

EUROGRAPHICS
2007,
CGF
Vol
26,
No
3

EUROGRAPHICS 2007, CGF Vol 26, No 3

, pdfkeywords=Computer Graphics Forum, EURO-
GRAPHICS]]sty

df

Extended Game Platform for Novice Programmers

Yolanda Rankin[†], Tom Lechner[‡] and Bruce Gooch[§]

Abstract

In an attempt to recruit and retain computer science majors, game design courses have become increasingly popular in academia. Game design encompasses multiple stages of product development, takes an average of two or more years and includes a team of individuals who possess strong programming abilities. Additionally, game development platforms consists of complex architectures that are difficult for novice programmers to comprehend let alone navigate. Due to this complexity, game design courses are typically taught as capstone courses intended for students who have intermediate or advanced programming skills. Consequently, introductory computer science courses do not include game development as typical programming assignments. We introduce a learning model that employs team-based pedagogy as the basis for students acquiring object-oriented programming skills, equipping programmers with the ability to design 3D games. Furthermore, we modify an existing game platform to include built-in scaffolds that assist students with the comprehension and application of object-oriented concepts. In response to the criticism that sustained hours of game-play contribute to physical inactivity and the growing obesity epidemic affecting America, novice programmers are given the creative task of designing a physically interactive game module that aids the player in meeting fitness goals and provides a source of entertainment.

Categories and Subject Descriptors (according to ACM CCS): I.3.8 [Computer Graphics Applications]: K.3.1 Computer Uses in Education, collaborative learning,

1. Introduction

Statistics show that number of students pursuing computer science as a major in American colleges and universities has dropped by as much as 60%[Sny06, Ves05]. These statistics indicate decreasing interest of college freshmen in computer science at a time when industry advertises employment opportunities in computer related fields. Several misconceptions deter college students from pursuing computer science as a major, one being that computer scientists are males who lack social skills and whose lives revolve around sitting in front of the computer all day long[GF05, GS02, MF02]. Industry has voiced its concern about the decreasing supply of qualified employees in computer related fields, deciding to partner with academia to find solutions to address the issue of recruitment and retention of computer science majors[Car05]. One solution

involves invigorating traditional core computer science curriculum with pedagogical strategies that leverage the use of digital media as a means to generate interest in courses perceived to be designed exclusively for techies, geeks and programming gurus[Car05, GS02]. In an attempt to address recruitment and retention issues, computer science departments across the country have begun to adopt the digital medium of computer games as a context for software development[AV02, BS00, Jon00, PKR06, PRK05, RGG07].

Game design involves multiple stages of development; the process is largely a team effort and takes an average of two years[FSH04]. Individuals comprising the development team understand object oriented design and possess strong programming skills. Unsurprisingly, game development platforms consists of complex software and architecture that is difficult for novice programmers to comprehend or use effectively. Though game design has become an increasingly popular course offering in academia, students new to programming lack the necessary skill set to develop games. Due to this complexity, game design courses are typically taught as capstone courses in-

[†] Northwestern University, Evanston, IL USA

[‡] Northwestern University, Evanston, IL USA

[§] University of Victoria, Vancouver, British Columbia

tended for advanced programmers or computer science majors[BS00, Jon00, PKR06, PRK05, RGG07]. For these reasons, introductory computer science courses either do not include game development as typical programming assignments or offer simple game programming assignments (e.g. hangman) that lack sophisticated graphics and animation associated with 3D games[LWS06]. We suggest that designing interactive media such as games is one creative way to challenge students' negative perceptions of computer science and possibly recruit computer science majors.

In contrast, introductory computer science courses typically do not include game design. These courses expose students to various programming paradigms and serve as building blocks to assist students with the transition from novice to intermediate programmers. However, teachers often struggle with creating meaningful programming assignments that cohesively represent the underlying principles of a specific programming paradigm and assist students in developing their programming abilities[LWS06, Try99, Try05].

We propose an alternative to assist students with the transition from novice to intermediate programmers; we anchor object-oriented programming to game design. Students are given the task of designing a video game that promotes physically interactive game play. To accomplish this goal, we transform Microsoft's Flight Simulator X (FSX) platform into a manageable development environment that supports novice programmers. The infrastructure includes:

- Modification of Microsoft's Flight Simulator X as a manageable game platform
- Library of starter code of game objects
- Laboratory programming assignments that emphasize object-oriented design concepts
- Built-in software evaluation tool to assess students' object-oriented programming skills.

Though we have extended the Flight Simulator X (FSX) platform, our design would not be complete without identifying team-based pedagogical strategies that reinforce the learning objectives of a CS102 course. Thus, team-based learning determines the manner in which students acquire object oriented programming skills in the context of game design.

2. Team-based Pedagogy

Traditional computer science curriculum emphasizes individual achievement, discouraging students from working together for fear of unethical practices such as cheating. In contrast, the game industry values teamwork as a necessary practice for game development[Bla94, FSH04]. Previous experience from teaching game design courses has demonstrated that unrealistic expectations, low productivity, poor quality work and missed deadlines become the norm when students choose to work solo on game development projects.[RGG07]. To facilitate students' abil-

ity to work with others and strengthen their interpersonal skills, we incorporate teamwork as an essential component of introductory computer science courses and present a team-based learning model applicable to programming assignments[Try05]. The team-based learning model promotes collaborative learning among students, dividing them into groups for the purpose of designing a game module that is virtually impossible for one student to do given specific time constraints and students' lack of programming experience. We define the process of team-based programming according to the following steps:

Students review the rubric for each programming assignment. We equate rubric to the course requirements, providing a framework for assignments, learning objectives and methods for student assessment. A well-defined rubric serves as a guide for students' learning and communicates the teacher's expectations for each programming assignment [MKF04]. We post the rubric for each laboratory programming module on the course wiki so that the information is available in a public place. Public display of the criteria for team programming assignments initiates discussions among team members and invites students to share their ideas and knowledge with others; thus communication becomes a crucial element that facilitates individual and team learning while students learn to work together to complete programming assignments [Bru98, FB06, EPG95]. For example, the first programming assignment requires each team to render the game background to be displayed on the screen. Students complete mock-ups of the visual layout for one level of the game module. Each team explores the formal elements of game design (e.g. genre, objectives, rules, and possible outcomes) prior to developing software to draw the scene. We expect students to use graphics art tools (e.g. Adobe PhotoShop) to create original artistic content for the background. Learning objectives include demonstrating familiarity of graphics arts packages, applying knowledge of bitmaps to create textures for a game background, and writing code that generates images for the background and text that represents player status during game-play.

Team members negotiate workload. Trytten[Try05] recommends dividing students into groups based on a psychological assessment tool that classifies students as either introverts and extroverts[Jun]. Students are divided into teams based upon the results of the psychological assessment; teams meet during class time to avoid social loafing and conflicting schedules which impede team members ability to work effectively[Try05, Try99]. Teams participate in a top-down design approach to derive a solution for the programming assignment, breaking down the design of the game module into manageable pieces. Team members negotiate the portions of the assignment that each student will complete; member assignments are posted on the wiki for review to ensure that each team member contributes to team's goals. The act of negotiating engages team members in supportive communicative practices such as sharing information that

builds group knowledge, acknowledging peers' ideas, mediating conflict when group members disagree, and asking for assistance[Sol01]. Referring to the previous example of generating a virtual world, team members discuss what type of game they would like to design. Once the team makes a decision, each member assumes responsibility for a specific task. One team member may take on the role as graphics artist, generating several bitmap images while another student may assume the role of user interface designer, determining the best layout for presenting player status on the screen. Students work together to create the look and feel of the game. These social interactions form the basis for building a high-performance team that works together to accomplish the common goal of designing a game module.

Individual students write code and complete assessment. Although team-based learning emphasizes collaboration among group members to acquire object-oriented design skills, it is not a substitute for individual students mastering these skills. Using Visual Studio Express 2005, students write and compile code for their portion of the team programming assignment. Each student then uploads individual code segments to the course wiki and completes an online quiz that measures the student's comprehension of object oriented concepts. The quiz lists open-ended questions and the answers are available only to the teacher. The individual code segments and on-line assessment supplement individual learning and serve as preparatory work for the team programming assignment. The library of starter code provides a model of object-oriented concepts, scaffolding each student's ability to write code. The first programming assignment would require students to modify the *Background Class* in the library of starter code in order to write code that renders a scene of the virtual world.

Team members share individual code segments and derive a solution. Team-based learning maximizes opportunities to socialize with students of various levels, resulting in more creativity, greater productivity, and increased individual knowledge [KG04, MKF04]. Because each student is accountable for developing a portion of the code that contributes to the team's final product, students share code with team members and conduct peer reviews. By sharing individual solutions, different approaches to creating a game module become visible to all team members. Michaelson et al.[MKF04] posit that groups function best when each member works on the same problem, is given a specific task and reports status to team members. This aids the design process and helps the team experiment with different implementations (e.g. visual layout of player status). As a consequence of group social interactions, team members function as the first tier of support for collaborative learning by provoking members to propose ideas about implementation, ask and offer assistance when team members experience difficulty writing software and build knowledge of object-oriented concepts[SB96]. Teams upload their software and present the final solution to the entire class for review. Stu-

dents critique each team's code, developing analytical reasoning skills applicable to both game design and object-oriented programming.

3. Existing Game Platforms

Existing game platforms have been used in the academic setting. For example, the Unreal Game Engine and Steam Engine platform embody complex development kits that are ideal for intermediate programmers but unintuitive for novices [Lai01, LvL99, RGG07]. These game development kits require several hours spent tracing thousands of lines of code just to comprehend how the software modules fit together to create a game. Asking students who are new to programming to review code that includes exception handlers, multi-thread processing and windows programming overwhelms novices, possibly demotivating them and decreasing their interest in computer science.

Manageable development platforms for novice programmers do exist. Panda 3D, developed as a joint effort between Disney Studios and Carnegie Mellon's Entertainment Technology Center features a 3D engine complete with a C++ library of subroutines for 3D rendering, 3D models and artistic content for game development. The Panda 3D game platform supports a short learning curve and rapid prototyping. Alice represents a software application developed at Carnegie Mellon University that gives students with little or no programming experience the ability to design interactive media featuring 3D graphics [CAB*00]. While students have used Alice to write programs to construct virtual worlds, Alice lacks a game engine that supplies sophisticated graphics or artificial intelligence. XNA Game Studio Express developed by Microsoft includes a game engine; students with limited experience programming in C can use XNA Game Studio Express to design 2D and 3D games for the Xbox or Windows XP SP2[CK07].

Unlike the XNA game kit, Microsoft's FSX is a flight simulator designed for users who want to learn how to pilot various types of aircraft or direct air traffic. Its complex architecture generates real life scenery that places the player in the cockpit. The FSX SimConnect API allows developers to create game add-ons using existing code; We strive to hide the complexity of the FSX architecture while taking advantage of its graphical capabilities to create a simplified game development platform for novice programmers.

4. Design and Implementation

The Kaiser Family Report 2005[RFR05] indicates that 84% of children ages 8 - 14 years old live in homes with at least one video game and 30% of this same population own three or more video games. Additionally, children ages 8 to 18, spend an average of 44.5 hours per week in front of a computer or television screen. This accounts for the maximum



Figure 1: FSX cockpit view of panel instrumentation.

amount of time spent on any other activity except sleeping [RFR05]. The lack of physical activity that accompanies these stationary hours of game-play contributes to a growing obesity epidemic affecting America[VSC04]. In response to this growing epidemic, we take a different approach to solving the problem of physical inactivity due to sustained hours of sedentary entertainment. Rather than limiting the amount of time spent playing video games, we offer a solution that maximizes the appeal of video games as a fitness alternative in an attempt to prevent obesity and increase cardiovascular health. Our research strives to transform exercise into a mode of media entertainment that attracts an audience comparable in size to popular video games with similar time usage. Thus, we transform Microsoft's FSX SimConnect API into a manageable software development environment that scaffolds novice programmers as they design a fitness intervention game. The modified game platform embodies four components:

- G-Force recumbent bike,
- C# Library for creating game objects,
- C# Interface for testing students' code,
- and *Student Tracker* for evaluating students' code.

System requirements include Microsoft Visual Studio 2005 Express to maintain the C# library and team lab projects, Microsoft Flight Simulator X installed on a machine that has a minimum of 10 GB of memory, a graphics card and access to the internet.

4.1. G-force Recumbent Bike

The G-force recumbent bike represents the game interface and has been engineered to provide input (e.g. resistance and heart rate)during game play. See figure 2. The recumbent bike features a 19-inch screen that displays the video game. Students are tasked with designing and implementing software that integrates the activity of riding the bike with monitoring the user's physiological factors such as heart rate to assist players with attaining their fitness goals.



Figure 2: FSX Prototype of Flying bike interface.

4.2. C Library for Flight Components

The C# library serves as the foundation for assisting novice programmers in developing their object-oriented programming skills as individuals and as a team. We define a super-set of classes that represent game objects that students will need to implement to successfully construct a game module. The C# library provides starter code that defines the properties and behaviors of the game objects. Additionally, the C# library represents good and bad examples of object-oriented design concepts and serves as a starting point for students to analyze, evaluate and generate C# code. For example, the game module should possess the capability to accept user input such as the player's name, age, and fitness goal and display this information on the screen. We stub a *GamePlayer* class that includes the physiological factors of each player and methods that keep track of the player's training heart rate zone (THR). See psuedo code below.

Public GamePlayer Class

```

name
age
maximum heart rate
trainingzone
lower thzone
upper thzone
fitnessgoal
    
```

GamePlayer Constructor

```

ID = "Stranger"
age = 0
maxheartrate = 220
trainingzone = false
lower thzone = 0
upper thzone = 0
fitnessgoal = 1
    
```

Method defined for calculating training heart rate zone
 GamePlayer's maxheartrate = 220 - player's age
 GamePlayer's lower thzone = maxheartrate * .65

```
GamePlayer's upper thzone = maxheartrate * .85
```

Method for displaying info on screen

(Student needs to add code that displays info on screen.)

end of GamePlayerClass

4.3. C# Interface for Team Programming Assignments

The Microsoft FSX the API is highly advanced and in its original state would be overwhelming for a set of novice computer programmers. For this reason, we have simplified the interface, hiding key components of game play (e.g. take off or landing of an aircraft) from the user. The interface conceals the technical inner workings of the game engine, allowing students to focus their attention on the learning objectives associated with each programming assignment.

The C# interface also grants students access to a set of generic data collection interfaces with the intention that students understand the distinction between the data structures (e.g. arrays, linked lists, stacks, vectors, etc.) and their respective implementations. Given that the number of permutations for each data structure is quite large, each student can choose a particular data structure for implementation and share code segments with other team members. For example, students may choose to use a tree data structure to render a scene of the game world. In this example, the C# interface offers a plug and play test environment of for students' code and foster's students understanding of 3D graphics (e.g. matrix transformations) to move objects on the screen. Therefore, the C# interfaces support students' implementation of algorithms for various data structures.

4.4. Student Tracker Evaluation Tool

One of the criticisms of using Visual Studio is the cryptic error code messages students encounter during the course of writing code. To procure further insight about students' programming abilities, we develop and integrate a Microsoft Visual Studio 2005 plug-in known as the *Student Tracker*. The *Student Tracker* serves as a scaffolding tool that collects debugging information, accumulating a list of code errors each time the student compiles code. Students must download and install the Microsoft Visual Studio 2005 SDK prior to the initial use of the *Student Tracker* add-in. After the Visual Studio 2005 SDK has been installed, students open Visual Studio 2005 and build the *Student Tracker* project solution. Students access the *Student Tracker* under the Tools options on the menu bar whenever they begin or resume software development on a program. *Student Tracker* exports debugging information such as compiler errors, warnings and the session timestamp as an XML file that can be viewed by the teacher. This data helps the instructor to identify the strengths and weakness of each student as it relates to the learning objectives for each programming assignment. Thus,

the instructor makes an informed decision about which pedagogical strategies to use to assist students with adopting object-oriented design concepts.

5. Game Design Laboratory Modules

Students accomplish the goal of designing a video game that promotes a healthy lifestyle by completing four laboratory modules as part of the CS102 course requirements. We specifically design the four modules to correspond to two stages of game design: concept phase (project plan, teamwork, idea) and pre-production phase (playable prototype and playtesting) [FSH04]. Modules 1 and 2 are completed as individual assignments; modules 3 and 4 are team assignments. Module 1 requires students to playtest video games to generate ideas for the team game module and develop an appreciation for the formal and dramatic elements of game-play. Module 2 gives students the opportunity to become familiar with the game development platform and Microsoft Visual Studio. Students complete the first design iteration of their physically interactive game module in Module 3. Module 4 enables students receive critical feedback regarding ways to improve the game-play experience for users. Each module specifies a learning objective and includes a game development exercise which requires students to demonstrate a fundamental concept of object-oriented programming.

To assess students' knowledge of object oriented design principles, students complete quizzes on the course wiki to assess individual and group learning and submit code for peer review. Students are required to spend 3 hours per week in the computer lab in fulfillment of course requirements, eliminating the issue of social loafing and scheduling outside class meetings with team members [Try05, Try99]. Teachers can access the course wiki (<http://cs102.pbwiki.com>) for CS102 and update the website using a group password. Ultimately, all four laboratory modules will comprise each team's game module that requires physically active game-play in an effort to promote a healthy alternative to video games. FSX's SimConnect API will provide an intuitive test harness for students to test their newly developed game module.

5.1. Laboratory Module 1: Playtesting Video Games

- *Learning Objective:* Students will establish criteria that evaluate the game-play experience.
- *Game Development Exercise:* Students will playtest FSX and one games their choice and compare the formal and dramatic of elements each.

The primary objective of the first module is for students to become proficient playtesters. Fullerton et al. [FSH04] describe playtesting as a crucial component of game design that ensures delivery of a quality product.

Playtesting occurs throughout the various stages of game design and drives game revisions that enhance the dramatic and formal elements of game design. Students evaluate FSX and one game of their choice according to the following playtest criteria:

1. What was your first impression?
2. How did your impression change as you played the game?
3. Describe the objective of the game.
4. Were the procedures and rules easy to understand?
5. Could you find the information you needed on the interface?
6. Was there anything that you found frustrating?
7. Were there particular aspects that you found satisfying?
8. What was the most exciting thing about the game?
9. What was missing from the game?
10. In what way did you interact with other players?

See the website (<http://cs102.pbwiki.com/LabModule1>) for additional details for laboratory module 1.

5.2. Laboratory Module 2: Microsoft Visual Studio Development Environment

- *Learning Objective:* Using Visual Studio, students will learn how to profile code.
- *Game Development Exercise:* Each student must step through code in the C# library, identifying the algorithm, run-time performance of algorithm, and logical flow.

The second laboratory assignment introduces students to Microsoft Visual Studio Software Development Environment (SDE). The primary objective of this module is for students to become proficient users of Visual Studio. As students understand the menu options available in Visual Studio, they will be able to step through code and use built-in debugging tools to assist them with programming assignments. Each student is responsible for submitting a portion of code and a code analysis report. Visit website <http://cs102.pbwiki.com/LabModule2> for specific instructions for module 2.

5.3. Laboratory Module 3: Game Module

- *Learning Objective:* Students will attain knowledge of constructors, destructors, copy-constructors, public and private members and methods for classes. Students will be able to differentiate between public vs. private members and demonstrate knowledge of data abstraction.
- *Team Programming Exercise:* The primary goal of this exercise is for students to work together to design a game module.

Each team determines the design of physically interactive video game. Students design original artistic game content including bitmap images for the game background, moving objects on the screen, instructions for game-play, and game

mechanics (e.g. player's score). To complete this task, students work in teams as each individual writes software for specific game artifacts and shares code segments that culminate in the team's final game module. To assist with this effort, students modify and extend a previously defined library of class objects (i.e. data members and member functions) to create instantiations of their own. Thus, students develop software modules that demonstrate the concept of data abstraction and encapsulation.

5.4. Laboratory Module 4: Playtesting Team Game Modules

- *Learning Objective:* The learning goal is for students to evaluate each game module's design and identify strengths and areas for improvement.
- *Game Development Exercise:* The learning method encourages students to evaluate C# code that demonstrates the culmination of the four laboratory exercises.

FSX SimConnect functions as a test harness for each team's human-powered aircraft implemented in module 3. Students are responsible for modifying existing interface files which define the data structures representing data objects in the scene, implementing files defining the methods for manipulating the data objects in the scene, and the client program which provides a higher level of abstraction and uses methods defined in the implementation file. Students evaluate the game using previously defined playtest criteria in module 1 and consider aesthetic appeal, integration of physical activity into game-play and object-oriented design techniques. Feedback serves as input into the revision process of game development. This module allows students to playtest two team's game modules. Students evaluate the game-play experience and provide feedback to the authors. Based upon the feedback, the game designers modify the game to enhance the game-play experience. Each team demonstrates their game module during lab time.

6. Proposed Evaluation & Future Work

To determine the effectiveness of our game development platform, we will evaluate the laboratory modules as part of an introductory programming course and compare students' performance in our course to students who enroll in a traditional object-oriented programming class. Both groups of students will complete a mock registration that compares students' interest in enrolling in a traditional introductory computer science course to an introductory course that features game design. To further assess students' motivation and interest in game design, students enrolled in an introductory programming course will complete an online survey during the first week of class. The survey measures students' interest in the discipline of computer science/computer related majors, previous programming background, and interest in game design. Students will complete the same online

survey once they have complete course requirements. The pre-assessment will enable us to correlate students' interest in game design to pursuing a major in a computer related field. The post assessment will determine the success of the game design course, evaluating our pedagogical approach to anchoring game design to object-oriented programming in comparison to students who complete the traditional object-oriented class.

Understanding that this is the first design iteration of the simplified FSX game platform, we will ask students' for feedback that will inform future revisions of both the game development platform and computer science curriculum. Based upon each team's game module, we will identify additional game artifacts and enhance and refine the C# library. We will continue to improve the *Student Tracker* to collect additional metrics such as time spent completing programming assignments to identify potential student misconceptions of object oriented concepts that need clarification. This will give the teacher a better picture of what the student has learned and more importantly, can be shared with the student to maximize the time spent on programming assignments.

7. Conclusions

We have modified Microsoft's FSX game platform to support novice programmers' ability to design their own game modules. We have stubbed the C# library for aircraft design components and player's physiological factors. The C# interfaces sustains students' ability to test and share code with peers as each team builds a fitness training video game. Concurrently, we have implemented a pedagogical tool known as the *Student Tracker* that collects debugging data such as type and frequency of compiler errors to assess individual and team acquisition of object-oriented design principles. Moreover, we outline four laboratory modules that give students the opportunity to practice object-oriented programming skills as students incrementally design a game module. Thus, we extend advanced game technology that enables introductory computer science students' ability to implement a video game-play as an incentive for physical fitness. In summary, we introduce a team-based learning model that assists students with acquisition of intermediate programming skills in the context of game design. We posit that this model can be implemented in other introductory computer science courses, hopefully increasing the number of students who pursue majors and careers in computer science.

8. Acknowledgements

We would like to thank Microsoft Research for their generous support, especially John Nordlinger whose advice has proven to be most beneficial in our attempts to maximize the appeal of video games as an effective learning tool. We also appreciate the National Science Foundation who generosity makes this reserach possible.

References

- [AV02] ALPHONCE C., VENTURA P.: Object orientation in cs1-cs2 by design. In *In Proceedings of 7th Annual Conference on Innovation and Technology in Computer Science Education* (2002), ACM Press, pp. 70–74.
- [Bla94] BLACK K.: "an industry view of engineering education", 1994.
- [Bru98] BRUCKMAN A.: Community support for constructionist learning. In *The Journal of Computer Supported Collaborative Work* (1998), vol. 7, pp. 47–86.
- [BS00] BAYLISS J., STROUT S.: Games as a flavor of cs1. pp. 500–504.
- [CAB*00] CONWAY M., AUDIA S., BURNETTE T., COSGROVE D., CHRISTIANSEN K., DELINE R., DURBIN J., GOSSWEILER R., KOGI S., LONG C., MALLORY B., MIALE S., MONKAITIS K., PATTEN J., PIERCE J., SCHOCHET J., STAAK D., STEARNS B., STOAKLEY R., STURGILL C., VIEGA J., WHITE J., WILLIAMS G., PAUSCH R.: Alice: Lessons learned from building a 3d system for novices.
- [Car05] CARLESS S.: Postcard from sgs 2005: Computer gaming to enhance computer science curriculum, 2005. <http://www.gamasutra.com/features/20051101/carless01.shtmlhttp://www.g>
- [CK07] COX C., KLUCHER M.: Got game? unleash your imagination with xna game studio express, 2007. <http://msdn.microsoft.com/msdnmag/issues/07/05/xna/default.aspxS1http://>
- [EPG95] EDELSON D., PEA R., GOMEZ L.: Constructivism in the collaboratory. Englewood Cliffs, NJ. Educational Technology Publications.
- [FB06] FORTE A., BRUCKMAN A.: From wikipedia to the classroom: Exploring online publication and learning. In *Proceedings of the 7th International Conference on Learning Sciences, Bloomington, IN* (2006), p. 182–188.
- [FSH04] FULLERTON T., SWAIN C., HOFFMAN S.: *Game Design Workshop: Designing, prototyping, and playtesting games*. CMP Books, 2004.
- [GF05] GUZDIAL M., FORTE A.: Design process for a non-majors computing course.
- [GS02] GUZDIAL M., SOLOWAY E.: Teaching the nintendo generation to program: Preparing a new strategy for teaching introductory programming. In *Communications of the ACM* (2002), vol. 45.
- [Jon00] JONES R.: Design and implementation of computer games: a capstone course for undergraduate computer science education. In *In Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education* (2000), ACM Press, p. 260–264.
- [Jun] JUNG C.: *The Collected Works of Carl Jung*, vol. 6. Princeton/Bollinger.
- [KG04] KWON S. M., GOMEZ L.: Strengthening learning communities by promoting social skill development.

- In *Proceedings of the 6th International Conference on Learning Sciences, Santa Monica, CA* (2004), pp. 286 – 293.
- [Lai01] LAIRD J.: Using a computer game to develop advanced ai. *Computer* 34, 7 (July 2001), 70 – 75.
- [LvL99] LAIRD J., VAN LENT M.: Developing an artificial intelligence engine.
- [LWS06] LAYMAN L., WILLIAMS L., SLATEN K.: Note to self: Make assignments meaningful. In *Proceedings of the Thirty-Eighth SIGCSE Technical Symposium on Computer Science Education* (2006), ACM Press, pp. 459 – 463.
- [MF02] MARGOLIS J., FISHER A.: *Unlocking the Clubhouse: Women in Computing*. MIT Press, Cambridge, MA, 2002.
- [MKF04] MICHAELSON L., KNIGHT A., FINK L.: *Team-Based Learning: A Transformative Use of Small Groups in College Teaching*. Stylus Publishing, Sterling, VA, 2004.
- [PKR06] PARBERRY I., KAZEMZADEH M., RODEN T.: The art and science of game programming. In *Proceedings of the 2006 ACM Technical Symposium on Computer Science Education, Houston, TX* (2006), pp. 510 – 514.
- [PRK05] PARBERRY I., RODEN T., KAZEMZADEH M.: Experience with an industry-driven capstone course on game programming. In *Proceedings of the 2005 ACM Technical Symposium on Computer Science Education, St. Louis, MO* (2005), pp. 91 – 95.
- [RFR05] ROBERTS D., FOEHR U., RIDEOUT V.: *Generation M: Media in the Lives of 8-18 Year-olds Report Kaiser Family Foundation*. 2005. <http://www.kff.org/entmedia/7251.cfm>
<http://www.kff.org/entmedia/7251.cfm>.
- [RGG07] RANKIN Y., GOOCH B., GOOCH A.: Interweaving game design into core cs curriculum. In *Proceedings of the 2007 Microsoft Academic Days Game Development Conference, Nassau, Bahamas, February 21 Ú 25, 2007* (2007).
- [SB96] SCARDAMALIA M., BEREITER C.: Student communities for the advancement of knowledge. 36–37.
- [Sny06] SNYDER N.: Universities see a sharp drop in computer science majors.
- [Sol01] SOLLER A.: Supporting social interaction in an intelligent collaborative learning system. 40 – 62.
- [Try99] TRYTTEN D. A.: Progressing from small group work to cooperative learning: A case study from computer science. In *Journal of Engineering Education* (1999), vol. 90, pp. 85 – 92.
- [Try05] TRYTTEN D.: A design for team peer code review. In *Proceedings of SIGCSE '05, February 23 - 27, 2005, St. Louis, MO* (2005), pp. 455 – 459.
- [Ves05] VESGO J.: Interest in cs as a major drops among incoming freshman.
- [VSC04] VANDEWATER E., SHIM M., CAPLOVITZ A.: Linking obesity and activity level with children’s television and video game use. vol. 27, Elsevier Science, Oxford, pp. 71–85.