Using Scalable Game Design to Teach Computer Science From Middle School to Graduate School

Ashok R. Basawapatna
University of Colorado Boulder
Department of Computer Science
Boulder, CO. 80303
(720) 838-5838, 001
basawapa@colorado.edu

Kyu Han Koh
University of Colorado Boulder
Department of Computer Science
Boulder, CO. 80303
(303) 495-0357, 001
kohkh@colorado.edu

Alexander Repenning
University of Colorado Boulder
Department of Computer Science
Boulder, CO. 80303
(303) 492-1349, 001
ralex@cs.colorado.edu

ABSTRACT
A variety of approaches exist to teach computer science concepts to students from K-12 to graduate school. One such approach involves using the mass appeal of game design and creation to introduce students to programming and computational thinking. Specifically, Scalable Game Design enables students with varying levels of expertise to learn important concepts relative to their experience. This paper presents our observations using Scalable Game Design over multiple years to teach middle school students, college level students, graduate students, and even middle school teachers fundamental to complex computer science and education concepts. Results indicate that Scalable Game Design appeals broadly to students, regardless of background, and is a powerful teaching tool in getting students of all ages exposed and interested in computer science. Furthermore, it is observed that many student projects exhibit *transfer* enabling their games to explain complex ideas, from all disciplines, to the general public.

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

General Terms
Algorithms, Design, Experimentation, Human Factors.

Keywords
University Programming Education, Middle School Programming, Scalable Game Design, Computational Thinking Pattern, Transfer, Student Observation.

1. INTRODUCTION
Since the early 1990's there have been multiple efforts to use end-user video game creation in an effort to teach programming. Examples of these game creation tools include Alice, Scratch, and AgentSheets among others [1,2,3]. The inherent appeal of video games to students gives teachers an entertaining way to introduce the otherwise technical practice of programming [4,5]. These tools have multiple advantages over conventional programming languages such as C++ or Java [2]. For example, to create a simple game, complete with graphics, in C++ or Java can take weeks, or even months of learning. In contrast, as we have observed implementing AgentSheets in numerous classes at different levels ranging from middle school to graduate school, students with no prior programming experience can create their first game in 5 hours. Though game creation is rapid, students are able to learn everything from low level if/then conditionality statements to important programming computational thinking patterns, object-oriented design, and high level concepts like designing games for educational purposes [6]. This ability of game design to scale up as students learn more is called Scalable Game Design.

Scalable Game Design is important in educational applications because it enables learning among both inexperienced and experienced students [7]. A Scalable Game Design method of teaching allows students to learn simple to complex concepts as they gain expertise. This makes Scalable Game Design applicable to the greatest number of students regardless of prior technical knowledge, programming fluency, exposure to technology, or any other background.

This paper introduces our research findings based on experiences related to using game creation as a means of computer science education. These experiences include work with middle school students, university students, and middle school teachers creating a diverse set of games from classic arcade games such as Frogger to educational ecological simulations.

2. METHODOLOGY

2.1 AgentSheets
AgentSheets is a rapid visual agent-based game authoring environment [8]. AgentSheets allows end-users to easily create games and simulations providing abstraction at a sufficiently high level so that novice programmers need not be concerned with low-level implementation details. In addition to having a low starting threshold, AgentSheets also has a high ceiling allowing for complex interactions between agents, such as sophisticated AI techniques [3]. AgentSheets games can be exported as Java applets, allowing for games to be emailed or embedded into a student's webpage. In most classes, students upload their games to the Scalable Game Design Arcade (SGDA).

ITiCSE' 10, June 26–30, 2010, Bilkent, Ankara, Turkey.

2.2 Scalable Game Design Arcade

The Scalable Game Design Arcade is an open classroom cyberlearning infrastructure that allows students to play, download, and comment on and star rate fellow classmates' games, as well as upload their own. At the university level, students are asked to submit their assignments early to garner feedback from other students before the final due date. Students are also encouraged to play classmates' games. An emerging benefit of creating video games to teach programming is that since many students inherently like playing games, they naturally play, give feedback, and take ideas from fellow classmates' games. This is shown in questionnaire data, administered to the University of Colorado "Educational Game Design" class, wherein 22 students were surveyed; 85% of students said they played 2 or more classmates' games each week and 90% said they played one or more [7].

2.3 Computational Thinking Patterns

Computational thinking patterns are agent interactions that not only show up in other programming contexts, but also, show up in other disciplines such as scientific and mathematical contexts. For instance, the Diffusion pattern shows up in many different scientific examples such as heat distribution problems in physics as well as osmosis in biology and chemistry. Computational thinking patterns are fundamental to the idea of transfer--essentially, if a student wants to make a science simulation one or many of these patterns are often employed [9]. We are currently exploring a method of automatic recognition of computational thinking patterns to detect transfer, however, our investigation so far has focused on motivation.

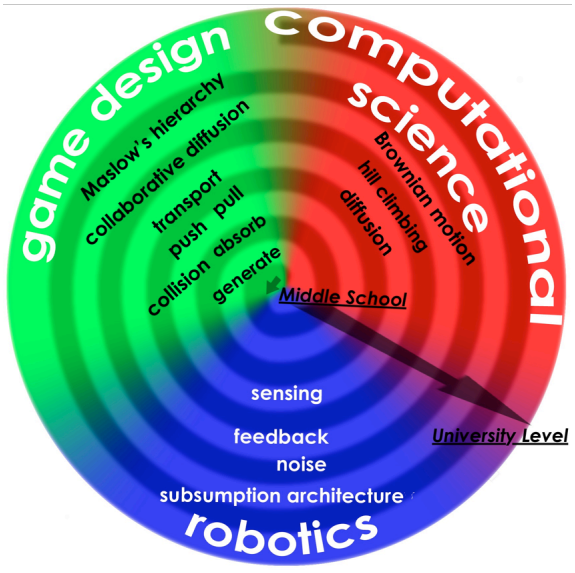

Figure 1 Computational Thinking Pattern spiral and examples of how computational thinking concepts relate to other fundamental concepts in science. The patterns get more complex as one travels away from the center

Figure 1 depicts various computational thinking patterns and how they might transfer to other disciplines. Students learn computational science and computational thinking through making their own games which consist of computational thinking patterns [9,10]. The following computational thinking patterns are used in making games such as Frogger, Sokoban, Centipede, Space Invaders and/or Sims. Table 1 lists these games and their corresponding computational thinking patterns.

Table 1 Games and their corresponding computational thinking patterns

Games	Computational Thinking Patterns
Frogger	Generation, Absorption, Collision, Transportation
Sokoban	Push, Pull
Centipede	Generation, Absorption, Collision, Push, Pull
Space Invaders	Generation, Absorption, Collision
Sims	Diffusion, Hill Climbing

Example patterns include:

Generation: To satisfy this pattern, an agent is required to generate a flow of other agents; for example, a bullet leaving a gun (ie: the gun agent generates a flow of bullets) or a car appearing from a tunnel are both examples of generation. In predator/prey models generation is used to create offspring.

Absorption: This is the opposite pattern of generation. Instead of an agent generating other agents, an agent absorbs a flow of other agents in the absorption pattern (i.e. a tunnel absorbing cars). For example, absorption is used to program a predator eating its prey.

Collision: This pattern represents the situation when two agents physically collide. For example, a bullet or a missile hitting a target creates a collision situation wherein the agents must react to being collided with. In the game Frogger, for example, if a truck collides with a frog, the frog must be "squished."

Transportation: Transportation represents the situation when one agent carries another agents. For example, a turtle in Frogger carries the frog as it crosses the river. In ecological simulations, transportation can be used to have bees carry pollen for example.

Push: Push pattern is the pattern we see in the game of Sokoban. A player in Sokoban is supposed to push boxes to cover targets. As the player pushes the box in Sokoban, the box moves towards the direction (up, down, right or left) it is pushed.

Pull: This pattern is the opposite pattern of push. An agent can pull another adjunct agent or any number of agents serially connected to the puller. For example, you can imagine that a locomotive pulls a large number of railroad cars.

Diffusion: You can diffuse a certain value of an agent through neighboring agents with a diffusion pattern. For example, a torch agent can diffuse the value of heat through neighboring floor tile agents. The closest eight neighboring floor tile agents to the torch agent will have the highest value of heat, and tile agents that are further away from the torch agent will have a lower heat value.

Hill Climbing: Hill climbing is a searching algorithm in computer science. A hill climbing agent will look at neighboring values and move toward the one with the largest value. Hill climbing can be found in the game of Sims or Pacman. In the game of Pacman, Ghosts chase Pacman by following the highest value of Pacman's scent that is diffused throughout the level. This is depicted in Figure 2. As with the torch above, the floor tiles around where Pacman is currently situated have the greatest scent value.

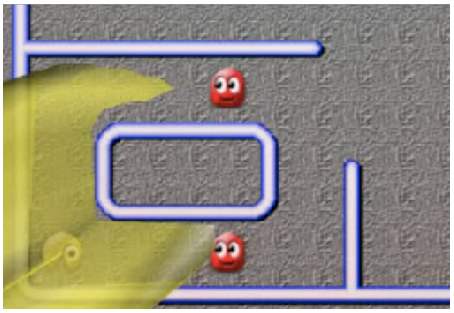

Figure 2 Ghosts use hill climbing on Pacman's diffused scent (pictured around Pacman) to track down Pacman

3. FIELD EXPERIENCE

3.1 Field Observations with Middle School Students

The iDREAMS project at the University of Colorado Boulder is funded by the National Science Foundation (NSF) to bring Information Technology literacy into middle school curriculums. Middle school computer education teachers from schools in tech-hub, urban, rural and remote areas of Colorado have been trained to teach a one to two week AgentSheets course wherein students make the classic arcade game Frogger [9]. Students are taught numerous introductory programming concepts such as if/then conditionality, Boolean logic, procedural abstraction and message passing between agents. Students are also introduced to the notion of computational thinking patterns.

Some of these middle school experiences come from the eCSite project (Engaging Computer Science in Traditional Education), also funded by the NSF through the GK-12 program. This program gives graduate students in science and technology the opportunity to share their research with students in middle and high school classrooms.

From our personal experience in middle school classes and computer clubs we have noticed various significant factors related to teaching strategies when implementing game creation curricula. In previous research we found that games enable the emerging benefit of peer-to-peer interaction in the middle school environment [7]. Students, especially in middle school computer clubs, will wander around the classroom, play other student's games, ask about their game, give feedback to their classmates, and then go back to their computer and implement what they have just learned. Since video games are entertainment, viral peer learning takes place in classrooms with little or no external input.

In our observations, adopting the creation of video games increases class engagement. Students in a couple of schools asked their teachers for extra game creation assignments which, according to the teachers, is quite unusual for a middle school computer science class. One of the most important observations we noticed is that game programming, and specifically this game programming module, is well liked by middle school students regardless of gender and native language. This observation is furthered by Table 2, which shows the initial findings of a post survey given to middles school students after their game design unit was completed. This table indicates that an overwhelming majority of students regardless of language and gender enjoyed the game programming class. Furthermore, almost all students said that they would take another game design class. These results indicate the benefits of employing video game design at the middle school level as it uniformly increases all students' interest in computer science.

When teaching game programming to students in middle schools we have found a marked difference as to how students treat learning materials. Initially, in addition to brief teacher instruction, a step by step in-depth tutorial was given to each student to guide them through the game programming process. This tutorial was long and had a huge amount of text. Not surprisingly, students tended not to read the tutorial, even when they needed help, opting instead to ask for the teacher or a neighbor to help them.

In order to make game instruction more efficient, we developed a new method of teaching materials entitled "cheat sheets." Cheat sheets for each day are about a page long and give students the bare minimum amount of knowledge they need to accomplish a given step in developing a game. More importantly, cheat sheets at the most have 3 sentences of text, instead relying more on pictures to describe the higher level programming tasks that must be accomplished as well as specific images as to how to accomplish them. Cheat sheets lend themselves extremely well to video game learning due to the fact that video games are visual in nature. Unlike other more conventional programming lessons, video game projects can be described extremely well using nothing but pictures. In our experience it was found that this cheat sheet strategy was much more effective in getting students to use the supplementary material.

Table 2 Game design with AgentSheets software can be motivational: preliminary statistics from use in middle school shows that student motivation is high regardless gender and native language.

		Gender				**Language**			
		Boy		Girl		English		Spanish	
		#	%	#	%	#	%	#	%
a) I enjoyed designing games on the computer.	Strongly Disagree/ Disagree	13	16%	17	15%	21	14%	7	20%
	Strongly Agree/Agree	68	84%	98	85%	133	86%	28	80%
	Totals	**81**	**100%**	**115**	**100%**	**154**	**100%**	**35**	**100%**
b) I would like to take another game design class.	Strongly Disagree/ Disagree	26	32%	25	22%	36	23%	13	37%
	Strongly Agree /Agree	56	68%	90	78%	119	77%	22	63%
	Totals	**82**	**100%**	**115**	**100%**	**155**	**100%**	**35**	**100%**

3.2 Field Experience with University Students: Educational Game Design Class

The Educational Game Design Class taught at the University of Colorado Boulder is markedly different from the middle school class. This class lasts for a whole semester, as opposed to 1-2 weeks, and includes a diverse group of upper division computer science, engineering, education, media multidisciplinary, and psychology undergraduate students as well as graduate students. The majority of the students have typically been exposed to programming before, though this is not a pre-requisite. This allows all of the sophisticated computational thinking patterns in Figure 1 to be taught and allows the class to emphasize the theory of effective video game design as opposed to just programming [6,11]. Since the class lasts an entire semester, the Scalable Game Design Arcade becomes an integral part of instruction enabling peer-to-peer learning among students [7].

The class is structured with 9 different gamelet programming assignments. The first four are the same for everyone in the class: Frogger, Sokoban (a box-target moving puzzle game), Centipede, and a 'Sims' type simulation game. Since these assignments are same for everyone, students benefit from downloading a fellow classmate's assignment submission from the Scalable Game Design Arcade, if they do not understand a given topic.

The next four game assignments are individual student assignments. This stage of the class is called "gamelet madness" and students are asked to make one educational gamelet each week. At this point students begin to apply the theory behind creating educational video games and use the computational thinking patterns they have learned. In addition to programming a game, as part of the assignment, they are asked to evaluate one another on how engaging and educational their fellow classmates' games are.

At the end of gamelet madness students start working on their final projects. Final projects can be any type of educational game that the student deems would be effective in teaching a concept. As they program their game, students have middle school students 'play-test' their games at nearby after school computer clubs. Play-testing allows students to garner critical feedback on their final project and to see how engaging and educational the game actually is to one of their target demographics.

In our observations, students learned much more than the above computational thinking patterns in the Educational Game Design Class. As mentioned above, the students in the class have diverse backgrounds and skills, and often, the games they create try to explain complex phenomena from their specific major to other students and the general public. The emphasis on engagement and educational value of games, as well as play testing with middle school students, highlights the importance creating lessons that are effective. Students exhibited an essential component of computational thinking by combining computational thinking patterns to create informative real-world learning simulations often inspired by their respective areas of study. These projects ranged from games that taught language to math puzzles, ecology simulations, and even human body visualizations and DNA sequencing simulations.

Through game creation, learning key points of educational game theory, and play testing games with target audiences, students could better understand what was effective and what was not in terms of educational game creation. In play testing sessions and final presentations we observed that many students had important insights regarding the creation of educational materials for student consumption.

3.3 Field Experience with Community College Students and Middle School Teachers: iDREAMS Summer Institute

In the summer of 2009, as part of the iDREAMS project, the University of Colorado Boulder held a 2 week Summer Institute. The goal of this Summer Institute was to enable middle school teachers and community college students the ability to teach computer science through game design in middle school classrooms across the state of Colorado and even in South Dakota Native American Reservations. In the first week of the Institute, 12 community college students from across the state of Colorado participated in an intense 8-hour a day program wherein they created Frogger, Space Invaders, a Diffusion Simulation, an Ecosystem simulation and a Final Project. During the second week, 21 middle school teachers, aided by community college students from their respective geographical areas, were taught how to create and teach AgentSheets units in their classes. Teachers created the games Frogger, Sokoban, Pacman, and an optional Final Project. The goal of the Institute is that the teachers will go back to their middle schools and teach weeklong AgentSheets units in the coming year with the help of a community college student. So far this school year 8 districts have taught a weeklong AgentSheets unit with over 1,100 middle school students creating AgentSheets games. Preliminary results from this implementation are shown in Table 2.

From our observations of the first week of the Summer Institute, the community college students, much like the university students in the Educational Game Design Class, were able to exhibit transfer. As one of the projects in the Summer Institute, the students were given general guidelines on how to create various ecological simulations such as predator prey models. Using these general guidelines along with applying the programming skills and computational thinking patterns they had obtained throughout the first half of the week, the students were able to create real world ecological simulations. Furthermore, students were able to graph and analyze data related to their simulations such as populations of species.

The second week of the Summer Institute focused on not only creating AgentSheets projects, but also, learning how to teach game design, computational science, and programming concepts with AgentSheets projects. Most of the middle school teachers had very little, if any, prior programming experience and many had never taught a programming unit in their middle school classes. During the second week, through video game design, teachers and community college students were able to learn key educational concepts that allowed them to teach programming units in their classes. Furthermore, the inherent appeal of video games enables teachers to have a captive student audience for their programming unit (see middle school experience above). At present time, these teachers have successfully completed 50 different classes, and as mentioned above, over 1,100 video games from individual middle school students have been created. Moreover, these video game units have not been restricted to computer education classes; for example, in a local school district, teachers are trying to integrate the creation of AgentSheets video games as part of the middle school Spanish curriculum.

A surprising emerging development we found through interviewing community college students after the Institute was that many of these students decided to continue their computer science education beyond the community college as a direct result of their exposure to this project. Multiple students stated that they became motivated to pursue further computer science education, some even computer education paths, while helping middle school teachers create AgentSheets video games. Immediately following the Summer Institute, one community college student transferred to a 4-year university and three more students are preparing their transfer for this coming academic year.

4. DISCUSSION

Video games have universal appeal to audiences ranging from young children to adults. Scalable Game Design in the middle school computer education classes increases student engagement over a variety of demographics and enables the development of more effective teaching strategies. At the university level, results show that video game creation could be used as a springboard into computational thinking, education, and as an avenue to better represent and explain one's own discipline to others. Similar to students, middle school teachers with little to no prior programming experience can effectively teach game design to their classes. Finally, it was observed that Scalable Game Design has the ability to get middle school students as well as college level students interested in the field of computational science.

5. ACKNOWLEDGMENTS

This material is based in part upon work supported by the National Science Foundation under Grant Numbers No. 0833612 and DMI-0712571 and DGE-0841423. 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. Special thanks to Andri Ioannidou and Fred Gluck for their valuable input.

6. REFERENCES

[1] Cooper, S., Dann, W., Pausch, R., Teaching Objects-first In Introductory Computer Science, In Proc. SIGCSE 2003, Reno, Nevada, USA, 2003

[2] Peppler, K. & Kafai, Y. B., Collaboration, Computation, and Creativity: Media Arts Practices in Urban Youth Culture. In C. Hmelo- Silver & A. O'Donnell (Eds.), In Proc. Computer Supported Collaborative Learning, New Brunswick, NJ, USA, 2007

[3] Repenning, A., Excuse me, I need better AI! Employing Collaborative Diffusion to make Game AI Child's Play. In Proc. ACM SIGGRAPH Video Game Symposium, Boston, MA, USA, ACM Press, 2006.

[4] Sturtevant, N. R., Hoover, H. J., Schaeffer, J., Gouglas, S., Bowling, M. H., Southey, F., Bouchard, M., and Zabaneh, G. 2008. Multidisciplinary students and instructors: a second-year games course. In proc 39th SIGCSE Technical Symposium on Computer Science Education, Portland, OR, USA, 2008.

[5] Squire, K., Video games in education. International Journal of Intelligent Simulations and Gaming, (2) 1. 2003

[6] Lewis, C., and Repenning, A., "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, 203-229, 2008

[7] Repenning, A., Basawapatna, A., and Koh, K. H., Making university education more like middle school computer club: facilitating the flow of inspiration. In Proc. 14th WCCCE 2009, Burnaby, British Columbia, Canada, 2009

[8] Repenning, A., "AgentSheets®: an Interactive Simulation Environment with End-User Programmable Agents,"In Proc. Interaction 2000, Tokyo, Japan, 2000

[9] Repenning, A., Webb, D., and Ioannidou, A., "Scalable Game Design and the Development of a Checklist for Getting Computational Thinking into Public Schools," In Proc. SIGCSE 2010, Milwaukee, WI, 2010.

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

[11] Salen, K. Zimmerman, E., Rules of Play: Game Design Fundamentals, MIT Press, 334-337, 2004