Scalable Game Design and the Development of a Checklist for Getting Computational Thinking into Public Schools

Alexander Repenning
University of Colorado
Computer Science Department
Boulder 80309-430
+1 (303) 492-1349
ralex@cs.colorado.edu

David Webb
University of Colorado
School of Education
Boulder 80309
+1 (303) 492-0306
dcwebb@colorado.edu

Andri Ioannidou
AgentSheets Inc.
6560 Gunpark Drive
Boulder, CO, 80301, USA
+1 (303) 530-1773
andri@agentsheets.com

ABSTRACT
Game design appears to be a promising approach to interest K-12 students in Computer Science. Unfortunately, balancing motivational and educational concerns is truly challenging. Over a number of years, we have explored how to achieve a functional balance by creating a curriculum that combines increasingly complex game designs, computational thinking patterns and authoring tools. Scalable Game Design is a research project exploring new strategies of how to scale up from after school and summer programs into required curriculum of public schools through game design approaches. The project includes inner city schools, remote rural areas and Native American communities. A requirement checklist of computational thinking tools regarding curriculum, teacher training, standards and authoring tools has been developed and is being tested with thousands of students.

Categories and Subject Descriptors
K.3.2 Computer and Information Science Education

General Terms
Design, Human Factors, Languages

Keywords
Game design, computational thinking, computational science.

1. INTRODUCTION: THE SCALABLE GAME DESIGN INITIATIVE
Scalable Game Design is an initiative with the goal to expand opportunities to motivate, engage, and educate students about Computer Science through game design, starting at the middle school level. For over 15 years, funded mainly by the National Science Foundation (NSF), we have carried out investigations on new approaches to programming resulting in game and simulation authoring systems such as AgentSheets [1, 2] and AgentCubes [3]. While the goal has largely remained the same, the degree of ambition has steadily increased, in that we have gradually moved away from communities of self-selected users towards what is perhaps the most challenging educational context: implementing new IT curricular as part of the regular public school program.

The main goal of our latest NSF-funded ITEST project called "Reforming IT Education through Game Design: Integrating Technology-Hub, Inner City, Rural and Remote Regions" (iDREAMS[1] for short) is to bring Computer Science to middle schools with the ultimate aim of developing a larger IT workforce. Numerous problems with existing high school advanced placement courses have been discussed [4], but Computer Science education at the middle school level has received comparably little attention. As a result, programming has almost completely disappeared from the middle school curriculum. Existing IT opportunities at the middle school level often include little more than keyboarding, web browsing, and use of application training.

The rapidly increasing number of summer camps, after-schools programs, female and minority focused special programs, and computer clubs at the middle school level strongly suggests that there is a demand from students for such opportunities. Many, including numerous ITEST projects [5] and our own projects [1], have invested a great deal of time and effort in broadening participation through motivational extracurricular activities. We believe this is an important first step. However, we also believe that now is the right time to bring Computer Science into middle school curricula to develop a stronger and bigger IT workforce. One reason to do this is that motivational concerns need to be addressed at the middle school level, using early IT experiences to support future career choices. The middle school years are critical for students in reaching conclusions regarding their own skills and aptitudes [6, 7]. This is the age at which children prematurely and often falsely conclude that math and science is not for them, or that Computer Science is all about programming or is a field that is hard as well as boring. Another reason to do this is to expose all students, including minorities and women, to Computer Science at a level of participation that no combination of extracurricular programs could achieve. After all, one should not forget that participation in extracurricular programs is based on self-selection and typically involves additional fees. Students showing up at these events usually are already excited about information technology. What about the large majority of students who are skeptical towards IT or may not have the resources to participate? While many extracurricular programs have been successful, they only reach out to a small subset of children compared to the number of students enrolled in middle schools.

Skepticism towards programming in schools [8] is entirely justified. A student running towards us when we entered a school with an early prototype of the LEGOsheets [9] perhaps best summarized the programming in school situation in 1996. Excited to see the LEGO blocks, he asked us what we would be doing. We explained we would do programming. "Programming, oh no!" he replied, "I know what is going to happen. The teacher writes a program onto the blackboard, we type it into the computer and it

SIGCSE'xxx, xxx, xxxx, USA.

[1] http://scalablegamedesign.cs.colorado.edu

never works." This points out that programming, as an educational activity, must be heavily scaffolded, but also grounded in students' interests, insights, and creativity. Ultimately, programming in schools is not just about picking the right software, but about a process reconceptualizing what the right skills to teach are and what kinds of pedagogical and motivational models need to be employed to make Computer Science a feasible and integral part of K-12 education.

Given the less than ideal track-record of programming in schools in general and specifically in middle schools [10], the question arises: why should we bring programming to middle schools in a systematic way? And by systematic, we mean initiatives involving entire school districts, as opposed to grass root efforts of individual teachers. We believe that the field of Computer Science education may approach a critical tipping point [11]. Results of the 2009 CSTA National secondary CS survey indicate that in only two years high schools offering courses featuring game design have increased from 0.6% in 2007 to 10% in 2009. Many interesting strategies, tools and curricula have been explored in isolation. It is time to investigate how to integrate some of these results in a way that would make them sustainable for public schools at a large scale. In particular, the notion of Computational Thinking [12-14] has refueled research in IT education by re-examining the core values of Computer Science education.

Our iDREAMS project is specifically exploring a number of pragmatic dimensions related to computational thinking of how to bring Computer Science education to public schools. The project started in early 2009 with the goal to provide game design and programming experiences to over 2000 students over three years. Specifically, the project engages a vertical segment of diverse inner city, remote rural and Native American communities from South Dakota to southern Colorado including some of Colorado's poorest rural school districts. A major research question for this project is whether it is possible to introduce computational thinking at the middle school level through game design to diverse communities of non self-selected teachers and students.

While, conceptually speaking, computational thinking is at the core of this project we are less interested in creating a new definition of what computational thinking is (or is not), and are mostly concerned with the pragmatics of computational thinking. How can we use tools, train teachers, scaffold game design education, support teachers in the classroom, and motivate the general student, teacher, parent, and school administrator populations? If we want to advance the notion of computational thinking beyond self-selected groups of teachers and students, what kind of conceptual computational thinking tools do we need?

This paper describes what we call the computational thinking tool checklist. This is an early and evolving version of suggested requirements that conceptual tools should satisfy to facilitate computational thinking in public schools. A discussion section briefly talks about experiences with the iDREAMS project so far.

2. COMPUTATIONAL THINKING TOOLS CHECKLIST

The version of the computational thinking tools checklist presented here is the result of building and using computational tools, e.g., AgentSheets, for many years for game design and computational science [15] applications. Our latest initiative, Scalable Game Design, enhances K-12 education by creating game design based curricula and teacher training aligned with computational literacy frameworks and standards [16, 17]. *Scalable* refers to the scope of applications starting with simple game design in middle schools and advancing along a gentle learning slope [18, 19] all the way to graduate school. At the middle school level, Scalable Game Design consists of two modules. In 6th grade a one-week module is integrated into an existing required course. In 7th grade a four-week module in elective courses allows students to move on to more complex games or computational science simulations.

We have started to use and evolve the notion of *computational thinking tools* as a combination of curriculum based on a computational thinking pattern inventory, authoring tools, and teacher training. We claim that for systemic impact, a computational thinking tool used in K-12 must fulfill *all* these conditions:

1) ***has low threshold***: a student can produce a working game quickly.
2) ***has high ceiling***: a student can make a real game that is playable and exhibits sophisticated behavior, e.g., complex AI.
3) ***scaffolds Flow***: the curriculum provides stepping stones with managed skills and challenges to accompany the tool.
4) ***enables transfer***: tool + curriculum must work for both game design *and* subsequent computational science applications as well as support transfer between them.
5) ***supports equity***: game design activities should be accessible and motivational across gender and ethnicity boundaries.
6) ***systemic and sustainable***: the combination of the tool and curriculum can be used by *all* teachers to teach *all* students (e.g. support teacher training, standards alignment etc).

The following sections describe these requirements in detail.

2.1 Low Threshold

An ideal strategy to include Computer Science in a way that will be inclusive to women and minorities may be to make it part of existing required courses (e.g., computer power or exploratory wheel courses[2]). In this context it is typically feasible to squeeze in a one-week (5x45 minutes) module. In that time it must be possible for students to make one complete game such as Frogger. If even a simple game is hard to build and game design activities lead to frustration, then little progress towards computational thinking will be achieved. With AgentSheets, many students finish a simple Frogger-like game (cursor controlled frog, moving cars, some kind of collision handling between frogs and trucks) in the first three sessions, and additional game creation activity follows.

To make this possible, one may have to differentiate between a programming tool and a computational thinking tool. As pointed out by Wing [12], computational thinking should not be considered a synonym for programming. Given the experience of many teachers (who have never made a game, never programmed, and in many cases, not even played a game), it is essential that computational thinking offers a simple mapping between problem and solution. For example, if the task is to simply program the frog in Frogger, a user-controlled object trying to cross a busy highway, then we would expect a relatively simple solution.

[2] Exploratory Wheels are courses that cover a variety of topics so that students can get a taste of different technical domains and decide if they are interested in pursuing the topic in more detail. Typically the topics covered in the exploratory wheel are offered as subsequent elective courses.

The true challenge for a low threshold is not a question of whether there is some kind of drag and drop programming, but whether the resulting program includes excessive need to *code*, rather than to *represent* the problem description. In comparing implementations of a cursor-controlled character in AgentSheets and Scratch (in Figure 1 and Figure 2 respectively), both systems feature a drag and drop programming style, but in the Scratch solution, the use of doubly nested loops and "magic" constants (e.g., where is the value of -162 coming from?) cannot be conceptually traced back to the original problem description. In other words, in one case we have a program that is closer to a computational thinking level whereas in the other case it is at a much lower code level.

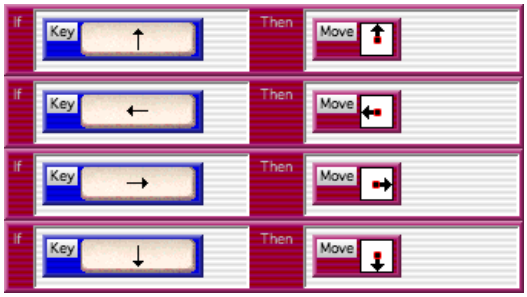

Figure 1. Programming at Computational Thinking level: Program to make a cursor-controlled frog

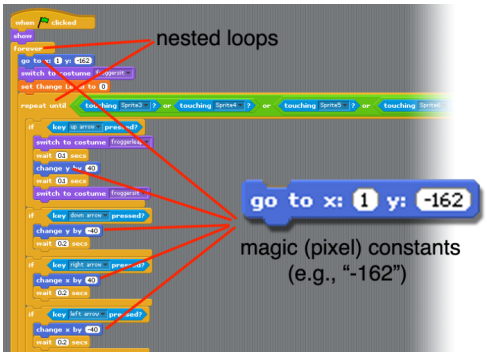

Figure 2. Programming at Code level: program contains many elements that cannot be traced back to problem.

The main point of low threshold is not to compare programming languages, but to illustrate that the notion of thresholds may mean vastly different things to different people. A computational thinking tool must include a design scaffold for teachers and students to transparently map a problem description into solution. Pragmatically speaking, the most important aspect of a low threshold tool is not if – in theory – a programming language may allow a simple solution but whether or not teachers with little or no programming background can be systematically trained to teach their students to find solutions to computational thinking challenges.

2.2 High Ceiling

If the students cannot make interesting, playable games, then their initial excitement quickly gives way to disappointment. Students need ways to create games with complex behavior using sophisticated math and Artificial Intelligence. How can my characters find the shortest path in a maze? How can I make them collaborate and compete? This type of sophistication may seem out of the reach of middle school students, but we have found ways to scaffold game design, including 3D visualizations (Figure 3), with computational thinking patterns to the point where middle school students can build games that not too long ago would have been challenging for Computer Science university students.

Collaborative Diffusion [20] is a collaborative agent programming approach based on diffusion equations initially used in graduate and undergraduate Computer Science courses on educational game design [21]. This approach can be used to make highly sophisticated games with Sims-like behaviors. The need to deal with advanced math concepts, i.e., the need to program, tweak and debug diffusion equations, did not dissuade middle school students [22]. On the contrary, students in many cases found math, for the first time, to be useful because math became a tool that allowed them to build their video game. Of course, not all students progress to this point at the middle school level. However, we believe it is essential not to trap students into toy-like programming languages that may provide a short burst of enthusiasm, but ultimately fail to help them progress from motivational game design to educational STEM applications.

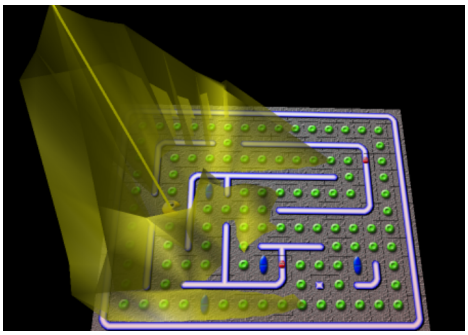

Figure 3: The use of visualization can explain complex concepts such as diffusion and how they can be used for Artificial Intelligence applications.

2.3 Scaffolds Flow

Low threshold and high ceiling are important but what is the process to effectively progress from basic to sophisticated game design? Working with teachers and students worldwide, we have analyzed the kinds of games they have built in terms of challenges and skills. Optimal flow [24] in game design requires balancing design challenges and developing skills by scaffolding the process with well-defined stepping stones based on increasingly complex computational thinking patterns, e.g.:

- **Collision**; in Frogger: frog meets truck
- **Push**; in Sokoban: person pushes boxes
- **Transport**: in Frogger: logs and turtles transport frogs
- **Generate**: in Space Invaders: defenders shoot rockets
- **Absorb**: in Frogger: tunnel absorbs cars
- **Choreography**: in Space Invaders: mothership coordinates alien ships movement and descent
- **Polling / Counting**: in Pac-Man: game ends when all the dots are eaten
- **Diffusion**: electricity, heat, rumors, toys: spread of information
- **Path Finding**: in The Sims: people finding food
- **Collaborative Diffusion**: in a soccer game: players collaborate and compete
- **Hierarchy of Needs**: Maslow's model of human motivation.

These computational thinking patterns are language as well as application independent. For instance, once a student understands how to conceptually represent a collision in one programming language, e.g. Java, then the student is more likely to be able to create a corresponding solution in a different language.

The Scalable Game Design curriculum is based on a number of increasingly demanding game designs, for instance, moving from a game like Frogger, to Pac-Man, SimCity, and all the way to The Sims. Each design, in addition to tutorials and sample solutions, offers links to computational thinking patterns[3]. The curriculum covers an extended duration of the Computer Science education pipeline ranging from middle school to graduate school, but does not prevent advanced students from moving ahead. Indeed, many of the advanced middle school students build sophisticated games compared in complexity to ones found typically at the undergraduate level.

2.4 Enables Transfer

"Now that you can make Space Invaders can you build a science simulation?" teachers ask their students. Perhaps, this question really gets to the core of computational thinking. While the jury is still out on defining what computational thinking really is, this kind of pragmatic interpretation provided by teachers essentially provides a litmus test for what computational thinking should be able to achieve. Educators believe it should be able to achieve transfer. How can game design skills transfer to model building, which is part of computational science and STEM education? Many educators are willing to explore the idea of game design for its motivational benefits. If, however, students can only make a game using a particular software tool, then ultimately game design will not be accepted at a large scale in K-12. One could argue that if there is no transfer to STEM there is no computational thinking.

Of course, we know that transfer does not just happen [25]. What does random movement in a game have to do with Brownian motion in a computational science model? These connections need to be established explicitly by teachers and integrated into a set of interconnected computational thinking courses including, for instance, game design, computational science, and robotics. We have been using AgentSheets extensively to teach students game design and computational science but have not yet systematically explored mechanisms of transfer. We have started to develop a higher-level computational thinking pattern inventory that is explicitly connects these patterns to different applications such as game design and computational science (Figure 4).

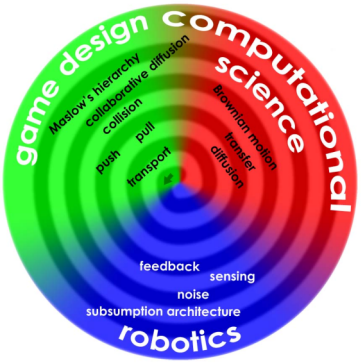

Figure 4. Computational Thinking Inventory: an inside out gradual and iterative exploration of transferable computational thinking patterns.

At a technical level computational thinking tools would also have to include certain affordances to be truly useful. According to the President's Information Technology Advisory Committee, computational science has tremendous potential for STEM education [15]. However, even the most basic computational science applications require tools for numerical analysis including the ability to define sophisticated mathematical expressions, the ability to collect and export data, and support for visualizing data.

2.5 Supports Equity

Tools have to be effective in both motivating and educating students across ethnicity and gender in a variety of educational settings, including elective classes or programs, and required courses within the curriculum. Formal studies (e.g. an independent research study by the Stanford School of Education [26]), concluded that both boys and girls express the same high levels of desire to continue with game design using AgentSheets. In our local school district (which is the first district in Colorado to bring programming to its middle schools by using an early version of our Scalable Game Design curriculum and AgentSheets in all its middle schools), teachers already report that, after students complete AgentSheets units in their Exploratory Wheel courses, both boys and girls are motivated by their experiences and so energized that they go to the counseling office to put computers as their first elective choice. They also report that participation of girls in elective courses significantly increases. As one teacher reported, "I used to only have 2 or 3 girls in my elective classes, now half of the class is girls." In iDREAMS schools, the participation of women is close to 50% because in many of these schools these courses are required.

2.6 Systemic and Sustainable

For computational thinking tools to be successfully integrated into K-12 education, they need to be systemically adopted by schools and districts. We have developed teacher training and curricula aligned with ISTE NETS standards [17] and have integrated Scalable Game Design into the middle school computer education curriculum of entire school districts. The Scalable Game Design wiki pages include specific links from each game design activity to ISTE standards covered. The game design activities with their intrinsic need to engage students in problem solving including accessing, compiling and integrating information, are also consistent with learning outcomes suggested by the K-12 Computer Science model curriculum[4] recommend by the ACM.

Integration with standards is essential, especially when trying to reach a tipping point for a Computer Science education strategy that is more systemic and sustainable. When shifting towards implementation models that move away from self-selected teachers and students, participation can reach critical levels. The iDREAMS project takes place in 16 schools. Some schools have multiple IT teachers with some schools teaching Computer Science to an estimated 600 students per semester.

3. DISCUSSION

Currently there are 19 middle school teachers participating in the iDREAMS project. Over the course of the 2009-10 school year, based on responses received from teachers, there will be approximately 75 cycles of the Frogger unit taught to over 2,000 students. Twelve community college students are also serving as classroom support liaisons in select classrooms.

Prior to the 2009-10 school year, community college students completed one week of training that included an opportunity to design five games from Frogger to The Sims, design a science

[3] http://scalablegamedesign.cs.colorado.edu/wiki/Frogger_Design

[4] http://csta.acm.org/Curriculum/sub/CurrFiles/K-12ModelCurr2ndEd.pdf

simulation of an ecosystem, and observe a summer session game design class. During the second week of the institute, the community college students were joined by participating teachers to design similar games and further explore the methods and activities proposed for middle school students to construct games.

Teachers who have already started to implement the Frogger unit in their classes have been completing daily lesson logs to document their observations of students, monitor the pacing and activities completed, and indicate how they have adapted the proposed unit to address perceived student needs. Even though we are at the early stages of implementation and data collection, four teachers have already reported that the lessons went exceptionally well, with unusually high engagement: students who are usually not engaged, are showing strong interest. Students also seemed to comprehend ideas that had previously been troublesome.

The computational thinking tool checklist presented here is an early framework of evolving recommendations for introducing computer science into the regular school program through game design. We invite interested parties to participate, challenge and refine this framework through the Scalable Game Design wiki.

4. ACKNOWLEDGEMENTS

This material is based in part upon work supported by the National Science Foundation under grant numbers 0833612 and DMI-0712571. 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.

5. REFERENCES

[1] A. Repenning and A. Ioannidou, "Broadening Participation through Scalable Game Design," in Proceedings of the ACM Special Interest Group on Computer Science Education Conference, (SIGCSE 2008), Portland, Oregon USA, 2008, pp. 305-309.

[2] A. Repenning and A. Ioannidou, "Agent-Based End-User Development," Communications of the ACM, 47(9), pp. 43-46, 2004.

[3] A. Repenning and A. Ioannidou, "AgentCubes: Raising the Ceiling of End-User Development in Education through Incremental 3D," in IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'06), Brighton, United Kingdom, 2006, pp. 27- 34.

[4] Board of Directors, Computer Science Teachers Association, "Achieving Change: The CSTA Strategic Plan," 2005.

[5] ITEST Learning Resource Center, "ITEST Project Abstracts: Cohorts 1, 2, 3, 4 & 5," 2008.

[6] E. Gootman, "The Critical Years: For Teachers, Middle School Is Test of Wills," New York Times, March 17, 2007.

[7] S. J. Sears, "Career and Educational Planning in the Middle Level School," NASSP Bulletin, April 1995, 1995.

[8] T. Oppenheimer, The Flickering Mind: The False Promise of Technology in the Classroom and How Learning Can Be Saved. Toronto, Canada: Random House, 2003.

[9] J. Gindling, A. Ioannidou, J. Loh, O. Lokkebo, and A. Repenning, "LEGOsheets: A Rule-Based Programming, Simulation and Manipulation Environment for the LEGO Programmable Brick," in Proceeding of Visual Languages, Darmstadt, Germany, 1995, pp. 172-179.

[10] R. S. Cohen, "Logo in the Primary Classroom: Should Simplified Versions Be Used?," The Computer Teacher, pp. 41-43, 1990.

[11] M. Gladwell, The Tipping Point: How Little Things Can Make a Big Difference: Back Bay Books, 2002.

[12] J. M. Wing, "Computational Thinking," Communications of the ACM, 49(3), pp. 33-35, March 2006.

[13] S. Papert, "An Exploration in the Space of Mathematics Education," International Journal of Computers for Mathematical Learning, 1(1), 1996.

[14] G. H. Fletcher, and J. J. Lu, "Education, Human computing skills: rethinking the K-12 experience," Communications of the. ACM, 52(2), pp. 23-25, 2009.

[15] President's Information Technology Advisory Committee (PITAC), "Report to the President: Computational Science: Ensuring America's Competitiveness," June 2005.

[16] Committee on Information Technology Literacy, National Research Council, Being Fluent with Information Technology. Washington, D.C.: National Academy Press, 1999.

[17] International Society for Technology in Education (ISTE), National Educational Technology Standards for Students (NETS), 2nd ed., 2007.

[18] M. Dertouzos, "Creating the People's Computer," MIT Technology Review, Cambridge, MA, 100(3), pp. 20-28, 1997.

[19] A. I. Mørch, "Three Levels of End-User Tailoring: Customization, Integration, and Extension," in Computers and Design in Context, M. Kyng and L. Mathiassen, Eds. Cambridge, MA: The MIT Press, 1997, pp. 51-76.

[20] A. Repenning, "Collaborative Diffusion: Programming Antiobjects," in OOPSLA 2006, ACM SIGPLAN International Conference on Object-Oriented Programming Systems, Languages, and Applications, Portland, Oregon, 2006, pp. 574-585.

[21] C. Lewis and A. Repenning, "Creating Educational Gamelets," in Educating Learning Technology Designers: Guiding and Inspiring Creators of Innovative Educational Tools, C. DiGiano, S. Goldman, and M. Chorost, Eds. New York: Routledge, 2008, pp. 203-229.

[22] A. Repenning, "Excuse me, I need better AI!: employing collaborative diffusion to make game AI child's play," in ACM SIGGRAPH symposium on Videogames, Boston, Massachusetts, 2006, pp. 169-178.

[23] K. Schneider and A. Repenning, "Deceived by Ease of Use: Using Paradigmatic Applications to Build Visual Design," in Proceedings of the 1995 Symposium on Designing Interactive Systems, Ann Arbor, MI, 1995, pp. 177-188.

[24] M. Csikszentmihalyi, Flow: The Psychology of Optimal Experience. New York: Harper Collins Publishers, 1990.

[25] S. M. Barnett and S. J. Ceci, "When and where do we apply what we learn? A taxonomy for far transfer," Psychological bulletin, 128(4), pp. 612-37, 2002.

[26] S. Walter, B. Barron, K. Forssell, and C. Martin, "Continuing Motivation for Game Design," in CHI 2007, San Jose, California, USA, 2007, pp. 2735-2740.