Game-Themed Programming Assignments: The Faculty Perspective *

Kelvin Sung
University of Washington, Bothell
ksung@u.washington.edu

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

Scott Wallace
Washington State University, Vancouver
wallaces@vancouver.wsu.edu

Ruth Anderson
University of Washington, Seattle
rea@cs.washington.edu

John Nordlinger
Microsoft Research
johnnord@microsoft.com

ABSTRACT

We have designed and implemented game-themed programming assignment modules targeted specifically for adoption in existing introductory programming classes. These assignments are *self-contained*, so that faculty members with no background in graphics or gaming can *selectively pick and choose* a subset to combine with their own assignments in existing classes. 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 our design, implementation, and assessment efforts. Our result is a road map that guides faculty members in experimenting with game-themed programming assignments by incrementally adopting/customizing suitable materials for their classes.

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

General Terms: Design, Experimentation

Keywords: CS1/2, Games, Programming Assignments, Adaptation

1. INTRODUCTION

Encouraged by the recent successes (e.g., [38, 42]) in teaching computer science (CS) concepts based on programming computer games, we have experimented with and accomplished modest successes in teaching computer graphics concepts based on building interactive computer games [37]. Inspired by the positive feedback from students and local employers, we started investigating potential approaches for integrating games programming in other CS courses.

Because of the relatively straightforward assignments and the abundant, favorable results (e.g., [8, 27]) we began our investigation with introductory programming courses (CS1/2). Our initial intent was to study and adapt from existing approaches. However, as discussed in the next section, most existing results are exploratory, and do not facilitate adoption by time-constrained faculty members. This is especially true for those faculty members who are primarily interested in limited experimentation and incremental changes to their existing well-established CS1/2 curricula.

In 2006, we began our investigation into designing/developing CS1/2 game-themed programming assignments specifically targeted for *selective* adoption and limited-scope experimentation by faculty with no gaming/graphics experience [36]. Our work follows the observation that the modern generation of students demands multimedia experiences, i.e., real time *interaction* in a *visual* environment [21]. Because existing results demonstrating that students' needs can be met (e.g., [9]) and that the learning outcomes can be achieved (e.g., [8]), our work focuses on the needs of the faculty. Our project consists of two phases. First, develop materials that are suitable for adoption. Second, adopt the materials in existing courses and study the following: faculty effort; student learning outcome; and student engagement.

We have completed the first phase and started on the second phase of our project. In the first phase of the project, under the assessment of an independent, external faculty member, we have developed a set of *self-contained* game-themed assignment modules, a simple library, and a set of detailed tutorials. We expect faculty members to *selectively pick and choose* from the game-themed assignments to combine these with their existing assignments, or to design/develop their own game-themed assignments based on the provided tutorials/library. In the rest of this paper, we describe our efforts and our results from the first phase of the project.

Our assignments are simple "real-time interactive graphics programs". Strictly speaking, these programs do not qualify as "games" because they have unknown entertainment values. However, in our current implementation, the programs run on both PCs and the *XBOX 360* gaming console. As such, we use the term: "game-themed".

2. GAMES AND CS1/2 CLASSES

While there are many types of "games" that are suitable for teaching CS subjects (e.g., non-computer games [15]) our focus is interactive graphical computer games. As discussed in [37], recent work in this area can be classified into: *game development* (e.g., [32]) where students learn about building games; *game programming* (e.g., [38]) where students study games related algorithms; and *game client* (e.g., [37]) where

* This work is supported in part by the National Science Foundation grant DUE-0442420 and Microsoft Research under the Computer Gaming Curriculum in Computer Science RFP, Award Number 15871. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or Microsoft.

SIGCSE'08, March 12–15, 2008, Portland, Oregon, USA.

students learn about CS concepts via games. Integrating games into CS1/2 classes belongs to the *game client* category because after these classes students are expected to understand abstract programming concepts and not about building games.

When examining existing work that integrates games into CS1/2 courses, we observe three broad approaches according to the efforts from the faculty in preparing the programming assignments:

1. Few or no game programming assignments (e.g, [19, 22]). In these courses students learn by *playing* custom games but they do not actually program the games.

2. Per-assignment game development [17, 25, 9, 24, 35, 29, 39, 8]. All these classes developed games as part of individual programming assignments. These games are typically designed around the technical topics being studied.

3. Extensive game development. For example, faculty must develop and/or design programming assignments based on game engines suitable for general games development (e.g., [10]), game engines specifically designed for a course (e.g., [28]), specialized programming environments (e.g., [26]), custom object-oriented class hierarchies (e.g., [28]), specific curricula (e.g., [27]), or new programming languages (e.g., [14]).

Results from the last approach typically involve large amounts of adoptable/adaptable courseware materials. However, using these materials usually requires significant investments, e.g., understanding a games engine, etc. Because of the considerable overhead, results from the third approach are not suitable for *selective* adoption.

Ideally, results from the per-assignment game development approach are best suited for *selective* adoption. For example, one would selectively replace non-game assignments in existing classes with the corresponding game assignments. However, because of the pioneering nature, many of the results are "anecdotal", and without holistic considerations with respect to the CS1/2 curriculum. For example, [24] only involves turn-based strategic games, [35] only discuss puzzle games, and the discussion in [39] is based on one single game. The systems described by [25, 29] are designed specifically for an *objects-first* approach. While results from [9] are based on an ascii character game, and the excellent ideas presented in [9, 8] are without implementations.

3. DESIGN CONSIDERATIONS

Based on the above survey, we articulate the following considerations as guidelines for designing game-themed programming assignments targeted for selective adoption.

- *Institutional Oversight*: Departmental committee consensus is often required for significant changes to the core courses. Because of the potential impact, it can be especially challenging to arrive at consensus for modifications to introductory level courses like CS1/2. However, limited scope experimentation by individual faculty is usually acceptable. Our assignment modules are limited in scope and self-contained, to facilitate selective experimentation by individuals. In this way, faculty members can gain experience (and collect results) to assist the committee's decision making process.

- *Faculty Experience*: Many faculty did not grow up playing computer games and most are not familiar with graphics programming. To facilitate adoption, game-themed assignment modules should be skeleton applications with complete graphics and user interaction functionality. In this way, faculty and students only need to concentrate on the *missing core CS concepts*. In addition, tools and support should be provided for the interested faculty to develop their own game-themed assignments. It is important that the provided tools define the proper abstractions to hide the details of real-time graphics programming.

- *Material Neutrality*: As with any powerful tool, inappropriate use of games can backfire and result in further alienation of under-represented groups [40]. It is important that the materials built are *gender* and *expertise* neutral. For example, it is important to avoid violence (shooting), and unnecessary competitions [2, 20, 31]. The materials should discourage superfluous graphics and user interaction programming.

- *Infrastructure Support*: *Free* and *simple* are the keywords here. Given the financial reality, all materials must be freely available; the associated infrastructure requirements must be modest and straightforward.

- *Academic Integrity*: Since our work focuses on the needs of the faculty, the discussion and analysis have been centered around the perspective of the adopting faculty. It is important to remind ourselves that ultimately, the goal of this work is to facilitate students' learning about core CS concepts. Any dilution, in favor of even acquiescing to their motivation as game developers, would do the students a disservice.

4. DESIGN AND IMPLEMENTATION

Our efforts are guided by three principles. (1) *Neutrality*, with respect to both gender and programming experience. The importance of this cannot be overstated, as we cannot afford to alienate under-represented groups. (2) *Simplicity*, with respect to resource requirements (as opposed to being technically *easier* for the students). Simpler is better, unless the instructional integrity of the faculty member's existing course materials is an issue. (3) *Modularity*, meaning that each assignment module should be completely self-contained, and should include sufficient supporting materials that faculty can consider the adoption of each assignment in isolation.

Based on these principles, this section discusses our design decisions and describes our implementation efforts. All materials mentioned in this and next sections are available on-line at [1].

4.1 Implementation Platform

There has been work done in integrating the tools required for building interactive graphical computer games into introductory programming courses, including event handling (e.g., [13]), graphical user interfaces (GUI) (e.g., [33]) and graphical application programming interfaces (API) (e.g., [30]). In our case, to maintain *simplicity*, we are interested in *hiding* these aspects of games programming. We need a platform that *transparently* integrates all of the above tools so

that faculty and students do not need to be aware of their existence.

We have chosen the C# programming language and the Microsoft XNA framework [4]. Our choice is governed primarily by the fact that C#, XNA, and Microsoft's Games Studio Express combination is the only publicly and freely available solution that provides seamless integration of the development environment, programming language, GUI API, and graphics API[1].

A very important benefit of our choice is the fact that XNA based programs can run on both the PC and the XBOX 360 platforms. With our secondary goal of developing an introductory games programming course (e.g., [22]), this benefit is of special importance.

4.2 The Assignment Modules

The first decision is the technical topics for the assignments. Becker [9] and Bayliss [8] are excellent references for ideas that relate fundamental programming concepts to existing commercial video games. In our case, we must be concerned with any unnecessary complexity; we need simple interactive graphical applications that assist students in implementing the relevant programming concepts. At the same time, we must ensure that our assignments are *interchangeable* with those of *typical* CS1/2 courses.

We have taken a *reverse* adoption strategy and adopted the technical topics for the game-themed assignments from the console-assignments in our existing CS1/2 courses [3]. There are several advantages to this strategy. (1) Our existing CS1/2 courses are well-established with many batches of 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 perfect vehicle for the second phase of our project, where we can simply replace corresponding assignments and study the effects. (3) The console-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 version and a game-themed version. The console version of the assignment is an excellent and familiar reference for faculty members unfamiliar with games programming. The next section will describe the assessment procedure that ensures that the console and game-themed versions of the assignment are *equivalent*.

Within its particular technical topic, 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 of prerequisites and learning outcomes, a sample pre- and post-test, a sample grading rubric, sample solutions for both console and game-themed versions, frequently asked questions, and an *implementation tutorial*. The implementation tutorial is a step-by-step guide on the implementation of the game-themed assignment. For interested faculty, this tutorial demonstrates how they can implement game-themed assignments on their own.

For the students, each module includes a description of the assignment, and a skeleton starter project. The game-themed starter project is a game-like application where all necessary graphics and user interactions are provided. Students work with the starter project to *fill-in* the missing, relevant, core CS concepts to complete each assignment. In this way, the assignments can maintain *gender neutrality* and *programming experience neutrality*, because students are *not* expected or encouraged to develop any games from scratch.

We have implemented 6 assignment modules. We plan to use these to replace the corresponding assignments in our existing CS1/2 courses in Phase 2 of our project. The 6 modules cover topic areas that include integer division and the modulus operator, random number generation, inheritance, 2D arrays, class hierarchy/inheritance, linked lists and queues, and arrays of object references. After the discussion of the assessment procedures in the next section, Section 6 presents two of the assignments in detail.

4.3 The Supporting Library

All game-themed assignments are implemented based on a simple 2*D* library. This library is designed to support faculty members interested in developing their own game-themed assignments. To maintain simplicity, the library consists of a single superclass for assignments, a handful of drawing functions, and an instance of the game controller object for handling input.

```
// Label A: AssignmentBase Subclass responsibilities:
    public Constructor (...) // construct and define drawing dimension
    protected override void InitializeWorld() // called once at init time
    protected override void UpdateWorld()     // called every 25 msec
    protected override void DrawWorld()       // called every 25 msec
// Label B: Output drawing support
    void EchoToBottomStatus(String msg)
    void EchoToTopStatus(String msg)
    void DrawCircle(Vector2 at, float radius,...String imageFile)
    void DrawRectangle(Vector2 center,...dimension...String imageFile)
// Label C: Input support
    XnaAssignmentBase.GamePad. .... // poll object for state of input device
```

The above listing shows all the functions in our library. With this library, the only requirement for faculty to develop real-time graphical applications is an understanding of the *UpdateWorld* function (which updates game state) and the *DrawWorld* function (which draws all graphical primitives). Under Label **B** are the output drawing functions. Notice that, besides text output, the only supported graphical primitives are squares and circles. Our design agrees with results from previous approaches, which indicated that only simple primitives are required for CS1/2 courses [34]. The *GamePad* object under Label **C** contains instances of relevant *button* and *trigger* objects that reflect the state of the input device. Associated with this library is a step-by-step tutorial explaining each of the above functions within a simple interactive graphical application.

The simplicity of the library is noteworthy, as this simplicity enables a wide range of uses. There is no graphical class hierarchy or any utility functions (e.g., collision detection). Our design decision is based on experiences from building graphics libraries [37]. We have learned that clean and efficient utility functions enforce requirements on graphical objects and the complexity can increase rapidly resulting in elaborate class hierarchies. As discussed in recent results (e.g., [12, 33]), such hierarchies may not be well suited for supporting different approaches [11] to teaching CS1/2 classes. Moreover, many games (e.g., Reversi [39]) do not require any additional functionality from our library. Our goal is to facilitate experimentation by novice faculty members, and so simplicity is the key.

[1] At the time when the project started, our main reservation with a Java based approach was the prospect of working with the separate Java3D library.

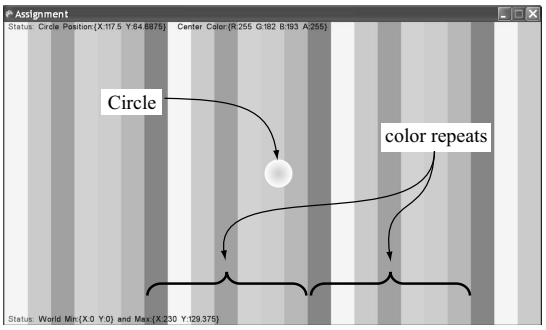

Figure 1: The Operator Game.

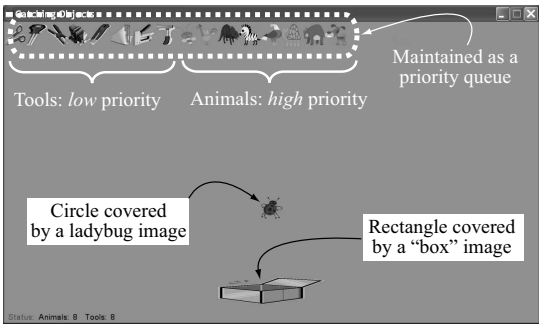

Figure 2: Catch Objects Game.

5. ASSESSMENT

Because we are developing "game-themed" applications for teaching concepts, it is important to have an independent evaluator assessing the integrity of the materials. In addition, the technical equivalence between the console and game-themed assignments must be verified. Professor Ruth Anderson is our 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. She has won multiple teaching awards (e.g., [5]) and is active in CS education research (e.g., [7, 23]). In addition to these excellent credentials, Professor Anderson is perfectly suited for evaluating our 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. During the project, we avoided in-person or verbal communications to maintain impartiality, and to simulate investigations by curious faculty. The actual assessment was performed across our project web-site [1] where newly released materials were downloaded, examined, and tested. Feedback was posted via a custom evaluation form.

Our evaluation form is designed to collect both formative feedback and quantitative scores [18]. Each assignment module is assessed in two areas. (1) *Quality of the assignment*, which focuses on the technical equivalence between the console and game-themed assignments, and the supporting materials (e.g., pre/post test, etc.). (2) *Potential for adoption*, which focuses on the factors that may prevent adoption of high quality assignments.

Based on the review feedback, the assignments were well received overall. We believe that because the assignments were *reversely* adopted, technical merits were never an issue. Professor Anderson agrees with us that the assignments are appropriate for *typical* CS1/2 courses. Assignments with low *quality of assignment* scores were revised and re-assessed. This process continues until the formative comments are positive and the numeric scores are above 4 (out of 5). As expected, all of the low scores were caused by game-themed assignments. Typically, the reasons are in the following categories.

1. **Differences in difficulties**: initial game-themed assignments are often too difficult, complex, or intimidating. Based on the feedback, we have adjusted the assignments accordingly.

2. **Inappropriate use of concept**: designing assignments around *negative* results from game play is a bad idea. For example, a linked list structure is invoked for bookkeeping 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. This problem was remedied with keyboard simulation.

Finally, because Java is the language of choice at Professor Anderson's institution, we were not surprised when we received consistently lower scores in the *potential for adoption* area. We are fully aware that the language issue must be addressed for wide adoption of our results. Now that we have experience building these assignments, and understand the important attributes of the library, we are investigating the possibility of porting our results to other environments.

6. EXAMPLE ASSIGNMENTS

As mentioned, the game-themed assignments are simple "interactive graphics applications". This section uses two of the assignments to demonstrate the general flavor of the programs.

Example 1: Assignment on integer arithmetic. This is designed to be the very first CS1 assignment. Figure 1 shows the game-themed version of the assignment. In this case, the user can control the horizontal position of the circle in the front. 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 circle, such that as the circle is moved horizontally its color always reflects that of the rectangle underneath it. The console version of the assignment is a simple character based flash-card quiz program. In both cases, the solution consists of 10's of lines of C# code.

Example 2: Assignment on linked lists based queues. In this assignment, students must maintain a priority queue. The priority queue consists of two linked list based queues with "high" and "low" priorities. Figure 2 depicts the game-themed version. In this case, the user can insert either high (animals) or low (tools) priority objects into the priority queue located at the top of the window. A high priority object would be enqueued to the front queue, and low priority object to the back queue. The game continuously dequeues

and drops objects from the priority queue. The user moves the box to *catch* the dropping object. The console 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 queue respectively. Similar to the game-themed version, the retrieval of requests is a simple dequeue operation on the priority queue. The skeleton starter projects for both versions contain all necessary I/O functionality: graphics/GUI for game-themed, and character I/O for console. In both cases, students only need to implement the linked list queue and the priority queue. The solutions are almost identical involving about 200 lines of C# code.

Based on these assignment modules, we have begun the second phase of our project. Since both versions of any given assignment cover identical technical topics, we are able to adopt the game-themed version transparently. We are interested in verifying that the game-themed assignments do not require extra resources from the faculty, that they can achieve similar student learning outcomes, and that they can increase student interest and engagement. We plan to demonstrate students' assignments on the XBOX 360 consoles in our high school recruitment trips.

7. REFERENCES

[1] XNA-based assignment home page: http://depts.washington.edu/cmmr/Research/XNA_Games.

[2] In J. Cassell and H. Jenkins, editors, *From Barbie to Mortal Kombat: Gender and Computer Games*. MIT Press, 1998.

[3] BIT142/143: Intermediate programming and data structure, 2007. Cascadia Community College.

[4] XNA game studio, 2007. Microsoft Inc, *http://msdn2.microsoft.com/en-us/directx/Aa937794.aspx*.

[5] R. E. Anderson, ACM Faculty Award, *Dept. of CS, U. of Virginia*, 2004.

[6] J. Alm, R. Baber, S. Eggers, C. O'Toole, and A. Shahab. You'd better set down for this!: creating a set type for cs1 & cs2 in c#. In *ITiCSE '02*, PP. 14-18, 2002.

[7] 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.*, 39(1):69–73, 2007.

[8] J. D. Bayliss and S. Strout. Games as a "flavor" of cs1. In *SIGCSE '06*, PP. 500-504, 2006.

[9] K. Becker. Teaching with games: the minesweeper and asteroids experience. *J. Comput. Small Coll.*, 17(2):23-33, 2001.

[10] 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 *SIGCSE '06*, PP. 354-358, 2006.

[11] K. B. Bruce. Controversy on how to teach cs 1: a discussion on the sigcse-members mailing list. *SIGCSE Bull.*, 37(2):111-117, 2005.

[12] K. B. Bruce, A. Danyluk, and T. Murtagh. A library to support a graphics-based object-first approach to cs 1. In *SIGCSE '01*, PP. 6-10, 2001.

[13] H. B. Christensen and M. E. Caspersen. Frameworks in cs1: a different way of introducing event-driven programming. In *ITiCSE '02*, PP. 75-79, 2002.

[14] W. Dann, S. Cooper, and R. Pausch. *Learning to Program with Alice*. Prentice Hall, 2006.

[15] P. Drake. *Data Structures and Algorithms in Java*. Prentice Hall, 2006.

[16] J. Ernie Giangrande. Cs1 programming language options. *J. Comput. Small Coll.*, 22(3):153-160, 2007.

[17] N. Faltin. Designing courseware for active learning with virtual board games. In *ITiCSE '99*, PP. 135-138, 1999.

[18] J. Frechtling, L. Sharp, and Ed. *User-Friendly Handbook for Mixed Method Evaluations*. Directorate for Education and Human Resources, Division of Research, Evaluation and Communication, National Science Foundation, 1997.

[19] R. Giguette. Pre-games: games designed to introduce cs1 and cs2 programming assignments. In *SIGCSE '03*, PP. 288-292, 2003.

[20] D. Gürer and T. Camp. An acm-w literature review on women in computing. *SIGCSE Bull.*, 34(2):121-127, 2002.

[21] M. Guzdial and E. Soloway. Teaching the nintendo generation to program. *CACM*, 45(4):17-21, 2002.

[22] P. Haden. The incredible rainbow spitting chicken: teaching traditional programming skills through games programming. In *ACE '06: Austalian conference on Computing education*, PP. 81-89, 2006.

[23] T. B. Horton, R. E. Anderson, and C. W. Milner. Work in progress-reexamining closed laboratories in computer science. Proc. FIE 2004, 2004.

[24] T. Huang. Strategy game programming projects. In *CCSC-NE '01*, PP. 205-213.

[25] R. Jiménez-Peris, S. Khuri, and n.-M. Marta Pati˙Adding breadth to cs1 and cs2 courses through visual and interactive programming projects. In *SIGCSE '99*, PP. 252-256, 1999.

[26] M. Külling and P. Henriksen. Game programming in introductory courses with direct state manipulation. In *ITiCSE '05*, PP. 59-63, 2005.

[27] S. Leutenegger and J. Edgington. A games first approach to teaching introductory programming. In *SIGCSE '07*, PP. 115-118, 2007.

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

[29] T. Lorenzen and W. Heilman. Cs1 and cs2: write computer games in java! *SIGCSE Bull.*, 34(4):99-100, 2002.

[30] S. Matzko and T. A. Davis. Teaching cs1 with graphics and c. In *ITICSE '06*, PP. 168-172, 2006.

[31] M. J. Natale. The effect of a male-oriented computer gaming culture on careers in the computer industry. *SIGCAS Comput. Soc.*, 32(2):24-31, 2002.

[32] I. Parberry, T. Roden, and M. B. Kazemzadeh. Experience with an industry-driven capstone course on game programming. In *SIGCSE '05*, PP. 91-95, 2005.

[33] V. K. Proulx, J. Raab, and R. Rasala. Objects from the beginning - with guis. In *ITiCSE '02*, PP. 65-69, 2002.

[34] E. S. Roberts. A c-based graphics library for cs1. In *SIGCSE '95*, PP. 163-167, 1995.

[35] J. M. Ross. Guiding students through programming puzzles: value and examples of java game assignments. *SIGCSE Bull.*, 34(4):94-98, 2002.

[36] K. Sung. Xna based games-themed programming assignments for cs1/2. *Microsoft Research, Computer Gaming Curriculum in Computer Science, Award Number: 15871*, 2006-2008.

[37] K. Sung, P. Shirley, and B. R. Rosenberg. Experiencing aspects of games programming in an introductory computer graphics class. In *SIGCSE '07*, PP. 249-253, 2007.

[38] E. Sweedyk, M. deLaet, M. C. Slattery, and J. Kuffner. Computer games and cs education: why and how. In *SIGCSE '05*, PP. 256-257, 2005.

[39] D. W. Valentine. Playing around in the cs curriculum: reversi as a teaching tool. *J. Comput. Small Coll.*, 20(5):214-222, 2005.

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

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

[42] U. Wolz, et. al. Digital gaming as a vehicle for learning. In *SIGCSE '06*, PP. 394-395, 2006.