DROPLET, A BLOCKS-BASED EDITOR FOR TEXT CODE

David Anthony Bau
Phillips Exeter Academy
20 Main Street, Exeter, NH
781-795-2906
dab1998@gmail.com

ABSTRACT

Droplet is a new programming editor, created by the author, which has a drag-and-drop block interface like Scratch, but allows users to edit in mainstream text programming languages. It is intended to close the learning gap between blocks and text. Droplet accomplishes two-way conversion by using the program text in the target text language as the primary data model. A preliminary analysis of data from several months of student use of Droplet is also presented.

1. INTRODUCTION

Drag-and-drop block languages such as Scratch and Blockly are accessible to learners: Scratch has over four million young users in its online community [1], and Blockly has been used by over 20 million learners through code.org. [2] However, these tools do not enable direct work with mainstream text programming languages. This prevents their users from interacting with the large community of practice using text languages such as C, Java, Python, and JavaScript [3], which may lead to a confidence gap. For example, Lewis found that beginners introduced to text code first (rather than blocks) agree more strongly with the statement that "I am good at writing computer programs" [4]. Powers et al. report that weaker students moving from blocks to text "lost confidence" and even strong students reported "syntax overload" [5].

2. RELATED WORK

Several other tools have attempted to address the gap between blocks and text.

Alice [6] is a block programming language built around 3-D storytelling. Alice projects can be converted into readable Java code, but not vice-versa.

Scratch [7] allows advanced users to create new blocks using JavaScript. However, existing Scratch programs cannot be turned into well-formatted JavaScript programs, nor vice versa. Scratch is also related to the text language Sniff [8], which is designed to carry over similar vocabulary and thought processes, although there is no automatic conversion between Sniff text and Scratch blocks.

Blockly [9] is a programming tool that lets students work in blocks. It can show a text language equivalent, but no work is done in text. Blockly-based AppInventor [10] is a block language for mobile devices that includes a new text language, TAIL [11] (Fig 1). TAIL is designed to be

isomorphic to AppInventor blocks and can be converted to blocks. This approach relies on a new text language, so it does not directly generalize to other pre-existing text languages.

Tiled Grace [12] (Fig 2) is a block-based editor for the nascent programming language Grace [13]. It is notable because it allows students to work both in a pre-existing text language, Grace, and blocks. Tiled Grace supports the syntax, semantics, and libraries of Grace, but it does not support other languages.

Fig 1. An example of TAIL conversion with App Inventor

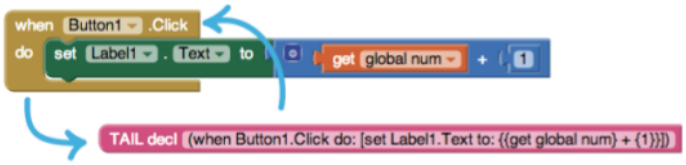


Fig 2. An example of the Tiled Grace text/blocks transition

3. APPROACH

Droplet was created as a block editor for Pencil Code [14], a learning tool oriented around Coffeescript and Javascript. Hence, Droplet loads existing text programs, edits them as blocks, and saves them as text again, with 100% fidelity to the text when switching modes.

Droplet currently edits Javascript and Coffeescript, and is easily extended to other languages by following two principles

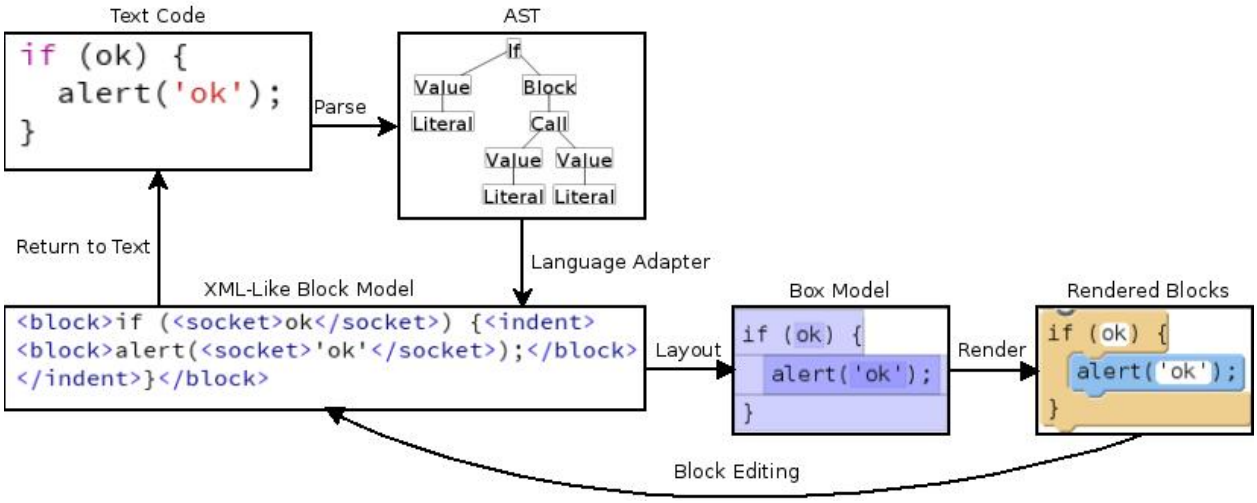
1. The text is the core piece of data, even when editing blocks. The data model used for rendering and editing is a marked-up text stream.
2. Language-specific logic is minimal. All blocks are treated the same.

3.1. Text-based Data Model

Droplet's data model is a text stream marked up with XML-like tokens such as **<block>**, **</block>**, **<socket>** or **</socket>**, where each token represents the boundary of a visual block or a drop-region within a block. This model is created from a text program by adapting a traditional language parser to insert tokens where blocks are desired (Fig 3). To convert from blocks to text, the markup is discarded, returning to the original text.

The Droplet renderer and editor treat all blocks in the same manner. Block color and decorations are determined by the language adapter, and block shape is determined by the format of the text. Drag-and-drop operations splice the text.

Fig 3. An overview of Droplet architecture



3.2. Language Extensibility

Extending Droplet to another language requires only writing a new language adapter. Language adapters insert markup into the text stream, along with metadata such as color. Adapters currently exist for Javascript (acorn.js parser) and CoffeeScript (native CoffeeScript parser).

To do type-checking or deal with order of operations, the language adapter receives callbacks. Before a block is dropped into a new place, the adapter may prohibit the drop if it is type-incompatible. To preserve order of operations, it may also mutate the block, usually to add or remove parentheses.

3.3. Rendering

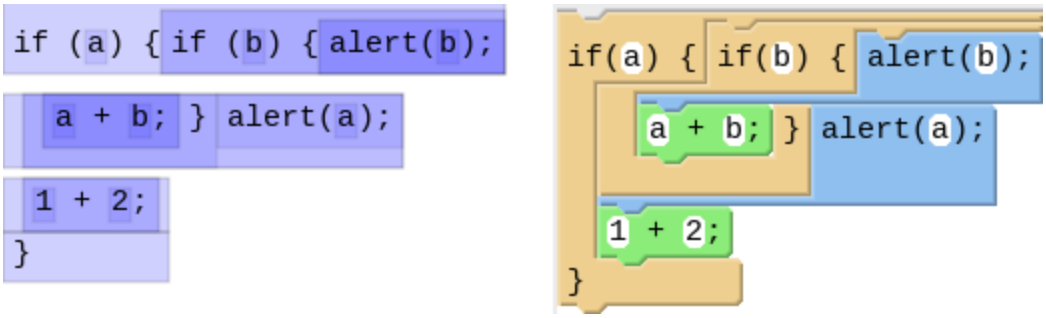
To render blocks, Droplet considers the text line by line. Each Block computes a box for each line that surrounds all of its text on that line. Spacing is sometimes inserted between lines in the box model (Fig 4, left) to ensure that, when paths are drawn, each block has enough space to be connected (Fig 4, right).

Droplet then draws a path surrounding all of its rectangles, but avoiding indented areas. For normal indentation, this avoidance creates a C-shaped mouth similar to Scratch's container blocks (Fig 2). However, it can also create blocks of other shapes to accommodate text in other formats (Fig 4).

Because all text is rendered in the blocks, Droplet also shows smooth animation between blocks and text (Fig 5). Block colors are faded away first, then text characters are linearly animated to their final positions.

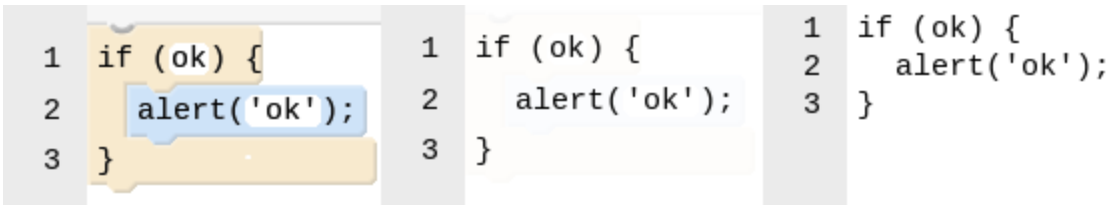
Droplet is designed to edit any Javascript or Coffeescript program. It is confirmed to process and preserve a range of real-world code including the source for jQuery, Coffeescript's unit test suite of 3,000 syntax edge cases, and the source for the Coffeescript compiler itself. Every markup stream that represents a valid tree is guaranteed to render. Thus, every language that can have an AST could be supported by Droplet.

Fig 4. An example of Droplet rendering unusually-formatted text.
Left: box model; right: final rendering.



3.4. Animation Between Blocks and Text

Fig 5. An example of the Droplet animation.



4. USER EXPERIENCE

Between 2014-9-9 and 2014-11-1, the block-text transition was used in the Droplet editor over 81,000 times on Pencil Code. We recorded the number of programs with syntax errors and the number of programs that used indented constructs like loops or functions.

Students using Droplet had almost 4 times fewer syntax errors than students using text, but used constructs such as loops and functions just as often (Fig 6). Students, in general, preferred block mode, but approximately 26% used an even combination of the two modes (Fig 7).

As users stay longer on pencilcode, they smoothly become more likely to use text mode (Fig 8). Droplet assists in this transition. However, this may also be affected by the fact that the less motivated users, who are dependent on block mode, quit pencilcode early.

Figs 6 and 7. Droplet usage statistics on pencilcode.net.
Left: errors and control flow in blocks vs text. Right: number of users who prefer each mode.

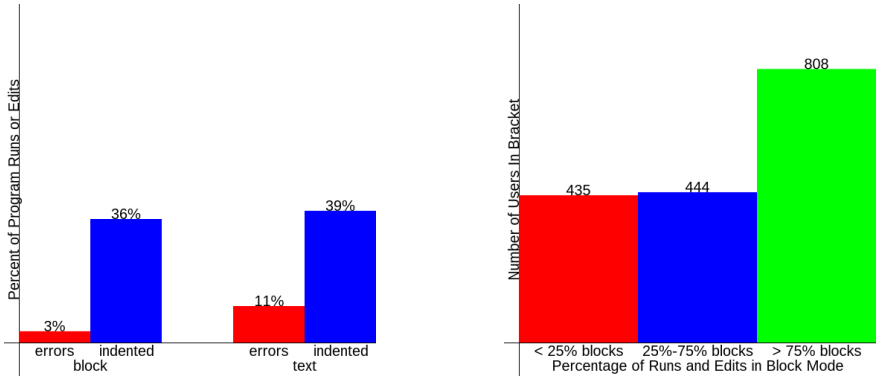
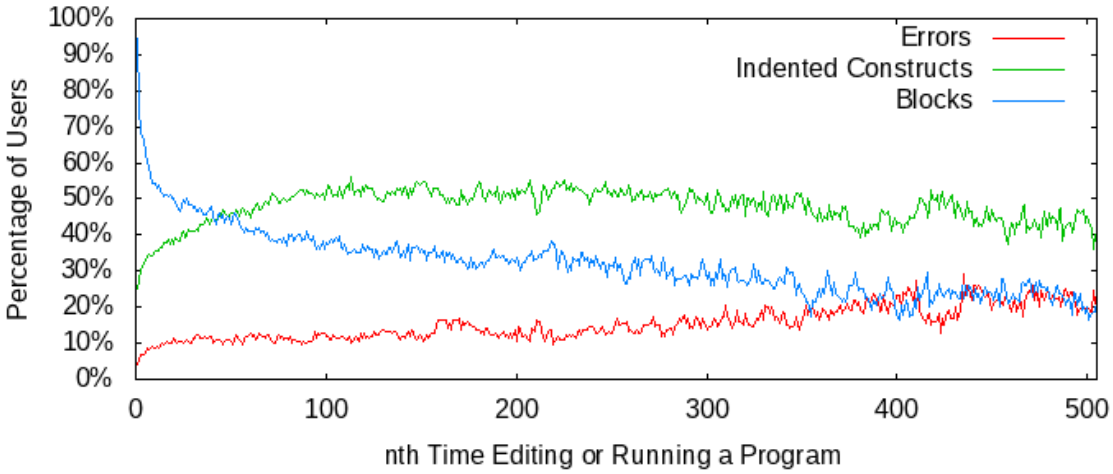


Fig 8. The average Pencilcode user's usage progression



We conducted a survey of fourteen high schoolers (many with prior experience) after a one-hour session, users said that they used blocks for the following purposes:

At the beginning, when I was getting started	7
To learn new constructs, like “for” and “tick”	2
To discover new commands in the palette	5
To write most of my programs	0

Thirteen out of the fourteen users responded that they used “mostly text” to program, and one “both equally.”

We also conducted a separate survey of five high schoolers with no prior experience after a two-hour session. The distribution was more even: two used mostly blocks, one mostly text, one started in text and moved to blocks, and one started in blocks and moved to text.

5. DISCUSSION

As far as we know, Droplet is the first block editor that is able to edit programs in a language as widely used as JavaScript. Future work includes integrating with a parser generator like ANTLR, which would support over 100 languages, including Java and C.

By giving students the ability to switch freely between blocks and text, Droplet opens new possibilities for teaching. Students do switch modes often, and in future work we are interested in studying whether this helps to build confidence and understanding.

REFERENCES

[1] Scratch Statistics. http://scratch.mit.edu/statistics/, retrieved October 26, 2014.
[2] Every student in every school should have the opportunity to learn computer science. http://code.org/files/Code.orgOverview.pdf, retrieved October 26, 2014.
[3] TIOBE Index for October 2014. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html, retrieved October 26, 2014.
[4] Lewis C M. 2010. How programming environment shapes perception, learning and goals: logo vs. scratch. In *Proceedings of the 41st ACM technical symposium on Computer science education* (SIGCSE '10). ACM, New York, NY, USA, 346-350. DOI=10.1145/1734263.1734383 http://doi.acm.org/10.1145/1734263.1734383
[5] Powers K, Ecott S, Hirshfield L M. 2007. Through the looking glass: teaching CS0 with Alice.*SIGCSE Bull.* 39, 1 (March 2007), 213-217.
[6] Cooper S, Dann W, Pausch R. 2000. Alice: a 3-D tool for introductory programming concepts. In *Proceedings of the fifth annual CCSC northeastern conference on The journal of computing in small colleges* (CCSC '00), John G. Meinke (Ed.). 107-116.
[7] John M, Mitchel R, Natalie R, Brian S, and Evelyn E. 2010. The Scratch Programming Language and Environment. Trans. Comput. Educ. 10, 4, Article 16 (November 2010),
[8] Stephenseon I. Scratch and Sniff. ICT in Practice Blog. http://www.ictinpractice.com/scratch-and-sniff-by-ian-stephenson/, retrieved October 26, 2014.
[9] Fraser N, Blockly -- Google Developers. https://developers.google.com/blockly/, retrieved October 26, 2014.
[10] MIT App Inventor. http://appinventor.mit.edu/explore/, retrieved October 26, 2014.
[11] Chadha K. Improving the Usability of App Inventor through Conversion between Blocks and Text. http://cs.wellesley.edu/~tinkerblocks/chadha-thesis.pdf, retrieved October 26, 2014.
[12] Homer M. Combining Tiled and Textual Views of Code. in 2nd IEEE Working Conference on Software Visualization (Victoria, CA, 2014), IEEE, 1-4.
[13] The Grace Programming Language. http://w3.cs.jmu.edu/arch/grace/doc/, retrieved October 26, 2014.
[14] Bau D, Bau A. A Preview of Pencil Code. A Preview of Pencil Code. Proceedings of the 2014 ACM Conference on Systems, Programming and Applications. (In press.)