J Sci Educ Technol (2015) 24:628–647
DOI 10.1007/s10956-015-9552-x

Comparing Virtual and Physical Robotics Environments for Supporting Complex Systems and Computational Thinking

Matthew Berland · Uri Wilensky

Published online: 7 March 2015

Abstract Both complex systems methods (such as agent-based modeling) and computational methods (such as programming) provide powerful ways for students to understand new phenomena. To understand how to effectively teach complex systems and computational content to younger students, we conducted a study in four urban middle school classrooms comparing 2-week-long curricular units—one using a physical robotics participatory simulation and one using a virtual robotics participatory simulation. We compare the two units for their effectiveness in supporting students' complex systems thinking and computational thinking skills. We find that while both units improved student outcomes to roughly the same extent, they engendered different *perspectives* on the content. That is, students using the physical system were more likely to interpret situations from a bottom-up ("agent") perspective, and students using the virtual system were more likely to employ a top-down ("aggregate") perspective. Our outcomes suggest that the medium of students' interactions with systems leads to differences in their learning from and about those systems. We explore the reasons for and effects of these differences, challenges in teaching this content, and student learning gains. The paper contributes operationalizable definitions of *complex systems perspectives* and *computational perspectives* and provides both a theoretical framework for and empirical evidence of a relationship between those two perspectives.

Keywords Computational thinking · Systems thinking · Robotics · Participatory simulations

M. Berland (✉)
Department of Curriculum and Instruction, University of Wisconsin–Madison, 225N. Mills Street, Madison, WI 53706, USA
e-mail: mberland@wisc.edu

U. Wilensky
Departments of Learning Sciences and Computer Science, Center for Connected Learning and Computer-Based Modeling, Northwestern Institute on Complex Systems, Northwestern University, Chicago, IL, USA
e-mail: uri@northwestern.edu

Introduction

In the last 10 years, there has been a significant amount of literature reviewing both computational thinking and complex systems thinking, but there has been relatively little work connecting the two. Computational thinking is often situated in both computer programming and understanding computer systems and often defined by the modes of thought engendered by computer programming (National Research Council 2010). Complex systems thinking is often defined in relationship to computational systems modeling and the emergence of behaviors from the interaction of individual elements (Jacobson and Wilensky 2006). In this paper, we present a comparative study of students using a visual programming language to program either virtual or physical robots in an attempt to understand the ways that complex systems thinking and computational thinking interact and how we can best support students in learning them. This paper presents the argument that the two types of thinking work together by leveraging student *perspectives* on complex systems and computational content. This argument contributes to the field both by operationalizing a definition of those perspectives and by providing clear multimodal evidence of their overlap.

Framing a phenomenon as a complex system provides an accessible way for students (and scientists) to understand complex scientific content. Complex systems theory and methods have been used extensively to investigate scientific phenomena, and there is evidence that by breaking down scientific phenomena into constituent parts and the relationships between them, students may better develop inroads to understanding (viz. Holland 1999; Wilensky and Reisman 2006). However, there has been some debate as to the usefulness of teaching secondary school science with complex systems. Hmelo-Silver and Pfeffer (2004) argue that complex representations are unintuitive for younger students, and Chi (2005) argues that structuring phenomena as complex systems can encourage students to stop at surface level features rather than delve into more systemic understandings. On the other hand, Wilensky and Reisman (2006) show that teaching students to model complex systems can lead to deep systems understanding more easily and more quickly than traditional science instruction, and recent work suggests that teaching complex systems can help students transfer knowledge between relatively disparate content areas (Davis and Sumara 2006; Goldstone and Wilensky 2008).

One potential issue is that modeling scientific phenomena with complex systems methods often involves some form of computer programming (Johnson 2002). There is considerable disagreement about the importance and the feasibility of teaching young students to program computers. Some have argued that programming is too difficult for younger students (Lahtinen et al. 2005; Pea and Kurland 1984). In contrast, there is a body of research that claims to show that K-12 students can program computers to perform a variety of useful tasks without sustained instruction (Berland et al. 2011; Kelleher et al. 2007; Papert 1980; Wyeth 2008). Indeed, both learning scientists such as diSessa (2001) and computer scientists such as Ben-Ari (2001) or Guzdial and Forte (2005) find that many of the reported difficulties in teaching computer programming have been due to the structure and representations of the tasks in traditional computer science instruction. The implication is that if we design and facilitate more relevant, engaging, and powerful activities, these difficulties will matter less.

In this paper, we describe a learning environment, VBOT (Berland and Wilensky 2008), that we designed and developed to teach both complex systems thinking using a form of computer programming and computational thinking. Using VBOT, we investigate how framing complex systems modeling as a collaborative computer programming task affects how students interpret systems content and, conversely, how framing computational thinking content in a systems context affects how students interpret computational content. As framing content is fundamentally a design question, this study is structured as design-based research (Cobb et al. 2003; Collins et al. 2004). For the study, we designed, developed, and deployed two versions of VBOT, one that uses actual physical robots (Physical VBOT) and one that uses simulated robots on a screen (Virtual VBOT).

The central design-based research question investigated here is how these two instantiations of VBOT, physical and virtual, differently affected student perspectives and understanding of computational and complex systems thinking skills.

To address this question, we conducted a study in four 8th grade classrooms across two public schools in Chicago. The study compared 2-week-long robotics curricular units using VBOT—one that used a physical robotics participatory simulation and one that used a virtual robotics participatory simulation (Colella 2000; Klopfer et al. 2004; Wilensky and Stroup 1999a). Units were matched for equivalence in user interface, curriculum, teacher effects, and school effects. These two units were compared for their effectiveness in generating student understandings of complex systems content (in this case, being able to use systems modeling to address science content) and computational content (in this case, being able to use computer programming to address science content). The remainder of the paper describes the rationale for the study, the design of the units and tools, and the resulting student outcomes. We define and describe our theoretical framework for understanding computational and complex systems content and assess the effectiveness of the units in supporting students' understanding.

This study suggests that the way that students interact with and model complex systems creates meaningful differences in student perspectives on those systems. In particular, students who worked with the virtual systems gained a more top-down ("aggregate") perspective on both complex systems and computational content, and the students who worked with the physical systems gained a more bottom-up ("agent") perspective on that content. While average performance gains were similar across all classrooms, we found significant differences in both the process of learning the content and student understanding of the content between virtual class and physical class students. These differences persisted across a variety of measures, in two different schools, and with three different teachers. This work explores the reasons for these differences of perspective, the effects of the differences of perspective, the student learning gains, and the challenges in teaching complex systems and computational thinking.

Computation and Complex Systems

Both computational thinking and complex systems thinking are not yet well defined, but core to our interpretation of both is that learning the *methods and perspectives* of a domain gives a student the ability to perceive and understand new content from other domains. Computational thinking is a term used to describe how one can use the methods and perspectives of computer science (diSessa 2001), and complex systems thinking used to describe how one could use the methods and perspectives of complex systems research (Jacobson and Wilensky 2006).

Computational thinking is a term often used to describe the ability to think with the computer-as-tool. diSessa (2001) argues that developing cognitive ability requires that students adapt their thinking processes to align with their tools. Functionally, this might describe being able to think like a computer programmer or a computer artist. Thinking in such a way allows the student to use computation to solve even those problems in which computers might not be used. Moreover, being able to conceptualize which elements of a given problem the computer can assist with is important to computational thinking (National Research Council 2010). There are a variety of reasons why we would want to support computational thinking skills, but a core reason is that it enables the student to use a computer as a protean "tool to think with" (Papert 1980). Furthermore, evidence shows myriad benefits in supporting broader introductions to computation, including better problem solving skills (Schoenfeld 1992), better communication of computational content (diSessa 2001), and increased likelihood to report interest in STEM careers (Martin et al. 2013). In this paper, we use a new term: *computational perspectives*. Computational perspectives suggest that computational thinking is not monolithic and that thinking with the computer-as-tool can be affected deeply by how it is contextualized and constrained. Some definitions of computational thinking stress technical skill in learning to program, but what sets computational thinking apart from computer science is an emphasis on translating that understanding of computation to domains that are not necessarily computer scientific, whether by communicating computational content to nontechnical colleagues or by applying processes learned in programming to contexts that are not necessarily programming (Basawapatna et al. 2011; National Research Council 2010; Wing 2006).

Similarly, *complex systems thinking* can be used to describe a student's ability to think in terms of systems of elements. As with computational thinking, this term is hotly debated and somewhat divergent (c.f. Chi 2005; Goldstone and Wilensky 2008; Grotzer and Basca 2003), but, in this paper, we focus on a specific form of complex systems thinking called *levels thinking* (Wilensky and Resnick 1999). Levels thinking describes the ability to think with and from complex systems theories and models in terms of component aspects (which we term "agents"), groups of those agents (which we term "aggregates"), and the models and meaning that emerge from their relationships. Levy and Wilensky (2008) demonstrate that learning to think about phenomena in "levels" can support deeper understandings of many scientific phenomena quickly and effectively. Furthermore, as students become more familiar with the relationships between levels of complex phenomena, they can begin to use emergence and complex systems thinking as a tool to think about everyday physical and social phenomena such as traffic flowing on a highway, students self-organizing in a gym class, or the clapping of an audience.

Complex systems thinking enables students to understand scientific phenomena that are otherwise quite difficult to comprehend. Here, we use the term *complex systems* to mean systems in which effects or constructs emerge from aggregations of individual agents. A common example is the "V" shape of a flock of geese flying overhead. In this case, the "V" emerges from the aggregation of the behaviors of individual birds (or "agents"). There has been considerable research showing that understanding complex systems can be difficult for learners (Chi 2005; Penner 2000; Resnick 2003; Wilensky and Resnick 1999). To address this difficulty, agent-based modeling environments have been developed that help people make sense of complex systems (e.g., Collier 2003; Klopfer et al. 2002; Luke et al. 2005; Wilensky 1999). In the past decade, we have seen a growing body of research showing that using agent-based modeling is more comprehensible to middle and high school students than traditional equation-based science (Ioannidou et al. 2003; Klopfer et al. 2005; Wilensky 2003; Wilensky and Reisman 2006). As in computational thinking, complex systems thinking is distinct from traditional science learning in its emphasis on translating broad technical skills to a variety of science content domain contexts, communicating complex content to nonexperts, and learning how to apply complex systems methods to content that is not necessarily framed as a complex system (Blikstein and Wilensky 2009; Sengupta and Wilensky 2009).

Computational Perspectives and Complex Systems Perspectives

In this paper, we offer two new terms: *computational perspectives* and *complex systems perspectives*. Focusing on perspectives as an aspect of (computational or complex systems) thinking allows us to ask what is flexible about what one sees in a phenomenon. A *computational perspective* suggests that computational thinking is not monolithic and that thinking with the computer-as-tool can be affected by how it is contextualized and constrained. People can take up different "perspectives" on context—many of which can be complementary—such as reader, author, analyst, and critic. Perspectives are positions that a learner can take, a "stance" rather than a state. In contrast, some definitions of computational thinking stress technical skill in learning to program, but taking a computational perspective suggests that the learner is seeing computation across domains that are not necessarily computer scientific. Analogously, a *complex systems perspective* uses the model of a complex system as a kind of lens: how one understands a complicated phenomenon has a lot to do with how one frames the problem.

The central argument of the paper is that features of the design of the learning environment affect the students' perspectives. Specifically, we argue that the physical environment engendered an agent perspective on both computational systems content and that the virtual environment engendered a more aggregate perspective.

Connections Between Complex Systems Thinking and Computational Perspectives

Though hinted at in previous literature, the connections among complex systems thinking and computational thinking have not been explicitly examined. Discussions related to the implications of computational thinking extend well beyond contemporary discussions (Bundy 2007; Wing 2006). Approaching current discussions of computational thinking, Papert (1975) identified that students' construction of more refined understandings of mathematical (and other) concepts could be supported in their articulation of representations and problems in computational (algorithmic) terms. More recently, Pea (1987) outlined significant benefits of utilizing thought progressions similar to those of computers while building upon unique human abilities to plan and adapt increasingly complex algorithms. Similarly, Soloway (1986) highlighted the cognitive affordances of internalizing the structures of computer programs.

Though Wilensky and Reisman (2006) and other complex systems researchers do not specifically name computational thinking (Hmelo et al. 2000; Klopfer et al. 2005; Perkins and Grotzer 2005), the algorithmic, procedural nature of the systems explorations they describe reflects computational thinking descriptions provided elsewhere (Wing 2006). Also, within complex systems literature, there is reliance on programming agent-based behavior (e.g., individual agents) to utilize simple sets of rules (algorithms) when interacting, which results in emergently complex systems (Resnick and Wilensky 1998; Wilensky and Reisman 2006). As such, computational thinking skills are fundamental to perspectives that require the modeling of complex systems.

Despite the fact that the use of complex systems thinking has been shown to help learners better understand the functioning of individual components within multiple scientific and technical disciplines (Jacobson and Wilensky 2006) and in varied *levels* (as per Wilensky and Resnick 1999), there has been little explicit exploration of how complex systems thinking might support students' computational thinking. Similarly, there has been little exploration of how (general) computational thinking might be supported through complex systems thinking (Resnick and Wilensky 1998), which closely resembles Wing's (2006) description of "deeper computational thinking" (pp. 3719–3720). We have identified the overlapping parallels among complex systems thinking and computational thinking, the identified benefits of complex systems thinking for various content area concepts (Goldstone and Wilensky 2008), and calls to better understand how to support students' computational thinking (Wing 2006). As such, the current study is structured to explicitly examine those relationships among complex systems thinking, computational thinking, and student learning progressions. To date, there has been no simultaneous, comparative exploration of the impact of such environments in supporting the disparate skills of complex systems thinking, computational thinking, and novice programming simultaneously.

Our argument, in part, is that by focusing on the relationship between computational perspectives (as opposed to computational thinking more broadly) and complex systems perspectives, the connection will be both clearer and easier to identify. In short, this work addresses a specific hole in the existing research. In Wing (2006) description of the importance of computational thinking, she also highlights the necessity of identifying pedagogy that best supports students' computational thinking. Previous research suggests that complex systems thinking may prove beneficial in doing so (Goldstone and Wilensky 2008; Wilensky and Reisman 2006; Wilensky and Resnick 1999). Reliant on the previously identified literature, we attempt to provide some insight into some very specific questions related to facilitating students' computational and complex systems thinking. Specifically, we attempt to document potential differences in supporting students' computational thinking and complex systems thinking gains with physical or virtual novice programming environments. This is motivated by the aforementioned literature. In order to evaluate, if and how such affordances work together to better support students' learning, we focus on comparing complex systems thinking, computational thinking, and novice programming in two strongly related though disparate groups—one which utilized a purely virtual interface and virtual robots and another utilizing more physical (non-virtual) robots.

Supporting Complex Systems and Computational Perspectives

This study is necessary for two core reasons: We do not yet understand many of the connections between complex systems and computational perspectives, and we do not yet understand how the design of learning environments can affect how students learn that content. There are few studies addressing those connections, and even fewer studies addressing why and how we might support those connections.

However, there has been significant research into how students separately learn both programming and complex systems content, and there are confluences between the two domains. Much of the research in teaching both computational and complex systems content has used a constructionist *framework for action* (diSessa and Cobb 2004), as constructionism has been shown to be particularly well suited to teaching both systems modeling and computer programming in a social space. Research has repeatedly shown that robotics learning environments can be used to effectively teach computational content; building on such findings, both physical and virtual robotics systems have been used to teach computation (Berland et al. 2013; Hancock 2003; Portsmore 2005; Resnick et al. 1988; Schweikardt and Gross 2006; Sklar et al. 2003a). One core impetus for undertaking this study is that previous work by Wilensky and Reisman (2006) has suggested that computational thinking and complex systems thinking are related. Specifically, here we focus on examining the components of computational and complex systems thinking that require iterative problem solving, recursive thinking, and abstraction and decomposition when designing large complex systems (Wing 2006; viz. Wilensky and Reisman 2006). However, there have been no comprehensive studies of the relative affordances of physical and virtual environments in constructionist learning of these skills.

Outside of education research, in the literature on human–computer interaction, there have been several studies of the relative affordances of virtual and physical environments (see Sharlin et al. 2004, for a review of research on tangible interfaces), but that research is more narrowly technical and it is not typically empirical.

Similarly, though there is considerable work using constructionist methods to teach complex systems content and methods (e.g., Klopfer et al. 2005; Wilensky 2003), the field's youth leaves many topics unaddressed. There has been relatively little research concerning the use of physical robotics in teaching complex systems thinking, though it is common for robotics research to use groups of robots as examples of complex systems (see Parker and Schultz 2005, for a variety of examples).

As such, this paper presents two design-based research questions: How do virtual and physical robotics differently support complex systems and computational thinking? How do they engender perspectives on complex content?

Method

Participants

In this study, we deployed the VBOT learning environment (described below) in two classes at two Chicago public schools (four classes total). We worked with two 8th grade classes from a small nonmagnet middle school on Chicago's northwest side (henceforth, *Old Grove*[1]) and two 8th grade classes from a large nonmagnet public high school on Chicago's south side (henceforth, *Bayville*). At each school, one class used VBOT with virtual robots and the other used VBOT with physical robots. Each of the four enactments lasted for 5 school days. At Old Grove, separate teachers conducted the two classes, and Bayville had one teacher for both classes. No students overlapped between any two classes. Each class consisted of 15–24 8th grade students, with 78 students consenting (as detailed below). The schools were selected because they represent broadly different populations of low-/mid-SES, low-/mid-performing schools, as described below. For maximum equivalence, we selected the two biology classrooms at each school for the intervention. Both schools offered only two 8th grade biology classes, so no further sub-selection was necessary. That said, this work is *analytic* and *procedural* rather than confirmatory.

Old Grove School

In the two classes at Old Grove School, 43 8th grade students participated in the study. The school is approximately 50 % White (non-Hispanic) with 40 % of middle school students receiving free or reduced lunch. Our participants included multiple students from every inhabited continent except Australia. Old Grove is small, the classes tend to be small, the students greet each other freely in the halls, and the principal often stops students to chat in the halls.

Old Grove Class One (Virtual)—Mr. Wilson: In this class, 20 of 22 students had consented by the commencement of the study. Data were not collected on the two unconsented students because their consent forms were only received after the beginning of the study. Mr. Wilson taught the class that used the virtual VBOT system. His class expressed excitement at the prospect of using robots and games in the classroom.

Old Grove Class Two (Physical)—Mr. Cleveland: In this class, 23 of 24 students consented. One student refused consent for personal reasons. The class also engaged in whole class discussions mediated by Mr. Cleveland that exhibited strict turn taking. Mr. Cleveland had worked with robotics in his 3rd grade classroom several years prior and

[1] The names of all schools, teachers, and students have been changed.

had enjoyed using them, but had not yet found them to be useful for teaching 8th grade biology.

Bayville School

In the two classes at Bayville School, 35 8th grade students participated in the study. Bayville is classically institutional, with enormous buildings, large distances between classes, and few informal interactions outside the classroom. The school contains grades 7–12 and is located in an urban setting on the south side of Chicago. The school is approximately 95 % African-American, with approximately 70 % of the students receiving free or reduced lunch.

Bayville Classes One and Two (Virtual and Physical)—Ms. Adams: In the virtual class, all 24 students consented, and in the physical class, 11 of 15 students consented. Three students that refused consent reported religious prohibitions on video recordings, and one student refused consent for personal reasons. Ms. Adams taught both the virtual VBOT class and the physical VBOT class at Bayville. She has an unmistakable rapport with her students, who spoke freely with her and readily conversed with her about the material. At the time of the study, she had only been teaching for 3 years. She had previously used LEGO Mindstorms robots in a summer class she taught at Bayville, although she reported that she had not felt comfortable using them due to the lack of a curriculum and a self-perceived lack of technical experience.

VBOT: Virtual/Physical Robotics

The VBOT programming environment was designed and implemented for this project. It is a networked participatory simulation, using the HubNet module (Wilensky and Stroup 1999b) of the NetLogo agent-based modeling environment (Wilensky 1999). In VBOT, users program a virtual or physical robot (“vbot”) on a shared space of a physical or virtual soccer pitch. This works similarly to a networked soccer video game, but, rather than each user controlling a virtual player directly, they must program the actions of that player. While there is only one user interface for VBOT (see Fig. 1), there are two types of robots that the interface might control: virtual or physical. A virtual robot is similar to a player in the soccer video game, but with a visual, computer-based representation of a robot. A physical robot is a LEGO Mindstorms robot, which, in this case, was pre-built for the students to work with the VBOT system. In the virtual environment, users follow all vbots on an associated NetLogo screen, while watching their vbot’s behavior and history in their own interface. In the physical environment, users see all of the vbots currently playing on the ground, and they modify their programs on their own computers and then upload those programs to their physical vbot on the ground.

Beyond creating and manipulating the programs, the user interface supports users in monitoring vbots’ progress through three features: It shows the position of the user’s individual vbot on a radar-like display; it shows the positions of the other users’ vbots (for the virtual class only); and it shows a tapering history trail for the associated vbot.

VBOT Programming Language[2]

VBOT is a graphical programming language inspired by Braitenberg’s robots (1984). Braitenberg reframes programming into a set of visual *circuits* that control anthropomorphic robots. This framing is intended to introduce programming using an immediately accessible metaphor and imbue the act of programming with a “human” element. The most canonical example of one of his robots is the LOVE robot. LOVE uses just two light sensors, two simple motors, and two wires that each connects one sensor to one motor (left sensor to right motor and vice versa); it behaves in a surprisingly complex way, wandering toward the light and occasionally orbiting it. We designed and developed VBOT to be a graphical programming language using the metaphor of Braitenberg’s circuits with a functional programming paradigm. As such, students program by creating functions (in both the mathematical and computational sense) that take a numerical input and generate a numerical output. It is relatively novel in that: each program is a mathematical function that maps sensor data to motor actions; all programs compile; all programs generate some behavior; and the behavior of a program (in any state) is always visible on the screen.

The sensor data consist primarily of the proximity and direction to either other robots or a light source (on a scale of 0–100 %). The motor actions consist of signals given to the two back wheels of the physical robot or the simulated back wheels of the virtual robot. To map a set of sensor signals to motor actions, a user can use up to 25 *building blocks*. These building blocks include mathematical operators, if–then statements (*logical operators*), and variables. The mathematical operators are add, subtract, multiply, divide, and negate; the mathematical operators take multiple inputs (from sensors or other blocks) and perform an operation on those inputs (see Fig. 1 for an example of addition). The logical operators receive multiple inputs and produce output contingent on those inputs. For example, the logical operator “IF (A = B) THEN C OR ELSE D” takes four numerical inputs (A, B, C, D), compares A and B for equality; it outputs (or “returns”) the

[2] Note that this section requires some knowledge of programming languages.

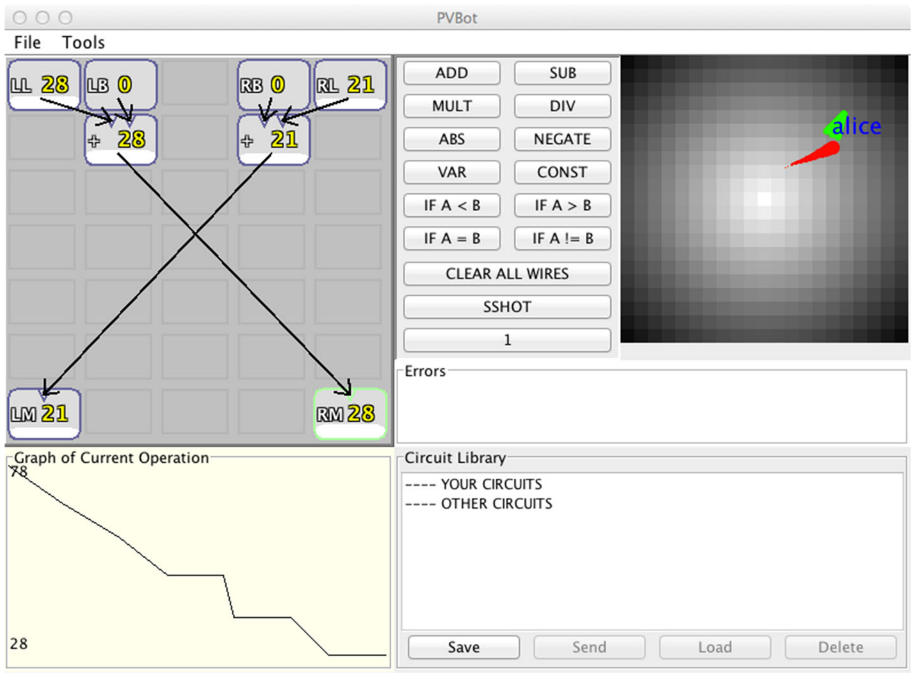


Fig. 1 VBOT client interface. In this example, the right motor will impel the robot forward at 28 % of maximum power, the value of adding the left light and left robot sensors. The function is reversed left to right but otherwise identical for the left motor. The student's VBOT ("alice") is on the *top right* (in the "field") in *green*, and its recent movement is tracked in *red* behind it. The lamp is the *white square* in the middle of the field, and light radiates from it

C input if A equals B, and it outputs D if A does not equal B.[3] Building blocks are connected with *wires*; wires connect outputs to inputs. A program that connects sensors to motors with wires and building blocks is called a *circuit*. Each circuit is evaluated for movement twice each second.

A major benefit of visualizing and building a program in terms of a circuit is that the programmer can follow both the logic of a signal through the circuit and understand the whole circuit by looking at the relationships between all of its elements simultaneously. However, it is important to note that this can limit the complexity of an end program. Hancock (2003) designed a novel programming language ("Flogo") also based to some extent on Braitenberg's circuit metaphor; he uses Flogo to teach computation to undergraduate and graduate students at MIT. Though we are targeting different populations, many of our initial design decisions came from his work (see Berland and Wilensky 2005, for more details).

While this work focuses primarily on the differences between the two environments, Berland (2008) covers implementation in significantly more detail, including a genealogy of design decisions, multiple detailed "play-throughs," and development details.

Differences Between Virtual and Physical Robotics

Some of the constraints for this work were as follows: It must be possible to use this material in real classrooms; it must be inexpensive to deploy; and it must not require hardware that is difficult to procure or that is proprietary to this project. Many otherwise excellent robotics research projects have been rendered ineffective due to the difficulty of deploying the work in real classrooms. Sipitakiat and Blikstein (2010) describe the difficulty of deploying even a very inexpensive proprietary robotics kit. As such, it was paramount to interact with off-the-shelf robotics kits readily available to classrooms. The most popular robotics kit at the time of deployment was LEGO Mindstorms.

Though there are benefits of using a kit that teaćhers can obtain, it is impossible to make the classroom-ready virtual and classroom-ready physical environments into fully equivalent technologies. Because we are evaluating, in part, the differences between virtual and physical design as they can be used in classrooms (rather than some philosophical difference between virtuality and physicality), making them fully equivalent would handicap either the physical or virtual environment or render them different enough from classroom use as to be inauthentic.

Virtual and physical robotics have each been implemented and researched in classroom settings hundreds of times, and two major differences between virtual and physical robots that stand out across those projects are (a) the speed of development and (b) in the specificity of the sensor data (e.g., Azhar et al. 2006; Druin and Hendler 2000). In this case of VBOT, the virtual classes had circuits downloaded to their robots wirelessly, but the physical classes had to plug their robots into their computer to upload circuits to the robots (which took more time). Second, as it is difficult and expensive to create sensors with

[3] In pseudocode, this would be:
function f(A,B,C,D) {if (A ==B) {return C;} else {return D;}}.

Table 1 VBOT activities

	Activity description
Orbit (day 1)	In the orbit activity, each student built a circuit to move her vbot to the middle of the screen so as to circle a light source. In the virtual class, the middle of the screen is marked by a virtual light source. In the physical class, the light sources were lamps. The activity is complete when all students complete a circuit successfully. This activity requires using only light sensors and motors
Flocking (day 2)	The goal of the flocking activity is to create a stable group ("flock") of bots that travel together away from the light. This activity requires vbot sensors, motors, and students can use a mathematical building block
Tag (day 3)	In tag, the goal for the class was to maximize total *tags*. A tag occurs when a vbot touches another vbot that has not been tagged before. The untagged vbot then becomes tagged. Logical operators are introduced
Soccer (days 4 and 5)	As a final activity, all students take part in a soccer game. In the physical class, the game looks much like traditional robot soccer (Sklar et al. 2003b); two teams, each consisting of 3 robots, attempt to push a ball into the other team's goal. The virtual class played a variant of robot soccer in which they attempted to move multiple balls into the goal

Each day, the students were presented with a challenge

reliable precision and accuracy (Martin 1996), inexpensive kits such as LEGO Mindstorms have inaccurate and imprecise ("noisy") sensors in a dynamic (real-world) space that pollute sensor data in a random-seeming manner. Due in part to noise, the behavior of virtual and physical robots is rarely identical, and we attempted to mitigate those differences in part by simulating noise similar to a physical sensor in the virtual environment, as per Maes (1990).

Procedure

VBOT Activities

VBOT includes an ordered set of activities for the classroom (described in Table 1). The activities are designed to support the students in building programming skills that were developed the previous day. Due to the differences between virtual and physical robots, the activities could not be identical in both classes; however, we were able to match the activities, so that the space of writable programs would be the same. For example, a circuit written by a student on day 3 in a physical class would work for a student on day 3 in a virtual class (and vice versa).

Research Design

A researcher co-taught the class with the class's regular teacher. The researcher taught VBOT syntax for ~30 min on day 1 and ~20 min on days 2 and 3 using two predetermined curricula. Other than that instruction, the sessions required minimal intervention. After day 3, the researchers' primary role was to aid the teacher in facilitating the activities in the classroom. The teacher was responsible for classroom management and describing activity instructions. Throughout the activity, the researchers provided technical support.

The first author met with each teacher for 1–2 h prior to the first day of the activities and after the last day of the activities. The meeting prior to the activities consisted of an interview about his or her students (~30 min), an opportunity to experiment with VBOT (~30 min), and a discussion of the teacher's and the researchers' roles in the classroom.

Data Collection

The data collected for this study consisted of two videotapes of each implementation day, full activity logs of each student's interactions with the vbot program, and pre-/-posttests. The tests were about 20 min long, and we describe them in detail below. The classroom enactments were videotaped using two cameras at all times. One camera focused on the activity of the whole classroom. Another camera focused on one group of students at a time, recording the behaviors and interactions of the students using the system in the activity.

The primary source of data for this paper is the activity log. On the activity log, we have record of every interaction with the software: when they created circuits, how they program circuits, every change they made to any circuit, every time they downloaded their circuit to their vbot, and every time they used the keyboard or mouse in VBOT. Our intention was that by capturing all of the interaction with the program by the students, we could understand the process by which they progressed in learning to program and learned to interact within the system. The pretest and posttest were designed to better understand that progress at a summative level. On the other hand, the video was primary used in this study only to verify our understanding of the results from the activity logs and tests.

Measures

Performance Measures

Our pre-/posttest measures are designed to gain insight into students' computational and complex systems thinking. The full text of all relevant questions can be found in the

"Appendix." All questions were graded on a scale of 0–3 by two graders with a 100 % inter-rater reliability: A score of 0 is an unanswered question; a score of 1 means "incorrect"; a score of 2 means "partially correct or shows some understanding"; and a score of 3 is "correct or shows full understanding." Note that question 3 is irrelevant to this paper and, as such, is not included.

Question 1 (Complex Systems Thinking) Q1 is designed to test a student's complex systems thinking using Jacobson and Wilensky's (2006) hierarchy of complex system understanding. The question presents a picture of birds flying in a V-shape; one of the birds has an arrow pointing to it with the name "Shelby." The description of the picture in the question reads, "When birds fly south for the winter, they often form a V-shape. You might have seen this in the sky. This is a picture of flock of birds flying in V-shape."

Wilensky and Resnick (1999) suggest that people tend to perceive biological complex systems from a deterministic-centralized perspective: that is, they posit (often erroneously) that a leader directly controls the system. As such, question 1 prompts students to evaluate flocking in terms of one agent (1A, "how does 'Shelby' know where to go?"), multiple agents (1B, "how do the birds know where to go?"), and aggregates (1C, "why do birds fly in a V-shape"). Levy and Wilensky (2008) show that these framings can affect how students perceive and understand levels thinking differently. We wrote these questions to be explicitly similar to those in Wilensky and Resnick (1999) and scored them in the same way. We used three rater-researchers, all of whom had previous experience with the scoring scheme, and, as such, we achieved a perfect inter-rater reliability of 100 %. This is not as surprising as it might seem, as there were only three possible codes on any answered sub-question ("incorrect," "partially correct," or "fully correct").

Question 2 (Computational Thinking) In Q2, each student both interpreted and modified a simple flowchart. Flowcharts have a deep connection to computer programming, but they do not require computer programming. In particular, this question was designed to evaluate a students' ability to follow and create computational logic. It is broken into three parts: (a) self-report of prior experience; (b) follow a flowchart with several branches; and (c) add to the flowchart to change its functionality. The self-report of prior experience (2A) did not correlate with the rest of Q2 on the pretest or the posttest, so its reliability is questionable and, as such, it is not used in this study. There was very little ambiguity in questions 2B and 2C, and 97 % of the students (all but 2) got them either fully right or fully wrong. As such, no human raters were used.

Question 3 (Essay Question on the Relationship Between Classroom Space and "Robot Space") Q3 was an essay-style question about how students navigate space. It was less relevant to this work and, as such, has been omitted for space.

Question 4 (Programming Skill) Question 4 requires students to build a VBOT circuit on a paper facsimile of an actual VBOT circuit board. It is well supported from a variety of theoretical perspectives, including our own, that by engaging students in a variety of authentic tasks, and evaluating those tasks, is an effective way of assessing their ability to perform that task. Question 4 describes four different scenarios and asks students to sketch out the circuit they would build for those scenarios. The four sub-questions were as follows:

4A. Wire up a vbot to go in a loop around the screen.
4B. Wire up a vbot that would make a smaller loop.
4C. Wire up a vbot that makes either loop using NO LIGHT SENSORS.
4D. Wire up a vbot that makes either loop using NO VBOT/BUMP SENSORS.

Questions 4A and 4B were designed to test student's understanding of programming a vbot without regard to the surrounding system, while questions 4C and 4D were designed to test student's understanding of programming a vbot in the context of a system (by restricting the sensors used). By restricting the sensors, the students were forced to consider contextual sources of data from the system, rather than program out of context. Though a transition from the VBOT itself to a paper posttest may seem inauthentic, the simplicity of vbot and the verisimilitude of the paper VBOT circuit board mitigated most difficulties. More than 90 % of the answers were either right or wrong without ambiguity, and the remaining answers were tested by inputting them as program code into VBOT, so that the VBOT software could evaluate their functionality. No human raters were used.

Activity Log Measures

The activity log measures were designed to reveal information about the types of circuits that students were building. There are three core quality metrics (density, difficulty, and uniqueness, described below), which were derived from salient quality metrics by Boehm et al. (1976).

Density is the number of working operations in a vbot circuit; it is a simple size metric. In the case of the basic light-finding circuit described by Braitenberg (1984) (in VBOT, it is "RM = LL; LM = LR"), the operation density is 4, as there are four wired operations on a working circuit. Density provides insight into the manner in which a student creates a circuit. High-density ("dense") circuits have accreted operations over time out of previously working circuits. Low-density ("sparse") circuits are short but often easier to understand, due to their brevity. It takes a relatively long time to build a dense circuit in VBOT; the interface privileges smaller circuits. Therefore, a student with high average density is building circuits in a higher relative "time per circuit." Density is not, in itself, a measure of understanding, but we can use density to compare how much a student's programs grow over time, see how often and when students restarted their programs, and see how large a students' program might grow. Density is not a performance measure, as size does not entail any valence. Indeed, a sparse circuit can often be the most appropriate. However, dense circuits can (but do not necessarily) deal with a variety of scenarios (e.g., many robots surrounding it; many balls surrounding it) with different actions resulting from each scenario. In complex systems thinking, successful models are often those that show robustness in a variety of conditions (Holland 1995), regardless of size. Computational thinking, on the other hand, often privileges simplicity (Basawapatna et al. 2011), possibly suggesting a low density. Thus, this measure should provide some information toward situating circuit building in those two domains.

Difficulty measures the number and amount of the "high-level" operations that a student uses per circuit. Logic operations (e.g., IF/THEN) are weighted double that of arithmetic operations (e.g., ADD), which are weighted double that of sensors and motors. Therefore, logic operations are measured to be four times more "difficult" than sensors or motors. This is based (roughly) on the relative frequency of the operations as logic operations occur 75.4 % less frequently than sensors and motors. Difficulty is a metric designed to measure the "adventurousness" and syntax vocabulary of a student.

Academic adventurousness and self-efficacy are often correlated with measures of understanding (Schunk 1983), and in the case of VBOT, adventurous students can see a wider variety of scenarios and behaviors in context. Students creating high difficulty programs are likely to have tested all of the operations and used high-level operations frequently. Students whose circuits show a high average difficulty but a low average density are most likely building simple circuits out of complicated elements. A student whose circuits show a high average difficulty and a high average density would likely be using several complicated operations simultaneously. A student whose circuits show a low average difficulty and a high average density is probably building a complicated circuit out of simple components.

In itself, difficulty is also not a measure of understanding or performance, but, instead, it measures the degree to which a student could successfully deploy complex operations. However, these complex operations are at the core of more advanced programming, suggesting that students writing more difficult circuits are primed to better understand computation in the future. On the other hand, work in complex systems thinking, such as Wolfram (2002), often privileges easily comprehensible rules (i.e., less difficult) from which more complex behavior will emerge.

Uniqueness is the number of unique operations per circuit. For instance, a circuit with the left light sensor (LL) attached to the right motor (RM) through an ADD would have a uniqueness of 3. A circuit that routed signal through two ADD operations would still have a uniqueness of 3 ($\text{ADD} = 1, \quad \text{LL} = 1, \quad \text{RM} = 1; \quad 1 + 1 + 1 = 3$). The uniqueness metric provides insight into the semantics of difficulty and density. A student with a high uniqueness and a high difficulty is creating an extremely complicated circuit, no matter the density. A student with a low uniqueness and a high density is creating a complicated circuit out of simple repeated elements. Uniqueness is a partial measure of efficiency, as VBOT is designed such that sets of more complex operations can often be collapsed into a single, higher-level operation, much like other functional programming languages (e.g., Scheme). Highly unique circuits are more efficiently written, and efficiency of programming is a core concept across most definitions of computational thinking. On the other hand, it is easier to predict the emergent behavior of a system with many simple elements rather than a few complex ones (Holland 1995), so complex systems thinking might privilege low uniqueness.

Furthermore, we performed simple counts of the number of modifications (circuits) per class, and we also calculated a mean per circuit of each individual element in a circuit. We describe the relationship of these activity log measures to computational and complex systems thinking in the discussion below. No metric of "program correctness" is possible, due to the shifting goals of the class and of individual students in playing the game. This is purposefully designed into VBOT, as part of the learning task is that students can fluidly take on different roles in a complex system. For instance, if a student played as a goalie, any measure of goals scored would be misleading, at best. Freed from explicit measures of performance, students could shift roles as frequently as needed.

Results

In this section, we will present: contrasting case studies of a group from a virtual VBOT class and a group from a physical VBOT class; analytics and statistics about the significant and salient differences between virtual and physical classes and overall performance metrics. We will present these data in terms of their impact on computational and complex systems thinking. In the discussion below, we will describe the relevance, implications, and impact of the results presented here.

This section is designed to highlight salient differences in the computational and complex systems thinking in students using the virtual VBOT and physical VBOT environments. Case studies, activity logs, and performance measures show very different patterns of activity and performance in the two groups. Comparing the two classes highlights the common benefits in constructionist robotics. The case studies provide insight into why the two systems support the learning goals differently; the logs provide a backbone of those case studies and provide a portrait of day-to-day patterns; and the pre-/posttest differences (described below) highlight the aspects supported and provide some evidence of progress toward learning goals.

These differences should be framed in terms of the differences in the classes and environments themselves. The classes using physical vbots dealt with both the benefits and vagaries of the physical world: physical robots are generally slower and more unpredictable than virtual robots. Of course, the physical world is also more intuitive and meaningful for novices, as we exist in it and have intuitions about physical movement (Papert 1980). These differences may sound obvious, but they frame our results. The differences between how the classes interacted with VBOT were, in most cases, a function of speed, meaning, and intuition.

Less than a minute of video would be sufficient for an informed observer to distinguish differences between the patterns of activity in a virtual and a physical classroom, even if no physical or virtual vbots were visible. The virtual classes are characterized by constant action on the computers punctuated by short discussion. In addition, there is a constant rumble of discussion between students at nearby computers. The class feels traditionally teacher oriented until the start of those activities in which everybody looks at the common screen in the front of the classroom. At that moment, the class explodes in yelling, movement, and students begin to tinker with their virtual vbots. The first comments heard, usually, are “there I am!” as students locate their individual vbot on the projected, shared screen. At particular decisive points in the action (near the end, often), students often analyze the actions of their fellow students: “Alice, move your bot! We need to cluster!” In contrast, the physical classes move around the room much more. This is made most obvious in Mr. Cleveland’s classroom in which he worked hard to maintain a teacher-centered environment. Despite his attempts to quell noise, however, students walked to other tables to discuss the design of their physical vbot behaviors during the vbot activities. These actions result in an environment that feels more like that of an undergraduate engineering laboratory than middle school classroom.

These differences between virtual and physical classes show up on the activity-by-minute graphs (see Fig. 2). The activity in the physical classrooms occurs in many short bursts, whereas the activity in the virtual classrooms is significantly more consistent, though it peaks around three quarters of the way into class as the culminating activity is reaching its end. The activity then trails off as students finalize their circuits for the day. In addition to differences in when students used the VBOT programming environment, they differed in what they did: Due to their constant work with the computers, the virtual classes produce far more circuits than the physical classes.

Circuit Progressions

The contrasting case studies below (Maribel’s group and Claudia’s group) exemplify the differences between the virtual and physical vbot classes as a whole. This section investigates a representative working circuit from the two groups that accumulated the most points in their respective classes during the last activity, one group is from a virtual class and one is from a physical class. By comparing them, we can highlight differences between the virtual and physical classes and provide context for differences. We can use these differences to highlight the ways that students learned computational and complex systems thinking.

These students exemplify both the qualitative and quantitative differences between the classes, though the groups themselves were not strictly “representative”—we selected these student groups not for generalizability across SES,

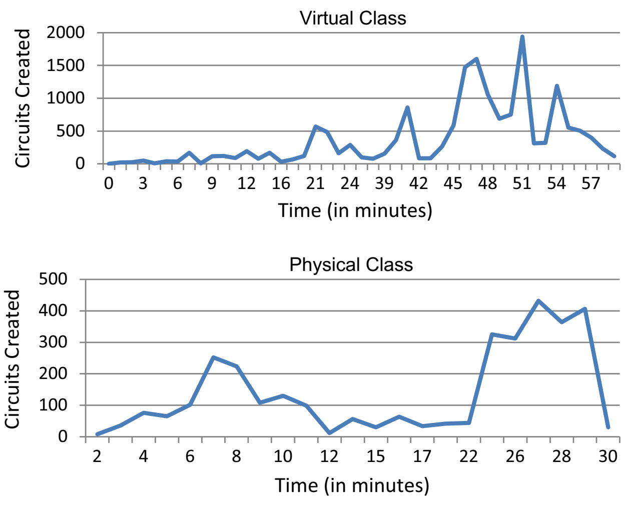


Fig. 2 Number of circuit changes per minute at Bayville in both the virtual (*top*) and physical (*bottom*) classes on Day 5

race, or ability, but because their experiences shed light on how and why computational and complex systems perspectives interrelate. That said, if a measure (on a particular day) from the physical class is higher than the measure from the virtual class, that relationship will also hold true for the relationship of the two groups in this section. This section serves primarily as a description of how specific students gained skill in programming VBOT in their groups.

Claudia's Group, Physical Class, Old Grove

Day 1: The first day of the physical class is spent inventing and investigating this circuit and testing it on the physical robots. Both the virtual and the physical classes start with the same, very basic circuit adapted from Braitenberg's LOVE circuit.

Day 2: This student's day 2 circuits consisted of various permutations of constants and light sensors. She also created a circuit identical to the circuit she had created on day 1, substituting bump sensors for light sensors. Her circuit was designed to move the robot forward at full speed with both motors working 100 %. However, due to the imprecise nature of the motors, each robot makes an arc rather than a perfectly straight line; the angle is contingent on the power of the motors attached to it. The student adapted her circuit to work with the imperfection, which took about 10 min. Those adaptations have no direct correlate in most virtual robotics, as virtual motors tend to be more predictable. That time was spent on the adaptations, rather than social competition or collaboration, focuses more of the student's attention on her individual robot; that was her own decision. This shows the beginning of what we call an *agent perspective*—a focus on the agent as it will perform in the system.

Day 3: Students played the Tag Activity on day 3. The goal of the game is for a student to program her vbot to swarm the light and knock away other robots. Claudia's best circuit did this exceptionally well. This circuit logic is as such: move toward the light until bumped; if bumped, move backward away from the light. This is a complicated circuit; her later circuits are not as complex. In part, this code builds on the code that she wrote on day 2 in which she was spending time working on her program rather than working extensively with other students. That this program was quite complex and successful suggests that she has considered how her own robot would act in context—an agent perspective on the system itself.

Day 4: Her circuits on day 4 were significantly simpler than the successful circuit that she created on day 3. It moves the robot directly forward at half speed until it touches another robot. At that point, the robot lurches forward at full speed; this is an effective goalie strategy. Again, she saw success by creating a self-consistent behavior for her robot to enact in a systems context. Her circuit helped determine the winner of a contest on day 4.

Day 5: Her primary day 5 circuit is an evolution of the goalie circuit on day 4. It moves toward the goal until it is touched, at which point it lurches forward at full speed. This is an efficient strategy for a midfielder. This circuit effectively won the game for its team of three robots by both pushing the ball toward the opposing team's goal-light, knocking both the ball and the other robots away.

All of Claudia's group's later circuits would only have been effective in a physical robotics setting. They all rely on the physical mass of the vbot, in that the circuits were designed to ram away other robots by using mass and speed and the physical mass of individual robots was not simulated in the virtual setting. This strategy was effective, and it gave Claudia's group significant confidence. Their robot was repeatedly described as "tough," and, indeed, they treated it as if it had a "tough" personality.

Other than the differences in perspective, a key feature to contrast between Claudia's circuit and Maribel's circuit below is that Claudia's circuits are much more complex (as measured by the metrics described above).

Maribel's Group, Virtual Class, Old Grove

Day 1: Maribel's day 1 circuits make little sense. She makes a long series of similarly incorrect circuits, suggesting that the group did not yet understand how to program. Several groups showed similar confusion, as it can take 20–30 min to learn basic programming in VBOT and not every group is equally engaged in the process. It is notable, however, that Maribel's circuits were unique in her class; she was testing her conceptions of programming and neither sharing with nor copying others' work. Her group was tinkering and working with their circuits, though they were not communicating effectively with other groups, nor did they program them correctly.

Day 2: The group tried a single circuit several times, tinkering with variations. This circuit causes a vbot to search out a light or goal at variable speed. Although the group produced several circuits that day, day 2 marked few significant deviations from basic working circuits. However, they did manage to make working programs with technically correct, if simplistic, program code.

Day 3: On day 3, the group tinkered with circuits relatively similar to a teacher-provided ("starter") circuit. Maribel's modifications were technically correct, but, again, they show few significant deviations from basic working circuits. At this point, Maribel is making circuits that work well in context—they tag other robots efficiently—but the code itself is quite simple. This is indicative of an *aggregate perspective*—a focus on the system as it works, rather than the robot itself. If her vbots were unsuccessful, simple circuits would seem like laziness, but their success suggests that she is considering the role of her simple robot in the context of other robots.

Day 4: Indeed, after tinkering, the group settled on a simple starter circuit. Again, this circuit did well in the context of the game activity that the students were playing. Since the goal of the activity was to tag students, Maribel's vbot stayed near the light and tagged all the students that passed by it.

Day 5: Day 5 provided further evidence of Maribel aggregate perspective. They saved two different simple, but relevant, circuits and switched quickly between them depending on the action of their classmates' vbots. The circuits, respectively, accelerated toward the light and toward other vbots. Few other students used this combination of circuits on the final day, and although the circuits were uncomplicated, they were effective. Her effectiveness at using the circuits in context suggested an understanding of the relative value and logic of the two circuits in the context of the game as played.

Perhaps the key feature of Maribel's group's work is that they learned to create simple circuits that work well in context. Her final circuits required some understanding of the functioning of the system as a whole to incorporate to function successfully. Maribel's actions require an integrated understanding of the ways that taking on a new role would affect the function of the team. For instance, if a player on, say, the Chicago Fire soccer team could freely switch between goalie and fullback, when would he choose to do that? To make that switch successfully, he would have to understand the roles and behaviors of the other players, including which other players might decide to also switch roles and when they would do so.

Contrasting Virtual and Physical Student Performance and Activity

While it is difficult to tease out all of the perceptual differences between virtual and physical interactions, it is possible to measure differences in effect and action. In this section, we will describe their contrasting patterns of behavior and describe ways in which the virtual and the physical classes' performance and activity differed. The physical class was most constructive in two distinct periods of programming activity in the first 30 min; this was followed by 30 min of testing their creations against each other in various scenarios (see Fig. 2). In the virtual class, programming activity was interspersed more evenly. This discrepancy was due, in part, to students' belief that physical robots must be tested against other physical robots. Note that this behavior was not required by the teacher or the setting, but rather, determined by the students; indeed, the teacher in these two classes was the same teacher. The virtual class was constantly and consistently changing their circuits. By working on their circuits individually and testing them against each other on the floor, the physical students had more opportunity to hone the individual behaviors of their robots in between testing session. In contrast, the virtual students wrote more program code more often.

The difference in patterns of behavior emerged, in part, from the different ways that students built circuits. We evaluated a student group's circuits on the basis of *uniqueness*, *difficulty*, and *density*. There existed very large significant differences overall between the means (by student) of virtual and the physical classes in uniqueness ($n = 33$, $p < 0.001$, $F = 181.04$, partial eta sq. = 0.92), difficulty ($n = 33$, $p < 0.001$, $F = 198.87$, partial eta sq. = 0.93), and density ($n = 33$, $p < 0.001$, $F = 127.78$, partial eta sq. = 0.89). Figure 3 shows the relative means.

As shown in Fig. 3, students in the physical classes typically created more unique, difficult, and dense circuits than students in the virtual classes. These three metrics are not necessarily correlated with positive performance, however. As seen in Fig. 4, uniqueness and density decreased each day over the course of the activities in both virtual and physical classes, while difficulty stayed roughly the same across all days. Figure 4 shows the graphs of these three metrics plotted against time. In this section, we will investigate several explanations for this.

As seen in Fig. 4, as students understood the system better over the course of the days, their circuits became more targeted and, often, simpler. This mirrors our analysis in the case studies above that learning to program did not necessarily result in higher metrics. Indeed, as the activities became more fast-paced and intricate, the circuits became simpler. That is not to say that all simple circuits were good circuits, but in Maribel's group above, we can see that optimal placement of a simple circuit achieved more success than others' more complex circuits elsewhere.

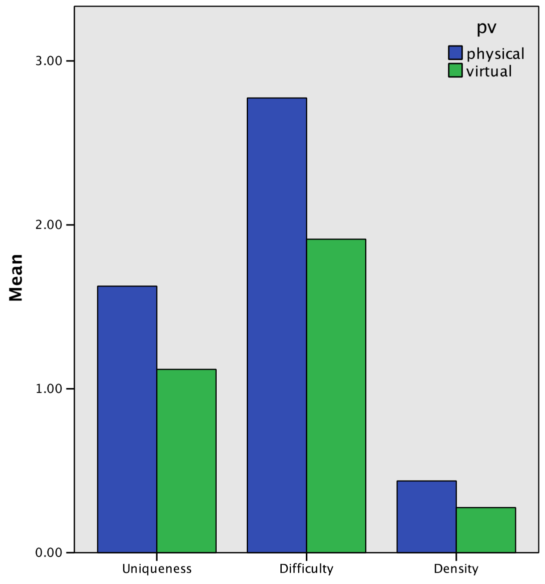


Fig. 3 Means of circuit metrics in virtual/physical classes

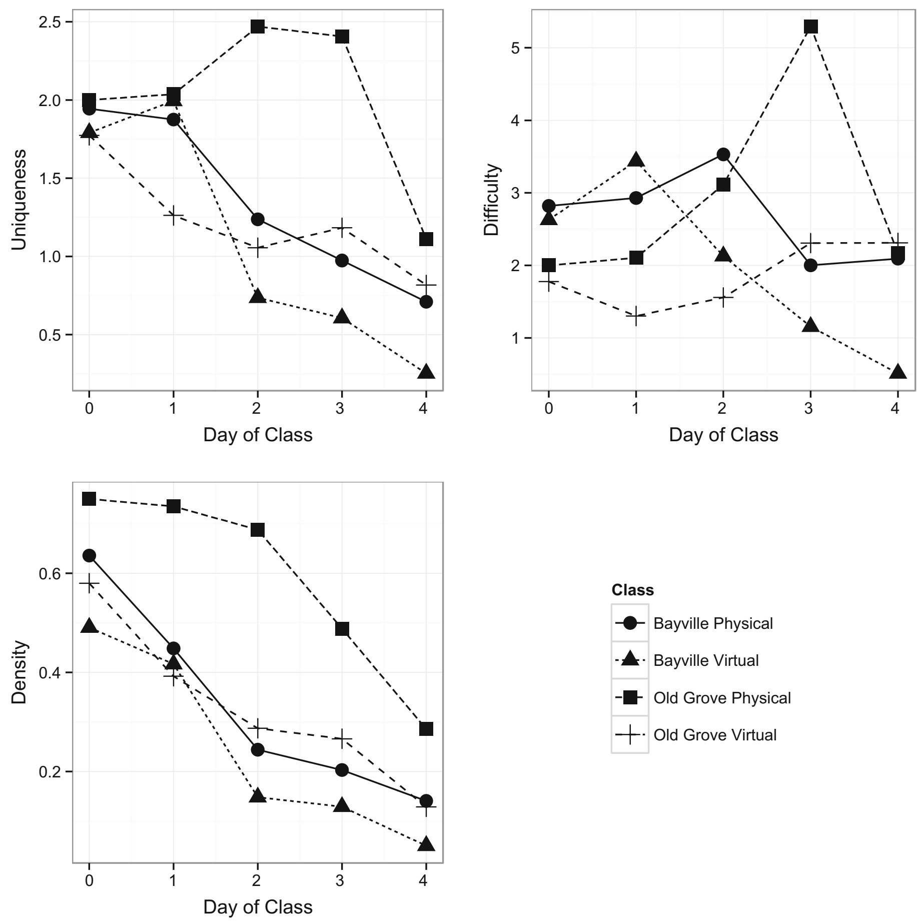


Fig. 4 Day-by-day graphs of the circuit metrics in each class

Table 2 ANCOVA for differences in pre-/posttest gain by physical/virtual classes within school

	Physical				Virtual				F	Sig	Part. Eta^2
	Pretest		Posttest		Pretest		Posttest				
	M	SE	M	SE	M	SE	M	SE			
Q 1A	2.60	0.24	2.64	0.24	2.70	0.20	3.02	0.20	1.85	0.15	0.07
Q 1B	1.76	0.14	1.67	0.18	1.76	0.14	2.23	0.15	2.82	0.04*	0.05
Q 1C	2.30	0.16	1.84	0.14	2.07	0.13	2.10	0.11	3.45	0.02*	0.12
Q 2B	1.88	0.19	2.21	0.18	1.93	0.15	2.73	0.15	1.61	0.19	0.06
Q 2C	1.58	0.18	2.00	0.19	2.08	0.15	2.62	0.16	0.16	0.92	0.01

Each question is on a scale of 0 (lowest grade) to 3 (highest grade)

* $p < 0.05$; ** $p < 0.01$; *** $p < 0.001$

Tables 2 and 3 both present models of pretest/posttest performance. Table 2 shows the mean differences between the pretest and posttest by type (virtual or physical) for all sub-questions of Q1 and Q2 and presents an ANCOVA of each of the components of Q1 and Q2 (e.g., Q1A Posttest by Physical [virtual, physical] within School [oldgrove, bayville] controlling for Q1A Pretest). Table 3 shows the average overall scores for Q4 by Physical [physical, virtual] and presented an ANOVA of the differences by Physical [virtual, physical] within School [oldgrove, bayville] for the components of Q4, which was the VBOT programming question (and did not have a corresponding pretest).

The physical classes were more likely to write *difficult* circuits using more constants and branching logic (IFs); this is consistent with the findings of time spent by physical students on individual programs. These relatively difficult operations did help them in the posttest performance metric question 4B, in which they did significantly better than their virtual class counterparts (posttest question 4B, $n = 33$, $p < 0.001$). However, the simplicity of the virtual class's circuits ostensibly helped them perform better on the slightly more difficult question 4D that addresses complex movement around the shared space (posttest question 4D, $n = 33$, $p < 0.05$). As described above, questions 4C and 4D were

Table 3 ANOVA for differences between physical/virtual classes within school

	Physical		Virtual		F	Sig.	Part. Eta^2
	M	SD	M	SD			
Q 4A	2.24	1.23	2.42	0.96	0.22	0.88	0.01
Q 4B	0.94	0.98	2.49	0.96	18.10	0.00***	0.46
Q 4C	2.06	1.28	1.79	1.06	1.64	0.19	0.05
Q 4D	1.21	1.34	1.63	1.07	2.99	0.04*	0.11

Each question is on a scale of 0 (lowest grade) to 3 (highest grade)

* $p < 0.05$; ** $p < 0.01$; *** $p < 0.001$

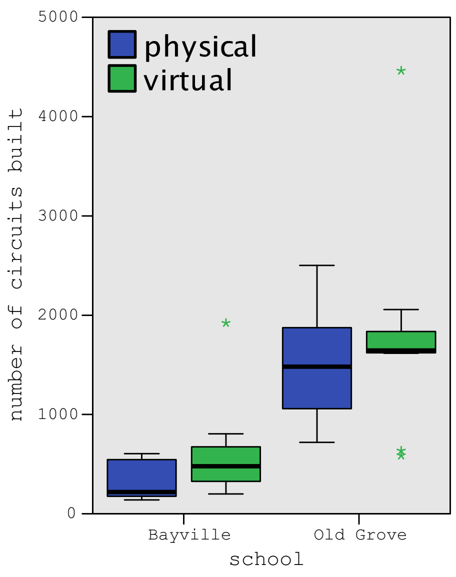


Fig. 5 Activity metric averaged by virtual/physical classes across schools

designed to test students understanding of multi-robot systems and the aggregate behavior of multiple robots.

Virtual class students spent the entire time at their desks building circuits—the physical classes spent only part of their class time doing so. The number of circuits the virtual classes built was only about 20 % more, on average; we expected greater magnitude in the difference. Figure 5 (above) shows how the classes differed across conditions between the two schools.

The data suggest that the virtual class learned concepts that the physical class did not and vice versa. The physical class used their understandings of the robot as a physical object as a basis for their primary mode of programming. Claudia's group designed most of its circuits for the specific goal of navigating and bumping other robots. The students' understandings of the capabilities of a robot were mediated by the physical presence of the robot. As such, physical class students used more CONSTANT operations and more IF/THEN operations, both of which are more easily adaptable to sensor noise. Virtual students, on the other hand, worked more contextually. Students in the virtual classes built more, simpler circuits that worked best for specific contexts. As shown, Maribel's group took into consideration the roles that her fellow students played and used those circuits that would be most appropriate. These data suggest that she could understand and exploit the emergent nature of the system. As such, we can see the ways in which even subtleties in the differences between physical environments and virtual ones can have effects.

Discussion

As we have seen, both the physical and the virtual classes made gains in understanding complex systems. However, looking more closely, the gains for the two groups had a different "character." Log data, interaction patterns, and pretest/posttest differences suggest that the physical class students tended to understand the systems from a more agent-based perspective. The virtual students, on the other hand, tended to understand the systems from a more aggregate perspective. Evidence for this finding can be seen across the results presented.

Physical class students created more complex individual robots; they focused on improving the agent's behavior rather than maximizing the function of the system (i.e., team goals scored). Figure 3 shows that the physical class students created more complex and more difficult circuits. They spent more time working per circuit. Claudia's group designed circuit behaviors in which the robot would move and act independently. The virtual class students, on the other hand, created circuits that worked in context. They created more circuits overall, and those circuits were simpler. Maribel's group created contextually relevant circuits, and they created these circuits quickly to respond to the actions of system as whole.

This finding is corroborated by the posttest results, which show that the physical students performed significantly better on question 4B (about directing agent behavior), while the virtual students performed significantly better on question 4D (about the behavior of a vbot in the context of other vbots). As stated earlier, both classes performed relatively similarly overall on question 4 (programming vbots). These differences suggest that while both sets of classes were learning how to program, they did so differently.

Why Did the Virtual and Physical Classes Differ?

Our data suggest that virtuality enabled students to think about their agents from a more aggregate perspective. In particular, speed allowed students to see the effects of an agent on the system as a whole, the ability to see multiple representations simultaneously enabled them to frame the actions of their agent in its system, and the ease of modification enabled students to do low-level tinkering. Likewise, fundamental features of physicality enabled physical class students to think about the system from a more agent-based perspective. Students could take the time to communicate with their groups in person and in depth, and that depth and lack of constantly updating new data allowed them to improve specific programs that they were writing and to think significantly more about how an individual agent might affect a system. Therefore, while both virtual class students and physical class students exhibited significant improvements in levels thinking, the virtual classes did so from an aggregate perspective, while the physical classes did so from an agent-based perspective, i.e., students in the physical environment better understood how individual agents contributed to emergent systems behavior, whereas students in the virtual environment showed greater understanding of how systems effects impacted individual agents. To use another example, it is as if students in the physical environment better understood how individual agents (robots) could make a goal, whereas students in the virtual environment better understood how the complex interaction of multiple robots impacted individual robots and goals scored. In regard to previous "levels" literature (Wilensky and Resnick 1999), it is as if students in the physical environment could better understand how individual cars could cause a traffic jam, whereas students in the virtual environment could better understand how the traffic jam could move backward. It is not new to frame physicality/virtuality in terms of speed and data input—much modern HCI work is framed on this difference (Ishii 2008). This is, however, the first study to frame and investigate these differences in terms of computational thinking and systems thinking.

That students were able to learn systems content by engaging in computational practices suggests that complex systems and computational thinking can be mutually reinforcing. Harel and Papert's (1990) "Integrated Learning Principle" suggests that "learning more can be easier than learning less"—when complex content is framed comprehensibly and in context, it is often easier to learn than when presented simplistically out of context. Work such as Wolfram's (2002) and Goldstone and Wilensky's (2008) suggest that complex systems theory is more easily available as a mode for understanding science precisely because of the relative power and comprehensibility of scientific computation, even in a context in which many believe programming is inherently difficult. However, there are few studies that examine the relationship between computational thinking and complex systems thinking at the middle school level, and in this study, students made real progress in learning to program while engaging with complex systems. This interplay appears valuable. Indeed, this suggests that it may be valuable to design curricula in which students engage both with virtual simulations (such as NetLogo) and physical manipulatives (such as LEGO Mindstorms) around the same content.

This work is a study of one system (VBOT) used to teach two intertwined learning goals (complex systems and computational thinking) to a specific population (middle school students). This study was situated in four middle school classrooms from two different Chicago public schools over the course of 1 week. From this perspective, this is an enormous task—these students have no programming experience and many of them are doing quite poorly in traditional academic settings. To teach these students about computer programming and complex systems content simultaneously might seem too much content, taught too quickly. However, students learned not only how to program in a proprietary system, but they did so in a complex systems environment, in a way that shows remarkable understanding and improvement. Our results led us to conclude that complex systems and computational thinking provide *each other* a meaningful context. Moreover, findings that students' physical and virtual *perspectives* framed how they learned content indicate potential benefits to considering how those perspectives might reinforce one another when designing learning environments.

Overall, we found that students in all classes learned as a result of being able to play relatively freely with the system. The amount of time given by the individual teachers, the design of the class material, and the school environments mattered far less than simply motivating students to share and tinker with the system. What we found was that students learned a great deal across the board: A significant majority of students learned how to program in VBOT, they could collaborate and compete in the classroom using VBOT, and they could take that knowledge and apply it.

Learning, playing, and sharing are complex activities. By tinkering, playing, and sharing, students came to better understand a complex and complicated set of content in a short period of time. Our findings suggest that complex systems and computational thinking can be mutually reinforcing because of the agent and aggregate perspectives that the students adopt. Many of these students entered the project with a view of robots as Martian invaders and left the project believing that they knew how to program robots. Other students came in assuming that every flock of birds had a predetermined leader and that every anthill was "radio-controlled" by a hidden queen. These same students came to understand that sometimes the only way to a common goal was an array of nearly identical simple agents.

In this project, the tools provided affordances for students to learn in equivalent virtual and physical constructionist learning environments and they interacted with those environments meaningfully but differently. It is our belief that by adapting the environment design to facilitate cognitive learning goals and that by having a better understanding of students' perspectives when designing those environments, we will be able to better teach students to work and reason with complex content.

Appendix: Pretest and Posttest

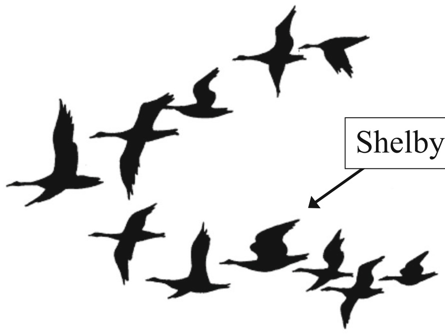
Pre-Questionnaire

Name:

Date:

Just write what you think! Answer each question with a sentence or two.

Thank you!

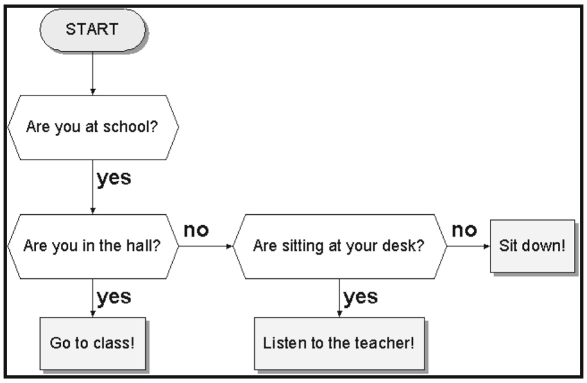


Q1
When birds fly south for the winter, they often form a V-shape. You might have seen this in the sky. This is a picture of flock of birds flying in V-shape:

How does Shelby, the bird, know where to fly in the V-shape?

How do the birds know where to go when they are flying in a V-shape?

Why do birds fly in V-shapes?



Q2

Imagine that this "flowchart" describes your day at school. You can follow the flowchart by answering questions about your day. Depending on your answers to the questions, it tells you what to do next. Start at "START" (in the image above) and follow the arrows.

Have you ever seen a flowchart before? YES NO

Using the chart, list the things that happen during a day of school.

Change the flowchart so that if you are not at school, you read a book. (*Draw on the picture above.)*

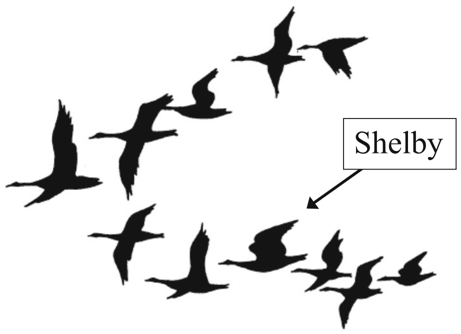
Post-Questionnaire

Name:

Date:

Just write what you think! Answer each question with a sentence or two.

Thank you!



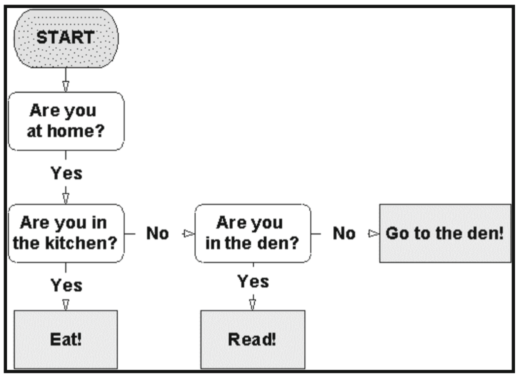
Q1

When birds fly south for the winter, they often form a V-shape. You might have seen this in the sky. This is a picture of flock of birds flying in V-shape:

How does Shelby, the bird, know where to fly in the V-shape?

How do the birds know where to go when they are flying in a V-shape?

Why do birds fly in V-shapes?



Q2

Imagine that this “flowchart” describes a day at home. You can follow the flowchart by answering questions about that day. Depending on your answers to the questions, it tells you what to do next. Start at “START” (in the image above) and follow the arrows.

Using the chart, list the things that happen during this day at home.

Change the flowchart so that if you are not at home, you go home. (*Draw on the picture above.)*

Q4

Your vbot is the airplane in the picture. The arrows in the middle are other people’s vbots, and they are stopped in the middle of the screen. There is a light source in the middle of the screen.

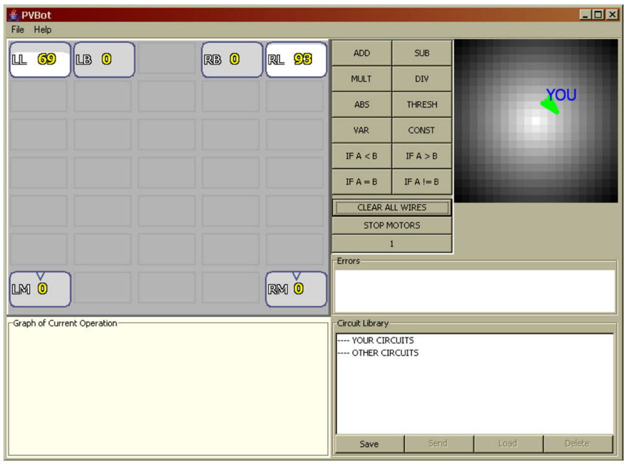
4.A. Wire up a vbot to go in a loop around the screen.
(Use attached vbot breadboard. ***Circle Q 4.A. at the top of the sheet.)***

4.B. Wire up a vbot that would make a smaller loop.
(*Use attached vbot breadboard.* ***Circle Q 4.B. at the top of the sheet.****)*

4.C. Wire up a vbot that makes either loop using NO LIGHT SENSORS (LL and LR)?
(*Use attached vbot breadboard.* ***Circle Q 4.C. at the top of the sheet.****)*

4.D. Wire up a vbot that makes either loop using NO VBOT SENSORS (LB and RB)?
(*Use attached vbot breadboard.* ***Circle Q 4.D. at the top of the sheet.****)*

Circle one:
Question 4.A.
Question 4.B.
Question 4.C.
Question 4.D.



References

Azhar MQ, Goldman R, Sklar E (2006) An agent-oriented behavior-based interface framework for educational robotics. In: Proceedings of the conference on autonomous agents and multiagent systems (AAMAS 2006)

Basawapatna A, Koh KH, Repenning A, Webb DC, Marshall KS (2011) Recognizing computational thinking patterns. In: Proceedings of the 42nd ACM technical symposium on computer science education. SIGCSE 2011, pp 245–250

Ben-Ari M (2001) Constructivism in computer science education. J Comput Math Sci Teach 20(1):45–73

Berland M (2008) VBOT: Motivating complex systems and computational literacies in virtual and physical robotics learning environments. Retrieved from ProQuest Digital Dissertations. AAT 3307005

Berland M, Wilensky U (2005) Complex play systems—results from a classroom implementation of VBOT. In: The annual meeting of the American Educational Research Association, Montreal, Canada, April 11–15, 2005

Berland M, Wilensky U (2008) VBOT (computer software)

Berland M, Martin T, Benton T, Petrick C (2011) Programming on the move: design lessons from IPRO. In: Proceedings of ACM SIG-CHI 2011, pp 2149–2154

Berland M, Martin T, Benton T, Smith CP, Davis D (2013) Using learning analytics to understand the learning pathways of novice programmers. J Learn Sci 22(4):564–599. doi: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 agent-based modeling. Int J Comput Math Learn 14(2):81–119

Boehm BW, Brown JR, Lipow M (1976) Quantitative evaluation of software quality. In: Proceedings of the 2nd international conference of software engineering, Los Alamitos, CA

Braitenberg V (1984) Vehicles: Experiments in synthetic psychology. MIT Press, Cambridge, MA

Bundy A (2007) Computational thinking is pervasive. J Sci Pract Comput 1(2):67–69

Chi M (2005) Commonsense conceptions of emergent processes: why some misconceptions are robust. J Learn Sci 14(2):161–199

Cobb P, Confrey J, diSessa A, Lehrer R (2003) Design experiments in educational research. Educ Res 32(1):9–13

Colella V (2000) Participatory simulations: building collaborative understanding through immersive dynamic modeling. J Learn Sci 9(4):471–500

Collier N (2003) Repast: an extensible framework for agent simulation. The University of Chicago's Social Science Research, p 36

Collins A, Joseph D, Bielaczyc K (2004) Design research: theoretical and methodological issues. J Learn Sci 13(1):15–42

Davis B, Sumara D (2006) Complexity and education: inquiries into learning, teaching, and research. Lawrence Erlbaum, Mahwah, NJ

diSessa A (2001) Changing minds: computers, learning, and literacy. MIT Press, Cambridge, MA

diSessa A, Cobb P (2004) Ontological innovation and the role of theory in design experiments. J Learn Sci 13(1):77–103

Druin A, Hendler JA (2000) Robots for kids: exploring new technologies for learning. Morgan Kaufmann, Burlington

Goldstone RL, Wilensky U (2008) Promoting transfer by grounding complex systems principles. J Learn Sci 17(4):465–516

Grotzer TA, Basca BB (2003) How does grasping the underlying causal structures of ecosystems impact students' understanding? J Biol Educ 38(1):16–29

Guzdial M, Forte A (2005) Design process for a non-majors computing course. ACM SIGCSE Bulletin 37(1):361–365

Hancock C (2003) Real-time programming and the big ideas of computational literacy. Unpublished doctoral dissertation, MIT, Cambridge, MA

Harel I, Papert S (1990) Software design as a learning environment. Interact Learn Environ 1(1):1–32

Hmelo CE, Holton DL, Kolodner JL (2000) Designing to learn about complex systems. J Learn Sci 9(3):247–298

Hmelo-Silver C, Pfeffer MG (2004) Comparing expert and novice understanding of a complex system from the perspective of structures, behaviors, and functions. Cogn Sci 28(1):127–138

Holland JH (1995) Hidden order: how adaptation builds complexity. Basic Books

Holland J (1999) Emergence: from chaos to order. Basic Books, New York, NY

Ioannidou A, Repenning A, Lewis C, Cherry G, Rader C (2003) Making constructionism work in the classroom. Int J Comput Math Learn 8(1):63–108

Ishii H (2008) Tangible bits: beyond pixels. In: Proceedings of the 2nd international ACM conference on tangible and embedded interaction, pp xv–xxv

Jacobson M, Wilensky U (2006) Complex systems in education: scientific and educational importance and implications for the learning sciences. J Learn Sci 15(1):11–34

Johnson S (2002) Emergence: the connected lives of ants, brains, cities, and software. Scribner, New York, NY

Kelleher C, Pausch R, Kiesler S (2007) Storytelling ALICE motivates middle school girls to learn computer programming. In: Proceedings of the SIGCHI conference on Human factors in computing systems, pp 1455–1464. San Jose, CA, April 28–May 3, 2007

Klopfer E, Colella V, Resnick M (2002) New paths on a StarLogo adventure. Comput Graph 26(4):615–622

Klopfer E, Yoon S, Rivas L (2004) Comparative analysis of palm and wearable computers for participatory simulations. J Comput Assist Learn 20(5):347–359

Klopfer E, Yoon S, Um T (2005) Young adventurers—modeling of complex dynamic systems with elementary and middle-school students. J Comput Math Sci Teach 24(2):157–178

Lahtinen E, Ala-Mutka K, Järvinen HM (2005) A study of the difficulties of novice programmers. ACM SIGCSE Bull 37(3):14–18

Levy ST, Wilensky U (2008) Inventing a "mid level" to make ends meet: reasoning between the levels of complexity. Cogn Instruct 26(1):1–47

Luke S, Cioffi-Revilla C, Panait L, Sullivan K, Balan G (2005) MASON: a multiagent simulation environment. Simulation 81(7):517

Maes P (1990) Designing autonomous agents: theory and practice from biology to engineering and back. MIT Press, Cambridge, MA

Martin FG (1996) Ideal and real systems: a study of notions of control in undergraduates who design robots. In: Kafai Y, Resnick M (eds) Constructionism in practice: rethinking the roles of technology in learning. MIT Press, Cambridge, MA

Martin T, Berland M, Benton T, Smith CP (2013) Learning programming with IPRO: the effects of a mobile, social programming environment. J Interact Learn Res 24(3):301–328

National Research Council (2010) Report of a workshop on the scope and nature of computational thinking. National Academies Press, Washington, DC

Papert S (1975) Teaching children thinking. J Struct Lang 4:219–229

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

Parker LE, Schultz A (eds) (2005) Multi-robot systems: from swarms to intelligent automata, vol III. Kluwer, Netherlands

Pea RD (1987) Cognitive technologies for mathematics education. In: Schoenfeld A (ed) Cognitive science and mathematics education. Lawrence Erlbaum Associates Inc, Hillsdale, NJ, pp 89–122

Pea RD, Kurland DM (1984) On the cognitive effects of learning computer programming. New Ideas Psychol 2(2):137–168

Penner DE (2000) Explaining systems: investigating middle school students' understanding of emergent phenomena. J Res Sci Teach 37(8):784–806

Perkins DN, Grotzer TA (2005) Dimensions of causal understanding: the role of complex causal models in students' understanding of science. Stud Sci Edu 41(1):117–166

Portsmore M (2005) ROBOLAB: intuitive robotic programming software to support lifelong learning. Apple learning technology review. Spring/Summer, 2005

Resnick M (2003) Thinking like a tree (and other forms of ecological thinking). Int J Comput Math Learn 8(1):43–62

Resnick M, Ocko S, Papert S (1988) LEGO, logo, and design. Child Environ Q 5(4):14–18

Resnick M, Wilensky U (1998) Diving into complexity: developing probabilistic decentralized thinking through role-playing activities. J Learn Sci 7(2):153–172

Schoenfeld AH (1992) Learning to think mathematically: problem solving, metacognition, and sense making in mathematics. Handbook of research on mathematics teaching and learning, pp 334–370

Schunk DH (1983) Ability versus effort attributional feedback: differential effects on self-efficacy and achievement. J Educ Psychol 75(6):848

Schweikardt E, Gross MD (2006) roBlocks: a robotic construction kit for mathematics and science education. Proceedings of the 8th international conference on Multimodal interfaces, pp 72–75

Sengupta P, Wilensky U (2009) Learning electricity with NIELS: thinking with electrons and thinking in levels. Int J Comput Math Learn 14(1):21–50

Sharlin E, Watson BA, Kitamura Y, Kishino F, Itoh Y (2004) On humans, spatiality and tangible user interfaces. Pervasive Ubiquitous Comput 8(5), 338–346. Theme issue on tangible interfaces in perspective

Sipitakiat A, Blikstein P (2010) Think globally, build locally: a technological platform for low-cost, open-source, locally-assembled programmable bricks for education. In: Presented at the conference on tangible, embedded, and embodied interaction TEI 2010, Cambridge, USA

Sklar E, Eguchi A, Johnson J (2003a) RoboCupJunior: learning with educational robotics. RoboCup 2002: robot soccer world cup VI, pp 238–253

Sklar E, Parsons S, Stone P (2003b) Robocup in higher education: a preliminary report. In: Proceedings of the 7th RoboCup symposium

Soloway E (1986) Learning to program = learning to construct mechanisms and explanations. Commun ACM 29(9):850–858

Wilensky U (1999) NetLogo [Computer software]. Evanston, IL: Northwestern University, Center for Connected Learning and Computer-Based Modeling. Retrieved September 20, 2011, from http://ccl.northwestern.edu/netlogo

Wilensky U (2003) Statistical mechanics for secondary school: the GasLab modeling toolkit. Int J Comput Math Learn 8(1):1–4

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 Instruct 24(2):171–209

Wilensky U, Resnick M (1999) Thinking in levels: a dynamic systems perspective to making sense of the world. J Sci Educ Technol 8(1):3–19

Wilensky U, Stroup W (1999a) Learning through participatory simulations: network-based design for systems learning in classrooms. In: Proceedings of the 1999 conference on computer support for collaborative learning, CSCL '99 Palo Alto, CA

Wilensky U, Stroup W (1999b) HubNet [Computer software]. Northwestern University, Center for Connected Learning and Computer-Based Modeling, Evanston, IL

Wing JM (2006) Computational thinking. Commun ACM 49(3):33–35

Wolfram S (2002) A new kind of science. Wolfram Media, Champaign, IL

Wyeth P (2008) How young children learn to program with sensor, action, and logic blocks. J Learn Sci 17(4):517–550