Game-Themed Instructional Modules: A Video Case Study*

Robin Angotti
U. of Washington, Bothell
rrider@uwb.edu

Cinnamon Hillyard
U. of Washington, Bothell
chillyard@uwb.edu

Michael Panitz
Cascadia Community College
mpanitz@cascadia.edu

Kelvin Sung
U. of Washington, Bothell
ksung@uwb.edu

Keri Marino
U. of Washington, Bothell
kerij@myuw.net

ABSTRACT

Integration of video games into introductory programming (CS1/2) courses motivates and engages students while contributing to their learning outcomes [17, 1, 2]. However, it is challenging for general faculty members teaching CS1/2 courses, few of whom have computer graphics or games backgrounds, to integrate video games. Game-Themed Instructional (GTI) Modules are designed specifically to encourage general faculty members to teach CS1/2 concepts using interactive, graphical, game-like examples. Six independent and self-contained GTI modules were created as a collection of interactive graphical example programs designed to demonstrate one single programming concept (e.g., conditional statements). This paper discusses the design parameters and implementation of the GTI modules and describes a case study of selectively adopting some of the GTI modules in an existing CS1 class. The results of the study demonstrate that it is possible for a faculty member with no games or graphics background to blend GTI modules into an existing CS1 class with minimum alterations to established course materials. The GTI modules are excellent catalysts, enabling faculty to begin exploring teaching with game-themed materials and helping students to be more engaged in CS1 topics.

Categories and Subject Descriptors

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

General Terms

Design, Experimentation

*This work is supported in part by Microsoft External Research under the Computer Gaming Curriculum in Computer Science RFP, Award Numbers 15871 and 16531.

FDG'10, June 19-21, 2010, Monterey, CA, USA.

Keywords

CS1/2, Assessment, Games, Courseware, Adoption

1. INTRODUCTION

Despite the proven success of using computer video games as a context for teaching introductory programming (CS1/2) courses (e.g., [17, 1, 2]), significant barriers prevent interested faculty members from experimenting with this approach. These barriers include the lack of: adoptable materials, required background expertise in graphics and games, and institutional acceptance [13, 16, 10]. One approach to overcoming these obstacles is to support interested faculty members in gradually developing expertise by allowing them to experiment and gain experience with game-themed materials in well-defined topic areas as part of their usual classes. In this way, as these faculty members develop their expertise they can also collect and demonstrate results on student engagement and achievement to assist the decision making process of institutional oversight committees [25].

The Game-Themed Introductory Programming project[1] design follows the "*Web 2.0*" model of providing simple tools for users to build their own content. In addition to providing easily adoptable Game-Themed Instructional (GTI)[2] modules, a framework that supports GTI implementation, *XNACS1Lib*, is also provided as an integral tool for adopters to design and build their own materials. To allow for ubiquitous adoption, principle design goals of the GTI modules are that they are "*simple*," "*self-contained*,""*limited in curriculum scope*," and serve as example courseware using the *XNACS1Lib* library. Simplicity ensures easy comprehension while the independence and restricted scope allows interested faculty members to pick and choose any subset of the materials to pace integration into their existing classes.

GTI 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 the term "game-themed" is used, because the programs run on both PC and the XBox 360 gaming platforms.

This paper focuses on the design and implementation of GTI modules. Further, it gives results from a pilot case

[1] *http://depts.washington.edu/cmmr/Research/XNA_Games.*
[2] A GTI module is a collection of game-themed examples with accompanying notes designed to support the teaching of one programming concept (e.g., conditional statements).

study in which an instructor selectively adopted GTI modules. It is important to note that there is nothing magical about teaching with games. As highlighted by Bayliss [3], faculty buy-in and experience are some of the most important factors in realizing the potential student engagement of a game-themed teaching approach. GTI modules were designed to support faculty members developing expertise in the area. The objective of this pilot study was to verify that GTI modules could be adopted with minimal extra efforts 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 GTI modules "*do no harm*" to student learning while faculty members incrementally experiment with and develop experience in game-themed context. This pilot project focused on the effectiveness of the GTI modules and did not require the adopting faculty to develop their own game-themed materials.

2. BACKGROUND

When evaluating new and innovative teaching materials for adoption, faculty must attend to factors such as preparation time, material contents, institutional oversight procedures, and infrastructure support [13, 16]. Additional adoption concerns include finding interested faculty to develop expertise in computer graphics and games, developing the content of games to avoid alienation of under-represented groups [26, 8], gathering evidence to facilitate departmental curriculum committee decision making processes, and ensuring existing systems are compatible with and capable of supporting graphics-intensive programs (e.g., programming language comparability, or sufficiently advanced graphics hardware).

Existing projects for teaching CS1/2 courses using the context of video games permit students to play with custom games (e.g, [6]), to develop individual games as assignments (e.g., [2, 15]), or to work with custom game environments throughout the entire course (e.g., [14, 4]). These projects were developed by faculty with expertise in graphics and games. The main goals of these projects were student engagement and achievement of student learning outcomes [22]. Adaptability and generalizability of the resulting materials were not main concerns. Utilizing these existing materials requires a significant investment of time (e.g., understanding a game engine [14]), or new programming environment (e.g., [12]), or significant reworking of current curriculum (e.g., [4]). These factors prevent limited scope investigation of the materials by time-constrained faculty members who have little or no background in graphics or games.

Success of any new approach to teaching hinges on the instructor's expertise and enthusiasm. This project aims to provide *simple* tools and materials to make it easy for interested faculty to develop game programming skills while they experiment with the materials in their classes, teaching the core concepts that are already part of their CS1/2 curriculum. The rest of the paper focuses on the GTI modules, *XNACS1Lib* framework, and a mixed-method study of students using modules in an existing class.

3. GTI DESIGN AND IMPLEMENTATION

To meet our goal of making GTI modules easily adoptable, each module is a complete teaching aid and can be used either as a student self-study guide or as a laboratory exercise manual. This completeness means that modules can serve both as classroom courseware and as templates for faculty seeking to develop game-themed teaching materials.

```
// Label A: Variables: the ball and paddles
private XNACS1Circle PongBall;                          // The pong ball
private XNACS1Rectangle LeftPaddle, RightPaddle; // Left & right paddles
// Label B: Initialization of the game
protected override void InitializeWorld() {
    LeftPaddle = new XNACS1Rectangle( ... );      // Create and Initialize
    RightPaddle = new XNACS1Rectangle( ... );     // the left/right paddles
    PongBall = new XNACS1Circle( ... );           // Create the pong ball
    PongBall.Velocity = Random( ... );            // Set Pong ball velocity
}
// Label C: Periodic updates of the game state
protected override void UpdateWorld() {
    // Let user move left/right paddles in the y-direction
    LeftPaddle.CenterY += ThumbSticks.Left.Y;     // Left paddle up/down
    RightPaddle.CenterY += ThumbSticks.Right.Y; // Right paddle up/down
   //Label D: Conditional Statement
    if (LeftPaddle.Collided(PongBall))
        PongBall.VelocityX *= -1;                  // Left paddle/ball collided
}
```

Listing 1: Simple Pong game with XNACS1Lib

3.1 Design Considerations

Instructional goals of the GTI modules for CS1/2 programming concepts are satisfied by a collection of strategically constructed examples, each with detailed step-by-step guides and relevant followup exercises. To support student self-study, each module begins with simple examples and builds gradually in complexity. Faculty pick and choose any combination of GTI modules with each self-contained module covering a well defined concept (e.g., conditional statements). Though language-specific implementation details are important, GTI modules reflect that the foundational programming concepts are programming language neutral (e.g., "if" statement syntax and usages are similar across Java/C++/C#).

These considerations lead to GTI modules organized according to chapters and examples from a language independent textbook [5]. Each GTI module corresponds to a chapter in the textbook, with the concepts demonstrated by game-themed examples. Following each example is a set of follow-up questions which challenge students to refine the game in order to demonstrate their understanding of the concepts. Examples within a module build on each other and increase in difficulty. Examples from different GTI modules are completely independent, ensuring self-contained units.

Whenever possible and appropriate, examples in a module build to a variation of the classical Pong game. This is a popular game with simple rules and lends itself well for demonstrating introductory programming concepts.

3.2 The XNACS1Lib Framework

In order to support fast prototyping, the C# programming language and the Microsoft XNA framework were chosen as the implementation platform. This platform is a public and freely available solution which provides seamless integration of the development environment, programming language, GUI API, and graphics API on PC, mobile device (Zune), and console (XBox 360) platforms.[3]

Listing 1 shows the source code to a simple Pong-game based on the *XNACS1Lib* framework. Under Label **A**, the Pong ball and the two paddles are defined as the circle and rectangle predefined data types. These objects behave *intuitively* (e.g., automatically drawn, and change position according to velocities). The *InitializeWorld()* function un-

[3]The main reservation with a Java based solution was the prospect of working with many independent APIs.

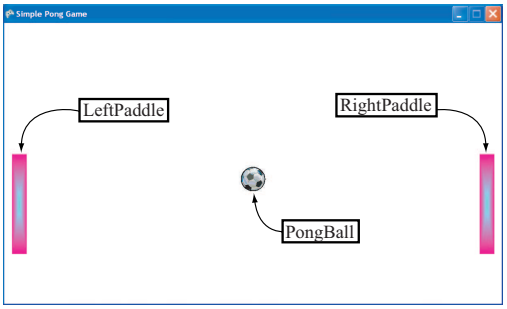

Figure 1: Simple Pong-Game of Listing 1.

der Label **B** is invoked once before the game begins. In this case, memory is allocated and initialized. The *UpdateWorld()* function under Label **C** is invoked continuously at 25-millisecond intervals during the game. In this case, with the ball traveling according to its velocity, a Pong game only requires updating the paddles positions (based on the user's *ThumbStick*), and detecting ball–paddle collisions (under Label **D**). Figure 1 shows the simple Pong-game resulting from the code of Listing 1.

The *XNACS1Lib* defines a small framework that supports the development of simple game-like applications. It is important to note that *XNACS1Lib* is suitable for building *simple* game-like applications. With careful design, such simple applications can be suitable courseware materials for teaching CS1/2 courses. However, the framework is not suitable for building general purpose games.

Figure 2: Games by Faculty with no prior backgrounds.

3.2.1 XNACS1Lib Workshops For Faculty

Outside of classrooms, *XNACS1Lib* related faculty development workshops have been offered for interested faculty members at regional and national conferences (e.g., [23]) and at institutions internationally (e.g., [24]).[4] These workshops guided faculty to develop game-themed applications based on the *XNACS1Lib*. Images in Figure 2 are screen shots of a "*Worm-like*" and a "*Pizza Delivery*" games resulting from the second day of multi-day workshop (e.g., [24]). 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

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

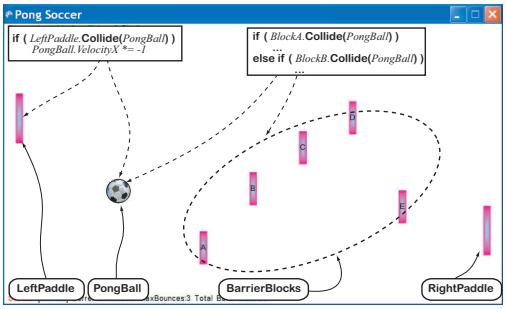

Figure 3: Conditional Statements: Pong Soccer.

- "*good balance of complexity and new materials.*"

While the written feedback on "*the presented materials will help me develop game-themed applications*," included:

- "*For sure!*" and
- "*In this environment, YES!*"

The results and feedback from these workshops showed that, though they found the materials to be non-trivial, faculty participants with no prior expertise in graphics/games comprehend and develop game-like applications within a matter of hours.

3.3 GTI Modules

CS1 Topic	# Examples	Final Game
IO & Variables	14	N/A
Functions	10	N/A
Conditionals	12	Pong Variation
Loops	21	Laser Zap
Arrays	10	Pong Variation
Classes & Objects	12	Pong Variation

As shown in the above table, six GTI modules were developed with a total of 79 examples based on topics covered in typical CS1 classes. The first two modules include trivial examples as it is difficult to build such simple examples into complete games. The *Loop* module exploits the repetition construct in defining paths of laser beams, while the remaining three examples build to different variations of the Pong game.

As discussed, each example is accompanied by detailed notes. The following uses the *Conditionals* (or Decision Structure) GTI module to explain the organization of game-themed examples. The structure of the accompanied lecture notes is illustrated in Figure 4, which outlines the notes for the *Case Statement* example.

3.3.1 Examples from the Conditional GTI Module

The first example in the *Conditionals* module contains all the source code in Listing 1 except the very last conditional statement under Label **D**. This first example serves as a template for introducing subsequent conditional concepts. The goal of this module is to build up to a Pong-like game, and in this example the pong ball will travel rightward and disappear out-of-bound. The second example introduces simple floating point comparisons by replacing the following statement under Label **D**:

```
if (PongBall.CenterX > 100f)          // Pong ball out of bound
    PongBall.CenterX = 0;             // Reset to the left bound
```

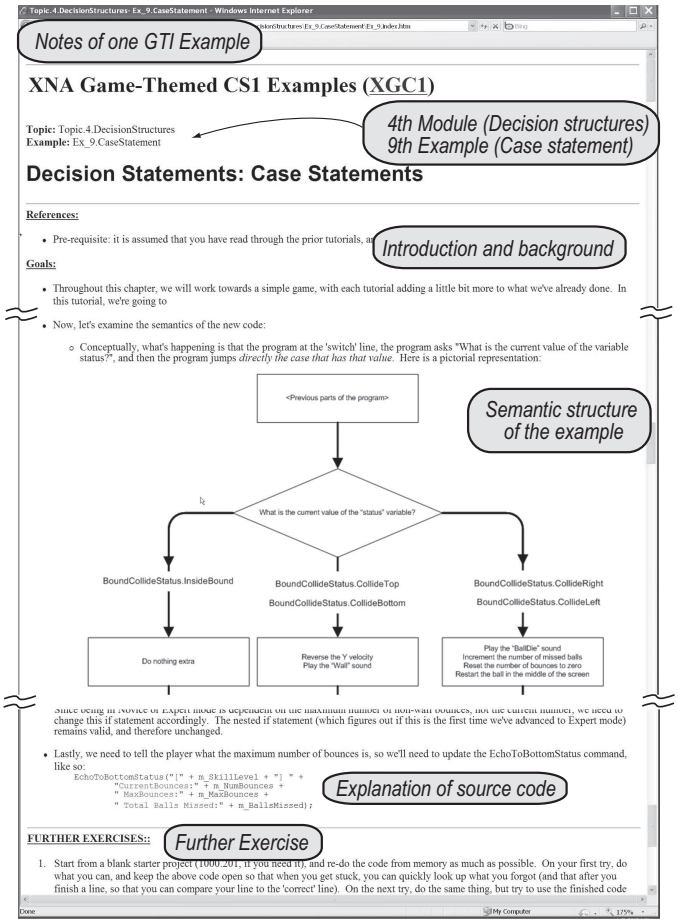

Figure 4: Notes for the Case Statement.

The third example is identical to Listing 1 where the left paddle is capable of bouncing the pong ball. The fourth example introduces the *if-then-else* construct by replacing the following statements under Label **D**:

```
if (LeftPaddle.Collided(PongBall))
    PongBall.VelocityX *= -1;                    // Left paddle/ball collided
else if (RightPaddle.Collided(PongBall))
    PongBall.VelocityX *= -1;                    // Right paddle/ball collided
```

The subsequent examples are equally simple: introducing nested *if* statements for collision with *BarrierBlocks* in Figure 3, logical comparison operators for detecting winning conditions, and *case* statements for alternates to nested *if* statements. Because all subsequent examples are related and introduce new concepts by modifying a similar region of the code (and by changing the game behavior accordingly), once students become familiarized with the initial template example they will find subsequent examples easier to understand. Figure 3 shows the Pong game from the last example of the *Conditionals* module.

3.3.2 Notes from One GTI Example

As illustrated in Figure 4, detailed notes accompany each game-themed example. In this case, a portion of the notes for the *case* statement example is shown. In general, notes are divided into four main sections. The first section introduces the example and includes a list of prerequisite knowledge as well as associated reference links in case students need review. The second section offers a detailed explanation of concepts illustrated in the example. The third section offers a detailed explanation of how the concept is implemented in source code, frequently explaining how the code works step-by-step. The fourth section lists exercises for students to complete reinforcing the intended concepts. Notes for the 79 examples are extensive, and amount to 500+ web-pages.[5]

4. EXPERIMENT DESIGN

As indicated, use of Game-Themed instruction can increase student engagement in CS1/2 courses. However, that fact alone is not enough to facilitate faculty buy-in to the use of the GTI modules. The purpose of this study was to demonstrate that the modules can be selectively adopted in existing classes with no adverse effects on student learning. From the position of an interested faculty with limited expertise in graphics and games, it is speculated that initial classroom adoption will be of relatively small scale. One reasonable scenario is for such faculty members to experiment with a limited number of game-themed examples in the classroom and recommend the modules as an out-of-classroom self-study guide. With these goals and considerations, this study focused on initial examples from three different GTI modules as in-classroom exercises. Researchers wanted to understand if instructors and students, without graphics and games background, could read and follow the game-themed examples with little or no assistance.

In order to investigate the effectiveness and ease of adoption, a mixed-method research plan was employed to study how students interact with GTI modules. This included collecting data from student surveys and evaluating videotaped classroom human-computer interactions of students working with GTI modules. It was hoped that this analysis would give insight into students' use of GTI modules when given little faculty support. In addition, this analysis facilitates future research on using GTI as a lab exercise workbook or self-guided tutorials outside of typical classroom instruction. Video recording was employed to capture both what students were doing with GTI modules as well as student-student and student-teacher interactions. This type of video-based observation had the unique properties of allowing for capturing and reflecting on a complex classroom setting and giving a more complete picture of how students were using the modules [7, 9].

4.1 Experiment Setup

The instructor selected the first exercise from each of the *Variables*, *Loops*, and *arrays* modules on and integrated them into the course featured in this research. The use of three independent modules tested the theory that the modules could be selected and used as needed rather than as a complete curriculum. All students in the class were required to do the selected exercises in class at the same time. Students were given approximately one hour for the first module and 30 minutes for the remaining two subsequent modules. Student surveys were given at the end of each exercise and measured student opinions on clarity of module content, effectiveness of the module as a learning tool, as well as enjoyment of the game-themed environment. Six groups

[5]All GTI modules are freely available at *http://depts.washington.edu/cmmr/Research/XNA_Games.*

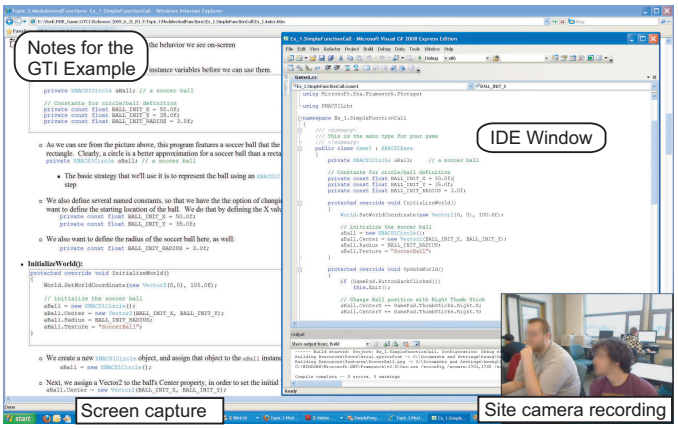

Figure 5: PIP from Screen Capture and Site Camera.

of students were recorded and the recordings transcribed. Data was collected via two video and voice captures: first on the interaction of the pairs with a site camera focused on the students; and the second by computer screen capture. Figure 5 shows that the two data streams were synchronized into a picture-in-picture (PIP) video where the computer screen capture was the primary screen scene [7] with the site camera action overlaid in the corner. Data were analyzed across cases for each module and within cases for each pair of subject participants. Students were given no faculty assistance other than an explanation of how the game is initialized, and how the UpdateWorld routine is periodically called to update the game. The objective was for students to try to solve the problem in the module on their own with little or no lecture. The instructor did not have specific graphics or games background but did understand the basic real-time model-view-controller interaction model.

In addition to the short survey after each lab exercise, a pre-course and a post-course survey were administered to the entire class. The pre- and post-surveys combined two instruments: one used for studying a similar project [10] and one developed and validated by Wilson and Shrock [27] which included the Self-Efficacy Scale developed by Ramalingan and Weidenback [19, 18]. These surveys were used to get a sense of student demographics as well as monitor change in attitudes across time.

5. RESULTS AND ANALYSIS

Results from the mixed-method study of students including pre- and post-course surveys and module surveys as well as the analysis of the video case study are outlined below.

5.1 Surveys

As summarized in Table 1, twenty-one students completed both pre- and post-course surveys. All of the students had taken a previous class in programming and just over 85% of the student had taken at least one calculus course, yet only 23% had received some sort of encouragement to go into computer science. Although changes between pre- and post-surveys vary, the only statistically significant changes were on the two questions: "*I plan to pursue advanced studies in a technical field*" ($p = 0.029$) and "*number of hours per week I play games on the computer*" ($p = 0.030$). These results indicate that students started to see themselves moving to more professional users of computers. The class contained only two female students so results were not disaggregated by gender. As seen in Table 2, positive, statistically significant gains were made in the overall self-efficacy scale and subscales. These gains are comparable to other documented uses of the scale [27, 19, 18], validating the claim that using these modules maintains student learning.

Results from the surveys given after the individual exercises reveal that students were generally satisfied with each of the modules in the areas of clarity in write-up ($mean = 4.1$),[6] content ($mean = 4.2$), learning achieved ($mean = 3.6$), and enjoyment of the game-theme ($mean = 4.2$).

5.2 Video Analysis

The interactive graphical characteristics of the GTI modules allowed students to begin with more concrete, visual, on-screen representations of the computer game, and then move into the more abstract source code that was responsible for creating that game. Because of ease of use of GTI modules, students could repeatedly refine their solutions and interact with visual results immediately. Using this process, students were able to construct an understanding of which specific lines of code were responsible for certain output in the game. The flow of learning became a continuous cycle of visualizing output and modifying code. This cyclical process of learning is highlighted in the following transcript of student interaction during one video when they were working with an example that produced one laser beam and were asked to create a second beam:

S2: "*See, I'm wondering if it's this? Something to do with this*" [**points to screen**].
S1: "*You think so? Maybe change it too.*"
S2: "*Cause, what- Hero beam dot x.*"
S1: "*Do you think that means how fast it's- it's going across?*"
S2: "*Hmm.*"
S1: "*Something like. ... Let's try this here.*"
[**Code changes, compile/run**]
S2: "*Well we got two beams.*" [instead of one previously]
S1: "*We got two beams. I don't know what's going on with that*"

Through interactions with GTI modules, evidence of code classification according to functionality were observed. GTI modules allowed students to start with visualizing the output and then delve into source code producing the visualization. In disciplines such as math and statistics, research has found that curricular approaches which emphasize flexibly moving from concrete graphical and numeric representations to abstract symbolic representations strengthens students' understanding of the underlying concepts [11, 20, 21]. At the very least, concrete representations give students a visual cognitive reference which aids in abstract symbolic manipulation skills. It is posited that the GTI modules gave computer science students similar flexibility by providing visual referents. The students were able to first see graphical output and then examine the underlying code producing that output. Repeated iterations of viewing graphical output and examining symbolic code may assist in strengthening their understanding of the underlying code.

Because of the immediate interactive graphical feedback, students were engaged and motivated to experiment with the

[6]On a scale of 0 to 5, with 5 being the best.

Question	Pre	Post
I am in this class because I like computer science.	4.05	4.05
I plan a career in technology.	4.57	4.33
I plan to pursue advanced studies in a technical career.	3.81	4.19*
My interest in computing is focused on programming.	3.24	3.38
My interest in computing is focused on gaming.	2.86	2.57
I have experience programming.	3.52	3.14
I expect this class to be difficult./This class was difficult.	3.48	3.29
I came into this class well prepared.	3.67	3.76
My preparation is equal to my classmates.	3.19	3.43
I will spend XX hours/week on this course outside of class time.	9.14	7.58
Working with my classmates will/did contribute to my learning.	3.81	3.29
Number of hours per week spent on surfing the web.	11.60	6.93
Number of hours per week spent on playing computer games.	11.29	9.1*
Number of hours per week spent using application software.	4.40	3.42
Level of comfort asking and answering questions in class.	1.9	1.86

*Statistically signi↓cant changes ($p<0.05$).

Table 1: Descriptive Statistics

	Pre-SE Mean	Post-SE Mean
Factor 1: Independence & Persistence	4.9170	5.9405
Factor 2: Complex Programming	3.7879	5.5671
Factor 3: Self Regulation	4.4048	5.0952
Factor 4: Simple Programming	3.0000	5.8134
Overall	121.62	175.40

$p < 0.025$

Table 2: Self-Efficacy Scales

programs, sometimes beyond what the assignment required. As evidenced in the following interaction between one pair of students who were interacting with the module environment before they started the assigned exercise:

S2: "*I know. We just did 'an else' so that it will uncolor it if you uncollide them.*"
Instructor: "*Are you guys following the exercise now?*" [while students were looking at the screen]
S1/S2: "*Yeah, oh yeah!!*"
S1: "*Cool! That's cool. We got the collision working!*"
S2: "*We have our colliding circle.*"
Instructor: "*Do the exercise now.*"
S2: "*Okay, what's the exercise? Bring it up.*"

Sometimes, this was seen as a problem in which the instructor had to refocus students on the assignment at hand because ease of programming allowed them to produce tangential results which were not part of the assignment (as evidenced by producing the colliding circle in the previous example). The visual feedback, although a powerful learning tool, could also be a source of distraction for students.

Another distraction was found to be the time involved in reading the background material in class while completing the module. The videos revealed that students were taking up to a third of the lab time to read this material. This invaluable insight resulted in the recommendation that students be instructed to read material before attending class. This would allow students more time to collaborate with partners, participate in the hands-on nature of GTI activities, complete assignments, and explore tangential extensions.

The stand-alone nature of the modules allowed students to work independently and at their own pace. In this way, students are able to construct their understanding of the underlying code from their pre-existing cognitive structures of programming. Some students with more programming experience were able to complete and go beyond assignment expectations in a shorter time, while others with less programming background needed more time and support in order to finish the assignment. Thus, the modules provided flexibility of use with multiple levels of learning and varying learning styles.

5.3 Faculty Feedback

After each video recording session and at the end of the academic quarter, the instructor was debriefed. He reported that given the limited in-classroom time it was challenging for students to become familiar with modules, and yet students found GTIs were "*interesting*" to work with once they were able to overcome the initial learning curve. In fact, other students wanted more opportunities to work with the GTIs after they become comfortable with the modules.[7] In the end, because the GTIs were essentially "*dropped*" into the class without any dedicated lecture time, the modules had minimal effect on the class structure as a whole. It is encouraging that as the instructor became more comfortable with the GTIs, he did begin experimenting with graphics and game programming and developed a simple card matching game. Even after this study, the instructor continues to select appropriate GTI modules/examples to use in his classes.

6. CONCLUSION

The uniqueness of the GTI modules is that they provide stand alone exercises that can be incorporated into any existing course structure. They do not need to be adopted as an entire curriculum. In this case study, after interaction with the GTI modules, improvements in students' self-efficacy and confidence levels were observed. It is believed that this is directly related to students' ability to experience the visual output of existing code first and to be able to explore the structure of the code that produced that output.

Typically, programming is taught by having the students

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

first write the code and then examine the output of what they wrote. When using the GTI modules, that process is reversed. GTI modules allowed students to start with visualizing the output and then delving into the source code producing the visualization. This backward design process gives an alternate method of learning programming. After modifying the code, the students reexamine the changes in the visualization, honing and focusing their understanding on the factors in the source code which affected the visualization. This cyclical, iterative process of learning fosters construction of knowledge and thus meets the needs of learners at different stages of concept formation rather than a typical "one size fits all" method of teaching the code and expecting students to apply it correctly. Even working independently with little or no instruction, students were successful in completing assignments, although at varying levels of time commitment depending on their previous programming experience. This is similar to any class assignment where students have varying levels of pre-existing knowledge.

GTI modules were created to be utilized by instructors who also have varying levels of pre-existing knowledge of computer graphics and games. The modules allowed the instructor to use GTI and to improve his own understanding of game-like programming with a minimal time investment. This allowed the instructor to increase his expertise and confidence in building his own game-themed teaching materials without the pressure of radically changing his course and feeling he/she must become proficient in game and graphics programming.

The video analysis allowed researchers and the program's creators to examine and document student engagement, community building in the classroom through independent use by students with little or no instructor feedback (i.e. students had to rely on each other for questions), and to determine where GTI modules needed improvement. This analysis is paving the way for modifications of existing modules and is adding to future work of creating a multilingual platform. Changes in the modules based on the analysis featured in this research include modifying the modules to be similar to lab experiments where the students will be assigned preliminary reading outside of class.

Several modifications are planned for video data collection. One of the challenges of video recording with two different scenes involved synchronizing the audio feed from two recordings. Future video collection would allow for the audio feed to be collected into both devices with one microphone, thus there would only be one audio feed for both videos. Lapel microphones would be used instead of stationary microphones to improve the quality of audio from softer speaking students.

This research suggests that the core concepts presented in GTI modules are programming language and API independent. Currently, a multilingual and API independent platform is being designed to support GTI modules in multiple programming languages and APIs. In addition, results of this work are being shared with colleagues from community colleges.

7. REFERENCES

[1] T. Barnes, H. Richter, E. Powell, A. Chaffin, and A. Godwin. Game2learn: building cs1 learning games for retention. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, pages 121–125, New York, NY, USA, 2007. ACM.

[2] J. D. Bayliss. The effects of games in cs1-3. *Journal of Game Development*, 2(2), 2007.

[3] J. D. Bayliss. Using games in introductory courses: tips from the trenches. In *SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education*, pages 337–341, New York, NY, USA, 2009. ACM.

[4] W. Dann, S. Cooper, and R. Pausch. *Learning to Program with Alice*. Prentice Hall, Upper Saddle River, NJ, 2006.

[5] T. Gaddis. *Starting Out with Programming Logic and Design*. Addison-Wesley/Prentice Hall, Upper Saddle River, NJ, 2008.

[6] P. Haden. The incredible rainbow spitting chicken: teaching traditional programming skills through games programming. In *ACE '06: Proceedings of the 8th Austalian conference on Computing education*, pages 81–89, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[7] R. Hall. Video recording as theory. In D. Lesh and A. Kelley, editors, *Handbook of Research Design in Mathematics and Sceince Education*, pages 647–664. Maweh, NJ: Lawrence Erlbaum, 2000.

[8] S. Haller, B. Ladd, S. Leutenegger, J. Nordlinger, J. Paul, H. Walker, and C. Zander. Games: good/evil. In *SIGCSE '08: Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pages 219–220, New York, NY, USA, 2008. ACM.

[9] J. Hiebert, R. Gallimore, H. Garnier, K. B. Givvin, H. Hollingsworth, J. Jacobs, A. M. Chui, D. Wearne, M. Smith, N. Kersting, A. Manaster, E. Tseng, W. Etterbeek, C. Manaster, P. Gonzales, and J. Stigler. *Teaching mathematics in seven countries: Results from the TIMSS 1999 Video Study*. Washington, D.C.: National Center for Education Statistics, 2003.

[10] C. Hillyard, R. Angotti, M. Panitz, K. Sung, J. Nordlinger, and D. Goldstein. Game-themed programming assignments for faculty: a case study. pages 270–274, 2010.

[11] E. J. Knuth. Student understanding of the cartesian connection: An exploratory study. *Educational Studies in Mathematics*, 31(4):500–507, 2000.

[12] M. Külling and P. Henriksen. Game programming in introductory courses with direct state manipulation. In *ITiCSE '05: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, pages 59–63, New York, NY, USA, 2005. ACM Press.

[13] 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.*, 39(3):246–250, 2007.

[14] M. C. Lewis and B. Massingill. Graphical game development in cs2: a flexible infrastructure for a semester long project. In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 505–509, New York, NY, USA, 2006. ACM Press.

[15] A. Luxton-Reilly and P. Denny. A simple framework for interactive games in cs1. In *SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education*, pages 216–220, New York, NY, USA, 2009. ACM.

[16] L. Ni. What makes cs teachers change?: factors influencing cs teachers' adoption of curriculum innovations. In *SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education*, pages 544–548, New York, NY, USA, 2009. ACM.

[17] I. Parberry, T. Roden, and M. B. Kazemzadeh. Experience with an industry-driven capstone course on game programming: extended abstract. In *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education*, pages 91–95, New York, NY, USA, 2005. ACM Press.

[18] V. Ramalingam, D. LaBelle, and S. Wiedenbaeck.

Self-efficacy and mental models in learning to program. *ACM SIGCSE Bulletin*, 36(3):171–175, 2004.

[19] V. Ramalingam and S. Wiedenbaeck. Development and validation of scores on a computer programming self-efficacy scale and group analysis of novice programmer self-efficacy. *Journal of Educational computing Research*, 19(4):367–381, 1998.

[20] R. Rider. *The effect of multi-representational methods on studentsŠ knowledge of function concepts in developmental college mathematics*. PhD thesis, North Carolina State University, March 2004. http://www.lib.ncsu.edu/theses/available/etd-03182004-090043/.

[21] R. Rider and H. Stohl Lee. Differences in students use of computer simulation tools and reasoning about empirical data and theoretical distributions. In A. Rossman and B. Chance, editors, *Proceedings of the Seventh International Conference on Teaching Statistics*, 2006.

[22] K. Sung. Computer games and traditional computer science courses. *Communications of the ACM*, 52(12):74–78, December 2009. Invited Paper, Peer Reviewed.

[23] K. Sung. Xna game-themed applications for teaching introductory programming courses. *Invited Pre-Conference Workshop, The Fourth International Conference on the Foundations of Digital Games, Orlando, Florida*, April 2009.

[24] K. Sung. Xna game-themed applications for teaching introductory programming courses. *Invited 3-Day Workshop, Microsoft Mexico and Digital Arts University, Guadalajara, Mexico*, February 2009.

[25] K. Sung, M. Panitz, S. Wallace, R. Anderson, and J. Nordlinger. Game-themed programming assignments: the faculty perspective. In *SIGCSE '08: Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pages 300–304, New York, NY, USA, 2008. ACM.

[26] H. M. Walker. Do computer games have a role in the computing classroom? *SIGCSE Bull.*, 35(4):18–20, 2003.

[27] B. Wilson and S. Shrock. Contributing to success in an introductory computer science course: a study of twelve factors. *ACM SIGCSE Bulletin*, 33(1):184–188, 2001.