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 new block editor, Droplet. Droplet follows in the tradition of block languages such as Scratch, Alice, and Blockly, providing users with a visual view of a program that can be created by assembling blocks.

Droplet takes the unique approach of staring as a text editor, and then providing an additional block interface on top of parsed text code. In its initial development, Droplet provided the following features:

- Automatic parsing to allow users to convert freely between text and blocks.
- A palette of short prewritten code fragments, represented as blocks.
- A editor supporting drag-and-drop assembly and editing of the blocks in single program file.

Although Droplet resembles other block editors, at its core, it is a text editor. Underneath the drag-and-drop gesture, it was oriented towards freeform text entry of a linear text file. Many features of other block languages were not supported.

What features are needed for a block language to be usable? 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 its text-oriented nature, the first version of Droplet did not support it.

In this work, the goal of the author has been to carefully consider usability principles in the context of block editors, and to find ways to implement important usability improvements in Droplet.

The principles we recognized and considered were:

1) Two dimensional code assembly, as proposed by Weintrop [4]. The challenge is to relate this to the underlying linear text code.
2) User control and freedom through undo. In the context of block editing, two forms of undo are important. Both were implemented and are 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, it is helpful to borrow ideas from traditional text programming editors.

In this paper the author describes solutions for these four usability issues. The work was done in the context of integrating Droplet's Javascript mode into Code.org's Applab environment. This paper also compares Droplet's approach to the approaches taken by Scratch and Blockly.

II. Background

Droplet is an editor that display text code as visual blocks, and can toggle between the two. A Droplet document is simply 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 is invoked and 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 simply discarded and, if no edits have occurred, the original text stream is guaranteed to be generated again.

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

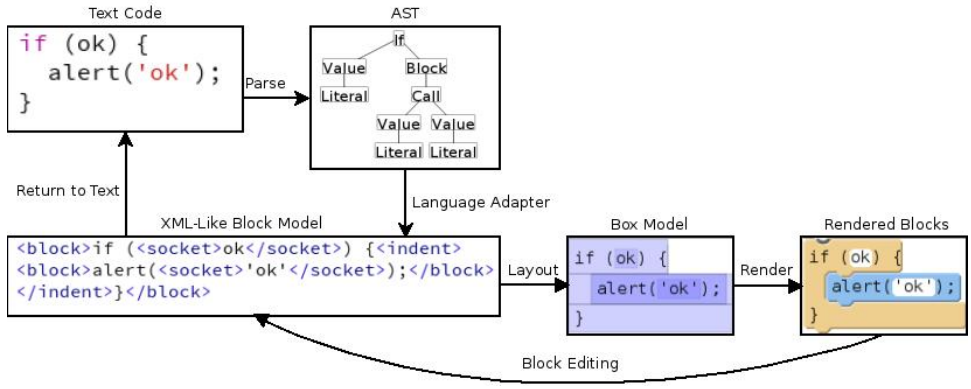

Fig. 1. Droplet's Lifecycle for a JavaScript Program

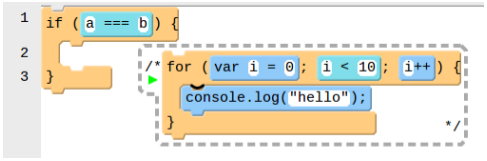

Fig. 2. An Example of Droplet's Floating Block Graphics

assembling and viewing code side-by-side while retaining a one-to-one relationship to text code.

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.

Blockly does not have this problem, 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.

Scratch has this problem, but it is less pronounced, since sockets that contain strings are not drop targets. Scratch repopulates sockets with hand-chosen default values when blocks are dragged out of them.

Neither Scratch nor Blockly implements a full undo stack. Blockly has progress on an undo stack based on the serialized operations it will make for real-time collaboration. Scratch does not have an undo stack, but has support for "undeleting" the last block that was deleted.

A. Full Support for Undo and Redo

Undo stacks are important to usability, and are included in Neilsen's widely-recognized user interface heuristics [7]. Undo was difficult to implement in Droplet because of the way Droplet reparses and replaces blocks whenever they are changed, invalidating pointers to blocks. Droplet needed a clean model for serializing and maintaining the locations of blocks in the document.

Droplet now supports two different locations models: a token-based locations model identifying blocks by their token offset from the beginning of the document, and a text-based locations model identifying blocks by their character offset, type, and string length. The text-based locations model is more human-readable but is slightly ambiguous. Editing entities like the cursor, focused socket for text input, and selection are stored as token-based locations.

Droplet now also confines mutations to three fundamental operations: insert, delete, and replace (used for reparsing). When an insert or delete occurs, Droplet preserves a location by retrieving a pointer to the linked-list token at the location, doing the mutation, and retrieving the new location of the block. When a replace occurs, locations inside the replaced area are converted to text locations, then converted back afterward. Each of mutation operation now generates an undo operation described by token-based locations.

B. Remembering Old Socket Values

Droplet now also remembers the old values of a socket when a block is dropped into it, so that when the block is

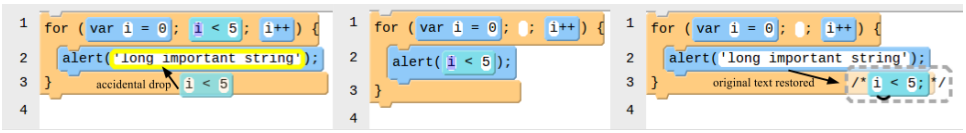

Fig. 3. An Example of Droplet Restoring Old Socket Values

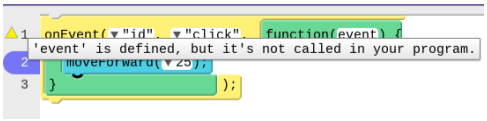

Fig. 4. An Example of Droplet Gutter Decorations in Applab

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

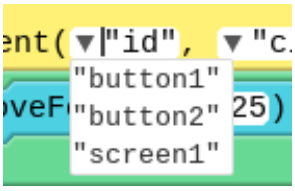

Fig. 6. An Example of Droplet Dropdowns in Applab

dragged back out the socket is repopulated with the old value (fig. 3). This provides an alternate way for students to correct an accidental drop by moving 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. By preserving the location of the socket (using the same infrastructure as the new undo stack), the Droplet editor can track the socket and preserve the remembered value this until the block is deleted. The map from sockets to remembered values is also tracked in the undo stack, so the remembered values are preserved across undo and redo.

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.

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.

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.