BlockyTalky: A Low-Cost, Extensible, Open Source, Programmable, Networked Toolkit for Tangible Creation

Elise Deitrick
Tufts University
Medford, MA
elise.deitrick@tufts.edu

Joseph Sanford
Tufts University
Medford, MA
joe@cs.tufts.edu

R. Benjamin Shapiro
Tufts University
Medford, MA
ben@cs.tufts.edu

ABSTRACT

BlockyTalky is a low cost software and hardware toolkit that enables users to create their own networked devices. The platform may be used to create devices that interact purely in the physical world, such as autonomous robots, or that bridge the virtual and physical worlds, such as devices that provide a tangible, interactive representation of data from social networks. BlockyTalky provides users with a web-based visual programming environment hosted on a Raspberry Pi, and a cloud service allows users' creations to communicate with one-another, as well as other web services, over the Internet. Support for the BrickPi shield offers interoperability with LEGO NXT motors and sensors, and therefore a gradual pathway for learners from LEGO to more advanced computation and engineering. BlockyTalky enables users to learn introductory programming principles, basic robotics, and networked communication as they build custom devices.

Categories and Subject Descriptors

H.5.2 [**Information Systems and Presentation**]: Prototyping

General Terms

Design, Human Factors

Keywords

Tangible, Programming, Network

1. INTRODUCTION

Researchers have created a number of technologies and teaching approaches to broaden participation in computing. For example, Storytelling Alice [5], LilyPad Arduino [3], and Scratch [14] are programmable environments engineered to attract a broad range of learners to computing through storytelling, crafting, and game design. These technologies connect to youths' existing interests and participatory cultures, and offer youth the opportunity to produce work that is meaningful to themselves and their peers [1, 10, 12].

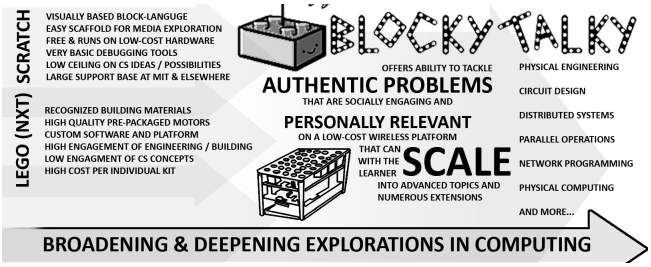

Figure 1: How BlockyTalky builds on existing systems to bridge gaps in engineering trajectories.

Paradoxically, the success of these efforts to broaden participation in computing has created a problem for learners: frequently these environments lack depth in computer science learning. Students write simple programs that utilize a limited range of computer science ideas (variables, conditional logic, loops, and, sometimes, procedural abstractions). These are exciting first steps, but when learners max out these introductory environments, there are no clear paths for them to follow to deepen their skills. Further, there is a significant gap between the abstractions and simplifications found in many of these introductory systems and in the formal and informal computer science educational materials and tools that learners can find in schools or online to reach more advanced levels. We believe that the scaffolded approach to computing that environments like Scratch and LilyPad represent *will only be successful in the long term if researchers in our field also create bridges between introductory experiences in computing and more advanced ways of participating.* Moreover, because scaffolding must, by definition, fade away [11, 4], we must create mechanisms to gradually pierce the simplifications and abstractions that beginning environments present (Figure 1).

Consider the following example of what the current state-of-the-art enables: *Jane gets excited about Scratch because it will let her make a video game she has dreamed up. She creates the game and her friends at school love it and ask her to add a leaderboard to track the best scores. She does. When she posts the game online she discovers that many people on the Scratch Online community enjoy her game, but her leaderboard doesn't work anymore because it's not just on her computer. An online friend suggests that she*

IDC 2014
June 17-20 2014
Aarhus, Denmark
www.idc2014.org

check out the Cloud Variables that are part of Scratch 2. She modifies her leaderboard using this approach, and now players' high scores are synchronized between all players of her game.

Now, suppose Jane gets curious about how those Cloud Variables work and where else they are used in computers. Do they work in the same way that her iPhone is able to show her the weather from places around the globe? What about how Twitter Direct Messages show up on her phone and on her computer? Are they all just using Cloud Variables? And how does the data stored in those variables get synchronized between all of those places?

Jane has no obvious way to find answers to these questions, let alone implement new systems using the relevant computing techniques. There is an unmet need to create next steps in computing for interested beginners like Jane: they need opportunities to see how the abstraction that Cloud Variables presents actually functions, and to understand what the mechanisms are that relate Cloud Variables, weather apps, and multi-device-synchronized instant messaging.

2. OUR APPROACH

We believe that creating networked physical devices and virtual services – nodes that participate in a distributed Internet of Things – can be a powerful way to offer learners the kinds of next steps just called for.

Why Networks and Distributed Systems? It is essential that we create learning opportunities and processes in computer science that are situated in the *kinds of problems, media, cultures, and communities that youth already know and care about.* Responding to such interests and backgrounds is the central principle of culturally-responsive pedagogy [6, 7]. And yet, there exists a deep well of youth interest that has yet been tapped into by computing education researchers: participation in and use of networked services, communities, and devices. 95% of American teens now use the Internet, and 47% of American teens have smartphones. Moreover, the rate of teen smartphone ownership is fairly stable across bands of family income (39% for teens in families with income less than $30,000/year vs 43% in families with income greater than $75,000/year). Meanwhile, 93% of teens own or have access to desktop/laptop computers and 81% of teens use social media sites like Facebook and Twitter [9, 8].

In short, today's youth, regardless of race, gender, and income, use a heterogeneous mixture of online services and devices. They make daily use of networks and distributed computing in order to connect with friends, play games, and find information. The programmed properties of different social media systems have powerful impacts on the ways that youth are able to connect, present their identities, and manage conflict with peers; differences between system design decisions embedded in these technologies are of great personal importance to many youth and are systems that they already think about in nuanced ways [2].

The deep importance of these networked, distributed systems to youths' daily lives offers an unprecedented opportunity to motivate and situate learning about advanced topics in computer science. Just as Scratch taps in to storytelling and games, we should create environments that enable youth to build new technologies that are deeply connected to the Internet. Doing so would create an opportunity for learners to encounter and apply a variety of computer science ideas and techniques around distributed computing, networks, and parallel computing.

These ideas feature prominently in recent standards like the joint ACM-IEEE Computer Science Curricula 2013 which highlights the need for learning about the following topics:

- Abstraction
- Specification of interfaces
- Internet organization and terminology
- Networked, client-server, and distributed communication
- Security
- Task, data, and event parallelism
- Asynchronous and synchronous communication
- Reliable and unreliable protocols
- The need for concurrency in operating systems
- Web services
- Race conditions and consistency
- Partial failures, partitions, and other problems of distributed systems.

Building networked technologies for social connection would enable youth to learn these skills while making things that would be of great personal and cultural relevance.

Why Networked Physical Devices? As noted above, youth connect online using a variety of devices and application types, ranging from mobile phones to laptops, and from web browsers to social network apps and games, respectively. They request data from web services anchored to physical devices, such as weather data fed by thermometers and wind meters and fed by databases. They play games on consoles hooked to TVs and guide their characters through those games using handheld controllers. In the coming years, household devices will be increasingly connected to the web as well, with today's networked thermostats and refrigerators signaling a burgeoning Internet of Things that pervades daily life.

The growing ubiquity of these connected devices and services is an opportunity to support next steps in computer science education: first by building on the remarkable success of physical computing in providing many youth an entry point into computer science, then by enabling youth to connect their physical inventions up to the web so that they can connect their own devices together and connect their inventions with others'. We call this class of student-invented technology *Connected Devices* and the work that goes into making them *Connected Engineering.*

We hypothesize that the opportunity to invent Connected Devices will motivate participants to learn powerful CS ideas like parallelism, distributed systems, networks, and web services. Examples of such projects could include ambient interfaces to social media, sensor networks that capture data about local air quality, home or classroom automation, and tools to keep in touch with distant relatives or care for pets.

3. SYSTEM DESCRIPTION

BlockyTalky offers a unique mix of low cost and usability that lowers both the financial and the instrumental barriers to entry to physical computing. BlockyTalky is an open-source software project that can run on any device capable of running Linux and Python. It is most readily used on Raspberry Pi hardware costing about $35 per device. We usually pair each Pi with a BrickPi ($60), which allows the system to inter-operate with LEGO sensors and

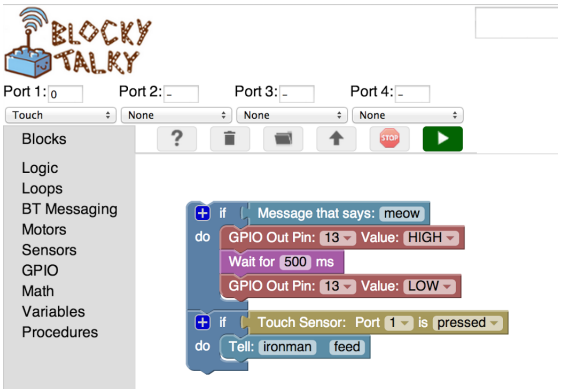

Figure 2: The BlockyTalky visual programming interface

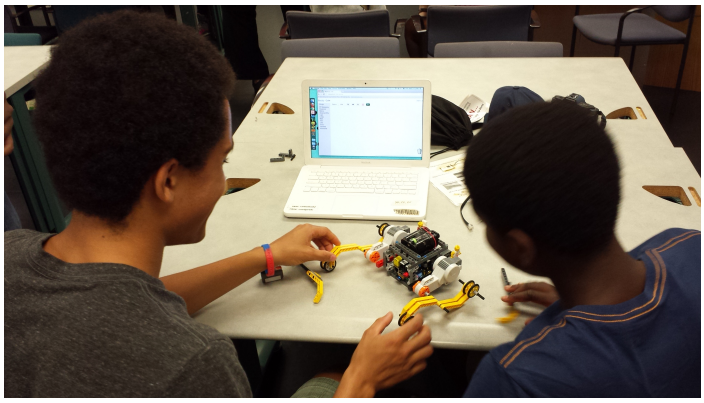

Figure 3: Students working with the BlockyTalky system using LEGO.

motors, thereby reducing the practical barrier to entry in device construction by enabling prototyping with plug-and-play sensors and motors. This toolkit supports the kinds of activities that have already proven successful in educational electronics and robotics (e.g., robotics competitions), but also supports the realization of thus-far broadly inaccessible project ideas, such as networked tangible games and toys, electronic musical instruments, and ambient representations of social network, weather, and scientific data. We wish to support users' creation of these technologies using a variety of materials, from commercially available systems like LEGO to combinations of 3D printed and craft materials. One example project, a remote cat treat dispenser, can be seen at http://youtu.be/osmJIX7HZic, and its program is presented in Figure 2.

A user begins by connecting to the BlockyTalky unit in his/her web-browser, which takes the user directly to the visual programming environment hosted on the unit. The programming interface is blocks-based (based on Google Blockly), much like Scratch, Alice, and App Inventor (which also uses Blockly, creating future opportunities for youth to build applications that span invented physical devices and mobile phones). We believe this will enable users to easily transition from other introductory environments to new kinds of programming projects. These projects can be rooted in their physical environments, using sensors, motors, lights, and speakers, or deeply connected to the web, such as by responding to events on social media or sending messages to other users' devices. Because it offers an easy, familiar interface learners can begin programming physical, networked devices, without wrestling with the intricacies of text-based programming, while also gaining entree to a variety of low-cost, open source hardware. This supports students' progression beyond introductory skills by offering a pathway for gradual deepening of skills from what they already know (relatively-simple blocks-based programming) into new territories for learning (networks and hardware design).

We are developing a system to bridge this gap created by introductory environments. To address this problem, we want to ensure that our tool is scalable within K-12 education. Using the heuristics set forth for scalable game design [13], we can evaluate if our system:

- Has a Low Threshold
- Has a High Ceiling
- Scaffold Flow
- Enable Transfer
- Supports Equity
- Systematic and Sustainable

By using a blocks-based interface, we create a low threshold which allows users to piece together programs in seconds. As described above, not only is the programming environment a snap, but the networking component of the system described above allows students to reach computational concepts beyond the scope of most novice-friendly systems, creating a relatively high ceiling. Due to the graphical nature of the system, users can explore complex concepts within a language free of syntax worries, allowing users to move at their own pace and in their own direction with little direct instruction. By connecting to hardware that is also programmable in text based languages, when a user does reach the ceiling of our system she or he can use the computational concepts they have learned as well as all her/his knowledge of the hardware (and the hardware itself) to create even more sophisticated inventions. BlockyTalky's main new affordance is an easy to use abstraction for networking, which should enable it to be used for project types that are more broadly attractive, unlike some other platforms (such as LEGO NXT) that are best suited to robotics and have largely attracted boys. Lastly, due to its low cost, wide availability, and open source nature, schools can order whole classroom sets at one time as well as customize the system to their needs.

4. DEMONSTRATION OVERVIEW

Participants in a 30 minute demonstration will be able to vote on an assortment of tasks to complete including: making an ambient representation of social media, creating controllers for a multi-player game (such as Simon or Rock-Paper-Scissors) to battle each other, and a musical instrument or sound effects board. These fun and youth-inspired tasks will show participants how these networked devices are both easy to use and powerfully open-ended. LEGO pieces, including NXT motors and sensors, will allow participants to snap together any necessary hardware for their devices in minutes. Through their use of BlockyTalky they will be exposed to many powerful computational concepts, such as asynchronous communication, without having to go beyond

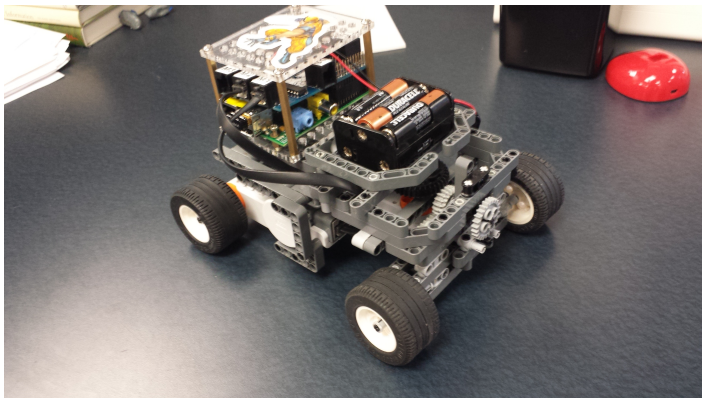

Figure 4: Car built with the BlockyTalky system and LEGO pieces.

the friendly blocks-based interface.

5. ACKNOWLEDGMENTS

We thank Ben Helm, ZoltÃąn StÃąl, Neekon Vafa, Isobel Redelmeier, and John Cole for their assistance in developing BlockyTalky.

6. REFERENCES

[1] P. Blikstein. Travels in troy with Freire: Technology as an agent for emancipation. *Paulo Freire: The possible dream. Rotterdam, Netherlands: Sense*, 2008.

[2] D. Boyd. *It's Complicated: The Social Lives of Networked Teens*. Yale University Press, New Haven, CT, USA, 1st edition, 2014.

[3] L. Buechley, M. Eisenberg, J. Catchen, and A. Crockett. The lilypad arduino: using computational textiles to investigate engagement, aesthetics, and diversity in computer science education. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 423–432. ACM, 2008.

[4] A. Collins, J. Brown, and S. Newinan. Cognitive apprenticeship: Teaching the craft of reading, writing, and mathematics. In *Knowing, learning, and instruction: Essays in honor of Robert Glaser*, pages 453–494. Lawrence Erlbaum Associates, Inc., 1989.

[5] C. Kelleher, R. Pausch, and S. Kiesler. Storytelling Alice motivates middle school girls to learn computer programming. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1455–1464. ACM, 2007.

[6] G. Ladson-Billings. But that's just good teaching! the case for culturally relevant pedagogy. *Theory into practice*, 34(3):159–165, 1995.

[7] C. D. Lee. Culturally responsive pedagogy and performance-based assessment. *Journal of Negro Education*, 67(3):269–79, 1998.

[8] M. Madden, A. Lenhart, M. Duggan, S. Cortesi, and U. Gasser. 2012 teens and privacy management survey. Technical report, Pew Research Center and The Berkman Center for Internet and Society at Harvard University, October 2012.

[9] M. Madden, A. Lenhart, M. Duggan, S. Cortesi, and U. Gasser. Teens and technology. Technical report, Pew Research Center and The Berkman Center for Internet and Society at Harvard University, March 2013.

[10] S. Papert. *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc., 1980.

[11] R. D. Pea. The social and technological dimensions of scaffolding and related theoretical concepts for learning, education, and human activity. *Journal of the Learning Sciences*, 13(3):423–451, 2004.

[12] K. Peppler. Media arts: Arts education for a digital age. *Teachers College Record*, 112(8):2118–2153, 2010.

[13] A. Repenning and A. Ioannidou. Broadening participation through scalable game design. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '08, pages 305–309, New York, NY, USA, 2008. ACM.

[14] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.