J Sci Educ Technol
https://doi.org/10.1007/s10956-017-9713-1

# EvoBuild: A Quickstart Toolkit for Programming Agent-Based Models of Evolutionary Processes

**Aditi Wagh**[1] · **Uri Wilensky**[2]

© Springer Science+Business Media, LLC 2017

**Abstract** Extensive research has shown that one of the benefits of programming to learn about scientific phenomena is that it facilitates learning about mechanisms underlying the phenomenon. However, using programming activities in classrooms is associated with costs such as requiring additional time to learn to program or students needing prior experience with programming. This paper presents a class of programming environments that we call *quickstart*: Environments with a negligible threshold for entry into programming and a modest ceiling. We posit that such environments can provide benefits of programming for learning without incurring associated costs for novice programmers. To make this claim, we present a design-based research study conducted to compare programming models of evolutionary processes with a quickstart toolkit with exploring pre-built models of the same processes. The study was conducted in six seventh grade science classes in two schools. Students in the programming condition used EvoBuild, a quickstart toolkit for programming agent-based models of evolutionary processes, to *build* their NetLogo models. Students in the exploration condition used *pre-built* NetLogo models. We demonstrate that although students came from a range of academic backgrounds without prior programming experience, and all students spent the same number of class periods on the activities including the time students took to learn programming in this environment, EvoBuild students showed greater learning about evolutionary mechanisms. We discuss the implications of this work for design research on programming environments in K-12 science education.

**Keywords** Technology · Science education · Computer modeling · Evolution

Computational modeling is one of the core disciplinary practices in K-12 science education (NGSS Lead States, 2013). One form of computational modeling is having learners engage in programming to model mechanisms underlying a scientific phenomenon in the form of code. Programming to model scientific phenomena involves learners iteratively constructing and debugging a program that instantiates mechanisms that underlie it. This has been found to be a powerful way of coming to appreciate and understand mechanisms underlying phenomena (e.g., Papert 1980; Sherin 2001; Simpson et al. 2005; Wagh 2016; Wilensky 1999a; Wilensky and Reisman 2006).

As learners engage in programming, the materials they use to construct the program include primitives in the environment. The form and grain size of these primitives is an important consideration in design (e.g., Simpson et al. 2005; Wilensky 2003). The nature of primitives plays an important role in influencing perceptions and learning (Weintrop 2015), activating relevant forms of intuitive knowledge learners can draw on (e.g., Bamberger 2001; Papert 1980), how they plan and break down the task (Louca and Zacharia 2007), and so on.

**Electronic supplementary material** The online version of this article (https://doi.org/10.1007/s10956-017-9713-1) contains supplementary material, which is available to authorized users.

✉ Aditi Wagh
aditi.wagh@tufts.edu

Uri Wilensky
uri@northwestern.edu

[1] Department of Education, Tufts University, Paige Hall, Medford, MA 02155, USA

[2] 2120 Campus Drive, Evanston, IL 60208, USA

Published online: 23 October 2017

This manuscript presents a class of programming environments that we call quickstart environments. Primitives in these environments combine visual programming with orientation to curricular domains. We use the descriptor "quickstart" to highlight how quickly and effortlessly even novice programmers are able to begin programming in these environments. However, these environments are limited in the extent to which sophisticated models can be built with them. These features make these environments particularly felicitous for short-term, one-off uses in classrooms with programming novices. In this manuscript, we present an example of a quickstart environment called EvoBuild and present a design-based research study to demonstrate that such environments can provide benefits of programming associated with learning about mechanisms without incurring costs related to spending additional time to learn programming.

## Range of Primitives Used in Programming Environments for K-12 Education

Programming environments for K-12 education include various forms of primitives. Some programming environments provide text-based primitives that are general-purpose and do not orient specifically to a single discipline. Some of the earliest work on programming for K-12 education was done using Logo—an environment that provides text-based primitives for modeling mathematical and scientific phenomena (Papert 1980). Primitives in Logo were designed to draw on learners' body syntonic knowledge to lower the threshold for entry into programming. Though Logo primitives had embedded structures of turtle geometry, they were not constrained to a specific disciplinary domain and were used to program simulations of other scientific and mathematical phenomena (e.g., Harel & Papert, 1991). Hence, Logo can be classified as a general-purpose text-based programming environment. Another example of a similarly classified environment is NetLogo (Wilensky 1999b). NetLogo inherits elements of turtle geometry from Logo. It is also designed for representing complex emergent systems and includes primitives to represent individual-level rules and interactions. Many of the primitives in NetLogo also draw on learners' knowledge about their bodies and interactions in the world, thereby lowering the threshold for learning to program. However, NetLogo primitives are also not targeted towards a specific disciplinary domain and are general-purpose.

Other environments provide primitives that are semantically general-purpose but visual in form. Many of these environments provide graphical primitives in the form of blocks as objects to manipulate and combine in different ways to write a program. Some examples of such environments are Scratch (Resnick et al. 2009), StarLogo TNG (Klopfer et al. 2005), DeltaTick (Wilkerson et al. 2015; Wilkerson and Wilensky 2010), ViMAP (Sengupta et al. 2013), BehaviourComposer (Kahn 2007b; Kahn and Noble 2010), Visual AgenTalk in Agentsheets (Repenning and Sumner 1995), and NetTango (Horn and Wilensky 2011). Some environments such as Stagecast Creator (Smith et al. 1996) involve graphical programming without blocks. In general, the graphical nature of primitives in these environments makes it possible for novice programmers to begin programming more easily as compared to in a text-based environment.

Finally, one class of programming environments provides domain-specific primitives that combine graphical programming with orientation to a curricular domain. These primitives serve as "micro-behaviors," "small, coherent, and independent program fragments" relevant to a target curricular domain (Kahn 2007a, p. 931). They are designed to align with learners' ways of thinking to reduce the distance between novices and the curricular domain (e.g., Bamberger 2001; Rader et al. 1998). Like other graphical blocks-based environments, these toolkits reduce syntax issues with a visual, drag-and-drop interface. In addition, by providing a constrained library of primitives that can be combined in different ways to model the core dynamics of a target curricular domain (Wilkerson et al. 2015), they further substantially lower the entry associated with programming for use in K-12 science classrooms.

The forms of primitives carry costs and benefits related to use in K-12 classrooms (e.g., Ioannidou et al. 2003). For instance, some of the costs associated with using general-purpose text-based environments include the time investment required for teaching programming to novices (Xiang and Passmore 2010) or requiring some prior experience with programming. Though these environments can be low threshold environments (e.g., Logo), they require some initial investment of time in learning to code. When amortized over the many potential uses of programming in STEM and across all subjects, these costs can be minimal. The general-purpose nature of these primitives offers greater flexibility and potential for sophistication in model construction. This allows for the benefit of a high ceiling that can enable students to program increasingly sophisticated models in the environment over several class periods. Indeed, prior work has shown that text-based programming can be a powerful way to learn science (e.g., Blikstein and Wilensky 2009; Kafai et al. 1997; Louca and Zacharia 2007; Wilensky 1999a; Wilensky and Reisman 2006). However, when considered for short-term use, the costs associated with time investment can be more significant.

Increasing the adoption of programming activities in science classrooms is an important goal. There has been abundant research on the affordances of engaging in programming to articulate and debug one's understandings of a phenomenon. Programming has been found to facilitate learning about

mechanisms and structures underlying a phenomenon (Sherin 2001; Wagh 2016; Wilensky and Reisman 2006), accommodating multiple approaches (Turkle and Papert 1992), and engaging in collaboration (e.g., Bruckman 1997). Interacting with and manipulating code also supports simultaneous engagement in conceptual and computational disciplinary practices (Wagh, Cook-Whitt & Wilensky, 2017).

However, these costs and benefits have implications for ease of adoption of programming in K-12 science classrooms. Individual teachers are often making choices about whether to incorporate programming activities into a single class session or a single unit. These choices weigh costs and benefits to decide whether and how to include programming activities in their classrooms. In the absence of more systematic adoption, costs associated with time investment are likely to deter short-term adoption of such activities or adoption with novice programmers in K-12 science classrooms. For short-term one-off uses in classrooms with novice programmers, environments that combine graphical programming with domain-specific primitives can provide an alternative. Though these environments have a much lower ceiling, they can provide an easy entry into programming and might encourage subsequent further investment.

We characterize these toolkits as *quickstart* because they allow even novice programmers to quickly and easily assemble code to construct models. By doing so, they enable learners to attend to conceptual issues related to representing the mechanics of the phenomenon instead of dealing with the technical aspects of programming. The descriptor "quickstart" is intended to capture the entry point into programming for novices in these environments: Novice programmers can begin assembling code within minutes. On the other hand, because they provide a small set of blocks-based primitives, these environments are unlikely to support building sophisticated models. This combination of an immediately accessible programming environment that is relatively less powerful for more sophisticated uses makes these environments particularly felicitous for short-term uses in classrooms. Some of the environments previously mentioned have been used to develop quickstart toolkits. For instance, BehaviourComposer has been used to create Epidemic Game Maker, a domain-specific toolkit for modeling epidemic disease (Kahn et al. 2012). NetTango has been used to construct Frog Pond for modeling adaptation in middle school classrooms (Horn et al. 2014). Similarly, DeltaTick has been used to design toolkits to model population dynamics in high school classrooms (Wilkerson and Wilensky 2010) and evolutionary processes in middle school classrooms (Wagh and Wilensky 2014). A common strand across these examples is their ease of accessibility for modeling phenomena through short-term interventions with little or no training in programming.

The potential of programming to support learning about mechanisms makes it a particularly exciting approach for learning about phenomena with complex mechanisms. This manuscript examines programming in a quickstart environment for programming agent-based models of evolutionary processes. This is a particularly promising domain for programming because reasoning about underlying evolutionary mechanisms is challenging for learners. Despite this, much of the work done using an agent-based modeling infrastructure for evolutionary processes has involved students exploring pre-built models of these processes. In what follows, we briefly review this work.

## Agent-Based Modeling for Micro-Evolutionary Processes

A substantial body of research has investigated various design approaches to support learning about the mechanisms underlying evolutionary change. Given the focus on agent-based modeling (ABM) in this study, we briefly describe work using ABMs for learning about evolutionary change.

Much of the research on using ABM to facilitate learning about evolutionary processes has involved students investigating pre-built models to conduct experiments by manipulating parameters, observing and explaining resulting trends. These model investigations have included students engaging in model investigation activities in a progressively complex sequence of models representing natural selection, drift, coevolution, and so on (Wagh and Wilensky 2012b; Wilensky and Novak 2010). Investigations with agent-based models have also been combined with case studies from real world systems to draw parallels between the modeled and the physical system (Wagh et al. 2016). Students analyzed trends from data from real world ecosystems and moved back and forth between making insights in ABM and relating those findings to the data. Investigations have also included the use of participatory agent-based simulations in which students "enter" a model by enacting the role of an agent in it (Wagh and Wilensky 2012a, 2013; Wilensky and Novak 2010). Pre-built models have also been used for guided interventions with a researcher to seed beginnings of ideas related to evolutionary change with elementary school students (Dickes and Sengupta 2013).

Model investigations have also taken more hybrid forms in which students examine or manipulate underlying model code. For instance, in a researcher-led small group intervention, undergraduate students participated in a trajectory beginning with exploring pre-built models, modifying code of existing models, and finally, building their own models to make sense of social evolutionary patterns (Centola et al. 2000; Wilensky and Centola 2007). In interventions at the high school level, viewing and examining the blocks-based code underlying StarLogo Nova models while conducting experiments was prompted and encouraged (Yoon et al. 2016).

However, very little work has been done with students programming their own models of evolutionary shifts in K-12 teacher-led classes. One study that involved students programming their own models took place in a researcher-led after-school workshop in which middle school students wrote text code from scratch or by modifying code from other models in NetLogo. The authors reported that the workshop took a considerable amount of time: Students first learned to code in NetLogo (over 15 h in a few weeks), and then spent another 15 h programming two models of natural selection (Xiang and Passmore 2010). The authors concluded that though programming was an evocative context for students to articulate their understandings about natural selection, distractions due to syntax errors in code gave short shrift to sophisticated reflections mapping their constructed model to natural selection. This observation reflects the costs associated with using general-purpose text-based programming described earlier.

Facilitating the adoption of programming activities in science classrooms is an important goal. Yet, as previously described, this can be tricky in terms of balancing time constraints of a classroom and training students who may be novice programmers. This results in a design tension between leveraging the value of programming particularly for sense making of mechanisms while also dealing with issues related to the feasibility of adoption in classrooms such as lack of prior programming experience, and time to learn programming. This tension between costs and benefits of programming is particularly evident in short-term interventions with novice programmers. We would argue that this tension could be mitigated by quickstart toolkits that, by providing a constrained library of graphical domain-specific primitives, make programming models immediately accessible for even novice programmers.

As mentioned previously, this manuscript has two goals. One goal is to present an example of a quickstart toolkit called EvoBuild. EvoBuild allows students to program agent-level mechanisms of evolutionary processes using domain-specific graphical primitives (Wagh and Wilensky 2014). EvoBuild was designed using DeltaTick (Wilkerson et al. 2015; Wilkerson and Wilensky 2010), a blocks-based programming interface for NetLogo (Wilensky 1999b). Our second goal is to demonstrate that even with novice programmers who did not spend additional time to learn programming, building their own models in a quickstart environment helped them experience benefits of programming associated with learning about mechanisms. To do this, we present a design-based research study (Collins et al. 2004) conducted to compare processes and outcomes of learning in two modalities of computational modeling, model building, and model exploration. As part of the study, we developed two agent-based modeling units on evolutionary processes for middle school students. The model-exploration unit, called EvoExplore, provided microworlds (Edwards 1995) or pre-built NetLogo models for students to manipulate relevant parameters to uncover and make sense of encoded mechanisms. The model building unit consisted of EvoBuild.

## EvoBuild: a Quickstart Toolkit for Programming Agent-Based Models of Evolutionary Processes

EvoBuild is a toolkit for designing accessible programming activities for students to construct agent-based models to

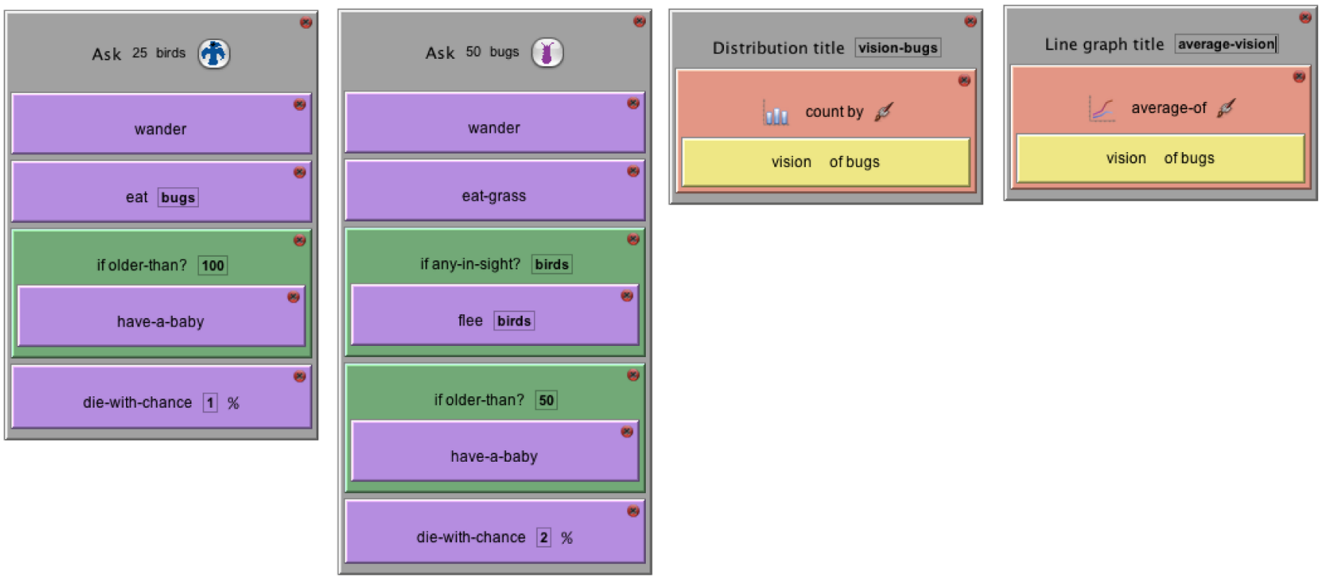


**Fig. 1** Code for a model representing two species, birds and bugs, and their respective rules. The model also includes two plots, a histogram tracking vision range of bugs and a line graph tracking the average vision range of bugs

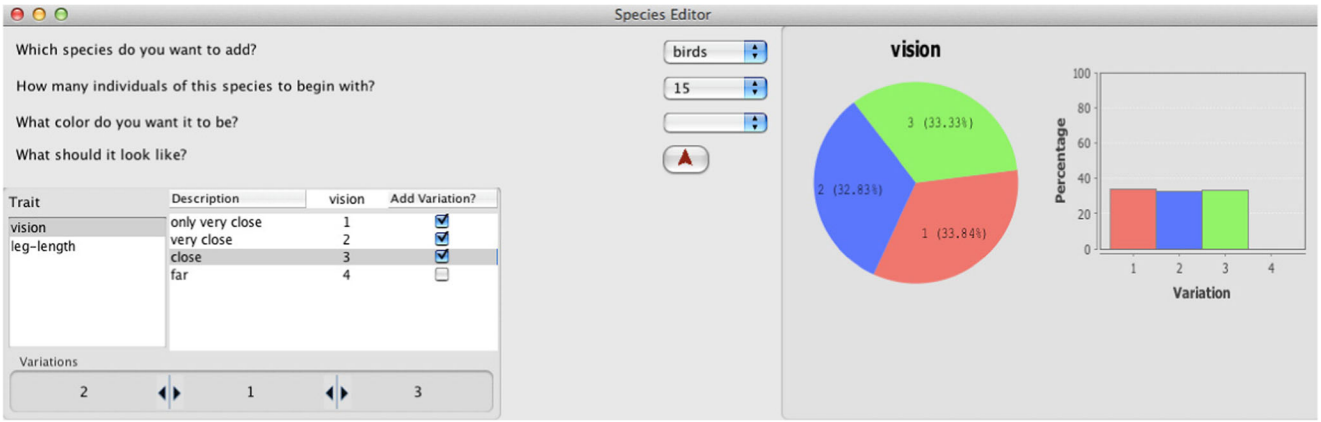


**Fig. 2** Available traits and selected variations in the Species Inspector in EvoBuild

represent and examine evolutionary processes (Wagh and Wilensky 2014). It draws on the infrastructure of DeltaTick (Wilkerson et al. 2015; Wilkerson and Wilensky 2010), a blocks-based programming interface for NetLogo. The blocks-based infrastructure allows designers and educators to provide students with a library of pre-defined domain-specific primitives in the form of blocks.

Each block, as a domain-specific primitive, constitutes an autonomous but self-contained fragment of code representing a conceptually relevant rule for agents in the system (Kahn 2007a). The code for each block is pre-defined in an XML file, a commonly used format for web-based files. A collection of these micro-behaviors forms a "conceptual library space," which is a set of micro-behaviors that can be assembled in different combinations to recreate and explore the relevant conceptual space (see Fig. 1). In addition to the blocks-based primitives to be made available, designers can also specify the kinds of breeds or collection of entities of a particular kind that should be made available and the maximum number of individuals in each breed, and available properties, and variations in the XML file (see Fig. 2).

A student can begin programming by loading a specific XML file into the environment. When an XML file has been loaded, the species, properties, and primitives pre-defined in the file become available to students in the programming environment. Students can use these to build a model. To begin programming their model, students can add one or more species to the model. To model variations within a species, learners add one or more properties for agents of a breed. These properties are called *traits*, and different values of a trait are called *variations*.

Learners can add one or more traits to a breed in their model. For each trait, they can pick variations to be included in the initial population at setup. For instance, in Fig. 2, two traits are seen available in the Species Inspector. The student has added the trait, vision to the breed, bugs to determine how far a bug can see. For this trait, three variations have been selected. These variations will be evenly distributed in the initial population at the start of a model run. This distribution can be manipulated.

## Programming a Model Using Agent-Based Domain-Specific Primitives

Once species and their properties have been initialized, students can specify rules for individuals of these species to follow as the simulation runs.

The screenshot above (Fig. 1) shows a model, which contains two breeds called "birds" and "bugs." There are 25 birds and 50 bugs at the start of the model. When the model is run, at each tick,[1] every bird and bug will follow commands encoded in their respective blocks. In this model, at each tick, each bird and bug will wander around the world. A bird will eat a bug when it is right next to it. Each bug will eat grass. When a bird is older than 100 ticks, it will have a baby, and it has a 1% chance of dying at each tick. If a bug notices a bird within its vision range, it turns away from it. Each bug can have a baby if it is older than 50 ticks, and it has a 2% chance of dying. The model also includes two plots to track the population, a histogram to track the vision distribution in bugs, and a line graph to track the average vision range of the bug vision.

## Running the Model

On the Run tab, students can play the model (See Fig. 3). They can also manipulate the chance of a mutation occurring in the specific trait. For instance, in the screenshot above, students can manipulate the change of a mutation occurring in the vision trait inherited by a bug offspring. At its current value, each offspring born in a model run has a 50% chance of having a slight variation in its vision range from its parent.

[1] In a NetLogo model, a tick denotes a unit of time.

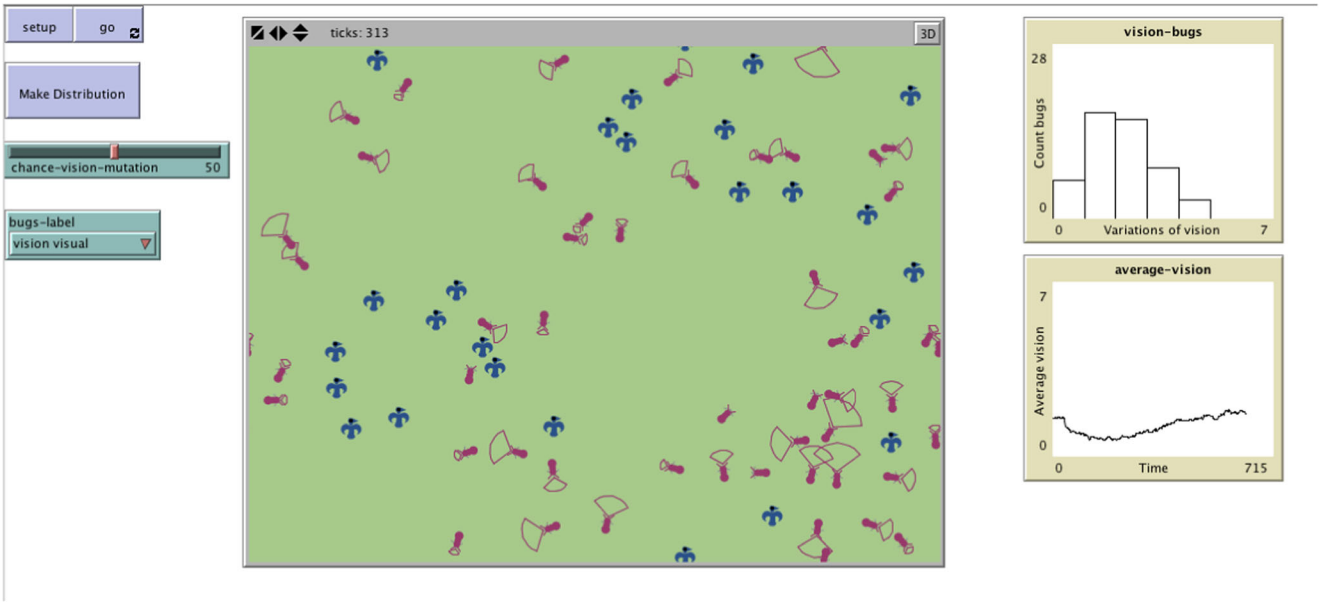


**Fig. 3** A model run from the code in Fig. 1

When the model is run, bugs with a greater vision range have a survival advantage because they are able to get away from the predatory birds. Hence, these bugs are more likely to live until 50 ticks and have an offspring as compared to ones with a shorter vision range. In addition, as the model runs, new variations in vision range appear in the bug population.

### Supports for Viewing the Changing Distribution

Finally, students can also view an "agent" distribution by a specific trait within the model world (See Fig. 4). By clicking on a button, students could line up agents based on their specific variations for the trait. This helps students track change in the size of specific groups and view agents themselves more carefully. This feature was in the spirit of representations in environments such as TinkerPlots (Konold and Miller 2005) that supports students in viewing the spread of specific variations in the population.

Over the last few years, we have engaged in iterations of design research (Collins et al. 2004) using EvoBuild activities in middle school science classes. In parallel, we have also engaged in iterations of design of a model exploration unit called EvoExplore (Wagh 2016). EvoExplore consists of modified NetLogo models and curricular materials from

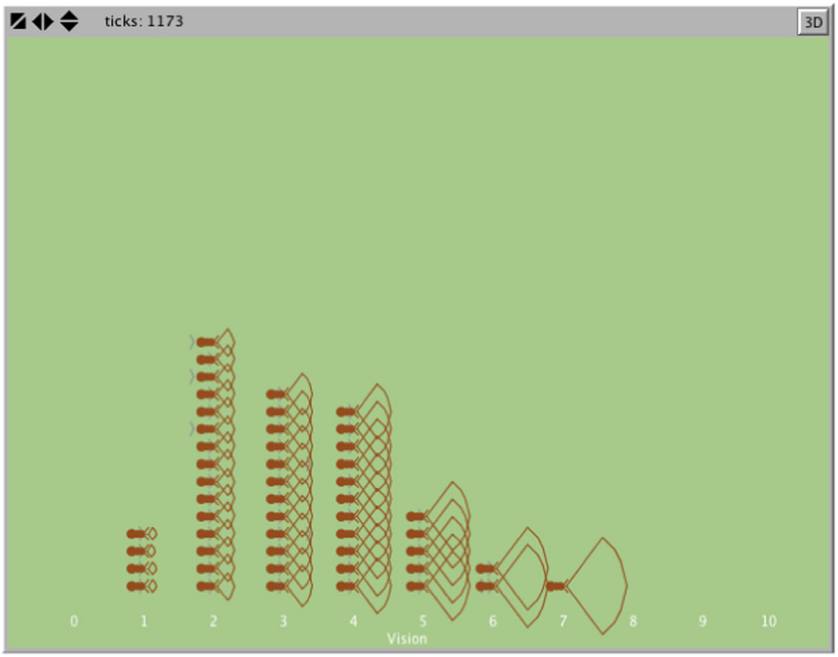


**Fig. 4** An agent distribution of vision range of the bug population

existing curricula (Wilensky and Novak 2010). In each activity, students investigated a pre-built model by manipulating parameters to observe and explain resulting changes in the population. Like EvoBuild, these models also allowed students to view agent distributions in a model run. The main difference between the two modalities was that EvoBuild students programmed the model by using a constrained library of domain-specific graphical primitives. EvoExplore students ran investigations in a pre-built model, and did not view or manipulate code.

Having described EvoBuild, we now turn to the second goal of this manuscript. We present the study we conducted to compare processes and outcomes of learning in the two modalities of model building and model exploration. We draw on this study to show that benefits of programming associated with learning about mechanisms were visible even when associated costs associated such as requiring additional time were minimized.

## The Study: Comparing EvoBuild and EvoExplore

The EvoBuild and EvoExplore curricula were implemented in seventh grade science classes in two schools,[2] Highland and Forest Park. The study also included a control condition in which students used physical manipulables instead of computational modeling environments. This condition was only implemented at Highland. The focus of this paper on computational modeling and space constraints prevent a discussion of the control condition in this manuscript. Two classes participated in each modality at Highland—by the teacher's account of student grades and performance on standardized tests, one class in each modality performed at seventh grade level, and the other class was performed below grade level. At Forest Park, one class participated in each modality.

### Activities and Teacher Support

In each modality, the curriculum consisted of four activities (see Table 1). The first activity was an introductory activity consisting of a whole class discussion about variations of traits within a species. This activity was the same for both conditions. The discussion was intended to elicit ideas about the spread of variations and was used to frame the central question of the unit: Does the spread of variations for a trait change over time?

The following three activities were anchored in this central question. In the EvoBuild condition, students were provided with a scenario consisting of information about one or more species in an ecosystem, their life cycle, and relevant traits. They worked in pairs to build a model to represent this scenario. Students debugged and ran the model to investigate resulting trends in the distribution of the property. In the EvoExplore condition, students were provided with pre-built NetLogo models that, when run, simulated specific evolutionary process. Students worked in pairs to manipulate parameters, and run the model to observe and explain resulting changes in the population.

**Table 1** Four activities in the unit

| Activity | Content |
|---|---|
| Activity 1 | Introduction to traits and variations |
| Activity 2 | How selection pressures change populations (natural selection) |
| Activity 3 | How populations can change simply due to chance (drift) |
| Activity 4 | How populations adapt to their environments (adaptation) |

Both EvoBuild and EvoExplore curricula were presented to teachers as instances of computer modeling curricula that were designed by the research team. The first author of the paper facilitated workshops for both teachers during after-school hours and provided supporting materials for both curricula.

### Data Collection

The study was designed to investigate how the modalities compare in terms of processes and outcomes of learning. To investigate processes of learning, we recorded Camtasia videos of focal student pairs as they worked on their models, and collected worksheets from all students. In each class, two researchers were present to take field notes and assist with data collection. The first author was present in each class at both schools. The second researcher varied on each day. A short debriefing was done with the second researcher after each class to capture his/her observations. These notes were incorporated in the field notes. In addition, the first author conducted teacher interviews to capture teacher observations of students' work in the two modalities. Interview questions did not ask teachers to specify which modality was better or to directly compare the modalities in any way. Instead, they focused on capturing teacher observations about how students' engagement and thinking in each modality.

To assess learning outcomes, we administered pre- and posttests to all students and conducted pre and post interviews with focal students. The written assessment consisted of scenarios of micro-evolutionary change. Questions were designed to elicit reasoning about how distributions of variations of a trait might change over

[2] The names of schools are pseudonyms.

**Table 2** Student demographics and teacher experience in the two schools (Illinois State Board of Education, 2014 and field notes)

| | Highland Middle School | Forest Park School |
|---|---|---|
| Number of students | 101 (62 students in EvoBuild; 59 students in EvoExplore) | 48 (22 in EvoBuild; 26 in EvoExplore) |
| Programming experience | None reported | Several students had some experience with programming |
| Seventh grade students' performance on ISAT standardized tests (Illinois State Board of Education, 2014) | Science<br>70.1% met and 13.2% exceeded standards<br>Math<br>43.1% met and 1.1% exceeded standards<br>Reading<br>42% met and 10.9% exceeded standards | Science<br>44.9% met and 46.9% exceeded standards<br>Math<br>65.3% met and 20.4% exceeded standards<br>Reading<br>49% met and 38.8% exceeded standards |
| Socioeconomic status and racial/ethnic composition of students | 81.7% from low-income households<br>Primarily Hispanic (79.5%), 10.2% white and others | 17.4% from low-income households<br>58% white, 21% Hispanic, 8% Asian, and 4% African American |
| Teacher background | No experience with programming<br>No experience using computer-modeling activities in class<br>Eighth year of teaching at the school<br>(Certified to teach social science and science) | Some experience with HTML programming<br>Experience using PheT simulations in class<br>First year teaching middle school science |

time due to natural and sexual selection, drift, and/or adaptation. Two forms of assessments were developed. Each question in the two forms was conceptually mapped (e.g., included selection pressures or not) but presented different scenarios with different organisms. Half the students in each modality were administered one form at pre and the other at post. Questions about distribution shifts in the assessment have been provided in Appendix A from one of the forms as a sample.

In this manuscript, we demonstrate that even with novice programmers who did not spend additional time to learn programming, building their own models in a quickstart environment helped them experience benefits of programming associated with learning about mechanisms. Specifically, we draw on field notes, teacher interviews, and pre and posttests to show that (1) the student sample included students from a range of academic backgrounds, many of who had no experience with programming; (2) EvoBuild students took the same amount of time as EvoExplore students including time taken to learn to program in this environment; and (3) EvoBuild students more frequently provided causal evolutionary mechanisms in their posttest responses.

## Student Sample from the Two Schools

As seen in Table 2, of the 101 students who participated in the two modalities at Highland, none of the students in either modality reported having programmed before, and many of them reported never even having heard of programming. As mentioned previously, one class in each modality performed below grade level. This class also included students from a special education program. Highland primarily consisted of a Hispanic student population (80%). About 81% of students came from low-income households. Of the 48 students from Forest Park, a sizeable group of students in both modalities had some experience with programming in school. The student population at Forest Park was largely white (58.7%) or Hispanic (21.6%). About 17% of students at this school were from low-income households.

Neither of the science teachers were experienced programmers. The Highland teacher had never programmed before and reported never having used computer-based modeling activities in her classes. This was her eighth year teaching in the school. The Forest Park teacher had experience with HTML programming several years before the study and had occasionally used PheT simulations in his class. This was his first year teaching science in middle school. Both teachers participated in teacher workshops and had access to teacher guides for the two curricular units.

## EvoBuild Students Spent as Many Class Periods on the Activities as EvoExplore Students

In both schools, EvoBuild students spent the same number of class periods for the entire curricular unit as the EvoExplore students. This included time spent on learning to use the software. Ensuring that the units took an equivalent amount of time was a decision made at the start of the study. Hence, when students in either condition fell behind in relation to the other, the teacher decided when to wrap up the activity, and proceed to the next one. Here, we draw on field observations and video data to present a detailed breakdown of how time was spent in the EvoBuild and EvoExplore conditions.

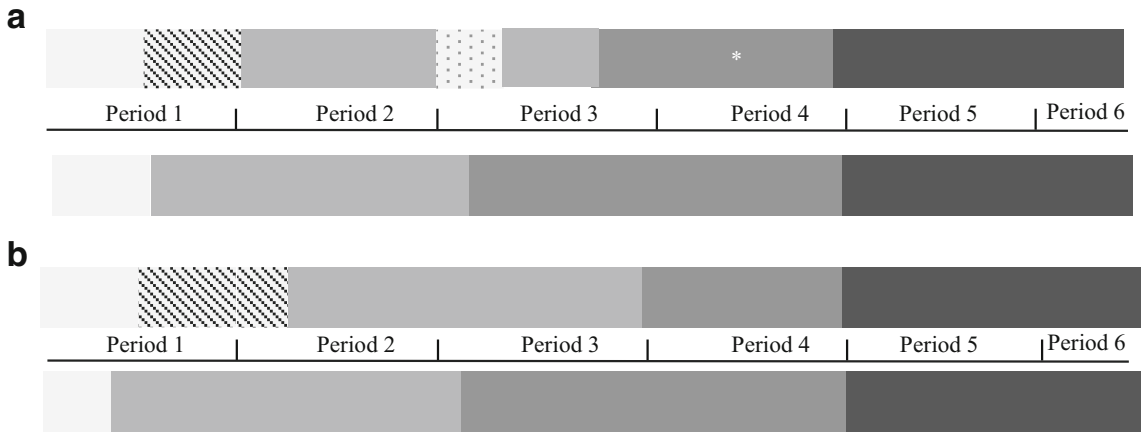


**Fig. 5** **a** EvoBuild (top) and model explorers (below) in the at-grade-level performing classes at Highland. Students had about an hour in each period. This depicts a total of 5.5 h (made to scale). **b** EvoBuild (top) and model explorers (below) in the low performing classes at Highland. Students had about an hour in each period. (Made to scale)

Figures 5 and 6 show the breakdown of activities by class period in the EvoBuild and model-exploration conditions at the two schools. Each gray solid colored cell depicts an activity. The striped cell marks time spent in the EvoBuild condition on learning to program. The spotted cell marks time spent due to technical issues with program files. An asterisk in a cell depicts that the teacher wrapped up the activity before most students finished working on it due to lack of time.

The study took place over 5 and a half periods (5.5 h) at Highland (Fig. 5a). As seen in the figure, overall, EvoExplore students spent more time working on the conceptual content as compared to EvoBuild students. This was because EvoBuild students spent some time experimenting with the software and building initial models on their own before starting with Activity 2 (striped cell). This time was not spent teaching EvoBuild students to program. It was spent showing them the location of XML library files were stored on the school computers so they could load them into the software. Once they loaded the XML file, students also experimented with available primitives to try out building models. In addition, EvoExplore students got nearly twice as much time on Activity 3. This was because of a technical disruption due to which EvoBuild students were unable to work on the activities (dotted cells). This disruption was not related to the programming task—software files were mistakenly deleted from the school computers because of an auto-maintenance weekly check run on them. The files had to be restored before students could proceed with the activity. As a result, EvoExplore students engaged in an additional exploration in Activity 3 that EvoBuild students did not participate in. Finally, to ensure that both conditions spent the same amount of time on activities, Activities 2 and 3 were ended early for EvoBuild students (cell with the asterisk). Similarly, in the low performing classes, EvoBuild students overall spent less time on the conceptual part of the activities as compared to EvoExplore students (Fig. 5).

As seen in Fig. 6, the study at Forest Park lasted for a total of six class periods. Students had approximately 45 min in each period to work on the activity. EvoBuild students were behind EvoExplore students through Activity 2 and caught up with them by Activity 3.

At both schools, students in the two modalities spent an equivalent amount of time on the activities. In some ways, EvoBuild students were disadvantaged and rushed along particularly in the beginning so the study could be completed within the same amount of time. This raised the question of whether programming their own model would still confer some of the expected advantages associated with learning about mechanisms on the builders. We turn to this issue next.

## Building Their Own Models Better Supported Learning About Mechanisms

Student responses to pre- and posttests from the two conditions were analyzed to investigate how students used

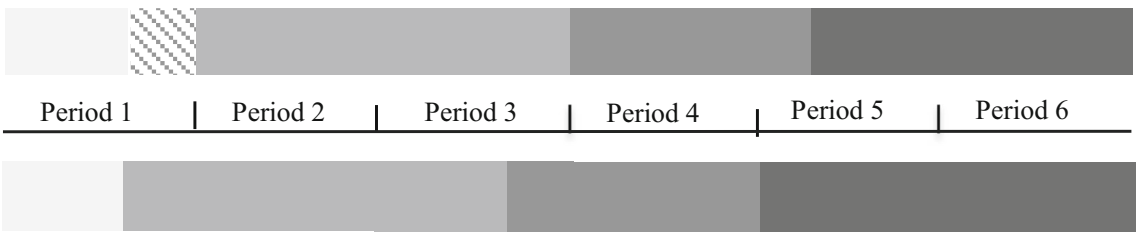


**Fig. 6** EvoBuild students and model explorers at Forest Park. Students had about 45 min in each period. (Drawn to scale)

ideas related to survival, death, reproduction, and inheritance to explain shifts in populations. The pre- and posttests consisted of a set of micro-evolutionary scenarios with open-ended questions asking students to predict and explain the occurrence of described shifts in population distributions.

Student responses were coded to investigate whether and how students used causal evolutionary mechanisms to account for population change. Three categories of codes were developed: evolutionary mechanisms, non-evolutionary mechanisms, and no mechanisms. Each of these categories had sub-codes to characterize the specific kind of explanation provided by students (see Table 3). Responses coded as evolutionary mechanisms used at least some individual-level behaviors of survival/ death, reproduction/ inheritance to provide a causal account of a shift in trait distributions. Responses coded as non-evolutionary responses used at least some of these individual-level behaviors without casually connecting them to explain a shift in trait distributions. Finally, responses that did not include any individual-level behaviors were coded as no mechanism.

Table 4 presents results from pre and posttest analysis. Several students in both modalities did not complete the pretest, and some students were absent. Because the unit of analysis was student responses to questions, incomplete tests reduced the number of responses available for analysis. This explains the difference in the difference in the total N between pre and post. Because students who did not complete the test were present through and participated in the intervention, we did not want to exclude their data. Hence, we compared the two conditions to one another at pre and post to examine whether students' responses were statistically comparable before and after the intervention. A chi-squared test was used to statistically compare the two conditions. Because a chi-squared test requires independent observations, combining student responses across questions in the test to conduct a single statistical comparison was not possible. Instead, we performed the test separately on three questions in the assessment. In the pretest, differences in the frequency of student responses identified into one of the three categories were not statistically significant between the two conditions. In contrast, in the posttest, a greater percentage of EvoBuild students provided evolutionary explanations as compared to EvoExplore students. This difference was statistically significant for two of the three questions analyzed. This finding demonstrated that programming their own model facilitated greater learning of mechanisms.

For example, one of the scenarios in the assessment described a scientist placing a group of guppies/bugs in a habitat with no predators (Grassland B). This group was described as consisting of three variations of body colors in equal proportions. Students were told that these guppies lived for 3–4 years. The scenario stated that when a scientist visited this habitat after several years, s/he found that the brightly colored individuals were the highest proportion in the population. Students were asked to:

1. Explain what led to the results.
2. Asked what the results might be if an identical experiment was conducted again.
3. Asked to explain the predicted results of a re-run of this experiment.

This question had been designed to elicit explanations related to sexual selection or drift. That is, it was expected that students would account for the population shift as occurring due to selection pressures acting on brightly colored guppies that were more visible to potential mates or as being driven by chance. Both these explanations were more common among the EvoBuild students than EvoExplore students. Given below is an example[3] of an EvoBuild student's response to the three questions in this scenario:

1. The red male bugs were mostly the only bugs after 70 years because females can easily find red male bugs and they could of had babies.
2. Yes, most of the bugs will be red because the females can spot them easily and maybe have babies.
3. Yes, 88% of the bugs will be red.

(EvoBuild Group 2 Post No. 19)

The student explained that there were more red bugs because females could spot them easily and have babies. In addition, the student predicted that in the re-run of the experiment, an identical percentage of guppies would be red. Though the latter part of the student's response (no. 3) was deterministic, the response provided an account for why a specific color thrived instead of the others: because guppies of that color were spotted easily by females and could have babies.

In contrast, a higher percentage of EvoExplore students provided responses that were non-causal. In other words, these responses mentioned some individuals reproducing, dying, or surviving without providing a causal explanatory account for the specific outcome. Given below is an example:

1. Predators are attracted to red male bugs and their [there] were no predators put into Grassland B.
2. Yes, red is easy to be noticed by predators and since they are no predators they will be more and they are going to keep reproducing.

[3] Student responses have been transcribed as is from their writing, retaining grammatical or spelling errors.

**Table 3** Sub-codes of the specific kind of explanation provided by students

| | |
|---|---|
| Evolutionary explanations | |
| Selection pressure (SP) | I think the scientist would explain why Stream A had mostly gray guppies after 50 years because the gray guppies can hide or camouflage in the background while the others can't |
| Described how a trait served as an advantage or disadvantage for the individual in an environment without elaborating further using rules such as survival or reproduction | |
| Natural selection (NS) | I made this prediction because it's the facts the poisonous toad was released and the ones who can eat them are more than likely to be dead. The ones that can't eat them are alive and repopulated. (EB Group 2 Post No. 27) |
| Described how the specific trait served as an advantage or disadvantage to individuals as well as included the idea of passing down of certain traits to successive generations | |
| Sexual selection (SS) | The scientist would explain this by saying that there were more red bugs because they were easy to spot for female bugs. |
| When the response described certain individuals mating because of specific traits that are attractive to females. A response was coded as SS only if a student also mentioned that this attraction made it possible or more likely to reproduce | Yes, I think that most bugs will be red after 70 years because female bugs can spot them and perduce [produce] more bugs. |
| | The bug that would be the most common will be red because female bugs can spot them and their will be more red bugs |
| | (EE Group 2 Post No. 13) |
| Drift-like (DRIFT) | I think that there wouldn't still be that many variations because *may by accident or coincidence the rats could have eaten all of the green. Then that one color would be illimenated* [eliminated]. Maybe the white ones were more attractive to females and they mated making more white anoles making it harder to make that color of anoles extincts since there would be more of them than the other colors so the other colors of anoles could have died out because their numbers would be littler. (EB Group 1 Post No. 3) |
| When it described the likelihood loss of some variations from the population because of the unpredictability of which individuals with which variations survived or reproduced. | |
| Some responses that did not describe loss of variations were also coded as DRIFT if they described proportions of variations in a population changing due to chance alone | There could be a possibility that they would all be there but I think there would be more yellow colored anoles because there was more of the anoles in the beginning. The brown and white might become extinct because there was less. |
| | I made this prediction because if there was more anoles in a certain color then there would be more of them later on. there would also be more chance of them reproducing more. The other anoles with less of it's kind won't have as many babies because there's less of them. (EB Group 1 Post No. 7) |
| Equal chance (EQ) | Yes, because it doesn't really matter what color the spots are because they pose no threat or help to survive. |
| When it described the distribution of a trait as remaining more or less equal because each individual had an equal chance of surviving or reproducing | I made this prediction because the spots really don't give away a survival way. I also made this prediction because the lizards will have babies and they might turn out with the same color spots. (EB Group 1 Post No. 15) |
| Non-evolutionary explanations | |
| Reproduce/Survive (REP/SURV) | I think the scientist in Grassland A had mostly green male bugs after 70 years because female reproduce and have babys [babies]. (EB Group 2 Post No. 8) |
| When a response mentioned individuals reproducing or surviving/dying without a causal account for why a specific variation/s thrived (e.g., due to chance or selection pressures that also account, to some extent, for why other variations did not thrive) | Yes, red is easy to be noticed by predators and since they are no predators they will be more and they are going to keep reproducing. (EE Group 2 Post No. 22) |
| Other Explanations (OTHER) | I made this prediction because I think after 100 years that the rats would have aten [eaten] most of the anoles so that why I thought their would be a little anoles. (EE Group 1 Post No. 24) |
| Change in population size | |
| Some student responses described the overall population as changing and were also coded as OTHER | |
| No mechanism | |
| Presence or absence of predators (PRED) | There were no predators to eat the orange ones so they lived. |
| When a response stated the presence or absence of predators as cause for a change without providing an additional reason for why that particular variation thrived while others did not, it was coded as PRED | We don't know for sure but I can guess that their will be a lot of orange guppies because there are no predators to eat them. |
| | Yes because their habitat only contains them and no other animal. (EE Group 1 Post No. 25) |
| No mechanism (NO MECH) | Yes because since the spots are so "mysterious" they probably wouldn't change. |
| When it provided no individual-level mechanisms to describe how populations might change over time or not | I think this because since the spots don't interfere with the brown anoles they probably won't change. (EB Group 1 Post No. 30) |

3. Probably not, it will come close maybe but not exact because they will not reproduce the exact number of times like in the previous experiment.

(EvoExplore Group 2 Post No. 21)

The student explained that because there were no predators, the red male bugs would keep reproducing. In addition, s/he predicted that in a re-run of this experiment, there would again be mostly red male bugs because in the absence of predators, they would keep reproducing. However, the exact number of times they would reproduce might be different from the first run of the experiment. Note that the explanation does not account for why none of the other variations thrived instead of the red ones. It simply mentioned that the red variation survived and kept on reproducing because there were no predators.

Finally, more EvoExplore students provided responses that did not include any mechanism. For instance, some students attributed the absence of predators as the sole reason for a certain variation thriving without elaborating on additional individual-level rules. Given below is an example:

1. Those red ones are maybe the ones that are just normal ones and since Grassland A and B are diffrent they have diffrent [different] amount of colored ones.
2. Yes they would because it seems that the ones without predators are the ones that are red and the ones with predators have green.
3. It would be the same color because it's the same thing that happened with grassland C[4] it's just this time it's a diffrent color since this Grassland has no predators and other grassland has them

(EvoExplore Group 2 Post No. 14)

Teacher interviews provided some insight into how students' work might have led to these results. The science teacher at Forest Park pointed out that because students were programming their own model, they had control over a wider spectrum of the model. In particular, students were able to use the code to explain population trends observed from running the model:

> *T2*: They, I think the first group [EvoBuild] really got the sense of they're the ones in control. And they're the ones manipulating things and um, the second group [EvoExplore] only got that in the sense of they could control the mutation slide, 'cause I could ask them who's controlling mutation in that model. So the first group, I felt, understood a lot more what was actually happening. And I think that that's a lot deeper learning when you're not guessing about that man behind the curtain and what's going on. Because they can see and if they ask me about, "Well is this happening in the model? Is this happening here?" Then I can say, "Did you program it to happen in the model?" Um, 'cause like, in the second period [EvoExplore], A asked me today, she's like, "Well maybe the, maybe the bugs are getting smarter and they–" And she's not able to see that no–*R1*: Mm. There is no intelligence–*T2*: Nothing's changing. There's no—'Cause that's a perfectly valid assumption for her to think. Maybe just, it's like, you don't know what's going on.

The teacher described that the EvoBuild students felt like they were in control and could manipulate a wider bandwidth of the model, which helped them understand what was actually going on. He added that this control facilitated deeper learning when contrasted with the experience of the model explorers who often tried to guess[5] the underlying rules ("about that man behind the curtain and what's going on"). Moreover, students' familiarity with and access to the code allowed him to leverage it as a resource when helping students. In contrast, inferences EvoExplore students made about the workings of the model were harder for him to challenge because they could not access the code.

On a related note, the science teacher at Highland emphasized that assembling the code themselves gave EvoBuild students greater familiarity with how the model worked:

> *T1*: But the build kids knew what they were doing because they told them [agents in the model] what they were doing. Um, whereas some of the kids had to just kind of explore the model, um, you know, the other kids put it, the build kids put it together. So there wasn't all that time having to figure out– I mean, there was the initial showing them how to build it. But it wasn't, you know, having to like, look at someone else's work. Essentially.

Moreover, she pointed out where students in each modality spent extra time in relation to the other modality: While EvoBuild students spent additional time in the beginning on learning how to build a model, EvoExplore students spent extra time trying to figure out how the model worked because they were looking at someone else's work.

[4] This was the name of the grassland for the re-run of Experiment 1, which was a camouflage experiment.

[5] Model rules were presented to EvoExplore students through a teacher demonstration at the start of the activity. They were also accessible on student worksheets.

**Table 4** Frequency of responses (percentage in brackets) to three questions in the pre- and posttests

| | Pretest | | Posttest | |
|---|---|---|---|---|
| | Builders | Explorers | Builders | Explorers |
| | Question 1 | | | |
| Evolutionary mechanism | 12 (22.22) | 7 (13.21) | 31 (60.78) | 19 (33.93) |
| Non-causal | 11 (20.37) | 10 (18.87) | 7 (13.73) | 13 (23.21) |
| No mechanism | 31 (57.41) | 36 (67.92) | 13 (25.49) | 24 (42.86) |
| Total Q1 | 54 | 53 | 51 | 56 |
| | Chi-square = 1.727 $P > 0.05$ | | Chi-square = 7.734 $P < 0.05$ | |
| | Question 2 | | | |
| Evolutionary mechanism | 6 (16.67) | 3 (6.52) | 30 (46.88) | 16 (22.86) |
| Non-causal | 7 (19.44) | 7 (15.22) | 8 (12.5) | 22 (31.43) |
| No mechanism | 23 (63.89) | 36 (78.26) | 26 (40.63) | 32 (45.71) |
| Total Q2 | 36 | 46 | 64 | 70 |
| | Chi-square = 2.685 $P > 0.05$ | | Chi-square = 11.169 $P < 0.05$ | |
| | Question 3 | | | |
| Evolutionary mechanism | 3 (7.5) | 2 (4.17) | 29 (46.03) | 26 (40) |
| Non-causal | 10 (25) | 10 (20.83) | 2 (3.17) | 3 (4.62) |
| No mechanism | 27 (67.5) | 36 (75) | 32 (50.79) | 36 (55.38) |
| Total | 40 | 48 | 63 | 65 |
| | Chi-square = 0.765 $P > 0.05$ Yates' chi-square = 0.158 Yates-$P > 0.05$ | | Chi-square = 0.568 $P > 0.05$ Yates' chi-square[a] = 0.205 Yates-$P > 0.05$ | |

[a] A Yates chi- squared test was also performed because the frequency of some cells was lower than 5.

To summarize, findings revealed that EvoBuild students manifested greater learning about evolutionary mechanisms as compared to EvoExplore students who investigated a pre-built model. This trend was also reflected in students' in-class activity as noted by the teachers in their interviews.

## Discussion

Since the time of Logo in the 1970s, there has been extensive research documenting that programming is a powerful way to learn about mechanisms underlying phenomena (e.g., Papert 1980; Sherin 2001; Simpson et al. 2005; Wilensky and Reisman 2006; Wilkerson et al. 2014). In the introduction, we briefly outlined a classification of programming environments based on the kinds of primitives available such as text-based, graphical, domain-specific, or more general-purpose primitives. When students engage in programming using general-purpose text-based programming environments, benefits are cumulative over a long period of time and are reflected in content learning as well as learning about programming (e.g., Harel & Papert, 1991; Wilensky 2003). Costs associated with programming in such environments such as requiring some time investment to learn to program might not be felt over a long time span. However, teachers might want to adopt programming activities for one-off uses for single activities or units. When working with first-time programmers in such scenarios, the costs of using general-purpose text-

based programming environments might be more salient than its benefits of a high ceiling. This, in turn, would deter the adoption of programming activities for one-off uses by teachers who are forced to deal with the constraints of an over-loaded curriculum and inadequate time. This deterrence would be particularly salient in classrooms where students and teachers are novice or even first-time programmers. This leads to a design tension between leveraging the benefits of students engaging in programming while dealing with constraints of a classroom.

Our argument in this paper lies at the heart of this tension, and seeks to address it. We claimed that quickstart programming toolkits such as EvoBuild can provide benefits of programming for learning without costs typically associated with programming activities such as requiring extended time or prior programming experience. These toolkits combine programming using graphical primitives with domain-specific primitives to make programming easily accessible to even first-time programmers. To make this claim, we presented a design-based research study that compared programming in a quickstart environment with model-exploration activities for learning about evolutionary mechanisms. The combined sample from two schools included students from a range of academic backgrounds, many of who had never programmed before. Moreover, the study took as a constraint the time spent on the programming and model-exploration activities, and both modalities spent the same number of class periods on the activities. Under such conditions and over such a short period of time, one might expect that the value of programming on learning about mechanisms might not be visible. If students were programming for the first time to learn about challenging content, this would reflect in their performance on posttests that were designed to assess content learning about evolutionary mechanisms. However, our findings indicate the opposite trend: The programming students performed quite well by manifesting greater learning about evolutionary mechanisms as compared to model explorers.

This work highlights the importance of investing in research on programming to learn science in two ways. First, it calls for continued attention on design research on developing programming environments for learning about scientific phenomena for K-12 education. In particular, it brings attention to quickstart environments such as EvoBuild as well as environments that support their development such as DeltaTick. By providing an infrastructure to parse and present NetLogo procedures as predefined blocks, DeltaTick provided an environment to engage in graphical blocks-based programming in NetLogo. This infrastructure allowed for designing a quickstart toolkit that integrated graphical primitives with orientation to a domain to make programming to represent domain structures immediately accessible even to first-time programmers. It is important to point out that the accessibility of primitives in EvoBuild also comes from close attention to the alignment between the specific content and the domain-oriented nature of primitives. Specifically, students have rich prior knowledge about individual-level behaviors relevant to evolutionary change such as survival, death and reproduction (Metz 2010; Wagh, 2016). Because of their domain-specific and graphical nature, the available primitives in the library reflected these individual-level behaviors in the form of blocks. Hence, even students who were first-time programmers had initial ideas about the kinds of rules to add to their model to represent individual-level interactions between organisms in an ecosystem. In addition, the existing infrastructure of DeltaTick (Wilkerson and Wilensky 2010) was extended for an alignment with the specific content, evolutionary processes. In particular, to design EvoBuild, DeltaTick was extended by adding the ability to define properties for breeds to allow learners to view and modify distributions of properties of a population. We need both kinds of learning environments, quickstart toolkits and environments that provide infrastructure for such toolkits, to enable teachers and students to leverage the benefits of programming for one-off short-term uses in science classrooms.

Quickstart toolkits provide environments to *initiate* the adoption of programming activities in science classrooms. However, because of their relatively modest ceiling, they offer limited potential for continued expansion. For this reason, another implication of this work lies in examining pathways that encourage and support shifts from using quickstart environments with a fairly modest ceiling to environments that offer a higher ceiling. Such shifts will facilitate meaningful long-term integration of programming activities in science classrooms.

**Acknowledgements** This research is made possible by support from the National Science Foundation under NSF grant DRL-1109834. However, any opinions, findings, conclusions, and/or recommendations are those of the investigators and do not necessarily reflect the views of the Foundation. The authors thank Jessica Watkins, Sharona Levy, and David Hammer for feedback on previous versions of this manuscript.

## References

Bamberger, J. (2001). Turning Music Theory on its Ear: Do we hear what we see; do we see what we say? In *Multidisciplinary Perspectives on Musicality: The Seashore Symposium*. Iowa City: University of Iowa Press.

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. *Int J Comput Math Learn, 14*(1), 81–119.

Bruckman, A. (1997). *Moose Crossing: Construction, Community, and Learning in a Networked Virtual World for Kids*. Cambridge: Massachusetts Institute of Technology.

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

Collins, A., Joseph, D., & Bielaczyc, K. (2004). Design Research: Theoretical and Methodological Issues. *J Learn Sci, 13*(1), 15–42. https://doi.org/10.1207/s15327809jls1301.

Dickes, A. C., & Sengupta, P. (2013). Learning Natural Selection in 4th Grade with Multi-Agent-Based Computational Models. *Res Sci Educ, 43*(3), 921–953. https://doi.org/10.1007/s11165-012-9293-2.

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). Heidelberg: Springer Berlin Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-57799-4_8.

Harel, I., & Papert, S. (1991). Constructionism : research reports and essays, 1985-1990. Norwood: Ablex Pub. Corp.

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

Horn, M. S., Brady, C., Hjorth, A., Wagh, A., & Wilensky, U. (2014). Frog Pond: A Codefirst Learning Environment on Evolution and Natural Selection. In Proceedings of the 2014 Conference on Interaction Design and Children (pp. 357–360). New York: ACM. https://doi.org/10.1145/2593968.2610491.

Ioannidou, A., Repenning, A., Lewis, C., Cherry, G., & Rader, C. (2003). Making Constructionism Work in the Classroom. *Int J Comput Math Learn, 8*, 63–108.

Kafai, Y. B., Carter Ching, C., & Marshall, S. (1997). Children as designers of educational multimedia software. *Comput Educ, 29*(2–3), 117–126. https://doi.org/10.1016/S0360-1315(97)00036-5.

Kahn, K. (2007a). Building computer models from small pieces. In G. Wainer (Ed.), SCSC Proceedings of the 2007 Summer Computer Simulation Conference (pp. 931–936). San Diego.

Kahn, K. (2007b). The BehaviourComposer 2.0: a web-based tool for composing NetLogo code fragments. Retrieved July 5, 2013, from http://academia.edu/329330/The_BehaviourComposer_2.0_a_web-based_tool_for_composing_NetLogo_code_fragments

Kahn, K., & Noble, H. (2010). The BehaviourComposer 2.0: a web-based tool for composing NetLogo code fragments. In J. Clayson & I. Kalas (Eds.), Constructionist approaches to create learning, thinking and education: Lessons for the 21st century: Proceedings for Constructionism 2010. Paris.

Kahn, K., Noble, H., & Hjorth, A. (2012). Three-minute Constructionist Experiences. In C. Kynigos, J. Clayson, & Y. Nikoleta (Eds.), Proceedings of Constructionism 2012, Theory Practice and Impact (pp. 349–358). Athens.

Klopfer, E., Yoon, S., & Um, T. (2005). Teaching Complex Dynamic Systems to Young Students with StarLogo. *J Comput Math Sci Teach, 24*(2), 157–178.

Konold, C., & Miller, C. D. (2005). *TinkerPlots: Dynamic data exploration. Computer Software*. Emeryville: Key Curriculum Press Retrieved from http://scholar.google.com/scholar?cluster=5929212600541009408&hl=en&oi=scholarr.

Louca, L. T., & Zacharia, Z. C. (2007). The Use of Computer-based Programming Environments as Computer Modelling Tools in Early Science Education: The cases of textual and graphical program languages. *Int J Sci Educ, 30*(3), 287–323. https://doi.org/10.1080/09500690601188620.

Metz, K. E. (2010). Scaffolding children's understanding of the fit between organisms and their environment in the context of the practices of science. In Proceedings of the 9th International Conference of the Learning Sciences - Volume 1 (pp. 396–403). International Society of the Learning Sciences. Retrieved from http://dl.acm.org/citation.cfm?id=1854360.1854411.

NGSS Lead States (2013). Next Generation Science Standards: For States, By States. Washington, DC: The National Academies Press.

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

Rader, C., Cherry, G., Brand, A., Repenning, A., & Lewis, C. (1998). Principles to Scaffold Mixed Textual and Iconic End-User Programming Languages. In Proceedings of the 1998 I.E. Symposium of Visual Languages (pp. 187–194). Nova Scotia.

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., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. & Kafai, Y. (2009). Scratch: Programming for All. In *Communications of the ACM* (Vol. 52, pp. 60–67).

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.

Sherin, B. (2001). A Comparison of Programming Languages and Algebraic Notation as Expressive Languages for Physics. *Int J Comput Math Learn, 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. *J Comput Assist Learn, 21*(2), 143–158. https://doi.org/10.1111/j.1365-2729.2005.00121.x.

Smith, D. C., Cypher, A., & Schmucker, K. (1996). Making Programming Easier for Children. *Interactions, 3*(5), 58–67. https://doi.org/10.1145/234757.234764.

Turkle, S., & Papert, S. (1992). Epistemological Pluralism and the Revaluation of the Concrete. *Journal of Mathematical Behavior, 11*(1), 3–33.

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

Wagh, A., Cook-Whitt, K., & Wilensky, U. (2017). Bridging inquiry-based science and constructionism: Exploring the alignment between students tinkering with code of computational models and goals of inquiry. *Journal of Research in Science Teaching*. https://doi.org/10.1002/tea.21379

Wagh, A., & Wilensky, U. (2012a). Breeding birds to learn about artificial selection: Two birds with one stone? In: J. van Aalst, K. Thompson, M. Jacobson, & P. Reimann (Eds.), *10th International Conference of the Learning Sciences: The Future of Learning* (Vol. 2: Short papers, pp. 426–430). Sydney, Australia, July 2-6.

Wagh, A., & Wilensky, U. (2012b). Mechanistic Explanations of Evolutionary Change Facilitated by Agent-based Models. Paper presented at the American Educational Research Association, Vancouver, April 13-17.

Wagh, A., & Wilensky, U. (2013). Leveling the Playing Field: Making Multi-level Evolutionary Processes Accessible through Participatory Simulations. In N. Rummel, M. Kapur, M. Nathan, & S. Puntambekar (Eds.), *To See the World and a Grain of Sand: Learning across Levels of Space, Time and Scale* (Vol. 2, pp. 181–184). Madison, Wisconsin, June 15-19: Proceedings of CSCL.

Wagh, A., & Wilensky, U. (2014). Seeing patterns of change: Supporting student noticing in building models of natural selection. In G. Futschek & C. Kynigos (Eds.), *Constructionism and Creativity,*

*Proceedings of the 3rd International Constructionism Conference*. Vienna: OCG (Österreichische Computer Gesellschaft).

Wagh, A., Novak, M., Soylu, F., & Wilensky, U. (2016). Integrating agent-based modeling & case Study to learn about population dynamics: A design framework. Paper presented at NARST, Baltimore, April 14-17.

Weintrop, D. (2015). Minding the Gap Between Blocks-Based and Text-Based Programming (Abstract Only). In Proceedings of the 46th ACM Technical Symposium on Computer Science Education (pp. 720–720). New York: ACM. https://doi.org/10.1145/2676723.2693622.

Wilensky, U. (1999a). GasLab—An extensible modeling toolkit for connecting micro- and macro- properties of gases. In N. Roberts, W. Feurzeig, & B. Hunter (Eds.), *Computer Modeling in Science and Mathematics Education*. Berlin: Springer-Verlag.

Wilensky, U. (1999b). *NetLogo. http://ccl.northwestern.edu/netlogo/.* Evanston: 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., & Centola, D. (2007). Simulated Evolution: Facilitating Students' Understanding of the Multiple Levels of Fitness through Multi-Agent Modeling. In Proceedings of the Fourth International Conference on Complex Systems. Nashua.

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

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. *Cogn Instr, 24*(2), 171–209.

Wilkerson, 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/Aug 10-14 .

Wilkerson, M. H., Gravel, B. E., & Macrander, C. A. (2014). Exploring Shifts in Middle School Learners' Modeling Activity While Generating Drawings, Animations, and Computational Simulations of Molecular Diffusion. *J Sci Educ Technol, 24*(2–3), 396–415. https://doi.org/10.1007/s10956-014-9497-5.

Wilkerson, M., Wagh, A., & Wilensky, U. (2015). Balancing Curricular and Pedagogical Needs in Computational Construction Kits: Lessons From the DeltaTick Project. *Sci Educ, 99*(3), 465–499. https://doi.org/10.1002/sce.21157.

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

Yoon, S., Anderson, E., Klopfer, E., Koehler-Yom, J., Sheldon, J., Schoenfeld, I., Wendel, D., Scheintaub, H., Oztok, M., Evans, C., & Goh, S.-E. (2016). Designing Computer-supported Complex Systems Curricula for the Next Generation Science Standards in High School Science Classrooms. *Systems, 4*(4).