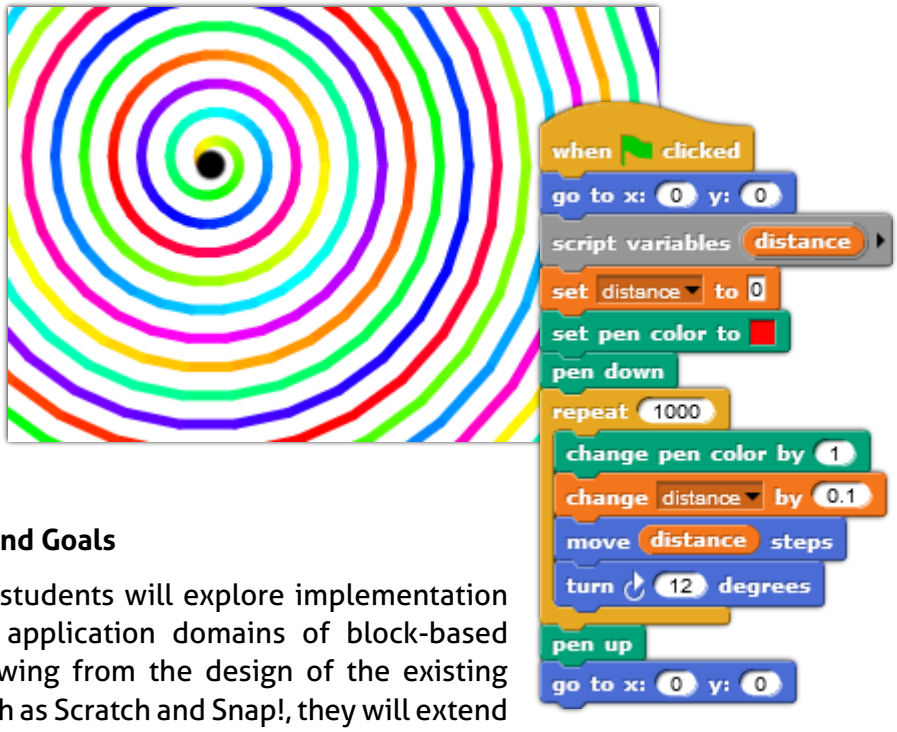
Blocks to the Rescue

Live Exploration of an Interactive Environment to Support Education, Construction, and Reflection in Program Design

Bachelor's Project Proposal 2018/2019
Software Architecture Group
Prof. Dr. Robert Hirschfeld



Project Scope and Goals

In this project, students will explore implementation strategies and application domains of block-based languages. Drawing from the design of the existing approaches such as Scratch and Snap!, they will extend the Squeak/Smalltalk live programming system to support the block-based paradigm as a flexible mechanism complementary to the Morphic graphics framework. The following goals form the starting point in this project and are likely to be refined and extended:

- *Design* and *implement* a new block-based scripting system in Squeak/Smalltalk
- *Explore* means to construct sophisticated object *state* such as pre-scripted sprites
- *Explore* means to construct sophisticated object *behavior* such as custom blocks
- *Refine* the script execution model considering concurrency and reflection
- *Explore* trade-offs in visual programming to teach messaging and objects

The architecture of Scratch 1.4 (http://scratch.mit.edu/scratch_1.4) can help understand the possible role of the Morphic graphics framework because that version runs in Squeak 5.1 (https://squeak.org/projects/).

Context and Motivation

The creation of interactive multimedia projects using text-based programming languages is challenging. The feedback loop is typically quite long and error-prone because programmers must learn and apply many low-level concepts to achieve high-level goals. Especially younger people might struggle with the conceptual overhead of algorithmic thinking and mental simulations while looking at chunks of text on screen. Given that we have high-resolution input and output devices, a more direct way of expressing the programmer's intent with an option to forgive mistakes is desirable.

Block-based programming systems such as Scratch and Snap! alleviate these challenges through a visual, puzzle-like metaphor. Programmers can directly place objects (called *sprites*) on a *stage* and begin adding and observing behavior via blocks in *scripts*. There are a few visual constraints to distinguish *commands* 

from *reporters* 

and *predicates*  to easily compose complex control flows. More advanced programming concepts such as recursion, higher-order functions, and reflection can be approached through user-defined blocks and "ringification"  . Consequently, such systems pose a carefully designed trade-off between learnability and expressiveness to serve programmers of all ages.

Sharing and collaboration is a complementary aspect in such environments, which make the creation of interactive content straightforward. People can learn from each other's design strategies and discover new elements. In Scratch, for example, this includes stages filled with animated sprites, scripts, libraries of costumes, sounds, and custom blocks. There is a large repository with projects to explore: https://scratch.mit.edu/explore/projects/all. We think that the *visual dimension* of block-based languages underlines the value of *seeing* what other programmers did with the same mechanisms.

Given any block-based language, application scenarios are diverse. In the scope of education, the transition from basic programming concepts to advanced techniques and maybe other (text-based) languages might be desirable. In the scope of program design, blocks could represent the dedicated scripting interface for any complex software system, comparable with Lua or Visual Basic Macros. We think that the direct-manipulation characteristics of block-based programming can complement any graphical, interactive application.

State of the Art

There are several implementations of block-based languages and environments. Many of them originated from an educational domain, where children should learn about how to tell computers to do something interesting. Later systems explored the integration into Web technologies or the temptation of having "blocks all the way down."

Etoys, http://squeakland.org/

> "Etoys [...] are models, simulations and games constructed by assembling tiles into scripts which send commands to painted objects intended to give a learner insight into an area of investigation. Later, as users become more adept at creating such scripts, they can move into other areas of Squeak's interface more suitable for their learning level. Expert users of Squeak [...] would not use the Etoy component for their creations, but a more expert level offering a different 'look and feel' and extended facilities."
>
> - Powerful Ideas in the Classroom, 2003

Scratch, https://scratch.mit.edu/

> "Scratch is a visual programming environment that allows users (primarily ages 8 to 16) to learn computer programming while working on personally meaningful projects such as animated stories and games. A key design goal of Scratch is to support self-directed learning through tinkering and collaboration with peers."
>
> - The Scratch Programming Language and Environment, 2010

Snap!, https://snap.berkeley.edu/

> "In the glory days of the MIT Logo Lab, we used to say, 'Logo is Lisp disguised as BASIC.' Now, with its first-class procedures, lexical scope, and first-class continuations, Snap! is Scheme disguised as Scratch."
>
> - Snap! 4.1 Reference Manual

Blockly, https://developers.google.com/blockly/

> "Block-based programming is often a starting point for programming. In the context of teaching computer programming, it is a gateway drug that gets students addicted, before moving them on to harder things. [...] How long this block-based programming period should last for students is hotly debated, [...]."
>
> - Ten Things We've Learned from Blockly, 2015

GP, http://gpblocks.org/

> "[...] a new blocks language that aims to be beginner-friendly, like Scratch, yet capable of scaling up to support larger applications. We hope to allow a world-wide community of users to share projects, sprites, and code libraries and to create new ones using a mash-up development style. [...] GP incorporates a strong notion of modularity. Modularity allows components created by different users at different times to interoperate without worrying about conflicts. [...]"
>
> - A Module System for a General-Purpose Blocks Language, 2015

Further Reading

Edwin L. Hutchins, James D. Hollan, and Donald A. Norman. *Direct Manipulation Interfaces.* In Journal for Human-Computer Interaction, vol. 1, no. 4, pp. 311-338, 1985.

John H. Maloney and Randal B. Smith. *Directness and Liveness in the Morphic User Interface Construction Environment.* In Proceedings of the Symposium on User Interface and Software Technology (UIST), pp. 21-28, 1995.

Dan Ingalls, Ted Kaehler, and John H. Maloney. Back to the Future: The Story of Squeak – A Practical Smalltalk Written in Itself. In ACM SIGPLAN Notices, vol. 32, no. 10, pp. 318-326, 1997.

B.J. Allen-Conn and Kim Rose. *Powerful Ideas in the Classroom: Using Squeak to Enhance Math and Science Learning.* Viewpoints Research Institute, Inc. 2003.

John H. Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. *The Scratch Programming Language and Environment.* In ACM Transactions on Computing Education, vol. 10, no. 4, article 16, 2010.

Brian Harvey and Jens Mönig. *Bringing "No Ceiling" to Scratch: Can One Language Serve Kids and Computer Scientists?* In Proceedings of Constructionism, 2010.

Mitchel Resnick and Eric Rosenbaum. *Designing for Tinkerability*. In Design, Make, Play: Growing the Next Generation of STEM Innovators, pp. 163-181, 2013.

Brian Harvey and Jens Mönig. *Lambda in Blocks Languages: Lessons Learned*. In Proceedings of the IEEE Blocks and Beyond Workshop, pp. 35-38, 2015.

Neil Fraser. *Ten Things We've Learned from Blockly*. In Proceedings of the IEEE Blocks and Beyond Workshop, pp. 49-50, 2015.

Jens Mönig, Yoshiki Ohshima, and John Maloney. Blocks at Your Fingertips: Blurring the Line Between Blocks and Text in GP. In Proceedings of the IEEE Blocks and Beyond Workshop, pp. 51-53, 2015.

Felienne Hermans and Efthimia Aivaloglou. *Do Code Smells Hamper Novice Programming? A Controlled Experiment on Scratch Programs.* In Proceedings of the International Conference on Program Comprehension (ICPC), 2016.

Alexander Repenning. *Moving Beyond Syntax: Lessons from 20 Years of Blocks Programing in AgentSheets*. In Journal of Visual Languages and Sentient Systems, 2017.

Implementation Details

The implementation of the project will involve programming with the Squeak/Smalltalk programming system and the Morphic framework **(**www.squeak.org**)**. It can involve learning about the Open Smalltalk Virtual Machine (www.opensmalltalk.org) and its plugin architecture to employ lower-level functionality and foreign libraries in the C programming language. An agile, iterative process will be employed for software development. All results will be published under the MIT license **(**www.opensource.org/licenses/MIT**).**

Organization

A group of about six to eight (6–8) students may participate in the project. Organization and tasks will be explored and mainly determined by the project participants. The project will be carried out at the Hasso Plattner Institute in Potsdam. Project participants are expected to communicate with our partner via the GitHub issues/wiki, e-mail, or video chat on a regular basis. Communication is likely to be conducted in English and German. In the winter term 2018/19, participants will work on exploring the domain and creating first prototypes. Main steps in design and implementation of a solution are to be executed in the summer term 2019. Expected results include a working software accompanied by appropriate documentation.

Partner and Contact

Jens Mönig, SAP Research, Deutschland

Profile https://www.linkedin.com/in/jens-mönig-b8a00925
E-mail jens@moenig.org
Video https://www.youtube.com/user/jmoenig
Code https://github.com/jmoenig

Prof. Dr. Robert Hirschfeld, Marcel Taeumel, Toni Mattis, Stefan Ramson, Jens Lincke
Software Architecture Group, Hasso Plattner Institute, Potsdam
http://www.hpi.uni-potsdam.de/swa, hirschfeld@hpi.uni-potsdam.de