Frog Pond: A Code-First Learning Environment on Evolution and Natural Selection

Michael S. Horn, Corey Brady, Arthur Hjorth, Aditi Wagh, Uri Wilensky
Northwestern University
Learning Sciences and Computer Science
2120 Campus Drive, Evanston, Illinois 60208 USA

{michael-horn, cbrady, uri}@northwestern.edu,
{arthur.hjorth, aditiwagh2012}@u.northwestern.edu

ABSTRACT
Understanding processes of evolution and natural selection is both important and challenging for learners. We describe a "code-first" learning environment called Frog Pond designed to introduce natural selection to elementary and middle school aged learners. Learners use NetTango, a blocks-based programming interface to NetLogo, to control frogs inhabiting a lily pond. Simple programs result in changes to the frog population over successive generations. Our approach foregrounds computational thinking as a bridge to understanding evolution as an emergent phenomenon.

Categories and Subject Descriptors
H.5.m [**Information interfaces and presentation (e.g., HCI)**]: Miscellaneous.

Keywords
Children; agent-based modeling; code-first environment; evolution; natural selection; design; learning.

1. INTRODUCTION
Understanding processes of evolution and natural selection is both important and challenging for learners [9, 15]. We present a learning environment called Frog Pond designed to introduce natural selection to elementary and middle school aged learners. With Frog Pond learners use NetTango [4], a blocks-based programming interface to NetLogo [21], to control the actions of colorful frogs inhabiting a lily pond environment (Figure 1). Chains of simple block with names like `hop`, `left`, `right`, and `hatch` form programs that can result in changes to the frog population over successive generations—in particular, frogs can become bigger or smaller (or both) depending on selection pressures exerted by the environment and frogs' behaviors. Importantly, children never program these outcomes directly; they never write code that explicitly says, "make the frogs get bigger". Instead, outcomes *emerge* as the aggregate result of hundreds of individual frogs enacting rules from the same program. These outcomes can be surprising and counterintuitive, but because they result from programs that learners create themselves, our hope is to encourage explanations that attribute change to emergence resulting from individual interactions ("bigger frogs are better hunters, so they get to reproduce more and have babies that are also big"). In this way our intention is to foreground *computational thinking* [10, 11, 27] as a bridge to understanding evolution as an emergent phenomenon.

With this goal in mind, we designed Frog Pond as a *code-first* learning environment (e.g. [20]). By this we mean three things. First, the primary way to interact with Frog Pond is by creating programs. Second, the programming interface is designed to be very easy to learn and use. And, finally, it is possible to create very short programs that nonetheless result in several distinct evolutionary outcomes. To achieve these design objectives, we have been conducting a design-based research study in which we have iteratively developed and refined prototypes over the past year and a half. As part of this process, we have tested versions of Frog Pond with visitors in a natural history museum. In this paper, we describe our resulting design and share findings from our most recent round of testing with museum visitors. Our findings suggest that Frog Pond enables learners to quickly build programs that lead to population-level changes. This provides an opportunity to reflect on mechanisms underlying evolutionary effects.

2. BACKGROUND
Understanding evolution is notoriously difficult for learners [2, 15, 23]. Developmental and cognitive psychologists have identified cognitive biases such as teleology and essentialism that hinder student understandings of evolution (see [15]). Other researchers have argued that students' unfamiliarity with emergent processes in general is problematic [2]. Wilensky and Resnick [22] note that in trying to understand complex systems such as evolution, students will often exhibit *slippage between levels* as they attribute properties or behaviors at the individual level to the population as a whole (or vice versa). For instance, when reasoning about natural selection, students might believe that changes in the population are the result of changes to individual organisms over the course of their own lifespans.

Emergent phenomena are often simulated using computational *agent-based models* (ABMs). In these models, individual computational agents have properties and can enact behavioral rules as the simulation runs. For example, the NetLogo modeling environment [21] is widely used in middle schools, high schools, and universities. By emphasizing the behavior of individual actors in a system, ABMs can help students draw on their own bodily and sensory experiences in the world [16, 24, 25].

IDC'14, June 17-20, 2014, Aarhus, Denmark.

http://dx.doi.org/10.1145/2593968.2610491

This project builds on more than fifteen years of work on evolution education using the NetLogo modeling environment [21]. This line of work has sought to present evolution as a set of emergent phenomena characterized by individual-level mechanisms that lead to population level effects. In the *SimEvolution project*, middle and high school students explored agent-based models of classic evolutionary phenomena such as peppered moths and genetic drift. In the EACH project [1], high school and undergraduate students explored and built models that assessed the advantages and disadvantages of selfish behavior and how cooperation could evolve. In the BEAGLE project [23] students use both agent-based modeling and participatory simulations to explore core mechanisms of evolution. These projects resulted in increased understanding of evolution and mechanisms that can lead to population-level changes. In the past few years, there have been efforts to include elementary students in this type of modeling [3, 5]. For example, *Evolution Readiness* [5] is a curriculum that uses interactive computer-based models and activities to help elementary students learn about evolution and natural selection.

While experimenting with a pre-existing agent-based model can be a powerful learning experience, there has been steady attention in involving K-12 students in programming their own models [6, 7, 26, 18]. Proponents argue that programming can expose underlying mechanisms and relationships [12, 17, 24] and can help students develop more personal connections in making sense of math and science [8, 17]. This emphasis on programming is also consistent with Constructionist theories of learning in the sense that students are engaged in building external artifacts that reflect internal conceptual structures and thereby make their thinking available for "debugging" [11]. Some of these recent projects have involved text-based programming languages and some have used graphical programming. Text-based languages are expressive and powerful, but they can sometimes be daunting to students and teachers, particularly at lower grade levels [28]. Blocks-based languages, on the other hand, while perhaps more inviting, can limit the complexity of programs and models that learners can create.

Several programming environments have been developed that attempt to preserve the advantages of agent-based modeling while being more accessible to younger learners. StarLogo TNG uses a domain-general blocks-based programming interface to allow learners to construct a wide range of models [7]. However, as it is a fully-featured modeling environment, the task of programming models can still be involved.

An alternative approach involves offering high-level primitives that are specific to a domain of interest. For example, Modelling4All [6] is an environment that allows for the construction of models from micro-behaviors. Modelling4All has been used to design museum exhibits and other short-duration learning experiences. Another example of this approach is DeltaTick [18, 26], a blocks-based programming available for NetLogo. DeltaTick enables designers to create blocks that are semantically close to the modeled domains. Building on DeltaTick, Wagh and Wilensky have involved middle school students in constructing models of evolution in a project called EvoBuild [18]. With EvoBuild, students model within-species variation by adding traits and manipulating the distribution of its variations in populations. Students can also assign rules to individuals by using behavior-based primitives such as "reproduce" or "die" that generate evolutionary outcomes over time.

To a certain extent these two efforts (addressing younger students and involving students in building models) have traded off against each other. For instance, the work of Dickes and Sengupta [3] have provided considerable scaffolding for younger students to explore models while model creation has been primarily tried with older students. In this project, our emphasis is on increasing the accessibility of programming even further, thus enabling young students to program their own models and grapple with evolutionary effects.

3. DESIGN OVERVIEW

We implemented Frog Pond using HTML5, CSS, and the Dart programming language. The use of these cross-platform web technologies allows the environment to run on several different types of devices including tablet computers, laptops, and multi-touch tabletops. Frog Pond is now available through app stores for some platforms. To reduce the visual separation between the code and its effect on the frogs, the programming workspace is superimposed directly on the pond. The workspace consists of a menu bar at the bottom of the screen from which users can select programming blocks; a toolbar with buttons for starting, stopping, fast-forwarding and restarting programs; and the user's current program (Figure 1). This language is similar to other blocks-based languages like Scratch [13], Blockly (code.google.com/p/blockly/), and Open Blocks [14], but it includes several interactive features designed to make it easier for inexperienced users to get started.

There are four distinct evolutionary outcomes that learners can generate with Frog Pond depending on the programs that they build. The easiest result to achieve is for frogs to get smaller over successive generations. This happens with programs similar to the one shown in Figure 2 (left). The important ingredients are a `hatch` block that allows frogs to reproduce and a `hop` block that moves frogs forward on their lily pads. If a frog hops off a lily pad, the app plays a "splash" sound effect and the frog is removed from

Figure 1. With Frog Pond learners use blocks-based programming to control colorful frogs in a pond.

the simulation. This mechanism creates a selection pressure that favors smaller frogs that take shorter hops. The second outcome results in an opposite effect: the frogs get bigger over successive generations (Figure 2, right). One way to achieve this outcome is to allow frogs to starve to death. Programs then need to include a `hunt` block (which cause frogs to "hunt" for food by sitting and waiting for a fly to pass into their field of view) and conditional logic that causes starving frogs to die. Bigger frogs see farther and have longer tongues, so larger size now conveys a survival advantage. The third outcome extends the second program, but creates a situation with countervailing selection pressures. This balances out the advantage of being large (due to tongue length) with the advantage of being small (due to hop distance). A final outcome extends the second outcome by creating regions in the environment that differentially favor larger or smaller frog populations. This is accomplished by dragging lily pads around the screen to form larger and smaller islands that frogs inhabit.

4. EVALUATION

We produced the current version of Frog Pond through iterative cycles of development and testing in a natural history museum. In our testing sessions, we recruited museum visitors between the ages of 9 and 16. When possible we invited pairs of children to use the software together. Families engaged with the environment for a 15-20 minute period. We collected a variety of data from theses sessions, including a demographic survey, video and audio recordings of the interactive sessions, and researcher field notes.

4.1 Findings

Accessibility and a low-threshold. In testing with our latest version *all* of our users were able to create and run programs within the first 1-2 minutes of encountering the Frog Pond. Of course, many of these early "programs" were extremely simple constructions, serving more to explore the space of functionality of the available blocks, rather than expressing intentions to produce emergent effects or to solve one of the programming challenges. Nevertheless, this rapid progress from introduction to the environment to first program execution in noteworthy, particularly since some of these initial programs actually did produce interesting emergent effects when they were run.

Advantages of the code-first environment. In part because the programming and simulation environments of Frog Pond are merged on a single screen, it was common for learners to attempt to interact with frogs using direct touch or drag gestures. However, the failure of these gestures to produce a response may have cued exploration of particular programming blocks. For instance, after failing to drag a frog that was facing the edge of a lilypad, one user tried using the `right` block as her first programming command. Observing the ways in which participants attempted to manipulate Frog Pond as they learned the logic of the environment provided us with several key insights. For instance, several users dragged a programming block directly onto a frog. Interpreting this gesture as a temporary command (e.g., as a way to preview a block's functionality) might be a useful feature that could support users in learning about the blocks. However, the design of Frog Pond seemed to create a clear distinction between such "one-off" commands on the one hand, and the program blocks that constitute the frogs' nature on the other. For children of a wide variety of ages, the code-first environment seemed to support reasoning about collective behavior through programming and thinking about patterns of behavior as a heritable trait.

Emergence and "runtime surprises". In spite of the increase in transparency in the code achieved through the blocks-based interface, the environment consistently provided "runtime surprises" for users. We attribute these surprises to two causes, each of which carries value for grappling with evolution and its mechanisms. The first of these has to do with the fact that a user's program is executed in a repeat loop. This is closely connected with the idea that the program is a behavioral trait of the organism; it is repeatedly executed and constitutes the frogs' relation to their environment. Because the user creates the program while observing inert frogs in initial locations and orientations, the effects on frogs as these situations change can be a surprise. The second cause for surprise has to do with the idea that the programmatic behaviors are transmitted to offspring. In users' experimentation with the environment, this emerges most clearly in their experience with the `hatch` block. Most of our users initially created programs that did not include reproduction; however, introducing and running a program that included the `hatch` block produced surprised laughter in multiple cases.

Effective engagement with evolutionary phenomena through programming. In our testing to date, we have limited ourselves to

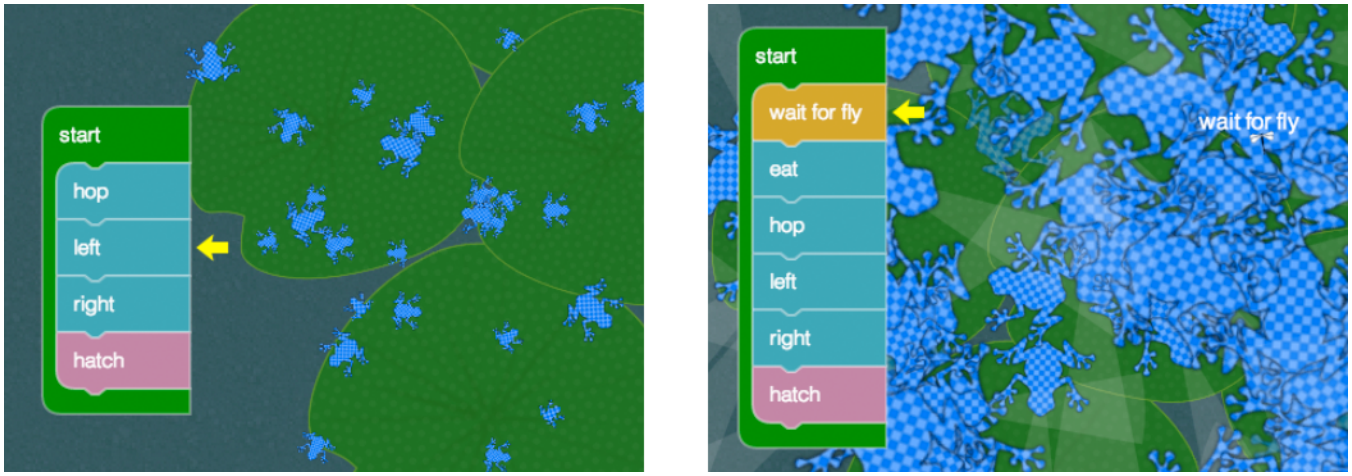

Figure 2. Frog Pond can produce four distinct evolutionary outcomes depending on users' programs: (a) frogs get smaller over time (left); (b) frogs get larger over time (right).

15 to 20 minute sessions. In this time, middle-school aged programming novices were able to produce one or more of the target evolutionary outcomes described in the design section above: most often, the small-frogs and big-frogs outcomes. In assessing the significance of this feature of the environment, we consider not only the emergent phenomena produced by users through their code but also the ways in which Frog Pond motivated reasoning about the mechanisms that led to those emergent effects. This meant that all of our users created or copied a program that yielded a clear change in the distribution of frog body sizes within a few minutes of engaging with Frog Pond. At that point, our users were faced with the challenge of making sense of this emergent result. Here, we saw different levels of success in learner explanations.

5. CONCLUSION AND FUTURE WORK

While our preliminary results are promising there is substantial work left to be done. In particular, we hope to test with a broader audience in a diverse range of settings including elementary school and middle school classrooms. In this way we will be able to understand how far leaners can go with this relatively constrained modeling environment.

6. ACKNOWLEDGMENTS

Amartya Banerjee contributed to this project. This work was supported by the National Science Foundation (grant DRL-1109834). Any opinions, findings, or recommendations are those of the authors and do not necessarily reflect the views of the NSF.

7. REFERENCES

[1] Centola, D., Wilensky, U., & McKenzie, E. (2000). Survival of the groupiest: Facilitating students' understanding of the multiple levels of fitness through multi-agent modeling—The EACH project. *The Interjournal Complex Systems, 337*.

[2] Chi, M.T.H., Kristensen, A.K., & Roscoe, R. (2012). Misunderstanding emergent causal mechanism in natural selection. In K. Rosengren, S. Brem, & G. Sinatra (Eds.), *Evolution Challenges: Integrating Research and Practice in Teaching and Learning about Evolution* (pp. 145-173). Oxford University Press.

[3] Dickes, A.C., & Sengupta, P. (2012). Learning Natural Selection in 4th Grade With Multi-Agent-Based Computational Models. *Research in Science Education,* 1-33.

[4] Horn, M.S., & Wilensky, U. (2011). NetTango [computer software]. Evanston, IL: Center for Connected Learning and Computer-Based Modeling, Northwestern University.

[5] Horwitz, P., McIntyre, C. A., Lord, T. L., O'Dwyer, L. M., & Staudt, C. (2013). Teaching "Evolution readiness" to fourth graders. *Evolution: Education and Outreach, 6*(1), 21.

[6] Kahn, K., Noble, H., Hjorth, A., & Sampaio, F.F (2012). Three-minute Constructionist Experiences. In *Proc. Constructionism*.

[7] Klopfer, E., Scheintaub, H., Huang, W., & Wendel, D. (2009). StarLogo TNG. In *Artificial Life Models in Software*, 151-182.

[8] Levy, S. T., & Wilensky, U. (2009). Crossing levels and representations: The Connected Chemistry (CC1) curriculum. *Journal of Science Education and Technology, 18*(3), 224-242.

[9] Miller, J.D., Scott, E.C., & Okamoto, S. (2006). Public acceptance of evolution. *Science, 313*, 765-766.

[10] National Research Council. (2011). *Report of a Workshop of Pedagogical Aspects of Computational Thinking.* Washington, D.C.: The National Academies Press.

[11] Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas.* New York: Basic books.

[12] Parnafes, O., & diSessa, A. (2004). Relations between types of reasoning and computational representations. *International Journal of Computers for Mathematical Learning, 9*(3), 251-280.

[13] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Sliverman, B. & Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM, 52*(11), 60-67.

[14] Roque, R.V. (2007). OpenBlocks: an extendable framework for graphical block programming systems (Doctoral dissertation, Massachusetts Institute of Technology).

[15] Rosengren, K. S., Brem, S. K., Evans, E. M., & Sinatra, G. M. (Eds.). (2012). *Evolution challenges: Integrating research and practice in teaching and learning about evolution.* Oxford.

[16] Sengupta, P., & Wilensky, U. (2009). Learning electricity with NIELS: Thinking with electrons and thinking in levels. *International Journal of Computers for Mathematical Learning, 14*(1), 21-50.

[17] Sherin, B., diSessa, A., & Hammer, D. (1993). Dynaturtle revisited: Learning physics through collaborative design of a computer model. *Interactive Learning Environments, 3*(2), 91-118.

[18] Wagh, A. & Wilensky, U. (2012). Evolution in blocks: Building models of evolution using blocks. In *Proc. Constructionism 2012*.

[19] Wagh. A. & Wilensky, U. (2012). Breeding birds to learn about artificial selection: Two birds with one stone? In *Proc. International Conference of the Learning Sciences (ICLS'12)*.

[20] Weintrop, D., & Wilensky, U. (2013). RoboBuilder: A Computational Thinking Game. In *Proc. ACM Technical Symposium on Computer Science Education*, 736-736.

[21] Wilensky, U. (1999). NetLogo [computer software]. Evanston, IL: Center for Connected Learning and Computer-Based Modeling, Northwestern University. http://ccl.northwestern.edu/netlogo.

[22] Wilensky, U., & Resnick, M. (1999). Thinking in Levels: A Dynamic Systems Approach to Making Sense of the World. *Journal of Science Education and Technology, 8*(1), 3-19.

[23] Wilensky, U., & Novak, M. (2010). Understanding evolution as an emergent process: learning with agent-based models of evolutionary dynamics. In R.S. Taylor & M. Ferrari (Eds.), *Epistemology and Science Education: Understanding the Evolution vs. Intelligent Design Controversy.* Routledge.

[24] Wilensky, U., & Papert, S. (2010). Restructurations: Reformulations of Knowledge Disciplines through new representational forms. In *Proc. Constructionism 2010*.

[25] Wilensky, U. & Reisman, K. (2006). Thinking like a wolf, a sheep or a firefly: Learning biology through constructing and testing computational theories. *Cognition and Instruction, 24*(2), 171-209.

[26] Wilkerson-Jerde, M. & Wilensky, U. (2010). Deltatick: Using agent-based modeling to learn the calculus of complex systems. In *Proc. Constructionism 2010*.

[27] Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33-35.

[28] Xiang, L., & Passmore, C. (2010). The Use of an Agent-Based Programmable Modeling Tool in 8th Grade Students' Model-Based Inquiry. *Journal of the Research Center for Educational Technology, 6*(2), 130-147.