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

Kelvin Sung
Computing and Software Systems
University of Washington, Bothell
ksung@uwb.edu

Cinnamon Hillyard
Interdisciplinary Arts and Sciences
University of Washington, Bothell
chillyard@uwb.edu

Robin Angotti
Education
University of Washington, Bothell
rrider@uwb.edu

Michael Panitz
Software Programming
Cascadia Community College
mpanitz@cascadia.ctc.edu

David Goldstein
Teaching and Learning Center
University of Washington, Bothell
dgoldstein@uwb.edu

John Nordlinger
Microsoft External Research
Microsoft
johnnord@microsoft.com

May 2, 2010

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. The 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 on-going GTA workshops for CS1/2 faculty members and from a year-long project in adopting the GTA modules in classes are then

*This work is supported in part by Microsoft External Research under the Computer Gaming Curriculum in Computer Science RFP, Award Number 15871 and 16531.
presented. In this case, the collected results verified that introductory programming concepts can be examined, practiced, and learned based on 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.

1 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, 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 generality 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, 6]). It is a challenge for departmental committees to arrive at 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 faculty in developing game-themed programming assignments. In this way, as the GTA modules are being adopted, faculty develop expertise and 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

\(^1\)All modules and related materials are freely available from the GTA project web-site: http://depts.washington.edu/cmmr/Research/XNA_Games.
of the modules [7], the assessment of the academic merits of the materials [8, 9], and finally the encouraging results from the on-going GTA workshops for CS1/2 faculty and a year-long 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.

2 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 has redesigned their 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 Georgia Institute of Technology [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 on-going 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 non-computer games (e.g., [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 (e.g., [19, 20]) where students learn about building games; game programming (e.g., [1]) where students study algorithms related to games; and game client (e.g., [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]: First, little or no games programming (e.g., [23, 24]) where students learn by playing custom games; second, per-assignment games development (e.g., [3, 7, 25]) where individual programming assignments are computer games designed around technical topics being studied; third, extensive game development where faculty and students work with custom games engines (e.g., [26, 27]), specialized programming language (e.g., [28]), environments (e.g., [29]), or specific curricula (e.g., [4]), etc. All three approaches reported resounding success with drastically increased enrollments and student success (e.g., [28, 3, 4]). 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 faculty understanding a game engine or significantly reworking 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.

3 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 backgrounds, 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.
3.1 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, 2D arrays, class hierarchy/inheritance, linked lists and queues, and binary search trees. Section 4 describes the assessment procedure that ensures that the console-based and game-themed versions of the assignment are technically equivalent.

3.2 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-themed versions,
• a sample grading rubric for each version,
• a list of frequently asked questions, and
• an implementation tutorial.

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, and
• 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.

3.3 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 (e.g., [33]), graphical user interfaces (GUI) (e.g., [34]) and graphical application programming interfaces (API) (e.g., [35]). To support faculty without 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 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.

![Figure 1: A Simple Pong Game.](image_url)
3.4 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 Figure 1). Listing 1 is the implementation for this simple application based on XGCS1, a simple 2D library designed specifically to support GTA modules.

```csharp
public class PongGame : XNACS1Base
{
    // All game-themed examples must subclass from the library XNACS1Base
    protected override void InitializeWorld()
    {
        // Define the pong field dimension
        Ball = new XNACS1Circle(...);
        LeftPaddle = new XNACS1Rectangle(...);
        RightPaddle = new XNACS1Rectangle(...)
    }
    protected override void UpdateWorld()
    {
        // The UpdateWorld function is called once every 25 milliseconds
        BoundCollideStatus status = World.ClampAtWorldBound(Ball)
        switch (status)
        {
            // if collided with the top/bottom bounds
            case BoundCollideStatus.CollideTop:
                Ball.VelocityY *= -1;
                break;
            // if collided with the top/bottom bounds
            case BoundCollideStatus.CollideBottom:
                Ball.VelocityY *= -1;
                break;
            // if a collision
            case BoundCollideStatus.CollideLeft:
                Ball.VelocityX *= -1;
                break;
            case BoundCollideStatus.CollideRight:
                Ball.VelocityX *= -1;
                break;
        }
    }
}
```

Listing 1: Simple Pong Game With XGCS1 Library

Label A in Listing 1 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 that, with the XGCS1 support:

- interactive, graphical, game-themed application can be simple and intuitive, 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, for example, the absence of programming code to draw, manipulate the Ball movements, and compute spatial collisions between geometric objects.

The fundamental programming logic flow is prominent in the implementation. In Listing 1, 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.

4 Results: The GTA Modules

The GTA modules are simple “interactive graphics applications” where the main goal of the assignments is to reinforce technical concepts and not on 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 on-line at [37].

Assignment one: Integer Arithmetic. This assignment is designed to be the first CS1 assignment. The left image of Figure 2 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 flash-card quiz program.

Assignment two: Random Number Generation and Operators. The right image of Figure 2 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 integration where students must approximate the area of a circle based on randomly generated sample positions.

Assignment three: Single Dimension Arrays of Object References. The left image of Figure 3 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.
Figure 3: **Left:** The Snakes and Ladders Game, and **Right:** The Othello Game.

at run-time. The game board is implemented as 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. The right image of Figure 3 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. The left image of Figure 4 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

Figure 4: **Left:** The Burst A Bubble Game, and **Right:** The Catch A Toy Game.
understand and subclass from a provided abstract *Projectile* class to implement the *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 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. The right image of Figure 4 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 en-queued to the front/back of the queue respectively. Similar to the game-themed version, the retrieval of requests is a simple de-queue 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 (BST). Figure 5 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).
5 Assessing the GTA Modules

The GTA modules are designed for students to practice and learn fundamental concepts in programming. It is important to independently evaluate 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.

5.1 Independent Reviewer

Professor 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, 39]). In addition to these excellent credentials, Professor 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, Professor Anderson did not know anyone on the project team.

5.2 Procedure

During the project, in-person or verbal communications were avoided to maintain impartiality, and to simulate investigations 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 Professor 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. \textit{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. \textit{Potential for adoption}: assesses the factors independent of the quality of assignments that may prevent adoption (e.g., programming language used).

5.3 Assessment Results

Based on the review feedback, the assignments have been well-received, overall. It is believed that because the assignments were reversely adopted based on existing CS1/2 classes, technical merits were never an issue. Professor Anderson agrees that the assignments are appropriate for typical CS1/2 courses. Assignments with low quality of assignment scores were revised and reassessed. This process continued until the formative comments are positive and the numeric scores are above 4 (out of 5).

\(^2\)ACM Faculty Award, voted best teacher by Department of Computer Science Students, University of Virginia, 2004
As expected, all of the low scores were caused by game-themed assignments. Typically, the reasons were in the following categories.

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 purposely 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 Professor Anderson’s institution, consistently lower scores for potential for adoption have been 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 in porting the results to other environments.

6 GTA Workshops For Faculty

Outside of classrooms, GTA-related workshops had been offered for interested faculty members at regional (e.g., [41]) and national (e.g., [42]) conferences, and at institutions internationally (e.g., [43, 44]). These workshops guided faculty to develop game-themed applications based on tutorials from the GTA modules. Images in Figure 6 are screen shots of a “Worm-like” and a “Pizza Delivery” games resulting from the second day of multi-day version of the workshops (e.g., [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.

3Refer to http://faculty.washington.edu/ksung for workshop lecture notes.
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, though they found the GTA materials to be non-trivial, faculty participants with no prior expertise in graphics/games were able to comprehend and begin developing game-like applications within a matter of hours.

7 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 the 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 (e.g., [47]) or continual student successes (e.g., [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. These were well established courses producing many successful alumni in advanced CS courses and in the industry [32].

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 and two GTA modules in combinations which varied from class to class. This verified that a faculty

---

4The involved instructor has no background in graphics/games.
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 using the GTAs.

8 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.

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 the table below:

<table>
<thead>
<tr>
<th></th>
<th>GTA</th>
<th>Console</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS1</td>
<td>72% to 76%</td>
<td>65%</td>
</tr>
<tr>
<td>CS2</td>
<td>86%</td>
<td>79%</td>
</tr>
</tbody>
</table>

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 above 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.
Assignment Scores

Figure 7 plots the average scores on assignments for all of the six assignments. The left bar above each assignment displays the results from GTA while the right is from the console version. The scores from GTA were consistently better than the corresponding console version. From the written feedback (to be detailed later), it was clear that students spent more time playing with the GTA assignments and thus resulted in 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 were small. By the end of CS1, the assignments became more challenging (CS1-D, 2D array), and the difference in scores also became more prominent (89% vs. 58%). As widely recognized (e.g., [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% vs. 81% for CS2E, and 82% vs. 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.

Self-Reported Time Spent on Assignments

Figure 8 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

---

5Note that each assignment has a 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.
time they 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.

**Per-Assignment Survey**

After students handed in their assignments and before they received their grades, the exact same questions were asked to all students: clarity and difficulty of the assignment, amount they have learned from the assignment, if completing the assignment made them feel satisfied, and if the assignment was interesting.

Figure 9 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 Figure 9 showed CS2 students found the console assignments easier to understand, slightly more challenging, and, contrary to intuition, they found that working on console assignments more satisfying and that the console assignments were more interesting.

Fortunately, written feedback from students helped explain the above 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 (e.g., 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.

Pre- and Post-Course Survey

Course survey forms were designed to understand students’ background, self-perception, interests, and general attitudes towards the CS discipline. Students completed these forms during first day of class (pre-course) and before the final exams (post-course). The tables of Figure 10 shows 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 Figure 10 was not statistically significant. This exception was the “Well prepared” question for CS1 students (left table of Figure 10). In this case, when students were asked after the class was over, “in hindsight, how well do you think you are 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 Figure 7) and the indifference attitude from the left of Figure 9, 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.

\(^6\)Students had the option to not participate in these surveys.
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 on student learning. As mentioned, the instructor did not have prior background in computer graphics/games and purposely avoided 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 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].

9 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 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 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 of 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.
Acknowledgements

First and foremost thanks to all BIT 142 and 143 students from Cascadia Community College for working with the chaotic schedules and draft versions of the assignments, and to Lee 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. This work is supported in part by Microsoft External Research under the Computer Gaming Curriculum in Computer Science RFP, Award Number 15871 and 16531. All opinions, findings, conclusions, and recommendations in this work are those of the authors and do not necessarily reflect the views of Microsoft.

References


