Block-based languages for professionals

Robert Holwerda
Delft University of Technology, The Netherlands
r.n.a.holwerda@tudelft.nl

Abstract
Block-based languages should no longer be regarded solely as a stepping stone to text-based languages. In fact, the design of block-based languages should evolve towards a programming-UI that (adult) professionals find productive and pleasant to use, even on a day-to-day basis. Current block-based languages are not there yet.

We intend to research the design-space for block-based language interfaces with these new users and usages in mind, focussing on web designers. An initial user-study, exposing design students to a block-based version of a programming language they already know, kicks off this research.

1. Introduction
Block-based programming languages such as Scratch, E-toys, Alice and various versions of Blockly have been used successfully to introduce children to programming (ref,ref,ref). Among the aspects that contribute to this success are:

1) A *playful domain* that is fun and motivating, such as simple games (Scratch, E-toys), interactive multimedia (Scratch), puzzles featuring characters from cartoons and games (Hour of Code version of Blockly), simple robotics (other versions of Blockly), and mobile phone applications (MIT AppInventor)
2) The *programming language is kept simple* and comprehensible by limiting the set of commands, using an intuitive semantics, and avoiding or simplifying complex programming concepts (such as object orientation, static typing or data structures more complex than lists). Alice is an exception here, but it is geared towards older children.
3) A UI-design that allows the *direct structural manipulation of program-components,* which is both intuitive (drag-and-drop) and eliminates the need to learn and edit syntax that delineates structure, such as bracket, braces, comma's, (semi) colons etc.
4) The UI makes the *language highly discoverable*: A palette presents building-blocks (commands, data-literals and control structures) visually, and individual blocks display labelled edit fields and menus for their parameters and visual indicators for how to connect the block to others. Users can rely on recognition instead of active recall of available components, and explore available options while solving problems inside the programming environment.

Each of the block-based languages mentioned is an educational language, and they are often seen as a first step for children, serving two purposes: (1) training the children in computational thinking (ref), and (2) preparing some children to make the transition to "real" text-based programming languages (ref). In our view, however, block-based languages hold much promise for application to a broader range of end-user programming situations, and for many people who are not children. In particular, aspects 3 and 4 are desirable in software where language-like information is edited by users who can't be expected to memorize the vocabulary and grammar of one, several of even many textual languages. Take, for instance the many languages used by system operators to build, configure, deploy and maintain complex systems. Or the proliferation, in the web development world, of mark-up languages, style-languages and JavaScript variants, libraries and frameworks.

Blockly, being a toolkit with which different block-based languages can be created, has already been used to create languages that aim to be both useful and easily accessible to different kinds of professionals who are not software engineers (ref,ref,ref). These efforts, however, have not investigated whether the block-based user-interface that Blockly supports (and which closely mimics Scratch's UI) could be adapted or enhanced to further support the needs and preferences of

professionals such as sysops, interaction designers, artists, data analysts and other end-user programmers.

We aim to research the design-space for block-based language interfaces aimed at end-user programmers who:

- do not need aspects 1 (playful domain) and 2 (simplified language) of the listing above;
- do not intend to use the block-based interface as a stepping stone toward text-based computer languages, or as an introduction to regular programming language semantics (all educational block-based languages mentioned above, embody an imperative structured programming language);
- want to be able to easily use some language(s) causally and sporadically, partly due to aspect 3 (direct structural manipulation) and 4 (high discoverability of language);
- but also, might grow into power-users and experience the language-UI as highly productive in day-to-day use;
- work with languages that may not fit the programming language grammar style (e.g. markup-languages like HTML and Latex where elements and text content can be mixed quite freely);
- work with language vocabularies and documents that are much larger than current block-based languages support comfortably;
- might benefit from immediate feedback from the system (e.g. type inferencers or live execution) and intuitive refactoring affordances that could be integrated into the block-based GUI more naturally than in text-based IDE's.

While we aim for a prototype that, like Blockly, can be applied to many different domains, we will be focussing on end-user programmers in the web development domain, primarily web designers who want to create working prototypes of their designs, *including* data-exchange with servers and other users.

2. First research activity

We have conducted a user-experience test to get a first exploratory sense of what web-designers consider useful or problematic in block-based language-interface. The participants were ten students of a 4-year multimedia-design bachelor programme (in their final year), who were tasked to create part of a moderately complex Arduino-application. On the basis of their 3,5 years of design-study, including at least a 5 month in-company internship, we consider these students to be sufficiently representative of our target audience (web designers). These students were also about 6 weeks into an entry-level programming course (covering JavaScript and Arduino), giving them enough experience to perform programming tasks that are not absolute beginner's level, but not enough for them to be averse to block-based languages just because of a long familiarity with text based code.

The participants were observed while performing two programming tasks lasting about 70 minutes, using a Blockly variant aimed at creating Arduino software that supported programming tasks that were very similar to task they had already performed in text-based Arduino code. Afterwards, they were interviewed about their thoughts and feelings about the way the interface supported the tasks. A screen recording of the task performance, including an eye-tracker overlay, was shown to the during the interview in order to prompt the participant to reflect in specific moments during the task performance. This method is called a *Retrospective Think Aloud* test which is considered to yield data that sheds more light onto the cognitive aspects of the user experience (as opposed to Concurrent Think Aloud, where the user's comments tend more to describe actions and sensory perceptions).

We are currently analysing the results of this test, and can not yet report definite findings. We will, however, informally discuss two things that were noted by the experimenter in the sessions with almost all participants.

3. Sources for improvements in the block-based user experience

Participants were quickly looking for UI-features they knew from professional graphical design tools such as Sketch or Adobe Illustrator, specifically keyboard-shortcuts and other features that speed up common tasks in direct manipulation interfaces like copying objects, creating selections of multiple

objects etc. This is, of course, not unexpected with participants with a background in design. But an often-stated motivation was that the drag-and-drop interface felt cumbersome when the user was executing a plan involving multiple steps he/she had already in mind. Rather than dismissing their requests for features found in graphical design software as mainly a desire to work in familiar ways, we surmise that these tools have been perfecting drag-and-drop direct manipulation interfaces (and accommodating user's need for speed of manipulation) for decades, and that block-based language UI's could improve for many more kinds of users by learning from the UI-design of these professional design tools.

Another source for ideas to improve block-based UI design turns out to be text-based languages. Like many Blockly-based tools, the ArduBlockly tool used in the test displays the generated textual source-code next to the canvas where the blocks are edited. We did not remove this source-code panel, because it provides a very quick way to our participants to learn the semantics of the (Arduino-specific) blocks offered by the tool, given that they already understand Arduino C code. Many participants, however, used the source code panel for a second purpose: to quickly scan the entire code of their program (their task started with a given program of 177 blocks, equivalent to 72 lines of C code) while looking for specific parts of the program, and getting an overall sense of the program's structure. During the interviews, some participants stated that they were simply more familiar with the text-based code, so could scan and read that code more easily. Others, however, explained their preference for reading the text-based code by pointing out several aspects of the block-based UI: the abundance of distracting colors, the verbosity of labels in blocks, the messy feel caused by unaligned top-level blocks, the unavailability of a linear sequence for scanning code, and the need to switch between scrolling horizontally and vertically, or dragging the canvas (which was error-prone: one could accidentally drag blocks apart instead of making another part of the program visible). These complaints point to the possibility that some design-decisions that work for children editing small programs (e.g. free-form placement of blocks on 2d-canvas) are, currently, not working well at larger scales. We do not suggest that a block-based language UI for professionals should emulate the linear nature of plain text files, but that it is worth finding out if designs are possible that combine the best of both worlds.

These are not the only sources for design improvements we have in mind. The Cognitive Dimensions of Notations framework was very much developed as a tool for designers to discuss and evaluate designs and design goals. A systematic analysis of block-based language UI, using the CDN framework, and with our new set of design goals in mind, is likely to yield useful and important requirements and directions for exploring the design space. This analysis will also incorporate the results of the Cognitive Dimensions questionnaire that all the participants of our user experience test have filled out.

4. Conclusion

Block-based programming languages have some demonstrated benefits that can help end-user programmers of all kinds. But being designed for children learning to program, they are likely to have some drawbacks that might prevent professionals from adopting block-based tools as part of their day-to-day work. In initial user study, focused on web designers is currently being analysed, and is expected to result in requirements and design opportunities for evolving block-based language UI's to be useful and productive for new user groups and new use cases.