Broadening Participation through Scalable Game Design

Alexander Repenning
University of Lugano
Faculty of Informatics
Lugano, TI, 6904, Switzerland
+41 (58) 666 4304
alex.repenning@unisi.ch

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

ABSTRACT
Game development is quickly gaining popularity in introductory programming courses. Motivational and educational aspects of game development are hard to balance and often sacrifice principled educational goals. We are employing the notion of scalable game design as an approach to broaden participation by shifting the pedagogical focus from specific programming to more general design comprehension. Scalable game design combines the Flow psychological model, the FIT competency framework and the AgentSheets rapid game prototyping environment. The scalable aspect of our approach has allowed us to teach game design in a broad variety of contexts with students ranging from elementary school to CS graduate students, with projects ranging from simple Frogger-like to sophisticated Sims-like games, and with diverse cultures from the USA, Europe and Asia.

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

General Terms
Design, Human Factors, Languages

Keywords
Game design, rapid prototyping, flow.

1. INTRODUCTION: THE IT CRISIS
The US needs trained IT personnel to be competitive in science and engineering. In spite of this growing need for IT people, the enrollment in undergraduate Computer Science (CS) programs in North America dropped an astonishing 60% between 2000 and 2004 [1]. The dismal showing of US programming teams at the Association of Computing Machinery (ACM) International Collegiate Programming Contest has raised concerns about the lack of new American programming talent. CS has become highly unpopular for reasons that include an unfounded fear of job outsourcing. A more fundamental problem is a broken pipeline effect in which K-12 students simply fail to get interested in CS based on negative experiences early on. Negative exposures typically fall into the following categories:

SIGCSE'08, March 12–15, 2008, Portland, Oregon, USA.

- ***No exposure***: In the US there are no national IT/CS standards and many states require no IT-related certification for K-12 teachers. The development of state-level curriculum standards for CS in the United States is nearly nonexistent [2]. Some schools simply do not offer any IT courses.
- ***Programming***: The US high schools that do offer CS as an essential discipline typically provide Advanced Placement (AP) courses. This type of curriculum is usually focused on programming [3], and fails to provide motivating applications. Courses of this nature do not attract many students and are even less successful in attracting women and minorities [4, 5]. In 2004 only 11% of the CS AP course takers were female (in contrast to 56% for all AP courses) and only 6% were from under-represented minorities [6].
- ***Multimedia***: These courses are quite popular, but rarely inspire students to pursue IT careers. Multimedia courses are often little more than advanced PowerPoint tutorials.

Even before the recent CS enrollment drop, teacher conferences and organizations such as ACM formed numerous task forces to create a model K-12 CS curriculum. The success of these task forces, as measured by the number of students participating in AP courses, has been limited. According to the task force's final report [2] the 1993 ACM model curriculum [7] was not widely implemented for a variety of reasons, including the emergence of the World Wide Web — this emergence prematurely dated the model curriculum materials. The Computer Science Teacher Association (CSTA), an organization that grew out of the ACM K-12 CS task force, recently declared K-12 CS education to be a "crisis" [6].

Many K-12 students are interested in technology. However, in spite of happily operating technology such as MP3 players, cell phones, and game consoles, the same students shy away from CS courses that take a long time to transition from basic concepts to interesting projects. Model curricula such as the ACM 2003 curriculum [2] suggest participating students start at grade 8 and work on CS foundations for many semesters before they can engage in "interesting subjects like robotics, simulations, and animation" at grade 12. The objects-first [8] introduction to programming movement provides an early, positive example of this pedagogical reversal. Experience with Alice [9] and AgentSheets [10] suggest that it is possible and beneficial to reverse the transition from fundamentals to interesting projects. The challenge, however, is to balance motivational and educational aspects by aligning interesting materials with principled educational frameworks.

A further problem is students' perception of programming itself. In a study conducted in the US and India that explored why girls

are substantially less likely to participate in CS activities, one middle school girl summarized her negative position by saying "I don't like programming, I like creating things." When programming is employed for mundane tasks such as sorting numbers, it becomes especially difficult for students to perceive the programming process as being particularly creative.

The need for IT workers is rising [5] but not all these workers need a full-fledged university CS degree. IT and CS have a lot in common but are not identical [2]. The current K-12 CS models are not working particularly well – they only attract and sustain students who are already strongly interested in CS. We believe that it is possible and necessary to address the pipeline problem with computational environments and a K-12 IT curriculum that complements existing curricula by shifting focus from programming to design. The applied nature of IT is likely to attract more K-12 students, especially when combined with a popular but educational application such as game design [11]. The ideas presented in this paper are not intended to compete with existing curricula, but to complement them by broadening the participation of students not attracted by current course offerings.

2. SCALABLE GAME DESIGN

The evidence that game programming motivates students is increasing [12, 13]. Students are spending more time on assignments and generally exhibit larger levels of energy when working on meaningful applications. However, game programming is no silver bullet. To introduce students to computer science via games is no simple task especially when trying to balance educational and motivational aspects.

Scalable Game Design utilizes the connections between a psychological model, a skill competency framework and a technological environment to help a diverse group of students learn about CS through games. Figure 1 depicts a general focus on design employed to balance skills and challenges when making increasingly complex games and, in the process, acquiring relevant computer science skills.

We have identified three main challenges that need to be addressed in order to support a more principled approach to the educational aspect of game programming: 1) understanding the connection between motivation and learning, 2) being able to conceptualize competencies and 3) supporting the design and development process with technology. Our specific goal is not to educate CS undergraduates to become the next generation of game developers but to use games as a means to broaden participation in CS and to advance design understanding. For us, specific programming languages are less important than conveying a sense of design and process. The following three sections discuss some answers to the three fundamental challenges of scalable game design mentioned above.

2.1 Psychological Model of Motivation: Flow

Balancing design challenges and design skills is hard. There is a need to scaffold game design and to have a well defined space of game projects connected by pedagogical stepping-stones. We use the notion of Flow [14] (Figure 1) as a psychological model of learning. The skills versus challenges space of Flow can index and relate a number of game projects starting with a simple Frogger game all the way up to Sims-like games. The path in the figure denotes a scaffolded learning trajectory, which can be supported by scalable game design. The goal is to remain in the optimal flow zone as much as possible. This can be achieved through various forms of scaffolding, such as explicit just-in-time instruction, social learning support from interactions with other students, or curricula designed in anticipation of the next challenge. Especially from the viewpoint of broadening participation in CS, the lower left point in the Flow space needs to be extremely conservative. We cannot assume any background of the participants. Our benchmark is to have 10-year-old elementary school children without programming experience make a playable Frogger-like game in about three hours.

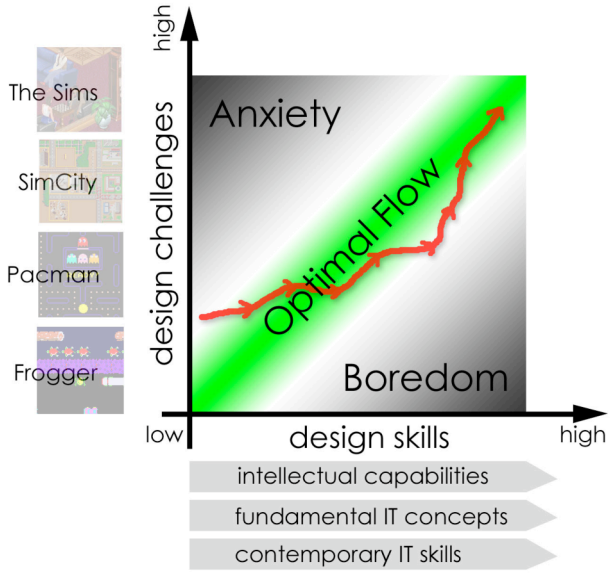

Figure 1. Flow balances design challenges with skills. Design skills map onto the three main threads of the FIT framework.

2.2 Competency Framework: FIT

Computer science and game design are not just about learning how to code. Indeed, the common "computer science = programming" perspective is one of the main reasons why girls especially at the middle school level lose or never gain any interest in computer science. As described in the introduction, existing information technology curricula [7] work poorly in terms of attracting and motivating students [6]. Unlike these curricula, the Fluency with Information Technology (FIT) framework [3] defined by National Academy of Sciences does not focus on programming knowledge. In contrast, it stresses conceptual and design related problem-solving knowledge. FIT includes a variety of skills including working in teams, with non-programming IT tools, and being able to communicate with others. In Figure 1 we include the three essential kinds of knowledge of the FIT framework aligned with the skills dimension of our Flow diagram:

- ***Intellectual capabilities*** problem solving, sthe experience of design trade-offs.
- ***Fundamental IT concepts***: understanding of digital representations, the use of modeling, algorithmic thinking.
- ***Contemporary IT skills***: how to use, but also how to effectively communicate through applications such as word processors and spreadsheets.

A complete description of FIT is beyond the scope of this paper. FIT provides detailed descriptions of 30 fundamental CS/IT skills categories. Relevant for the discussion here are the three kinds of knowledge outlined above in that they cover three levels of skills ranging from universal principles unlikely to change over time to highly specific skills that are relevant but perhaps only short lived.

2.3 Technological Environment: AgentSheets

In order to enable 10-year-old children to make even simple games in just a few hours, it is essential to have the right kinds of rapid prototyping tools. AgentSheets [10] was an early incarnation of the now popular objects-first philosophy [8]. Built-in "rough and ready" drawing tools are similar to fat pen approaches used in architectural drawing design [15]. They allow students to explore design variations with great speed and low commitment. In the spirit of scalability, the visual programming language featured in AgentSheets is highly accessible to students without any programming background. It enables novices to create simple games such as Frogger, yet it is powerful enough for experienced programmers to create sophisticated games such as Sims-like games and scientific simulations using AI techniques [16]. This kind of versatility is essential for scalable design. A tool that has either a low threshold (i.e. makes it easy for novices to get started) or a high ceiling (i.e. makes it easy to continue and enable working on sophisticated content with increasing expertise) but not both will not work. Also important for high challenge / high skill Flow is the ability of AgentSheets to turn games artifacts consisting of rule-based behaviors and multimedia content into Java source code and bytecode to provide learners an under the hood look on how games can be programmed using traditional programming languages. Other educational programming environments could be used to substitute AgentSheets as long as they fit the above scalability profile.

3. CONTEXTS & RESULTS

This section discusses the three scalable game design challenges in context by presenting the settings, audience, goals, and results of game design activities that we conducted in the USA, Europe and Asia in the last 10 years.

3.1 Game Design University Outreach

The problem of declining numbers of CS undergraduates is an international phenomenon. At the University of Colorado at Boulder and the University of Lugano in Switzerland we have conducted game design activities as outreach to attract K-12 students to undergraduate programs.

Initially the target audience was high school honors students. The perception by the faculty was that the focus should be the high school level, as high school student directly feed into the undergraduate pipeline. However, there is increasing evidence that especially minority students and girls lose interest in science and technology-flavored courses already in middle school. In this light it is essential to gear game design courses at earlier stages of the education pipeline. Therefore we and gradually dropped the requirement to include middle school students.

Outreach game design activities are highly constrained in terms of time. In a three-hour period, students need to be able to have a complete experience. Less than 10% of the participating high school students had any programming experience. In terms of Flow, essentially no design skills can be assumed and consequently the design challenges need to be minimal.

After some basic introduction illustrating the history of games students are led through a design process we call Gamelet Design. Whenever possible, the students are given original game descriptions. For instance, the Sega description of Frogger:

> You are a frog. Your task is simple: hop across a busy highway, dodging cars and trucks, until you get the to the edge of a river, where you must keep yourself from drowning by crossing safely to your grotto at the top of the screen by leaping across the backs of turtles and logs. But watch out for snakes and alligators! (Sega, 1980)

Consistent with the element of analysis of the intellectual capabilities thread of FIT, students begin to classify objects and operations by identifying nouns and verbs of the project description. This is an activity found in many introductory object-oriented design courses. Additionally, students discuss behavior of objects and explore how to deal with object interaction. For instance, in the case of Frogger, a fundamental design question is how to deal with the interaction between cars and frogs. In a car-centric design, the car (autonomous agent) looks out for the frog (user controlled agent) and crushes it when it comes too close. A discussion will contrast this with a frog-centric solution.

Learning how to use AgentSheets is not the objective of the activity, but the idea of modeling and creating digital representations is part of the fundamental IT concepts of the FIT framework. Modeling becomes more involved as students need to think how to model the traffic. Where are the cars and trucks coming from? Are they all placed by the user, generated periodically or generated by some stochastic process? Finally, the fact that frogs can jump onto logs, which in turn float on water, introduces the notion of a stack and creates some non-trivial algorithmic decisions on how to implement transportation.

At the end of the session most students are highly excited about their games in spite of limited production quality. The artwork is limited to cars, frogs and other agents that they have created in the 3-hour period as part of their game design. Most of the drawings are crude and certainly do not rival any of the artwork found in modern video games. However, this is completely irrelevant to most students as the creation of their own content results in more engagement.

The university outreach activity included surveys. Ten Engineering School departments offered such activities. Each high school student could participate in up to three department activities. Even with strong competition from the Electrical Engineering and Aerospace Engineering departments the game design Computer Science activity was ranked first.

3.2 Object Oriented Programming

Especially in early stages of object-oriented programming education it is essential to get a sense of what objects are and how they relate with each other. In a university setting for CS undergraduate students we use the first part of the course to acquire this kind of sense through game design.

The introduction of the object-oriented programming course employs an aggressive project-oriented game design component based on weekly game design and implementation homework. Each student individually produces the same set of games ranging over a large part of the design Flow diagram from Frogger-like to the Sims-like games. Each week the games are collected as Java applets in a web pages open to the entire class. An open project format allows students to get inspired by other students' implementation. This has raised the average quality of the game

designs. The social aspect of seeing other students' product is essentially used as a social scaffolding mechanism.

Each week the teacher presents new interaction patterns relevant to the games helping students to build a growing repertoire of language-independent patterns. Patterns include ways to organize large numbers of synchronized agents, e.g. the space ships attacking in Space Invaders, or interaction through delegation, e.g., for the person pushing boxes onto targets in Sokoban.

The scalability aspect of game design in this context is not limited to an increasing complexity of the games. Consistent with the intellectual capabilities thread of the FIT framework, students also learn to employ visualizations for problem solving. Few students initially like the idea of UML diagrams and conceptualize them as a documentation approach. In the context of game design at least some begin to appreciate the potential value of these diagrams for thinking through complex object interactions.

Again the goal of the course is not to learn how to use a tool such as AgentSheets but to acquire general design skills. These design skills need to be connected back to implementation skills. In other words the three threads of the FIT framework need to be connected. A good way to do so is by helping students think explicitly about these connections. AgentSheets has a rule-based visual programming language. Students have to correlate their rule-based programs with UML sequence diagrams. Which object/agent is sending what kinds of messages to what kind of other agent? When are these messages being sent? What are the guards of the messages? As a different exercise students need to map high level to low-level representations. As part of the contemporary skills FIT thread, they use the tools du jour to turn Java class files into Java sources and Java bytecode assembly languages. They have to correlate their game creations at a structural level with Java source and bytecode. The frog agent turns into a frog Java class. Into what kind of Java construct do the rules turn? Into what kind of bytecode does a case statement in Java turn into? These connections can be quite abstract and initially many computer science students without a solid compiler background struggle to comprehend them. However, the reverse engineering nature of the process aids enormously because they are not trying to find arbitrary mappings, but they are analyzing their own constructions. Even at the bytecode level, they begin to recognize patterns based on identifiers they used in their game.

With an overall course satisfaction of 90%, the last incarnation of this course was ranked second of all Spring 2007 courses (n=15), taught in the Informatics department at the University of Lugano, Switzerland. Student satisfaction was established by having a curriculum that was incremental enough to keep students in the optimal flow of learning. Expectations management also is key. Students may want to build the next version of Halo during the class, but curbing their expectations to realistic yet fulfilling for the students goals is important.

3.3 Educational Game Design

This annual course at the University of Colorado generally attracts a large group of undergraduate and graduate students from computer science and other departments. This course starts similarly to the introduction to object-oriented programming course. Students quickly follow the scaffolding trajectory of the design Flow diagram within the first few weeks of the course. UML diagrams and reverse engineering are not used in this course since non-CS majors take it as well. The education aspect of the course has no apparent filtering effect as most students claim to be interested in games but willing to tolerate the educational flavor.

The weekly game part of the course prepares students also for their semester project. In this project, they have to build an educational game for middle school students. This requirement is initially met with a lot of resistance. Undergraduate as well as graduate students are typically not used to the notion of customers. Worse, middle school students tend to be highly vocal and critical customers of educational games. In the first week of the final project the university students need to be dragged into middle schools. Most university students expect praise for their clever games but get critique instead. An explicit part of the course is the option and the necessary time to radically redesign or even abandon game designs. This works well to the point where, in contrast to the first few times of middle school fields trips, the university students need to be dragged out of middle schools.

In terms of the FIT framework this context provides opportunities for explaining approaches, discussing problem solving trade offs and negotiating design decisions.

3.4 Science Discovery

Science Discovery is an experience-based educational outreach organization of the University of Colorado at Boulder with a mission to stimulate scientific interest, understanding, and literacy among Colorado's students and teachers through after school and summer programs. A course called Video Game Creations allows students to design and build their own video games. The course has been offered for two years. In this time it has become one of the most popular Science Discovery courses. During the school year, the Video Game Creations class runs once a week for five weeks. The approach starts similar to the shorter outreach courses mentioned in section 3.1 above with Gamelet Design to illustrate the entire process of creating a complete game. Having more time to our disposal, we can start even lower in the design challenges with building a simple example simulation; this is necessary since these classes are targeted to middle school students and in reality get some elementary school students as well. However, we also have the time to go deeper in some aspects of game design and really connect game design with computer science. For instance we discuss OO concepts such as message passing.

Game design is important for attracting students to take IT/CS classes when their time is being solicited by other activities. We tried offering very similar classes under the heading of Interactive Science Simulations, a different version of the same course but slightly more focused on science simulations (e.g. how fires spread) instead of building video games. The Video Game Creation course always works much better to attract students and also keeps them motivated throughout the duration of the class. Motivation, however, is not the only goal for offering these classes. Educating students about IT/CS is achieved as well. In the FIT framework, these game design courses offer a good context for covering a lot of ground in intellectual capabilities. While we do not use formal design methods (e.g. UML diagrams) that would not be appropriate for young students, we can still use OO design methods such as the definition of objects and methods via noun and verb identification of game descriptions to scaffold the process of creating games.

The results of these courses as expressed in standard evaluation surveys administered by Science Discovery are very encouraging. Students overwhelmingly report that they are "ecstatic" or

"happy" for taking the class. Students and their parents report that they are learning a lot about creating video games and abstract thinking. The biggest complaint is that the course does not last long enough. Most students want to have follow-up courses on "advanced topics" in game design. We have not yet offered such a class through Science Discovery, but it would make sense to do so to continue in the Flow trajectory with more complex games.

3.5 Computer Club

Computer clubs at various middle schools are highly informal CS learning contexts. Students typically do not have to enroll. They come and go. Computer clubs manage to attract large crowds of students. In this often noisy and seemingly chaotic kind of environment, students' skills are all over the place in terms of the design Flow. The nature of skills is typically at the FIT contemporary IT level. That is, they know how to use all kinds of applications including sophisticated ones such as Photoshop. However, intellectual capabilities and fundamental IT concepts are often lacking.

Computer clubs can be ideal social learning ecologies. Initially, we sent graduate students to help kids and teachers in a computer club to design games using some of the techniques described above. After a short while we realized that our help was no longer necessary. Social scaffolding takes place by having children with more advanced skills helping others. "How did you do that?" is a question we often heard as a starting point of design reuse.

When students drift out of the Flow zone (design challenges = design skills) into early anxiety (design challenges > design skills), they are highly receptive to just-in-time learning even if the topic is complex and traditionally considered over their head from an academic point of view. Case in point: a student working on a maze game with zombies tracking robots needed to improve the currently randomly moving robots with "better AI". We explained the collaborative diffusion [16] approach employing 3D visualization built-in to AgentSheets. Needless to say, diffusion equations are not part of the middle school math curriculum. When we returned the following week we found not only that student, but an entire group of students working on games using diffusion. It was truly amazing to see students allegedly not interested in math explaining to each other how to make even more sophisticated games by tweaking diffusion coefficients.

4. CONCLUSIONS

The psychological models of Flow, the competency framework called FIT and the technological environment of AgentSheets have been combined into the notion of Scalable Design in the context of video games. By providing a low-threshold/high-ceiling framework and supporting skills beyond programming, ranging from theoretical design skills to concrete development skills, we can broaden computer science participation.

5. ACKNOWLEDGMENTS

We would like to thank University of Colorado's Science Discovery and Centennial Middle School for their continued support for our research. This material is based in part upon work supported by the National Science Foundation under Grant Numbers No. 0205625 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.

6. REFERENCES

[1] Microsoft, "More than Fun and Games" Available at http://www.microsoft.com/presspass/features/2005/sep05/09-12CSGames.mspx.

[2] ACM K-12 Task Force Curriculum Committee. "A Model Curriculum for K–12 Computer Science," Association for Computing Machinery, Computer Science Teachers Association, New York, NY. October, 2003.

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

[4] National Science Foundation, "Women, Minorities, and Persons with Disabilities in Science and Engineering," Arlington, VA NSF 04-317. May 2004.

[5] AAUW Educational Foundation Commission on Technology, Gender, and Teacher Education, "TECH-SAVVY: Educating Girls in the New Computer Age". April 2000.

[6] Computer Science Teachers Association, "Achieving Change: The CSTA Strategic Plan," (http://csta.acm.org/About/sub/StrategicPlanWebNew.pdf) 2005.

[7] Corporate Pre-College Task Force Committee of the Educucational Board of the ACM, "ACM model high school computer science curriculum," Communications of the ACM, vol. 36, pp. 87-90, 1993.

[8] D. J. Barnes and M. Kölling, Objects First with Java: A Practical Introduction using BlueJ, Third ed: Pearson Education / Prentice Hall, 2006.

[9] M. Conway, S. Audia, T. Burnette, et al., "Alice: Lessons Learned from Building a 3D System For Novices," presented at CHI 2000, The Hague, Netherlands, 2000.

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

[11] Y. Kafai, "Playing and making games for learning: Instructionist and constructionist perspectives for game studies.," Games and Culture, vol. 1, pp. 36-40, 2006.

[12] J. D. Bayliss and S. Strout, "Games as a "Flavor" of CS1," presented at SIGCSE'06, Houston, Texas, USA, 2006.

[13] I. Parberry, M. B. Kazemzadeh, and T. Roden, "The Art and Science of Game Programming," presented at SIGCSE'06, Houston, Texas, USA, 2006.

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

[15] M. D. Gross, "The Fat Pencil, the Cocktail Napkin, and the Slide Library," presented at Proceedings of Association for Computer Aided Design in Architecture (ACADIA '94) National Conference, St Louis, 1994.

[16] A. Repenning, "Collaborative Diffusion: Programming Antiobjects," presented at OOPSLA 2006, ACM SIGPLAN International Conference on Object-Oriented Programming Systems, Languages, and Applications, Portland, Oregon, 2006.