Fun and Engaging Pre-CS1 Programming Languages

Daniel D. Garcia (Moderator)
UC Berkeley
Berkeley, CA, USA
ddgarcia@cs.berkeley.edu

Michael P. Rogers
University of Wisconsin Oshkosh
Oshkosh, WI, USA
mprogers@mac.com

Andreas Stefik
University of Nevada, Las Vegas
Las Vegas, NV, USA
stefika@gmail.com

ABSTRACT

The *CSforALL* movement to bring computational thinking to K-12 has been a boon for practitioners and language developers. This panel features three educators passionate about a particular language that has been successful with K-12 audiences. Each will demonstrate their language, describe what makes it unique, and share some of the fun and engaging projects students have created.

CCS CONCEPTS

• **Social and professional topics** → **Computer science education**;

KEYWORDS

Outreach, K-12 CS Education, CS0, Pre-CS1, Languages

1 SUMMARY

The growth in interest in computer science education for K-12 and CS0 learners has been a boon for language developers, who have designed engaging and interactive languages and programming environments. The purpose of this panel is to provide a venue for experts in three languages to give demonstrations of their systems and share their benefits with the community, in hopes of inspiring a new generation. The intended audience is any educator working with K-12 or CS0 students. This is relevant to the SIGCSE community, since so many attendees are part of the *CSforALL* movement.

2 PANEL STRUCTURE

The panel will begin with the moderator briefly sharing the origin behind the idea for the panel, its format, and an introduction of the panelists, then each of them will have 10-15 minutes to demonstrate their language. Questions can begin to be answered in the chat as soon as a panelist finishes speaking, since they will be free to respond. The audience will have thirty minutes at the end for audio/video questions and discussion, which might be aimed at a specific presenter (e.g., "earlier you said *X*, can you clarify what you meant?"), or might be addressed to all presenters (e.g., "can you all show how your language can do *Y*?"). To keep things orderly, attendees will raise their virtual hand to ask questions, and the moderator will choose them, one-by-one, from the question queue.

SIGCSE 2021, March 17-20, 2020, Toronto, Ontario, Canada

ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.1145/1122445.1122456

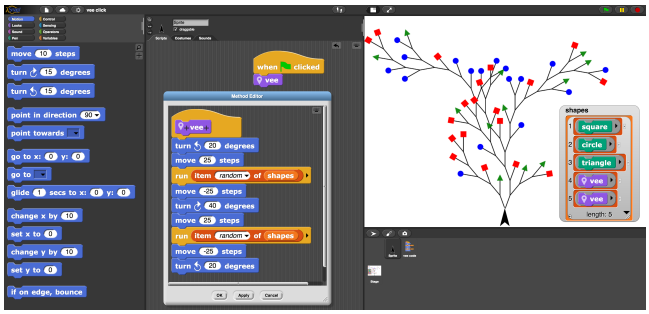

Figure 1: The browser-based Snap*!* programming environment, and the "Vee" project we use to introduce recursion.

3 DAN GARCIA, SNAP*!*

Snap*!* is a visual, drag-and-drop, browser-based programming language. It is an extended reimplementation of *Scratch* (a project of the Lifelong Kindergarten Group at the MIT Media Lab) that allows you to Build Your Own Blocks. Like **Scratch**, you can author procedures (called *Commands*) without return values. Unlike **Scratch**, you can author functions (called *Reporters*) that *do* have return values. It also features first class lists, first class procedures, and continuations. These added capabilities make it suitable for a serious introduction to computer science for high school or college students. In fact, we've had over 800 teachers in our worldwide professional development programs!

In our *Beauty and Joy of Computing (BJC)* AP CS Principles high school and CS0 university classes, we use Snap*!* for many reasons[1, 2]. Since it is based on the design of Scratch, students can create rich, object-based, multimedia projects easily. Projects are stored on the cloud and can be shared with any smart device, so creating a mobile app is as simple as pointing a smart phone to the URL of a shared project. It also allows us to teach powerful computer science ideas, like recursion and functions-as-data. We emphasize a functional-style approach, and students eventually find our built-in utilities `map`, `keep`, and `combine` to be second nature.

The excitement for many students comes at the end of the course when they work in teams and are free to choose any final project they want. Many use this opportunity to explore some of the advanced features of Snap*!*. These are:

- The `JavaScript function` block to author raw JavaScript and use some of its powerful built-in libraries.
- The `url` block to talk to Internet APIs.
- Hardware devices (e.g., the LEAP motion controller, Finch and Hummingbird robots, Sphero, Lego NXT, and Arduino).
- Built-in libraries (e.g., streams, infinite precision integers, animation, parallelization, and media computation).

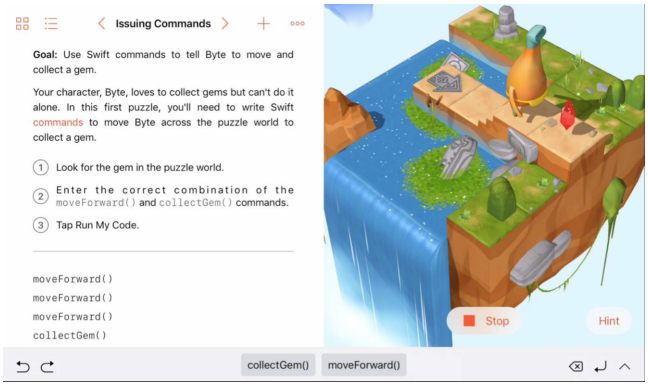

Figure 2: One of the first exercises in the Swift playground "Learn To Code 1".

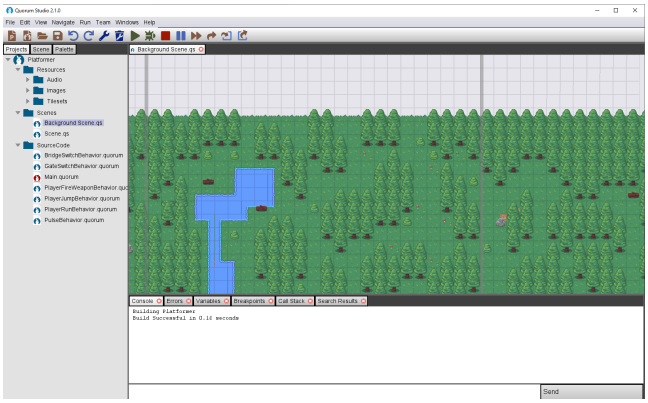

Figure 3: An image of a visual, but fully accessible, scene editor in the Quorum Studio programming environment.

4 MICHAEL P. ROGERS, SWIFT PLAYGROUNDS

Swift is a relatively new programming language, sprung upon an unsuspecting developer community during Apple's World Wide Developer Conference in 2014, as a replacement for the venerable Objective-C [3]. Swift is a modern, safe, syntactically-pleasing-to-the-eye language that lives up to its name, outperforming Objective-C in most benchmarks. One highly-lauded aspect of Swift is its ability to run in a REPL, known in Apple's IDE, Xcode, as a playground, allowing developers to quickly try out ideas without having to go through a lengthy compilation process.

Although designed for professional developers, Swift's elegant design makes it eminently suitable for beginning coders, including those in secondary education. Rather than burden novices with Xcode, however, Apple created an app that runs on iPads and Macintosh computers, Swift Playgrounds, and bundled it with three "Learn To Code" playgrounds that borrow some ideation from "Hour of Code", but are Swift-based. These tutorials disguise a rigorous, extensive course in Swift programming as a game, in which users have to instruct adorable animated characters Byte, Blu and others, how to perform assorted tasks.

Swift Playgrounds is backed by an impressive amount of instructional material that teachers in secondary education will appreciate, and mappings to the CSTA-K12 standards that will help make the argument for using Swift Playgrounds in the classroom.

5 ANDREAS STEFIK, QUORUM

The **Quorum** programming language originally started in 2009 as a language that was designed for the blind or visually impaired [4]. The approach, at first, was to carefully adjust syntax and semantics to both be simpler to a younger audience and to be output through screen reading devices. These initial designs were based on a considerable amount of human factors evidence. Since that time, Quorum has continued to gain popularity (especially among high school students), and remains the language of choice in the disabilities community.

Overall, Quorum is what is called an "evidence-oriented" programming language, which means that when decisions are made on how the language should change over time, they are based on empirical data, not just the personal opinions of the development team. Generally, these decisions are based on either randomized controlled trials with human-subjects, field data from other scholars in the field, or other approaches. As an example of another evidence-based approach to development besides traditional methods, priorities for API development are voted on annually by our community. This helps give users a "buy-in" for what the development team is working on. This has helped lead to the user community for Quorum growing over time from around 5 students in 2010 to more than 103,000 people writing programs online annually today, and more using the offline environments.

Additionally, as Quorum has developed, it has garnered a rich set of built-in libraries. This includes connections to many other domains, 2D and 3D graphics routines, and libraries for user interface creation, networking for service architectures, digital signal processing, LEGO robotics, and physics-based animation. Notably, Quorum Studio, the development environment for the language, uses a specialized graphics technology for creating fully visual and fully accessible game scenes, even if a user is blind. Quorum works online in a browser at `quorumlanguage.com`, can be used offline in Quorum Studio, and can compile programs to Android, JavaScript or Java Bytecode.

REFERENCES

[1] BJC. 2020. The Beauty and Joy of Computing. http://bjc.berkeley.edu/. (2020). Cited 2020 August 27.
[2] Dan Garcia, Brian Harvey, and Tiffany Barnes. 2015. The beauty and joy of computing. *ACM Inroads* 6, 4 (2015), 71–79.
[3] Michael P. Rogers and William M. Siever. 2015. A Swift Introduction to Swift App Development (Abstract Only). In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. ACM, New York, NY, USA, 706–706. https://doi.org/10.1145/2676723.2678281
[4] Andreas Stefik and Richard Ladner. 2017. The Quorum Programming Language (Abstract Only). In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17)*. ACM, New York, NY, USA, 641–641. https://doi.org/10.1145/3017680.3022377