Scalable Game Design: Broadening Participation by Integrating Game Design and Science Simulation Building into Middle School Curricula

Introduction

At the University of Colorado we have worked on visual programming languages used in computer science education for over 20 years. In this paper we lay out our strategy of our Scalable Game Design curriculum, which has been funded through a series of NSF (ITEST Strategy, CE21 Type II, and ITEST Scale Up) grants as well as the Google CS4HS program, and list some research questions relevant to bringing Computer Science education to middle schools.

Background: The Scalable Game Design Strategy

Educational programming environments have tried to broaden the participation of women and minorities in computer science education by making programming more exciting and accessible. Starting in 1991 our research at the University of Colorado with AgentSheets explored the idea of supporting kids with game design and simulation building through drag and drop interfaces. Early on AgentSheets enabled kids to build and share their creations through the web and more recently AgentCubes is exploring the idea of 3D fluency through gentle slope 3D. In 2008 we started the Scalable Game Design project with the goal to develop a Computer Science curriculum broadening participation of students particularly at middle schools. While the broadening participation situation at the high school level looks still bleak it is much worse at the middle school level. To the degree that programming is found at middle schools at all, it is usually offered as after school programs. Middle school is an essential period of life during which students, especially women and minority students, make decisive but often unfortunate career decisions such as "science is not for me." How can we shift middle school computer science education from isolated after school efforts to a systemic model in which computer science is integrated into the school curriculum and taught in required classes at district wide levels?

The Scalable Game Design (SGD) initiative, started as NSF ITEST project in 2008, had the ambitious goal to revolutionize computer science education in public schools by introducing students to computer science through a combination of game design and STEM simulation creation integrated into the middle school curriculum. A number of early and highly successful SGD pilot programs were followed by the USA's largest middle school computer science education study. To date the SGD project has over 10,000 subjects from inner city, remote rural and Native American schools participated in designing games and creating STEM simulations. SGD researchers at the University of Colorado systematically developed and evaluated a sustainable education strategy based on four core principles:

- ***Exposure***: Broaden participation and reach every student by injecting an easy-to-teach one week game design module into currently mandatory keyboarding and PowerPoint classes. Forty-five percent of the students in the SGD project are female.
- ***Motivation***: Motivate students by carefully balancing challenges and skill levels through game design activities in a SGD curriculum that ranges from simple Frogger-like games to advanced Sims-like games. SGD student motivation is extremely high: 74% of male participants and 64% of female participants want to continue with similar courses as electives.
- ***Education***: Build instruments that analyze student projects for STEM-critical skill acquisition so that learning outcomes can be objectively measured. Over 10,000 student games and simulations were analyzed using a latent semantic analysis inspired approach for evidence of computational thinking and skill transfer between game design and simulation creation.
- ***Pedagogy***: Investigate the interaction of pedagogical approaches and motivational levels across genders and ethnicities. We found that choosing the correct approach was essential to broadening participation. With an optimal pedagogical approach thirty-five hours of careful instruction were enough to train teachers to teach SGD curriculum in a gender friendly way.

Project data suggest that the SGD strategy works extremely well: 74 percent of male participants and 64 percent of female participants wanted to continue with similar courses as electives.

Philosophy

Our philosophy stems from a project-first, as opposed to a principles-first, approach. Whereas a principles-first approach focuses on conveying theoretical aspects of a subject and learning necessary skills before students have opportunities to apply them, a project-first approach allows students to immediately engage in computer programming design experiences and to learn concepts as the need arises.

Early on it became clear to us that students did not sign up for computer science courses because of their negative perceptions of the subject. Asked in the context of a typical computing course, one middle-school student summarized her perception of programming as "hard and boring," which does not suggest a workable tradeoff but instead a heartbreaking lose-lose proposition. The "hard" part is a cognitive challenge that we have addressed with new visual programming approaches such as the drag and drop programming in AgentSheets. We have also advanced beyond just syntactic programming support to semantic programming support through the use of innovative debugging tools in more recent versions of AgentSheets and AgentCubes. The "boring" part is an affective challenge that relates back to motivation. Why should students really want to program? In our research we found ownership to be the key to motivation. When students are enabled to create artifacts that are personally meaningful to them, they are much more likely to be motivated. For instance, with AgentCubes, students create 3D shapes such as people, animals, and cars. Once they have created these shapes and assembled them into a world they are highly motivated to bring them to life through the process of programming.

The project-first approach affects motivation to the point where, at the middle school level, with students as young as 11 years old, students essentially demand access to advanced programming concepts to build their games. For instance, students want to build artificial intelligence that allows the characters of their game to collaborate with each other and to engage in tracking, even if this means that they will have to master sophisticated mathematics concepts such as diffusion equations. A principles-first approach would not likely work for these students. Advanced concepts such as diffusion equations are not considered suitable material for middle school mathematics education. And yet, these very students demand to learn these advanced topics because they are solving a problem relevant to them.

In order to work well the project-first philosophy requires appropriate scaffolding, and can be best described in the context of our theoretical framework called the Zones of Proximal Flow. The Zones of Proximal Flow framework is a combination of Csíkszentmihályi's Flow theory with Vygotsky's Zone of Proximal Development conceptualization. The essence of Scalable Game Design is that programming challenges and skills should be balanced and that there are different paths, some better suited than others, for students to acquire new skills and tackle more advanced challenges.

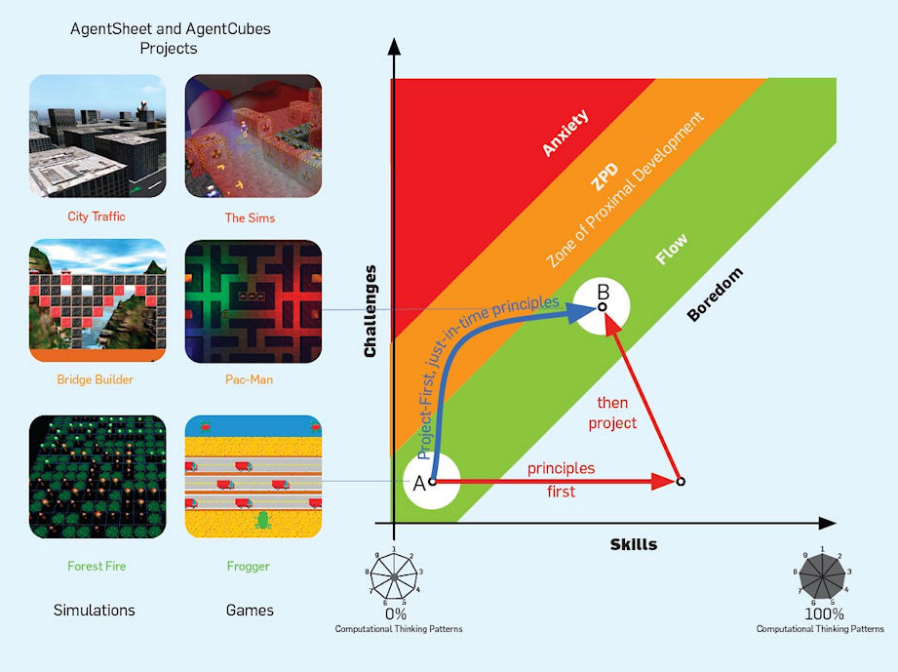

Figure 1: The Zones of Proximal Flow, a combination of Vygotzky's Zone of Proximal Development and Csikszentmihalyi's Flow.

The Zones of Proximal Flow framework (Figure 1) illustrates the difference between the principles-first and the project-first philosophies. Imagine that a student has just created a simple game representing a combination of limited skills and minimal challenge (point A). This student wants to progress to point B by creating a more sophisticated game involving greater challenge and requiring additional skills. The traditional, principles-first route would suggest the abstract acquisition of facts and concepts without any concrete application of these principles. This path is likely to navigate the students into the "boredom" zone. Only later, perhaps many semesters later, will that same student finally have an opportunity to

apply these skills in a meaningful project. By then, many of the skills may have been forgotten. The "project-first, principles just-in-time" path, in contrast, immediately engages the student in project work. The project will be challenging and is likely to push students to their threshold of understanding (the Zone of Proximal Development) but with the help of the teacher (and peers) they manage to learn the relevant concepts in an optimal way that is highly engaging.

Computing Computational Thinking

For computing Computational Thinking and automatically measuring student learning outcomes in the context of the Scalable Game Design project, we built a cyberlearning infrastructure called the Scalable Game Design Arcade to collect and analyze games and simulations created by students using AgentSheets. The submitted game is analyzed in terms of the Computational Thinking patterns it uses and results are displayed on a graph (Figure 2, right, orange spider graph). If there is a tutorial for a submitted game/simulation, then the CTPA graph also depicts the computational thinking patterns of the tutorial (Fig 2, right, green spider graph). The relationship between the two graphs is an indication of how close the submitted game is to the tutorial implementation of the game, if a tutorial is available.

CTPA compares a given game/simulation with nine pre-defined canonical computational thinking patterns: cursor control, generation, absorption, collision, transportation, push, pull, diffusion, and hill climbing. CTPA compares the given game/simulation with each canonical Computational Thinking pattern (Figure 2, left) to produce the corresponding values in the CTPA graph (Figure 2, right). The calculated value represents how each Computational Thinking pattern is similar to a given game/simulation. Higher value means higher similarity. If a specific Computational Thinking pattern is used frequently in the implementation of the game/simulation, then the similarity between that pattern and the game/simulation is higher.

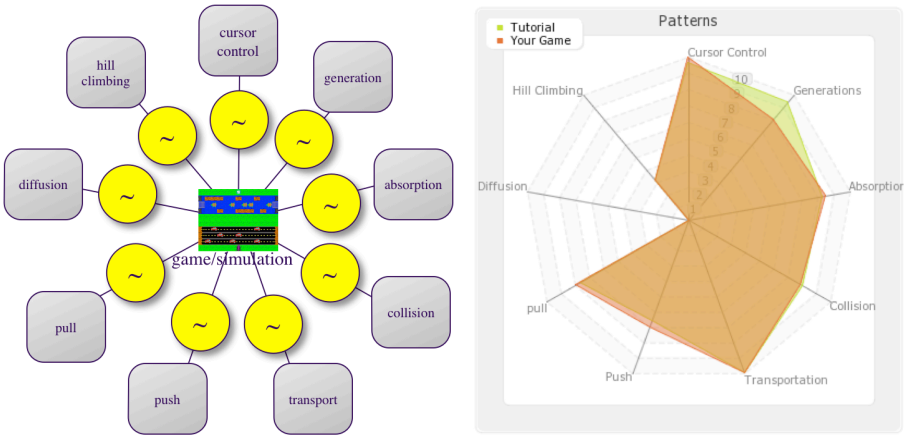

Figure 2: Using LSA-inspired similarity (denoted as on the left), a game submitted to the Scalable Game Design Arcade gets compared to nine canonical Computational Thinking Patterns. A CTPA graph showing the similarity values for each pattern is produced (right).

Research Questions

1. ***How can we shift the pedagogy of Computer Science education to broaden the participation of women?*** Our data from over 10,000 subjects in schools all around the USA suggests that pedagogy is one of the key factors at the middle school level to get women interested in CS education. We cannot simply tell teachers to use certain pedagogies. How does one integrate pedagogical approaches into existing teacher profession development?

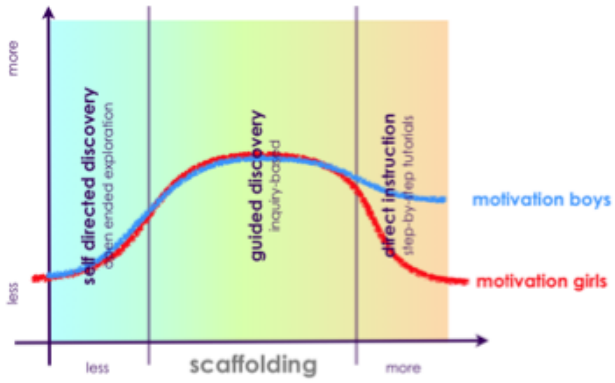

Figure 3: Scaffolding vs. Motivation

2. ***How can we prevent MOOCs to be negative for broadening participation?*** Our framework (Figure above) suggests that pedagogy based on direct instructions has negative consequences for the motivational levels in women in computer science education. Early experience with CS related MOOCs seem to confirm this potential issue by reporting even lower that average participation of women for introductory level CS courses.

3. ***How can computational thinking be integrated into STEM courses?*** New curricula such as the Next Generation Science Standards are explicitly mentioning computational thinking including the creation of simulations by students. How feasible is this given that neither students nor teachers are prepared for these kinds of activities?

4. ***What are promising computational ecosystems to allowing computational thinking to be successfully integrated into schools?*** In isolation the introduction of a new computer science classes and the use of computational skills in STEM courses, particularly at the middle school level, seems somewhat unlikely.

5. ***What could be done to increase the quality of CS education research?*** It is still too common for studies to be of relatively low quality (especially compared to similar research in other disciplines such as mathematics education) to be published in CS education conferences. The number of subjects is often small; in some cases there are more paper authors than there are research subjects. Other problems include recruiting approaches that are not clear. For instance, authors sometimes do not seem to appreciate the enormous differences between self-selected and non-self selected students. This can dramatically reduce the applicability of finding to a more general context.