

Extending Scratch: New Pathways into Programming

Sayamindu Dasgupta, Shane M. Clements, Abdulrahman Y. idlbi, Chris Willis-Ford, and Mitchel Resnick
Media Lab, Massachusetts Institute of Technology
Cambridge, MA. USA

Email: {sayamindu,shanemc,adlogi,cwillisf,mres}@media.mit.edu

Abstract—We present the Scratch extension system, a toolkit that enables anyone to extend the vocabulary of the visual Scratch programming language through custom programming blocks written in JavaScript. The extension system is designed to (i) enable *innovating on* the Scratch programming language itself, in addition to *innovating with it* through projects, and (ii) enable the *creation of new interest-driven pathways* into Scratch programming. In this paper, we describe some of the prior work done in this space, our design and implementation, open questions and challenges, and some preliminary results.

keywords: Scratch, extensions, block languages

I. INTRODUCTION

Scratch [1], [2] is a visual, block based programming language for children. In Scratch, visual blocks are snapped together to program “sprites”, which are graphical objects on a Scratch stage (Figure 1). With Scratch, millions of children all over the world have created a wide range of projects, from games to interactive stories, from computational art to science simulations. Scratch has an associated online community where Scratch can users share and remix projects.

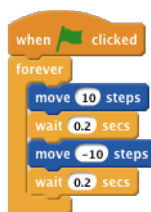


Fig. 1. Scratch code to move a sprite back and forth

Since the launch of the Scratch programming environment and online community in 2007, the core language has remained largely unchanged in terms of grammar, as well as vocabulary. However, as the official Scratch language remained unchanged, enthusiasts and users from all around the world have created unofficial modifications (“mods”) of Scratch, using the freely available source code and other means. These mods extend the grammar of the language, often adding more advanced computational concepts (e.g. advanced data structures) but more frequently, they added new capabilities to the language through the addition of new blocks (blocks in Scratch are the visual equivalent of language primitives). The functionality offered by these blocks range from the ability to communicate with hardware devices (e.g. Microsoft Kinect), to the ability to fetch and parse web pages.

While a lot of focus has been on how young people build a diverse range of projects with Scratch [3]–[6], the phenomenon of modding clearly shows another level of creativity and

innovation within the Scratch ecosystem. In this paper, we present the Scratch extension system, which we have designed and implemented to fulfill the two closely related goals of:

- enabling young Scratch programmers (and others) to innovate on the language itself, by extending it.
- enabling learners with a diverse set of interests to engage in programming with Scratch by opening up a number of previously unavailable pathways, through new domain-specific programming primitives.

We start by describing related work in this space and then discuss the design of the Scratch extension system. By an explicit design choice, on one hand, Scratch extensions allow for a subset of the functionality possible with Scratch mods; on the other hand, this constraint makes extensions more coherent with the rest of the Scratch language, making it easier to incorporate them into the official version of Scratch. In the design & implementation section, we discuss this trade-off, along with some challenges and open questions that emerge around it. We conclude by sharing some preliminary results from a limited (invite-only) developer program for Scratch extensions and charting our future directions.

II. PRIOR AND RELATED WORK

The source code of the first generation of Scratch (1.x) has been freely available, and this has allowed advanced Scratch users and enthusiasts to modify and extend the programming language in different ways. These modification and extensions to Scratch would typically introduce new functionality to the language, and in some cases, extend its grammar.

The team behind Scratch would also create mods to experiment with new ideas without affecting normal users. For instance, NetScratch [7] was an early modification that had new blocks to receive data from the Internet. In another mod, Scratch was extended to support image-processing primitives, so that children programming with Scratch could create their own image filters [8]. Some of these experiments were incorporated into the official version of Scratch – one such example being the mod that made it possible to use the LEGO WeDo kit with Scratch.

Outside of efforts by the Scratch development team, Chirp was a Scratch mod that focused on adding new features like exporting and importing code as XML, and compiling projects into executable files (to make standalone applications). Another notable mod, Build Your Own Blocks (later called Snap!) [9] extended the language by allowing users to define new blocks (procedures and functions). Additionally, multiple BYOB projects could interact by sharing variables and broadcasts.

TABLE I. CATEGORIES OF SCRATCH MODS [10]

<p>Hardware Support: Enchanting (LEGO Mindstorms), S4A (Arduino) and Kinect2Scratch (Microsoft Kinect)</p>	<p>Web Data: NetScratch, Snap! and Insanity.</p>
<p>Grammar extensions: Snap!</p>	<p>Operators: Insanity and Snap!</p>

BYOB also extended the grammar of Scratch by adding new computational concepts like programmatic cloning to let sprites duplicate themselves at program runtime, and new data structures like first-class lists. The design team behind Scratch considered adding some of these features to Scratch but they were always cautious in establishing a balance between making Scratch more powerful by adding new functionality and still having a language that is easy to get started with¹.

Mods not only extended Scratch within the realm of the virtual world, but also they enabled Scratch to connect to hardware devices. S4A is a mod that provides new blocks to handle sensors and actuators connected to an Arduino board. Enchanting is another mod that is used to program LEGO Mindstorms NXT robots. In contrast to most other mods, Enchanting removed some of the Scratch blocks, thereby making itself more specialized, and incompatible with projects created with the official version of Scratch. Table I shows the different categories of Scratch mods.

Apart from direct source code modification, another method to extend Scratch was introduced with version 1.3.1 through what was called the remote sensors protocol. When the remote sensor connection feature is enabled, custom Scratch blocks can be created with programs that talk to Scratch through a network protocol. Two notable examples of Scratch extensions made using this protocol are JoyTail, which adds joystick support to Scratch, and Kinect2Scratch, which allows creating Scratch programs that communicate with Microsoft Kinect.

It should be noted here that regardless of the method that was used to extend Scratch, projects created with a mod could not be shared on Scratch online community website. Links to get new mods could only be shared on the forums or Scratch Wiki, but they were not endorsed or curated by the Scratch team.

Outside of Scratch, the equivalent of extensions is common in programming languages. Notable examples include Perl (through CPAN [11]), R (through CRAN [12]), NodeJS (through NPM [13]), etc. Additionally, end-user tools, such as the Atom text editor [14] or web-browsers such as Chrome and

¹Some of these features (cloning, procedures) were added to Scratch 2.0 in a modified form.

Firefox have often relied on extensions [15], [16] to provide additional functionality.

III. DESIGN & IMPLEMENTATION

Based on observations from modding practices in the wider Scratch community, along with results from experiments within the Scratch team (e.g. workshops with the remote sensor protocol), the Scratch extension system was developed in order to allow easier additions to the functionality of the core programming language. Also, the emergence of Open Data and web-APIs served as an additional motivating factor behind the design and implementation of the extension system [17]. The affordances of the extension system allow for the augmentation of the vocabulary rather than the grammar, and hence, certain features that have been seen in mods (e.g. new control structures) are not implementable through the extensions system. This section covers the design and the trade-offs made, along with brief references to the implementation.

The Scratch extension system is designed to allow experienced Scratch users (as well as external developers) to expand the Scratch language into new domains, e.g. interaction with hardware devices such as the Arduino (Figure 2) or the Kinect, or access to online web-services, such as Google spreadsheets, or the US National Oceanic and Atmospheric Administration (NOAA) weather service.

```

ext.speak = function(text) {
  msg = new SpeechSynthesisUtterance(text);
  window.speechSynthesis.speak(msg);
};

// Block definition
var descriptor = {
  blocks: [
    [' ', 'speak %s', 'speak', "Hello!"],
  ]
};
    
```

Fig. 3. Simplified text-to-speech extension implementing a “speak” block. Each extension block is defined in JavaScript as a list with the block-type, a specification, associated method-name, and default parameter (if any).

At the core of this system are individual Scratch extensions – small sets of blocks that enable new functionality in Scratch. An extension adds a set of new blocks and expands the Scratch vocabulary, while adhering to the language grammar. These extensions themselves are implemented in JavaScript; effectively, each extension block is mapped to a JavaScript method that gets invoked through a “bridge” layer implemented within Scratch. A simplified example of a “text-to-speech” extension that uses the HTML5 Speech API [18] is shown in Figure 3. In this example, there is a single block called `speak _` which is mapped to the JavaScript method called `ext.speak` that calls the HTML5 API to convert text into speech. We chose JavaScript as the programming language for implementing extensions not only because Scratch runs on the browser (and thus has easy access to Javascript), but also due to JavaScript’s increasingly widespread use, and wide range of powerful features². Based on observations from the modding community, we expected extensions to be developed by some of the more advanced users using Scratch, as well

²Access to hardware devices from JavaScript was made possible by a custom browser plugin

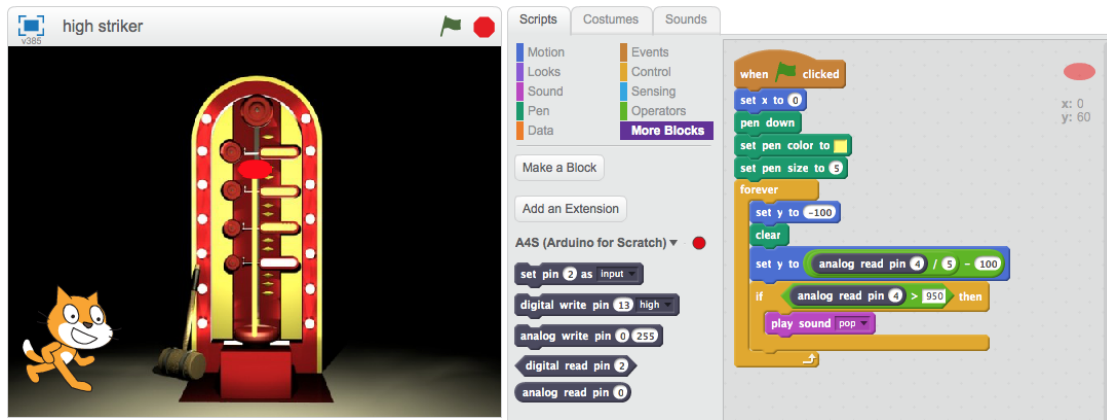


Fig. 2. A Scratch project using the blocks from the Arduino extension, reading values from a pressure sensor

as developers of hardware kits who want to see their devices supported in the Scratch language.

It is worth re-iterating here that extensions are limited in scope by what they are allowed to do. They are not allowed to make alterations or additions to the grammar of the Scratch language, neither are they allowed to affect the look or behavior of sprites, or the stage (background). This was by design, as we felt that any other choice would give extension authors too much control over the Scratch user-experience. However, even with this somewhat limited extension system, we see a few challenges in ensuring that the extensions are coherent with the rest of the language. Some questions around ensuring coherence include:

- Maintaining the right level of abstraction:* In their 2005 paper, Resnick and Silverman [19] highlight the importance of choosing “block boxes” carefully. They give an example where a block to control the color of a LED turned out to be too complicated when it had three parameters (one for the red-color component, one for green, and another for blue). They ended up using a single, numerical value to represent color, as their goal in that specific context was to enable children to express themselves with color, and not to impart an understanding of the red-green-blue composition of light. We expect that this type of design question will arise for most extensions, and it will be important to help extension developers understand and apply the “choose black boxes carefully” principle for their specific design goals appropriately.
- Maintaining consistency with conventions used in other blocks:* Scratch has some carefully chosen conventions. For example, for blocks that control motor movement, Scratch uses the direction indicators ‘this way’ and ‘that way’, instead of clockwise and counter-clockwise. Another commonly used convention is that numerical properties and variables are changeable in two different ways, through a direct “set” block, as well as by an incremental “change by” block. A specific use of this convention can be seen in how the coordinates of a sprite on the Scratch stage can be set – there is a `set x to _` block, and a `change x by _` block. Maintaining a list of commonly used

conventions in Scratch can help extension developers be consistent with the rest of the Scratch environment, as well as with other extensions.


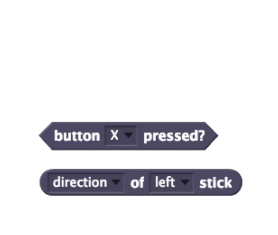

- Maintaining a preference for low floor:* The Scratch design team has preferred to keep a low barrier to entry (low floor) [1], especially when given a choice between a low floor and sophisticated possibilities (high ceiling). There has been also a preference towards minimalism in Scratch – e.g. blocks are designed carefully to not to have a large number of parameters. Ideally extensions should also follow these preferences.

To address these challenges, we have included some guidelines and best practices in the technical documentation for the extension authors. As the number of extensions increase, we continue to iterate on these guidelines – our general goal is to enumerate a small set of rules to follow in order to avoid “raising the floor”. Things to avoid in this category would include overly complex blocks with multiple inputs, highly specific or specialized blocks, blocks that have hard-to-understand labels, blocks that can disrupt or interrupt the Scratch application, etc. Additionally, we are working towards guidelines that would include a list of commonly used conventions in Scratch, in order to ensure consistency in the overall Scratch ecosystem.

IV. PRELIMINARY RESULTS

In August 2014, we started an invite-only Scratch extension developers program, where we invited Scratch users and developers who had already shown an interest in this space through various means (e.g. by creating Scratch mods previously, participating in extension workshops that we conducted, etc.). Additionally, as the news spread by word-of-mouth, a few Scratch users and external developers applied to the program as well, and were accepted. Once a member of the extension developer program, a user on the Scratch website got access to the ability to load experimental extensions into the Scratch programming environment (editor) and create projects with extension blocks. Projects containing extension blocks were barred from being publicly shared on the website, as we wanted to avoid confusing other users with blocks that are not a part of the standard Scratch vocabulary, or worked under special circumstances (e.g. the text-to-speech extension,

TABLE II. SOME OF THE EXTENSIONS DEVELOPED BY PARTICIPANTS IN THE DEVELOPER PROGRAM

		
Leap Motion extension	Gamepad extension	Soccer World Cup extension

described earlier, works only in certain browsers). We have future plans for a curation mechanism where we would be able to approve and accept certain extensions and make them a part of the “official” Scratch extension library.

Over the course of 2014 and early 2015, the developer program had 70 members, 28 of whom created at least one extension. The total number of extensions created and used within the system was slightly more than 100 (this is approximate, as at least some of these are different versions of a single extension). Among these 28 members, only 6 self-report their age as being between 5 and 18, so in this sample, the majority of extension developers were adults. However, it would be premature to read much from this data as the program was restricted, and also not advertised widely. The extensions that emerged from this developer program fall into three broad categories:

- *Hardware extensions:* These support connecting Scratch with a wide range of hardware devices, such as the Arduino, LeapMotion, littleBits, etc. This category form the majority of the extensions created within the developer program.
- *Web-API extensions:* These extensions utilize different types of web-APIs – e.g. an extension provides a block to translate a string in a given language to another using an online translation API; another extension provides blocks to return the results of matches in the 2014 Soccer World Cup.
- *Pure JavaScript/HTML5 extensions:* These extensions take advantage of functionality built into JavaScript and HTML5 APIs, in a way similar to the text-to-speech one described earlier (which was created by us). Examples include extensions that add to the existing list of math blocks in Scratch (e.g. a block to calculate factorials); another extension in this category adds blocks to process strings (e.g. turn a string into uppercase, reverse a string, etc.).

Table II show some of the extension blocks that were created by users and developers outside of the Scratch team.

V. CONCLUSIONS & FUTURE WORK

We presented the design of the Scratch extension system, along with a brief overview of its implementation. We described some of the questions that emerged out of the design process – especially questions around maintaining the coherence, ease-of-use, and understandability of Scratch when

a large number of extensions become accessible to all. These questions are important to consider in the context of any language that is specifically designed for first-time programmers.

Gauging from the wide variety of extensions that has been created within the limited developer program, the Scratch extension system seems to be on a path to meet the two design goals of (i) enabling members of the Scratch community to extend the language itself, and (ii) introducing new pathways into programming with Scratch. Hardware extensions seem to be more popular than other types of extensions, and a majority of extension-developers seem to be adult Scratch enthusiasts, but as we open up the extension system to more users and developers, this imbalance may become less pronounced.

In terms of future work, we are moving away from the invite-only developer program model to a system where anyone can develop and try out extensions. We hope that this new system (separate from the Scratch community website) will encourage more people to develop, as well as use extensions. Another next step for us is to come up with a curation criteria and process to start including extensions within the “official” Scratch extension library. The curation criteria will be driven by the questions raised in the design section of this paper, and our hope is to achieve the best possible balance between a diverse-range of possible activities powered by extensions, and the low barrier to entry that has always been at the core of Scratch.

ACKNOWLEDGMENT

We would like to thank members of the Lifelong Kindergarten group at the MIT Media Lab, especially John Maloney, Natalie Rusk, Ray Schamp, Kasia Chmielinski, and Amos Blanton for their support, guidance and suggestions. Additionally, we would like to thank Brian Silverman and Paula Bonta of the Playful Invention Company for their feedback and thoughtful suggestions. This work stands on, and depends on efforts of the community of Scratch modders, extension developers (especially Kreg Hanning and Connor Hudson), as well as the Scratch community itself, and we would like to express our sincere gratitude towards them as well. Part of this paper (especially sections on hardware extensions) draws from co-author Abdulrahman Y. idlbi’s masters thesis [10]. Financial support for this project has come from National Science Foundation (grant numbers 1002713, 1027848, and 1417952). Any opinions, findings, and conclusions, or recommendations expressed in this document are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

- [1] M. Resnick, B. Silverman, Y. Kafai, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, and J. Silver, "Scratch: Programming for All," *Communications of the ACM*, vol. 52, p. 60, Nov. 2009. [Online]. Available: <http://portal.acm.org/citation.cfm?doid=1592761.1592779>
- [2] A. Monroy-Hernández, "ScratchR: sharing user-generated programmable media," in *Proceedings of the 6th international conference on Interaction design and children*, ser. IDC '07. New York, NY, USA: ACM, 2007, pp. 167–168. [Online]. Available: <http://doi.acm.org/10.1145/1297277.1297315>
- [3] K. Brennan, A. Valverde, J. Prempeh, R. Roque, M. Chung, K. Brennan, A. Valverde, J. Prempeh, R. Roque, and M. Chung, "More than code: The significance of social interactions in young people's development as interactive media creators," vol. 2011, Jun. 2011, pp. 2147–2156. [Online]. Available: <http://www.editlib.org/p/38158/>
- [4] K. Brennan and M. Resnick, "Imagining, Creating, Playing, Sharing, Reflecting: How Online Community Supports Young People as Designers of Interactive Media," in *Emerging Technologies for the Classroom*, ser. Explorations in the Learning Sciences, Instructional Systems and Performance Technologies, C. Mouza and N. Lavigne, Eds. Springer New York, Jan. 2013, pp. 253–268.
- [5] J. V. Nickerson and A. Monroy-Hernández, "Appropriation and Creativity: User-Initiated Contests in Scratch," in *2011 44th Hawaii International Conference on System Sciences (HICSS)*. IEEE, Jan. 2011, pp. 1–10.
- [6] B. M. Hill and A. Monroy-Hernández, "The Remixing Dilemma The Trade-Off Between Generativity and Originality," *American Behavioral Scientist*, vol. 57, no. 5, pp. 643–663, May 2013. [Online]. Available: <http://abs.sagepub.com/content/57/5/643>
- [7] T. Stern, "NetScratch: a networked programming environment for children," Master's thesis, Massachusetts Institute of Technology, 2007.
- [8] E. Eastmond, "New tools to enable children to manipulate images through computer programming," Master's thesis, Massachusetts Institute of Technology, 2006.
- [9] B. Harvey and J. Mönig, "Bringing "No ceiling" to Scratch: can one language serve kids and computer scientists," *Proceedings of Constructionism 2010*, 2010.
- [10] A. Y. Idlbi, "Personalized extensions : democratizing the programming of virtual-physical interactions," Master's thesis, Massachusetts Institute of Technology, 2014. [Online]. Available: <http://dspace.mit.edu/handle/1721.1/95605>
- [11] "The Comprehensive Perl Archive Network." [Online]. Available: <http://www.cpan.org/>
- [12] "The Comprehensive R Archive Network." [Online]. Available: <http://cran.r-project.org/>
- [13] "NPM." [Online]. Available: <https://www.npmjs.com/>
- [14] "Atom Text Editor." [Online]. Available: <https://atom.io/>
- [15] "Chrome Web Store." [Online]. Available: <https://chrome.google.com/webstore/category/extensions>
- [16] "Add-ons for Firefox." [Online]. Available: <https://addons.mozilla.org/en-US/firefox/>
- [17] S. Dasgupta and M. Resnick, "Engaging Novices in Programming, Experimenting, and Learning with Data," *ACM Inroads*, vol. 5, no. 4, pp. 72–75, Dec. 2014. [Online]. Available: <http://doi.acm.org/10.1145/2684721.2684737>
- [18] Glen Shires and Hans Wennborg, "Web Speech API Specification," Oct. 2012. [Online]. Available: <https://dvcs.w3.org/hg/speech-api/raw-file/tip/speechapi.html>
- [19] M. Resnick and B. Silverman, "Some reflections on designing construction kits for kids," in *Proceedings of the 2005 conference on Interaction design and children*, ser. IDC '05. New York, NY, USA: ACM, 2005, pp. 117–122. [Online]. Available: <http://doi.acm.org/10.1145/1109540.1109556>