JOURNAL OF RESEARCH IN SCIENCE TEACHING

Research Article

Bridging Inquiry-Based Science and Constructionism: Exploring the Alignment Between Students Tinkering With Code of Computational Models and Goals of Inquiry

Aditi Wagh,[1] Kate Cook-Whitt,[2] and Uri Wilensky[3]

[1]*Tufts University, Medford, Massachusetts*
[2]*Thomas College, Waterville, Maine*
[3]*Northwestern University, Evanston, Illinois*

Received 15 May 2016; Accepted 8 November 2016

Abstract: Research on the design of learning environments for K-12 science education has been informed by two bodies of literature: inquiry-based science and Constructionism. Inquiry-based science has emphasized engagement in activities that reflect authentic scientific practices. Constructionism has focused on designing intuitively accessible authoring environments and microworlds that embody the structure of a content domain in program code. Learners build, extend, or explore artifacts to make sense of underlying mechanisms. In this paper, we bridge these bodies of work to argue that interacting with program code of a computational model can facilitate engagement in inquiry-based science. Tinkering with code involves students playfully manipulating the code of a computational model to generate and pursue goals or questions in the model. We use data from video-recorded interviews with eleven 10th grade students in which they demonstrate their tinkering explorations with code of models of biological phenomena, and describe related interactions with other students. We analyze these data using a conceptual framework of inquiry-based science consisting of three components: pursuit of investigations, sense making of investigations, and engagement with a community. We characterize points of alignment between students' tinkering and these components to argue that tinkering with code underlying computational models facilitated engagement in inquiry-based science. We also demonstrate how it provided opportunities for disciplinary engagement in two ways: *Computational engagement* or using code as a representational medium to pursue questions of interest, and *conceptual engagement*, or coming to notice and explain resulting changes in the modeled phenomenon. More broadly, we argue that the constructionist approach of interacting with and manipulating program code of computational models can facilitate productive forms of engagement with inquiry-based science. We discuss affordances of interacting with code as a way to engage in inquiry, and provide design recommendations for the adoption of manipulation of code as an inquiry practice. © 2017 Wiley Periodicals, Inc. J Res Sci Teach 9999:XX–XX, 2017
Keywords: programming; computational modeling; inquiry-based science; constructionism

The design of learning environments for K-12 science education is heavily informed by two bodies of research, inquiry-based science and Constructionism. Both bodies of work date back to the 1960s, and have since progressed rather orthogonally. Inquiry-based science has emphasized facilitating participation in authentic scientific practices to provide opportunities for students to

Correspondence to: A. Wagh; E-mail: aditi.wagh@tufts.edu
DOI 10.1002/tea.21379
Published online in Wiley Online Library (wileyonlinelibrary.com).

engage in the cognitive, epistemic, and social processes underlying professional science (e.g., Chinn & Malhotra, 2002; Grandy & Duschl, 2007; NRC, 2010; Reiser et al., 2001). Inquiry is generally described as generating questions about a scientific phenomenon, making systematic observations and/or conducting experiments to collect data, and interpreting the data to generate evidence-based claims about the phenomenon. The second body of literature, Constructionism, has focused on the design of computational *microworlds* that represent the structure of scientific content in the form of program code in ways that are intuitively accessible and engaging for learners (Edwards, 1995). Learners interact with these microworlds to program and debug models, extend existing models, and/or explore pre-built models. By interacting with these microworlds, learners think through and with program code to make sense of mechanisms underlying the scientific content (Papert, 1980).

Our goal is to bridge between these two bodies of work by arguing that interacting with and manipulating program code can facilitate engagement in inquiry-based science. We make this argument by examining points of alignment between inquiry-based science and students' on-the-fly tinkering explorations with code underlying a computational model. We demonstrate how these explorations provided opportunities for disciplinary engagement in the form of conceptual and computational engagement. Drawing from the literature, we present a conceptual framework of inquiry-based science that abstracts three components of inquiry: pursuit of open-ended investigations, sense-making of investigations, and engagement with the learning community. We use this framework as a lens to examine students' tinkering explorations with code underlying computational models to argue that even simply tinkering with code facilitated engagement in inquiry-based science.

Inquiry-Based Science in K-12 Classrooms

Inquiry-based science is grounded in Dewey's assertion that science learning should be authentic to science practice (1964). Since then, the vision and practice of inquiry-based science in K-12 classrooms has transformed over time (Bybee, 2000). Approaches to fostering inquiry have ranged from highly structured, teacher-prescribed ones to more open-ended, learner-centric forms of inquiry (Herron, 1971; Schwab, 1960). In recent years, more open-ended and learner-centric forms of inquiry reflecting the cognitive, epistemic, and social practices of scientists have been emphasized by the research-based community in science education (Chinn & Malhotra, 2002; Reiser et al., 2001). The "open-ended" aspect of inquiry involves students investigating meaningful questions and problems that are content rich, and do not have a single correct answer or require a single approach to work on them, while the "learner-centric" aspect entails giving students agency their investigations (e.g., Blumenfeld, Krajcik, Marx, & Soloway, 2000; Bransford, Brown, & Cocking, 2000).

Several approaches have been proposed to engage students in open-ended and learner-centric inquiry. These approaches vary in terms of the specific student activities and contexts for inquiry. For instance, one form of inquiry consists of students working on a project in groups to conduct investigations with the goal of answering a question rooted in a real world problem (Krajcik et al., 1998). Students generate their own sub-questions, design, and plan investigations using provided materials or by constructing their own apparatus, select variables to investigate, and then collect data by conducting experiments and making observations. Through this process, students frequently receive feedback from the learner community on the design of their investigations and their findings. Learning by design constitutes another form of inquiry wherein students collaboratively work on a design challenge to build a physical device by participating in iterative interrelated inquiry cycles (Kolodner et al., 2003). The first cycle involves designing their artifact by developing formative plans, getting feedback from the class community, revising their designs,

and actually building them, and then presenting their work to class. The other cycle involves designing and conducting investigations using their artifact, analyzing the results, and sharing their work with classmates. Other approaches to inquiry involve students conducting extensive investigations of a real world system and building physical models of the system for the purpose of conducting their own investigations. These physical models serve as contexts for data generation as students iteratively work on problems to make sense of their initial questions (Lehrer, Schauble, & Lucas, 2008).

Sites for inquiry have also been situated in computer technologies. For instance, students might be given direct access to archival data to record and identify interesting trends that explain the driving question of the unit (Reiser et al., 2001; Tabak & Reiser, 1997), and to develop explanations by drawing on their observations from their investigations and providing causal mechanisms for these observations (Sandoval & Reiser, 2004). Scientific visualizations of archival data have been used to foreground trends and make them more readily avail for inspection and sense making (Edelson & O'Neill, 1994). Computational models of scientific phenomena have been used to provide contexts for students to design and conduct experiments to engage in inquiry (White & Frederiksen, 1998). Inquiry sites have also been provided using scaffolded virtual worlds in which students can record and analyze data to solve real world problems in realistic settings (e.g., Metcalf, Kamarainen, Tutwiler, Grotzer, & Dede, 2011). Finally, computer technologies have been used to engage in model building to represent causal relationships between variables of a system (Jackson & Stratford, 1996; Stratford, Krajcik, & Soloway, 1998).

The goal of presenting these various approaches is to illustrate that contemporary research-based forms of inquiry involve scaffolding students through characteristically open-ended and learner-centric inquiry (NRC, 2000). Students are supported in designing and conducting investigations to work on open-ended questions and develop explanations to account for the phenomenon under investigation. Often, the whole class community plays an important role in students' inquiry process as they get feedback from and share ideas with their teacher and classmates. The rationale that undergirds these diverse approaches is to support students in engaging in practices that reflect the cognitive, epistemic, and discourse processes of real-world scientists (e.g., Chinn & Malhotra, 2002).

Constructionism

Constructionism emphasizes designing *objects* for learning that embody structural ideas of a content domain in ways that are intuitively accessible and engaging to learners. These objects can take the form of *microworlds*, cognitively accessible representations in which the structures and rules constituting a phenomenon are instantiated in the form of program code (Edwards, 1995). Microworlds are representations that "do justice to the powerful logical structure of the subject, but which at the same time mesh properly with the cognitive reality of human beings (diSessa, 1979, p. 239)." Learners interact with microworlds in various ways to discover mechanisms of a phenomenon that is represented in program code.

Learners can also use authoring environments to build or program sharable public artifacts that represent the structures and rules underlying a phenomenon (Papert, 1980). Building a representational artifact requires learners to articulate their prior intuitive conceptions in program code, thereby making these conceptions available for inspection and debugging. This process of inspecting and debugging one's conceptions by assembling and revising program code to represent a phenomenon allows for the generation of refined understandings (Papert, 1980). Learners are supported through this building by easily interpretable feedback from the authoring environment in response to their actions.

Since Logo, research in Constructionism has focused on designing authoring environments that make programming artifacts easily accessible to learners, while enabling a range of constructions. Over the years, construction environments such as NetLogo (Wilensky, 1999), AgentSheets (Repenning & Sumner, 1995), Scratch (Resnick et al., 2009), StageCast Creator (Smith, Cypher, & Tesler, 2000), StarLogo TNG (Klopfer, Yoon, & Um, 2005), and others have been widely used to engage students in building personal artifacts (e.g., Kafai, 1995) as well as models of scientific and mathematical phenomena (e.g., Simpson et al., 2005). These models can often be used as microworlds for learners. A central concern in the design of these construction environments has been to provide primitives[1] that are easily interpretable to learners and yet allow for a range of possible artifact constructions (Simpson, Hoyles, & Noss, 2005; Wilensky, 2003).

Programming scientific models in these construction environments has taken two forms: *phenomena-based modeling* in which learners use the construction units or primitives available in a microworld to replicate a *specific* phenomenon, and *exploration-based modeling* in which learners can experiment with different combinations of available primitives to examine resulting patterns (Wilensky, 2003). Both forms of programming have been used to support learning (e.g., Klopfer, 2003; Louca, Druin, Hammer, & Dreher, 2003; Simpson et al., 2005; Wilkerson-Jerde & Wilensky, 2010).

Besides programming an artifact or model from scratch, learners can also interact with models by building on or adding to the existing code to explore new dimensions of the content or making it more personally meaningful by adding new entities, rules, or variables to the model as well as incorporating additional conditions using "extensible models" (Wilensky, 2003). Learners can also build on "half-baked microworlds" which are less developed representations of a scientific phenomenon, primarily designed to serve as initial starting points or idea generators for learners to extend them (Kynigos, 2007). Finally, learners can also run explorations with pre-built models to discover how the structures and rules instantiated in code result in a phenomenon (Blikstein & Wilensky, 2009; Edwards, 1995; Sengupta & Wilensky, 2009; White & Frederiksen, 1998; Wagh & Wilensky, 2012).

Although inquiry-based science emphasizes making engagement in authentic scientific practices accessible to students, Constructionism focuses on designing objects for learning that embody the structure of the content domain in code and align with learners' existing ways of thinking. Learner interactions with environments involve either directly manipulating program code or by discovering the encoded mechanisms by exploring microworlds.

The constructionist activity that we examine in this paper involves students tinkering with the program code underlying a computational model in NetLogo (Wilensky, 1999) to make changes to a model. NetLogo is an agent-based modeling environment in which users can examine how patterns at the level of the population arise from simple rules and interactions at the level of individuals in a system. On running the model, students can explore how these simple rules can result in complex and often unpredictable patterns at the population level. The underlying program code in NetLogo consists of variables and rules for individuals in a system. In NetLogo, the program code can be easily accessed by clicking on the Code tab available at the top of the model interface.

Tinkering With Code

Tinkering with code involves playful experimentation with minimal planning or a long-term goal (Beckwith et al., 2006), by iteratively making and testing minor changes to code by trial and error (Brandt, Guo, Lewenstein, Dontcheva, & Klemmer, 2009; Dorn & Guzdial, 2010; Law, 1998). It is often initiated and driven by children and involves minimal adult involvement (Petre & Blackwell, 2007). Turkle and Papert (1992) contrasted tinkering and bricolage with more logical

and top-down approaches to working on a task, arguing that both approaches are equally valid and valuable, and share the same status: "Bricolage is a way to organize work. It is not a stage in a progression to a more superior form" (p. 141). Indeed, studies have shown that the guess-and-check processes (Burke & Kafai, 2010; Resnick et al., 2009) involved in tinkering can facilitate sense making of important ideas in a programming language for novices (Perkins, Hancock, Hobbs, Martin, & Simmons, 1986; Turkle & Papert, 1992). Others have found that tinkering activities can serve as a valuable bridge between exploration and refinement of code for novice programmers (Berland, Martin, Benton, Smith, & Davis, 2013).

In this paper, we characterize the ways in which on the fly tinkering with program code of computational models facilitated engagement in inquiry-based science. To do this, we use a conceptual framework of inquiry-based science as a lens to examine students' content-related tinkering explorations. In addition, we describe how tinkering with code provided opportunities for engagement in computational and conceptual disciplinary practices.

Conceptual Framework for Inquiry-Based Science

Earlier in this paper, we presented numerous approaches designed to foster open-ended and learner-centric inquiry in K-12 science classrooms. These approaches have underlying commonalities that can be abstracted to identify some of the core components of inquiry. Indeed, some researchers have made explicit attempts to do so. For instance, Chinn and Malhotra (2002) put forth an evaluation framework to outline the cognitive and epistemic processes underlying authentic inquiry and argued that inquiry-based curricula should incorporate these processes. Quintana et al. (2004) identified three components of inquiry: sense making or testing out hypotheses and interpreting data, managing control of the inquiry process, and articulating what has been learned. They used these components to inform and strategize about design features to support and scaffold students' inquiry processes in a software system. NSES (2000) identifies five important aspects of inquiry: engaging students in scientifically oriented questions, analyzing the evidence with the goal of answering the question, constructing and critiquing explanations that address these questions, comparing and evaluating one's own explanations with other alternative explanations, and communicating and justifying explanations to the learning community.

Drawing on these existing frameworks, we developed a conceptual framework to identify core components of inquiry-based science that are common across these frameworks. These three components are pursuit of open-ended investigations, sense-making of investigations, and engaging with the learning community.

Three Components of Inquiry-Based Science

The first component, *pursuit of open-ended investigations* involves two dimensions: generation or identification of questions relevant to the scientific content under study, and the design and implementation of investigations. Students can develop their own questions or engage in more guided investigations in which they are presented with an overarching goal and supported in developing sub-questions of interest to them. Students design and conduct their own investigations to answer these questions by planning procedures and implementing their investigations to record data that will help answer this question or solve the problem. Students may be supported in their experimental design, data collection, or analysis of the data. The design and implementation of investigations may include processes such as selecting variables to investigate, developing measurement tools, planning experimental procedures, controlling for extraneous variables, creating a data table, and recording data.

The second component constitutes *sense making of investigations*. This entails synthesizing results from investigations, analyzing the data to draw general conclusions about the findings, and

formulating a claim. Synthesizing results and analyzing data involve students noticing and attending to relevant trends in the data that answer investigation questions. An integral aspect of sense making includes making a claim, using evidence to support the claim, and providing mechanistic reasoning that links the evidence and the claim. This process of sense making can subsequently lead to the generation of new questions, and new investigations.

Finally, the third component involves *engagement with the learning community* comprising primarily of the teacher and other classmates, but possibly extending to other teachers or students in different classes. This engagement can take the form of collaborations in which students work with each other to work on similar investigations, or build off of each other's investigations. Students can also get feedback from each other both during various stages of their inquiry projects as well as on their analysis and conclusions as well as engage in a dialog to share and persuade classmates in this community of their claims.

The Study

This paper uses a case study approach situated in a high school Life Sciences classroom. Students in the 10th grade Life Sciences classes were engaged in units on Population Dynamics and Evolutionary Change. As part of the units, students explored NetLogo (Wilensky, 1999) models of population dynamics and evolutionary change embedded in curricular supports designed in the Web-based Inquiry Science Environment (WISE) (Linn, Clark, & Slotta, 2003). Students engaged in guided investigations to develop questions using the models, develop hypotheses, run the models to make observations and record data, and formulate evidence-based explanations for their explorations. The goal was to use the models to make sense of specific mechanisms of population dynamics and evolutionary change. Each unit lasted for approximately 2 weeks. These units were designed as part of a large project in which two computer-based modeling units were implemented in high school science classrooms (NSF #XXXX).

The modeling tool used by students in these units was NetLogo (Wilensky, 1999). NetLogo is a freely available and widely used open-source agent-based modeling platform. Agent-based modeling is a representational system for modeling emergent systems. An agent-based model consists of entities that comprise the system. Each entity is assigned properties and rules that are relevant to the system being modeled. When the model is run, each entity repeatedly follows the encoded rules to interact with other entities in the system. Over time, these interactions result in emergent trends at the population level.

A NetLogo model includes three tabs: the Interface, Info, and Code tabs. The model, accompanying graphs and select parameters for manipulation are displayed on the Interface tab. The Info tab describes the encoded properties and rules assigned to the entities in the model as well as ideas for extending the model. Finally, the Code tab includes all the code for the model. The code tab is used to build the model. Any changes made in the code tab will result in changes to the model.

Toward the end of the two units, the teacher[2] noticed that students were browsing through the Code tab in NetLogo to manipulate program code in their free time in and outside of class. This observation intrigued the teacher because viewing and modifying model code was not a part of the curriculum. Some students were also sharing findings from their tinkering explorations with other students. The teacher followed up with these students to discover that they had been tinkering with code outside of class through multiple days of the units. She reported her observations to the research team. Because interacting with code was student initiated, involved a diverse set of students (see Table 1), and continued for a number of days, a decision was made to closely examine this case and the kinds of changes students were making to the models.

Table 1
Students and their respective backgrounds in biology and programming

Student	Programming Experience	Academic Success in Biology
Sarah	None	High
Stephanie	None	High
Dave	Some	High
Jay	Some	Low
Ricky	Some	Low
Bryan	Some	High
Aaron	Some	Modest
Kyle	Considerable	Modest
Vinay	Considerable	High
Ethan	Considerable	Low
Mike	Considerable	Low

The decision to investigate how students were tinkering with code was made after the class had finished working on the units. This decision was also motivated by the fact that high school biology teachers from other schools participating in the project had reported noticing students modifying model code in their free time in class without any external prompting. We decided to use this site to investigate how students were modifying model code because the teacher had taught the unit for 3 years, and was deeply familiar with the pedagogical underpinnings of the project. Though she had no formal training in programming, she had learned to read NetLogo code, by observing students tinkering with code and through interactions with the research team.

Student Recruitment and Data Sources

Because our goals were to investigate the nature of students' tinkering explorations, we first tried to identify all the students who had modified model code. The teacher followed up with students whom she had noticed tinkering with code, and asked for names of other students who were doing the same. This led to the identification of 11 students, 9 boys and 2 girls, in the class who had modified model code. These students differed in their academic performance in biology and prior experience with programming. Table 1 illustrates students' academic success in biology and self-reported experience with programming. None of these students had programmed in NetLogo before.

Each of these students participated in a semi-structured interview. Ten out of the eleven interviews were audio- and screen-recorded using Camtasia (TechSmith Corporation, 2010). One student, Bryan, did not give consent for audio recording. Hence, in his interview, the teacher typed notes as he described code changes he had made, and recorded screenshots of the outcomes of his code changes. The interviews were conducted between 1 and 3 weeks after the end of the second unit. Each interview consisted of three parts: in the first part, students were asked to identify the models in which they had, in some way, manipulated the code, and briefly describe these code changes. The second part formed the bulk of the interview. Students were asked to select an exploration to demonstrate. Each student was asked to think aloud as s/he demonstrated code modifications s/he had made, and to explain how s/he had expected code changes would impact model outcomes. On running the modified model, s/he was asked to describe whether and how her/his code change impacted model outcomes. Five students demonstrated more than one exploration. Finally, in the third part of the interview, students were asked about their process of

tinkering, including what motivated him/her to engage in tinkering with code, whether s/he had worked with any peers on the code, and so on.

Analysis

Data analysis took place in two parts: identifying the kinds of changes students made by tinkering with the model code, and examining the ways in which their *content-related* explorations aligned with the three components of inquiry identified in the framework.

Kinds of Changes Made by Tinkering With Code. Interviews were open-coded to identify all the changes that students reported having made. Codes were refined and redeveloped through discussion between the first two authors, and based on commonalities agreed upon by both. This resulted in four general categories for kinds of changes. Explorations in which students manipulated content-relevant variables with an intended or consequent goal related to content-specific outcomes were identified as *content-related explorations*. Explorations in which the original or consequent goals were not related to biological phenomena were not coded in this category. For instance, if a student modified how an environmental disturbance, fire, spread in the ecosystem, but did not describe running the model to investigate resulting outcomes, the exploration was not coded as content related.

Besides content-related explorations, students tried to make three other kinds of changes to the models. First, several students made *aesthetic changes* to the model in which they modified visual features of agents in the model such as their color. Students also made *user-related changes* to make the model easier to use such as by adding a monitor to track the count of grass patches in the model or attempting to add a notification to inform the user that a particular species in the model was extinct. Finally, students attempted to *make the model more realistic* by making the environmental disruption of fire in an ecosystem kill bugs instead of killing only the grass, or changing the way fire spread in the model.

In Table 2, we present the kinds of changes each student made. Each "✔" marked in a cell denotes that the student reported having attempted at least one such change to a model. Several students explicitly mentioned having made more than one change to many different models. Because of the quick and iterative nature of this activity, it was often difficult for students to delineate each change. Hence, it was not meaningful or feasible to record a count of number of changes that students had made in each category.

Table 2
Four kinds of changes students made by tinkering with code

	Aesthetic Changes	Making the Model More Realistic	User-Related Changes	Content-Related Explorations
Sarah				✔
Stephanie	✔	✔		✔
Dave	✔		✔	✔
Jay	✔		✔	✔
Ricky	✔		✔	✔
Bryan	✔	✔		✔
Aaron		✔		✔
Kyle				✔
Vinay			✔	✔
Ethan	✔	✔	✔	✔
Mike	✔	✔		✔

Alignment Between Content-Related Tinkering Explorations and Engagement With Inquiry-Based Science. Content-related explorations were identified from the second part of the interview in which students demonstrated an exploration. Content-related explorations that students mentioned but did not demonstrate were not included in the analysis. This resulted in a total of 16 content-related explorations[3] from interviews with nine students. All students reported tinkering with more than one model and five students demonstrated tinkering with more than one model. Though all 11 students engaged in some form of content-related tinkering, 2 students' content-related explorations were not included in this analysis. One of these students reported not remembering the changes he had made, and hence was unable to demonstrate them in detail. This student's interview occurred 2 weeks after the other interviews, and nearly a month after the units. The other student substantially extended a model by incorporating elements from another model in the unit in order to address a conceptual shortcoming in the model he noticed in class. He reported working over an afternoon on a weekend with his father's help on this change. Though this change was content related, it involved a deliberate model extension, and not tinkering on the fly, and hence was not included in the analysis.

For the first two components of inquiry, pursuit of open-ended investigations and sense making, we used student demonstrations of content-related explorations from the interview to identify indicators that provided evidence for participation in them. To do this, we went through segments in the interviews where students demonstrated their exploration to find specific utterances and behaviors that indicated students were pursuing an investigation or working toward noticing and explaining what they found. For the third component, engagement with the learning community, we used data from all 11 interviews, specifically from the third part of the interview, to identify actions involving reaching out to or working with other students in the class. For each of the three components, student utterances and actions in the demonstrations as well as student reported behaviors that aligned with the description of the component was marked as an indicator. We present a compiled list of indicators for each component that we found in the data in Table 3.

Forms of Disciplinary Engagement. Because tinkering involved interacting with code (a computational activity) to observe and explain resulting changes in the model (a conceptual activity), we examined how each exploration provided opportunities for engagement in computational and conceptual practices. Given the quick and iterative nature of students' tinkering explorations, the computational and the conceptual were closely intertwined in students' tinkering. However, for the sake of analysis, we attempted to tease them apart.

By conceptual engagement, we mean coming to notice and explain changes in the biological phenomenon and underlying mechanisms at play in the model. In this case, because the models were related to population dynamics, we expected conceptual engagement to be related to noticing and explaining the underlying mechanics of and relationships between variables in the context of population dynamics. By computational engagement, we mean coming to read and interpret code in the context of the phenomenon being modeled, and manipulate it in a meaningful way, given their goals.

Although coming to notice and explain scientific phenomena have long been seen as important (e.g., Lehrer et al., 2008), researchers and educators are increasingly recognizing the importance of providing opportunities for computational thinking in science education (e.g., Sengupta, Kinnebrew, Basu, Biswas, & Clark, 2013; Wilkerson-Jerde, Wagh, & Wilensky, 2015). Hence, the K-12 science education community values both these forms of engagement as ways of engaging with the discipline.

Table 3
Aligning components of inquiry-based science and tinkering explorations with program code

Inquiry Component	Description	Indicators in Tinkering Explorations
Pursuit of open-ended investigations	Generation and/or identification of questions for investigation Designing and conducting investigations by planning and implementing specific procedures	Students: Generating content-relevant goals or questions Reading through the code Strategically selecting and manipulating variables relevant to their goals Justifying the selection of variables/ predicting the effect of changing a variable on the model outcomes Ignoring variables that do not seem to affect model outcomes
Sense-making of investigations	Analyzing data and synthesizing results to develop claims and generate evidence-base explanations for claims, noticing outcomes, making evidence-based claims, proposing theoretical mechanisms, or conceptual models that help predict and explain outcomes	Noticing outcomes from code changes Running the model several times Comparing observations from different model runs Drawing general conclusions about model outcomes from across different model runs Making a claim about outcomes Postulating a mechanistic explanation for the outcome
Engagement with the learning community	Students collaborating with other students on investigations Arguing from evidence about claims, and attempting to persuade an audience beyond the teacher of their claims	Working together on specific explorations Helping others through their explorations Forming a working group Sharing exciting findings from their tinkering explorations with classmates and the teacher

Findings: Alignment Between Tinkering With Program Code and Engagement in Components of Inquiry-Based Science

Analysis revealed that tinkering explorations with program code facilitated engagement in the three components of inquiry-based science identified in our framework. In addition, these tinkering explorations provided opportunities for engagement in computational and conceptual practices. In what follows, we describe each component, and the opportunities for disciplinary engagement afforded by each exploration.

Component 1: Pursuit of Open-Ended Investigations

By tinkering with code, students engaged in a number of open-ended investigations in which they generated questions or goals of interest to them, and pursued these questions by tinkering with the code for variables and rules in the model. Specifically, students engaged in three kinds of investigations: asking questions of the model, generating questions by experimenting with variables, and attempting to produce a specific outcome. Each of these explorations provided opportunities for computational and conceptual engagement. Computationally, students were

pursuing open-ended investigations by reading and manipulating code as a representational medium. Conceptually, each exploration provided opportunities for noticing and explaining changes in the biological phenomenon being investigated. We describe each kind of exploration with an example below, and describe the kinds of computational and conceptual learning each facilitated.

Asking a Question of the Model. Students began this kind of investigation with a question and tinkered with the code to modify variables and rules that they expected would help answer that question. Four out of nine students demonstrated tinkering with the code to conduct an investigation triggered by a specific question. We present an example from Stephanie's interview.

Example 1: Will Invasive Species Take Over the Bug Species? Stephanie fared well in her biology class, and had never programmed before this unit. One of the models Stephanie tinkered with was Bug Hunt Invaders (Novak & Wilensky, 2011) that modeled a predator–prey ecosystem with birds, bugs, and grass. Students could introduce an invasive species into the ecosystem that survived off grass, the same food source as the bugs.

Stephanie wanted to investigate what would happen to the bugs and invaders species if the invaders had some advantages over the bugs. In the following excerpt, Stephanie describes her exploration and demonstrates specific code changes she recalled having made:

> Stephanie: Basically I made the bug–or the invaders reproduce a lot younger. I made them faster, I don't remember which one is, I think the higher number is faster. And then I made it so that they could basically live a long time, and a few times I made the birds gain less energy from the invaders to see if that would affect if they ate more bugs or not. Um. . .and made their minimum reproducing energy lower.
>
> [*In the Code tab, changes "invader-reproduce-age" from 30 to 5, "invaders-stride" from 0.3 to 0.5, "max-invaders-age" from 100 to 1000, "birds-energy-gain-from-invaders" from 25 to 20, and "min-reproduce-energy-invaders" from 10 to 5.*]
>
> I: Why'd you do that?
>
> Stephanie: So that they could reproduce even if they didn't have like a lot of food around them.
>
> I: Okay, and why'd you–you also made them faster you said? Why did you make them faster?
>
> Stephanie: So that they could get more food if it wasn't like directly around them.
>
> I: Okay, and then the other–the minimum reproducing age, why'd you do that one?
>
> Stephanie: Because they could re–so they could reproduce like sooner than the bugs can.
>
> I: Okay. And then the one as we were just talking you just changed, invaders offspring?
>
> Stephanie: Yeah, the bugs only can have so many offspring before they die, so I made it so that the invaders can have more before they have to die.

Stephanie wanted to give the invader species certain advantages over the bugs to investigate whether the bugs would die out and the invaders take over or if the bugs would survive alongside the invaders. To conduct this investigation, she read through the code

consisting of variables for bugs and invaders (See Figure 1). While comparing variable values for both invaders and bugs, she changed five variables to give the invaders an advantage over the bugs: she made the invaders reproduce faster/younger than bugs and made them move faster than the bugs to enable them to scout more quickly for food. She also gave the invaders a longer life span than bugs, and lowered the minimum reproduction energy of invaders so they could reproduce even if they did not have sufficient food. Finally, she enabled them to have more offspring than bugs before dying. For each of the variables she changed, Stephanie reasoned about its relevance in the context of her exploration. It is also evident that Stephanie's exploration was iterative in that she experimented with different variables and ran the model multiple times ("a few times"). Later in the interview, Stephanie revealed her initial hypotheses, unexpected model outcomes, and how she experimented with different approaches to figure out what was going on.

Stephanie's description of her tinkering exploration aligns with the first component in a few ways. Stephanie generated a content-related question of interest to her and conducted an exploration to answer it by tinkering with the code. She strategically selected invaders' variables relevant to her question and tweaked their values by comparing them to the values of respective variables for the bugs. She manipulated the variables with an initial prediction in mind, and adopted a new approach in the face of unexpected model outcomes. For each of the variables she manipulated, she justified its relevance in the context of her question. Her description also indicates that her exploration was iterative, and she ran the model on different settings multiple times.

```
to setup
  clear-all

  set max-plant-energy 100
  set grass-growth-rate 10
  set sprout-delay-time 25
  set bugs-stride 0.3
  set invaders-stride 0.3
  set birds-stride 0.4
  ;;  a larger birds-stride than bugs-stride leads to scenarios where less birds
  ;;  can keep a larger number of bugs in balance.
  set bug-reproduce-age 20
  set invader-reproduce-age 20
  set bird-reproduce-age 40
  set max-birds-age 100
  set max-invaders-age 100
  set max-bugs-age 100
  set birds-energy-gain-from-bugs 25
  set birds-energy-gain-from-invaders 25
  set min-reproduce-energy-birds 30
  set min-reproduce-energy-bugs 10
  set min-reproduce-energy-invaders 10
  set max-bugs-offspring 3
  set max-invaders-offspring 3
  set max-birds-offspring 2
  set bird-size 2.5
  set bug-size 1.2
  set invader-size 1.5


  set bird-color-region-1 (orange  )
  set bird-color-region-2 (orange - 1)
  set invader-color-region-1 (red - 3)
  set invader-color-region-2 (red - 4)
  set bug-color-region-1 (magenta)
  set bug-color-region-2 (blue )
  set grass-color (green)
  set dirt-color (white)
  set rock-color [216 212 196]
  set-default-shape bugs "bug"
  set-default-shape birds "bird"
  set-default-shape invaders "mouse"
```

Figure 1. Screenshot of part of the code that Stephanie read through to manipulate. The total code of the model is roughly four times the length of this screenshot. Scrolling through the code, Stephanie selected to focus on this part.

The excerpt above indicates that Stephanie's exploration provided opportunities for disciplinary engagement on both computational and conceptual dimensions. Computationally, Stephanie had to read through the code and identify variables that would differentially affect the two species in order to examine conditions under which one species would take over another. Doing so provided exposure to a range of factors that could play into which one would have an advantage. Stephanie had never programmed before so she did not have much prior knowledge to rely on that front. However, her familiarity with the ecosystem modeled in this simulation gave her enough support to read through code, identify, and modify variables that might be relevant to her question. As a result, this exercise provided an opportunity to learn to translate a question of interest about biological phenomena into code. Conceptually, this exploration provided an opportunity for Stephanie to investigate and reason about conditions under which one species would have an advantage over another.

In Table 4, we present other instances of question-driven explorations. Like Stephanie, these students also generated a content-related question to ask of the model on the fly, strategically experimented with variables that they thought were relevant to their exploration and were able to justify their selection. Students typically tried out new approaches in the face of unexpected outcomes.

Experimenting With Variables. In this kind of exploration, students selected and manipulated variables in the code to test their impact on model outcomes, without any targeted question or goal in mind. If the name of a variable sparked interest, students changed its value and ran the model to see how it affected model outcomes. By continued experimenting, students generated questions and goals along the way. Four out of nine students described experimenting with variables in this way. Below, we present an excerpt from Kyle's interview to provide an example of this kind of exploration.

Example 2: Messing With Variables to See What Happens. Kyle had considerable experience in programming and fared modestly in biology class. To demonstrate his exploration, Kyle opened the Bug Hunt Invaders model (Novak & Wilensky, 2011), went to the Code tab, and began perusing through the code. As he started to select variables to manipulate, he explained his process to the interviewer:

> Kyle: Okay. Um, like the ones I'm picking right now, it's mostly random, like–and then I'll just see what all they change, I'll just start picking at random though. But the ones that change the most are usually the ones that I remember the one I try multiple values with. The

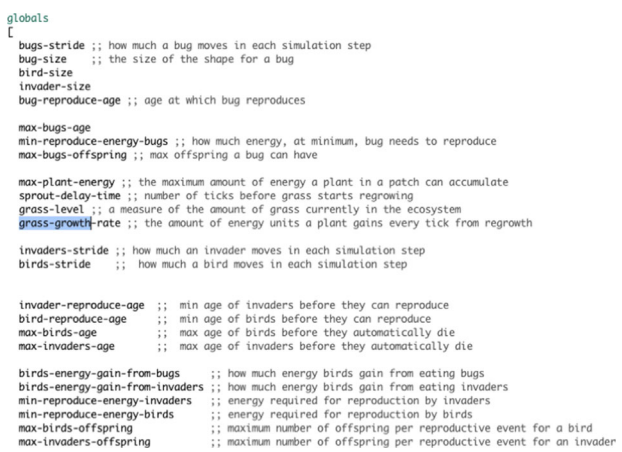
Table 4
Additional examples of question-based explorations

Dave	Question: will the population continue to expand? After setting variables to make the population explode, Dave described running a model overnight to see what would happen to the population over a longer period of time.
Aaron	Question: what would happen if there were multiple disruptions? Aaron described making the program allow for multiple disruptions (fires or disease). Question: what would happen if the fire "started everywhere" instead of from one end of the model? (did not demonstrate this exploration)
Bryan	Question: can the population live forever? Bryan described manipulating code in order to see what might happen to the population of bugs with an unlimited food source and what might happen if there is no food source but bugs do not require energy.

> ones that don't change the simulation much, that don't uh, change the outcomes much, I typically leave those wherever they were.
>
> [*In the code tab, Kyle finds the variable "grass-growth-rate." He goes back to the command line on the model, and enters the command "set grass-growth-rate 100"*]
>
> I: Okay. . .So when you're changing them, what are you looking for?
>
> Kyle: Um, being able to change the way–being able to change the uh, outcomes from running the simulation, being able to like–I remember grass growth rate, that was one that I actually remember that we played with. I think this is it at least. So yeah, I remember the command was set. I should see what it does right now. So I think it was. So it's 10, set, grass growth rate. I think we did things like 100 so like it'd grow really quickly. . .

Kyle described that he randomly picked variables to tweak in the model at first to figure out how they changed outcomes in a model run (See Figure 2). When he noticed variables that visibly changed model outcomes, he tried multiple values with them to see how they changed model outcomes. He reset values for variables that did not change the simulation to their original value. Hence, his initial approach in selecting variables to manipulate was to change the simulation outcomes in *some* way. To demonstrate this approach, he picked "grass regrowth rate," a variable he remembered having tinkered with, and changed its value from 10 to 100. It is noteworthy that his description suggests that he tinkered with this value together with another student to see if it would lead to the grass growing more quickly, "that was one that I actually remember that *we* played with." However, on running the model in the interview, he did not notice a visible change in the model outcome. So he further increased the value of grass-regrowth-rate to 1,000. At this point, the interviewer asked him to describe what his next step would be if changing a variable had impacted the model outcome:

> I: So like what do you think–let's say if you were messing around you got that outcome, what would happen next? Like would you say like "Oh why did that happen?" or would you say like "Cool, let me try something different," or what would you, what would then be the next thought?



```
globals
[
  bugs-stride ;; how much a bug moves in each simulation step
  bug-size    ;; the size of the shape for a bug
  bird-size
  invader-size
  bug-reproduce-age ;; age at which bug reproduces

  max-bugs-age
  min-reproduce-energy-bugs ;; how much energy, at minimum, bug needs to reproduce
  max-bugs-offspring ;; max offspring a bug can have

  max-plant-energy ;; the maximum amount of energy a plant in a patch can accumulate
  sprout-delay-time ;; number of ticks before grass starts regrowing
  grass-level ;; a measure of the amount of grass currently in the ecosystem
  grass-growth-rate ;; the amount of energy units a plant gains every tick from regrowth

  invaders-stride ;; how much an invader moves in each simulation step
  birds-stride    ;;  how much a bird moves in each simulation step


  invader-reproduce-age  ;;  min age of invaders before they can reproduce
  bird-reproduce-age     ;;  min age of birds before they can reproduce
  max-birds-age          ;;  max age of birds before they automatically die
  max-invaders-age       ;;  max age of invaders before they automatically die

  birds-energy-gain-from-bugs     ;; how much energy birds gain from eating bugs
  birds-energy-gain-from-invaders ;; how much energy birds gain from eating invaders
  min-reproduce-energy-invaders   ;; energy required for reproduction by invaders
  min-reproduce-energy-birds      ;; energy required for reproduction by birds
  max-birds-offspring             ;; maximum number of offspring per reproductive event for a bird
  max-invaders-offspring          ;; maximum number of offspring per reproductive event for an invader
```

Figure 2. Part of code Kyle reads through to identify variable that he selects to demonstrate in interview. The entire code of this model is much longer than this screenshot. Reading through it, Kyle decided to demonstrate manipulating this variable.

> Kyle: Uhh, a little bit of both because like I would assume that this because there would be more food regenerating quicker that it'd be able to support a higher population of bugs that would be [inaudible] the shorter conclusion then I'd go ahead and try with different values, may be only 500 if I change it here. And then it would lower down a bit more. Or like what was the value after which point it didn't really matter how much higher you got it if there was like a limit that it didn't affect it anymore or. . .

After having produced *some* outcome in the simulation, Kyle would develop a shorthand explanation for it—in this case, his explanation was that because grass regrowth rate was at a higher value, it would produce food more quickly for the bugs, which in turn would support a higher bug population. This initial explanation would generate follow-up questions such as experimenting with a range of values for the same variable to examine corresponding outcomes, or finding a limit beyond which a change in the variable did not impact model outcomes.

Kyle's tinkering exploration indicates engagement in the first component of the framework, pursuit of open-ended investigations, in a few ways. He adopted a bottom-up approach in which he started by selecting variables for experimentation to produce *some* kind of outcome in the model, strategically used resulting feedback from the model to select variables that produced an outcome, and then pursued new questions and goals in response to that feedback. He focused on variables that produced an outcome and ignored ones that did not seem to produce any change in the model. He generated new ideas for subsequent explorations on the fly as he tinkered with the code: further tinkering with different values for a variable to see their subsequent outcomes or attempting to find the limit for the variable beyond which it did not affect a model's outcome. It is also noteworthy that developing an explanation for his observations after one round of tinkering was an integral part of his process. In other words, in order to continue with experimenting, Kyle needed to make sense of his observations.

We would argue that Kyle's exploration provided an opportunity for him to use computational code as a medium to generate questions as well as for engagement with conceptual practices.

To experiment with variables, Kyle read through code of a fairly complex model to identify variables of interest to him. Though his experimentation was seemingly unsystematic, it presented opportunities for computational engagement. First, he made coarse changes to the code to look for potential shifts, and then made more refined changes to identify more subtle differences resulting from code changes. In doing so, he encountered the idea of threshold values, an important computational idea.

Conceptually, this iterative process led to the beginning of explanations about how the variable change impacted biological phenomenon. This is an important point because the tinkering exploration *necessitated* an explanation to move to the next step. In other words, Kyle's decision about what to do next did not occur in a vacuum—it was informed by his conceptual sense making of impacts of his tinkering.

In Table 5, we present instances from other interviews of these kinds of explorations. These students also messed around with variables of interest in the code, strategically selected ones that produced some kind of outcome in the model (typically outcomes that were easily visible) and ignored ones that did not. Students adopted an iterative approach as they tried out new ideas based on observed outcomes, and generated new goals along the way.

Attempting to Produce a Specific Outcome. These explorations started with students wanting to produce a specific outcome in the model. They perused through the code in search of variables and rules that might be relevant to the desired outcome. On selecting a variable, they (often drastically) increased or decreased its value and ran the model to

Table 5
Additional examples of explorations by experimenting with variables

Sarah	Manipulated speed, energy, and age-related variables for the grass, the bugs, the invaders, and the birds. By changing the speed, energy, and age-related variables, Sarah started questioning what would happen to the populations (Would the populations completely die out? Would the populations take over?).
Ethan	Described looking through the code to find interesting variables. When he found an interesting variable he would change it to see what would happen (Ethan described this practice as one of his approaches to manipulating the code. He did not give specific examples of this approach).
Aaron	Started out tinkering by randomly changing variables to see what it would do to the model, and to see if he could understand what the modeler was trying to do with the model (Aaron described this practices as one of his approaches to manipulating the code. He did not give specific examples).

inspect resulting outcomes. If they did not notice their desired outcome, they returned to the code to change the same variable in the other direction or selected different variables until they obtained their desired outcome. Seven out of nine students demonstrated a total of five such explorations. Below, we present Dave and Jay's investigation undertaken to increase the population size of bugs to over a million:

Example 3: Can We Get a Million Bugs? Dave had some experience coding and fared modestly in biology. In the interview, Dave described himself as someone who was "not good at programming." Dave and Jay manipulated the Bacteria Food Hunt model (Novak & Wilensky, 2015a) in which bacteria could gain energy and reproduce, or lose energy and die with the goal of growing the bacteria population size so that they would have millions of bacteria. In the excerpt below, Dave begins to demonstrate to the interviewer the specific code changes he made to the model in an effort to produce this outcome.

> Dave: Minimum-reproduce-energy-bacteria – I set that to 0 so they just reproduce infinitely (*Laughing*).
>
> [*Changes "minimum-reproduce-energy-bacteria" from 60 to 0 in the code*]
>
> Dave: Max-reproduce-off-spring to 50
>
> I: What was it before?
>
> Dave: 0 – or 2. [*Changes "max-reproduce-off-spring" from 2 to 50*]
>
> Dave: I set bacteria size to 2 but I didn't see a noticeable change for that. [*Changes "bacteria-size" from 1 to 2.*] Amount-of-food-bacteria-eat. I moved that to 500. [*Changes "Amount-of-food-bacteria-eat" from 5 to 500.*]

To produce an outcome of having a million bugs, Dave and Jay drastically altered some of the existing variables: they decreased the minimum reproduction energy of the bacteria to 0 so they would reproduce infinitely, increased the maximum offspring a bacteria could have, changed the bacteria size, and the amount of food they ate. While demonstrating these code changes, Dave also noted that the change in the size of the bacteria had not visibly changed model outcomes.

At this point, the interviewer asked them to explain why they made those code changes.

Dave: Because I wanted to make the amount that they eat higher just because 5 is like more a lower number but it's also more realistic and I just wanted to boost the good stats and lower the bad stats. [*Scrolling down and reading through the code as he speaks*] And then I also just saw this. I set "max-plant-energy" to, I think, a million. [*Changes "max-plant-energy" from 100 to 1,000,000.*]

I: Why did you do that?

Dave: Because (*laughing*) that way they just like continually get energy no matter cause their movement is not gonna like if they get one grass they are gonna be alive for like. . .

I: Ever?

Dave: Yeah pretty much (*Laughing*)

Dave justified changing the variables as an attempt to "boost the good stats and lower the bad stats." That is, he changed variables that would help him meet their goal of producing a million bacteria (e.g., decreasing the amount of energy each bacteria needed to reproduce, increasing the maximum number of offspring each bacteria could have, and increasing the amount of food a bacteria ate). He described increasing the amount of energy each plant provided (patch of grass, in NetLogo parlance) so bacteria can get enough energy to live forever by eating just one unit of grass. He also alluded to the movement of bugs, perhaps because the dramatic spike in variables slowed down the speed of the model considerably, and hence the bacteria appeared to move very little through a model run (See Figure 3). This code change ensured that the bacteria were not disadvantaged by their lack of movement.[4]

Indeed, when Dave and Jay ran the model, the bacteria population increased from 5 to over 100,000 in a few ticks (unit of time in NetLogo), with bacteria appearing in thick clusters due to the model having slowed down considerably, which made them laugh. In the interview, the boys described what caused this outcome as well as why there were fewer bacteria in the right ecosystem than in the left one (See Figure 1).

By reading through the code, Dave and Jay were exposed to underlying computational mechanics of this ecological system. In order to meet their goal outcome, they engaged with these mechanics to think about how to manipulate them to achieve the desirable outcome. Though playfully, in this excerpt, see Dave and Jay reading through the code and making inferences about how specific variables would impact the bacteria population size. The activity they engage in is like engineering. Learning to read through the code to identify and think about what kinds of changes would produce their desired outcome. This computational strand here is intertwined with opportunities for conceptual engagement. Making these changes presented opportunities for them to reason about why specific changes produced an outcome, specific changes did not, and contrast two different ecosystems. It is important to note that here again a frame of explaining what happened was necessary to continue messing about with the code. It was a necessary part of the process.

Dave and Jay's exploration reflects engagement in the pursuit of an open-ended investigation: they began their exploration with an initial goal of producing a million bugs, and carefully selected variables that would enable the desired outcome. When doing this, they took notice of variables that affected or did not seem to affect model outcomes, and justified the relevance of their variable selection in the context of their goals to draw a comparison of the investigation in two different scenarios. Table 6 summarizes other instances of students engaging in these kinds of explorations.

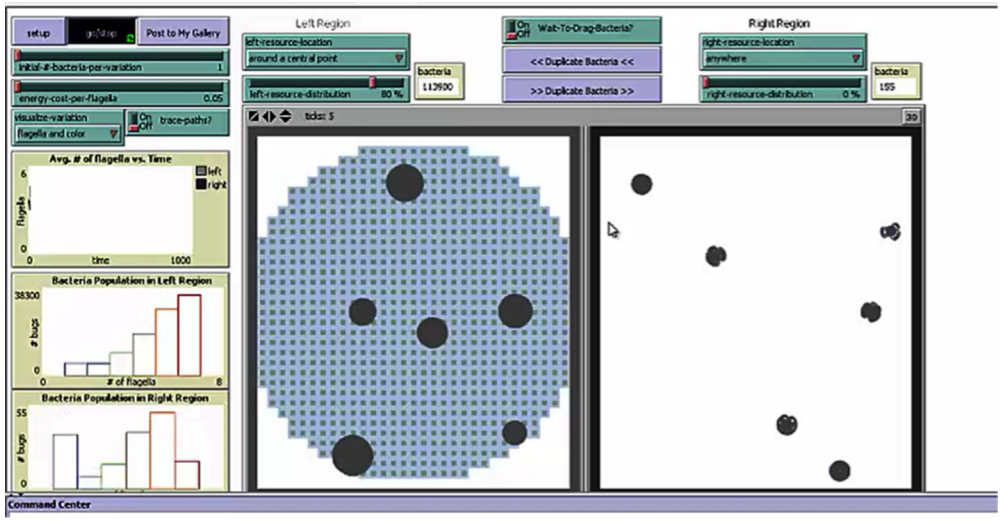


Figure 3. Model outcome from Dave's outcome-based exploration: bacteria reproducing in circles.

Component 2: Sense-Making of Investigations

The second component, sense making of investigations involved noticing patterns and making observations in the model outcomes, making a claim to describe how code changes impacted model outcomes as well as generating some explanation for it. This component was closely intertwined with students generating goals and tinkering with the code. That is, attempting to make sense of the impact of their code change naturally followed from tinkering with code, and deciding next steps. This natural follow-up is evident in the examples already presented. For instance, in Example 1, Kyle developed a shorthand explanation based on his observations, and generated ideas for his next exploration. His explanation suggested evidence of new information about the question, and his pursuing investigation suggested learning how to meaningfully follow through investigations. Kyle fluidly alternated between sense making and designing new investigations. To illustrate a specific instance of students' sense making, we present an excerpt from Aaron's interview.

Example 4: Explaining the Impact of Multiple Disruptions on an Ecosystem. Aaron fared modestly in biology and had some experience programming. One of the models Aaron tweaked

Table 6
Additional examples of outcome-based explorations

Ethan	Goal: to increase the population of the bugs to over a million Changes made: Ethan adjusted the amount of energy bugs gain from grass and the amount of energy required to reproduce
Aaron and Ricky	Goal: to destroy the population of bugs Changes made: enabled the program to set multiple disruptions (fires or disease)
Bryan	Goal: to make the population to live forever Changes made: changed the amount of food bacteria eat in the bacteria model
Sarah	Goal: to "Explode" the population Changes made: adjusted the amount of grass in the model and the energy use of bugs

was the Bug Hunt Disruptions model (Novak & Wilensky, 2015b) in which students could add disruptions such as wild fires or diseases to an ecosystem with bugs and grass. The model only permitted adding a single wildfire or disease in one model run, but Aaron wanted to investigate what would happen if he set multiple wildfires or added multiple disruptions. He tinkered with the code to conduct a question-based exploration to make sense of this. In the interview, Aaron demonstrated this by changing the code, "set left-ecosystem-disrupted? true" to "false" in a button widget on the model interface. Then, he ran the simulation to demonstrate that he could now set multiple disruptions in the ecosystem. Prior to running the model, the interviewer asked Aaron what he had expected multiple disruptions would do:

> I: And did you. . .what did you expect would happen when you did that?
>
> Aaron: I can't remember whether I did them in quick succession or it if it was at regular intervals, but, um, I kind of expected it to like do the same thing. But with the wildfires at least because if you did a wildfire while it was fluctuating and had larger fluctuations and then you did one when it had smaller fluctuations, the one with smaller fluctuations had less of an effect.
>
> [*Starts running the model, and repeatedly clicks on the button to set disruptions. The bug population decreases each time Aaron clicks, but several bugs remain and the population quickly re-grows.*]

Aaron could not recall whether he had set disruptions in rapid succession or at regular intervals, but he had initially expected that both disruptions would impact the ecosystem in similar ways. From his previous explorations, he claimed that there was a relationship between size of fluctuations in the bug and grass populations and the impact of the disruptions. He also claimed that when disruptions were added to the model "when there were more bugs, there was less of an effect," that is, the fire died out quickly.

Later in the interview, Aaron proposed a mechanism for the relationship between size of the bug population and duration of the disruption:

> Aaron: Well, um, what I was thinking was because there's more bugs there's less grass available for the fire to consume because the fire is kind of like. . .both the fire and the grass are getting rid of the — or fire and the bugs are getting rid of the grass. So if there are more bugs there's less grass for the fire to burn.

Aaron claimed that when disruptions were added to the model when there were more bugs, the fire died out more quickly. He postulated an explanation for this relationship: that the more bugs there were in the model, the more grass was being consumed because the bugs ate grass. This resulted in there being overall lesser grass for the fire to burn, which led to the fire dying out more quickly when there were more bugs in the model.

This exploration provided Aaron with opportunities for noticing and making sense of the relationship between fluctuation size and impact of disturbances, one that Aaron had not previously considered. He started the exploration by assuming that each disruption would have a similar impact on the population, an inaccurate hypothesis. Modifying the code and setting this up helped Aaron investigate previously unfamiliar conceptual relationships—thinking about the relationship between size of the fluctuation and impact of the disturbance.

Aaron's tinkering exploration provided opportunities for him to run the model multiple times to make meaningful observations about the impact of multiple disruptions on model outcomes, use evidence from the model to make claims, and propose mechanistic explanations for his claim.

These indicators for sense-making of an investigation were visible in other students' tinkering exploration as well: students sought changes in model outcomes as a result of their tinkering, made observations about these outcomes, attempted to connect the outcomes to code changes by making a claim about their observations, and provided an explanation for the outcome. While the richness of student explanations varied across explorations, every student had pieces of an explanation for how their tinkering exploration impacted model outcomes. Hence, engaging in these tinkering explorations provided a space for students to learn to use code to ask questions of the model, and make sense of new conceptual connections.

Component 3: Engagement With a Learning Community

Data from student interviews indicated that tinkering with code was a student-centric activity that sparked and sustained engagement with a community consisting of other students. We characterize this community as "student-centric" because the tinkering was initiated and sustained by the students. When asked *why* they started tinkering with the code in the interview, students either reported that they started on their own or were motivated by noticing other students tinkering with code. Modifying or even viewing the model's code was not a part of the curriculum. Moreover, none of the students reported modifying code or sharing their modifications with others in response to the teacher's suggestion or prompting.

Within this student-centric community, students shared their work with each other, and helped each other out. Students' interviews indicated that each of the 11 students had engaged with at least one other student in this group to share their observations and/or help each other out. Students reported that while working with others in a group, they sometimes worked on the same explorations, but also worked on different ones. Table 7 illustrates specific sub-groups of students who worked together.

For instance, Vinay, Sarah, and Stephanie worked together on their tinkering explorations in their free time in class. Sarah and Stephanie who had never programmed before, described wanting to tinker with the code after noticing Vinay making code changes in class. Perhaps surprisingly, Vinay, who had experience in programming mentioned taking help in making sense of his exploration from Sarah and Stephanie in some of his explorations.

Kyle reported that Ethan, Aaron, Ricky, and he formed a lunch group to work together in the weeks during the unit. In his interview, he described the role of these lunches:

Table 7
Engagement with the learning community through tinkering with code

Sharing Findings With the Teacher	Student	Collaboration With Classmates
Yes	Sarah	Worked together in class to help each other out
	Vinay	
	Stephanie	
	Dave	Worked together, sometimes on the same explorations, and sometimes on different ones
	Jay	
Not often	Aaron	Worked in the same space over lunch time, independently tinkering, but sharing cool findings
Yes	Ethan	
	Kyle	
Not often	Ricky	
Yes	Mike	Was shown code changes by a group of students Sought help from an expert programmer, his father
	Bryan	Worked with Dave on some investigations

> Kyle: Uhh, every now and then we'd share some of our findings, it wasn't that often though. Like occasionally like if I'd noticed that, I remember from earlier, the population size spiked dramatically. Usually the dramatic changes we'll show to each other, be like "Hey, if you change this variable, the population size goes crazy." So those are the things that we'd share more. These, something like this, I'd probably just explore in greater detail, once I found the limit I might mention it to them.

As Kyle described, students in this group worked in the same space on their own explorations, pursuing in-depth questions of interest to them. Though they were each working on independent explorations, when one of them stumbled onto an exciting finding such as a finding a variable that made the population spike dramatically, he would share it with the others. Hence, the lunch group became a space for these students to pursue and investigate questions of interest to them, and share interesting outcomes of their pursuits with their classmates.

Other students also voluntarily shared findings from their work with one another. For instance, Dave mentioned that he had tinkered with code to increase the bug population and kept it running overnight to see how big the population could get. The next day, he shared his observations with his classmates and the teacher.

These findings indicate that students formed and sustained a community in which they worked together on their explorations and shared observations with each other. The target audience was other students, though some of them informally shared their work with the teacher as well.

We see the formation of this community as noteworthy and important to engagement in inquiry-based science because it was driven by student initiative and interest. Through this activity, students coalesced into smaller groups, helping each other and occasionally sharing their work. The student-driven nature of this activity reflected agency, and authentic engagement around the science. This then marks the productive beginnings of a space, which can be leveraged for engagement in scientific practices such as argumentation in which students share claims and supporting evidence from their investigations with other students.

We set out to characterize the ways in which tinkering with content-related code facilitated engagement in inquiry-based science, and found evidence for engagement in each of the three components. We also demonstrated the ways in which these explorations provided opportunities for disciplinary engagement in the forms of computational and conceptual engagement. Students generated content-related goals or questions on the fly and devised ways to pursue these investigations by manipulating code of their models. Students' tinkering explorations were iteratively designed in that they ran the model multiple times, tried out different approaches in the face of unexpected outcomes, noticed model outcomes, and made claims about their observations and postulated mechanistic explanations for the outcomes. Finally, students engaged with a learning community by sharing findings, collaborating with, and helping their classmates. Based on these findings, we argue that tinkering with the code facilitated engagement with inquiry-based science for these students, and provided opportunities for computational and conceptual disciplinary engagement.

Discussion

Inquiry-based science has focused on engaging students in scientific practices while Constructionism has emphasized "thinking with" code by building, extending, or exploring computational objects. Our goal was to bridge these two bodies to argue that even a simple form of interacting with program code such as playful tinkering can facilitate participation in the goals of inquiry-based science. To make this argument, we examined students' on the fly tinkering explorations using a conceptual framework consisting of three components of inquiry-based

science: pursuit of investigations, sense-making of investigations, and engagement with the learning community.

The analysis revealed that by tinkering with program code, students generated and pursued three kinds of explorations: manipulating variables to see how it would impact model outcomes (Example 1), asking specific questions of the model (Example 2), and attempting to generate specific outcomes in the model (Example 3). In each exploration, students perused through the code, and strategically selected variables and rules with which to tinker. On tweaking model code, they ran the model to examine changes in outcomes. When running the model, students developed claims for how their code changes might have impacted the model. This process was iterative as students also tried out new approaches in the face of unexpected outcomes. Both noticing outcomes resulting from code changes, and constructing candidate explanations for the outcomes were an inherent part of the tinkering process. Students also generated candidate explanations to connect their code changes to respective model outcomes. Finally, tinkering with code initiated collaboration, sharing and a dialog centered about their explorations and findings among this group of students.

This finding that students' tinkering explorations with code facilitated productive forms of inquiry is intriguing and particularly noteworthy because it occurred in the absence of explicitly structured guidance from the teacher or strategic support from the modeling environment.[5] We believe that the very availability of program code representing the scientific content of the model triggered the generation of questions of interest to students, and their explorations and sharing with the community naturally followed from these explorations. The availability of code made explicit the variables and rules underlying the scientific content represented in the model and allowed for the generation and pursuit of a diverse set of explorations which involved "messing" with the code. In addition, because students were deliberately selecting and changing variables, they were able to explain what a variable did, and justify their selection.

We argue that there are unique affordances to tinkering or interacting with program code as a way to engage in inquiry. First, interacting with program code provides students with access to the structures, variables, and rules underlying the scientific phenomenon. While at least some variables are usually available for manipulation on the user interface, perusing through code provides students with access to a wider bandwidth of the conceptual structure of a model. This wider bandwidth allowed for the generation of a diverse set of investigations. In addition, interacting with the code offers the learning opportunity to look inside the model, break it down, and make its underlying structure more transparent (Resnick, Berg, & Eisenberg, 2000). This transparency is likely to have numerous advantages: making students aware of the mechanisms at work and helping them conceptualize a model as a representation of a scientific phenomenon that foregrounds some aspects and backgrounds others. The availability is also likely to make a scientific model feel like a "work in progress" that can be modified, extended, and added to, much like work in scientific inquiry. One student, Aaron, explicitly voiced appreciation for the availability of the code. He said that he "found it really interesting that they left the Code tabs and the editing buttons and the Command line there," and that he "like[d] that." When asked to elaborate, he said that "generally when you give a program to someone, you do not leave all of the development tools there," but he liked that it was there because he was "able to see the code and look at what they were doing."

Besides the very availability of program code enabling the familiarizing and generation of questions, there are other affordances of interacting with code to engage in inquiry. Research has shown that programming and expressing through code helps students think through and learn about mechanisms (Sherin, 2001; Wagh, 2016; Wilensky & Reisman, 2006).

Tinkering with code as a way to facilitate inquiry-based science also has social advantages. In the study, it was a socially pervasive activity through which students initiated shared findings with each other and the teacher, and helped each other out. Perhaps tinkering as a children-centric socialization practice (Petre & Blackwell, 2007) helped to circumvent the authority of the teacher, and connected students to one another. Varying levels of expertise in programming across students, and the teacher, might have further bolstered this community, helping students connect with one another. In science classrooms, researchers strive to create learning communities in which students direct their inquiry and claims to other students as the audience rather than the teacher (Berland & Reiser, 2010), a dynamic that naturally arose in this setting. This dynamic offers promise for using tinkering with code as a way to support authentic disciplinary engagement at an individual and social level.

Recommendations: Manipulating Program Code to Engage in Inquiry

Supporting students in interacting with program code in science classrooms is a design issue that requires making the code available for manipulation, and, designing activities that value and support experimentation with code.

Hoyles, Noss, and Adamson (2002) distinguish between the *platform* and *superstructure* level of a microworld. The platform is "the base level at which it is possible for users (rather than professional programmers) to interact" (p. 9). On the other hand, the superstructure level "describes the objects in the microworld and ways to manipulate them." The authors assert that the platform level should be "intrusive" (p. 9) so that its structures and rules are visible at the superstructure level. However, they add that making the platform level directly accessible enables these structures and rules to become available for inspection and manipulation. The availability of the platform level has been a central design feature of many constructionist technologies including NetLogo. Here, we are arguing for making code available in a way that invites and supports students (many of whom may be novice programmers) in science classrooms to engage in interacting with the code. We can infer clues from commonalities across students' tinkering explorations in this study to do this.

The platform level could constitute a *tinker-friendly* segment of code containing properties and rules of structures in the model that are inherently more interesting or relevant to mess around with than others. In particular, a tinker-friendly code segment can be designed based on three guidelines for tinkerability put forth by Resnick and Rosenbaum (2013): immediate feedback, fluid experimentation, and open exploration.

To provide immediate feedback, the platform level can contain variables and behaviors that quickly produce (or not, if that is expected to elicit surprise like it did for Stephanie in Example 2) interesting outcomes in the model. In our study, this theme repeatedly emerged in students' tinkering explorations: students tweaked variables that produced visible outcomes, and ignored ones that did not. The second principle, "fluid experimentation" suggests that tinkerable environments should allow for quick iterations, and rapid generation and refinement of ideas. To facilitate quick generation of ideas, properties and rules directly related to learners' model investigations at the superstructure level could be included in the available code. This would allow learners to leverage insights from their model investigations to ensure a seamless transition to tinkering with code. For instance, several students in our study tweaked energy variables of the agents and grass, as well as rules related to reproduction and death, which were related to their curricular activities. Subsequently, their tinkering explorations focused on the conceptual aspects of computational modeling rather than the technical aspects of programming. To follow the third design principle, open exploration, this tinker-friendly code segment could allow for a diverse set of investigations to allow variability across explorations. We would emphasize that the

tinker-friendly segment need not only support content-related explorations. Instead, it could support a sampling of a spectrum of kinds of changes learners make, including changes to personalize their model by changing shapes and colors as well as making it easier to use.

Conclusions

Our goal was to bridge the inquiry-based science and constructionist bodies of literature to argue that interacting with program code underlying computational models can facilitate engagement in inquiry-based science. To make our case, we analyzed instances of data students demonstrating on the fly tinkering explorations with code using a framework for inquiry-based science. Findings revealed that even tinkering with code facilitated engagement in dimensions of inquiry-based science. In addition, tinkering with code provided opportunities for disciplinary engagement along two dimensions: computational engagement by using code as a representational medium to conduct scientific investigations, and conceptual engagement by coming to observe and trying to make sense of how their code changes impacted the phenomenon being modeled. We discussed this alignment and proposed recommendations for incorporating interacting with program code as an inquiry practice in K-12 science classrooms.

Notes

[1]Primitives are units of construction or building blocks in a construction environment or microworld. It is what learners use to build a program.

[2]The teacher is the second author of this paper.

[3]Two students, Ricky and Aaron, collaborated on and jointly conducted one of these 16 explorations.

[4]This code change reflected a lack of his understanding of the notion of "time" in a NetLogo model, and that it was asynchronous with real time.

[5]Model code in NetLogo is usually commented as is good coding practice and names of variables are easily interpretable. Though this would have supported students, it was not explicitly designed with this goal for the units.

References

Beckwith, L., Kissinger, C., Burnett, M., Wiedenbeck, S., Lawrance, J., Blackwell, A., & Cook, C. (2006). Tinkering and Gender in End-user Programmers' Debugging. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 231–240). New York, NY, USA: ACM. https://doi.org/10.1145/1124772.1124808

Berland, M., Martin, T., Benton, T., Petrick Smith, C., & Davis, D. (2013). Using learning analytics to understand the learning pathways of novice programmers. Journal of the Learning Sciences, 22(4), 564–599. https://doi.org/10.1080/10508406.2013.836655

Blikstein, P., & Wilensky, U. (2009). An atom is known by the company it keeps: A constructionist learning environment for materials science using multi-agent simulation. International Journal of Computers for Mathematical Learning, 14(1), 81–119.

Blumenfeld, P., Krajcik, J., Marx, R. W., & Soloway, E. (2000). Promising new instructional practices. In New Teachers for a New Century. San Francisco: McCutchan.

Brandt, J., Guo, P. J., Lewenstein, J., Dontcheva, M., & Klemmer, S. R. (2009). Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1589–1598). New York, NY, USA: ACM. https://doi.org/10.1145/1518701.1518944

Bransford, J. D., Brown, A. L., & Cocking, R. R. (2000). How people learn: Brain, mind, experience, and school: Expanded Edition. National Academy Press. Retrieved from http://www.nap.edu/openbook.php?isbn=0309070368

Burke, Q., & Kafai, Y. B. (2010). Programming & Storytelling: Opportunities for Learning About Coding & Composition. In *Proceedings of the 9th International Conference on Interaction Design and Children* (pp. 348–351). New York, NY, USA: ACM. https://doi.org/10.1145/1810543.1810611

Bybee, R (2000). Teaching science as inquiry. In J. Minstrell & E. V. Zee (Eds.), Inquiring into inquiry in science learning and teaching (pp. 20–46). Washington, DC, USA: AAAS.

Center for Science, Mathematics, and Engineering Education. (2000). Inquiry and the national science education standards: A guide for teaching and learning. Washington, DC: National Academy Press.

Chinn, C. A., & Malhotra, B. A. (2002). Epistemologically authentic inquiry in schools: A theoretical framework for evaluating inquiry tasks. Science Education, 86, 175–218.

Dorn, B., & Guzdial, M. (2010). Learning on the Job: Characterizing the Programming Knowledge and Learning Strategies of Web Designers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 703–712). New York, NY, USA: ACM. https://doi.org/10.1145/1753326.1753430

Edelson, D. C., & O'Neill, D. K. (1994). The CoVis Collaboratory Notebook: Supporting collaborative scientific enquiry. Conference Proceedings—National Educational Computing Conference, Boston, MA, June 13–15 1994, pp. 146–152.

Edwards, L. D. (1995). Microworlds as representations. In A. A. diSessa, C. Hoyles, R. Noss, & L. D. Edwards (Eds.), Computers and exploratory learning (pp. 127–154). Berlin Heidelberg: Springer. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-57799-4_8

Grandy, R., & Duschl, R. A. (2007). Reconsidering the character and role of inquiry in school science: Analysis of a conference. Science and Education, 16(2), 141–166.

Herron, M. D. (1971). The nature of scientific enquiry. School Review, 79(2), 171–212.

Hoyles, C., Noss, R., & Adamson, R. (2002). Rethinking the microworld idea. Journal of Educational Computing Research, 27(1), 29–53.

Jackson, S. L., & Stratford, J. S. (1996). Model-It: A case study of learner-centered design software for supporting model building. Interactive Learning Environments, 233–257.

Kafai, Y. B. (1995). Minds in play: Computer game design as a context for children's learning. New York, NY: Routledge.

Klopfer, E. (2003). Technologies to support the creation of complex systems models-using StarLogo software with students. Biosystems, 71(1–2), 111–122. https://doi.org/10.1016/S0303-2647(03)00115-1

Klopfer, E., Yoon, S., & Um, T. (2005). Teaching complex dynamic systems to young students with StarLogo. Journal of Computers in Mathematics and Science Teaching, 24(2), 157–178.

Kolodner, J. L., Camp, P. J., Crismond, D., Fasse, B., Gray, J., Holbrook, J., . . ., Ryan, M. (2003). Problem-based learning meets case-based reasoning in the middle-school science classroom: Putting learning by design™ into practice. The Journal of the Learning Sciences, 12(4), 495–547.

Krajcik, J., Blumenfeld, P. C., Marx, R. W., Bass, K. M., Fredricks, J., & Soloway, E. (1998). Inquiry in project-based science classrooms: Initial attempts by middle school students. The Journal of the Learning Sciences, 7(3/4), 313–350. https://doi.org/10.2307/1466790

Kynigos, C. (2007). Using half-baked microworlds to challenge teacher educators' knowing. International Journal of Computers for Mathematical Learning, 12(2), 87–111. https://doi.org/10.1007/s10758-007-9114-2

Law, L.-C. (1998). A situated cognition view about the effects of planning and authorship on computer program debugging. Behaviour & Information Technology, 17(6), 325–337. https://doi.org/10.1080/014492998119283

Lehrer, R., Schauble, L., & Lucas, D. (2008). Supporting development of the epistemology of inquiry. Cognitive Development, 23(4), 512–529. https://doi.org/10.1016/j.cogdev.2008.09.001

Linn, M. C., Clark, D., & Slotta, J. D. (2003). WISE design for knowledge integration. Science Education, 87(4), 517–538. https://doi.org/10.1002/sce.10086

Louca, L., Druin, A., Hammer, D., & Dreher, D. (2003). Students' collaborative use of computer-based programming tools in science. In B. Wasson, S. Ludvigsen, & U. Hoppe (Eds.), Designing for change in

networked learning environments (pp. 109–118). Netherlands: Springer. Retrieved from http://link.springer.com/chapter/10.1007/978-94-017-0195-2_15

Metcalf, S. J., Kamarainen, A., Tutwiler, M. S., Grotzer, T. A., & Dede, C. J. (2011). Ecosystem science learning via multi-user virtual environments. International Journal of Gaming and Computer-Mediated Simulations, 3(1), 86–90.

Novak, M., & Wilensky, U. (2011). NetLogo Bug Hunt Predators and Invasive Species model. Evanston, IL: Center for Connected Learning and Computer-Based Modeling, Northwestern University. http://ccl.northwestern.edu/netlogo/models/BugHuntPredatorsandInvasiveSpecies

Novak, M., & Wilensky, U. (2015a). NetLogo Bacteria Food Hunt model. Evanston, IL: Center for Connected Learning and Computer-Based Modeling, Northwestern University. http://ccl.northwestern.edu/netlogo/models/BacteriaFoodHunt

Novak, M., & Wilensky, U. (2015b). NetLogo Bug Hunt Disruptions model. Evanston, IL: Center for Connected Learning and Computer-Based Modeling, Northwestern University. http://ccl.northwestern.edu/netlogo/models/BugHuntDisruptions

NRC. (2000). Inquiry and the National Science Education Standards: A guide for teaching and learning. Washington, DC, USA: National Academies Press. Retrieved from http://www.nap.edu/openbook.php?record_id=9596

NRC. (2010). Report of a Workshop on The Scope and Nature of Computational Thinking. Retrieved September 2, 2013, from http://www.nap.edu/openbook.php?record_id=12840

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

Perkins, D. N., Hancock, C., Hobbs, R., Martin, F., & Simmons, R. (1986). Conditions of learning in novice programmers. Journal of Educational Computing Research, 2(1), 37–55.

Petre, M., & Blackwell, A. F. (2007). Children as unwitting end-user programmers. In *IEEE Symposium on Visual Languages and Human-Centric Computing* (pp. 239–242).

Quintana, C., Reiser, B. J., Davis, E. A., Krajcik, J., Fretz, E., Duncan, R. G., . . ., Soloway, E. (2004). A scaffolding design framework for software to support science inquiry. Journal of the Learning Sciences, 13(3), 337–386. https://doi.org/10.1207/s15327809jls1303_4

Reiser, B., Tabak, I., Sandoval, W., Smith, B., Steinmuller, F., & Leone. (2001). BGuILE: Strategic and conceptual scaffolds for scientific inquiry in biology classrooms. In: Cognition and instruction: Twenty-five years of progress (pp 263–305). Mahwah, NJ: Erlbaum.

Repenning, A., & Sumner, T. (1995). Agentsheets: A medium for creating domain-oriented visual languages. Computer, 28(3), 17–25. https://doi.org/10.1109/2.366152

Resnick, M., Berg, R., & Eisenberg, M. (2000). Beyond black boxes: Bringing transparency and aesthetics back to scientific investigation. The Journal of the Learning Sciences, 9(1), 7–30.

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., . . ., Kafai, Y. (2009). Scratch: Programming for all. Communications of the ACM, 52, 60–67.

Resnick, M., & Rosenbaum, E. (2013). Designing for tinkerability. In M. Honey & D. E. Kanter (Eds.), Design, make, play: Growing the next generation of STEM innovators (pp. 163–181). New York, NY: Routledge.

Sandoval, W. A., & Reiser, B. J. (2004). Explanation-driven inquiry: Integrating conceptual and epistemic scaffolds for scientific inquiry. Science Education, 88(3), 345–372. https://doi.org/10.1002/sce.10130

Schwab, J. (1960). Inquiry, the science teacher, and the educator. The School Review, 68(2), 176–195.

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

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

Sherin, B. (2001). A comparison of programming languages and algebraic notation as expressive languages for physics. International Journal of Computers for Mathematical Learning, 6(1), 1–61. https://doi.org/10.1023/A:1011434026437

Simpson, G., Hoyles, C., & Noss, R. (2005). Designing a programming-based approach for modelling scientific phenomena. Journal of Computer Assisted Learning, 21(2), 143–158. https://doi.org/10.1111/j.1365-2729.2005.00121.x

Smith, D., Cypher, T., & Tesler, L. (2000). Stagecast creator. California.

Stratford, S. J., Krajcik, J., & Soloway, E. (1998). Secondary students' dynamic modeling processes: Analyzing, reasoning about, synthesizing, and testing models of stream ecosystems. Journal of Science Education and Technology, 7(3), 215–234.

Tabak, I., & Reiser, B. J. (1997). Domain-specific Inquiry Support: Permeating Discussions with Scientific Conceptions. In *Proceedings of From Misconceptions to Constructed Understanding*. Ithaca, NY.

TechSmith Corporation. (2010). Camtasia [Computer Software].

Turkle, S., & Papert, S. (1992). Epistemological pluralism and the revaluation of the concrete. Journal of Mathematical Behavior, 11(1), 3–33. TechSmith Corporation (2010). Camtasia [Computer Software].

Wagh, A., & Wilensky, U. (2012). Mechanistic Explanations of Evolutionary Change Facilitated by Agent-based Models. Presented at the AERA, Vancouver, April 13–17.

Wagh, A. (2016). Building v/s exploring models: Comparing learning of evolutionary processes through agent-based modeling (A dissertation). Evanston, IL: Northwestern University.

White, B. Y., & Frederiksen, J. R. (1998). Inquiry, modeling, and metacognition: Making science accessible to all students. Cognition and Instruction, 16(1), 3–118.

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

Wilensky, U. (2003). Statistical mechanics for secondary school: The GasLab modeling toolkit. International Journal of Computers for Mathematical Learning [*Special Issue on Agent-Based Modeling*], 8(1), 1–41.

Wilensky, U., & Reisman, K. (2006). Thinking like a wolf, a sheep, or a firefly: Learning biology through constructing and testing computational theories—An embodied modeling approach. Cognition and Instruction, 24(2), 171–209.

Wilkerson-Jerde, M., Wagh, A., & Wilensky, U. (2015). Balancing curricular and pedagogical needs in computational construction kits: Lessons from the DeltaTick project. Science Education, 99(3), 465–499. https://doi.org/10.1002/sce.21157

Wilkerson-Jerde, M., & Wilensky, U. (2010). Restructuring change, interpreting changes: The DeltaTick modeling and analysis toolkit. In J. Clayson & I. Kalas (Eds.), Proceedings of the Constructionism 2010 Conference. Paris, France. https://doi.org/Aug10-14