

**Integrating computer games into existing CS courses may help attract students to the field, but there are guidelines to be considered.**

BY KELVIN SUNG

# Computer Games and Traditional CS Courses

SINCE COMPUTING IS the foundation of modern society, a proficient computing work force is essential for maintaining the country's leadership and competitiveness in the global economy. The recent decline in enrollments across computer science (CS) departments and the decrease in student diversity pose significant challenges to the continuation of the nation's prominent position in the global high-technology arena. The CS education community responded to this challenge with a general critical self-reexamination where the entire traditional CS education system is being evaluated, from the outreach to K-12 education, to the fundamental philosophies behind the curriculum design. One of the emerging results from these developments is

the push for presenting abstract CS concepts in the context of familiar real-world applications.

Relating abstract principles to real-world experience has become increasingly prominent in mathematics and general science education. For example, the *Calculus Reform* movement of the 1990s included both pedagogical changes and foci on real-world problems, while the Carl Wieman Science Education Initiative at the University of British Columbia has redesigned its freshmen introductory physics course such that:<sup>a</sup>

*"As much as possible, the standard introductory physics material will be presented in connection with real-world situations and issues such as home heating, transportation, and electricity generation."*

In the CS education arena, the Media Computation of Georgia Tech<sup>18</sup> is an excellent example where foundational programming concepts are presented in the context of popular digital multimedia applications. This contextualization of computing education<sup>18</sup> is an ongoing effort and interactive computer video games, being one of the most familiar application areas for our students, is a context favored by many CS educators.

This article presents the USC GamePipe Laboratory effort where the entire CS curriculum is redesigned in the context of game development (Please refer to the USC GamePipe Laboratory effort by Michael Zyda on page 66 where the CS curriculum is designed in the context of game development). This article examines the ongoing efforts to integrate computer video games in existing traditional CS courses. The discussion is divided into introductory programming courses and elective CS courses, and concludes with guidelines for considering integrating computer game content into existing CS classes.

## Games and CS Classes

There are many types of games that

a <http://www.cwsei.ubc.ca/departments/physics-astro/courses.htm> (Nov. 2007 update).

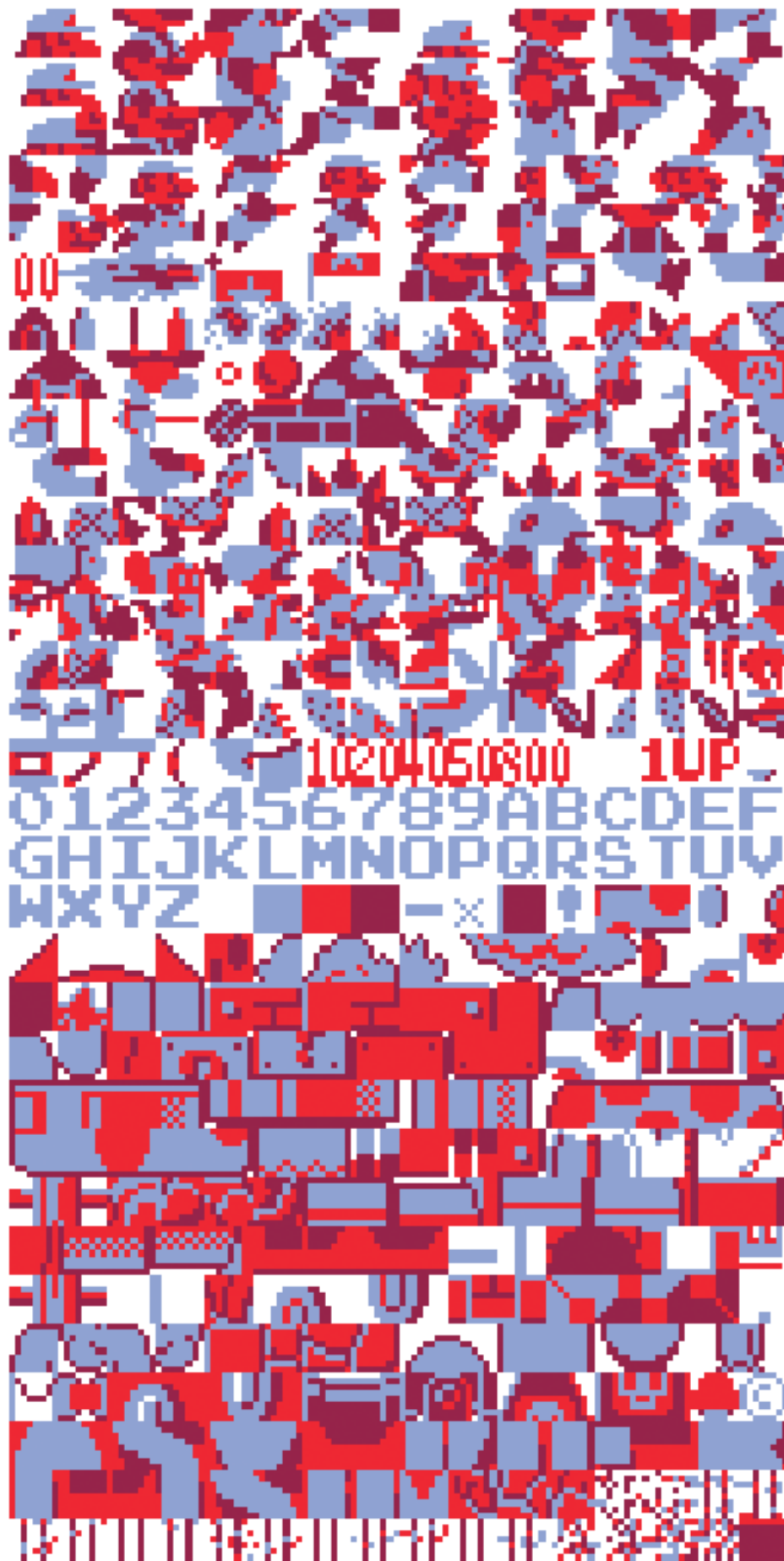
are suitable for teaching CS subjects including many noncomputer games or games that are based on dedicated devices (for example, Lego robots). Our focus is on interactive graphical computer games. It is important to recognize that in the CS education arena the term “computer game” is often used to refer to the attempts at, and the results of, effective and meaningful integration of *animated graphical visualization and various degrees of interactivity*. Because of the unknown entertainment value, strictly speaking, much of these are interesting and innovative teaching materials and are not computer games in a commercial sense.

As discussed in Sung et al.,<sup>33</sup> when examining recent efforts in integrating computer games into CS classes, we observe three general categories.

**1. Game development classes.** These are entire curricula,<sup>7,40</sup> individual classes,<sup>4,6,14,31,39</sup> or capstone projects<sup>2,29</sup> designed specifically to develop new games as an end product. The education committee of the main professional organization for the games industry, The International Game Developer Association (IGDA), has proposed a comprehensive curriculum framework conveying the industry’s articulated desires for well-trained college graduates seeking jobs in the game industry. When evaluated against the IGDA curriculum framework, we see that these classes cover all the major core topic areas. Students in these classes must be concerned with all aspects of producing a real game including entertainment value, visual quality, audio effects, physics simulations, and real-time performance.

**2. Game programming classes.** These are classes (for example, Kuffner’s CMU course<sup>37</sup>) designed specifically to study technical aspects and issues involved in building games. For example, topics covered may include event loops, path planning algorithms, and terrain representation. These classes typically do not require building an end product and the topics covered are general and usually are applicable to different domains. These classes concentrate on covering the game programming topic area in the IGDA curriculum framework.

**3. Game development client.** These are existing CS classes that creatively in-



tegrate games into their existing curriculum. Typically, games are used as programming assignments,<sup>1,3,5,22,25,33,34,37</sup> or to teach abstract concepts,<sup>11,15,30</sup> or as an example application area to teach the concepts involved in an entire topic area.<sup>9</sup> These are traditional CS classes that exist independent of game programming. These classes are actually clients of game development where they use game development as a vehicle to deliver specific abstract concepts. After these classes, students are expected to understand the abstract CS concepts, and not the details of game development.

Courses in the first two categories are *new* courses designed to teach students about game development. Over time, as the game development field matures, it is expected that these courses will evolve and eventually some of the contents will become part of the standard CS curriculum. This is not unlike the early years of many existing disciplines in CS (for example, software engineering<sup>13</sup> or computer graphics<sup>8</sup>), where the syllabi of pioneering courses consolidated as the disciplines mature. Courses in the third category, the “game development clients,” are traditional CS courses that can be found in existing CS curricula. The earliest work in this area<sup>1,12</sup> adapted games almost anecdotally without holistic considerations; most of the more recent work is structured around addressing core competency areas with reference to the ACM Curriculum. Accordingly, courses in the “game development clients” category can be divided into two broad efforts: introductory programming classes (CS1/2) and advanced/elective classes.

### Games and Introductory Programming Classes

Many CS educators recognized and took advantage of younger generations’ familiarity and interests for computer video games and integrate related contents into their introductory programming courses. Because these are the first courses students encounter, they build excitement and enthusiasm for our discipline.<sup>24, b</sup> Based on the type of

effort required by faculty, existing work done in this area can be classified into three broad approaches:

*Little or no game programming.*<sup>9,17</sup> In these courses students learn by playing custom games but they do not actually program the games.

*Per-assignment game development.*<sup>3,21,32,33,38</sup> All these classes developed games as part of individual programming assignments. In each case, isolated games are designed around technical topics being studied.

*Extensive game development.* For example, faculty must design programming assignments based on custom library,<sup>39</sup> general game engines,<sup>4</sup> dedicated game engines,<sup>25</sup> specialized programming environments,<sup>22</sup> custom object-oriented class hierarchies,<sup>25</sup> specific curricula,<sup>23</sup> or new programming languages.<sup>11</sup>

Much of this work reported resounding successes with drastically increased enrollments and student successes.<sup>3,11,23</sup> Based on these results, it is well recognized that integrating computer gaming into CS1 and CS2 (CS1/2) courses, the first programming courses students encounter, is a promising strategy for recruiting and retaining potential students. With the enrollment challenges faced by the CS discipline, it is desirable and important that this strategy can be adopted widely by all interested faculty and departments.

However, most of the existing work in this area is based on pioneering exploratory projects by faculty members with expertise in computer graphics and gaming.<sup>3,23,28</sup> With few exceptions, these projects are *student-centric* where the main goals of study are student engagement and various learning outcomes. Adaptability and generality of the resulting materials are usually not main concerns. For the faculty members teaching CS1/2 courses, most of which are without computer graphics or gaming background, it can be challenging to take advantage of these results.

In addition, when considering experimentation with CS1/2 courses, it is important to appreciate institutional oversight procedures. Though becoming less controversial in recent years, many CS educators continue to be unsure about integrating gaming in formal educational settings.<sup>20</sup> It can be challenging in departmental commit-

tees to arrive at consensus for significant modifications to CS1/2 courses, especially if the modifications involve computer games. For these reasons, to be widely adaptable, game-related CS1/2 materials should be designed with the following considerations:

1. The materials should not demand knowledge in computer games or graphics.

2. The materials should include independent modules that are limited in curriculum scope.

3. The materials should support selective experimentation by individual faculty members in small-scale pilot demonstration projects in their existing courses.

*Selective Gradual Adoption.* Results from the extensive game development approach discussed previously typically include large amounts of adoptable/adaptable courseware materials. However, using these materials often requires a significant investment of time, for example, understanding a game engine, or significant reworking of an instructor’s existing curriculum. Because of the considerable overhead, results from this approach are typically not suitable for selective adoption.

In terms of suitability for selective adoption, we expect that results from the per-assignment game development approach will be most applicable. For example, one could selectively replace nongame assignments in existing classes by the corresponding games assignments. However, because of the pioneering nature of work in this area, many of the results on per-assignment game development are “anecdotal” and do not discuss the impact of such assignments on the CS1/2 curriculum holistically. For example, the results from Huang only involve turn-based strategic games,<sup>21</sup> Ross only discusses puzzle games,<sup>32</sup> and the discussion from Valentine is based on a single game.<sup>38</sup>

The *Game-Themed Introductory Programming Project* at the University of Washington, Bothell<sup>c</sup> is specifically designed to address these issues. In the first phase of our project, we have designed and built general game-themed CS1/2 programming assignment modules that demand no existing knowl-

<sup>b</sup> It is important to reiterate that, after these classes students are expected to understand abstract programming concepts rather than concepts specific to building games.


<sup>c</sup> <http://depts.washington.edu/cmmr/Research/XNA Games/index.php>

edge of games or graphics from the faculty,<sup>33</sup> and have demonstrated it requires minimum changes to existing classes in order to successfully adopt these materials. Currently in the second phase of our project, we are building game-themed examples and tutorials designed to provide a pathway for interested faculty to gradually incorporate game-related materials into their existing courses. Our project is student-centric because our materials allow students to practice CS concepts in a more real-world-like context. More importantly, the materials are also *faculty-centric* because these materials are the stepping-stones for faculty to begin experimenting with a promising new approach to teaching CS1/2 courses.


### Games and Elective CS Courses

As highlighted earlier, the CS education community has a sound understanding of how to integrate visualization and interactivity in delivering CS1/2 content and has achieved impressive successes. In comparison, there is a relatively modest amount of work done in integrating computer games into existing traditional CS elective classes. This is not surprising as a successful systematic integration requires the delivery of an entire technical topic area to lend itself well in visualization and interactivity. There are anecdotal examples of using game content in delivering selective topic areas (for example, design patterns,<sup>16,27</sup> or spatial search algorithms<sup>34</sup>). These are small-scale projects not meant to address entire courses as identified in the standard CS curriculum.

Artificial intelligence (AI),<sup>9</sup> software engineering (SE),<sup>5,10,37</sup> and computer graphics (CG)<sup>36</sup> are examples of elective courses where published results describe attempts at systematically integrating game development. In all of these cases, the stated student learning outcomes are similar to those from the typical CS curriculum and do not include competencies involved in game development as defined by the IGDA curriculum framework. In these classes, students study the core topic areas and implement games to demonstrate their understanding of the fundamental CS concepts. This work reported high student engagement and enthusiasm, while pointing out that the faculty



**While it is the case that proper integration of game development and game content in CS classes have the potential to further engage students resulting in higher success rates, it is not the case that any game content will result in having a positive impact.**



members involved must develop large amounts of software infrastructure to facilitate and support students' game development.

Notice that all three of these topic areas have significant overlaps with computer games in general: intelligent behavior (AI) is one of the most important attributes of modern games, SE methodologies are applicable in any software product development, and topics in CG are the conceptual framework for visualization in games. One can argue that for these topic areas, it is relatively straightforward to integrate game content in a consistent manner. In general, for topic areas that do not offer obvious overlaps with computer games or game development (for example, compiler or programming languages), dedication and creativity would be required to develop the elaborate infrastructure and to systematically integrate the new contents. In these cases, one should carefully examine the trade-offs between required efforts, expected benefits, and consider other perhaps more appropriate practical contexts (for example, popular applications on the Internet).

### Guidelines for Consideration

While it is the case that proper integration of game development and game content in CS classes have the potential to further engage students resulting in higher success rates, it is not the case that any game content will result in having a positive impact. In addition, when exploring the potential for development or adoption of game content, we must work within the bounds of institutional oversights and be conscious about the expertise areas of faculty members. The following are some factors for consideration:

- ▶ *Institutional oversight.* Departmental committee consensus is often required for significant changes to core courses. Because of the potential impact, it can be especially challenging to arrive at a consensus for modifications to introductory-level courses like CS1/2. A strategy is to design limited-curriculum-scope experiments to gain experience (and collect results) to assist the committee's decision-making process.

- ▶ *Faculty background.* Many faculty members did not grow up playing computer games and most are not familiar with graphics programming. When

developing or evaluating materials for adoption, it is essential to pay attention to the prerequisite knowledge. An ideal approach would be to clearly separate and hide the graphical and user interactivity functionality. In this way, faculty and students only need to concentrate on the core CS concepts.

► *Gender and expertise neutrality.* As with any powerful tool, inappropriate use of games can backfire and result in further alienation of underrepresented groups.<sup>20</sup> It is important that the gaming materials are gender and expertise neutral. For example, it is important to avoid violence and unnecessary competitions.<sup>26</sup> The materials used should discourage the addition of superfluous “eye-candy” graphical enhancements, or user interaction programming by students with extensive prior programming experience. Doing so will help to avoid intimidating other less-experienced students.

► *Infrastructure support.* Free and simple are the keywords here. Given the financial reality of most schools, all materials must be freely available; the associated institutional infrastructure requirements must be modest and straightforward.

► *Conceptual integrity.* Our discussion focuses on the traditional CS courses. It is important to remember that ultimately, the goal is to facilitate students’ learning about the core CS concepts. Any dilution, even in favor of acquiescing to some students’ desire and motivation to become game developers, would do the students a disservice.

► *Textbook availability.* As in all pioneering work, mature and well-organized materials are mostly under development. Although there are some textbooks available for specific approaches (for example, CS1/2,<sup>11</sup> computer graphics<sup>35</sup>), mostly, one must be ready to develop custom reading material to guide students along.

## Acknowledgments

This work is supported in part by the National Science Foundation grant DUE-0442420 and Microsoft Research under the Computer Gaming Curriculum in Computer Science RFP, Award Number 15871 and 16531. All opinions, findings, conclusions, and recommendations in this work are those of the authors and do not necessarily reflect the

views of the National Science Foundation or Microsoft. □

## References

1. Adams, J.C. Chance-It: An object-oriented capstone project for CS-1. In *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, NY, 1998, 10–14.
2. Barnes, T., Richter, H., Powell, E., Chaffin, A., and Godwin, A. Game2learn: Building CS1 learning games for retention. In *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*. ACM, NY, 2007, 121–125.
3. Bayliss, J.D. The effects of games in CS1-3. *Journal of Game Development* 2, 2 (2007).
4. Bierre, K., Ventura, P., Phelps, A., and Egert, C. Motivating OOP by blowing things up: An exercise in cooperation and competition in an introductory Java programming course. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, NY, 2006, 354–358.
5. Chen, W-K and Cheng, Y.C. Teaching object-oriented programming laboratory with computer game programming. *IEEE Transactions on Education* 50, 3 (Aug. 2007), 197–203.
6. Clark, B., Rosenberg, J., Smith, T., Steiner, S., Wallace, S., and Orr, G. Game development courses in the computer science curriculum. *J. Comput. Small Coll.* 23, 2 (2007), 65–66.
7. Coleman, R., Krembs, M., Laboureur, A., and Weir, J. Game design and programming concentration within the computer science curriculum. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, NY, 545–550.
8. Cunningham, S., Brown, J.R., Burton, R.P., and Ohlson, M. Varieties of computer graphics courses in computer science. In *Proceedings of the 19th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, NY, 1988.
9. da Silva, F.S.C. Artificial intelligence for computer games. University of Sao Paulo, Microsoft Academic Alliance Repository Newsgroup, 2006; <http://www.msdnaacr.net/curriculum/pfv.aspx?ID=6210>.
10. da Silva, F.S.C. Software engineering for computer games. University of Sao Paulo, Microsoft Academic Alliance Repository Newsgroup, 2006; <http://www.msdnaacr.net/curriculum/pfv.aspx?ID=6211>.
11. Dann, W., Cooper, S., and Pausch, R. *Learning to Program with Alice*. Prentice Hall, Upper Saddle River, NJ, 2006.
12. Faltin, N. Designing coursework on algorithms for active learning with virtual board games. In *Proceedings of the 4th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education*. ACM Press, NY, 1999, 135–138.
13. Freeman, P. Software engineering education: Needs and objectives. In *Proceedings of the ACM SIGCSE-SIGCUE Technical Symposium on Computer Science and Education*. ACM Press, NY, 1976, 266.
14. Frost, D. Ucgime, A Java library for games. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, NY, 2008.
15. Gestwicki, P.V. Computer games as motivation for design patterns. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, NY, 2007, 233–237.
16. Gestwicki, P.V. Computer games as motivation for design patterns. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, NY, 2007, 233–237.
17. Giguette, R. Pre-games: Games designed to introduce CS1 and CS2 programming assignments. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, NY, 2003, 288–292.
18. Guzdial, M. Contextualized computing education. Invited Presentation, Microsoft Research Faculty Summit; <http://home.cc.gatech.edu/guzdial/169> (July 2008).
19. Haden, P. The incredible rainbow spitting chicken: Teaching traditional programming skills through games programming. In *Proceedings of the 8th Australian Conference on Computing Education*. Australian Computer Society, Darlinghurst, 2006, 81–89.
20. Haller, S., Ladd, B., Leutenegger, S., Nordlinger, J., Paul, J., Walker, H., and Zander, C. Games: Good/evil. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, NY, 2008, 219–220.
21. Huang, T. Strategy game programming projects. In *Proceedings of the 8th Annual CCSC Northeastern Conference on the Journal of Computing in Small Colleges*. Consortium for Computing Sciences in Colleges, 2001, 205–213.
22. Külling, M. and Henriksen, P. Game programming in introductory courses with direct state manipulation. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*. ACM Press, NY, 2005.
23. Leutenegger, S. and Edgington, J. A games-first approach to teaching introductory programming. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, NY, 2007, 115–118.
24. Lewis, M., Leutenegger, S., Panitz, M., Sung, K., and Wallace, S.A. Introductory programming courses and computer games. In *Proceedings of the 40th SIGCSE Technical Symposium on Computer Science Education*. Mar. 2009.
25. Lewis, M.C. and Massingill, B. Graphical game development in CS2: A exible infrastructure for a semester long project. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, NY, 2006, 505–509.
26. Natale, M.J. The effect of a male-oriented computer gaming culture on careers in the computer industry. *SIGCAS Comput. Soc.* 32, 2 (2002), 24–31.
27. Nguyen, D.Z. and Wong, S.B. Design patterns for games. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*. ACM Press, NY, 2002, 126–130.
28. Parberry, I., Kazemzadeh, M.B., and Roden, T. The art and science of game programming. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, NY, 2006, 510–514.
29. Parberry, I., Roden, T., and Kazemzadeh, M.B. Experience with an industry-driven capstone course on game programming: Extended abstract. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, NY, 2005, 91–95.
30. Pulimood, S.M. and Wolz, U. Problem solving in community: a necessary shift in CS pedagogy. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, NY, 2008, 210–214.
31. Repenning, A. and Loannidou, A. Broadening participation through scalable game design. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*. New York, NY, USA, 2008.
32. Ross, J.M. Guiding students through programming puzzles: Value and examples of Java game assignments. *SIGCSE Bull.* 34, 4 (2002), 94–98.
33. Sung, K., Panitz, M., Wallace, S., Anderson, R., and Nordlinger, J. Game-themed programming assignments: The faculty perspective. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, NY, 2008, 300–304.
34. Sung, K. and Shirley, P. Algorithm analysis for returning adult students. In *Proceedings of the Sixth Annual CCSC-NW Conference, J. Computing Sciences in Colleges* 20, 2 (Dec. 2004) 62–72.
35. Sung, K., Shirley, P., and Baer, S. *Essentials of Interactive Computer Graphics: Concepts and Implementation*. A.K. Peters, Wellesley, MA, 2008.
36. Sung, K., Shirley, P. and Reed-Rosenberg, R. Experiencing aspects of games programming in an introductory computer graphics class. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, NY, 2007, 249–253.
37. Sweedyk, E., deLaet, M., Slattery, M.C., and Kuffner, J. Computer games and CS education: Why and how. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, NY, 2005, 256–257.
38. Valentine, D.W. Playing around in the CS curriculum: Reversi as a teaching tool. *J. Comput. Small Coll.* 20, 5 (2005), 214–222.
39. Wallace, S.A. and Nierman, A. Using the Java instructional game engine in the classroom. *J. Comput. Small Coll.* 23, 2 (2007), 47–48.
40. Zyla, M. Guest editor’s introduction: Educating the next generation of game developers. *IEEE Computer* 39, 6 (June 2006), 30–34.

**Kelvin Sung** (ksung@u.washington.edu) is a professor in the department of Computing and Software Systems at the University of Washington, Bothell, WA.

© 2009 ACM 0001-0782/09/1200 \$10.00