack, R.L. (Eds.), Usability Inspection Methods, John Wiley & Sons, New York, NY

[8] Scratch. https://scratch.mit.edu/

[9] Blockly. https://blockly-games.appspot.com/

[10] Maloney, J., Resnick, M., Rusk, N., Silverman, B., and Eastmond, E. 2010. The scratch programming language and environment. ACM Trans. Comput. Educ. 10, 4, Article 16 (November 2010), 15 pages. DOI = 10.1145/1868358.1868363. http://doi.acm.org/10.1145/1868358.1868363.