Introductory Programming Courses and Computer Games

Panelists: Mark Lewis, Trinity University
Scott Leutenegger, University of Denver
Michael Panitz, Cascadia Community College
Kelvin Sung, University of Washington, Bothell (Moderator)
Scott A. Wallace, Washington State University, Vancouver

SUMMARY

Programming games in computer science (CS) classes creates high levels of excitement and motivation [1]. Although there are potential pitfalls [2] and it has been argued that gender biased-games can further alienate under-represented groups, it has also been shown that with careful design and articulation, combining CS classes with games at the introductory level can help recruit and retain students [3]. The focus of this panel is not the debate about whether games should be used in introductory computer science, but rather if they are going to be used what are some possible ways to do so. This panel presents four recent approaches at integrating computer gaming into introductory programming courses. At their core, these approaches can be broadly classified as either: experiences in which students create complete games of their own; or experiences in which students implement fundamental CS concepts to complete "skeleton" games that have been provided by the instructor. These approaches are based on different pedagogical philosophies and implementation platforms, yet all are designed to teach fundamental concepts via programming computer games. In all cases, the panelists will present examples and results from their recent work where CS concepts are learned while programming games.

Faculty members who are interested in finding out more about gaming, or considering/interested-in adapting gaming related approaches/materials in their classes will find this panel especially relevant.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education – *computer science education*.

General Terms: Design, Experimentation.

Keywords:

Introductory programming courses, games, assignments.

Mark Lewis: At Trinity we have developed a 2-D game infrastructure that we use in our second semester course [4]. All the assignments the students do during the semester are parts of a game that they build up to produce final result. The infrastructure itself is extremely flexible, allowing nearly any 2-D game to be implemented. This has been critical as it allows students who don't see themselves as serious gamers to still implement something they have some experience with. We have yet to meet a current college student who has never played some form of arcade style game.

SIGCSE'09, March 3–7, 2009, Chattanooga, Tennessee, USA.
ACM 978-1-60558-183-5/09/03.

The framework has two main goals in this project. It allows the first assignments to have a graphical interface when students haven't yet learned how to do graphics. It also gives the students experience in doing design in the context of an existing framework. Having a single project that spans the entire semester allows students to get a better feel for design as they wind up producing a larger product and get to go back and revisit decisions made early in the semester.

Using a project where students get to select a game and build the entire game with significant flexibility also allows them to express their creativity. This is vital for keeping the interest of many students and showing them the true potential of what we are trying to teach. In the program at Trinity, the use of this project across all section of CS2 has also led to an enhanced sense of community, as it provides a common experience that most viewed as being both educational and enjoyable.

Looking to the future, another advantage of games is likely to come into play: games are one of the few applications that can push modern computers. At Trinity we have been thinking of ways to extend parallelism through the curriculum and the game project will be part of this.

Scott Leutenegger: We believe that allowing students to create a complete self-designed game increases student interest. Further, such an approach encourages alternative game genres. We agree that building part of a predefined game has many advantages, but at the same time argue that taking ownership of the entire game design and outcome greatly motivates students. In order to pull off such an ambitious goal we must focus on more modest 2D games with a tool that does much of the “heavy lifting”. Our approach has been to use Flash/Actionscript [5]. We have now offered introductory programming using Flash/Actionscript three times [6]. Extensive class notes and exercises in the form of a preliminary book are currently available from our web site [7].

Students create simple characters using the Flash drawing environment. Placing characters on the screen and moving them around is made simple using the actionscript MovieClip class. Actionscript also provides an easy to use hitTest() method that does collision detection. Flash also provides easy methods for keyboard control, mouse control, and sound. In the panel presentation we will demonstrate programming assignment solutions that show how elementary programming concepts are taught. We will also show a few final games created by students.

Although we personally feel Flash/Actionscript is an excellent tool/language, we realize there are other pragmatic concerns. First, the high school AP exam is currently in Java and many schools and industry jobs use Java. Second, most computer science college programs use Java, C++, C, or C# as the most common programming languages. Although we feel that transitioning from Actionscript to Java or C++ is not an issue,

some of our students did find this difficult. For these and other reasons, we plan to explore using Java inside of Greenfoot [8] in the upcoming year. Regardless of the exact tool, the concept of Games First, instead of objects first versus procedural first, remains our focus.

Michael Panitz and Kelvin Sung: We believe faculty members should be able to experiment with game-themed programming assignments, examine the results, and gradually adopt suitable results into their classes in order to meet the needs of their students. However, most existing game-themed projects require tight integration with a specific pedagogical philosophy; demand in-depth knowledge of computer gaming or graphics; or require extensive alterations to existing curricula. Based on this, it can be challenging for a faculty member with limited time to experiment with and incrementally improve their existing well-established introductory programming classes with game-themed assignments.

We have developed game-themed programming assignment modules [9, 10] specifically targeted for adoption by CS1/2 faculty who have no gaming or graphics backgrounds. These assignment modules are gender and programming experience *neutral* because students are *not* expected to develop any games from scratch on their own. Instead, each assignment is a skeleton of a game-like application, where students must fill-in the relevant, missing, core CS concepts. Each assignment module is also *self-contained*, so as to facilitate easy experimentation and incremental integration by faculty. Each module includes: a summary of prerequisite knowledge; a list of learning outcomes; a sample pre- and post- test; a sample grading rubric; a sample solution for the instructor; and the assignment skeleton for students. To ensure faculty with no gaming/graphics background can adopt the modules, all needed graphics/gaming code is provided in the assignment skeleton. In addition, a detailed step-by-step tutorial is provided for those faculty interested in the game-specific details of the assignment. Finally, these assignments have been assessed by an external independent faculty member to ensure the technical integrity. In this way, faculty members currently teaching CS1/2 courses can pick and choose from our assignment modules and combine them with their own, existing, non-game assignments.

We have adopted these assignment modules in our own console-based CS1/2 classes where selected console-assignments were replaced by game-themed ones. We will summarize our experience: the extra efforts required, student learning outcomes, and students' excitement and satisfaction.

Scott Wallace: I believe that games provide a rich and visual environment for students to explore a variety of important Computer Science topic areas and skills. The Java Instructional Game (JIG) Project [11, 12] was established as a collaboration between Washington State University Vancouver and the University of Puget Sound to provide support for resource limited colleges and universities that want to bring game related projects or courses into their CS curriculum. The goal of the JIG Project is not to train computer game developers. Rather, the JIG team is interested in training *excellent* computer scientists and believes that games offer a good vehicle to achieve this goal.

For the members of the JIG team, Java is a natural choice for such software since it is a leading choice, if not the leading choice, for first programming languages and is often used throughout the CS curriculum. Furthermore, because Java is also used on the Advanced Placement exam for high school students, many students exposed to programming before college will likely end up learning Java.

To help bring games into the Java classroom, the JIG team has created a software foundation (game engine) suited for students at all four years of study. This engine enables instructors to offer dedicated game design courses and also provides the infrastructure for curricular modules. Our curricular modules enable instructors to add individual game projects to traditional CS courses in the same spirit as Panitz and Sung's project. They are designed as short (one or two week) stand-alone projects and target concepts from CS1/2 (e.g., recursion, flow control, and data structures) to the senior level (e.g., computational geometry, and graph algorithms). By providing the game skeleton, our modules attempt to ensure that the student's effort is focused on the learning objectives of the project, and not devoted to learning how to use the JIG game engine or learning to program complete games from scratch.

IMPORTANCE: The intended audience of this panel is anyone considering using a game approach in an introductory computer science class. The panel is especially timely as more computer science departments are adding tracks or majors in game development or at least considering the use of games within the existing curriculum to attract more students to the major. The audience will come away with an overview of possibilities and arguments for and against specific approaches. This panel is both timely and important.

REFERENCES

[1] U. Wolz, T. Barnes, I. Parberry and M. Wick, "*Digital gaming as a vehicle for learning*," in SIGCSE '06, PP. 394-395, 2006.

[2] S. Hallern, B. Ladd, S. Leutenegger, J. Nordlinger, J. Paul, H. Wallker, and C. Zander, "*Games: good/evil*," in SIGCSE'08, PP. 219-220.

[3] M. Guzdial, E. Soloway, "*Teaching the Nintendo generation to program*," Communications of the ACM, 45(4), pp. 17–21, 2005.

[4] M. C. Lewis and B. Massingill, "*Graphical game development in cs2: a flexible infrastructure for a semester long project*," in SIGCSE'06, PP. 505-509, 2006.

[5] S. Crawford and E. Boese, "*Actionscript: a gentle introduction to programming*," J. Comput. Small Coll., 21(3):156-168, 2006.

[6] Course web site: www.cs.du.edu/~leut/1671/06_Fall, 2007.

[7] S. Leutenegger, J. Edgington, "*A Games First Approach To Teaching Introductory Programming*", SIGCSE'07, PP. 115-118.

[8] P. Henriksen, and M. Kulling, "*Greenfoot: Combining object visualisation with interaction*", in Companion to OOPSLA conference, PP 73-82, 2004.

[9] Games-themed Introductory Programming Project web-site, 2008, http://depts.washington.edu/cmmr/Research/XNA_Games/.

[10] K. Sung, M. Panitz, B. Rosenberg, R. Anderson , "*CS1/2 Game-Themed Programming Assignments for Faculty*," Journal of Game Development, Vol. 3, Issue 2, March 2008, PP. 27-47.

[11] S. Wallace, A. Nierman. "*Addressing the need for a Java based game curriculum*," J. Comput. Small Coll. 22(2), 20-26. 2006.

[12] S. Wallace, JIG—Java Instructional Gaming, http://ai.vancouver.wsu.edu/jig/, retrieved July 3, 2008.