Dream Coders: A 2D Role Playing Game For Teaching Introductory Programming Concepts

Jack Keng-Wei Chang, Long Hoang Dang, Jebediah Pavleas, Joseph F. McCarthy, Kelvin Sung
University of Washington, Bothell
Bothell, WA 98011
{wei0831 | se3d | jebp | joemcc | ksung}@uw.edu

Jason Bay
Glu Mobile Inc.
Kirkland, WA 98083
Jason.Bay@glu.com

ABSTRACT
The Dream Coders Project uses a video game to facilitate teaching and learning of computer programming concepts. The project is designed to accomplish two goals. First, to produce a collection of game-based assignments that faculty can use to teach introductory programming concepts, even if they have no background in computer graphics or video game development. The faculty should be able to adopt any of the assignments and integrate them into their existing classes with minimal overhead. Second, to develop a framework for overcoming the typical shortcomings of using video game-based assignments in introductory programming classes: they are typically "*pseudo*" games that are neither fun nor playable outside of class assignments. The Dream Coders Project is designed to be a traditional top-down Role Playing Game (RPG) where students play the role of the hero, and can explore and extend a fantasy environment that is incomplete in functionality. Through a series of programming assignments, students can add functionality and increase the playability (and fun) of the game. The programming assignments are independent from one another, so that the faculty can pick and choose specific ones to integrate into their existing courses. This paper describes the Dream Coders Project from two perspectives: first, the paper discusses how the project was implemented as a 10-week-quarter undergraduate independent study course; second, the paper describes the process, schedule, and results from the project. As an independent study course, the project was a success. We were able to deliver a completed RPG game with multiple "quests" integrated as programming assignments. The paper concludes with lessons learned from this project and a discussion of the pros and cons of using a complete playable game for teaching introductory programming concepts.

Categories and Subject Descriptors: K.3.2 [**Computer and Information Science Education**]: Computer Science Education.

Keywords: CS1/2, Introductory Programming Classes, Video Games, RPG.

1. INTRODUCTION

Games and other playful or fun activities often enhance educational effectiveness by providing intrinsic motivation for students to learn. It is well understood that we can engage the younger generation with computers by taking advantage of their familiarity and interests for video games (e.g., [9]). In the context of computer science education, video games have long been integrated into many formal classes [20]. This is especially the case for introductory programming classes, where because these are usually the first computer science courses students encounter, the video games related materials can build excitement and enthusiasm for the discipline [15].

Teaching introductory programming classes based on video games can increase student engagement and achieve desired learning outcomes [4, 5, 18]. For faculty members without a computer graphics or game development background, adapting to video game programming can be a daunting task. To date, only the Game-Themed Introductory Programming Project [1] is conceived specifically to address this issue by providing video game-like Game-Themed Assignments (GTA) [22] and Game-Themed Instructional (GTI) modules [2]. Faculty with no background in graphics or games can selectively adopt GTA and GTI modules to integrate into their existing classes while gradually acquiring expertise in video game development [21]. While effective as engaging educational tools, GTA and GTI modules are designed to be simple and modular. These tools generally lack the fun and playability attributes of typical video games [11].

In order to continue engaging students outside of formal classroom settings, teaching materials must exhibit *typical* video game characteristics, e.g., game character development and skill improvement. The Dream Coders Project was conceived as an attempt to develop a 2D Role Playing Game (RPG) that itself is a *programming assignment framework*. The project aims to build an RPG that is complete and playable but missing elements of functionality that students can fill in through completing programming assignments, e.g., implementing file input and output functions to fill the inventory of a shop so the hero can purchase items. The design and development of the Dream Coders game were implemented as an undergraduate independent study course that is part of an ongoing effort to transform traditional introductory game development courses into a “Serious Game” development course that challenges students to build *interactive graphical applications* to address real-world problems [3].

This paper presents the Dream Coders Project. The paper first presents a survey of related work done in the area to frame our work in a disciplinary context. The paper then describes the 10-week process of developing the Dream Coders Project over an academic quarter. The details of the Dream Coders game and the programming assignment framework are then discussed. We conclude with an evaluation of the project and our results.

2. BACKGROUND

Previous work on integrating video games into introductory programming courses can be classified into three broad approaches, based on the different kinds of effort required of faculty:

1. Little or no game programming (e.g., [8, 10]) where students learn by playing and do not actually program any games.
2. Per-assignment game development (e.g., [12, 19, 23, 5, 22]) where faculty develop isolated games as part of individual programming assignments, and challenged students to implement specific technical topics being studied.
3. Extensive game development where faculty design programming assignments based on custom libraries (e.g., [24]), general or dedicated game engines (e.g., [6, 16]), or special programming environments (e.g., [13]), curricula (e.g., [14]), or new programming languages (e.g., [7]).

Much of this prior work has successfully demonstrated an increase in student engagement and improvement in learning outcomes (e.g., [7, 5, 14]). However, it is also the case that most of the existing work is based on pioneering exploratory projects by faculty members who have expertise in computer graphics and game development. With few exceptions, these projects are designed to study student engagement and various learning outcomes; adaptability and generality of the resulting materials are usually not main concerns. For faculty members teaching introductory programming courses, most of whom do not have computer graphics or game development backgrounds, it can be challenging to take advantage of these results.

The *Game-Themed Introductory Programming Project* at the University of Washington Bothell [1] is designed specifically to address these issues. With carefully designed video game-like programming assignments [22] and instructional modules [2] that demand no existing knowledge of graphics or games, faculty can gradually integrate video games into their existing courses while acquiring their own expertise in developing video games. While this project has been successful in engaging students and developing the faculty [21], student classroom survey results reveal that the existing game-themed materials lack fun and do not encourage creativity [11]. More importantly, outside of formal classroom settings, these materials do not motivate students to "*play with*" or otherwise further develop or practice their programming skills.

Students need to have access to a *complete* game that they can experiment with and be challenged to improve upon. Faculty members who are interested in incorporating games into their classes but do not have graphics or game backgrounds need to be able to integrate selective video game materials without being required to substantially alter their entire course curricula. These are the goals for the Dream Coders Project: a complete RPG game with missing functionality where faculty can choose to integrate any part of the system into their existing classes and students can fill in the gaps of the game system through working on programming assignments.

3. PROCESS

The Dream Coders game development schedule follows an established process of integrating serious game development with user-centered agile software design in an academic quarter [3], where the 10-week time frame is divided into four sprints: a 3-week pre-production design sprint, 2-week alpha and 3-week beta production sprints, and a 2-week final testing sprint. To ensure proper background research, more than 30 Computer Science faculty members from the Pacific Northwest Region were polled for their introductory programming course assignments before the project began, in order to learn more about the concepts that were incorporated into the assignments. The collected sample programming assignments formed the foundation of the challenge: design and build an RPG game with missing functionality that can be filled in with solutions to programming assignments that exercise these core concepts. The Java programming language was selected because it is the most widely adopted language based on our poll.

The project team consisted of 7 undergraduate programming students, and an external professional game designer. All students had previously taken an introductory game development class. During the design and initial development sprint, the team met twice weekly. The meeting frequency decreased to once a week after the project went into an intense development phase. In each meeting students would present the progress they've made for the week, the problems they have encountered, possible solutions to these problems and their goals for the following week. A variety of online collaboration tools were used to bridge the gaps between the limited face-to-face meeting times, including: Appache Subversion (SVN) for source code control, Dropbox for sharing art assets, Google Docs for collaborative documentation (e.g., recording meeting minutes and creating a design doc), and FogBugz for bug tracking. The team met with the professional game designer once every other week for general guidance and feedback.

During the design sprint, students brainstormed game narration, mechanics, and incentives. At the same time, students also studied the collected sample programming assignments, mapped out the programming concepts areas (e.g., conditional statements, 2 dimensional arrays, file I/O), and brainstormed approaches to include missing functionality that could be filled in with programs developed based on the relevant concepts.

With guidance and advice from the professional game designer, the team decided on a quest-based RPG game where specific programming concepts must be implemented before each quest can be completed. For example, a quest to locate a map that is locked in a chest which can only be unlocked by students writing a function to sort three random numbers. Based on this design, the team identified five key programming concepts: a first CS1 assignment (simple typing), conditionals, loops, file I/O, and arrays, and began working on separate quests in parallel based on each of these concepts.

4. The Dream Coders RPG Game

Figure 1: Dream Coders Opening Sequence (*Credit: Alisa Lew*)

As illustrated in Figure 1, the Dream Coders story narration begins with a student who falls asleep in a programming class and, unable to wake up from the dream, becomes the hero of the game. The student's dream takes him[1] to a little town where he must interact with the townsfolk to find a way to wake up back into the real world. He soon discovers that he must conquer many quests given by the townsfolk to free himself from his dream.

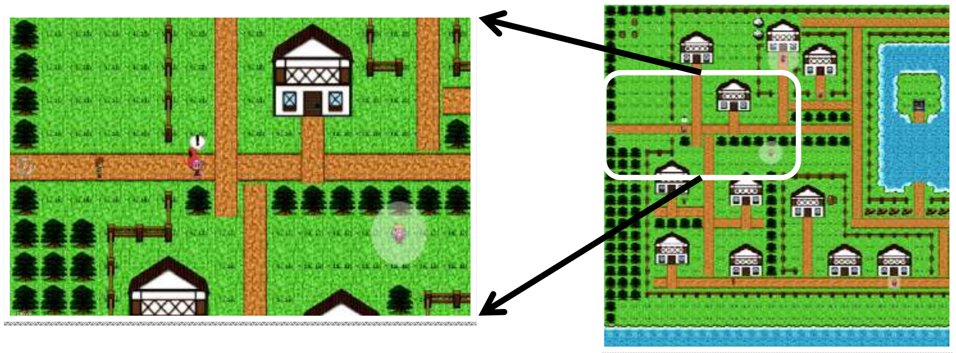


Figure 2: First Quest and The Little Town

Figure 2 shows the beginning of the first quest where the hero of the game first wanders into The Little Town. Being the very first programming assignment, students can complete this quest by singing a song (typing the lyrics with print statements) to calm a little girl that the hero encounters.

[1] Throughout the paper, to simplify our description, we will use masculine pronouns such as "he" and "him" to refer to a student, though we note that our approach can be used by male or female students.

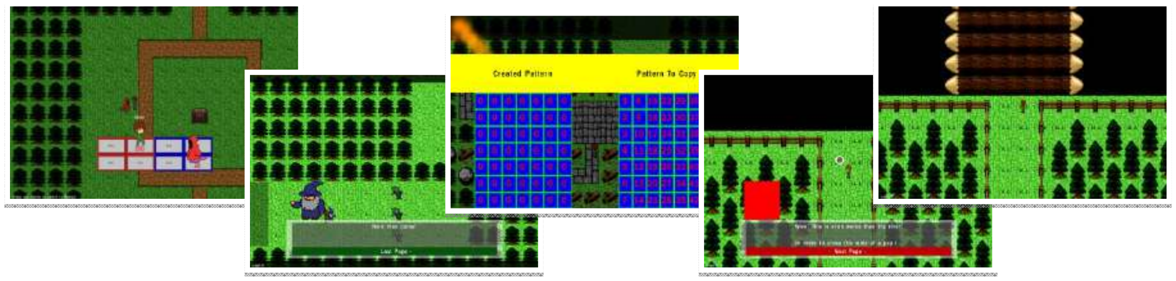
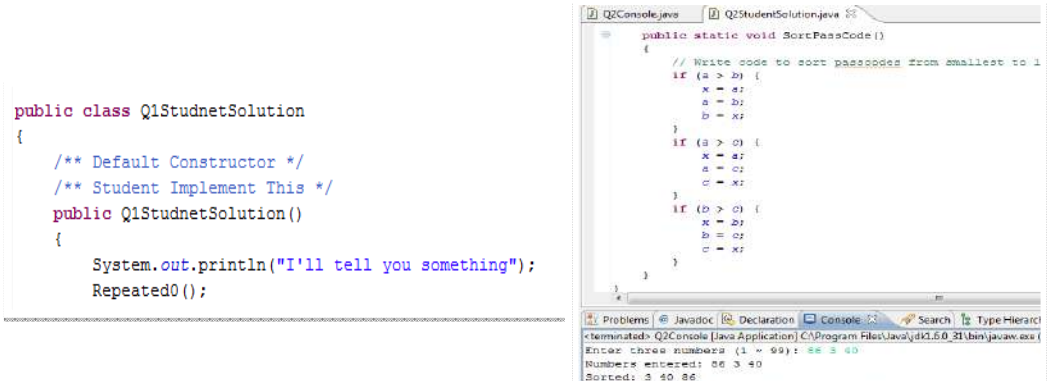


Figure 3: Screen Shots from different Quests

Figure 3 shows the screen shots from some of the other available quests such as:

- sort three passcodes to unlock a chest,
- loop over patterns to build a bridge,
- traverse arrays to fight monsters, and
- implement file I/O to stock an empty shop.


```
public class Q1StudnetSolution
{
    /** Default Constructor */
    /** Student Implement This */
    public Q1StudnetSolution()
    {
        System.out.println("I'll tell you something");
        Repeated0();
```

```
    public static void SortPassCode()
    {
        // Write code to sort passcodes from smallest to l
        if (a > b) {
            x = a;
            a = b;
            b = x;
        }
        if (a > c) {
            x = a;
            a = c;
            c = x;
        }
        if (b > c) {
            x = b;
            b = c;
            c = x;
        }
    }
}
```

```
<terminated> Q2Console [Java Application] C:\Program Files\Java\jdk1.6.0_31\bin\javaw.exe (
Enter three numbers (1 ~ 99): 86 3 40
Numbers entered: 86 3 40
Sorted: 3 40 86
```

Figure 4: Quest Solution and Console Mode

The Java code fragment on the left side of Figure 4 shows that solutions to the quests are in the form of completing function prototypes with comments indicating the code that needs to be written by the student. In this case, the `Q1StudentSolution()`function was initially empty and students must print out lyrics by making the appropriate `System.out.print()`function calls. The Dream Coders game system is unable to determine the correctness of students' implementations, so faculty must still grade the functions as usual. To facilitate faculty with no graphics or game development backgrounds, and to support rapid interactive debugging cycles without invoking the graphical game system, a traditional text-based console testing environment can be used for all the quests. In Figure 4, the code fragment to the right uses the testing of the passcode sorting quest to show that text-based console mode testing is similar to the

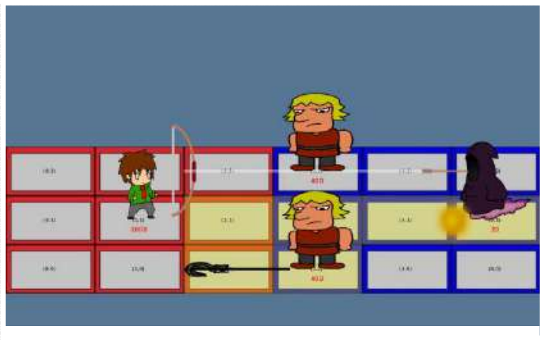


Figure 5: The Battle System

traditional programming paradigm. In game mode, each passcode can only be obtained after the hero defeats a monster.

Figure 5 shows the Dream Coders battle system where the hero movement and weapons-deploy sequence can all be controlled by the player (student) either interactively, or via Java code. The player maneuvers the hero on the grid systems to the left battling the monsters on the right side. The customizable grid resolution offers a straightforward platform for defining quests based on arrays. This battle system is the main mechanism for advancing the game.

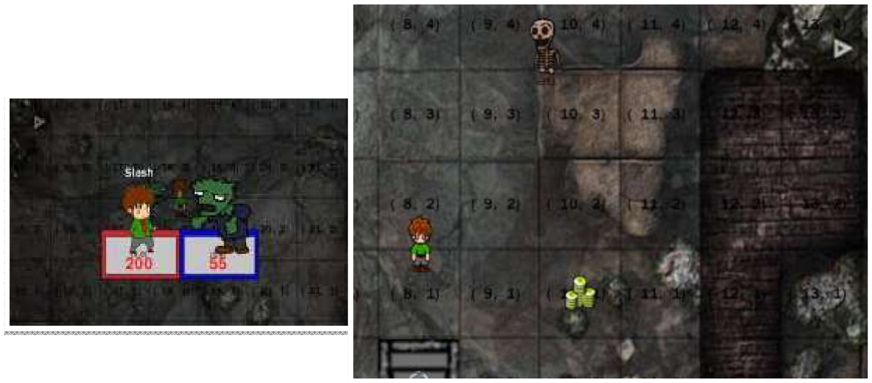


Figure 6: Secret Tunnels

To maintain fun and keep an element of surprise, various secret tunnels can be uncovered after successful completion of quests. Figure 6 illustrates that these tunnels are dark and populated with vicious monsters and gold pieces. By battling the monsters, the players can increase their skill levels, and the collected gold pieces can be exchanged for weaponries and armories from the shop once the file I/O quest has been completed.

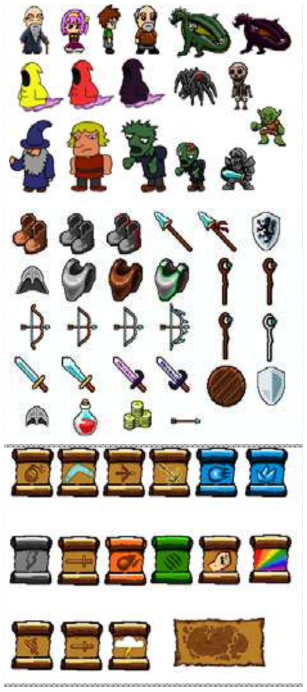
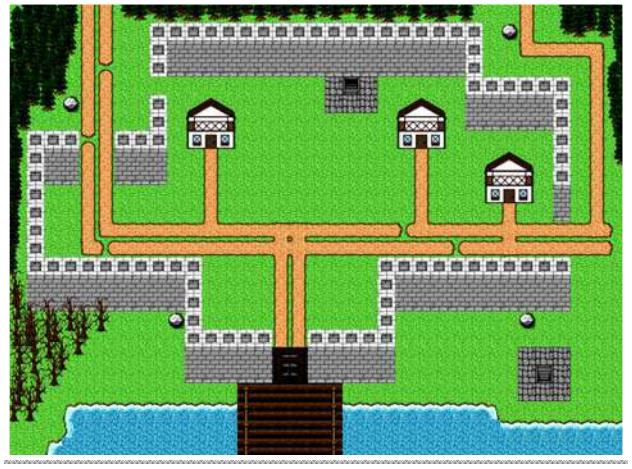



Figure 7: Art Assets and Unused Design

5. RESULTS AND CONCLUSION

Towards the end of the development cycle, the team presented the game system to four faculty members from the Pacific Northwest Region who teach introductory programming courses. The team presented the RPG story narration, demonstrated the quest game play with and without student solutions, and explained the philosophy of supporting faculty with no graphics and gaming background, emphasizing that a faculty member can choose to only use some of the quests in their existing courses. While the faculty found the project to be interesting, they expressed a neutral attitude towards using the game in their existing classes. One of the main concerns raised was the "*battle-theme*" where a non-violent mechanism for game advancement was preferred.

At the conclusion of the project, the team delivered a functional RPG system that consists of five quests which students can complete by coding introductory programming concepts. As an initial investigation, the team has successfully demonstrated the feasibility of building a complete game consisting of missing functionality that can be filled-in with introductory programming concepts as assignments.

However, the team did not fully succeed in delivering a completed programming assignment environment that can also serve as a fun and coherent RPG game. This is due in part to the complexity and difficulty of desiging and building fun and coherent RPG games. In our case, the added difficulties of integrating pre-specified introductory programming concepts severely restricted the potential dimensions for creativity. In addition, the multimedia asset requirements of a "*complete*" game can quickly overwhelm a quarterly academic time schedule. For example, Figure 7 shows some of the art assets and an unused town design developed for the Dream Coders game. Not shown are the numerous animated sprite sheets, the large number of special effect audio clips, and extensive font support. Even with these assets, our game environment appears relatively simple, repetitive and sparsely populated.

At the beginning of the production phase, the team assigned each member to a different quest and began the design and development of all the quests in parallel. The rationales for the decision were that the quests seemed straightforward and that the team members would be able to learn from each other as they all would be working on similar tasks in slightly different context. Unfortunately, while the quests themselves may have been straightforward, the asset requirements and integrations with the rest of the game were not always well-defined. For these reasons, the game has a lack of consistent behavior across different quests, e.g., the look and feel of Non-Player-Characters and the dialog interaction sequences were slightly different for each quest.

In hindsight, the team fell into the classic trap of "*feature creep*." Towards the end of the production phase, as the team began play testing the entire game as a whole, the inconsistency between different quests became obvious. Instead of addressing the inconsistency, the team directed its efforts at designing an additional "*coherent framework*" to host the quests. The hidden tunnels, inventory system, changeable armory system were all late-stage attempts at building this "*coherent framework.*"

It is important to focus on the fact that the "*game*" is a platform for students to develop and learn programming concepts. As such, the integration of the game development within a friendly IDE is also crucial. To build a fun, and complete game where students can amend or fill in functionality by hands-on coding, we believe a team should be separated into three collaborating groups with distinct goals: an intrinsic game group in charge of designing fun and coherent gameplay, an academic group in charge of integrating academic contents as assignments, and an IDE group in charge of providing a friendly and familiar development environment.

Finally, it is worth reiterating that teaching programming with video games is not a silver bullet for promoting greater engagement by students. As we observed in our final project presentation, faculty members must also be engaged, or at least interested, and 2D RPG games – especially those with battle-oriented activities – may not be acceptable to all such faculty.

ACKNOWLEDGEMENTS

We would like to acknowledge the other members of the Dream Coders Project: Stephen Jonany, Howard Lee, Vince Ly, Shahrzad Mahmoudi, and Antonio Martinez. Colleagues who responded to our requests for sample assignments include: Jim Howe from Lake Washington Technical College; Rob Nash, Carol Zander, and Clark Olson from University of Washington Bothell; Paul Bladek from Edmonds Community College, Michael Panitz from Cascadia Community College, Richard Weiss from Evergreen College; Scott Wallace from Washington State University at Vancouver; and Jim Hendricks from Pierce College. Colleagues who came to the final presentation and provided feedback are: Clark Olson from University of Washington Bothell; Robert Shields from Central Washington University; and Susan Reeder from Seattle University.

REFERENCES

[1] The game-themed introductory programming project, 2012. Computing and Software Systems, University of Washington, Bothell, *http://depts.washington.edu/cmmr/Research/XNA_Games*.

[2] Robin Angotti, Cinnamon Hillyard, Michael Panitz, Kelvin Sung, and Keri Marino. Game-themed instructional modules: a video case study. In *FDG '10: Proceedings of the Fifth International Conference on the Foundations of Digital Games*, pages 9–16, New York, NY, USA, 2010. ACM.

[3] Hazeline Asuncion, David Socha, Kelvin Sung, Scott Berfield, and Wanda Gregory. Serious game development as an iterative user-centered agile software project. In *Proceeding of the 1st international workshop on Games and software engineering*, GAS '11, pages 44–47, New York, NY, USA, 2011. ACM.

[4] Tiffany Barnes, Heather Richter, Eve Powell, Amanda Chaffin, and Alex 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.

[5] Jessica D. Bayliss. The effects of games in CS1-3. *Journal of Game Development*, 2(2):7–18, 2007.

[6] Kevin Bierre, Phil Ventura, Andrew Phelps, and Christopher Egert. Motivating oop by blowing things up: an exercise in cooperation and competition in an introductory java programming course. In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 354–358, New York, NY, USA, 2006. ACM Press.

[7] Wanda Dann, Steve Cooper, and Randy Pausch. *Learning to Program with Alice*. Prentice Hall, Upper Saddle River, NJ, 2006.

[8] Ray Giguette. Pre-games: games designed to introduce CS1 and CS2 programming assignments. In *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education*, pages 288–292, New York, NY, USA, 2003. ACM Press.

[9] Cecilia M. Gorriz and Claudia Medina. Engaging girls with computers through software games. *Commun. ACM*, 43(1):42–49, 2000.

[10] Patricia 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.

[11] Cinnamon Hillyard, Robin Angotti, Michael Panitz, Kelvin Sung, John Nordlinger, and David Goldstein. Game-themed programming assignments for faculty: a case study. In *SIGCSE '10: Proceedings of the 41st ACM technical symposium on Computer science education*, pages 270–274, New York, NY, USA, 2010. ACM.

[12] Timothy Huang. Strategy game programming projects. In *CCSC '01: Proceedings of the sixth annual CCSC northeastern conference on The journal of computing in small colleges*, pages 205–213, , USA, 2001. Consortium for Computing Sciences in Colleges.

[13] Michael Külling and Poul 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.

[14] Scott Leutenegger and Jeffrey Edgington. A games first approach to teaching introductory programming. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pages 115–118, New York, NY, USA, 2007. ACM Press.

[15] Mark Lewis, Scott Leutenegger, Michael Panitz, Kelvin Sung, and Scott A. Wallace. Introductory programming courses and computer games. In *SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education*, pages 204–205, New York, NY, USA, 2009. ACM.

[16] Mark C. Lewis and Berna 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.

[17] Ian Parberry, Max B. Kazemzadeh, and Timothy Roden. The art and science of game programming. In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 510–514, New York, NY, USA, 2006. ACM Press.

[18] Ian Parberry, Timothy Roden, and Max 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.

[19] John Minor Ross. Guiding students through programming puzzles: value and examples of java game assignments. *SIGCSE Bull.*, 34(4):94–98, 2002.

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

[21] Kelvin Sung, Michael Panitz, Cinnamon Hillyard, Robin Angotti, David Goldstein, and John Nordlinger. Game-themed programming assignment modules: A pathway for gradual integration of gaming context into existing introductory programming courses. *IEEE Transactions on Education*, 54(3):416 –427, August 2011.

[22] Kelvin Sung, Michael Panitz, Scott Wallace, Ruth Anderson, and John 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.

[23] David W. Valentine. Playing around in the CS curriculum: reversi as a teaching tool. *J. Comput. Small Coll.*, 20(5):214–222, 2005.

[24] Scott A. Wallace and Andrew Nierman. Using the java instructional game engine in the classroom. *J. Comput. Small Coll.*, 23(2):47–48, 2007.