Game-Themed Programming Assignment Modules: A Pathway for Gradual Integration of Gaming Context Into Existing Introductory Programming Courses

Kelvin Sung, Cinnamon Hillyard, Robin Lynn Angotti, Michael W. Panitz, David S. Goldstein, and John Nordlinger

***Abstract*—Despite the proven success of using computer video games as a context for teaching introductory programming (CS1/2) courses, barriers including the lack of adoptable materials, required background expertise (in graphics/games), and institutional acceptance still prevent interested faculty members from experimenting with this approach. Game-themed programming assignment (GTA) modules are designed specifically for these faculty members. The GTA modules are independent, and each is a self-contained game-like programming assignment that challenges students on concepts pertaining to a specific curriculum topic area. A faculty member can selectively pick and choose a subset of GTA modules to experiment with and gradually adopt the materials in his or her own classes. Each GTA module also includes a step-by-step tutorial guide that supports and encourages interested faculty to develop their own expertise and game-themed materials. This paper begins with a survey of previous results. Based on this survey, the paper summarizes the important considerations when designing materials for selective adoption. The paper then describes the design, implementation, and assessment of the GTA modules. The results from ongoing GTA workshops for CS1/2 faculty members and from a yearlong project in adopting the GTA modules in classes are then presented. In this case, the collected results verified that introductory programming concepts can be examined, practiced, and learned by means of GTA modules when neither the faculty nor the students involved have backgrounds in graphics or games. More importantly, these results demonstrated that it is straightforward to *blend* the GTA modules into existing classes with minimum alterations. In these ways, the GTA modules are excellent catalysts enabling faculty to begin exploring and developing their own expertise and materials to teach with games.**

***Index Terms*—Adaptation, assessment, assignments, CS1/2, games.**

Manuscript received November 02, 2009; revised May 03, 2010; accepted July 12, 2010. This work is supported in part by Microsoft External Research under the Computer Gaming Curriculum in Computer Science RFP, Awards 15871 and 16531.

K. Sung is with the Department of Computing and Software Systems, University of Washington Bothell, Bothell, WA 98011 USA (e-mail: ksung@uwb.edu).

C. Hillyard is with the Department of Interdisciplinary Arts and Sciences, University of Washington Bothell, Bothell, WA 98011 USA (e-mail: chillyard@uwb.edu).

R. L. Angotti is with the Department of Education, University of Washington Bothell, Bothell, WA 98011 USA (e-mail: rrider@uwb.edu).

M. W. Panitz is with the Software Programming Department, Cascadia Community College, Bothell, WA 98011 USA (e-mail: mpanitz@cascadia.ctc.edu).

D. S. Goldstein is with the Teaching and Learning Center, University of Washington Bothell, Bothell, WA 98011 USA (e-mail: dgoldstein@uwb.edu).

J. Nordlinger is with Microsoft External Research, Redmond, WA 98052 USA (e-mail: johnnord@microsoft.com).

Digital Object Identifier 10.1109/TE.2010.2064315

I. Introduction

TEACHING computer science (CS) concepts based on programming interactive graphical games motivates and engages students while accomplishing desired student learning outcomes [1], [2]. When properly integrated in introductory CS courses (CS1/2), these approaches build excitement and enthusiasm for the discipline and attract a bright new generation of students early in their academic careers (e.g., [3] and [4]). However, as a new approach, interested faculty need support to explore and experiment with teaching CS1/2 courses based on interactive graphical games. When designing support for these faculty members, there are two important areas of consideration: faculty background and institutional oversight.

As discussed in the next section, most of the existing results in integrating computer gaming in CS courses involve exploratory projects by faculty members with expertise in computer graphics and gaming. With few exceptions, these projects are often *student-centric*, where the main goals of study are student engagement and various student learning outcomes. Adaptability and general applicability of the resulting materials are usually not main concerns. For faculty members without computer graphics or gaming backgrounds, it can be especially challenging to take advantage of these results.

When considering experimentation with CS1/2 courses, it is important to be mindful of institutional oversight procedures. Though becoming less controversial in recent years, many CS educators continue to be unsure about integrating gaming in formal educational settings (e.g., [5] and [6]). It is a challenge for departmental committees to arrive at a consensus for significant modifications to CS1/2 courses, especially if the modifications involve computer games.

The CS1/2 game-themed programming assignment (GTA) modules are targeted specifically for adoption in existing introductory programming classes.[1] These assignment modules are self-contained so that faculty with no background in graphics/gaming can select a subset of the modules to combine with existing assignments in current classes. The assignment modules are limited in curriculum scope to facilitate selective experimentation by individuals. Finally, the assignment modules include detailed implementation tutorials to assist interested fac-

[1]All modules and related materials are freely available from the GTA project Web site: http://depts.washington.edu/cmmr/Research/XNA_Games.

ulty in developing game-themed programming assignments. In this way, as the GTA modules are being adopted, faculty develop expertise and can collect and demonstrate results to assist the decision-making process of institutional oversight committees.

This paper presents the results from the entire GTA project: the design and implementation of the modules [7], the assessment of the academic merits of the materials [8], [9], and finally the encouraging results from the ongoing GTA workshops for CS1/2 faculty and a yearlong study of classroom adoptions [10]. In the next section, the paper begins with a survey of related work done in the area. It is important to note that there is nothing magical about teaching with games. As highlighted by Bayliss [11], faculty buy-in and experience are some of the most important factors in realizing the potential student engagement of a game-themed teaching approach. GTA modules were designed to help faculty members develop expertise in the area. The objective of this pilot study was to verify that GTA modules could be adopted with minimal extra effort by faculty members with no background in graphics and games and with little change to a course syllabus. The primary goal was to verify that the GTA modules "*do no harm*" to student learning while faculty members incrementally experiment with, and develop experience and expertise in, game-themed context.

The GTA modules are simple "real-time interactive graphics programs." Strictly speaking, these programs do not qualify as "games" because they have unknown entertainment value. However, in the current implementation, since the programs run on both PCs and the XBox 360 gaming platform, the term "game-theme" is used. In this paper, "console-based" or "console assignments" are used to refer to conventional programming assignments that are designed around keyboard and (ASCII) character-driven console monitors.

II. Background

The GTA modules are designed for students to learn abstract CS concepts by programming and/or examining "real-world, game-like" applications. Relating abstract principles to real-world experience has become increasingly prominent in mathematics, science, and technology education. For example, the "*Calculus Reform*" movement of the 1990s [12] included both pedagogical changes and foci on real-world problems, while the Carl Wieman Science Education Initiative at the University of British Columbia, Vancouver, Canada, has redesigned its freshmen introductory physics course to present standard introductory materials in connection with real-world situations [13]. In the CS education arena, the *Media Computation* of the Georgia Institute of Technology, Atlanta, [14] is an excellent example where foundational programming concepts are presented in the context of popular digital multimedia applications. This *Contextualized Computing Education* [15] is an ongoing effort, and "*interactive games*," being one of the most familiar application areas for students, is a context favored by many CS educators.

There are many types of "games" that are suitable for teaching CS subjects, including many noncomputer games [16] or games that are based on dedicated devices (e.g., LEGO robots [17]). The focus of this work is on interactive graphical computer games. As discussed by Sung [18], recent work in this area can be classified into *game development* [19], [20] where students learn about building games, *game programming* [1] where students study algorithms related to games, and *game client* [21], [22] where students learn about CS concepts via games. Integrating games into CS1/2 classes belongs to the *game client* category because the objective for students in these classes is to understand abstract programming concepts and not to learn about building games.

Existing work on presenting CS1/2 concepts in the context of computer games can be broadly categorized into three approaches [18]: 1) little or no games programming [23], [24] where students learn by *playing* custom games; 2) per-assignment games development [3], [7], [25] where individual programming assignments are computer games designed around technical topics being studied; 3) extensive game development where faculty and students work with custom games engines [26], [27], specialized programming language [28], environments [29], or specific curricula [4], and so on. All three approaches reported resounding success with drastically increased enrollments and student success [3], [4], [28]. Based on these results, it is well recognized that integrating computer gaming into introductory computer science (CS1/2) courses is a promising strategy for recruiting and retaining potential students.

As discussed by Levy and Ben-Ari [30] and Ni [31], issues that faculty consider when examining new and innovative teaching materials for adoption include preparation time, material contents, departmental oversight committee, and compatibility of programming languages. Adopting/adapting results from an extensive games development approach requires a significant investment of time, which includes the need for faculty to understand a game engine or significantly rework an existing curriculum. This work-intensive adoption/adaptation is not suitable for limited-scope investigation. Projects and results from the per-assignment games development approach are typically from faculty members with expertise in graphics/games and are "*student-centric*," where the main goals of study are student engagement and various learning outcomes. Most instructors of CS1/2 courses do not have the time or expertise to adapt and/or implement these projects in their courses.

The GTA modules are "*student-centric*" because they are assignments that allow students to practice CS concepts in context. More importantly, these modules are "*faculty-centric*" because they are the stepping stones for faculty to begin experimenting with a promising new approach to teaching CS1/2 courses.

III. Implementation Considerations and Details

The above survey implies that in order to facilitate selective adoption and limited curriculum scope (e.g., per-assignment) experimentations by faculty members with no relevant background, the GTA modules must include all relevant materials and be self-contained. The assignments must be simple interactive graphical applications that assist students to practice relevant programming concepts. At the same time, the GTA modules should be interchangeable with those from typical CS1/2 courses.

A. Choice of Technical Topic Areas

It is important to differentiate *technical topic areas* (e.g., linked lists) from individual assignments. For example, one can design a console-based assignment to manipulate a linked list of numbers, or one can design a game-themed assignment where the in-game logic is based on linked lists. With careful design, both assignments would challenge students in implementing the basic linked list functionality. In this way, the two assignments can be technically equivalent, and yet one is a *traditional* console-based assignment while the other is a game-themed assignment.

The topics for GTA modules were chosen using a "reverse adoption" strategy; the technical topics for the game-themed assignments were adopted based on the console-based assignments in existing CS1/2 courses [32]. There are several advantages to this approach.

1) Existing CS1/2 courses are well established with many successful alumni in advanced CS courses and in industry. This success justifies the selected technical topic areas.
2) Assignments with identical technical topic areas imply they can be interchanged. This offers a vehicle for the subsequent phase of this project, where corresponding assignments can be replaced and the effects studied.
3) The console-based assignments are included as part of the assignment modules. In this way, each assignment module addresses a well-defined technical topic area and has two versions: a console-based version and a game-themed version. The console-based version of the assignment is *conventional* and does not necessarily include interactivity. This version serves as an excellent and familiar reference for faculty members unfamiliar with game programming.

Seven GTA modules have been implemented. In the subsequent phase of this study, these modules replaced the corresponding console assignments in existing CS1/2 courses. As will be detailed in next section, the current seven GTA modules cover topic areas that include integer division and the modulus operator, random number generation, single-dimensional arrays of object references, 2-D arrays, class hierarchy/inheritance, linked lists and queues, and binary search trees. Section IV describes the assessment procedure that ensures that the console-based and game-themed versions of the assignment are technically equivalent.

B. Contents of an Assignment Module

Each assignment module is designed to be self-contained and consists of materials for both the faculty and the students.

For the faculty, each module includes:

- a summary page describing the assignment, including prerequisite knowledge, and a list of expected student learning outcomes;
- a sample pre- and post-test;
- a sample solution for both the console-based and game-theme versions;
- a sample grading rubric for each version;
- a list of frequently asked questions;
- an implementation tutorial.

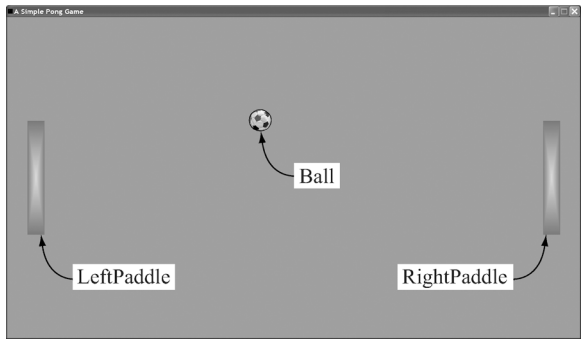

Fig. 1. Simple Pong game.

The implementation tutorial is a step-by-step guide that explains the implementation of the game-themed assignment. This tutorial is intended to help interested faculty better understand how to create their own game-themed assignment using the library that was developed to support this project.

For the students, each module includes:

- a description of the assignment;
- a skeleton starter project for both the console-based and game-themed versions.

The game-themed starter project is a game-like application where all necessary graphics and user interactions functionality are provided. Students work with the starter project to fill in the relevant core CS concepts to complete each assignment, without having to know anything about computer graphics and games.

C. Implementation Platform

There has been work done to integrate the concepts and tools involved in building interactive graphical computer games into introductory programming courses, including event handling [33], graphical user interfaces (GUIs) [34], and graphical application programming interfaces (APIs) [35]. To support faculty without a computer graphics and games background, these aspects of game programming were hidden. This provided a platform that transparently integrated all of the above tools so that faculty and students did not need to be aware of their existence.

The C# programming language and the Microsoft XNA framework [36] were chosen for this platform. This choice was governed primarily by the fact that the C#, XNA, and Microsoft's Game Studio Express combination is the only freely available solution that provides seamless integration of the development environment, programming language, GUI API, and graphics API.

D. Simple Game-Themed Example: A Pong Game

This section uses a simple "Pong game" example to illustrate game-themed application development. In this application, the *Ball* travels with a random velocity and bounces within the application bounds, and the players control the vertical positions of the *LeftPaddle* and *RightPaddle* to collide with and bounce the *Ball* (see Fig. 1). Fig. 2 is the implementation for this simple

```
public class PongGame : XNACS1Base {                    // Label A: All examples subclass from XNACS1Base
    XNACS1Circle      Ball;                                 //   A1: The pong ball is a XGCS1 circle
    XNACS1Rectangle LeftPaddle, RightPaddle;                //   A2: The paddles are XGCS1 rectangles
    protected override void InitializeWorld() {         // Label B: Called once at the beginning
        World.SetWorldCoordinate(...);                      //   Define the pong field dimension
        Ball = new XNACS1Circle(...);                       //   The pong ball: size, location, and velocity
        LeftPaddle = new XNACS1Rectangle(...);              //   The paddles: size and location
        RightPaddle = new XNACS1Rectangle(...)
    }
    protected override void UpdateWorld() {             // Label C: Called once every 25 milli-seconds
        LeftPaddle.CenterY += GamePad.ThumbSticks.LeftY     //   C1: Update paddle Y-locations
        RightPaddle.CenterY += GamePad.ThumbSticks.RightY   //   with gamepad thumbstick
        BoundCollideStatus status = World.ClampAtWorldBound(Ball); // C2: Collide the ball with world bounds
        switch (status) {                                   //   flip the velocity-y component
            case BoundCollideStatus.CollideBottom:          //   if collided with the top/bottom bounds
            case BoundCollideStatus.CollideTop:
                    Ball.VelocityY *= -1;
                    break;
             // ... cases for Left/Right bounds and flip in Velocity-X ...
        }
        if (LeftPaddle.Collided(Ball))                  // C3: Bounce the pong ball off the paddles
            Ball.VelocityX *= -1;                           //   collide the ball with the paddles
        else if (RightPaddle.Collided(Ball))            //   flip the velocity-x component if
            Ball.VelocityX *= -1;                           //   there is a collision
    }
}
```

Fig. 2. Simple Pong game with XGCS1 library.

application based on XGCS1, a simple 2-D library designed specifically to support GTA modules.

Label A in Fig. 2 explains that all XGCS1 applications must be subclasses of the *XNACS1Base* class. It is this class that abstracts the *Model-View-Controller* architecture into the *InitializeWorld()* and *UpdateWorld()* two-function protocol. The *InitializeWorld()* function at **Label B** is called once at the beginning of the application and, in this case, defines the Pong game dimension and instantiates the ball and paddles defined at *A1* and *A2*. The XGCS1 library automatically draws all geometries and moves the *Ball* object in the application window. The *UpdateWorld()* function at **Label C** is called periodically at a rate of 40 times a second. In this case, each update allows the user to control the y-positions of the paddles (at *C1*), bound the ball within the application window (at *C2*), and bounce the ball off each paddle (at *C3*). This simple game-themed example demonstrates the following with the XGCS1 support.

- Interactive, graphical, game-themed application can be simple and intuitive as in, for example, the parallel between the descriptive *narration* of the application and the actual logic control code at label positions C1, C2, and C3.
- The implementation is independent of details of drawing in computer graphics and complex object interactions in game development as in, for example, the absence of programming code to draw, manipulate the *Ball* movements, and computing spatial collisions between geometric objects.

The fundamental programming logic flow is prominent in the implementation. In Fig. 2, for example, the statements at *C2* and *C3* can be used as interesting examples for demonstrating different constructs of conditional statements. Since this application is game-themed, it is straightforward for students to change the conditional constructs (e.g., replace the switch statement with an if–then–else) and then interactively examine the effects of their changes. In this simple example, a GTA module can be defined around conditional constructs and, for example, challenge students to support the implementation of barrier blocks between the two paddles or determine winning conditions.

IV. Results: The GTA Modules

The GTA modules are simple "interactive graphics applications" where the main goal of the assignments is to reinforce technical concepts rather than for students to enjoy the *fun-ness* of the game. This section describes the developed modules focusing on the technical topic areas covered by each of the console-based and game-themed versions of the assignments. All materials presented are available online at [37].

Assignment One: Integer Arithmetic. This assignment is designed to be the first CS1 assignment. Fig. 3 shows the game-themed version of the assignment. In this case, the user can control the horizontal position of the chameleon-circle. The color bands in the background are vertical rectangles with repeating colors. Given this skeleton application, students must program proper integer arithmetic to control the color of the chameleon-circle, such that as it is moved horizontally, its color always reflects that of the rectangle underneath it. The console-based version of the assignment is a simple character-based flashcard quiz program.

Assignment Two: Random Number Generation and Operators. Fig. 4 shows the game-themed version, where the user controls the hero insect catcher to net randomly generated insects. Supplied with all the graphics and interaction functionality, students must implement random number generation and maintain proper accumulated results, success ratio, etc. The console-based version of this assignment is on Monte Carlo integra-

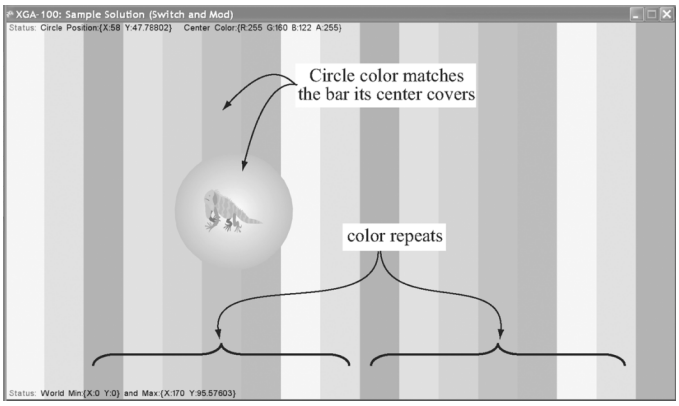

Fig. 3. Traveling chameleon.

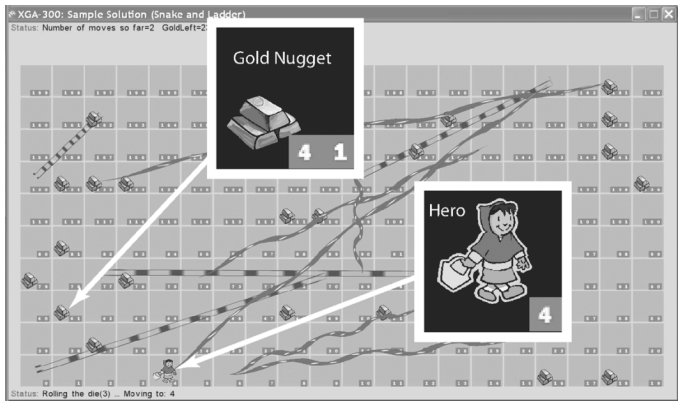

Fig. 5. Snakes and Ladders game.

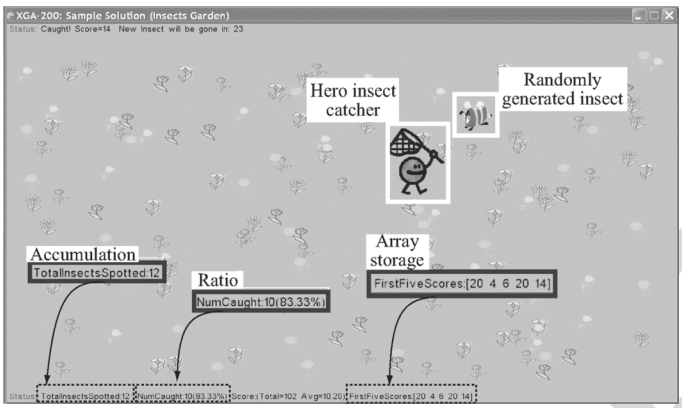

Fig. 4. Insects Garden game.

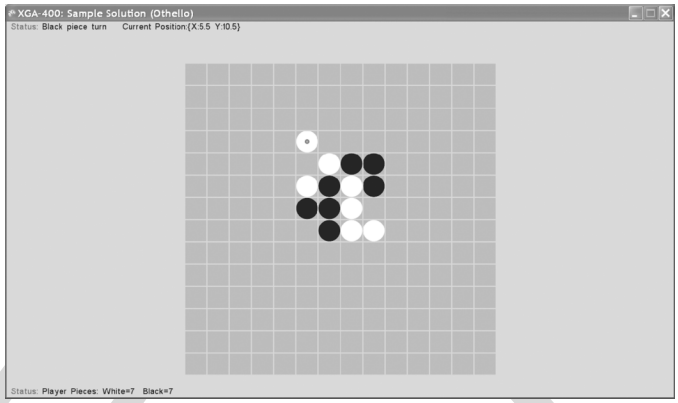

Fig. 6. Othello game.

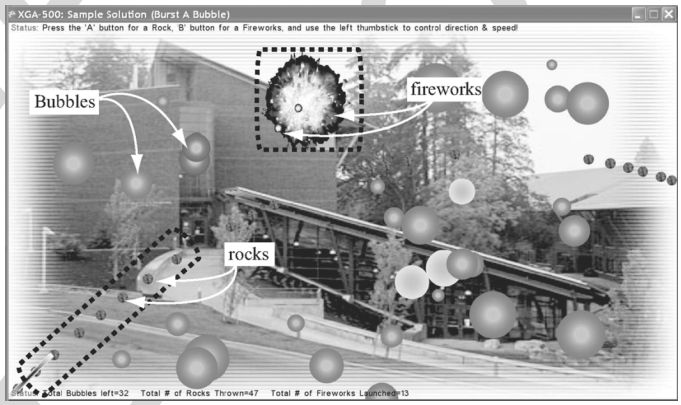

Fig. 7. Burst a Bubble game.

tion, where students must approximate the area of a circle based on randomly generated sample positions.

Assignment Three: Single-Dimension Arrays of Object References. Fig. 5 shows the game-themed version, a variation of the classic *Snakes and Ladders* game. In this case, the hero can pick up *gold nuggets*, and the user can dynamically create additional *snakes* and *ladders* at run-time. The game board is implemented as a single-dimensional array of game cells, where each game cell can either be empty (i.e., *null*) or contain a nugget/snake/ladder. Students' code must properly access and allocate new game cells for this single-dimensional array to support the above functionality. In the console-based version, students complete a program that maintains a partially filled periodic table of elements. The periodic table is implemented as a single-dimensional array that contains either null or a reference to an element object. The student is responsible for filling in code that creates new elements, edits existing elements, or prints out element objects.

Assignment Four: Two-Dimensional Arrays. Fig. 6 shows the game-themed version, the classic two-person *Othello* game. In this case, an empty two-dimensional array representing the game board is provided. The students' code must work with this array to enforce game play logic where the players are only allowed to place new game pieces in valid locations, and the color of relevant game pieces on the game board must be flipped after each successful play. This is the only assignment where the game-themed and console-based assignments are basically identical. The only difference in the two is that, in the console-based version, ASCII characters draw the game board on a character-display window.

Assignment Five: Class Hierarchy and Inheritance. Fig. 7 shows the game-themed version where the user launches *rocks* and *fireworks* from the lower left corner to burst bubbles that are randomly distributed in the application window. In this assignment, students must understand the given *Projectile* class and create a subclass to implement a *Firework* class. The console-based version is based on the same idea, except that the program is limited to turn-by-turn processing, and feedback to the user is in the form of ASCII text. Both the console-based and

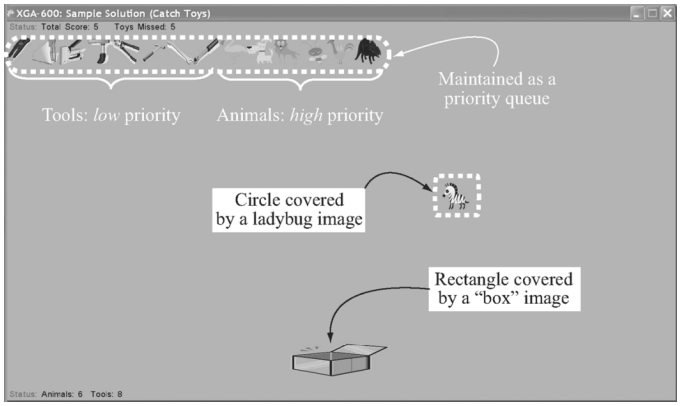

Fig. 8. Catch a Toy game.

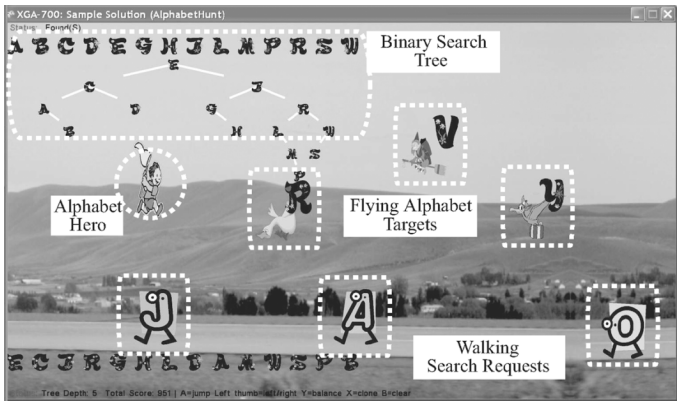

Fig. 9. Alphabet Hunt game.

game-themed versions of this assignment expose the students to the challenges of adding a new class to an existing, nontrivial code base.

Assignment Six: Linked Lists and Queues. Fig. 8 shows the game-themed version in which the player can insert either high-priority toys (animals) or low-priority toys (tools) into the overall queue located at the top of the window. The game continuously dequeues and drops the oldest toy from the priority queue. The user moves the box to catch the dropping toy. Students must implement the overall queue class, typically by maintaining items in one of two different (linked list-based) queue objects, based on each toy's priority. The game interfaces with the overall queue class and draws all the items in a single row so it appears that the toys are all in a single queue, regardless of student implementation. The console-based version of this assignment is a text-based "help desk" application. The user can enter high- or low-priority requests to be enqueued to the front/back of the queue, respectively. Similar to the game-themed version, the retrieval of requests is a simple dequeue operation on the queues. The skeleton starter projects for both versions contain all necessary I/O functionality: graphics/GUI for the game-themed version, and character I/O for the console-based version. In both cases, students only need to implement the linked list queue and the priority queue.

Assignment Seven: Binary Search Trees (BSTs). Fig. 9 shows the game-themed version where, in a side-scrolling game, the *alphabet hero* must leap to collect *flying* alphabet targets and process the *walking* alphabet search requests. The students' code must implement a BST to store the collected target letters and search the BST upon encountering *walking requests*. The drawing of the BST in the upper left is supported via an abstract base class in the form of a dynamically linked library (DLL) where the implementation details are hidden. The console-based version of the assignment is a character-driven BST implementation test program where outputs are printed based on simple commands (e.g., add, find).

V. Assessing the GTA Modules

The GTA modules are designed for students to practice and learn fundamental concepts in programming. It is important to evaluate independently the academic content of the materials. Additionally, it is important to verify the technical equivalence between the console-based and game-themed assignments in each module.

A. Independent Reviewer

Prof. Ruth Anderson is the independent external reviewer. She is an experienced instructor who has taught CS1/2 courses many times at multiple institutions and in a variety of programming languages, has won multiple teaching awards,[2] and is active in CS education research (e.g., [38] and [39]). In addition to these excellent credentials, Prof. Anderson is perfectly suited for evaluating the materials because she has never taught a graphics or gaming course and has limited experience with GUI programming. Before this project, Prof. Anderson did not know anyone on the project team.

B. Procedure

During the project, in-person or verbal communications were avoided both to maintain impartiality and to simulate the sort of investigations that might be made by curious faculty. The assessment of the assignments was conducted across the project Web site [37], where newly released materials were downloaded, examined, and tested by Prof. Anderson. Feedback was provided via a custom assignment evaluation form.

The assignment evaluation form is designed to collect both formative feedback and quantitative scores [40]. Each assignment module is assessed in two areas:

1) *Quality of the assignment*: assesses the merit, the technical equivalence between the console-based and game-themed assignments, and the supporting materials (e.g., pre/post-test);
2) *Potential for adoption*: assesses the factors independent of the quality of assignments that may prevent adoption (e.g., programming language used).

C. Assessment Results

Based on the review feedback, the assignments have been well received overall. It is believed that because the assignments were adopted based on existing CS1/2 classes, technical merits were never an issue. Prof. Anderson agrees that the assignments are appropriate for typical CS1/2 courses. Assignments with low

[2]ACM Faculty Award, voted best teacher by Department of Computer Science students, University of Virginia, Charlottesville, 2004.

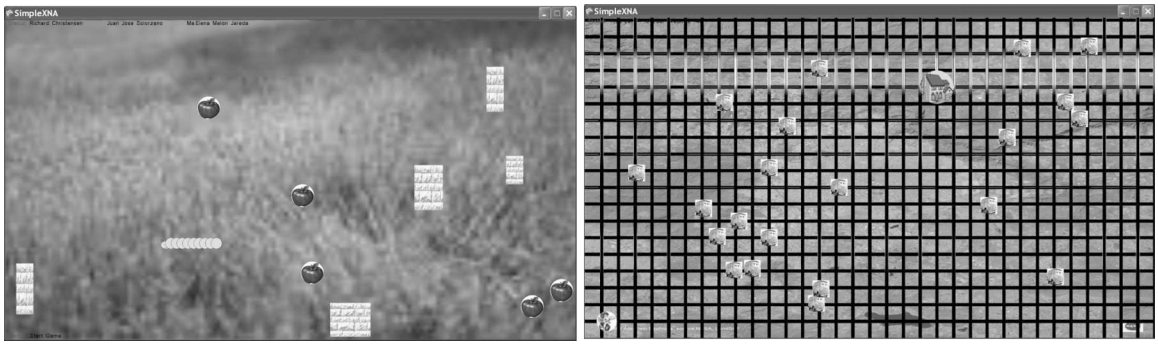

Fig. 10. Games by faculty with no prior background.

quality of assignment scores were revised and reassessed. This process continued until the formative comments were positive and the numeric scores were above 4 (out of 5).

As expected, all of the low scores were caused by game-themed assignments. This was typically due to the following.

1) Differences in difficulty: Initial attempts at game-themed assignments often resulted in highly difficult, complex, or intimidating programs. Based on feedback received, assignments have been adjusted accordingly.
2) Inappropriate use of concept: Designing assignments around negative results from game play is a bad idea. For example, a linked list structure might be used for tracking when the player has been unsuccessful at a given task. In this case, in order to test the linked list, students must deliberately be unsuccessful at playing the game. This can take the fun out of the assignment.
3) Deficiency in support: The development team often overlooked important details. For example, in the beginning, a specialized hardware controller was the only way to control a game. Based on the feedback, this problem was remedied with a keyboard-based software object simulation.

Because Java is the language of choice at Prof. Anderson's institution, consistently lower scores for potential for adoption were received. The developers are fully aware that the language issue must be addressed for wide adoption of results. With experience building these assignments, and understanding the important attributes of the library, the developers are investigating possibilities for porting the results to other environments.

VI. GTA Workshops for Faculty

Outside of classrooms, GTA-related workshops have been offered for interested faculty members at regional [41] and national [42] conferences and at institutions internationally [43], [44].[3] These workshops guided faculty to develop game-themed applications based on tutorials from the GTA modules. The images in Fig. 10 are screen shots of a "*Worm-like*" and a "*Pizza Delivery*" game resulting from the second day of a multiday version of the workshops [43], [44]. Although relatively simple, these games were designed and developed in a matter of hours by CS1/2 faculty members with no prior background in graphics/games.

The workshops have received overwhelmingly positive feedback from the participants. For example, written feedback on "*the appropriateness of material difficulty*" included: "*I found the materials challenging but able to comprehend*," and "*good balance of complexity and new materials*." The written feedback on "*the presented materials will help me develop game-themed applications*" included: "*For sure!*" and "*In this environment [GTA], YES!*" The results and feedback from these workshops showed that although they found the GTA materials to be nontrivial, faculty participants with no prior expertise in graphics/games were able to comprehend and begin developing game-like applications within a matter of hours.

VII. Classroom Assessment Tools and Procedures

As catalysts for faculty development, it is important to verify that the GTA modules "*do no harm*" to student learning. It is essential for faculty members to have the reassurance of consistent student learning outcomes when experimenting with this potentially powerful approach. Instruments for understanding the effectiveness of new teaching materials include analyzing quantitative exam scores, qualitative evaluation of projects, student opinion polls, and success rates [45], [46]. These instruments are employed to assess student learning outcomes and perceptions when the GTA modules are adopted in existing classes.

Based on the positive feedback and promising outcomes from the GTA workshops, in-classroom adoption of GTA modules began. Because the modules are designed as catalysts for faculty development, faculty members needed to have the reassurance of consistent student learning outcomes when experimenting this potentially powerful approach to teaching. This project utilized exam scores, qualitative evaluation of projects, a student attitude survey, and success rates [45], [46] to assess student learning outcomes and perceptions when the GTA modules were adopted in existing classes.

For long-term effects, new teaching materials are tracked over multiple semesters of the same course via course enrollments [47] or continual student successes [3]. In the case of GTA modules, one goal was to demonstrate the feasibility of simple replacement of selective assignments in existing classes. The same classes were followed over different semesters where different GTA modules were adopted. The assessment procedure was designed around existing *console-based* CS1/2 classes.[4] These were well-established courses producing many successful alumni in advanced CS courses and in the industry [32].

[3]Refer to http://faculty.washington.edu/ksung for workshop lecture notes.

[4]The involved instructor has no background in graphics/games.

TABLE I
[Table cannot run in middle of text. Provide caption for table.]

	GTA	Console
CS1	72% to 76%	65%
CS2	86%	79%

This study was designed around the adoption of six of the GTA modules: four for CS1, and two for CS2. The game-themed versions of these assignments were integrated into existing CS1/2 courses over three academic quarters. Two existing CS1/2 courses were offered without modification and served as control groups. In the experimental classes, two of the four existing console assignments were replaced with GTAs. Each GTA course consisted of a mixture of two existing console assignments and two GTA modules, in combinations that varied from class to class. This verified that a faculty member could select and replace some or all existing assignments with GTAs. To minimize variation between students in the classes, the modifications to the courses were not advertised, so students did not know they were registering for a class that would use the GTAs.

VIII. Classroom Adoption Results and Analysis

Throughout the experiment, the instructor *avoided* any extra effort when adopting the GTA modules. For example, no lecture time was spent covering graphics or game aspects of the GTAs. In all cases, the *exact* same assessment instruments were administered under similar conditions for both GTA and console courses.

1) Success Rates: The success rates of these CS1/2 classes have fluctuated between 65% to 85% historically. With this perspective in mind, analysis began by examining the overall success rates of all the classes. The percentage *passing* in each class is given in Table I. The percentages of GTA classes were higher than the console classes. Caution should be used when examining these figures since they are well within the historic ranges. The analysis includes all of the students in the participating classes. Because of the small number of female students, to ensure strictest anonymity, the results were not disaggregated by gender.

2) Assignment Scores: Fig. 11 plots the average scores for all of the six assignments.[5] The left bar above each assignment displays the results from GTA, while the right is from the console assignment. The scores from GTA were consistently better than the corresponding console version. From the written feedback (detailed later), it was clear that students spent more time *playing* with the GTA assignments, which resulted in their having a smaller number of errors in their final submissions. In addition, there were interesting observations of *trend* in the score differences. At the very beginning of CS1 when the assignment was trivial (CS1-A, simple arithmetic), the difference between GTA and console was small. By the end of CS1, the assignments became more challenging (CS1-D, 2-D array), and the difference in scores also became more prominent (89% versus 58%). As is widely recognized [48], interactive graphical feedback encourages experimentation, and this trend in assignment score differences reflected students' further engagement with the assignment. However, this increase in score differences became less prominent by CS2 assignments (88% versus 81% for CS2E, and 82% versus 77% for CS2F). As observed by Guzdial [15], CS2 students are more experienced and often prefer assignments without elaborate setups. Results verified that as students became more proficient, the advantage of interactive graphical feedback diminished.

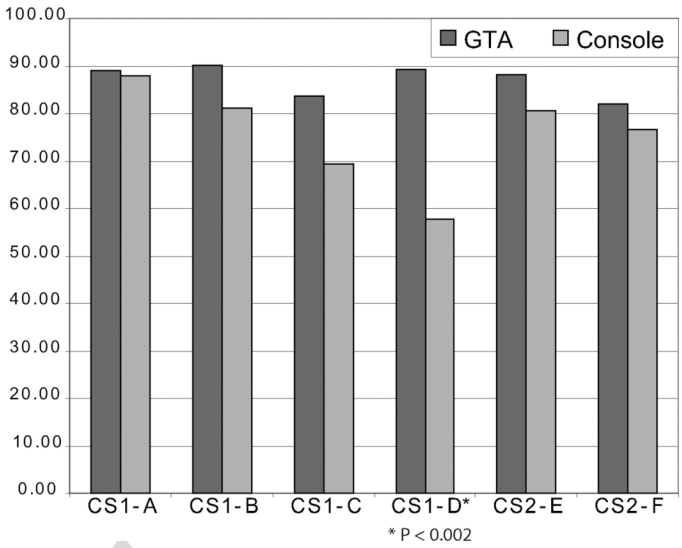

Fig. 11. Average assignment results.

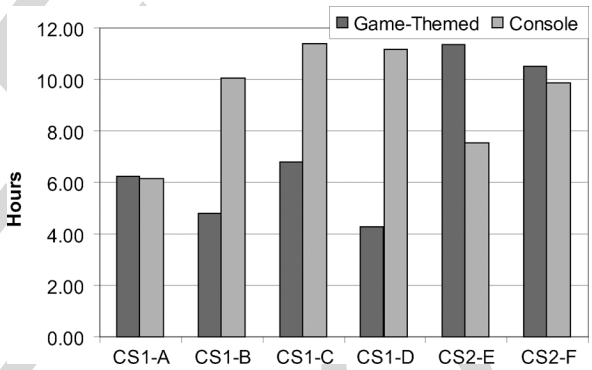

Fig. 12. Self-reported time spent.

3) Self-Reported Time Spent on Assignments: Fig. 12 shows students' self-reported time spent on the assignments. It was interesting to note when examining the time spent on the four CS1 assignments that students reported more than twice the amount of time spent on CS1-B, -C, and -D console assignments. From the average scores and success rates, it was clear that students were learning comparable material. These large discrepancies seemed to suggest *game-themed* assignments were much more efficient learning tools. While it may be true that visual feedback is an important learning tool, it was not found that it alone could accomplish these results. From the written feedback (e.g., "*counting only the time I actually 'worked' on and not 'played' with it*"), it appeared that some students discounted the time they

[5]Note that each assignment has GTA and console versions. For example, the GTA version of *CS1-A* was offered during Spring, while the corresponding console version was offered in the control course, Winter.

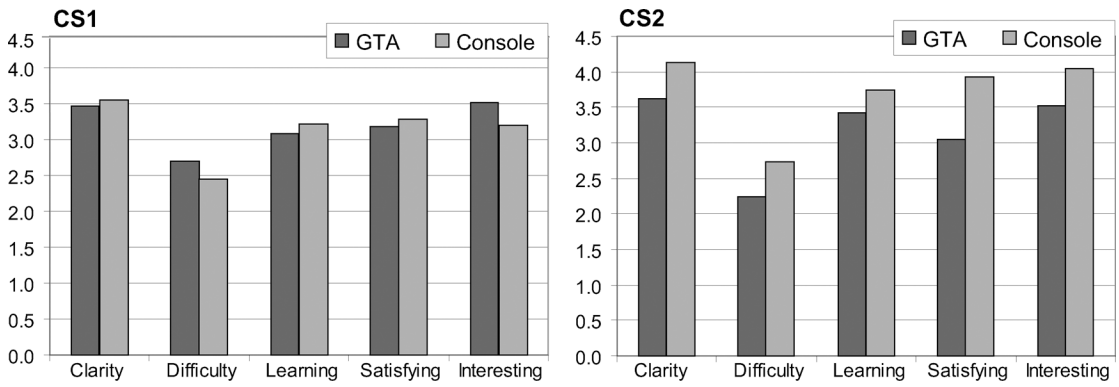

Fig. 13. Post-assignment surveys: (Left) CS1 and (Right) CS2.

CS1

		Like CS	Career	Coding	Prepared	Difficulty	Scope
GTA (N=22)	Pre	4.50	4.64	3.91	2.90	2.61	3.41
	Post	4.50	4.68	3.68	3.86	3.41	3.77
	Change	**0.00**	**0.05**	**-0.23**	**0.96***	**0.80**	**0.36a**
Console (N=10)	Pre	4.17	4.73	3.65	3.04	3.09	3.41
	Post	4.40	4.70	3.20	3.10	3.80	3.50
	Change	**0.23**	**-0.03**	**-0.45**	**0.06**	**0.71**	**0.09**

*P < 0.02

CS2

		Like CS	Career	Coding	Prepared	Difficulty	Scope
GTA (N=18)	Pre	4.64	4.82	3.86	4.27	2.64	3.59
	Post	4.61	4.94	3.89	4.56	3.06	4.11
	Change	**-0.03**	**0.13**	**0.03**	**0.28**	**0.42**	**0.52**
Console (N=9)	Pre	4.54	4.38	3.46	3.54	3.08	3.69
	Post	4.44	4.78	3.67	4.11	3.44	3.78
	Change	**-0.09**	**0.39**	**0.21**	**0.57**	**0.37**	**0.09**

Fig. 14. Pre- and post-course surveys: (Left) CS1 and (Right) CS2.

spent *playing* with the sample solution and starter project. Researchers believe the correct numbers for GTA should be closer to those from console assignments. In the case of CS2, the large time difference between the first GTA (CS2-E) reflected the fact that the assignment was more complicated and demanded time for familiarization. The similar amount of time spent for the second CS2 assignment (CS2-F) showed that students were able to take advantage of their initial time investment.

4) Per-Assignment Survey: After students handed in their assignments and before they received their grades, the exact same questions were asked of all students: to assess the *clarity* and *difficulty* of the assignment and the amount they had *learned* from the assignment, if completing the assignment made them feel *satisfied*, and if the assignment was *interesting*.

Fig. 13 plots the average of all results from GTA and console courses for all the CS1 (left) and CS2 (right) courses. Results on the right of Fig. 13 showed CS2 students found the console assignments easier to understand, slightly more challenging, and, contrary to intuition, they found that working on console assignments was more satisfying and that the console assignments were more interesting.

Fortunately, written feedback from students helped explain these observations. In both CS1 and CS2 GTA courses, students complained that they had to spend extra time understanding the given GTA starter project. However, in all cases, once they understood the given system, students reflected that completing the assignment was not "*as complicated as it first appears*." In CS2 assignments, many students expressed frustration at not being able to *improve* on the assignments (e.g., "*it was interesting to have an end result you can play with, I wish I was able to improve the boring game*," or "*I did learn a lot about BST [Binary Search Tree] doing this assignment. However the 'game' parts were useless as we don't actually get to do gaming*.").

The GTA modules were designed to be *expertise-neutral* to prevent superfluous graphics user interaction programming and inappropriate gaming contents (such as violence) [5], [6]. Developers made a conscious decision to restrict students' access to such functionality [7]. This feedback showed that restricting students' access in such a way was successful. However, the feedback also implied that the modules had literally taken the "fun" out of game programming. The delicate balance between allowing creativity and discouraging excessive graphics programming is currently being investigated. For example, the fun element in the developed games may be incorporated if scoring and rewards are included for every right concept/code.

5) Pre- and Post-Course Survey: Course survey forms were designed to elicit students' background, self-perception, interests, and general attitudes toward the CS discipline. Students completed these forms during first day of class (*pre*course) and before the final exams (*post*-course). The tables of Fig. 14 show the averages of all students from the *GTA* and *Console* classes separately: The table on the left shows CS1, and the table on the right shows CS2 classes.[6]

It is interesting that replacement of *some* assignments in the featured CS1/2 classes was almost *transparent*. Because student enrollment in particular sections of the class cannot be controlled, some differences in items across classes at the onset are expected. However, to assess the impact of the curriculum, the *change* in attitudes from before and after each class section was measured and analyzed for statistically significant changes over time. Except for one case, the measured change from pre- to post- in the items in tables of Fig. 14 was not statistically significant. This exception was the "*Well prepared*" question for CS1 students (left-hand table of Fig. 14). In this case, when students were asked after the class was over, "*In hindsight, how well did*

[6]Students had the option to not participate in these surveys.

you think you were prepared for the class?" there was a significant improvement in self-confidence for GTA students. These data, in combination with the better performance results from GTA (e.g., CS1-C and -D in Fig. 11) and the little difference in attitude seen from the left of Fig. 13, implied an interesting and potentially important observation. CS1 students performed better because the interactive graphical application supported experimentation and visualization. Since the applications were not really fun or flashy, they did not find the assignments especially interesting. However, after the class, they did become more confident about their abilities. Together, these data suggested that targeted "*uninteresting*" interactive graphical assignments can be a good tool for teaching CS1 students.

6) Feedback From Faculty: After grading each assignment, the instructor filled out a feedback form detailing his efforts specific to the assignments (e.g., lecture time, answer questions, grading time) and impressions of student learning. As mentioned, the instructor did not have prior background in computer graphics/games and purposely avoided allocating specific lecture time and extra help for GTAs. As a direct result, his feedback showed no significant difference in efforts between the GTA and console assignments. However, he reported students' verbal comments on GTAs to be that they were more work and more difficult.[7] In the end, because the GTAs were "*dropped*" into the classes without any dedicated lecture time, the assignments had minimal effect on the class as a whole. It is encouraging that as the instructor became more comfortable with GTAs, he did begin experimenting with graphics/game programming and developed a simple card-matching game based on the provided tutorials. Currently, the instructor is experimenting with incorporating game-themed instructional modules in his CS1 classes [49].

IX. Conclusion

The resulting games and survey feedback from GTA workshop participants indicated that it is straightforward for faculty without graphics/games background to understand and begin working with GTA modules and to develop their own game-like applications. This classroom case study demonstrated that it is possible for a CS faculty member with no background in graphics/games to integrate GTA modules in an existing course without adverse effects on student learning. These results are exciting because they mean that interested faculty can confidently begin limited curriculum scope experimentation with selected GTA modules in their own courses. To further support these faculty members, limited curriculum scope, self-contained "game-themed instructional" (GTI) modules for teaching individual programming concepts (e.g., linked lists or arrays) have been developed [49]. Interested faculty members will be able to experiment with selected GTA and GTI modules in their existing classes, become comfortable with the game-themed teaching approach, consult the tutorials provided with GTA and GTI modules, and begin to develop their own game-themed materials. Ultimately, the success of any new approach to teaching hinges on the instructor's expertise and enthusiasm. The true potential for engaging and exciting students can only be realized when the instructor becomes proficient in, feels ownership of, and develops his or her own game-themed instructional materials based on the needs and strengths of their students.

Currently, a multilingual and API-independent platform to support GTA/GTI modules in multiple programming languages and APIs is being designed. In addition, results of this work are being disseminated with colleagues from community colleges and high schools, and workshops at national conferences (e.g., [50]) and at international institutions are continuing to be offered (e.g., [51]).

[7]The instructor did not have access to any survey information during the classes.

Acknowledgment

The authors would like to thank all BIT 142 and 143 students from Cascadia Community College for working with the chaotic schedules and draft versions of the assignments, and L. Dirks at Microsoft Research for his ongoing support of this work. The reviewers' excellent attention to details and comments have greatly improved the quality of this paper. All opinions, findings, conclusions, and recommendations in this work are those of the authors and do not necessarily reflect the views of Microsoft.

References

[1] E. Sweedyk, M. deLaet, M. C. Slattery, and J. Kuffner, "Computer games and CS education: Why and how," in *Proc. SIGCSE*, 2005, pp. 256–257.

[2] U. Wolz, T. Barnes, I. Parberry, and M. Wick, "Digital gaming as a vehicle for learning," in *Proc. SIGCSE*, 2006, pp. 394–395.

[3] J. D. Bayliss, "The effects of games in CS1-3," *J. Game Dev.*, vol. 2, no. 2, 2007***[Pages?]***.

[4] S. Leutenegger and J. Edgington, "A games first approach to teaching introductory programming," in *Proc. SIGCSE*, 2007, pp. 115–118.

[5] H. M. Walker, "Do computer games have a role in the computing classroom?," *SIGCSE Bull.*, vol. 35, no. 4, pp. 18–20, 2003.

[6] S. Haller, B. Ladd, S. Leutenegger, J. Nordlinger, J. Paul, H. Walker, and C. Zander, "Games: Good/evil," in *Proc. SIGCSE*, 2008, pp. 219–220.

[7] K. Sung, M. Panitz, S. Wallace, R. Anderson, and J. Nordlinger, "Game-themed programming assignments: The faculty perspective," in *Proc. SIGCSE*, 2008, pp. 300–304.

[8] K. Sung, R. Rosenberg, M. Panitz, and R. Anderson, "Assessing game-themed programming assignments for cs1/2 courses," in *Proc. GDCSE*, 2008, pp. 51–55.

[9] K. Sung, M. Panitz, R. Reed-Rosenberg, and R. Anderson, "CS1/2 game-themed programming assignments for faculty," *J. Game Dev.*, vol. 3, pp. 27–47, Mar. 2008.

[10] C. Hillyard, R. Angotti, M. Panitz, K. Sung, J. Nordlinger, and D. Goldstein, "Game-themed programming assignments for faculty: A case study," in *Proc. SIGCSE*, 2010, pp. 270–274.

[11] J. D. Bayliss, "Using games in introductory courses: Tips from the trenches," in *Proc. SIGCSE*, 2009, pp. 337–341.

[12] D. A. Smith and L. C. Moore, *Calculus: Modeling and Applications*, 2nd ed. Boston, MA: Houghton Mifflin, 2007 [Online]. Available: http://www.math.duke.edu/education/calculustext

[13] "The Carl Wieman Science Education Initiative (CWSEI)," Univ. British Columbia, Vancouver, Canada, 2008 [Online]. Available: http://www.cwsei.ubc.ca

[14] M. Guzdial and B. Ericson, *Introduction to Computing and Programming With Java, A Multimedia Approach*. Englewood Cliffs, NJ: Prentice-Hall, 2007.

[15] M. Guzdial, "Contextualized computing education," Invited presentation, Microsoft Research Faculty Summit, Jul. 2008 [Online]. Available: http://home.cc.gatech.edu/guzdial/169

[16] P. Drake, *Data Structures and Algorithms in Java*. Upper Saddle River, NJ: Prentice-Hall, 2006.

[17] M. McNally, M. Goldweber, B. Fagin, and F. Klassner, "Do LEGO Mindstorms robots have a future in CS education?," in *Proc. SIGCSE*, 2006, pp. 61–62.

[18] K. Sung, "Computer games and traditional computer science courses," *Commun. ACM*, vol. 52, no. 12, pp. 74–78, Dec. 2009.
[19] D. Frost, "Ucigame, a Java library for games," in *Proc. SIGCSE*, 2008, pp. 310–314.
[20] J. Linhoff and A. Settle, "Motivating and evaluating game development capstone projects," in *Proc. FDG*, 2009, pp. 121–128.
[21] K. Sung, P. Shirley, and R. Reed-Rosenberg, "Experiencing aspects of games programming in an introductory computer graphics class," in *Proc. SIGCSE*, 2007, pp. 249–253.
[22] S. M. Pulimood and U. Wolz, "Problem solving in community: A necessary shift in cs pedagogy," in *Proc. SIGCSE*, 2008, pp. 210–214.
[23] P. Haden, "The incredible rainbow spitting chicken: Teaching traditional programming skills through games programming," in *Proc. ACE*, 2006, pp. 81–89.
[24] J. D. Bayliss and D. I. Schwartz, "Instructional design as game design," in *Proc. FDG*, 2009, pp. 10–17.
[25] A. Luxton-Reilly and P. Denny, "A simple framework for interactive games in cs1," in *Proc. SIGCSE*, 2009, pp. 216–220.
[26] K. Bierre, P. Ventura, A. Phelps, and C. Egert, "Motivating OOP by blowing things up: An exercise in cooperation and competition in an introductory Java programming course," in *Proc. SIGCSE*, 2006, pp. 354–358.
[27] M. C. Lewis and B. Massingill, "Graphical game development in cs2: A flexible infrastructure for a semester long project," in *Proc. SIGCSE*, 2006, pp. 505–509.
[28] W. Dann, S. Cooper, and R. Pausch, *Learning to Program with Alice*. Upper Saddle River, NJ: Prentice-Hall, 2006.
[29] M. Külling and P. Henriksen, "Game programming in introductory courses with direct state manipulation," in *Proc. ITiCSE*, 2005, pp. 59–63.
[30] R. B.-B. Levy and M. Ben-Ari, "We work so hard and they don't use it: Acceptance of software tools by teachers," *SIGCSE Bull.*, vol. 39, no. 3, pp. 246–250, 2007.
[31] L. Ni, "What makes CS teachers change?: Factors influencing CS teachers' adoption of curriculum innovations," in *Proc. SIGCSE*, 2009, pp. 544–548.
[32] "BIT142/143: Intermediate programming and data structure," Cascadia Community College, Bothell, WA, 2008 [Online]. Available: http://faculty.cascadia.edu/mpanitz/courses/2008Fa/BIT142/ [Online]. Available: http://faculty.cascadia.edu/mpanitz/courses/2008Sp/BIT143/
[33] H. B. Christensen and M. E. Caspersen, "Frameworks in CS1: A different way of introducing event-driven programming," in *Proc. ITiCSE*, 2002, pp. 75–79.
[34] V. K. Proulx, J. Raab, and R. Rasala, "Objects from the beginning—With GUIs," in *Proc. ITiCSE*, 2002, pp. 65–69.
[35] S. Matzko and T. A. Davis, "Teaching CS1 with graphics and C," in *Proc. ITICSE*, 2006, pp. 168–172.
[36] "XNA Game Studio," Microsoft, Inc., 2007 [Online]. Available: http://msdn2.microsoft.com/en-us/directx/Aa937794.aspx
[37] "Game-themed introductory programming project home page," Univ. Washington, Bothell, 2010 [Online]. Available: http://depts.washington.edu/cmmr/Research/XNA_Games
[38] R. Anderson, R. Anderson, K. M. Davis, N. Linnell, C. Prince, and V. Razmov, "Supporting active learning and example based instruction with classroom technology," *SIGCSE Bull.*, vol. 39, no. 1, pp. 69–73, 2007.
[39] T. B. Horton, R. E. Anderson, and C. W. Milner, "Work in progress—Reexamining closed laboratories in computer science," in *Proc. 34th Annu. ASEE/IEEE Frontiers Educ. Conf.*, Oct. 2004, vol. 2, pp. F3C-15–F3C-16.
[40] J. Frechtling and L. Sharp, "User-friendly handbook for mixed method evaluations," Division of Research, Evaluation and Communication, National Science Foundation: Directorate for Education and Human Resources, Arlington, VA, 1997.
[41] M. Panitz and K. Sung, "Incrementally incorporating video games into instruction using XNA game-themed assignments," in *Proc. CCSC-NW*, Oct. 2008***[Pages?]***.
[42] K. Sung, "XNA game-themed applications for teaching introductory programming courses," in *Proc. 4th Int. Conf. Found. Digital Games*, Orlando, FL, Apr. 2009***[Pages?]***.
[43] K. Sung, "XNA game-themed applications for teaching introductory programming courses," in *Proc. Invited 3-Day Workshop, Microsoft Mexico Digital Arts Univ.*, Guadalajara, Mexico, Feb. 2009***[Pages?]***.
[44] K. Sung, "Developing game-themed applications for teaching introductory programming courses," in *Proc. Invited 2-Day Workshop, Beijing Univ. Technol.*, Beijing, China, Jul. 2009***[Pages?]***.
[45] J. Cromack and W. Savenye, "Learning about learning in computational science and science, technology, engineering and mathematics (STEM) education," 2007 [Online]. Available: http://research.microsoft.com/ur/us/AssessmentToolkit/
[46] M. Eagle and T. Barnes, "Experimental evaluation of an educational game for improved learning in introductory computing," in *Proc. SIGCSE*, 2009, pp. 321–325.
[47] I. Parberry, T. Roden, and M. B. Kazemzadeh, "Experience with an industry-driven capstone course on game programming: Extended abstract," in *Proc. SIGCSE*, 2005, pp. 91–95.
[48] S. Cooper, W. Dann, and R. Pausch, "Teaching objects-first in introductory computer science," in *Proc. SIGCSE*, 2003, pp. 191–195.
[49] R. Angotti, C. Hillyard, M. Panitz, K. Sung, and K. Marino, "Game-themed instructional modules: A video case study," in *Proc. FDG*, 2010, pp. 9–16.
[50] M. Panitz, K. Sung, and J. Nordlinger, "Develop game-themed examples for CS1/2 without background in graphics or games," in *Proc. SIGCSE*, Mar. 2010***[Pages?]***.
[51] K. Sung, "Developing game-themed applications with XNA," in *Proc. Invited 3-Day Workshop, Serious Games Winter School 2010*, Feb. 2010***[Pages?]***.

Kelvin Sung received the B.E.E. degree from the University of Wisconsin-Madison in 1986, and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign in 1990 and 1992, respectively.

He was an Assistant Professor with the School of Computing, National University of Singapore, Singapore, and a Software Architect with Alias—Wavefront (now part of Autodesk)***[City?]***, where he played a key role in designing and implementing the first version of the Maya Renderer. Currently, he is a faculty member with the Computing and Software Systems Department, University of Washington Bothell, Bothell. His research interests include high-quality image synthesis, video game development, serious games, and computer science education.

Prof. Sung is a Member of the Association for Computing Machinery (ACM).

Cinnamon Hillyard received the Ph.D. degree in mathematics from Utah State University, Logan.***[Yr degree received?]***

She is an Assistant Professor with the Interdisciplinary Arts and Sciences Program, University of Washington Bothell, Bothell. She also completed a postdoctorate position in mathematics with the University of Arizona, Tucson. Her scholarship focuses on undergraduate mathematics and science education, especially in the assessment of learning outcomes.

Prof. Hillyard currently holds the position of Past-Chair of the Special Interest Group of the Mathematical Association of America on Quantitative Literacy (SIGMAA-QL) and is the Secretary/Treasurer for the National Numeracy Network (NNN).

Robin Lynn Angotti received the B.S. degree (1988) in mathematics and the M.A. degree***[Subject area?]*** from East Carolina University, Greenville, NC, in 1988 and 1990, respectively, and the Ph.D. degree in mathematics education from North Carolina State University, Raleigh, in 2004.

She taught secondary mathematics at D. H. Conley High School, Greenville, NC, and remedial mathematics at North Carolina State University. She was the Assistant Director of the Center for Science, Mathematics and Technology Education and an Assistant Professor with East Carolina University. In 2007, she became an Assistant Professor with the University of Washington Bothell, Bothell. She has published in *Mathematics Teacher*, *The NCTM Yearbook on Data and Chance*, *The AMTE Monograph*, and the *Statistics Education Research Journal*. Her fields of interest are mathematics, statistics, and technology education.

Dr. Angotti is a Member of the North American Chapter of the Psychology of Mathematics Education, the National Council of Teachers of Mathematics, and the Association of Mathematics Teacher Educators.

Michael W. Panitz received the B.A. and M.Eng. degrees in computer science from Cornell University, Ithaca, NY, in 1998 and 1999, respectively.

He has worked with Microsoft in the .Net Common Language Runtime group and is currently a Senior Founding Faculty Member with Cascadia Community College, Bothell, WA. He is currently interested in using innovative technologies and techniques to teach lower division computer programming and computer science, and he has been interested in reliable distributed systems.

Mr. Panitz is a Member of the Association for Computing Machinery (ACM).

David S. Goldstein received the B.A. degree in English in 1984, the M.A. degree in communication in 1985, the M.A. degree in American civilization in 1988, and the Ph.D. degree in comparative culture in 1997.***[From which universities?]***

He edited career-guidance books for engineers with Professional Publications, Inc., Belmont, CA, and was a Lecturer with the University of California, Irvine; Shoreline Community College, Shoreline, WA; and the University of Washington Bothell, Bothell, where he is now a Senior Lecturer and the Director of the Teaching and Learning Center. He has presented internationally on the scholarship of teaching and learning and co-edited, with Audrey B. Thacker, *Complicating Constructions: Race, Ethnicity, and Hybridity in American Texts* (Univ. Washington Press, 2007). He serves on the Editorial Board of *Ethnic Studies* and peer reviews for *Ethnic Studies Review* and *Multi-Ethnic Literature of the United States*.

Dr. Goldstein is a founding member of the Research Committee of the Association for Authentic, Experiential and Evidence-Based Learning.

John Nordlinger is currently pursuing the Master's degree in film production at the University of Southern California, Los Angeles.

He joined Microsoft Research, Redmond, WA, in 2001, where he collaborated with academic institutions in the northeast United States and India. After convincing Microsoft Research to open a research lab in Bangalore, India, he then focused on mitigating the decline in CS enrollments. Along with participating on various panels, he coauthored two papers at SIGCSE 2008: one on teaching with XNA GSE, and one on teaching CS with socially relevant themes. He also co-edited the book *World of Warcraft and Philosophy* (Open Court, 2009) and contributed to the tome *Ethics and Game Design: Teaching Values Through Play* (Inf. Sci. Reference, 2010), by David Gibson and Karen Schrier. His first film, *The Allegory of the Game*, has been selected for three film festivals.