Anchor Code: Modularity as Evidence for Conceptual Learning and Computational Practices of Students Using a Code-First Environment

Aditi Wagh, Tufts University, aditi.wagh@tufts.edu
Sharona Levy, University of Haifa, stlevy@edu.haifa.ac.il
Michael Horn, Northwestern University, Michael-horn@northwestern.edu
Yu Guo, Northwestern University, yuguo2012@u.northwestern.edu
Corey Brady, Vanderbilt University, corey.brady@vanderbilt.edu
Uri Wilensky, Northwestern University, uri@northwestern.edu

Abstract: In response to increasing calls to include computational thinking (CT) in K-12 education, some researchers have argued for integrating science learning and CT. In that vein, this paper investigates conceptual learning and computational practices through the use of a code-first modeling environment called Frog Pond in a middle school classroom. The environment was designed to enable learners to explore models of evolutionary shifts through domain-specific agent-based visual programming. It was implemented as a curricular unit in seventh grade science class. We analyzed video and log data of two contrasting student pairs. This paper presents one of our findings: Development of modular core functional code-units or what we call *anchor code*. Anchor code is a body of code that creates a stable base from which further explorations take place. We argue that anchor code is evidence for conceptual learning and computational practices.

Introduction and theoretical background

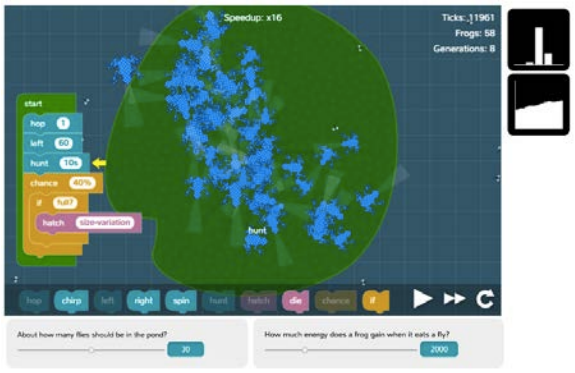
There are increasing calls to integrate computational thinking (CT) into K-12 education (e.g., diSessa, 2000; Weintrop et al., 2016; Wilensky et al., 2014; Wing, 2006). One thrust of this work has been to bring computational tools directly into science classrooms to help learners engage in authentic scientific practices and grapple with difficult concepts (e.g., Papert, 1980; Sengupta et al., 2013; Weintrop et al., 2016). Computation can help enrich science education by bringing tools, practices, and methods that more authentically align with modern science fields. On the other hand, the study of science can provide a context in which computational thinking is powerful.

This paper investigates student learning of conceptual ideas and computational practices around a “code first” (Horn et al., 2014) programming toolkit for adaptation in a middle school biology classroom. Using Camtasia video and computer log data from pairs co-constructing code, we investigate learning about evolutionary change and computational practices. To characterize CT practices, we draw on a taxonomy consisting of computational thinking practices specifically relevant to science and math education developed by Northwestern’s CT-STEM project (Weintrop et al., 2016). The scientific phenomenon we focus on is adaptation. An extensive body of work has shown that programming and computational models can help learners grapple with difficult concepts like natural selection and genetic drift (e.g., Centola, Wilensky, & McKenzie, 2000; Horn et al., 2014; Wagh, 2016; Wagh & Wilensky, 2013). Much of this work has used agent-based models (ABMs). Research has shown that programming agent-based models using graphical, domain-specific primitives (i.e. coding blocks) can help learners develop mechanistic understandings of evolutionary change (Wagh, 2016). This type of understanding is important for learners to move from thinking about evolution as a deterministic, directed process to thinking about it as a decentralized process that emerges from a multitude of events involving interactions between individual organisms.

Frog Pond: An example of a code-first environment

We designed a computer-based learning environment called *Frog Pond* to be used in conjunction with middle school science curriculum on evolution. Frog Pond is an example of a *code-first modeling environment* (Horn et al., 2014). A code-first modeling environment is one in which the primary mode of interaction is through code, it is extremely easy for a learner to create a program within a few minutes or even seconds of using the environment, and diverse outcomes can be observed from a small set of rules. Frog Pond is an agent-based code-first environment that uses a blocks-based interface. It was created using a blocks-based programming environment called NetTango (Horn & Wilensky, 2011) that provides an alternate blocks-based interface to NetLogo (Wilensky, 1999).

In the Frog Pond environment, learners program instructions for a group of frogs in an ecosystem using domain-specific, blocks-based primitives (See Figure 1). There are eight behavioral blocks ("hop", "chirp", "left", "right", "spin", "hunt", "hatch" and "die"), two logic blocks ("if" and "if- else"), and a probability block ("chance"). Students can drag and drop these blocks to construct a program. On running the program, each frog repeatedly enacts the encoded instructions to interact with other frogs and a simulated environment that includes lily pads and flies. Within this environment, variations in frog size have multiple tradeoffs. More information about Frog Pond is available here: http://tidal.northwestern.edu/nettango/. The simulation can result in changes in the frog population: 1) growing bigger or smaller (directional pressure), 2) staying around the same size (stabilizing pressure), or 3) separating into two distinct sub-populations, consisting of larger and smaller individuals (disruptive pressure).


```
start
  hop 1
  left 60
  hunt 10s
  chance 40%
    if full?
      hatch size-variation
```

About how many flies should be in the pond? 30

How much energy does a frog gain when it eats a fly? 2000

Figure 1. A student-generated program.

Frog Pond: The curricular unit

Students took part in a curricular unit driven by an overarching question: Why are there so many different kinds of living things on earth? To answer this question, we asked students to consider real-life examples of adaptation and to explore mechanisms of adaptation by programming virtual frogs in our simulated ecosystem, *Frog Pond*. Students engaged with five increasingly sophisticated challenges through the unit. Each challenge was designed to foreground concepts related to population dynamics and selection pressures. For example, in Challenge 2, students were asked to create a stable population consisting almost entirely of little frogs. The goal of this challenge was to experiment with directional selection pressure—one that drives organisms' traits in one direction over successive generations.

Research question

As they progressed through the curriculum, what forms of learning about evolutionary change and computational practices were visible in the student pairs' programming approach and discourse around code?

Methods

Data collection

We implemented the Frog Pond curriculum at a middle school in an ethnically diverse suburb of a large midwestern city. Nearly 130 students from six seventh-grade science classes participated in the unit over a period of 8 classes. About 100 students consented to participate in the study. The science teacher who usually taught these classes led the activities. Students worked in pairs throughout the unit. Camtasia screen capture recordings were collected from these focal students to capture their on-screen work and conversations. We also video recorded whole class interactions with two stationary video cameras.

Analysis

Video analysis of student pairs

We selected videos from an early and advanced challenge from two focal pairs as contrasting cases for analysis. Pair 1 had succeeded in both challenges while Pair 2 did not succeed in either. This contrast allowed us to compare learning interactions that resulted in different levels of success. We identified segments in which students made *code changes* (added or removed a block) or *code parameter changes* (changed parameter values of a block (e.g., chance %)). We then identified discourse segments before and after each change. These segments provided clues about students' rationale for modifying code or about what students observed when they ran the simulation, and how they accounted for it. These episodes were analyzed to examine themes related to conceptual ideas about evolutionary change, and computational practices from the NU CT-STEM framework.

Computer log analysis of student pairs

Each time a student clicked the Play button, a log entry was generated, recording what blocks were used with what parameters. Across the 5 days of deployment, 12,484 entries of runs were generated. We focused on the analysis of 2585 lines generated by focal students for triangulation. We focused on extracting two key features: Code blocks used in each run, and changes in parameters and blocks used in each run. Below is an example of a log entry: entry:`hop(1);left(60);hunt(10s);chance(40%);if(full?);hatch(no-variation);end;end;` `(1)`

This log entry shows a program composed of 6 blocks with 2 nesting blocks ("if" and "hatch"). Given this information about student programs, we could obtain differences in programs used in sequential runs. We chose to use Levenshtein Distance (LD) to measure this. LD is the minimum number of changes that are needed to make alphabets string identical to the next. We wrote a Python script to convert the original log to a string to obtain meaningful LD between runs.

Findings

Anchor code: Modularity as evidence of conceptual learning and CT practices

Our analysis of learners' code changes in an early and advanced challenge led to the development of a construct that we call *anchor code*. Anchor code refers to a body of code that creates a stable base from which further explorations take place. There were differences in the expression and grain size of anchor code in the two pairs as well as in their quality in stabilizing the system.

For Pair 1, anchor code was located in a set of code blocks that would make the population stable. For instance, when they began Challenge 5, Cory said: "How do we do what we did the one time, the one that was really stable?" They proceeded to construct a set of code that was nearly identical to what they had constructed as part of Challenge 2. Using this code, they attained a stable population that fluctuated around a steady carrying capacity. They then proceeded to make minor modifications to this code in order to meet features of this new challenge. In contrast, for Pair 2, anchor code was of a lower level of modularity and was less stable. Though this pair made several code changes in early and advanced challenges, they came to consistently rely on specific chunks of code to produce specific outcomes. Anchor code was seen in specific strategies using smaller chunks of code to produce specific effects in the model. For instance, Pair 2 did not succeed in stabilizing the population, though they avidly avoided a population explosion and extinction. This suggested that they recognized the importance of maintaining stability in the population, though they did not succeed in doing so through the code alone. Pair 2 used `chance % [die]` and repeatedly modified the `chance%` parameter to maintain stability.

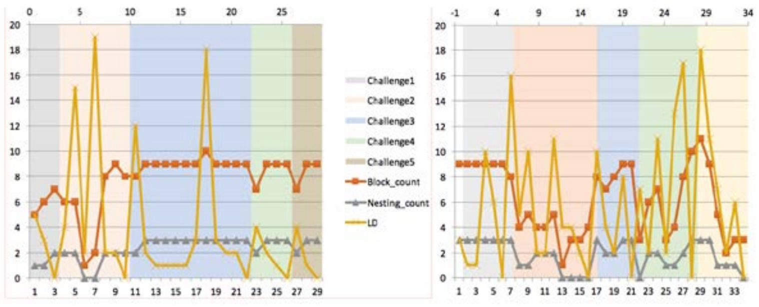


Figure 2. Pair 1's programming to the left, and Pair 2's to the right.

We found evidence for anchor code in the log data. Pair 1's progress in programing (Figure 2, left) showed increasing stability. In early challenges, the pair made radical changes to their code as shown in the high LD peaks. From Challenge 3, they entered a more stable stage of coding. They made one large change and then mainly small continuous tweaks to the code, as shown in the valleys after occasional high peaks. In contrast, Pair 2's programing progress (Figure 2, right) did not have a clear pattern. In Challenge 1, they used almost all of the available blocks. From the second half of Challenge 2, these measures changed without a clear trend.

We see anchor code as evidence of conceptual learning and computational practices. It is conceptual because students' ways of using the anchor code indicated that they had parsed down the challenges into different sub-problems. For instance, pair 1's work indicated that they broke down the challenge into a population stability problem (population dynamics), and a shifting distributions problem (adaptation). Computationally, anchor code aligns with computational problem solving practices related to developing modular computational solutions.

Discussion

Our goal was to explicate forms of conceptual learning and computational practices in Frog Pond, a code-first modeling environment. This paper presents one of our findings related to the development of *anchor code.* We argued that anchor code is evidence of conceptual learning and enactment of computational practices. Conceptually, the emergence of, and student discourse around this stable base of code suggested understandings related to mechanisms underlying maintaining stability in a population, and selection pressures leading to shifts in a population distribution. Though the grain size of their strategies was different, both pairs developed ways of dealing with these two problems in the model. Computationally, anchor code reflects the development of modularity, an important computational practice. This finding has implications for the design of programming environments as well as the design of activities for programming in science classrooms. In future work, we plan to extend these analyses to other student pairs and across challenges to investigate more nuanced shifts in learning of conceptual ideas and computational practices.

References

Centola, D., Wilensky, U., & McKenzie, E. (2000). A Hands-on Mondeling Approach to Evolution: Learning about the Evolution of Cooperation and Altruism through Multi-Agent Modeling- The EACH Project. In *Fourth Annual International Conference of the Learning Sciences*. Ann Arbor, MI.

Disessa, A. (2000). *Changing Minds: Computers, Learning and Literacy*. The MIT Press. Retrieved from https://mitpress.mit.edu/books/changing-minds

Horn, M. S., Brady, C., Hjorth, A., Wagh, A., & Wilensky, U. (2014). Frog Pond: A Codefirst Learning Environment on Evolution and Natural Selection. In *Proceedings of the 2014 Conference on Interaction Design and Children* (pp. 357–360). New York, NY, USA: ACM. https://doi.org/10.1145/2593968.

Horn, M., & Wilensky, U. (2011). *NetTango 1.0*. Evanston, IL: Center for Connected Learning and Computer-based Modeling, Northwestern University.

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. https://doi.org/10.1186/1936-6434-6-21

Papert, S. (1980). *Mindstorms: children, computers, and powerful ideas*. New York, NY, USA: Basic Books, Inc.

Sengupta, P., Kinnebrew, J. S., Basu, S., Biswas, G., & Clark, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies*, *18*(2), 351–380. https://doi.org/10.1007/s10639-012-9240-x

Wagh, A. (2016, March). *Building v/s Exploring Models: Comparing Learning of Evolutionary Processes through Agent-based Modeling* (A dissertation). Northwestern University, Evanston, IL.

Wagh, A., & Wilensky, U. (2013). Leveling the Playing Field: Making Multi-level Evolutionary Processes Accessible through Participatory Simulations. Presented at the CSCL, Madison, Wisconsin, June 15-19: Proceedings of CSCL.

Weintrop, D., Behesti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining Computational Thinking for Mathematics and Science Classrooms. *Journal of Science Education and Technology*, *25*(1), 127–147.

Wilensky, U. (1999). *NetLogo. http://ccl.northwestern.edu/netlogo/.* Evanston, IL: Center for Connected Learning and Computer-based Modeling, Northwestern University.

Wilensky, U., Brady, C. E., & Horn, M. S. (2014). Fostering Computational Literacy in Science Classrooms. *Commun. ACM*, *57*(8), 24–28. https://doi.org/10.1145/2633031

Wing, J. M. (2006). Computational thinking. *Commun. ACM*, *49*(3), 33–35. ttps://doi.org/10.1145/1118178.1118215