The Consume - Create Spectrum : Balancing Convenience and Computational Thinking in STEM Learning

Ashok Basawapatna
University of Colorado Boulder
Department of Computer Science
Boulder, CO. 80303 USA
basawapa@colorado.edu

Alexander Repenning
University of Colorado Boulder
Department of Computer Science
Boulder, CO. 80303 USA
ralex@cs.colorado.edu

Kyu Han Koh
University of Colorado Boulder
Department of Computer Science
Boulder, CO. 80303 USA
kohkh@colorado.edu

Mark Savignano
University of Northern Colorado, Dept. of Education
1868 Ute Creek Dr
Longmont, Colorado
Savi6521@bears.unco.edu

ABSTRACT

Future school science standards, such as the Next Generation Science Standards (NGSS), emphasize the integration of simulation and modeling activities in the classroom environment. The extremes of these activities have two vastly different implementations. On one hand, a teacher can have students experiment on a pre-made simulation associated with the material. On the other hand, students can use, for example, an end-user programming tool to create the simulation from scratch. This allows students to not only experiment on, but also, to model the real world phenomenon being studied– a key component of computational thinking. However, the greater amount of time necessary for student authoring of simulations can make such an approach infeasible in the classroom environment. This paper presents a spectrum of strategies for integrating simulations into classrooms emphasizing our research at the Scalable Game Design Lab, University of Colorado Boulder as well as research from other entities. Starting at consuming simulations and adding more user interaction and authoring elements begins to provide a gentle slope from consumption towards simulation creation. Results indicate that many of these strategies are quite effective.

Categories and Subject Descriptors

K.3.2 [**Computing Milieux**]: Computers and Education—*Computer Uses in Education*

General Terms

Algorithms, Design, Experimentation, Human Factors

SIGCSE'14, March 3–8, 2014, Atlanta, GA, USA.

http://dx.doi.org/10.1145/2538862.2538950 .

Keywords

High School Computer Education, Middle School Computer Education, Simulation and Modeling, Computational Thinking, Next Generation Science Standards

1. INTRODUCTION

Classrooms across the United States are looking for ways to integrate aspects of computational thinking into their curriculum [15]. As of now, computational thinking is defined as a problem solving process that includes the following: *Formulating Problems* such that they can be solved by a computer, *Logically Organizing and Analyzing Data*, *Representing Data Through Abstractions* such as models and simulations, *Automating Solutions* through algorithmic thinking, *Identifying Analyzing and Implementing Solutions Efficiently*, and *Generalizing and Transferring This Process* to solve a wide variety of problems [3].

Activities related to simulation authoring offer a unique opportunity to integrate computational thinking into the classroom environment. Future standards, such as the Next Generation Science Standards (NGSS) and the Common Core Curriculum Standards (CCC) emphasize the use of simulations. For example, the NGSS states in its *Structure, Function, and Information Processing* section that, in grades 9-12, "*students who demonstrate understanding can use modeling to explain the function of positive and negative feedback mechanisms in maintaining homeostasis that is essential for organisms* [2]."

Simulation authoring enables classrooms, such as Life Science and Biology classes for example, to meet upcoming science standards and effectively integrate computational thinking. However, at present time, very few teachers across the United States are employing simulation or modeling activities [10]. One major barrier to integrating simulations authoring activities is finding the class time as well as appropriate activities [6].

In our experiences, at the University of Colorado Scalable Game Design Lab, we have students successfully author simulations employing the end-user agent-based programming tool AgentSheets and its subsequent 3D version

AgentCubes. AgentSheets and AgentCubes is meant for students with little or no prior programming experience enabling them to make their first visual game or simulation within five hours [18]. Though it may seem like a relatively small amount of time, five hours equals a 1 week project which can be prohibitive in classrooms.

There are alternatives to authoring simulations from scratch that are time efficient, and begin to integrate computational thinking concepts. For a particular classroom's constraints, one might wonder if one of these alternatives yields a "sweet spot" between convenience and computational thinking efficacy. In this paper we attempt to outline a variety of options that create a gentle-slope path, from consuming pre-made simulations to authoring simulations, taken from both our research experiences and others. It is our aim to give teachers, curriculum designers, and researchers insight into the spectrum of simulation related classroom activities.

The remainder of this paper will outline and discuss this space in-depth along with data and observations from novel approaches, contained within this spectrum.

2. GENTLE SLOPE APPROACH TOWARDS SIMULATION AUTHORING

Figure 1 is a depiction of how computational thinking relates to simulation creation; the basic problem solving process outlined involves model development, computational expression of ideas, running the simulation code, visualizing the consequences of this implementation, and possibly revising the model based on this visualization. The subtext of this figure is that the student develops a strategy to gain insight into a real world problem through the use of simulations in the representational domain. The ability of a student to create a simulation enables the investigation depicted in Figure 1. Furthermore, Figure 1 begins to give us a criteria to evaluate the extent a simulation related activity contains computational thinking concepts.

Figure 2 depicts the Consume-Create spectrum of simulation activities that range from convenient activities to those that heavily integrate computational thinking. These include animations, interactive simulations, collective simulations, construction set simulations, pattern based authoring, end-user programming, and traditional programming languages.

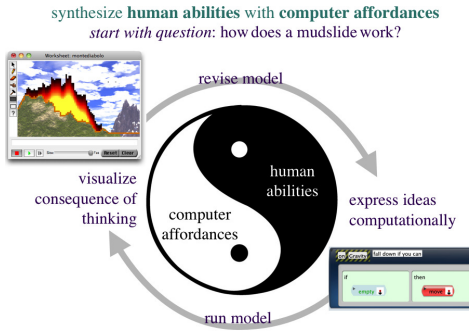

Figure 1: How creating simulations relates to computational thinking. Users start with a question, develop a model, express the model computationally, run the model, visualize the consequence of their thinking and possibly revise the model

It should be noted that Figure 2 is by no means an exhaustive list but serves as a good beginning point for a discussion of simulation use in the classroom environment. Starting from the left and going to the right of Figure 2, we see activities that begin to offer the user greater control over the simulation mechanics itself by enabling higher degree of expressive freedom. Furthermore, with each item there exists limitations that are overcome by traveling to the right. For example, one way an animation, located on the far left, is limited is that the animation does not enable changing a parameter associated with the animated phenomenon. By giving the user an interactive simulation instead, located to the right of animations, eliminates this limitation providing a more open-ended exploration of the phenomenon. Conversely, for any activity in Figure 2, we see that this activity contains the option of doing all the activities to the left of it. For example, by using a programming language, like C++ or Java, the user hypothetically has the ability to do everything offered through end-user programming all the way to creating a simulation animation that can be displayed to enhance student understanding (though such a tact might be infeasible for students with little programming experience). What follows is an analysis of each of the activities listed in Figure 2 emphasizing their benefits as an in-class activity, including supporting science learning goals, and a focused appraisal of their relationship to computational thinking.

2.1 Animations

Scientific animations are not simulations per se, but similar to simulations in that they create a representation of a scientific phenomena. Animation activities entail a student watching a movie depicting a representation of a given scientific system, often times accompanied by a narrative. Examples of these activities include the suite of animations on the BrainPop[1] website and The Institute for Chemistry Literacy through Computational Science (ICLCS) [2]. Figure 3 depicts screenshots of example animation activities.

In the left activity depicted in Figure 3, students watch an animated model of ice melting on a molecular level. These activities typically take minutes to complete (depending on the length of the animation) and are often accompanied by text that describes the phenomenon being depicted. This activity is usually accompanied by a worksheet for students to help synthesize the academically one way lesson. This pedagogy of learning regards to Bloom's Taxonomy revised of learning ranks on the pyramid– Factual knowledge/Remember [14]. However useful for efficiency when navigating the curriculum, the students do not get a chance to interact and synthesize learning.

By running an animation, students are not using any computational thinking skills– students do not have to create an algorithm or abstract the real world into the representational domain. However, there is good evidence that these visual representations do yield better student understanding. One large scale study, for example, claims that students in classes that used BrainPOP increased their science scores on the SAT 10 test by one to two grade levels, when compared to control classrooms [1].

2.2 Interactive Simulations

[1] www.brainpop.com
[2] iclcs.illinois.edu/

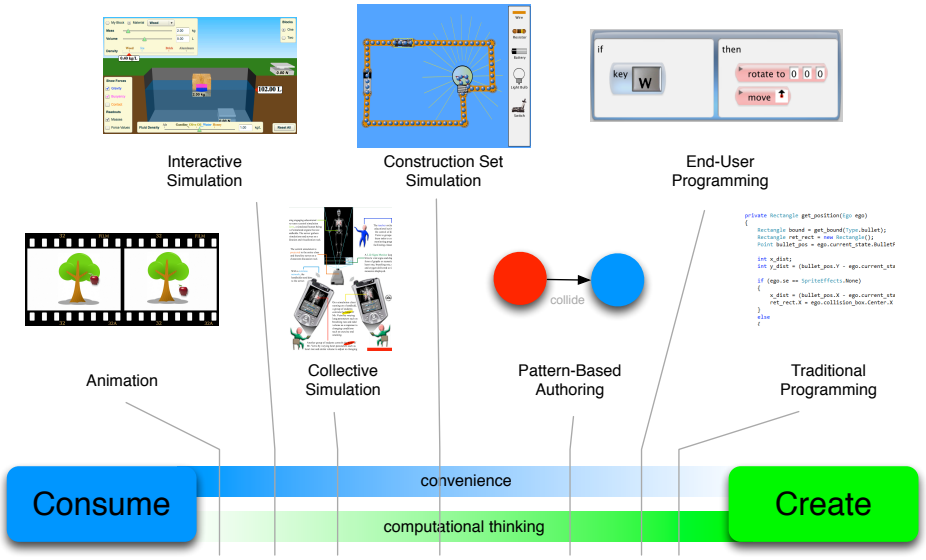

Figure 2: Gentle slope spectrum of options from using (convenient) to authoring (computational thinking) simulations.

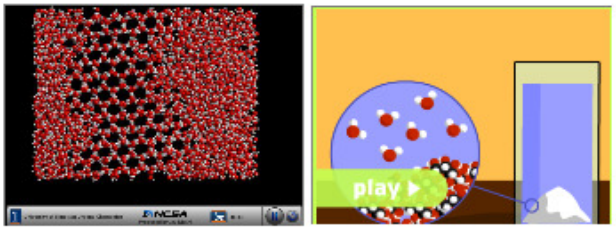

Figure 3: Two animation examples. The left image depicts an ICLCS simulation animation of ice melting and the right image depicts a BrainPOP animation on diffusion

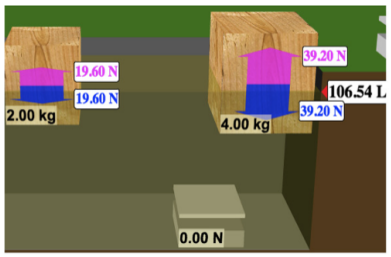

Figure 4: The image depicts PhET simulation that enables students to explore buoyancy

Interactive simulations allow users to alter parameters and run the simulation to see the effect of these modifications; unlike animations, students can run multiple experiments to better understand the effect of various parameter changes. Interactive simulations are usually centered around a specific concept, in a limited domain, and can take very little class time [16] . Examples of Interactive simulations are the Physics Education Technology (PhET)[3] Buoyancy simulation and the suite of simulations offered at National Taiwan Normal University's (NTNU) Virtual Physics Laboratory[4]. Figure 4 depicts a screenshot of the Phet simulation.

The NTNU wave simulation allows users to vary the direction and period of a plotted electromagnetic wave. The PhET buoyancy simulation in Figure 4, allows user to put two different blocks in water or oil and displays the buoyancy forces on the blocks. The blocks can be either the same mass, the same volume, or the same density.

PhET reports that students play with simulations for hours, find the activities engaging, and show a greater degree of conceptual learning as compared to control groups [19]. However, other research points to the idea that these simulations might give students an oversimplified view of scientific inquiry actually hindering their scientific literacy and ability to do science outside of the pre-made simulation environment; for example, many of these simulations assume ideal simulation environments with no experimentation flaws [7]. If students were to author these simulations, they could at least be aware of these ideal conditions.

Compared to animations, interactive simulations allow for a deeper inquiry of a given concept. However, students are exposed to little, if any, computational thinking concepts. In fact, much of the computational thinking, such as the abstraction necessary in creating this representational system and what parameters to include and ignore in this representational system, is done for the students by the simulation author.

2.3 Collective Simulations

Collective simulations (also called "participatory simulations") are similar to interactive simulations with the addition of a collaborative social element. In collective simulations, instead of controlling every parameter value, each student takes on a specific role and adjusts parameters associated with that role in real time. Often these simulations rely on students communicating with each other to develop

[3]phet.colorado.edu
[4]www.phy.ntnu.edu.tw/ntnujava/

an overarching strategy to achieve a specific goal. These interactions also begin to introduce students to the idea of local decisions yielding global consequences through peer-student feedback.

In Mr. Vetro, a National Institute of Health sponsored project that focused on how the body reacts to various activity, students played the role of lungs, heart, and dictated the amount of physical activity (exercise) applied to a virtual human named Mr. Vetro [11]. Each role has a set of parameters students can control. Students playing the role of the heart , for example, could define things like Mr. Vetro's heart rate and stroke volume. In Gridlock, written in HubNet, a collective simulation environment based off of NetLogo, students each control the state of a traffic light and and attempt to optimize traffic flow through a city [20]. Figure 5 depicts Mr. Vetro collective simulation.

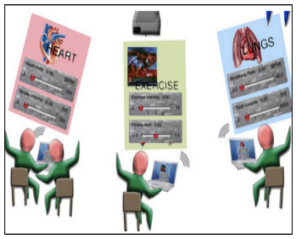

Figure 5: Students working on Mr. Vetro.

In terms of computational thinking, collective simulations enable students to embody the representation of the system. In this respect, students are the abstraction between the real world and the representational domain. Furthermore, students interact with classmates who represent other aspects of the real world. This necessitates peer organization and real-time integration of data among students to arrive at a collective algorithm through each moment in time. Like interactive simulations, students are varying parameters, but students are modifying these parameters constantly in response to peer feedback in order to achieve the collective group goals.

The results of Mr. Vetro show a strong indication of enhanced student understanding. For example, using Mr. Vetro for 1-2 periods in highschool classrooms indicated strong learning gains, as compared to control instruction covering the same material without the use of a collective simulation, over a diverse group of students [11]. Evidence also suggests strong student engagement during the activity with some students referring to it as one of the most intense classroom experiences [11].

The limitations of collective simulations mirror those of interactive simulations in that students can only change the parameters provided to them. Collective simulations begin to incorporate elements of computational thinking; however, students still do not get to set up the simulation or participate in simulation parameter selection deciding which aspects of the real world to emphasize and ignore.

2.4 Construction Set Simulations

Construction set simulations give users the freedom to set up the physical placement of pre-programmed simulation objects in addition to varying simulation parameters. Examples include construction kits, that enable users to solve a focused problem in a limited domain. For example, the Pinball Construction Kit allows users to prototype their own pinball game by placing flippers, bumpers, and other elements of a pinball game on a level [9]. Similarly, the PhET Circuit Construction Kit, depicted as an example of construction set simulations in Figure 2, allows users to place resistors, bulbs, and other electrical components in a circuit and run it [19]. The CyberMOD project, funded by the National Institute of Health, involves classroom integration of construction set simulations meant for the high school biology classroom. One CyberMOD construction set simulation allows students to investigate blood glucose homeostasis by placing liver cell agents and pancreas cell agents in different sections of the simulation world to achieve homeostatic regulation of blood sugar. In doing this, users not only see and interact with a simulation of how the human body works, but also, can experiment with different formations of cells investigating things such as type 2 diabetes. Figure 6 is a screenshot of the CyberMOD blood sugar regulation activity.

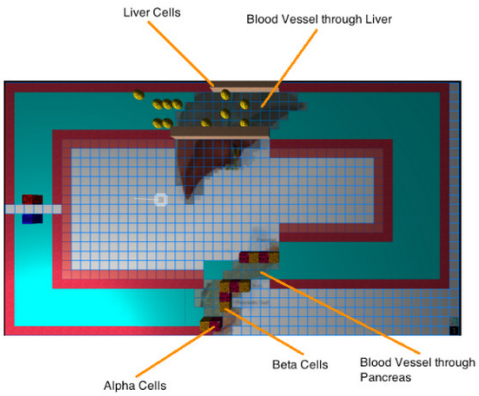

Figure 6: The Blood Sugar Homeostasis CyberMOD simulation.

Construction set simulations enable a measure of computational thinking in that students have to build the representational system world to solve or gain insight into a particular problem. Data from CyberMOD indicates students not only enjoyed the construction set simulation exercise, but after the exercise, the students were significantly more likely to answer questions related to the human blood sugar regulation system correctly. Furthermore, the effect sizes for 71 students between two pre and post test sections relating to blood sugar regulation, were 0.881 and 1.242, which is quite large for such a small (1 hour) in-class intervention [17].

Construction set simulations give students an opportunity to be exposed to computational thinking and have the potential to greatly increase understanding. The inability to program simulation behaviors present a big limitation. Investigating how different behaviors of agents effect the simulation, beyond parameter changes, necessitates a move towards simulation authoring.

2.5 Pattern Based Authoring

In pattern based authoring, users begin to author the behaviors of simulation actors present in the simulation. Instead of using low level behavior rules of end-user programming tools, pattern based authoring allows users to create

interactions between simulation agents at a higher level. Examples of pattern based authoring approaches include Magic Paper and the Simulation Creation Toolkit.

The Simulation Creation Toolkit, built on top of AgentCubes, allows users to apply high level Computational Thinking Patterns, such as one agent absorbing another agent or one agent tracking another agent, in order to create agent interactions [4, 5]. To accomplish this, users pick the interaction they want present in their simulation from a palette of generic objects acting out different behaviors– for example, one generic agent tracking another generic agent for the tracking pattern. Finally the user specifies how the pattern should work, i.e. how quickly the tracking should occur and the background agents that the tracking agent can move on etc. Figure 7 is a screenshot of the Simulation Creation Toolkit.

In Magic Paper, users can author physics simulations from scratch by drawing high level physical formations and defining things like where objects are anchored, the forces involved, and add wheels and friction to various objects [8]. The force that one object exerts on another object can be defined by outlining how one agent behaves in respect to another agent. Magic paper assumes sketching in a limited domain wherein a rich graphical lexicon has already been established– these symbols within the context of the sketching environment make up the "high level" patterns. Other domains in Magic Paper include a circuit construction environment and a molecular chemical configuration environment.

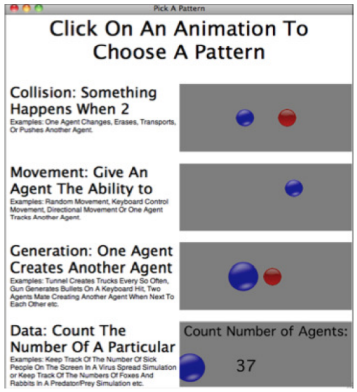

Figure 7: The Simulation Creation Toolkit

Both of these pattern based authoring examples rely on the user's ability to preserve interactions while abstracting out the specific agents involved in the interactions. In this sense, these approaches both force the user to think computationally to create these interactions. Research on the Simulation Creation Toolkit begins to show that this does indeed occur as users not only transfer the use of Computational Thinking Patterns to various contexts, but are also able to program simulations via analogy using these patterns [5, 4]. Pattern based authoring of simulations includes every mechanism outlined in Figure 1. Additionally, programming at the pattern level enables users to quickly create simulations without having to implement time-consuming lower level code. For example, seventh grade students with no prior experience using the Simulation Creation Toolkit, completed and experimented on a predator/prey simulation in four days or less of class time; this simulation, in previous experiences, would take high school students over a week to create including the time it takes to learn the AgentCubes end-user programming tool [6].

Pattern based authoring empowers users with the ability to quickly add interactions to simulations without using lower level code. However, a major limitation of pattern based authoring is that students are reliant upon the high level patterns offered; therefore, if a simulation creation exercise does not lend itself to one of the patterns offered, creating that simulation may be impossible or may lead students to make faulty representational systems [5]. Overcoming this limitation requires students to master lower level simulation programming.

2.6 End User Programming and Programming Languages

Tools such as AgentCubes, Scratch, and Alice allow students, with little prior programming experience, to create simulations using drag and drop rules instead of tedious syntax[12]. For example, in AgentCubes, agent behavior is dictated by dragging conditions and actions from a pre-made palette to construct if/then conditionality statements. This strategy has proven effective in enabling students to make visual games more quickly and easily than in traditional programming languages like C++ or Java.

In creating their first game students begin to learn the tools necessary to create simulations. Students actually implement Computational Thinking Patterns by combining if/then conditionality statements. Students begin to think computationally when they apply these learned Computational Thinking Pattern implementations to create and experiment on simulations giving them insight into a real world phenomenon [18]. In terms of Figure 1, end-user programming gives students much more expressive ability to model the real world than adding agent interactions at the pattern level.

The major barrier to integrating simulation end-user programming into the science classroom is finding the necessary time. As mentioned above, high school students can take over a week to learn an end-user programming tool and subsequently, create a simulation. Given enough time and resources, however, end-user programming of simulations allows students to accomplish everything outlined Figure 1.

2.7 Traditional Programming

Traditional Programming gives users the most expressive power out of all the choices in Figure 2. For example, since AgentCubes is agent-based, it is hard to create a physically accurate projectile motion simulation. However, in a traditional language like C++ or Java, such a simulation is possible, and in fact, has been done by authors of the PhET environment (which is written in Java)[19]. Other environments such as Greenfoot and Starlogo are educational tools that enable students to begin learning programming constructs and employ syntax akin to traditional programming as opposed to graphical drag and drop rules employed by end-user programming tools. Greenfoot, for example, gives students and introduction to Java programming and Starlogo, based on Logo programming, enables agent based simulation modeling [13, 12] .

Traditional programming languages have been used to create games and simulations in CS1 undergraduate classes [13]. Like end-user programming, students apply every step in

Figure 1. However, the time necessary to make students proficient at traditional programming is much greater than end-user programming; therefore, outside of actual programming classes, this strategy is unrealistic.

3. DISCUSSION AND CONCLUSION

Looking at the options outlined Figure 2, we see tradeoffs involved with every strategy presented that a teacher might employ to integrate simulation creation activities into their classroom environment. To the far left we see choices, such as pre-made simulations, that are convenient, in terms of preparation and class time, but do not necessarily require students to think computationally. Conversely, to the far right are choices, such as employing end-user and traditional programming to create simulations, that do require students to think computationally but are extremely time consuming. In between these two extremes are choices teachers can make that begin to introduce aspects of simulation authoring and computational thinking while preserving convenience. For example, teachers might substitute a construction set simulation or a collective simulation in place of a unit that currently integrates an interactive simulation without adding a significant amount of class time.

Figure 2 is an initial attempt to describe the space of simulation related activities that teachers can do in the classroom. As mentioned above, this figure is by no means exhaustive but is an interesting space to place any activities currently being researched and developed for in-class use. Using this spectrum, we can begin to populate this space with more activities and this can be an important tool helping teachers meet upcoming science standards while equipping students with the computational and problem solving skills necessary to be competitive in the 21st century.

4. ACKNOWLEDGMENTS

Part of this work is supported by the National Science Foundation under Grant Numbers 0833612, 0848962, 1138526 and the National Institute of Health under Grant Numbers 1R43 OD012081-01, 1R43 RR022008-0, and 1R43 RR022008-02 . Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or the National Institute of Health. Alexander Repenning is the founder of AgentSheets Inc.

5. REFERENCES

[1] *A Study Of The Effectiveness Of BrainPOP*. SEG Research, New Hope, PA, 2010.

[2] *Next Generation Science Standards*. Achieve, Inc. on behalf of the twenty-six states and partners that collaborated on the NGSS, Washington, DC, 2013.

[3] D. Barr, J. Harrison, and L. Conery. Computational thinking: A digital age skill for everyone. *Learning & Leading with Technology*, 38(6):20–23, 2011.

[4] A. Basawapatna, K. H. Koh, A. Repenning, D. C. Webb, and K. S. Marshall. Recognizing computational thinking patterns. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 245–250. ACM, 2011.

[5] A. R. Basawapatna. *Creating science simulations through Computational Thinking Patterns*. PhD thesis, University of Colorado, Boulder, 2012. 3549158.

[6] A. R. Basawapatna, A. Repenning, and C. H. Lewis. The simulation creation toolkit: an initial exploration into making programming accessible while preserving computational thinking. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 501–506. ACM, 2013.

[7] S. Chen. The view of scientific inquiry conveyed by simulation-based virtual laboratories. *Computers & education*, 55(3):1123–1130, 2010.

[8] R. Davis. Magic paper: Sketch-understanding research. *Computer*, 40(9):34–41, 2007.

[9] G. Fischer and A. C. Lemke. Construction kits and design environments: Steps toward human problem-domain communication. *HumanâĂŞComputer Interaction*, 3(3):179–222, 1987.

[10] L. Gray, N. Thomas, and L. Lewis. Teachers' use of educational technology in u.s. public schools: 2009. first look. nces 2010-040. *National Center for Education Statistics*, May 2010.

[11] A. Ioannidou, A. Repenning, D. Webb, D. Keyser, L. Luhn, and C. Daetwyler. Mr. vetro: A collective simulation for teaching health science. *International Journal of Computer-Supported Collaborative Learning*, 5(2):141–166, 2010.

[12] C. Kelleher and R. Pausch. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Comput. Surv.*, 37(2):83–137, June 2005.

[13] M. Kölling and P. Henriksen. Game programming in introductory courses with direct state manipulation. In *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, ITiCSE '05, pages 59–63, New York, NY, USA, 2005. ACM.

[14] D. R. Krathwohl. A revision of bloom's taxonomy: An overview. *Theory into practice*, 41(4):212–218, 2002.

[15] I. Lee, F. Martin, and J. e. a. Denner. Computational thinking for youth in practice. *ACM Inroads*, 2(1):32–37, February 2011.

[16] N. R. C. C. on Science Learning. *Learning Science Through Computer Games And Simulations*. The National Academies Press, Washington, DC, 2011.

[17] A. Repenning, A. Basawapatna, and M. Klymkowsky. Making educational games that work in the classroom: A new approach for integrating stem simulations. In *Games Innovation Conference (IGIC), 2013 IEEE International*, pages 228–235. IEEE, 2013.

[18] A. Repenning, D. Webb, and A. Ioannidou. Scalable game design and the development of a checklist for getting computational thinking into public schools. In *Proceedings of the 41st ACM technical symposium on Computer science education*, pages 265–269. ACM, 2010.

[19] C. E. Wieman, W. K. Adams, and K. K. Perkins. Phet: Simulations that enhance learnings. *Science*, 322(5902):682–683, November 2008.

[20] U. Wilensky and W. Stroup. Networked gridlock: Students enacting complex dynamic phenomena with the hubnet architecture. In *Fourth international conference of the learning Sciences*, pages 282–289, 2000.