Microsoft Academic Days on Game Development in Computer Science Education

Aboard the Disney Wonder Cruise Ship

February 22-25, 2007

Message from the Program Committee Chair

Welcome to the Microsoft Academic Days on Game Development in Computer Science Education, aboard the Disney Wonder Cruise Ship. This is the second year for this meeting, and the first that has an academic track with refereed papers.

This year 71 papers were submitted. Each paper received three reviews. Based on the reviewers' feedback, the Program Committee selected the best papers, 19 of which are being presented at the conference and included in these Proceedings. A further 9 papers are highlighted as poster presentations.

I would like to take this opportunity to thank the members of the Program Committee for their sterling work in reviewing and selecting these papers under a tight deadline. The Program Committee consists of:

The Program Committee would also like to thank John Nordlinger of Microsoft Reasearch and Kent Foster of Microsoft Corporation for their help in organizing this meeting, and Microsoft Corporation for providing travel funds for authors and participants.

Ian Parberry
Program Committee Chair

CONTENTS

Combining games with theatre to create an interdisciplinary learning experience for Computer Science students

Joe Geigel
Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
jmg@cs.rit.edu

Marla Schweppe
School of Design
College of Imaging Arts and Science
Rochester Institute of Technology
mkspph@rit.edu

ABSTRACT
In this paper we describe our experiences in combining theatre with games programming as a means of providing a collaborative experience for Computer Science students. With the goal of creating a theatrical production in a virtual space, we've designed a set of companion courses, geared toward those looking to work in the electronic entertainment industries, which emphasizes teamwork, cooperation, and successful collaboration between artists and technologists. This paper focuses on the Computer Science course, where students work together towards adapting a gaming engine for theatrical purposes.

Categories and Subject Descriptors
K.3.2 [**Computing Mileux**]: Computers and Education-Computer and Information Science Education [Computer Science Education]

General Terms
Design, Experimentation

Keywords
Interdisciplinary coursework, theatre, games education.

1. INTRODUCTION
Imagine a computer science course where students develop a theatrical performance for a live audience. Like most theatre, excitement is in the air. The 'crew' of programmers is anxious and excited, wondering how the audience will respond to their performance. The difference is that in this performance, the action takes place in a gaming world; actors are all avatars carefully created and designed by artists; the performance is realized by the use of motion control devices, data gloves and a virtual stage manager. The audience sits at computers, rather than a physical theatre, where they watch and respond to the performance.

Not the usual programming classroom, This classroom is alive. Students are seeking solutions to problems, asking for components to be completed so they can be tested or integrated into the whole. Everyone plays a key role. Everyone has to come through to make the event a success; these students are fully engaged, depending on each other, and excited about the end result. This is what Virtual Theatre is all about. A unique educational experience for Computer Science students that combines theatre with games programming with the goal of teaching skills beyond the traditional CS course.

For the Computer Science student looking to enter the electronic entertainment industry, the skills required for success expands well beyond programming and algorithms. Although technical expertise is important, the interdisciplinary nature of these industries additionally demands practice in a number of non-technical skills. Qualities such as teamwork, collaboration, independent thinking, experience with a large project, flexibility, and problem solving have been reported as being critical in the success of those looking to work in technical positions in both the computer animation[1] and the game development[2,3] fields. In addition, successful projects in these fields rely on effective communication and uncompromised cooperation between artists and technologists.

Courses that foster such collaboration and address these non-technical issues have been reported in Animation [4], Image Synthesis [5], Virtual Reality [6] and game programming [2,7]. Common themes shared by these courses include: a Capstone experience; Collaboration of artists and programmers working on a single project; and a team based approach.

In our work, we embrace theses themes listed and, adding a theatrical component, we create a learning experience with the goal of further sharpening students' collaborative and teamwork skills. The courses, which are an integral component of a larger project (Virtual Theatre), create an educational experience that serve as a springboard for direct learning through interdisciplinary collaboration and teamwork. An important objective of the project is to provide an opportunity for students interested in working in the electronic entertainment industries to experience, first hand, the type of collaboration between artists and technologists that is crucial to success in that industry.

The education component of the project is implemented as a set of companion courses, some offered in the School of Design and others offered by the Department of Computer Science. The focus of this paper is the courses taught in Computer Science.

The remainder of the paper is structured as follows: In section 2, we discuss our motivations for using theatre as an application domain for the courses. This is followed by a brief overview of the Virtual Theatre project and the details of the Virtual Theatre CS course, in Sections 3 and 4 respectively. Results and observations from the course offerings are discussed in Section 5 and we finish the paper with Conclusions and Future work in Section 6.

2. THEATRE AS AN EDUCATIONAL MOTIVATOR

Theatre, by its very nature, is a collaborative art. Theatre can be defined as a place of action; field of operation [8]. The telling of a theatrical story takes place in a common field of operation and involves a number of actors, directors, designers, and technicians, who create the storytelling space and the action that is viewed within it. This collaborative environment provides a natural foundation for a learning experience that emphasizes teamwork and cooperation between artists and technologists.

In addition to the collaborative aspects of theatre, the discipline of theatre teaches a number of other highly valuable skills that can contribute to student success, regardless of their area of study. [9] We list some of these skills in Table 1 below. Not surprisingly, this list mirrors the skills previously mentioned as critical to those looking to working in the electronic entertainment industries.

Table 1 - Skills learned from working in the theatre

- Oral Communication Skills
- Creative Problem Solving Abilities
- Motivation and Commitment
- The Ability to Work Independently
- Time-budgeting Skills
- Initiative
- Promptness and Respect for Deadlines
- Respect for Colleagues
- Adaptability and Flexibility
- The Ability to Work Under Pressure

Others in the field of Computer Science have certainly recognized the value of theatre as an educational stimulus. Brenda Laurel uses theatre as a metaphor for Human Computer Interaction [10]. Others have taken a more direct approach, combining computer graphics education with traditional theatre [11,12]) which serve as a motivation for our approach.

3. THE VIRTUAL THEATRE PROJECT

The Virtual Theatre project is an interdisciplinary project between the department of Computer Science and the School of Design at the Rochester Institute of Technology. The goal of the project is to explore distributed theatre and enable the presentation of a theatrical performance in virtual space, where actors, crew, and audience to share and participate, possibly from different physical locations, in a single theatrical performance over a distributed network.[13].

The Virtual Theatre project serves as both a research project and educational experiment. There are three major facets to project:

1. Artistic – Exploration of means to create engaging distributed performance and to create the aesthetic elements of such a production.
2. Technical – Defining, specifying, implementing, and documenting, the technical components to enable such a performance
3. Educational -- Allow for first hand student involvement the research aspects of the project, both technical and artistic and to create a collaboratory experience for students in both camps.

Computer Science students are responsible for the technical aspects of the project. The goal of the technical team is to build a distributed VR system to support this theatrical interaction. Rather than starting from scratch, the system is built on top of a gaming engine which provides layered access to low level capabilities (graphics, sound, networking) required by such a system. Similar to the spirit of Machinima [14], (filmmaking using gaming technology), our framework adapts the functionality provided by a gaming engine and wraps it in the language and processes of the theatre, as illustrated in Figure 1. Control of, and interaction with, the system during a production is done at this more intuitive layer.

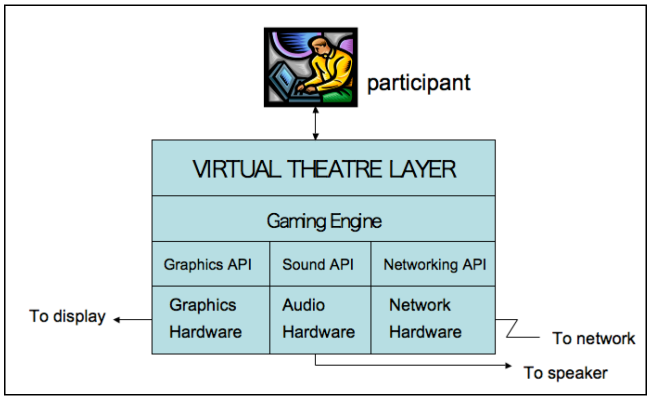


Figure 1 - Architecture of Virtual Theatre system

The framework assumes a shared object oriented database model where items in the virtual space are represented by software objects, which are distributed amongst all those accessing the shared space. 3D visual representations of these objects are created off-line by the design team using a modeling package and imported into the system prior to the production. The design team is also responsible for creating stage elements, defining lighting, and generating scripted animations for the avatars on the virtual stage.

During an actual performance, individual workstations, each resident with his or her own instance of the framework and assets, are networked together and configured to share a single virtual space as shown in Figure 2.

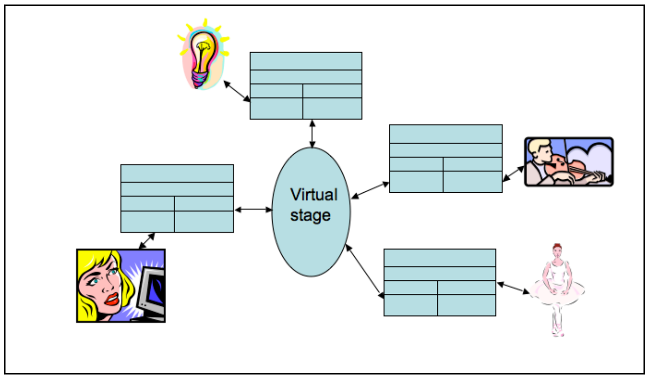


Figure 2 - Network configuration during a performance

4. VIRTUAL THEATRE COURSES

Logistically, the ultimate outcome of the courses is to create the technology and assets to realize a particular theatrical performance in a virtual space. Art students are tasked to design and create the aesthetic elements of the performance (models, textures, animation sequences) whereas the CS students are responsible for the design and implementation of the technical infrastructure to allow real time interaction and manipulation of these models. Unlike the courses presented in [4,5,6], we did not strive for a common classroom experience for both artists and technologists. Instead, students were instructed in their own discipline; however, their work was guided and shaped by constraints and requirements defined by the other half. As such, the project was implemented via a set of companion courses, some specifically for design students, and another for CS students, rather than a single course consisting of both sets of students. Efforts were made to have the companion courses offered at the same time, when possible.

The production to be realized in a given year's offering of Virtual Theatre is decided upon prior to the start of the school year. This production is chosen based upon technology and devices available. All course activities including the definition of the elements that student would be creating is driven by production needs both on the art and the tech side.

The remainder of this section will provide details of the course taken by Computer Science students.

4.1 Pre-requisites

The Virtual Theatre course is listed as an advanced course in the area of Computer Graphics. Within this area, the CS department also offers Graphics I, which focuses on fundamentals of graphics; Graphics II, which focuses on rendering and global illumination; Computer Animation, which takes a low level look at the algorithms and techniques of animation; and Procedural Shading. All graphics courses are co-listed for both Graduate and Undergraduate students. Students studying graphics are also encouraged to take AI for Interactive Environments, which focuses on the use of artificial intelligence methods in 3D environments, as well as the game programming courses offered by our Information technology department.

Graphics I is the only listed prerequisite for the course. Although it is helpful for students to have taken some of the more advanced graphics courses, this is not required as there is much flexibility in the work that any individual student will be tasked to do. In fact, since there are several required tasks involving areas that are non-graphics related (e.g. networking, audio), we do allow students with no graphics experience to enroll in the course with permission of the instructor. Such students should show an expertise in one of these non-graphics areas and an interest in graphics, games, or 3D environments.

4.2 Learning Outcomes

In designing the course for Computer Science students, we had the following goals in mind

- Provide students with a rich and relevant technical experience.
- Give students a taste of a real-world working environment. Thus, a major emphasis was placed on having students collectively work on a large-scale project from which any individual student will have the responsibility for a smaller piece.
- Define the project such that collaboration with artists is a requirement for success of the project as a whole.

Given these goals, the following student learning outcomes were defined for the course:

- Students will be able to describe the components of a distributed virtual reality system and the latest advances in designing such components.
- Students will be able to apply existing graphics/VR toolkits, APIs, and software packages in the construction of a large scale project.
- Students will be able to specify, design, implement, and document an integral component of a larger software project related to computer graphics and VR.
- Students will be able to successfully participate in a interdisciplinary team based project with responsibilities assigned within and between individual teams

Unlike other courses that involve capstone projects in gaming involving (e.g. [3][7]), an outcome of this course involves the application of a gaming engine as opposed to the creation of one. This is due, in part, to the short time span of the course (10 weeks) and the fact that the skills that we hope to develop in the students are addressed by collaborative nature of the course. Since the focus of the work for a particular individual is quite specific, the learning curve of working with the chosen gaming engine to address each task is manageable given the timeframe of the course.

4.3 Team Assignments

The primary goal of the Computer Science course was to build the Virtual Theatre layer as mentioned in Figure 1. Towards this end, students are divided into a number of teams; each team responsible for a particular technical component of this layer. A list of teams and team responsibilities is given in Table 2.

Table 2 - Responsibilities for student teams

Team	Responsibility
Asset Import	Creation of tools and classes for importing models from design team.
Audience Participation	Creation of software object representing audience members and providing capabilities for audience participation.
Audio / Music	Creation of a subsystem for distributed audio and music.
Character Motion	Creation of classes to enable playback of predefined animation sequences created by design team.
Motion Capture	Create classes for interface with motion capture devices.
Networking	Creation and management of networking infrastructure.
Staging / Lighting	Creation of software classes representing staging and lighting command and cues.

The instructor plays the role of technical director, keeping teams on track with regard to serving the ultimate needs of the entire production and the production schedule. Task requirements for work to be performed by each of the teams are predefined based upon the needs of the production. Given these requirements, each team must design a solution for the task at hand and develop an implementation plan. Students are pointed to relevant papers and articles from the VR, graphics, and animation literature to assist them in their design decisions.

Teams are carefully hand assembled by the instructor, based upon the interests and the expertise of the students taking the course in a given quarter. Graduate students are expected to take leadership roles in the teams. The goal is to have as many teams as grad students. With each course offering, enrollment has easily supported natural team assignments. On the occasion when grad leaders could not be assigned, the instructor stepped in as the director for the team.

4.4 Course Delivery

The course is given in a computer laboratory consisting of 20 Windows based PCs equipped with advanced graphics hardware. Software to be used by the class is preloaded on these machines. The software is also available for students to install on their own personal laptops, though the use of laptops is not a requirement for the course. In addition, students have access to our Virtual Reality laboratory, which houses specialized hardware dedicated to interface with the VR devices used during the performance. Currently, our list of VR devices include a single node Flock of Birds Motion Capture device, several datagloves, a head mounted display, and a full body motion capture system.

Classes are scheduled for 4 hours weekly, divided into 2 two-hour periods. Lectures are given only in first 2-3 weeks of course, with the purpose of familiarizing students with the processes of theatre, providing details on the given production being targeted, and instructing on the use the gaming engine and toolkit being utilized. The remainder of class periods is used as working sessions. Although it is expected that a good deal of the work will be done outside of class time, the working sessions assure guaranteed time when teams will have time together to meet.

It is important that each team makes steady progress as the quarter advances. With the goal of keeping the teams on track, each team is required to give a short demo periodically during the quarter ("checkpoints") illustrating the progress made. Each team leader, in conjunction with the instructor, will determine the expectations for the checkpoint demos.

The course culminates with a performance demonstration that takes placed during finals week. In preparation for this demonstration, team leaders integrate the deliverables from all of the teams are into a single unified framework. Assets created by the design team are imported and a demo performance is presented.

4.5 Student Assessment

Student assessment for the course is performed on two levels. The first involves direct instructor evaluation of student work. This is performed on a per team basis where each team is evaluated based on the design, implementation, and documentation of their assigned component given the requirements. Teams are also graded on the incremental demos shown during the checkpoint sessions. Secondly, students provide peer evaluations as input to the instructor. In these peer evaluations, students are asked to judge each team, as a unit and each member within their given team. In addition, grad students are evaluated on their leadership qualities in directing the work of the team.

5. RESULTS

We have been offering the Virtual Theatre course annually, in the Spring since 2004. For each course offering, a one-act improvisational theatrical piece was created as a target performance. A list is given in Table 3. The story of and interaction required by each piece was influence by the set of VR devices at our disposal at the time of the course.

Table 3 - Virtual Theatre Productions

When Offered	Production	Interface devices
Spring 2004	*What's the Buzz?*	Single node FOB motion capture device, 5DT dataglove keyboard / mouse
Spring 2005	*Getting By*	Full body motion capture keyboard / mouse
Spring 2006	*Critters*	Full body motion capture 5DT Dataglove P5 Dataglove keyboard / mouse

As a direct outcome of the course over the years, we have built two versions of the Virtual Theatre System (Table 4), each using a different gaming engine as a foundation. In the Spring of 2004, we built upon MUPPETS, a distributed, collaborative virtual environment (CVE) originally designed for enhancing student education in the areas of programming and problem solving [15]. We continued with MUPPETS in the Spring of 2005, building upon the work done in 2004. In an attempt to give students experience in working with a commercial engine, we switched to RenderWare Graphics[16] in 2005, using the RakNet library[17] for networking.

Table 4 - Virtual Theatre Systems

	Engine	Language	When used
V1.0	MUPPETS	Java	2004/2005
V2.0	RenderWare (graphics) RakNet (networking)	C++	2006

Enthusiasm among the students was overwhelming, greatly exceeding our expectations. In a sense, and quite unexpectedly, the theatrical metaphor extended to the overall feeling of the project. Consistently we've observed that by the end of the quarter, technical team boundaries spontaneously evaporated with all students working together equally towards the goal of completing the system. The project became more than a course, with the drive to finish mimicking that of stage hands on opening night.

Most of the collaboration between artists and the students in the class focused on importing and adapting the assets created by the design team for use in the gaming engine. It was of utmost importance that the artistic aesthetics of the final models be preserved given the constraints of the gaming system. Students had to work together to create innovative solutions to such issues, some of which involved modification of the models, whereas others were technical, involving the creation of software and tools for automating the conversion.

Interactive sessions, during which these problems were addressed, offered the richest collaborative experience. During these sessions, technologists learned the thinking process of the artists and learned how to specify and implement tools to meet the needs of these users. The artists, on the other hand, learned to adapt their work to the needs of a real time, interactive VR environment.

In addition to peer evaluations, students were also asked to evaluate the course. Table 5 lists some of the more popular responses, in order of frequency of replies, when students were asked "What have you learned in Virtual Theatre?" Note that although technical content is high on the list, many of the other goals of the course are well represented.

Table 5 -- Student response on what they learned

- Technical Content
- Integration on a Large Scale project
- The importance of deadlines and backups
- How to work with artists
- How to work independently and within teams
- Problem solving and the importance of communication
- That class can be fun

Finally, in several cases, the work started in the Virtual Theatre class has been used as the basis for a Graduate project in Computer Science. Aspects of the system have been extended as graduate work involving both motion capture and behavioral models.

6. CONCLUSIONS

We feel that collaborative nature of the course, in both the interaction between technical teams and between technologists and artists, resulted in an enhanced learning experience for all involved. Student work, excitement, and especially interactivity, consistently exceeded expectations. Defining student and team tasks as a portion of a larger project resulted in a communal feeling towards the project as a whole with all contributing and taking responsibility for the final product. Finally, placing he project in a theatrical paradigm, a natural structure for collaboration, further enhanced the interactive experience for both students and instructors.

We plan on offering the course again in the Spring of 2007. In addition, we are considering applying the same model and course structure to domains other than theatre, where interaction in a virtual 3D space would be appropriate.

7. ACKNOWLEDGMENTS

We'd like to acknowledge the contribution of our collaborators, Walter Wolf (stage direction) and Tina Chapman (actress). We especially appreciate the time put in by all the students who took the course, which far exceeded that, which was required. Special thanks go out to Jorge Diaz-Herrera, Stanley McKenzie, Jeff Pelz, Andy Phelps, Joan Stone, Paul Tymann, and Walter Wolf for their continued support of this work. This work was funded by an RIT Provost Learning Innovations Grant and a Faculty Development Grant from the B. Thomas Golisano College of Computing and Information Sciences at RIT.

REFERENCES

[1] Harris, E., 2003. *How to get a Job in Computer Animation*, Imprint Press, Raleigh, NC.

[2] Mencher,M. 2003. *Get in the Game! Careers in the Game Industry*, New Riders, Indianapolis, IN.

[3] Parberry, I., Roden, T., and Kazemzadeh, M. B. 2005. "Experience with an industry-driven capstone course on game programming". In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education* (St. Louis, Missouri, USA, February 23 - 27, 2005). SIGCSE '05. ACM Press, New York, NY, 91-95.

[4] Ebert, D.S, Bailey, D, 2000, "A collaborative and interdisciplinary computer animation course", *ACM SIGGRAPH Computer Graphics*, 34 (3), August 2000, 22-26.

[5] Hunkers, D., Levine, D.B., 2003, "The science of images: a cross-disciplinary introduction to the field of 3-D computer graphics", *Educators program from the 30th annual conference on Computer graphics and interactive techniques*, San Diego, California, 1-4.

[6] Zimmerman, G.W, Eber, D.E., 2001, "When worlds collide!: an interdisciplinary course in virtual-reality art", *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*, Charlotte, North Carolina, 75-79.

[7] Jones, R. M. 2000. "Design and implementation of computer games: a capstone course for undergraduate computer science education." In *Proceedings of the Thirty-First SIGCSE Technical Symposium on Computer Science Education* (Austin, Texas, United States, March 07 - 12, 2000). S. Haller, Ed. SIGCSE '00. ACM Press, New York, NY, 260-264.

[8] Zeller, D., 1999. Definitions and Directions of the Theatre. In *Theatre in Cyberspace*, S.Schrum, Ed. Peter Lang Publishing, New York, NY, 21-27.

[9] Catron, L.E., "What Theatre Majors Learn", http://lecatr.people.wm.edu/majorslearn.html.

[10] Laurel, B.,1991, *Computers as Theatre*, Addison-Wesley, Reading, MA.

[11] Lee, W.J. 2003 "Computer Graphics and Theatre". In *Proceedings of the Educators Program of the 30th annual conference on Computer graphics and interactive techniques*, San Diego, CA, August 2003, ACM Press, New York, NY, 1-4.

[12] Springel, S. 1998. "The Virtual Theatre: immersive participatory drama research at the centre for communications research, Cambridge University". In *Proceedings of the sixth ACM international conference on Multimedia Technologies for interactive movies*, Bristol, UK, 43-49.

[13] Geigel, J, Schweppe, M. 2004. "Theatrical storytelling in a virtual space". In *Proceedings of the 1st ACM workshop on Story representation, mechanism and context*, New York, New York, October 2004, B. Barry AND K. Brooks, Eds. ACM Press, New York, NY, 39-46.

[14] Marino, P, 2004. *3D Game-Based Filmmaking: The Art of Machinima*. Paraglyph Press, Phoenix, AZ.

[15] Phelps, A, Bierre, K. ,Parks, D. 2003. MUPPETS: multi-user programming pedagogy for enhancing traditional study. In *Proceeding of the 4th conference on Information technology education*, Lafayette, IN, 2003 , J. BREWER AND J. MENDONCA, Eds. ACM Press, New York, NY, 100-105.

[16] Criterion Technologies, Ltd., RenderWare Graphics, Guildford, Surrey, UK, 2000, http://www.renderware.com

[17] Rakkarsoft, RakNet – Multiplayer Game Network Engine, http://www.rakkarsoft.com/

Teaching Game Design through Cross-Disciplinary Content and Individualized Student Deliverables

Ursula Wolz
Computer Science and Interactive Multimedia
The College of New Jersey
Ewing, NJ 08628
+1 609 771 7266
wolz@tcnj.edu

Christopher Ault
Interactive Multimedia
The College of New Jersey
Ewing, NJ 08628
+ 1 609 771 2236
ault@tcnj.edu

Teresa Marrin Nakra
Music
The College of New Jersey
Ewing, NJ 08628
+1 609 771 2759
nakra@tcnj.edu

ABSTRACT

Video game development is used to teach collaborative software engineering principles. However, when the collaboration is exclusively between computer scientists, a balkanized perspective is unintentionally promoted. A multidisciplinary faculty addressed how to exploit video game development as a vehicle for a collaborative cross-disciplinary experience in technology development for upper-level students from our contributing majors. This paper addresses the issues of curriculum structure, student assessment and cross-disciplinary team teaching. We present a dual model of *cross-disciplinary content* and *individualized deliverables* that allows each student to determine how narrowly or broadly to focus his/her personal learning.

Categories and Subject Descriptors

D.2.10 Software Engineering: Design – *methodologies*
K.8.0 General Games

General Terms

Software Engineering, Games

Keywords

Game design, Game architecture, Project management, Multidisciplinary computing, Computer science education

1. INTRODUCTION

There is a resurgence of interest in using games as a domain of application for teaching computer science foundations. Video game design, development and architecture is also gaining credibility as a discipline in its own right within computer science, drawing from such disciplines as computer graphics, artificial intelligence and networks. Video game design can also be used as a vehicle to teach software engineering principles both in the construction of game engines and to create games themselves [2, 3, 4]. However, an unrealistic and artificial environment is created when the software engineering experience in game implementation is limited to computer science students. The construction of a fully robust video game is dependent upon expertise from disciplines outside computer science, including creative writing, music composition, sound technology, theater production, digital 3-D art, cinematography, and character animation. To a large extent game design and implementation is a compelling model for a more global question of how to teach skills in cross-disciplinary technology development. As eloquently stated in “The World is Flat” [5] the skills necessary for collaborative communication across disciplines will be critical to the continued success of the American workforce. This is coupled with the timely resurgence of interest in video games themselves as tools for education [7]. There is a predicted need for computer scientists with expertise in game development who have practical experience in cross-disciplinary collaboration.

At our institution, a multidisciplinary faculty raised the question of how to exploit video game development as a vehicle to provide a collaborative cross-disciplinary experience in technology development for upper-level students from a variety of majors in a single, year-long cohesive course. This paper reports on our experience with this approach, addressing the major issues of curriculum structure (Section 3), individual student assessment (Section 4), and cross-disciplinary team teaching (Second 5).

In particular we present a dual model of *cross-disciplinary content* and *individualized deliverables.* This supports assignments that allow each student to determine how narrowly or broadly to focus their personal learning within the breadth of disciplines. Furthermore, this approach provides a model for student-centered learning that attempts to dismantle the “silo” effect of undergraduate education that creates balkanization of disciplines.

Our original goal was a suite of courses that would share the objective of creating a single 3-D, multiplayer virtual world with a robust storyline supported by high quality sound and music. We envisioned that students would collaborate based on highly developed expertise in their chosen fields. We anticipated that established models of project management, wherein team members report through hierarchical organizations of skills-based accountability, would suffice to facilitate design and production.

2. INSIGHTS FROM OUR PILOT YEAR

Academic year 2005-2006 (AY05-06) was our pilot year, in which approximately 20 students per semester were enrolled in courses that supported this enterprise. A once-per-week four-hour workshop allowed students to participate in the collaboration through specific roles (summarized in Table 1). This was intended to model the professional game development environment, if not the 24/7 intensity of the industry. In 30 weeks we produced one

Table 1: Student Roles in Game Development

Title	Deliverables	Responsibilities
Art Director	Finalized and stylistically consistent art, story and sound	Overseeing the work of the art and writing staff
Tech Manager	Solutions for troubleshooting technical issues	Overseeing the tech staff and liaison to art staff
Project Manager	All the assets, completed on time and in the right format	Managing the workflow of the project as a whole,
Prop and Scenery Modelers	All non-character models and background models	Working with 3-D modeling tools
Character Modeler	Two main characters, three secondary characters	Working with 3-D tools to model, and animate all characters
Level Designer	Maps of all games levels, identifying interactivity triggers	Creating level maps and trigger points
Texture Designer	Textures for props, characters, and all other surfaces	Working with 3-D tools to create "skins"
Lighting Designer	Lights placed appropriately throughout the levels	Working with game engine tools to insert lighting
Story Analyst/ Writer	Game implementation consistent with established story	Creating dialog and directing voice actors
Gameplay Writer	Actions associated with interactivity triggers	Identifying all trigger points and the actions
Documenter	Web site for informational and publicity purposes	Standardizing documentation including tutorials
Support Software Manager	Installation and maintenance of all support software	Maintaining software including exchange server
Composer	All music necessary for all levels	Creating music that enhances the gameplay
Sound Technician	All sound effects including dialog	Ensuring integration of sound effects and dialog
AI Technician	All logic and algorithms for game implementation	Implementing logic for the game
Interface Designer	2-D user interface elements	Creating an intuitive player interface
Media Coordinator	All assets developed by artists are correctly installed	Inserting all assets into game

complete high resolution and one rough cut level of a "first person shooter" with full sound and high quality music. We used Valve's "Source" game engine. We also produced a story bible and sample assets for a full 3-D multiplayer game (see http://www.tcnj.edu/~Games.)

2.1 Limitations of a Product-Centered Model

Based on analysis of student work throughout the year we concluded that our assumptions about course structure and organization were off the mark for a pedagogic environment. Our mission is to educate students. The goal of producing a fully robust game was merely the vehicle through which to teach concepts and skills. Our pilot showed that the kind of narrow task assignment we envisioned, where students are segregated by skills (e.g. by major from the contributing disciplines), severely constrained students' ability to grow, learn and simply communicate in unanticipated ways.

Furthermore, we confirmed work by Constantine and Gillard [1, 6] that traditional hierarchical models of team organization with linear models of time management are insufficient. They do not support the cross-disciplinary communication necessary to create software as complex as a video game. We initially identified "programmers", "artists", "writers" and "sound composers" whose work would be integrated through production leaders (who emerged from the ranks.) In practice we saw, not unexpectedly, that individual artists needed to collaborate with individual programmers and writers. Clean boundaries between groups clustered by expertise thwarted the development process, reinforcing balkanization and discipline-based prejudice. Our simple models of collaboration and group expectations based on skill sets for deliverables were insufficient. They did not provide students with the rich immersive experience required to meet our initial pedagogical goals of collaborative, multidisciplinary software development.

This also made assessment problematic. We had clear and established criteria for judging artwork, writing and programming. However, this also reinforced balkanization. Traditional assessments of specific deliverables discouraged students from collaborating because their role in producing a specific outcome wasn't always clear. It also discouraged students from taking risks and trying their hands at developing skill sets outside their area of expertise. Why produce a terrible 3-D model that wouldn't be used in the game anyway if you can make a clear and valuable contribution with a novel algorithm? Furthermore, how does an instructor measure the worth of a deliverable when everyone is making a unique contribution?

Our original course framework envisioned a fall semester experience in which students learned the theory and craft necessary to spend the second semester in a large, single project group collaboration. For the fall semester we anticipated at least five "courses", one in each of the contributing disciplines (Digital Art, Communications, Computer Science, Interactive Multimedia, Music and Writing.) Given the size of the student population from which to draw (we are mid-sized primarily undergraduate college) as well as a realistic expectation for managing a single class project, we could not support faculty load or student enrollment at reasonable levels with so many courses. Also, conceptually we were balkanizing the disciplines, and needed to create a truly multidisciplinary experience that would meet load constraints.

A significant anticipated problem was how to recruit the right balance of students with diverse expertise and then exploit the emerging technical and leadership skills of the group that came together. An added complication is that we could not assume that all students who registered for the fall class would continue in the spring. The course offerings were constrained by upper-level in-major requirements and thus we could not necessarily expect a full-year commitment from all students.

2.2 The "Silo" Model Creates Balkanization

The segmented or "silo" model of contributing disciplines was problematic with regard to faculty load, course scheduling, and assignment of student seats. The topic foci were not equal in required breadth and depth. Faculty availability to teach within the suite was not consistent, nor was there a balanced need for faculty expertise. To implement our idealized model we would overburden some faculty (e.g. supporting 30 animators in one class, while supporting a single music composer in another.) Furthermore our assumptions about domain expertise did not account for subfields within a discipline. For example, a networks expert might require extensive professional development to teach the requisite knowledge in artificial intelligence. Similarly, a cinematographer would need training to teach 3-D animation.

We sought a coursework model in which the content and skills presented could be taught with broad strokes appropriate to students with a range of expertise, while requiring students to delve into content and develop skills commensurate with their background and interest. Put differently, student assessment would be based on a metric that balanced generic accountability (e.g. everyone writes journals in response to reading assignments) with highly individualized targeted measures of skills development. Furthermore, to de-balkanize the disciplines we needed to reward students willing to leave the comfort-zones of their major, and take risks by completing assignments from other disciplines. Their experience out of their major should create an enthusiastic respect for the work of others, and give them a vocabulary through which to communication across disciplines. Such communication is key to effective cross-disciplinary collaboration, which in turn is key to reducing the silo effect.

3. CROSS-DISCIPLINARY CONTENT

In May 2005, seven faculty members, representing six undergraduate majors (Art, Communications, Computer Science, Media and Writing) and including CS faculty in artificial intelligence, interface design and networks, participated in an intensive workshop to design a year-long experience in collaborative cross-disciplinary, 3-D, multiplayer video game development. We articulated the need for content teaching in five overarching areas: game genre, interactive storytelling, game engine architecture, production management and the social and ethical impact of games. We also identified three primary technical areas: character animation, interactivity and artificially intelligent agents. Furthermore we identified three secondary technical areas: sound composition (including dialog and music), theater production (including staging and lighting), and computer networks (for massively multiplayer games.)

Table 2 illustrates our course structure, which is a mix of (1) formal lecture of cross-disciplinary content, (2) studio for technical skills development (3) demonstration/practice in support software (e.g. Maya, Reason, XSI), and (4) workshop for product development. We organized the two-semester experience to front load content and skills development. Table 2 provides an overview.

The class is scheduled as one four-hour block per week for two fifteen-week semesters. Lecture topics typically occur during the first 90 minutes. On lecture days the remaining time is used for workshop, tutorial and studio. Whole group demo days (social ethics days, demonstration, focus group/product testing) have a structured agenda with assigned responsibilities.

Significant course content and skills development occurs outside of class through (1) assigned readings, (2) small group meetings, (3) small group and individual tutorials (with both faculty and students as tutors), (4) individual and small group design sessions, graphics development, and sound/music recording sessions. We also use the Microsoft Sharepoint server as a conduit for materials development and exchange. Students develop all game assets, most technical tutorials, as well as documentation and timelines. Faculty materials are primarily lecture notes and assignments.

Table 2: Year-long Syllabus and Faculty Roles

Week	Presentation Topic	Assignment Type	Student-Centered	Faculty Participation
1	Game genre		P	IR
2	Interactive storytelling	1. Art	P	G
3	Game architecture	2. Tech	P	IR as G
4	Project management	3. Mix	P	IR as G
5	Animation	4. Art	P	IR as G
6	Interactivity	5. Mix 6. Tech	P	G
7	Agents	7. Tech	P	G
8	Workshop: Story & game design		F	G & IR
9	Social & ethical impact I		N	IR
10	Sound, dialog, music	8. Art	P	IR as G
11	Theater production design	9. Mix	P	G
12	Networks for MUDDS	10. Tech or Art	P	G
13	Social & ethical impact II		N	G & IR
14	Workshop: deliverables review		F	IR
15	Focus group: deliverables demo		N	G & IR
	SEMESTER BREAK			
16	Story presentation, roles discussion		P	IR
17	Timeline, responsibility articulation		P	IR
18	Workshop		F	IR
19	Social/ethical impact analysis		P	G & IR
20	Workshop		F	IR
21	Focus group: Low res demo		N	G & IR
22	Timeline, deliverables review		P	IR
23-25	Workshop		F	IR
26	Focus group: High res demo, Social/ethics impact analysis		N	G & IR
27	Timeline, deliverables, product review		P	IR
28-29	Workshop		F	IR
30	Product unveiling		N	G & IR

Legend: F – Fully student-centered; P – Partially student-centered; N – Not student-centered; G: Guest lecturer; IR: Instructor of record

The formal lectures give an overview of essential topics based on assigned readings from a mixture of textbook genres (one computer science, one digital art/writing, one reflective.) The lectures are designed to make the material accessible to students with little background, so a programmer can appreciate the complexity of animating a character, for example. The lectures are also designed to give a novel perspective to majors in that field. For example, a computer science major who has taken artificial intelligence learns to appreciate the impact of agent technology on gameplay.

We fully integrate the social and ethical impact of video games into the curriculum through class discussions of storyline, character development, visual images and stereotypes. We also focus on these issues in two full sessions in the fall semester. The students create a forum in which timely issues are discussed. One session is a closed dry run. The second session is well publicized and open to the public. We return to these issues in the spring semester as we evaluate the implementation stages of the game.

4. INDIVIDUALIZED DELIVERABLES

Table 2 lists the degree to which learning in class is individualized. A "fully" individualized session occurs on studio/workshop days. A "partial" session occurs on lecture days when approximately 1.5 hours are devoted to lecture and 2.5 to studio/tutorial/workshop. The presentation days contain no individualized instruction. Only 19% of instruction over the year is lecture based, while 20% is student presentation and 61% is studio/workshop. If all students were developing similar skills and submitting similar deliverables, this would be a traditional art studio. However the studio/workshop time may involve a variety of tasks from the contributing disciplines. This requires a novel approach to the support and assessment of student deliverables.

Student work in AY05-06 led us to both constrain and loosen course requirements. Undergraduates tend to cram for tests, and deliver less than optimal results for project deadlines, hoping for extensions. This style is a severe detriment to projects with heavy task interdependency. Time analysis surveys, administered both at the middle and end of in AY05-06 showed that half of the students had little ability to manage deliverables, and a third insufficient time management skills for successful collaboration.

In reflective essays, students asked for help in time management and more direct accountability of weekly deliverables. The faculty concluded that students needed to be explicitly taught benchmark and dependency analysis skills, and needed carefully guided practice in fulfilling a weekly action item.

These results led us to redesign student assessment in both semesters. We defined a new methodology of *individualized deliverables* that provides a highly personal set of expectations. We can assist students in constructing a set of expectations that meet their personal learning goals (e.g. everyone doing something different), while fostering task dependencies that enhance collaboration in a safe way (e.g. one student's grade is not critically dependent another's work.) In AY06-07 we are combining generic expectations with an individualized contract of student deliverables. Final grades are assigned as follows:

Fall Semester
20% Journal entries: acceptable, insufficient, or not completed
20% Final take home exam, 20 questions based on journal entries
20% Lecture follow up assignments choose five out of 10
50% Project deliverables: percentage effort on three of four projects

Spring Semester
15% Journal entries: acceptable, insufficient, or not completed
20% Final take home exam, 20 questions based on journal entries
15% Benchmark recording and weekly action item report:
50% Project deliverables: percentage effort on three of four projects

4.1 Generic Assignments for All Students

All students are required to complete reflective writing assignment via journaling and a take home essay final exam on (1) course content, (2) personal skills development, (3) social and ethical impact (4) collaboration and communication.

All students are also required in both semesters to present an individual deliverables contract. This is an evolving document that includes a percentage breakdown of (1) selected assigned exercises, (2) contribution to large projects, and (3) timeline and dependency analysis.

In the first semester, correspondence on individualized deliverables occurs through documents submitted to a course management system "drop box", through email correspondence and face-to-face meetings with the instructors of record.

In the second semester, we use an accountability technique developed at the end of the AY05-06. Each workshop session begins with a review of action items from the previous week. Each student checks in by reporting on the status of the item and whether (1) a deliverable is ready for full group demonstration and evaluation, or (2) is ready to be included in the next version of the game. At the end of the workshop, each student checks out by identifying his or her action items for the week to come.

4.2 Breadth vs. Depth of Skills Development

In the fall semester, which is more content based, students must choose five of 10 lecture summary assignments from the contributing content areas. Table 2 shows them broadly categorized as "technical", "artistic" or "mixed." These exercises create opportunities to de-balkanize perceptions of skill sets. Students can select assignments that let them remain safely within their general area of expertise, but they must complete at least one exercise at the edge of their safety zone. For example, a computer science major could play it safe by selecting assignments 2, 3, 6, 7 and 10 tech, staying well within the bounds of computer science. A more adventurous student might select 1, 2, 3, 5 and 10 tech, adding a few exercises that are mixed or art. A student willing to significantly broaden her background might select all of exercises outside her safety zone, for example 1, 4, 8, 9, and 10 art.

4.3 Large Project Collaboration

Large project collaboration provides the focus of both content mastery and skills development. Such work is crucial to de-balkanizing the constituencies and providing an environment that fosters cross-discipline communication. In the context of highly individualized roles (see Table 1), we ask students to choose their participatory role and identify deliverables in a highly personal way. In order to prepare them for the second semester, when their contribution will be critical to the whole, we ask students in the first semester to split their personal deliverables between at least three of four projects. They tell us what percentage of their project grade will come from their contribution to each project.

Table 3 summarizes the commitments made in fall '06. All students are required to commit at least 10% to the "ethics" forum. The "story bible" requires designing the game to be implemented in the spring. The "enhance last game" develops skills in our SDK, pipeline processes and support tools. The “toy engine” project provides in-depth experience in augmenting a game engine. As shown in Table 3, students approached the deliverables from perspectives ranging from deep commitment (e.g. story bible at the maximum of 80%) to even distribution (all four projects at 25%). Students are identified as “tech” or “art” based on their registration in one of two co-listed courses.

Table 3: Percentages of Individual Deliverables

Student	Ethics	Toy Engine	Enhance Last Game	New Story Bible
A1	10		40	40
A2	10		70	20
A3	20	10		70
A4	30	10		60
A5	80		10	10
A6	10		60	30
A7	10		10	80
A8	10	20	70	
A9	**10**		**10**	**80**
A10	20		40	40
T1	15	50		35
T2	10	45	45	
T3	**25**	**25**	**25**	**25**
T4	25	40	35	
T5	10	60		30
T6	10	60	20	10
T7	20		50	30
T8	10	50	40	
T9	10	60		30

5. FACULTY ROLES

A single instructor cannot begin to manage a course sequence such as this. Fortunately our institution has moved toward a transformed curriculum in which team teaching and multi-disciplinary collaboration are encouraged.

In the fall ’05 course, a single instructor of record supervised the projects and gave only two of the formal lectures. Guest instructors presented the other lectures with externally funded stipends. In fall ’06, two faculty-shared faculty load of a single section. The guest-to-instructor of record ratio decreased as reflected in Table 2. Other models of instructor of record to guest lecturer are certainly possible. However, broad representation of contributing disciplines is needed to prevent balkanization.

In spring ’06, three faculty members shared responsibility for two separate sections (thus doubling the total allotment of faculty hours.) Our rationale for offering a single section in the fall and two in the spring was the significant increase in instructor of record participation as seen in Table 2. A problem remaining at our institution is how, in the future, to adequately compensate guest instructors without external funding, in both their roles as deliverer of instruction and formal evaluator.

Table 2 also highlights the novel set of responsibilities of the instructors. The primary instructors are not over-arching content experts, but rather production managers responsible for accountability during the workshop sessions that comprise 60% of the total contact time. Responsibility for articulating individual learning is a collaborative exercise between instructor and student. Assessment shifts from traditional grading of standard deliverables (including test answers) to analysis of time management, skills development and highly personalized demonstration of content mastery. The payoff for instructors is that this style of grading is far more satisfying.

6. SUMMARY

Analysis of student final exams from AY05-06 as well as journals from fall ’06 suggest that we are successfully integrating course content across disciplines in a manner that de-balkanizes the disciplines critical to video game development. Student deliverables for the AY05-06 game demonstrate significant cross-disciplinary contributions and consequent skill mastery.

Over 80% of the students in the fall ’06 class are successfully answering the reflective questions on specialized topic lectures, are relating assigned readings to the lectures, as well as to their assigned project work. There is evidence in their writing that they see the complexity and contributions of the various disciplines.

Of more significance are the reflective writings with regard to collaboration and communication. Last year’s students demonstrated deep understanding of the critical need for good communication across disciplines, and the value of at least a superficial understanding of disciplines outside their own major.

Game design and development will never be a field exclusively within the domain of computer science. Nor will it become a field entrenched in digital art or interactive storytelling. Our two-year experience in collaborative multidisciplinary teaching suggests that game design is indeed a field for the 21st century, that requires truly global, diverse communication skills.

7. ACKNOWLEDGMENTS

Our thanks to our colleagues in the Game Design Project at The College of New Jersey: Kim Pearson, Phil Sanders, Terry Byrne, Miroslav Martinov, JiKai Li, and Anita Allyn. Thanks also to the students who provided invaluable constructive criticism of our methods. We are also grateful to Microsoft Research, and especially John Nordlinger.

8. REFERENCES

[1] Constantine, L. L. Work organization: paradigms for project management and organization, Communications of the ACM, Volume 36 Issue 10, October 1993, pp 35-43

[2] Claypool K., and M. Claypool, Teaching software engineering through game design, ACM SIGCSE Bulletin , Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education, Volume 37 Issue 3 June 2005, 123-127

[3] Coppit, D. and J. Haddox-Schatz. *Large Team Projects in Software Engineering Courses*. Proceedings of SIGCSE 2005, (St. Louis, Missouri, February 2005), 137-141

[4] El-Nasr, M.S.and B. K. Smith, Games: Learning through game modding Computers in Entertainment, Vol. 4 (1), Jan. 06, pp 1 -13

[5] Friedman, T. L. The World is Flat: a brief history of the twenty-first century. Farrar, Straus and Giroux, New York, 2005.

[6] Gillard, S., Managing IT projects: communication pitfalls and bridges, Feb. 05 Journal of Information Science, Vol 31 (1) 37-43

[7] Federation of American Scientists, *Summit on Educational Games, Harnessing the power of video games for learning*, http://www.fas.org/gamesummit/, October 2006.

[8] Wolz, U. and M. S. Pulimood, *An Integrated Approach to Project Management through Classic CS III and Video Game Development*, to appear in the SIGCSE 07, March 7 -10, 2007, Covington, KY

A Tale of Two Classes: On Interdisciplinary Collaboration in Games Education

Arnav Jhala
Department of Computer Science
North Carolina State University
ahjhala@unity.ncsu.edu

R. Michael Young
Department of Computer Science
North Carolina State University
young@csc.ncsu.edu

Timothy Buie
College of Design
North Carolina State University
tim_buie@ncsu.edu

ABSTRACT

In this paper we present our experiences teaching a computer game design and development class in the Computer Science department at NC State University. This class is co-taught with a studio class in the NCSU College of Design. The two classes are taught in parallel, the computer science class focuses on programming and the design class focuses on game art. Students from both classes work on several common assignments as well as a semester long project in teams of 5 to 7 students. This paper documents our observations from four semesters spent teaching this course. Two of the four were taught without the involvement of design students and two with them. The unique feature of our classes is that while the computer science and design courses are taught independently, both classes' students work on common assignments and gain experience with interdisciplinary communication in addition to the discipline-specific content taught in each of the classes.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education]:** Computer Games Education, Interdisciplinary Education.

Keywords

Computer Games Education, Interdisciplinary Education.

1. INTRODUCTION

There were good programmers; there were good artists. There was a class in the computer science department for game development; there was a class in the industrial design department for development of game art…

Traditional Computer Science Departments prepare students for a software development career. Most students with a computer science degree have a strong background in programming but little appreciation for the arts or the artistic process. Programmers in the game industry have to work closely with designers, writers and artists. To better prepare our students for this professional environment, we designed two senior-level undergraduate courses – one in computer science and one in industrial design – so that students from both disciplines would be exposed to close interaction with each other while focusing on the joint creation of a game project. This paper focuses on the design and subjective evaluation of the computer science course.

2. STRUCTURE OF THE COMPUTER SCIENCE CLASS

The official description of CSC481-Computer Game Design and Development[5] is the following reads:

"An introduction to the technologies and practices underlying computer and console game development and the principles involved in effective game design and production. Topics include computer game graphics, sound and audio, level design, principles of gameplay, interactive storytelling, character control and artificial intelligence, user interface design. Programming project required. "

CSC481 at NCSU is an introductory class in game development. It is a senior level class whose sole prerequisite is the undergraduate data structures class (typically taken during the sophomore year). Students taking CSC481 are expected to have taken at least an intermediate level programming class and to be familiar with Object-Oriented Programming techniques. This course complements other computer science courses like Computer Graphics and Artificial Intelligence.

The class is taught in four phases. The first quarter of the class is devoted to familiarizing students with the Unreal Tournament 2004[3] game engine. During the second and third quarters of the course, students are introduced to various aspects of game development including Game Design, Graphics, Artificial Intelligence, Gameplay, and User Interface Design. In the fourth quarter, students work in teams of 5 to 7 on development of a game project. For the class assignments, students modify the implementation of an existing game engine. This frees students from the need to implement low level code for graphics, artificial intelligence, networking etc. and focus exclusively on better practices in design and finish of the game in a collaborative effort.

2.1 Choice of Game Engine

Our choice of the game engine for the introductory course on game development was based on the following factors: ease of learning, mod support, strength of development community and availability of development tools. The Unreal Tournament engine allows users to extend the game's functionality by exposing a scripting language called UnrealScript that provides a limited number of access points to the underlying API. UnrealScript is a well-designed object-oriented Java like language with additional features to support game programming. It is easy to learn for students who have a background in object-oriented programming.

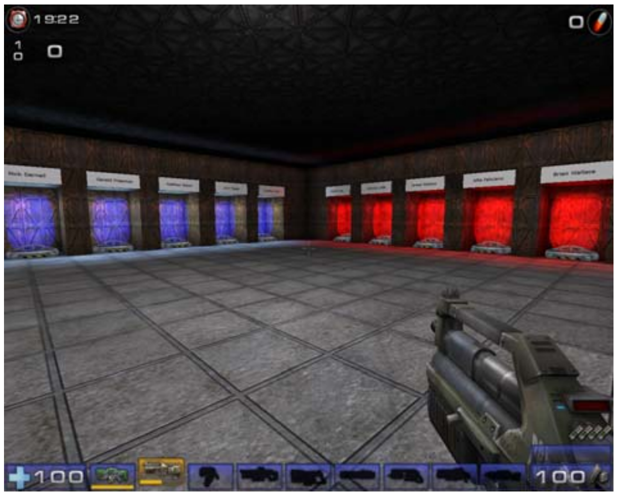


Figure 1: Level Design assignment with portals leading to each team member's part of the level, Spring 2005

There is a large informal development community and freely available stable tools for development. Recently, reference books for unreal programming have also been published[1]; video tutorials describing different aspects of modifying the unreal engine are also provided with the commercial off-the-shelf version of Unreal Tournament and are also available for free download via the internet[2].

UnrealScript is strongly object oriented. This gives students hands-on experience working with a large organized object oriented code base and encourages the development of skills that are transferable to other areas of computer science.

2.2 Assignments

The majority of class assignments are programming projects designed to draw from important aspects of game programming. They are also designed to familiarize students with both the commonly used and advanced features of the Unreal engine. Each student works through individual assignments related to a range of different areas of game development. This ensures that each student is well equipped to choose and handle the role that they get assigned to in their final project teams. This effort also provides them with an understanding of the scope of the work to be performed by each of their project partners fulfilling other roles in the team.

There are seven types of assignments given during the course of the semester; specific assignments are described in order in the following sections.

2.2.1 Game Review

This assignment is given during the first week of class and students are given a game to review. The game to be reviewed is a student project from another institution or from one of the previous years of this class. One of the games chosen for the review assignment was "That Cloud Game" developed at the games class in the University of Southern California[7]. This assignment has a two-fold advantage. First, it sets forth the instructor's expectations from the students by showing another student project with the same scope. Second, students are able to critically review the game and start thinking about improving on certain negative aspects of the reviewed game for their own course project.

2.2.2 Weapon Mod (Introduction to UnrealScript)

The purpose of this assignment is to get the students started with setting up the development environment for Unreal. Students create a weapon that selectively damages AI characters in the game and not human players. This assignment introduces them to the unreal programming pattern (subclass and override) that involves identifying how the game loop works and where to locate functionality to override. It also introduces them to the Unreal Object hierarchy and relationships.

2.2.3 Path Planning (Artificial Intelligence)

Path planning is one of the basic algorithms that is needed in most games. Although, unreal provides it's own path planning data structures and functions students have to use custom path node objects and maintain their own connectivity graph to compute all pairs shortest paths. This assignment serves as an example of AI programming. Students are expected to learn how to set up their own data structures and are encouraged to implement optimal algorithms.

2.2.4 Heads-up Display (HUD)

The design of the HUD assignment serves two purposes. It introduces students to the design of Heads-up display and its implementation on the unreal engine, and also acts as an example of creating a new game type on the engine. For this assignment students are tasked to implement the following features:

- Modify (improve) the design of the Heads-up Display so that it is less intrusive than the default unreal version and describe the design process and reasoning.
- Implement a new game type called the GrudgeMatch (a type of deathmatch) where the player keeps track of the opponent who has inflicted the most damage. The HUD of the player identifies the current grudge and maintains a tracking mechanism so that the player can locate and take down the grudge.

The main goal of the HUD assignment is to make it functional and ensure that it is non-intrusive and easy to understand rather than making it look pretty.

2.2.5 Project Design Document and Pitch (Group Assignment)

Students are assigned to groups of 5-7 by the instructor based on their interests and background. The team is given two weeks to flush out a game idea. This idea is submitted as a design document. The design document includes assignment of roles to each team member and details of three milestone deliverables. The suggested make up of the team positions is: Lead, AI Programmer, HUD programmer, Level Designer (either from Design or from CS), Menu/Gameplay programmer, 3D Artist (from Design class), 2D Artist (from Design class). Teams also prepare a marketing pitch that they present in class to all students. Grading is based on structure of the documents, detail, and quality of presentation.

2.2.6 Level Design (Group Assignment)

The level design assignment has been offered as an individual assignment on two of the occasions and as a group assignment in two classes. The minimum requirements for the level design assignment for each individual are:

- The level must be based on a theme (like fire, industrial, etc.)
- It should be a 4 to 6 player Deathmatch level.
- It should contain at least 3 sections/rooms that are laid out along vertically at least two levels.
- There should be at least 2 hallways, 9 textures, 10 static meshes, 3 weapon and health pickups each and 2 spawn points.

Extra credit is given to students who go above and beyond the basic requirement. Creative examples include the use of modified gravity or fluid volumes, portals, and strategic positioning of weapons and pickups for enhancing gameplay. For the group assignment, students are expected to base their complete level on a single theme with indicators for each individual's portion of the level. This assignment improves communication between team members, as they have to work closely to get their level geometry and theme to maintain consistency after merging all the parts. We have observed that even in this assignment some teams have found creative ways to combine different maps (Figure 1)

2.2.7 Project Milestones (Group Assignment)

Projects are group assignments that span over three quarters of the class. Projects are submitted in four parts: An initial design document and marketing pitch, two intermediate milestones, and one final showcase presentation. Each milestone contains Low, Target and High deliverables for each aspect of the game (AI, Gameplay, HUD, Modeling, Level Design, etc.). Deliverables are assigned to individual team members and deadlines for individual deliverables are set based on dependencies between modules. The team lead is assigned the responsibility of maintaining communication channels between teammates and integrating code and art assets from different sources. Two grades are given to students for each milestone, a team grade and an individual grade. The team grade reflects the level of integration, communication and the collective effort in organization. The individual grade reflects the quality of the specific tasks given to each individual. It is often difficult for 5 to 7 students working in the same team to find common time to meet frequently. In order to support face-to-face communication one day each week in the second half of the semester is dedicated to projects as a lab-meeting day.

The highlight of the course is the final showcase. All the games from the class are presented in a public showcase at the end of the semester. Students first give a presentation about their games and then the audience is given a chance to play the games in a LAN party. Having a showcase with a large audience motivates the students to go the extra mile in creating a finished product. This is something that not many academic classes offer.

For the projects, in addition to the submitted work we also collected peer evaluations from students working on the same team. Peer evaluations ensured that we could identify any contributions of students that were outside of their assigned tasks. We also gave credit to students who showed exceptional commitment towards their project and contributed to resolving any communication gaps among their teammates.

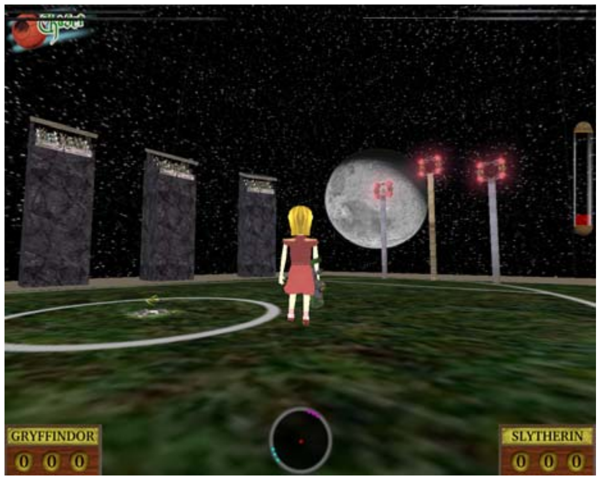


Figure 2: Quidditch, Spring 2003

3. STRUCTURE OF THE DESIGN CLASS

Seniors in College of Design take the ID400 class to learn character and asset modeling for games. Most of the students have at least some 2D or 3D art creation background. Design students, in association with Computer Science (CSC 481) students, and assisted by local industry professionals, learn to design and produce resources such as 3D digital models, animations, environments, etc., to be used in real-time interactive settings. In addition to expanding their creativity, students also gain basic education in, and understanding of, the uses and potential of real-time digital technology. Rather than somewhat passively playing games, or similar digital entertainments, students actively design and develop new content and resources to push art and technology in new and exciting directions.

4. INTERACTION BETWEEN CS AND DESIGN

The overall quality of games produced by our students is significantly improved when the class is taught in conjunction with the Industrial Design course. This is partly due to the creativity that they bring to the teams' process of game design and partly due to the increased range of possibilities for game design and game play when ID students create custom art assets for the games. Maintaining effective communication between 5 to 7 students during an entire semester is a challenging problem, however. This problem is made more difficult by the physical separation between the locations of the classes (Computer Science and Industrial Design are located on separate campuses two miles from one another). In order to reduce the impact of physical separation between departments, we set aside one class meeting each week for team meetings, and we also provide shared file space on the class server and access to collaboration and project management tools (e.g., wikis and version control software).

Designers and software developers speak very different languages. They are unaware, in most cases, of the constraints imposed on their work by other's respective domains. Students have often found that implementing certain designs is either too much work, not entirely possible in the engine, or is detrimental to performance of the game. There are different constraints on development of art assets when they are created for use in an

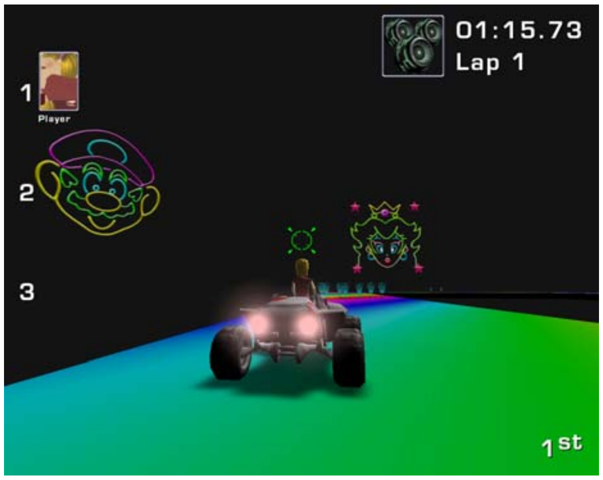


Figure 3: Strong Desire For Quickness, Spring 2006

interactive environment than when they are created in a non-interactive environment. What makes for effective art does not necessarily make for an effective rendering on a game engine. Artists and programmers have to work together in using standard naming conventions and agreeing on different parameters; for instance, in specification of character animations the frames of the walk animation cycle must match the speed of the character when placed in the game. Programmers and artists must work together so that the final result looks good and works as expected.

We present some case studies of games that were created as part of this course over the last few years. In some cases the course was taught only to computer science students an in others the two courses were taught together.

4.1.1 Case Study #1: Quidditch (Without Design Students)

In the Spring of 2003 when the class was taught only in the computer science department, one team proposed to implement the game of Quidditch (Figure 2). The basic gameplay followed the rules of the game mentioned in the books. The textures were created from DVD screenshots of the movie. While the overall game worked like Quidditch, there were no wizard-like characters flying on the broom. Instead there were characters from Unreal flying around with guns in their hands. None of the programmers on the team were skilled enough in 3D modeling to create characters with natural flying animations. Unreal rockets represented bludgers and the snitch was a big unreal character. Lack of 3D art assets was a drawback for this game as the environment and characters from unreal took away the immersion from the game. Having both fun gameplay and immersive ambience in the game significantly improves the perception and opinion of players of the game.

4.1.2 Case Study #2: Strong Desire For Quickness (With Design Students)

Strong Desire for Quickness is a racing game in which players can choose to either play a timed race (Figure 3) on a track or play in an open map in combat with other players. The game supports single as well as multiplayer game play. For this game, design students were involved throughout the design and development of the game. Due to the input of design students, programmers could implement a wide variety of weapons that could be used in battle mode as well as during the race. Each weapon was carefully designed and programmed. The maps were very artistic with carefully thought out placement of weapons for balanced game play.

4.1.3 Case Study #3: Bored With Paradise (With Design Students)

Bored with Paradise (Figure 4) is set in a future world where the advances in science have led the humans to build a paradise for themselves where all the resources are feely available to everyone and there is no need for money or economy. In such a time, there is a group of Luddites who, fascinated by the idea of having an economy, are trying to create disruptions in the paradise to create demands for objects that they can sell and thus have an economy. The students working on this game from both classes came

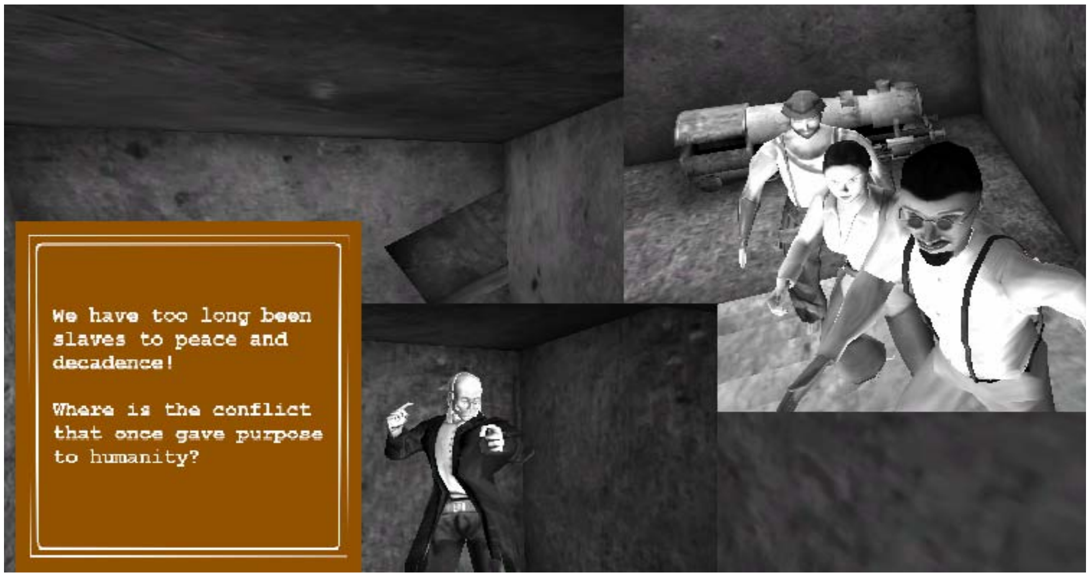

We have too long been slaves to peace and decadence!
Where is the conflict that once gave purpose to humanity?

Figure 4: Bored With Paradise cinematic introduction collage, Spring 2005

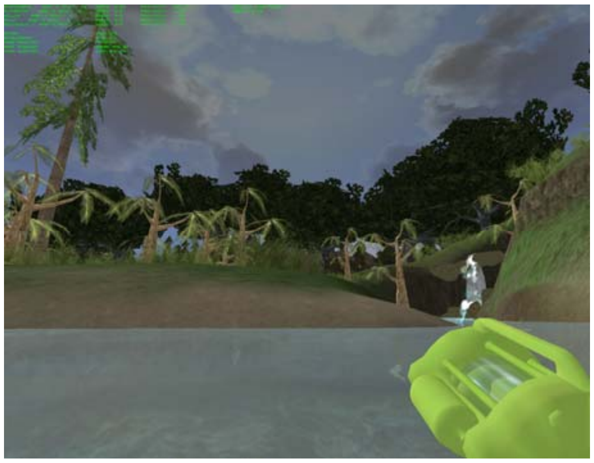


Figure 5: The Dryad Game, Fall 2006

together and created rich environments to support the gameplay. The main advantage to this team was the familiarity and interest of some computer science students on this team in film studies.

4.1.4 Case Study #4: The Dryad Game (Without Design Students)

Dryad is a simulation game with innovative gameplay. The objective of the game is to grow a forest with different types of trees. The player progresses through a series of small levels and learns how different types of trees interact with each other and the environment. Each type of tree provides the player with some powers (like control over a group of bumblebees). In addition to planting trees on different types of soil and beautifying the forest the player also has to protect the forest from natural calamities (like seasons, forest fires, earthquakes), and wood cutters. This is one of the more successful games that only computer science students have created. While the environments are rich and forest sounds are soothing, the students do acknowledge that they had to work much harder in getting some of the tree models and weapons into the game. They also had to throw away some ideas, as they were not able to find 3D models for weapons, trees and other props.

5. OBSERVATIONS AND CONCLUSIONS

Game development is an inherently multi-disciplinary endeavor. It is very difficult for most students to excel in all the relevant disciplines at the same time. A strong computer science background is needed for software development. Not all software developers with strong computer science background have the creativity or appreciation for art. There are many computer science students who are interested and have the right skills in their discipline but do not want to specialize in game development. Our approach to teaching game development is motivated by the need for giving traditional Computer Science students an introduction to the specific skills that are needed in the game industry with respect to software development, and also a chance to experience the inter-disciplinary interaction that is common in the industry.

There are some features of this class that it uniquely contributes to general computer science education. As noted in previous sections, students gain exposure to aspects of game development that are also present in most professional software development contexts: content management, teamwork, development schedules, and marketing. They have an opportunity to take up leadership roles within their teams. Some students choose to develop artistic skills and contribute with the creation of both software and art assets. We have received informal feedback from several students who have passed this course and moved on to traditional software companies that they have found the lessons learned in this class to be useful in their respective jobs. This course serves as a good introductory course for students aspiring to make careers in the games industry and also provides valuable experience to students who are interested in programming careers elsewhere. There is noticeable difference between a game with excellent gameplay and optimal algorithms but no ambience and a complete game with the same gameplay and algorithms with pleasing environments and ambience. Similarly, a fully animated character model is much more appealing if seen in an interactive setting involved in gameplay. Both components are essential for developing original and creative games. With this class, both computer science and design students benefit from the symbiotic relationship between the two disciplines.

In summary, we found that co-teaching the game design class was a positive experience. Involving design students improves the overall quality of games produced as well as provides the students the opportunity to come up with creative ideas without worrying about availability of artistic content. Having teams of 5 to 7 students from different backgrounds introduced the overheads of communication and coordination but it was a valuable experience for students. They had to deal with a lot of project management issues like team members dropping courses in the middle of the semester and reassignment of tasks. Finally, this is the kind of course where students are more worried about their project deadlines not because of their grade but because they have to show a working game in a big showcase at the end of the semester. Having a fully working game of a reasonably large scale is very satisfying to most students and is what attracts them to this course.

6. ACKNOWLEDGMENTS

We would like to acknowledge the support given by the Department of Computer Science and the College of Design. We also acknowledge the hard work that students put in developing the cool games in this class.

7. REFERENCES

[1] Busby, J. 2004. Mastering Unreal Technology: The Art of Level Design, *Sams Publishing.*

[2] Busby, J. 3D Buzz Video Tutorials, http://www.3Dbuzz.com

[3] Epic Games Inc. 2004 Unreal Tournament 2004.

[4] North Carolina State University, Department of Computer Science, CSC481: Computer Game Design and Development, http://liquidnarrative.csc.ncsu.edu/classes/csc481.

[5] North Carolina State University, College of Design, ID400, http://courses.ncsu.edu/ID400.

[6] Rabin, S. 2005 Introduction to Game Development. *Charles River Media, Inc.*

[7] University of Southern California, That Cloud Game, http://www.thatcloudgame.com

ARTS Lab and Game Technology

Edward Angel
University of New Mexico
Departments of Computer Science, Electrical and Computer Engineering, and Media Arts
1-505-277-2186
angel@cs.unm.edu

Thomas P. Caudell
University of New Mexico
Departments of Electrical and Computer Engineering, and Computer Science
1-505-277-5637
tpc@ece.unm.edu

Eric Whitmore
University of New Mexico
ARTS Lab and Art Technology Center
1-505-277-2253
whitmore@unm.edu

ABSTRACT

The Art, Research, Technology, and Science Laboratory (ARTS Lab) at the University of New Mexico was created as the key research and educational entity in the State's Media Industry Strategic Plan. In this paper, we will describe our efforts in creating a truly interdisciplinary course for seniors and graduate students that serves the diverse needs and interests of students, faculty, and local industry. We will also show some of the directions that the projects have taken using the special facilities provided by the ARTS Lab.

Categories and Subject Descriptors

I 3 [**Computer Graphics**] I 6 [**Simulation and Modeling**] K 3 [[**Computer and Education**] J 5 [**Arts and Humanities**]:

Keywords

Game technology, game design, computer science education.

1. INTRODUCTION

In 2004, the State of New Mexico embarked on an ambitious Media Industries Strategic Project (MISP). This project was a natural successor to the State's successful film incentive program, which has attracted national attention and a large number of film projects to the state. However, these incentives, which are a combination of rebates and direct investment, were recognized as being a short-term strategy and that a sustainable industry should be based on a more general media including game technologies and content development.

The MISP plan led to the creation of the Art, Research, Technology, and Science Laboratory (ARTS Lab) at the University of New Mexico as the key research and teaching entity focused on high technology media. A key component of ARTS Lab is extensive cooperation with the local National Laboratories (Sandia and Los Alamos) and with a growing local media industry.

As in virtually all universities and colleges, the students have an enormous interest in computer games, whether as users, potential content developers, or technologies. Locally in New Mexico, there is a particular interest in both "serious games," that come from the states long involvement with projects in simulation, visualization, and virtual reality, and games that involve the state's diverse artistic and cultural resources

Like many institutions of higher education, we are struggling with how the interest in games translates into our curricula and how to deal with the interdisciplinary nature of games and game technology.

In this paper we shall discuss a two semester class that we instituted at the University of New Mexico that involved interdisciplinary teams of students principally from the School of Engineering and the College of Fine Arts but also included members of the community and students from other colleges on campus.

However, we start with a description of our the ARTS Lab and its unique interdisciplinary nature which colors the way our program is evolving and hopefully provides a model for other universities and colleges, especially those with limited resources and diverse student bodies.

2. The ARTS Lab

ARTS Lab was created in response to New Mexico Governor Bill Richardson's Media Industries Strategic Plan (MISP). ARTS Lab seeks to support innovation and growth in areas such as film, new media, simulation, telehealth, game technology, image processing, scientific visualization, national security applications, and new markets for content. As a center for both technology and the arts, New Mexico provides a dynamic environment for programs that stimulate economic development. ARTS Lab finds opportunities to cultivate these assets by utilizing an interdisciplinary approach, encouraging ongoing participation across the University of New Mexico campus as well as building on ties with industry, community, and other educational institutions [1].

The ARTS Lab was seeded with a $3M grant from the State of New Mexico. This funding was used to create the ARTS Lab "Garage," so named because it is located in what was the garage area of a car dealership. The building is shared with UNM's Center for High Performance Computing which opens many new projects and interactions.

The ARTS Lab (Figure 1) space includes a reconfigurable black box studio with a control booth, sound booth, computer controlled lighting, a large green screen area, and a Vicon motion capture system. The adjacent machine room houses both the computers managed by the Center for High Performance Computing, the ARTS Lab render farm and 20TB of disk storage for our projects. In addition to a variety of workstations and media oriented computers, the Garage houses a unique six projector dome that will be discussed in Section 4 (Figure 2).

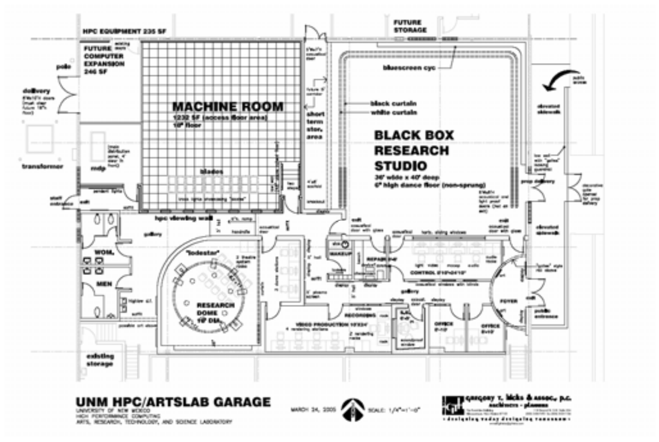


Figure 1: ARTS Lab Garage

The center also houses the Visualization Lab for the Center for High Performance Computing which can support projects in areas such as virtual reality.

The facility is run in a manner to encourage new projects involving all parts of the campus, local industry and the national laboratories in New Mexico.

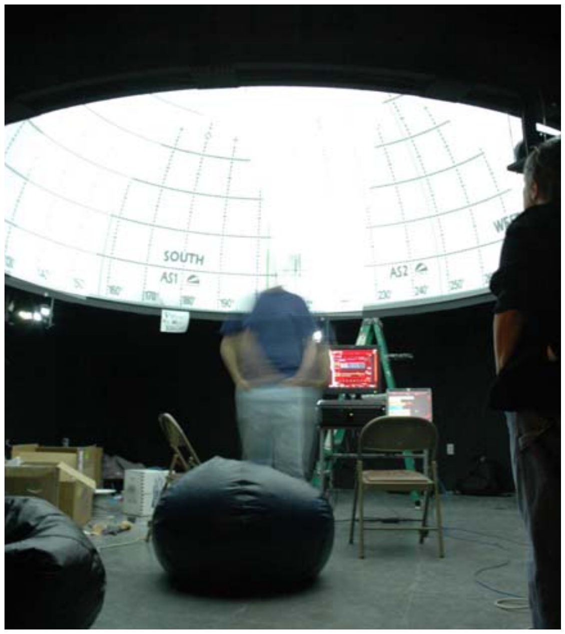


Figure 2: ARTS Lab Experimental Dome

3. COURSE DESIGN

We faced a number of challenges in creating a game course, not the least of which was the senior faculty instructors who were not brought up with computer games as a part of their environment. We wanted to create a course that could be part of senior and graduate programs in multiple departments, a requirement that implies both maturity on the part of the students and the importance of at least three years in a degree programs. Consequently, we chose a mode that would use interdisciplinary teams working in a project environment. In addition, because the mission of the ARTS Lab includes an economic development component, we wanted to create a course in which not only would there be participation by local companies working in the area but also a course that would help students learn more about the real industry from multiple perspectives.

3.1 First Semester

We decided to keep the first semester as a one credit seminar course in which we would spend the most of the semester bringing in a variety of speakers from industry, academia, and the national laboratories. The last part of the semester was spent on student presentations of proposals for game projects. The only requirement for credit was submission of a proposal for a game that would be a candidate for a team project the following semester.

Speakers included the author of a book on digital storytelling, a game company executive with extensive experience with massive multiplayer games in Asia, two creators for game development and animation studios in Canada and Asia, three developers of specialized software and haptic devices for the game industry, a writer with extensive experience writing interactive games, and two developers working on large scale simulations using game technology. The lectures were so popular that we often drew a large audience of people not registered for the course.

One of the most valuable outcomes of the course was the deeper understanding of the game development process the students obtained. The older Hollywood refrain of "I've got a script" has been replaced with "I've got an idea for a game." Most of the students entered the class with the hope that their ideas would instantly be recognized by the industry which would then develop their idea. When the students first discovered how the industry works and where the entry points are they were somewhat depressed and discouraged. But as the semester went on and students discovered the many facets of the game industry and opportunities that they had not known of, their attitudes became far more positive.

3.2 Second Semester

The second semester was run as a project course, much as handle our senior software engineering course or our capstone design classes. The first part of the semester was spent going over individual proposals which coalesced into team proposals. All of the final project teams were interdisciplinary.

The rest of the semester consisted of technical presentation by the instructors and others to support the projects, progress reports, and finally the project presentations. Remarkably perhaps, all the teams were able to get their project to the point where they had a working game.

We left the choice of a game engine to the teams. One or two teams did projects in which they wrote their own software. A couple of teams used the Game Studio [2] engine which is fairly simple and easy to get started with. Other teams used the Torque engine [3] which is more complex but still relatively inexpensive to obtain. One team was able to use the Half-Life engine [4].

Surprisingly, at least to the instructors, only one of the projects involved a traditional first person shooter game. We attribute a lot of this to the major interest in "serious games" by the local community and to the fact that the majority of the students were seniors and graduate students with considerable maturity in their chosen disciplines.

Some of the projects included

- **Getting through UNM**: In this game players have to negotiate their way through the University system. Obstacles and characters included campus parking police, the university mascot, and indecisive students in the Starbucks line.
- **Guerilla Game**: Players start of off as poor students and must acquire sufficient resources to be come revolutionaries.
- **Quick Racing**: A racing game (see Figure 2) using a haptic device provided by the local company Novint as input.
- **Dome Pong**: A version of the classic pong game played in the ARTS Lab dome.
- **The Language Game**: A learning game involving identifying objects in multiple languages.
- **Sonorous Phase**: A music composition and understanding game designed about a Mandela.

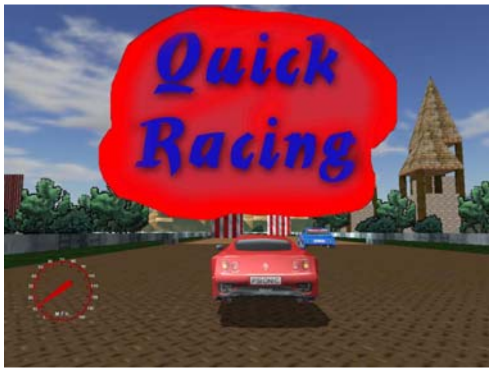


Figure 3: Interface to Haptic Interface Game

4. DOME GAMES

A unique feature of the ARTS Lab Garage is its 15' diameter multi-projector dome (or fulldome). We have been involved with creating dome content for over 5 years in a partnership with the LodeStar Astronomy Center (L*), a joint project of the University of New Mexico and the New Mexico Museum of Natural History and Science. L* was one of the first of a new generation of fully digital planetariums---there are now about 250 worldwide---and has been a leader in generating new applications of the technology. Although almost all fulldome shows are done in a playback mode using large disk arrays for each projector, both the L* and ARTS Lab systems use Sky-Skan's Digital Sky system which uses commodity graphics cards and thus is capable of realtime performance

Consequently, the ARTS Lab has been interested in exploring interactive applications of fulldomes with games being the most obvious one but there are endless other possibilities ranging from simulations to scientific visualization. Sky-Skan provided us with a method of connecting to their Digital Sky software through a DirectX plug-in.

Because fulldomes are fully digital, unlike other large display venues such as IMAX, and can hold up to 500 people, far more than immersive environments such as the Cave, the potential for interaction among large groups of participants is virtually unlimited. As shown by Loren Carpenter's use of simple two color paddles for large group interaction at SIGGRAPH [5], sophisticated input devices are not necessary (although we are investigating the use of wireless devices and high bandwidth networks for interaction among dome).

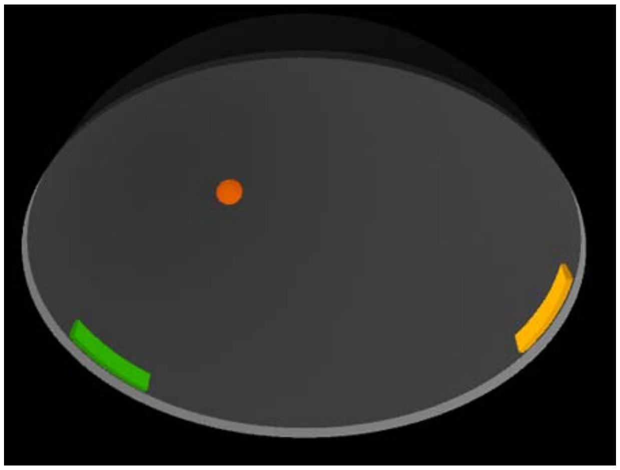


Figure 4: Dome Pong

One team led by a graduate student Jin Xiong did a version of the classic Pong game (Figure 4) for the dome. The paddles are controlled by two players and can move continuously around the rim of the dome. The ball is reflected from the paddles and is subject to gravity. Visually the game is somewhat like using the paddles to roll the ball over the surface of the hemisphere. The game was developed on the ARTS Lab dome and was later demonstrated at the larger LodeStar dome to a large audience.

5. DISCUSSION AND CONCLUSIONS

From the students perspective the courses were a great success. Even with the great variety of backgrounds and previous experience, every team was able to produce a project that (a) was reasonably complete (b) was original and (c) made use of the different backgrounds of the group members. All the members of the class felt that they had learned a considerable amount from the class. Participation by a variety of outside speakers provided invaluable insights to the class. At least three members of the class who received their BS or MS degrees that semester were hired by game companies.

As we prepare to do the class again in the spring semester of 2007, we will be making a few changes. First, we are combining the two semesters into one standard 3 credit course. Although a one credit seminar or colloquium on games is a worthwhile endeavor, the practical issues of scheduling and fitting both classes into student programs presents many practical difficulties.

This time around we plan to have all groups use the same game engine. Choosing the “right” game engine is a major problem in planning a course. Very sophisticated engines have a steep learning curve and their use in an interdisciplinary first course poses the danger of having the course reduce to a training course for that particular engine. In addition the cost either to the student or the University can be a problem. On the other hand, using a public domain or low cost engine, as we provided the first time, makes it easier to get started but its limitations can frustrate the teams as they get into development of their projects. Consequently, for the next class offering, we are planning to use the Torque engine which is somewhere in the middle and for which there is a large amount of supporting code and literature including possible textbooks.

The primary conclusion we draw from our recent experiences with this type of class is that teaching an interdisciplinary computer game class is completely consistent with academic programs in engineering and the arts.

6. ACKNOWLEDGMENTS

We wish to thank the students who were willing to take the risk of taking a new class with instructors who started off with virtually no experience with games. The media staff of the ARTS Lab, Hue Walker and Enrico Trujillo, were both participants and providers of help to the student projects. Finally, we wish to thank the many guest speakers from industry who gave their time, their advice, and often loans of equipment to the student teams.

7. REFERENCES

[1] http://artslab.unm.edu

[2] http://www.3dgamestudio.com

[3] http://www.garagegames.com

[4] http://www.gamespot.com

[5] http://www.cinematrix.com

Interweaving Game Design into Core CS Curriculum

Yolanda Rankin
Northwestern University
2133 Sheridan Road
Evanston, IL 60208
847-467-5635
yrankin@northwestern.edu

Bruce Gooch
Northwestern University
2133 Sheridan Road
Evanston, IL 60208
847-491-3500
bgooch@cs.northwestern.edu

Amy Gooch
University of Victoria
CS Department & ECS Bldg 504
PO Box 3055, STN CSC
Victoria, BC Canada V8W 3P6
Amy.a.gooch@gmail.com

Abstract
Computer Science departments across the country have embraced computer gaming classes as part of the core curriculum. However, instructors need to define guidelines that accommodate students' proficiency in game development and consequently address the growing needs of industry. We develop and evaluate a Game Authoring Class in an attempt to begin to close the gap between industry and academia. Learning objectives include students applying game development concepts to implementation of 2D and 3D games for multiple platforms. Subsequently, we evaluate the course based on two factors: formal assessment of students' understanding of game design principles and students' perceived learning. The results of our evaluation serve as the basis for establishing effective pedagogical strategies for game development curriculum.

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

Keywords
Computer science curriculum, game development, evaluation, educational assessment, pedagogy.

1. Introduction

Currently, enrollment in Computer Science programs across the nation has dropped 60% and the trend shows a continued decrease over the next few years [16][17]. While industry faces the impact of fewer computer science majors, academia has employed several different approaches to address this critical situation. In an attempt to combat the declining interest in computer science, computer science departments are leveraging the appeal of video games to entice the next generation of computer science majors. Thus, game development courses are quickly becoming a part of the Computer Science curriculum at universities and colleges across the country.

However, changes in the curriculum do not necessarily equate to students developing the necessary software skills and expertise appreciated by the game industry. Oftentimes discrepancies exist between the learning goals presented in a classroom setting and the skills and experience that are specific to the needs of the gaming industry [1][2][3][7]. It is not sufficient to simply write code that creates a game if students have failed to grasp design concepts that are crucial for game development.

In response to this dilemma, we design a Game Authoring Class and teach the course at Northwestern University during Spring Quarter 2006. We propose modeling game development courses as closely to the software development process followed in game industry. The course emphasizes research in the field of computer games and hands-on experience developing of games. Students review current trends in computer game programming and build their own 2D and 3D games on top of available game engines[7][10][15]. Additionally, this class encompasses building games for multiple devices, such as cell phones, PDAs, Pocket PCs, desktops and laptops. Adopting industry game design principles, we require students to implement these design principles in three project deliverables: 1) game design document and play-test criteria, 2) 2D game for a mobile device, and 3) 3D game module that runs on an existing game engine. Finally, we review the Game Authoring class and establish best practices for teaching any game development class that prepares students for careers in the gaming industry.

2. Apprenticeship Game Design

Research shows that video games are an often under-utilized learning environment that can be extended to various domains, including mathematics, physics, history, and language learning [6][8][9][14][15][16]. We delve into yet another domain, that of computer science, maximizing the attraction of games to revamp traditional computer science curriculum and train the next generation of software developers. The goal is to close the widening gap between industry and academia by modeling industry practices; before we can accomplish this goal, we must incorporate pedagogical strategies into the design of game development courses.

Cognitive scientists understand that learning occurs in the context of meaningful tasks [4][5][8][11]. In actuality, these tasks represent regular practices of a particular community or group or people. Rather than introducing theoretical concepts separate from real-world applications, effective teachers guide students through situated activities that demand the application of information or a specific skill [4][5][11]. As the student participates in activities under the guidance of the teacher, the student develops the skill set required to complete the task. Cognitive scientists refer to this as the apprenticeship model [5].

We extend the model of cognitive apprenticeship to the design and instruction of the Game Authoring Course. We carefully select assignments (e.g. writing a game design document, conducting play-tests of game prototype, etc.) that reflect authentic game design practices in the gaming industry. As students acquire knowledge necessary to complete the assignments, they develop proficiency using the tools of the trade. Under the careful guidance of the instructor, class discussions create a safe environment in which learners feel that they contribute to a community of learners while gaining expertise in a particular activity or task. Game play demonstrations give students the opportunity to assess their peers' work and to refine their ideas about what it means to be a game designer.

Interaction with students enrolled in the course provides secondary means for support and self-reflection as students develop proficient skills as game designers. Additionally, the assignments instantiate tasks that game developers do on a regular basis. Hence, the project deliverables represent opportunities for situated learning as students play the role of game developers throughout the Game Authoring Course. We now turn our attention to the learning objectives for each project and assess students' mastery of game design principles.

3. Game Authoring Course

We taught the Game Authoring class during the Spring Quarter 2006. Fourteen students, thirteen male and one female, were enrolled in the class. Fifty percent of the students were computer science majors. The class was scheduled for 10 consecutive weeks with the class meeting twice a week for lectures during the first seven weeks and ending with three weeks of laboratory meetings. The course required students to develop at a minimum level of basic proficiency for standard software development tools such as Microsoft Visual Studio, Microsoft DirectX, C++, OpenGL, and Python. One graduate and three undergraduate students were designated as teacher assistants (TAs) for the course. The TAs created a class wiki (http://cs.northwestern.edu/~gaming/), which facilitated communication between students and the instructor/TAs and served as a digital notebook of students' progress on assignments. Students were responsible for three projects: 1) creating a web-based game design document and defining play-test criteria for evaluating games, 2) developing a 2D game for a mobile device, and 3) developing a 3D game module that runs on top of the Half-Life2 game engine.

Project 1: Game Design Document

The first assignment set the stage for students to think deeply about the structure of games (i.e. players, objectives, rules, procedures, conflict and possible outcomes) and the dramatic elements (story, challenge, sense of fun, etc.) that create memorable experiences for gamers [7][12][16]. To assist students with writing a game design document, students first wrote a review comparing and contrasting two games of their choice. Students evaluated the games according to the following criteria:

- What is the appropriate audience for this game?
- What is fun about the game and why?
- What is bad/not fun about the game and why?
- How does it compare to similar games in the genre?
- Why is it better or worse than similar games?
- What are the highlights/low points of the game play?

Class participants posted game reviews on the wiki. One student added additional criteria, asking the following questions: "Does the game's artwork portray a realistic world or does it use a more abstract artistic representation? Is the artistic style reminiscent of another game's artwork or even art style in a different medium (such as a style of painting or architecture)? Does the game's artwork fit the mood of the game's story, environment, and game play?"

These questions indicate an understanding of how every detail adds to the immersive environment of games; one poorly conceived design detail could destroy this carefully crafted virtual world. As a result, each student identified play-test criteria used to evaluate the 2D mobile games developed by their peers. Thus, the assignment functioned as the initial point for students' forming a foundation that enables them to incorporate game design principles without sacrificing the element of fun.

Project 2: Developing a 2D Mobile Game

The second assignment emphasized the iterative process of game design and the importance of involving users in the early stages of game development [7]. Initially, students were required to use the Microsoft DirectX software development kit to design the 2D mobile game. However, the three undergraduates TAs and only one student enrolled in the class were able to get the DirectX SDK downloaded, installed and running. The remaining 11 students in the course experienced difficulty installing and executing the DirectX SDK and requested to use other software development tools to complete the assignment. Adequate support for use of software development tools is necessary if we expect academia to prepare students for employment in the game industry. Therefore, continuous communication between industry and academia helps to ensure that the computer science curriculum reflects the needs of industry. The remaining students relied on additional software development tools (PyGames, Flash 5.0, JavaScript, etc.) to implement working prototypes. While this was an unexpected outcome, it actually gave students experience developing 2D games for various platforms, a transferable skill to industry.

Students followed the iterative design process, generating storyboards, formalizing formal and dramatic elements, implementing a working prototype and conducting play testing [7][16]. Play-testing occurred on multiple levels, including self-test testing, testing with friends, and testing with strangers [7]. Self-testing ensured that the game demonstrated limited functionality (e.g. ability for players to navigate game controls) before release of the working prototype. In contrast, peer testers used the RITE method to provide immediate evaluation and recommended changes that would enhance the game play experience [13]. Students made changes accordingly and demonstrated the final version of the game during class time.

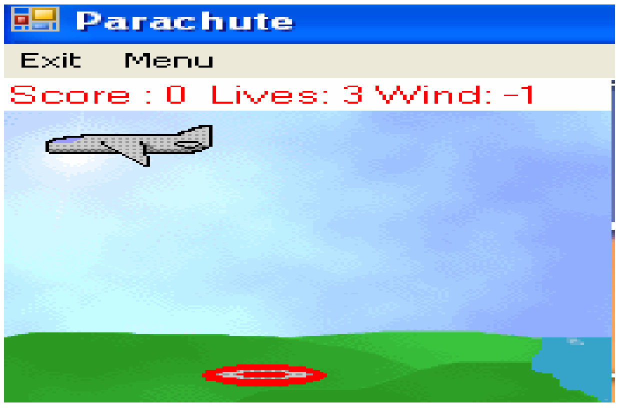


Figure 1. Parachute 2D Mobile Game.

Project 3: 3D Game Module

The third project gave students the opportunity to gain hands-on experience designing a 3D game module on top of the Half-Life2 game engine. The three undergraduate TA's had developed a game module entitled "Project Echo" in a previous undergraduate course. Project Echo is a first-person shooter that features one game level, two weapons (machine gun and leech launcher), sound effects for weapons and limited AI for non-playing characters (NPCs). Due to time constraints, students were tasked with software modifications that would increase game functionality and improve the game play experience. Before any software changes were made, students play-tested Project Echo and identified the strengths and weakness of the game. Students were encouraged to work in groups to foster teamwork typical of the game industry which involves a team of developers, including graphics artists, animation artists, user interface programmers, sound engineers, etc. Students pitched the high-level concepts of their game modifications, outlining a project schedule for software development. All files and executables, documentation, and proposals were posted to the class wiki.

Game modifications covered a wide range of additions: students created new models for weapons; rigged and skinned a model of a monster in Maya; designed an additional game level using Half-Life2's Hammer tool; added customized sound effects that were scripted to promote emotional involvement during game play; implemented a more sophisticated AI to allow NPCs to harvest local resources; and revised the game interface to give players current status information.

4. Best Practices

Each project afforded us the opportunity to identify what worked and what failed as it related to our learning objectives. In addition, 64% of the class completed Northwestern University Course and Teacher Evaluation Councils (CTECs), which gave us insight into students' expectations and opinions of the class. Based on student evaluations and lessons learned assessment, we identified effective practices for game development curriculum.

4.1 Pipeline between industry and academia

If there were one thing that we could do differently, it would be to engage companies in dialogue with our students as part of the course. One participant suggested that the course include guest speakers; this would have enabled students to establish connections with industry. Additionally, students expressed frustration with the inability to get DirectX up and running for the 2D mobile game project. One student reported, "We were supposed to use certain software to program a game but apparently the TA's never properly prepared the computers for this so we resorted to a different program." Unfortunately, this gave the student the impression that 2D mobile project was not well prepared. Oftentimes instructors as well as students experience a learning curve before becoming proficient in the use of industry tools. This feedback led us to consider direct access to Microsoft DirectX personnel as we modify the syllabus for future class objectives and assignments. By establishing a direct pipeline between industry and academia, we can attain real-time customer support, shorten the learning curve and assist faculty and students with developing proficiency of software development tools. This becomes a crucial factor for hiring qualified employees that can immediately contribute to company revenue.

4.2 Incorporating Industry practices

Industry has criticized academia for failing to embrace the latest technological practices and advancements supported by industry. In an attempt to address this issue, we designed a game development course that closely followed industry practices and methodology for game development. It is common practice for game designers to first create a paper prototype of a game as a means for testing their ideas [7][17]. To emulate this practice, students presented a paper prototype of their 2D game to their classmates before any software development. As a result, classmates helped students refine game mechanics, suggesting ways to increase the level of difficulty and identifying potential problems that might impede enjoyment of the game.

The Game Authoring class required students to embrace the iterative game design paradigm for both the 2D mobile game project and the 3D game module. Students created storyboards, identified the structure of the game, and implemented a working prototype [7]. As students communicate their ideas to their classmates, these same classmates would comment on the ideas, helping the author to conceptualize the formal and dramatic elements that support a memorable game play experience. Play-testing on multiple levels ensures early feedback and prevents costly mistakes that produce poorly designed games.

4.3 Teamwork amongst students

T he 3D game module included a written assignment regarding the "lessons learned" over the course of the last project, forcing students to compare their initial design proposal with the actual work completed. Students frequently admitted that they did not accomplish all of the goals. This failure was attributed to two factors. First, students underestimated the amount of work required to develop a game module. For example, before one can make any software changes, one must first sift through several lines of code to determine the logical flow. Once the student traces the logical flow, the student must master the software development tools (e.g. XSI for creating models of weapons) to implement code changes. Secondly, 71% of the class chose to work solo on the 3D game module project. Experienced game developers know that it takes a team approximately 2 – 3 years to design a game that may or may not reach the consumer market. This team of people is comprised of graphic artists, voice actors/actresses, sound engineers, music editors, user interface programmers, level designers, etc. who meet constantly to review the development schedule and work together to produce one final product [7]. Thus, our students failed to appreciate the benefits of teamwork. As a result, the amount of time students spent working on the game module did not correlate to high quality product created by students. One student suggested that we coordinate teams of students to design the game module. By allowing the majority of students to work alone, we missed a great opportunity to teach students that teamwork is a requisite for the gaming industry and that teamwork results in greater productivity. To better address this issue, we suggest that the assignment should be subdivided into individual assignments that form the whole product. Thus, no two students will have the same assignment and yet each assignment produces a component that is crucial to the construction of the final product.

Figure 2. Game level in for 3D Game Mod.

5. Course Assessment

64% of students completed a single blind survey that assessed students' perception of the class. In summary, students ranked the overall instruction 4.67 on a scale of 1 to 5 with 5 being the highest and 1 being the lowest. Scrutiny revealed that 30% of the survey participants thought the overall course was excellent while approximately 44% indicated that the class was a good or satisfactory course. We contribute the students' high rank of instruction to the instructor's ability to communicate his enthusiasm for game design and ability to establish rapport with the students. This led us to question whether students were learning anything in the class. More than 40% of survey participants expressed that they had learned a substantial amount about the game development process; 22.22% believed they had attained some knowledge. Only, 11.11% of the students indicated they had minimally increased their knowledge because of taking the course.

The fact that less than half of the class felt that they had learned a substantial amount raised a red flag, perhaps some students were not learning due to the open-ended structure of laboratory exercises. During the last three weeks of class, students attended programming laboratory to attain assistance with debugging code and develop proficiency with specific software packages such as XSI Modeling toolkit in the absence of course lectures. Students commented that reviewing the code for the Half-Life2 game engine was a lengthy process that proved to be frustrating at times. For those students who lacked patience and persistence to do work outside of the designated course time and laboratory meetings, these same students did not believe that they had accomplished the learning objectives of the class. Furthermore, students rated the intellectual stimulation provided by the class to be an average of 4.89 on a scale of 1 to 5, suggesting that the course challenged students to think about new concepts in a creative manner. 78% of the class spent a minimum of four additional hours outside of class and lab time. We posit that the number of hours spent outside of class and laboratory meetings played a critical role in assisting students with understanding game design and ultimately writing code that produced a working 3D game module. The time spent on time also suggests the need for in-depth tutorials that assist students to shortening the learning curve.

Table 1. Student Rankings of the Game Course

Course Criteria	**Class Average: Rank on Scale 1 (lowest) to 5 (highest)**
Provide an overall rating of the instruction.	4.67
Rate the effectiveness of the instructor in stimulating your interest in the subject.	5.22
Estimate how much you learned in the course.	4.67
Rate the effectiveness of the course in challenging you intellectually.	4.89
Provide an overall rating of the course.	5.11

In summary, students highly ranked the course to be 5.11 on a scale of 1 to 5 with five being the highest and 1 being the lowest. This is extremely high in comparison to traditional computer science courses taught at Northwestern University, which average a rating of 3.

4. Conclusion and Acknowledgments

We offered the Game Authoring course as part of the computer science curriculum at Northwestern University. We purposefully identified effective practices that support the implementation of industry practices in the classroom setting. Furthermore, we establish these practices based student evaluations of the mistakes we made and the things that worked well. These observations serve as the starting point for revising computer science curriculum as we prepare the next generation of computer programmers. We would like to thank Microsoft Research for providing the necessary funding and equipment for making this research possible. Additionally, we express our appreciation to the National Science Foundation for their support of this research.

References

[1] Adams, E. Bad Game Designer, No Twinkie! *Gamasutra*. (March 13, 1998).

[2] Adams, E. How to Get Started in the Game Industry Part 1. *Gamasutra* (December 11, 1998).

[3] Adams, E. How to Get Started in the Game Industry Part 2. *Gamasutra* (December 18, 1998).

[4] Brown, J.S., Collins, A., and Duguid, P. Situated Cognition and the Culture of Learning. *Educational Researcher,* (18)1, 1989, 32-41.

[5] Collins, A., Brown, J.S., and Newman, S.E. Cognitive Apprenticeship: Teaching the Crafts of Reading, Writing and Mathematics. In L.B. Resnick (Ed*.), Knowing, learning, and instruction: Essays in honor of Robert Glaser*, Erlbaum, Hillsdale, NJ 1990, 453-494.

[6] Elliot, J., Adams, L., and Bruckman, A. No Magic Bullet: 3D Video Games in Education. *In Proceedings of the International Conference of Learning Sciences* (ICLS 2002) Seattle, Washington, 2002.

[7] Fullerton, F., Swain, C., Hoffman, S. *Game Design Workshop: Designing, Prototyping, and Playtesting Games.* CMB Books, San Francisco, CA, 2004.

[8] Gee, J. *Situated Language and Learning: A Critique of Traditional Schooling*. Routledge, 2004.

[9] Gee, J. *What Video Games Have to Teach Us about Learning and Literacy*, Palgrave Macmillan, New York, NY, 2003.

[10] Koster, R. *A Theory of Fun for Game Design.* Paraglyph Press, Scottsdale, AZ, 2005.

[11] Lave, J. and Wenger, E. Situated Learning: Legitimate peripheral participation. Cambridge University Press, Boston, MA, 1991.

[12] Malone, T. W. What Makes Things Fun to Learn? Heuristics for Designing Instructional Computer Games. ACM (1980).

[13] Medlock, M. C., Wixon D., Terrano, M., Romero R., Fulton B. (2002). *Using the RITE Method to improve products: a definition and a case study*. *Usability Professionals Association, Orlando FL* July 2002

[14] Prensky, M. *Digital Game-Based Learning*. Donnelley and Sons Company, Chicago, IL, 2001.

[15] Rankin, Y., Gold, R., and Gooch, B. 3D Role-playing Games as Language Learning Tools. In *Conference Proceedings of EUROGRAPHICS Education Program 2006*. Vol. 25. Vienna, Austria, 2006.

[16] Rankin, Y., Gold, R., and Gooch, B. Evaluating Interactive Gaming as a Language Learning Tool. *In Conference Proceedings SIGGRAPH Educators Program*, Boston, MA, 2006.

[17] Salen, K. and Zimmerman, E. *Rules of Play: Fundamentals of Game Design.* MIT Press, Boston, MA, 2003.

[18] Snyder, N. *Universities See a Sharp Drop in Computer Science Majors*, Tennessean.com September 2006.

[19] Vesgo, J. Interest in CS as a Major Drops Among Incoming Freshman, Computing Research News, May 2005 Vol. 17/No. 3. http://www.cra.org/CRN/articles/may05/vesgo

Integrating Video Game Development Experience in an Academic Framework

Bernard Yee
Harmonix Music/MTV Networks
160 Cabrini Blvd, Apt 84
New York, NY 10033
+1 (212) 724-7564
bhy@bernieyee.com

David Sturman
Massive Inc. / Microsoft Corp.
627 Broadway, 7th Floor
New York, NY 10012
+1 (646) 778-3500
david.sturman@microsoft.com

Steven Feiner
Columbia University
Department of Computer Science
New York, NY 10027
+1 (212) 939-7083
feiner@cs.columbia.edu

ABSTRACT

Over the past three and a half years, courses covering game design, technology, and production have been offered in the Department of Computer Science at Columbia University. These courses were originally taught by two adjunct professors specializing in technology (David Sturman) and design (Bernard Yee) and have been among the most highly subscribed courses in the department taught by adjunct faculty. The pedagogical approaches taken have been modified nearly every time the courses have been offered, due both to a desire to vary and improve course content and style and to the constraints imposed by the professional commitments of the teachers. In this paper, we describe the assumptions underlying the courses and the different ways in which they have been taught, and discuss some of the issues raised by our experience.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer science education, curriculum*; K.8.0 [**Personal Computing**]: General—*Games*

General Terms

Design, Human Factors

Keywords

Video games, game design, game technology, game production, game curriculum development

1. INTRODUCTION

Given the tremendous cultural and commercial impact of video games many Computer Science students have a great interest in video games and aspire to join the industry. At the same time, because of the rapid growth and increasing sophistication of the game industry, there is a dearth of qualified game designers and developers, especially cross-disciplinary developers who are skilled at both game design and programming.

However, universities often lack the curriculum and expertise in video game technology and design needed to train students. This paper describes our effort over the past three and a half years to address this need by creating a curriculum in game design and development within the Columbia University Department of Computer Science that bridges the gap between academia and the game industry.

For review only. Please do not distribute.

Submitted to *2nd Annual Microsoft Academic Days Conference on Game Development.*

This effort started with Professor Steven Feiner inviting David Sturman to teach a course in video game technology. To help lighten the load of teaching a full course for which there were no prepared materials, and no suitable texts, Sturman teamed up with production executive Bernard Yee. Both veterans of the video game industry, Sturman and Yee designed a course to provide students with an introduction to video game design and development, with the intent to expose students as much as possible to life in the video game industry.

The course was based on three key tenets. First, students needed a foundation in both technology and design, since good designers must understand technology and good technologists must understand design. Second, students should learn to work in groups, as modern games are always collaborative. Third, game development involves not only good design, good technology, and teamwork, but also commercial viability.

For two semesters (the first iteration), Sturman and Yee taught a single class with alternating lectures in design and technology, short homework exercises, and a single term-long team project that involved specifying, designing, and prototyping an original video game. During a third semester, they split the curriculum into two classes, one specializing in technology and the other in design, with teams combined from the two classes developing term-long video game projects. While other professional obligations have prevented Sturman from teaching after that third semester, Yee has continued to teach a series of courses on various aspects of design.

In the remainder of this paper, we provide an overview of the general structure of the courses and the approaches that we have taken through the years in which they have been taught. We then conclude with a discussion of some of the issues raised by our experience that we feel need to be addressed.

2. APPROACHES

The faculty started with two key assumptions. First, they agreed that game technology and game design were so fundamentally interrelated as to be inseparable. Second, they agreed that because the ultimate goal of video games is to understand and manipulate consumer psychology, game software development had very different goals and processes than other forms of software development.

We also felt it important that the students had an industry context for video game design and technology. Video games are an industry, with political, financial, geographic, and practical issues coloring almost every aspect of design and technology. An appreciation of these issues makes one a much better contributor and more valuable to the industry as a whole. So, woven into the curriculum were lectures on the history of the industry, financial structure of the industry, guest lectures from industry developers and designers, and exercises mirroring important industry practices, such as design reviews.

2.1. Technology Track

For the technology side of the course, our intent was to expose the students to the basic concepts and technologies used in video game development. However, in our first iteration, we did not have time to develop a true programming course with base code, code exercises, and problem sets. Thus, the technology side consisted of lectures on the basic areas of game design, including engine architecture, modeling, animation, physics, AI, rendering, audio, and special effects. While each of these topics could have been the subject of a full semester course, we had only 14 weeks. Therefore, we gave just a broad overview of each topic, so that students would be familiar with the issues and know where to delve more deeply if they were interested.

Finding an appropriate text was difficult. Existing texts were either too tightly targeted to one aspect of video game technology, or were step-by-step how-to recipes for creating a certain type of game. Given the fast-moving and nascent quality of video game technology, we resorted to original materials created by the instructors and online articles, particularly Simpson's *Game Engine Anatomy 101* [12] and Salvator's *ExtremeTech 3D Pipeline Tutorial* [11].

For some topics, we brought in industry developers such as Paul Bettner and Thatcher Ullrich to give guest lectures. This helped to expose the class to the industry, as well as provide direct expertise for the areas we were lacking. In the first two semesters, we had guest lectures covering networking, AI, and developing for platforms such as the Nintendo Game Boy. On the design side, guest lectures focused on level design, emergent gameplay, and interactive narrative.

2.2. Design Track

The structure of the game design track had to be created without any real precedent. We referred to some industry efforts to create a game development curriculum [5], but found them only partially suitable to a single course offering. Several approaches were tried, including the use of Salen and Zimmerman's *Rules of Play* [9] and other game design textbooks. We settled on an approach that adopted the interactivity of a law school classroom using the Socratic Method, as opposed to a lecture model; students were *required* to participate in the debate over what constitutes the elements of a "game design." We also took an approach that borrowed from creative writing programs, in which students read and critiqued fiction, and wrote.

The most reliable text resources were McCloud's *Understanding Comics* [8], which served as a primer to the aesthetic and psychological goals, and applicable creative tools in the mixed media genre of comic books; Church's "Formal Abstract Design Tools" [1], a paper written by a noted game designer attempting to create a common vocabulary with which to discuss game design goals and processes; and the Mechanics/Dynamics/Aesthetics framework created by LeBlanc [4], building on Church's concept of formal abstract design tools, to serve as a more practical guide to the process of creating actual systems.

Students were also required to play a series of pen and paper games, and critique and discuss them in class; to modify the rules of simple pen and paper games to achieve different goals; and to play early digital games and more modern digital games, to see how design goals and execution were successful or unsuccessful. The class has consistently used two non-digital games, a three to four player game, *The Settlers of Catan* [14], and a two player card game, *Lost Cities* [6]. Game tuning exercises were modeled after the two-day game tuning workshop offered as part of the Game Developers Conference [7].

2.3. Final Deliverable/Term Project

We set up a semester-long project of developing the concept and prototype of a video game, example images from which are shown in Figures 1–2, to give the students an experience akin to working in the video game industry.

Classes were broken into project teams, consisting of four to six members. Students were allowed to select their own groups based on interest in the kind of game they wanted to develop. Anyone left without a group was assigned to an existing or new group. Each team was required to come up with a game concept, "pitch" the project, write a series of documents culminating in a design document, and choose the right technical proof of concept as a final deliverable. We explicitly stated that final "shipping" code was not necessary—even a technical implementation plan based around a thorough pen and paper prototype was sufficient.[1]

The project was broken up into important milestones throughout the semester. Groups were formed by the second week of class, project proposals were due by the fourth or fifth week of class, preliminary design documents were required in place of a mid-term exam, and final documents and presentations were due during the final week of class.

The professors and TAs graded and provided feedback to the project proposals, each team gave feedback to another team's project proposal, and a full class session was devoted to a five-minute long oral version of this feedback and a discussion of each proposal with the whole class.

Following the mid-term submission of preliminary design documents, the professors and TAs met with each team, critiquing and discussing their proposal. We approached these sessions with a balance of mimicking a publisher looking at obtaining rights to these games, and training the students to design and scope viable game projects. We were at times very critical, with more than

[1] In fact, one of the best projects over the course of the semesters was a pen-and-paper proof of concept that never made it to final code. By concentrating on refining game play with pen and paper mockups, they were able to refine the game much more than many of the groups that just wrote code.

one team having to change direction completely, and at times very collaborative, suggesting improvements and methodologies.

We placed emphasis on process and diligence, discounting ultimate success and failure. We reminded students that very few ideas find life as a shipping game, and fewer still achieve any commercial success and that they were more likely to learn from "failures" than from "successes." In lieu of a final examination, groups submitted final design documents and gave 20-minute presentations to the entire class. Each group had five minutes to present their concept, ten minutes to present their prototype, and five minutes for Q&A. The TAs and professors acted as moderators (keeping each presentation within their time limits) and judges (we had score sheets that we later used to determine grades).

Projects were graded on the group's ability to hit milestones, group participation, quality of work (not on the "goodness" of the game, but whether the documentation and prototype were thoughtfully and well executed), and quality of presentation. (See the section on issues below for a discussion of projects.)

3. COURSE ITERATIONS

3.1 Unified Technology and Design

Initially, Professors Sturman and Yee co-taught the first two versions of the course in the Spring and Fall 2003 semesters, covering both technology and design. They alternated lectures on technology, which focused on general concepts used in game codebases (e.g., rendering, AI, and networking), with lectures on the process of game design.

3.2 Separate Technology and Design

The next iteration, in Spring 2004, split the course into two separate courses, one on technology and one on design. The technology course chose the Fly3D engine [2] as a codebase and the companion book *3D Games: Real-Time Rendering and Software Technology, Volume 1* by Watt and Policarpo as the text [16]. The class was based on lectures in each of the key technologies in game development, with matching structured weekly programming assignments to implement these technologies in the step-by-step development of a 3D Pac-Man style game.

The design course was able to focus purely on the process of game design. We continued the practice of semester-long team projects by creating teams of eight students, half from the design class and half from the technology class, and managed them the same as we did in the unified class, with mid-term design reviews and final presentations.

3.3 Design Only

Due to Professor Sturman's involvement as CTO for a startup, he was unavailable to continue teaching game technology.

3.4 Design and Production

The Spring 2005 course focused on design and actual production processes. Most classes were modeled after real-world development processes, with the TA serving as a producer and the instructors serving as executive producers. In this iteration, we focused on driving students through a core set of deliverables, which required them to adhere to process and form as a way to organize their group projects.

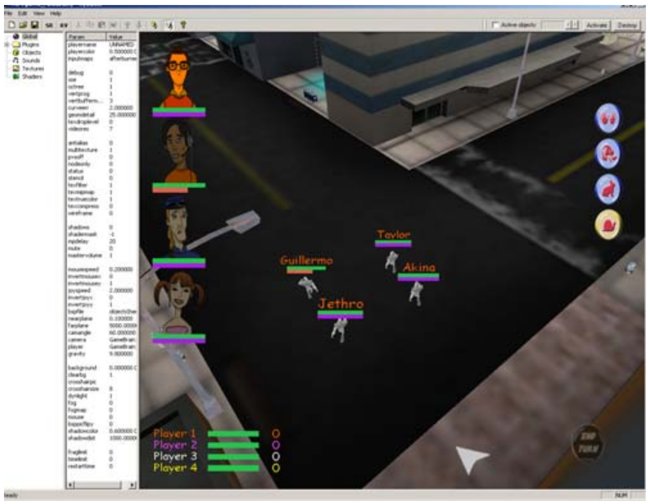


Figure 1. Student project, Spring 2004. Courtesy Amir Rao, John Waugh, Justin Titi, Mitch Morris, Pranav Behari and Fritz Meier

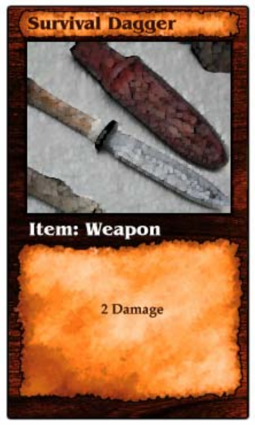


Figure 2. Student project. Pen and paper prototype, Spring 2006. Courtesy Dan Gant, Connie Shi, Steven Macanka, Shifan Mahroof, Juan Souki and Catherine MacInnes

3.5 Design and More Design

In the most recent iteration (Spring 2006), Professor Yee covered high-level design goals and production implications, co-teaching it with Austin Grossman, a well-established writer and game designer with many years of game design experience, who focused on specific design issues, such as emergent behavior and the use of physics in game design. Grossman's academic background in literature provided a backdrop to the concepts of plot and narrative, while his practical experience as a game designer gave him significant real world experience to share with the students. Much of his expertise is probably beyond the scope of the current class, and would be more useful to students in an advanced projects class, but was still good for illustrating real world examples of the methodologies taught in class. However, professional commitments in both game design and fiction have pulled Professor Grossman away from New York City.

3.6 Future Class

Professors Yee and Grossman's approach of combining practical "production and design" and a more academic "ludology and design" was probably the most successful design track taught. In an effort to maintain that dynamic, Professor Yee intends to co-teach the class with Jessica Hammer, a PhD candidate at Teachers

College, who has a more academic background in game design and has taught a similar class at Teachers College. Since she will be pursuing her PhD for several years, this also brings some stability to the faculty composition. We hope to develop a more refined curriculum after two semesters of teaching together. We also hope to use Salen and Zimmerman's *The Game Design Reader: A Rules of Play Anthology* [9] and/or Grossman's *Postmortems from Game Developer* [3], which utilizes real-world case studies and game post-mortems

4. ISSUES

4.1 Instructor Availability

Since game development is a relatively new industry that has only recently received academic attention, those who understand the development process through real-world experience are generally too bound by work on commercial projects to be able to serve as adjunct faculty.

Since few universities are willing to support an academic track devoted to game development there is no real viable full-time academic position for most game industry developers.

Furthermore, game developers are judged professionally on their contributions to shipping products, whereas in academic departments, a graduate degree is typically required for one to be hired as an adjunct. Few game developers have the requisite graduate degree needed for a faculty position.

Columbia's adjunct faculty generally must contend with the responsibilities of full-time careers, making it difficult to maintain a constant teaching presence. However, there is a strong incentive to teach the class once per academic year to be able to recruit previous students as TAs given the unique nature of our curriculum.

For the instructors, the process was immensely gratifying. We were subjected to endless variations of the management, process and creative problems we see everyday in our professional capacities. We managed many projects in a compressed timeframe, which, in some respects, served as continuing professional training. And watching students resolve to enter the game industry—and succeed—provided deep satisfaction.

Time (formal and informal) devoted to guiding projects was significant; discussion and support was not restricted to office hours so we spent many extra evenings on campus.

4.2 Teaching Design to Engineering Students

Teaching these courses in an engineering school has been a very powerful starting point, since game designers are concerned with rule systems. But pushing engineering students to think of creative game design ideas has been a challenge. We anticipated that engineering students would understand the concepts of tools, emergent behavior, and systems-based game design better than, say, their school of the arts counterparts; however, we have not found this to be true.

The instructors made great efforts to recruit students from Columbia's Teachers College (which includes gaming and media in their curriculum), and the School of the Arts. By including these "non-technical" students, we hoped to broaden the academic base for the class and the field in general and expose, perhaps even force, the engineering students to collaborate with non-technical colleagues as they would in the video game industry. By and large, this was a successful strategy, creating a broader class environment and learning experience than would have been typical in an engineering school, and much more in line with the realities of video game development.

4.3 Guidance

It was challenging to balance guiding the students away from problems and letting them make (and learn from) mistakes. We worked very hard to stay in communication with the students and encouraged them to promptly advise us of group issues. Projects that underachieved because of group problems were penalized much less if the students had communicated those problems to us all along the way than projects whose groups kept silent and used the problems as an excuse at the end. We found that this approach worked fairly well and groups who brought problems to our attention in time for us to make corrections were able to do more meaningful work than those that did not.

Another solution might be to force more project review meetings and milestone deliverables. However all approaches have the unfortunate effect of demanding a great deal of instructor time. Without our support, however, the learning experience would have suffered.

4.4 Materials

It was difficult to find a comprehensive set of materials and texts, forcing us to cobble together sources from the internet and our own resources. One of the most difficult decisions was the appropriate technology. In the first two semesters, we allowed the students to pick their own game engine technology. However, we found that groups struggled with choices (e.g., Flash vs. Game Maker vs. from scratch) and spent an inordinate amount of time struggling with technology rather than concentrating on design and development.

In the separate technology class, we standardized on the Fly3D engine, mostly because it was created for instruction and had an accompanying text. Having this standard also allowed the instructor and TAs to better support students with technology problems.

The Fly3D system was not without its problems, however, and there are several more powerful off-the-shelf tools and engines that we would consider for future classes, such as Torque [13] and Virtools [15]. Both of these engines are used in professional game development, and have the benefit of a professional developer community as well as actual support from their creators. Virtools is better suited as a designer's tool than a programming framework (although costly), while the source-code distribution of Torque would be Sturman's next technology platform of choice. Availability of a well-supported Nintendo Game Boy Advance emulation and development environment would be ideal, as that would have helped level project expectations (e.g., no overly-ambitious, sprawling 3D projects).

4.5 Coordinating Tracks

Much as we tried to coordinate the dual tracks, both in the unified and two-class model, we found it difficult to match technology topics with design topics and, so, settled for parallel instruction.

This often resulted in too much work for three credits, as we tried to fully exercise students in both disciplines.

"Technology-only" students had no clue about the design concepts being taught in the other section, and design students gained no understanding about technical constraints. Ideally students would go through both tracks in subsequent semesters with the non-technical students receiving a "technology-lite" syllabus.

4.6 Group Participation

To address the ever-present issue of unequal participation among team members we had each student submit a confidential written critique of themselves and of each of their teammates. The "360s" as we called them, gave us four independent evaluations of each student's performance. Even if they didn't all agree, we almost always saw a trend that was consistent with our own evaluations

It would have been nice to share problems and solutions more formally. We used BBS systems to encourage discussion; but, forcing more sharing and discussion is being considered.

4.7 Career Viability

Students have expressed significant interest in the industry be it case studies involving actual projects or formal internship programs. Of the major publishers, only Electronic Arts and Microsoft have formal internship programs; a student interned at EA in 2005 and is now a junior designer at their Los Angeles studio. Microsoft also generously sponsored attendance to the *Game Developers Conference* in 2004 as well. Several of our students have entered the game industry, working at Atari, EA, Large Animal, Massive, Microsoft and Nick Jr.. More formal support from the major publishers and platform owners would be welcome.

4.8 Guest Speakers

The game industry clearly has high visibility personalities; having influential, well-known and accomplished speakers—Warren Spector, Doug Church, Tim Stellmach, Bob Bates and Art Min, from companies like EA, Valve, Vicarious Visions, Eidos and Microsoft—come in was always well received, but hard to orchestrate without a budget. We were able to hold guest lectures on an opportunistic basis whenever potential speakers visited the New York area, and organized a panel sponsored by ACM in Spring 2003.

4.9 Challenges and Benefits of Cross-School Enrollment

Reaching out to non-engineering students was critical in creating a real world development team dynamic in which programmers and non-programmers had to work collaboratively. This was a big win. In fact, our best design-oriented students have been non-programming students with some informal technical/programming background. We were able to recruit students from COMS W4160 (Computer Graphics), taught by Professor Ravi Ramamoorthi, where the final project is a game written in OpenGL by two-student teams. With regard to inter-school collaborations, in Spring 2005, the final team project for COMS W4172 (3D User Interface Design), taught by Professor Steve Feiner, was done in conjunction with students in a Visual Studies course taught in the Graduate School of Architecture, Planning and Preservation. A similar interdisciplinary collaboration is being planned for Spring 2007, using a game-engine–based development infrastructure. More formal buy-in from other schools and departments would be a good next step.

4.10 Need for Lab Time

One of the important factors in designing games is playing games—it was hard to push students to play games because of the time commitment involved, especially as these games served a pedagogical purpose rather than an entertainment goal. In fact, we preferred to make students play games that put them out of their comfort zone (e.g., games that were not the most popular or current ones at the time).

5. ACKNOWLEDGEMENTS

We thank Microsoft Research for the generous donation of Xbox consoles and games for use in these courses; Electronic Arts, Vicarious Visions, Eidos, Powerhead Games, Acclaim Entertainment, Valve, and other organizations for allowing (and, in some cases, even sponsoring) their developers to give guest lectures; and all the guest lecturers who donated their time and efforts to our students, demonstrating incredible industry support, encouragement, and camaraderie to a potential new generation of video game professionals.

6. REFERENCES

1. Church, D. "Formal Abstract Design Tools." *Game Developer Magazine*, 1999.
2. Fly3D. http://fabio.policarpo.nom.br/fly3d/index.htm.
3. Grossman, A. (ed.). *Postmortems from Game Developer*. CMP Books, San Francisco, CA, 2003.
4. Hunicke, R., LeBlanc, M., and Zubek, R. "MDA: A Formal Approach to Game Design and Game Research." In Fu, D., Henke, S., and Orkin, J. (eds.), *Challenges in Game Artificial Intelligence: Papers from the 2004 AAAI Workshop*, Technical Report WS-04-04, AAAI Press, Menlo Park, CA, 2004. http://www.cs.northwestern.edu/~hunicke/MDA.pdf
5. IGDA, *Game Development, Design & Analysis Curriculum Framework*. International Game Developers Association, San Francisco, CA, 2002.
6. Knizia, R. *Lost Cities*. Kosmos Games. http://www.boardgamegeek.com/game/50
7. LeBlanc, M. *Game Tuning Workshop, 2005 Game Developers Conference* http://algorithmancy.8kindsoffun.com/GDC2005/index.html
8. McCloud, S. *Understanding Comics*. HarperPerennial, New York, NY, 1993.
9. Salen, K. and Zimmerman, E. *Rules of Play*. MIT Press, Cambridge, MA, 2003.
10. Salen, K. and Zimmerman, E. *The Game Design Reader*. MIT Press, Cambridge, MA, 2005.
11. Salvator, D. *ExtremeTech 3D Pipeline Tutorial*, ExtremeTech, http://www.extremetech.com/article/0,3396,s=1017&a=2674,00.asp, 2001
12. Simpson, J. *Game Engine Anatomy 101* ExtremeTech, http://www.extremetech.com/article2/0,3973,594,00.asp, 2002.
13. Torque. http://www.garagegames.com/
14. Teuber, K. *Settlers of Catan*. Mayfair Games. http://www.boardgamegeek.com/game/278
15. Virtools. http://www.virtools.com
16. Watt, A. and Policarpo, F. *3D Games: Real-Time Rendering and Software Technology, Volume I*, Addison Wesley, 2000

NOTE: Syllabuses, assignments, and other course materials can be found through links at http://www.cs.columbia.edu/~dsturman

Agent Augmented Game Development

Zhiqi Shen
Information Communication Institute
Nanyang Technological University
Nanyang Avenue, Singapore 679798
65-98576794
zqshen@ntu.edu.sg

Chunyan Miao
School of Computer Engineering
Nanyang Technological University
Nanyang Avenue, Singapore 679798
65-67906197
ascymiao@ntu.edu.sg

Yundong Cai
School of Computer Engineering
Nanyang Technological University
Nanyang Avenue, Singapore 679798
65-67906197
caiy0004@ntu.edu.sg

ABSTRACT

Computer Science students are not only interested in playing games but also highly motivated to learn how games are developed. This paper proposes a novel agent augmented game development framework in a 3D virtualized environment that allows students to become the players, the situated learners and the designers, who create games/stories in an immersive 3D environment, simultaneously.

Unlike most of existing games in which agents are designed to play some roles or to execute specific tasks, we augment the whole game world as an interactive multi-agent system. The agent intractability and autonomy enhance the user interactions and enable the dynamic story creation in a situated learning/playing environment. The proposed approach significantly increases the interests of students in learning to design the game from playing.

An interactive game for science learning in secondary schools is presented to illustrate our approach. The proposed game framework as well as the 3D virtualized game environment has been successfully used in a number of capstone projects in recent two years. Students enjoyed learning and designing games through playing.

Categories and Subject Descriptors

K.3.1 [**Computer Uses in Education**]: Collaborative learning – *game-like learning, games, virtual world.*

General Terms

Design.

Keywords

Game development, Agent augmentation, Multi-agent systems.

1. INTRODUCTION

Computer Science students are not only interested in playing games but also highly motivated to learn how games are developed. However, most of the computer science students think game development is challenging. Traditionally, students need to learn game development in the class first before they start to practice game development. Therefore, there is a big gap between the learning and real development of the games. In this paper, we present a novel approach that bridges this gap through an agent augmented 3-D virtualized environment for situated game design learning and game development. Each entity in the game world, whether it is a graphic object, an avatar or a user player, could be augmented with an agent. The whole game world will become an interactive multi-agent system. Such environment is supported by our agent augmented game development framework which acts not only as a game development teaching and learning tool but also a real game development environment.

Software agents as autonomous entities have many important characteristics, such as goal oriented, proactive, communicative, and intelligent. Agent augmentation facilitates the interactions between characters, players and dynamic stories. It also increases the involvements of players and interests of players to play and learn in a situated game environment. To enable students to learn game design in a situated environment and to develop games that support interactions and dynamic storytelling, an enabling agent augmented game development framework is critical.

Research efforts on game development with agent technology have been reported [1, 2, 3, 4]. However, in those game development environments, agents either act as specific roles or are designed to execute specific tasks. It is far from the expectation for the dynamic game design/storytelling and the interaction between game and player. In this paper, we present a novel game development framework whereby the story sequences could be decided by the players on the fly. We regard a game as a multi-agent system so that agent interactions and autonomy can be used to increase interactions and support dynamic storytelling/game design. Goal Net [5], a modeling tool, is used for story/game design and agent design. With this framework students can learn not only the conventional game development but also dynamic storytelling.

The paper is organized as follows. Following this section, the related work is studied. In Section 3, the theoretical background of the proposed game development framework is introduced. The agent augmented game development framework is described in Section 4. Section 5 presents a game for science learning in secondary schools developed using this framework by our students to illustrate our approach. Finally, the conclusion and the future work are discussed in Section 6.

2. RELATED WORK

Using agents in games has improved storytelling significantly. VISTA (Virtual Interactive Story Telling Agents) [1] project introduced an agent based architecture whereby players can interact with agents with questions they want to ask and therefore the players can understand the contents of the story better. SAGE (Storytelling Agent Generation Environment) [2] project constructed storytellers using agents to encourage players' emotional engagement. Mark Riedl, at. el. [3] proposed a narrative mediation to leverage the player involvement and the overall narrative. Marc Cavazza, at. el. [4] used HTN (Hierarchical Task Networks) for agents behavior planning.

In those projects, agents are used to increase interactions between characters in a story and players. On the other hand, techniques are used to plan the agents' behaviors and to control the interactions between agents and players so that the storyline is still maintained.

Unlike most of existing games in which agents are designed to play some roles or to execute specific tasks, we augment the whole game world as an interactive multi-agent system. More specifically, we use Goal Net, a goal oriented model to design both characters and plots in storytelling. Each entity in the game world, whether it is a graphic object, an avatar or a user player, could be augmented with an agent who carries a goal net. The whole game world will become an interactive multi-agent system. More over, we use Fuzzy Cognitive Map (FCM) [6] to model the emotions of characters or agents to simulate the human-like roles/characters in real life. The objective is to create a 3D virtualized immersive environment that allows students to become simultaneously the players, the situated learners and the designers who create games/stories in such an environment.

3. THEORITICAL BACKGROUND

3.1 Goal Net

Goal Net is a tool for modeling goals of an agent in a multi-agent system. It has been successfully used in the multi-agent system modeling for business forecasting, grid computing and E-learning. Goal Net model is composed of *Goals* and *Transitions*. *Goals*, represented by circles, are used to represent the goals that an agent needs to pursue. *Transitions*, represented by arcs and rectangles or vertical bars, connect from the input goal to the output goal, specifying the relationship between the two goals. Each transition is associated with a task list which defines the possible tasks that the agent needs to perform in order to transit from the input goal to the output goal. A goal net example is shown in Figure 1. There are two kinds of goals in Goal Net, *atomic goals* and *composite goals*. An *atomic goal* is a primitive goal which cannot be further decomposed, while a *composite goal* can be split into goals connected via transitions. Therefore, a complex goal can be recursively decomposed into sub-goals and sub-goal nets. The hierarchical structure simplifies the goal modeling process with different levels of abstraction.

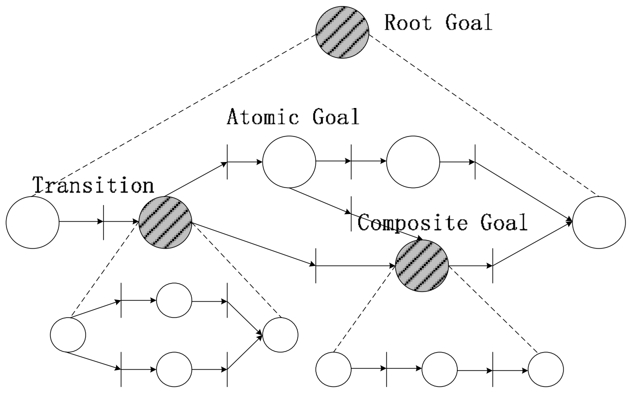


Figure 1. A Goal Net example.

In Goal Net, there are four types of temporal relations of goals represented by *transitions* connected the input goals and output goals: *sequence*, *choice*, *concurrency* and *synchronization*, as shown in Figure 2.

- Sequence: A direct sequential causal relationship between input goal *i* and output goal *j*.
- Choice: A selective connection from input goal *i* to possible output goals *j* and *k*, and only one output goal can be selected based on selection criteria.
- Concurrency: Input goal *i* at completing the tasks, all the output goals *j* and *k* can be achieved simultaneously.
- Synchronization: A synchronization point from different input goals *i* and *j* to a single output goal *k*, and transition to the output goal can only be started when all its input goals are achieved.

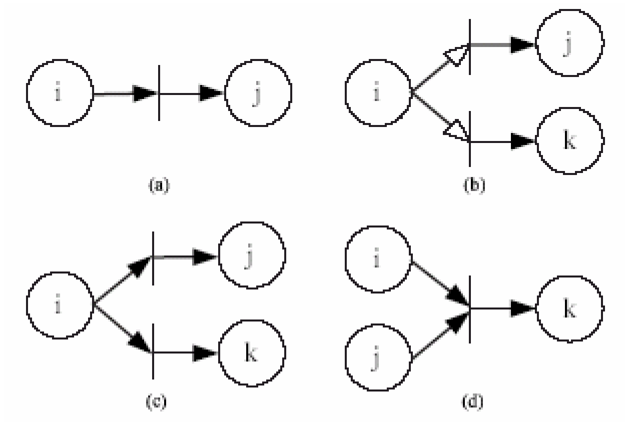


Figure 2. Temporal relationships among goals.

3.2 FCM

Fuzzy Cognitive Maps (FCMs) is a kind of causal relationship modelling tool. It provides a simple and straightforward way to model the relationships among different factors. FCMs include two elements: *concepts* and *causal relationships*. As shown in Figure 3, *concepts* are represented by circles, which represent the related causes and effects in the model. The *causal relationships* are represented by directed arcs, each of which has a sign and a weight. The '+' sign means *positive* causal relation, such that the increase of the starting concept value may cause the increase of the ending concept value. Conversely, the '-' sign means *negative* causal relation, such that the increase of starting concept value will cause the decrease of the ending concept value. The weights differentiate the important levels from the starting concept to certain ending concept. Each concept is represented with a state value whose range is in [0, 1] or [-1, 1], while the causal relation is represented as a weight, whose range is in [-1, 1].

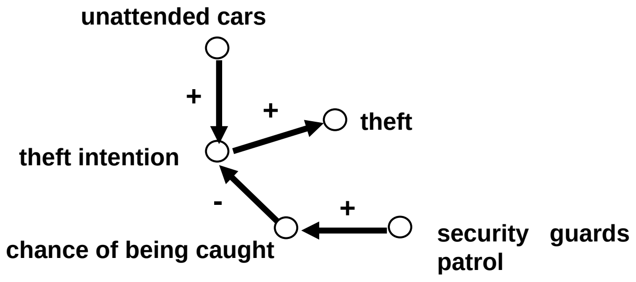


Figure 3. A FCM example.

3.3 Dynamic Storytelling

Goal Net is used in the framework to model the goals of an agent – a character in the game and the dynamic plans of the story – plots in the game for interactive storytelling. To model a character as an agent, goals and the temporal relationships of each character are modeled as a goal net. The four kinds of temporal relationships of Goal Net make the characters/agents behave according to the real situation during playing. To model plots of a story, a plot is regarded as a goal. A plot can be split to sub plots. The whole set of plots of a story can be designed as a goal net (plot net). The plots are organized according to their temporal relationships. The real storyline will be generated according to the goal net (plot net) during playing through interactions between the players and the characters.

3.4 Interactions

In the framework, FCMs are used as the reasoning tool for goal selection as well as action selection. The interaction from the player, context variables and the possible goals and actions are encapsulated as concepts. The causal relationships among different concepts are determined according to the expert knowledge or predefined rules in the knowledge base. High weights are assigned to those more assertive causal relationships.

Each character make decision for its behavior according to its goal net and FCMs. For example, the time a character *visitor* spent to talk with another character *patient* in a hospital is longer than a threshold, the character *visitor* may become sick. Hence its behavior may also change. The player may interact with the character for resolutions. Then the storyline may be also changed according to the dynamic plan of the goal net (plot net).

4. FRAMEWORK

4.1 System Architecture

The game system consists of four components as shown in Figure 4: *client application*, *game server*, *agent server*, *3D object server*.

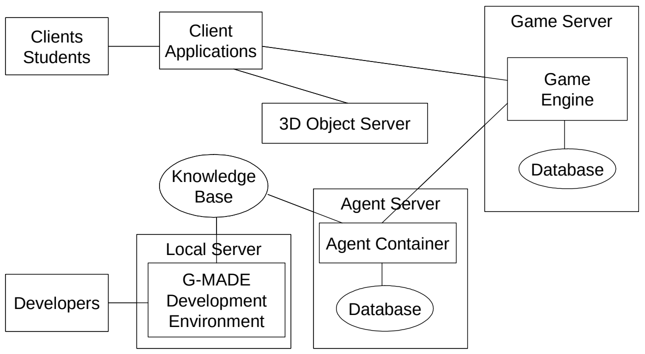


Figure 4. System architecture.

- Client application is user interface to players, where a player plays the game on his/her computer.
- Game server is the engine to run the game programs. It is also the collaborative space to synchronize different players in the game.
- Agent server is the container of the agents - the characters in the game. Each character has a 3D object in the game and an agent on the agent server. The behavior of the character is controlled by the agent.
- Object server is the container of the 3D objects. It contains the 3D objects used for the virtual world of the game. Each character has a 3D object on the object server.

When a game is started, the virtual world of the game will be created at the game server. The 3D objects are loaded to the client application to build the game environment. The characters will be loaded to the client application as well. The agents corresponding to the characters of the game including the avatar that represents the player are created on the agent server. The commands for the behaviors of the characters are controlled by the agents. Only the avatars can be controlled by both the player and the corresponding agents. Usually commands from the avatar agent are notifications or emotional behaviors reflecting the situation changes in the games. So the unreasonable conflict between the commands from the agent and those from the player is minimized. If another player joins the game, the whole set of the game environment will be loaded to his/her client applications.

4.2 Development Framework

The development framework consists of 3D object development, agent development, and game development. The game development and 3D object development can be done using commercial products in the market. In the paper, we only focus on the agent development. The agent development, namely G-MADE (Multi-Agent Development Environment for Game), consists of the Goal Net Designer, FCM designer, Agent Creator and Goal Net Loader and SDK (Software Development Kit) of game engine on the game server. Goal Net Designer, Agent Creator and Goal Net Loader are the key components of the MADE (Multi-Agent Development Environment) used for multi-agent system development [7]. G-MADE extends MADE by integration with FCM Designer and SDK of a game engine used in the framework. By this extension, agents can make decision on the actions using FCM inference and manipulate the characters in the game through APIs (Application Programming Interfaces) of the SDK.

4.3 Game Development

In this paper, we regard a game as a multi-agent system. So we map the game development to the agent development using the framework presented in this paper.

4.3.1 Agents Development

Typically, there are four steps developing an agent using G-MADE. They are:

- Function development: Functions are the actions a character may perform. In this step, all the functions will be developed using the SDK of the game engine.
- Goal Net design: Goal Net is the "brain" of an agent. In another words, an agent will pursue the goals according to the temporal relationships among them designed in a goal net. The functions developed at the last step will be used in this step to construct transitions of the goal net so that the agent can control the character represented by this agent.
- FCM design: Within a goal net, the agent needs to make decision to select suitable goal to pursue at the next step or select the next task to perform to pursue a goal according to the current situation. FCM reasoning is used to make decisions for the agent based on the selected context variables.
- Agent creation: With G-MADE, an agent is Goal Net enabled. Every agent created by the Agent Creator has the same structure. It does nothing after it is created. It becomes an active agent only after a goal net is loaded by the Goal Net Loader. So different agents/characters can be identified by different goal nets.

4.3.2 Characters

Each character in the game has at least one goal net associated with. When the character starts to appear in the game, a goal net will be loaded to the corresponding agent to control its behavior. In a game as a multi-agent system, agents/characters need to cooperate or coordinate with each other according to the story. This can be done through goal nets of the plots.

4.3.3 Plots

Plots design is important for storytelling. With Goal Net, a story can be decomposed to many smaller plots. Based on the temporal relationships and dynamics of the story, a goal net (plot net) can be constructed. Each atomic plot is a goal net containing all the goals of characters that should appear in the plot. Each goal represents a goal net of a character. So in this plot, all the goals of characters will be delegated to the corresponding agents to perform.

When a game is started, a special agent, namely director agent or instructor agent will be created who will take the goal net of plots to behave. The whole storytelling or the game will be controlled or directed by this agent. However the real sequence of plots will be generated dynamically according to the interactions with the players.

5. GAME FOR SCIENCE LEARNING

A game, namely "Mystery Illness Investigation at Nanyang Town", was developed using the framework by the capstone project students in the school of computer engineering. The purpose of the game is to teach students in secondary schools about illnesses in a 3-D immersive environment as if in the real life.

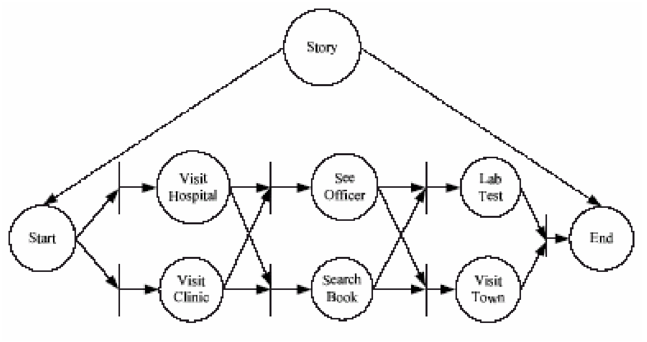


Figure 5. The goal net of the story.

The stories in the game guide players to explore the virtual town and investigate the mystery illness. By talking to different characters at different places in the game, or conducting lab experiments, the players need to find the symptoms of the mystery illness, study the differences among diseases, and conclude the thorough review over the mystery illness by the end of the game. The goal net (plot net) of the story is shown in Figure 5. As shown in the goal net (plot net) the players can go to either the hospital or the clinic to check the symptoms of the mystery illness and how widely the illness is spread. Depending on the availability of the officer in the ministry of health, the player can choose to ask for differences of different illnesses from the officer, or go to library to check them from the books. Moreover, the player can go to the town to verify the conclusion about the mystery illness, or he can do some further laboratory tests. The director agent selects a storyline as shown in Figure 6, in which the player should visit the hospital for illness symptoms, then go to meet officer to query about the differences of the illnesses, lastly go to visit the town to confirm the conclusion. Figure 7 shows a detailed design of plot "visit the hospital". The story involves three characters: a doctor, a nurse and the avatar representing a player. As shown in Figure 8, the director agent delegates the tasks of visiting different places to the player, and the player is able to interact in the first-person view and third-person view.

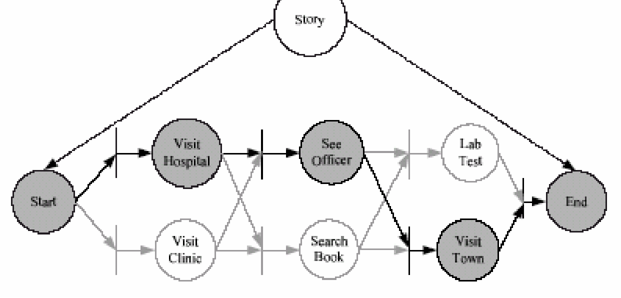


Figure 6. The selected storyline.

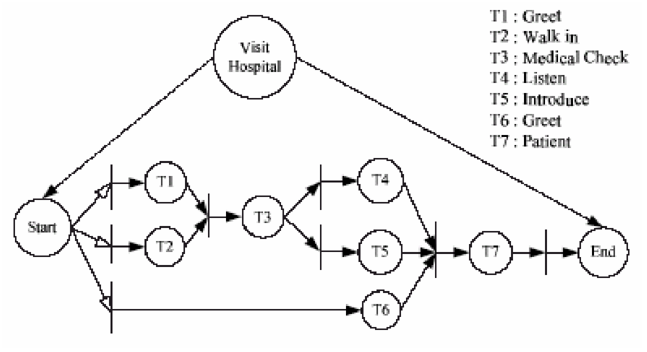


Figure 7. Goal net for visiting the hospital.

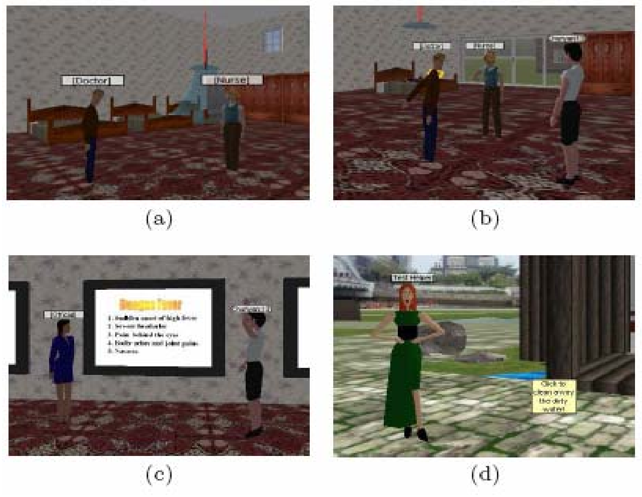


Figure 8. The screen shots of the game.

The whole game development infuses playing, learning, and development into an integrated process. In the beginning, students login as user avatars and start playing in a preliminary simple world. Then the students start to enrich the world or create their own new world with various virtual entities, i.e. graphic objects, avatars. After that students start to design game strategies, story scenes using Goal Net Designer for the virtual entities they created. Finally the students create the dummy agents, load goal nets into the agents and then augment the agents into the virtual entities to make their game world an interactive multi-agent system in a 3D immersive environment.

6. CONCLUSION AND FUTURE WORK

This paper proposes a novel game development framework based on Goal Net and FCM. Goal Net is used to design both characters and plots in storytelling so that character-based story design and plots-based story design can be combined and therefore a hybrid game system can be constructed. FCM is used to design the emotions of characters or agents so that interactions between agents and players and story dynamics are increased. The demonstration of game for science learning illustrates the proposed framework is practical and easy to use. With this framework students can learn game development in a situated learning environment and develop games with intelligence/agent augmentation in a 3-D immersive environment.

The novelty of our approach is that such agent augmented 3D virtualized game development environment allows students to become simultaneously the players, the situated learners and the designers who create games/stories in an immersive 3D environment. It bridges the gap between the learning content of game development and real game development. It also transfers the challenging of learning and development into playing. More over, such environment supports collaborative learning and game design in a virtual team.

Our on-going work focuses on extending our framework with Microsoft XNA game engine. Our future work will focus on the situation awareness and intelligent question answer so that players will experience different story plots with the same game during playing.

7. REFERENCES

[1] Figa, E. and Tarau, P. The VISTA Project: An Agent Architecture for Virtual Interactive Storytelling. *CSCW 2002 Workshop on Storytelling and Collaborative Activities* (New Orleans, Louisiana, USA, November 16, 2002).

[2] Umaschi,M., and Cassell, J. Storytelling systems: constructing the innerface of the interface. In *Proceedings of the 2nd International Conference on Cognitive Technology (CT '97)* (Aizu, Japan, August 25-28, 1997). IEEE Computer Society Press, 1997, 98-108.

[3] Fröhlich, B. and Plate, J. The cubic mouse: a new device for three-dimensional iput. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems (AAMAS2003)* (Melbourne, Australia, July 14-18, 2003). ACM Press, New York, NY, 2003, 741-748.

[4] Cavazza, M., Charles, F., and Mead, S. Agents' Interaction in Virtual Storytelling. In De Antonio, Aylett, and Ballin (eds.), Intelligent Virtual Agents, Springer Lecture Notes in Artificial Intelligence, vol. 2190, (2001), 156-170.

[5] Shen, Z. Q., Miao, C. Y., and Gay, R. Goal Oriented Modeling for Intelligent Software Agents. In *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'04)* (Beijing, China, September 20 - 24, 2004).

[6] Kosko, B. Fuzzy cognitive maps. *International Journal of Man Machine Studies*, 24, (1986), 66-75.

[7] Shen, Z. Q., Miao, C. Y., and Gay, R. Goal-oriented Methodology for Agent-oriented Software Engineering. *IEICE Transactions on Information and Systems, Special Issue on Knowledge-based Software Engineering*, Vol. E89-D, No. 4, (Apr. 2006).

Teaching Revulsion-Free Design Patterns through Game Development

Randy Connolly
Dept. Computer Science & Information Systems
Mount Royal College
Calgary, AB, T2N 0Z6
403 440 6061
rconnolly@mtroyal.ca

ABSTRACT

This paper describes how game development can be used to successfully teach design patterns to undergraduate computer science students. The abstract nature of design patterns can make them difficult for students to fully comprehend and successfully integrate into their applications. The familiarity of games along with their flexibility provides an ideal context for making the abstract patterns more concrete and understandable. Three different game projects are described along with several of the design patterns implemented within these games.

Categories and Subject Descriptors

D.1.5 [**Software**]: Programming Techniques – *Object-Oriented Programming.* K.3 [**Computers & Education**]: Computer & Information System Education – *Computer Science Education.*

General Terms

Design.

Keywords

Design Patterns, Game Development.

1. INTRODUCTION

"The mark of our time is its revulsion against imposed patterns." -- Marshall McLuhan [9].

McLuhan's assertion about a revulsion against patterns may indeed have been true when he wrote those words in the early 1960s, but they very much appear to be false when it comes to contemporary thinking in software design. Since 1995, the software design field has been great enhanced by a burgeoning literature in the area of software design patterns. This literature very much tends to be oriented towards the experienced software development practitioner. Less experienced developers, such as undergraduate computer science students, generally however find design patterns to be too abstract and their development experience too limited to find the same joy and attraction to patterns. We might say then that students very often do in fact have a sense of revulsion against the patterns that are imposed by their instructors!

A design pattern is a description of a class-level solution to a common generalized design problem. In the key text in the field, the so-called Gang of Four book [4], 23 patterns are described at a relatively abstract level and are related using examples which are usually unfamiliar to typical student readers. Other authors have endeavored to teach design patterns by using more approachable language and examples (see, for instance, [10], [12], [8] and the excellent [3]).

Yet despite these much more approachable texts, this author has often struggled to find appropriate contexts and examples for teaching design patterns. Students typically were able to parrot the description of the covered patterns in examinations but generally struggled to implement them and almost always completely failed to see the point of these patterns. (It should be noted that some researchers in contrast have had some success introducing design patterns into entry-level CS courses [2,5,6]). To this author's students, design patterns almost always seemed an unnecessary and painful complication. Students often solved a problem their own way and then tried at the last moment to hammer the square peg of the design pattern into the round hole of their solution after it was already working. It is no wonder then that the students often felt a revulsion against design patterns. This experience, however, changed quite substantially once design patterns were taught using game development projects.

This paper describes the author's relatively successful experiences teaching revulsion-free design patterns to third-year undergraduate students by using game development projects. It provides an overview of these games and then illustrates some of the design patterns that were introduced as solution mechanisms within these projects.

2. THE GAME PROJECTS

In our three-year applied degree program, students are exposed to a variety of application development environments. Students take three programming-focused courses (from structured to object-oriented) using Java, two courses devoted to web application development, and two courses teaching database design and development. One perceived lacuna in the students' education is that they never create native Windows client applications. Three years ago we created a capstone course in Windows development that has used C# and Windows Forms within Microsoft's .NET Framework.

Two years ago this author began to use game projects in this course. There was ample reason for doing so. Within the field of education, there is "an abundance of literature to support the use of games as tools that help learners."[11] Within the context of computer science, a variety of researchers have found game assignments to be helpful for teaching and motivating introductory programming students (see for instance, [1] and [5]). As well, it has been noted that games can provide "an extremely project-oriented, upper-division course to exercise and enhance the programming and problem-solving skills of advanced students" [7].

In the three iterations of this course, this author has taken two basic approaches. In the first approach the students dove right into Windows Forms (the students were already familiar with C# and ASP.NET) and created a role-playing game. Students worked in pairs to create a game in which a player (in the role of a barbarian, knight, wizard, or ninja, each with its own unique statistics) navigates a multi-screen map and fights monsters based on a configurable combat system. For simplicity sake, students used GDI+ rather than DirectX for drawing graphics. Various additional custom controls had to be created to handle status messages, the character's state, and the character's position in the game world. As little game information (e.g., map, actors, world, etc.) as possible was contained within the code; instead this information had to be contained in XML files. Figure 1 illustrates a sample student project.

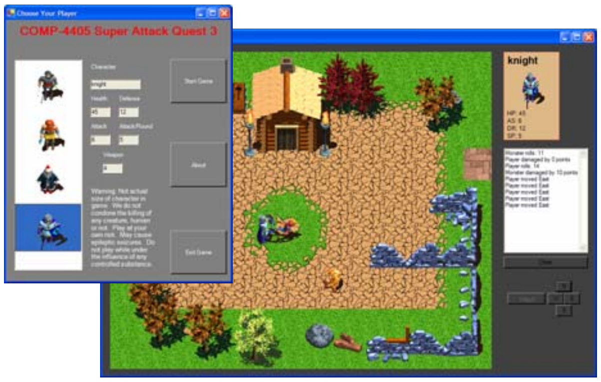


Figure 1: Sample student project for first game

The students were provided with a variety of royalty-free graphical resources (Figure 2 shows an example). These included several hundred tile files to construct maps, static and animated item images for placement on the map, as well as over a hundred animation strip images for player and monster actors. Each monster or player actor had four direction facings (north, east, south, and west) and five states (paused, walking, attacking, being hit, and dying) that had to be animated.

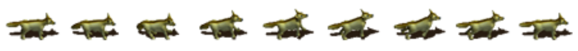


Figure 2: Sample animation strip (wolf attacking east)

In the course labs, students were introduced to the following: GDI+ development, creating custom controls, parsing XML, and working with timers. In one of the labs, students were guided through the construction of a simple sprite (an independent animated object) as a means of demonstrating how to use an event-based timer as a first step in learning multi-threaded programming.

In the second and third iterations of the course, the author took a somewhat simpler approach. These students did not know C# and were somewhat weaker since the course was now offered in the fifth semester. As a consequence, easier game projects were necessary.

One of these games was a Rogue-like dungeon crawl. A Rogue-like game is one that involves navigating a character who has certain attributes and inventory items around a maze-like map fighting monsters and collecting treasure. The next year's iteration of the course used instead a combat game, in which the player creates an army of units containing warriors that battle against an opponent's army. Figure 3 illustrates some sample versions of these two game projects.

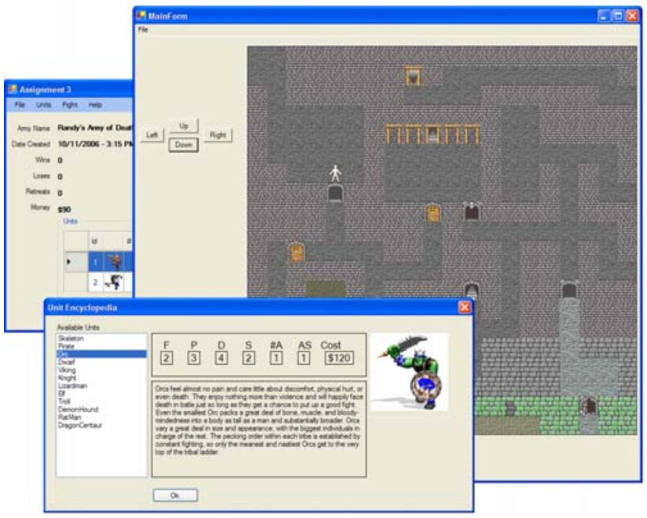


Figure 3: Two other game projects

In all three projects, the game project was broken down into four or five milestones delivered at various points through the semester. In the first milestone, the students created a console version of a simplified subset of the game. Since there was very little user-interface interaction required, the students could concentrate on designing and implementing the basic domain model and in the process familiarize themselves with C#.

The second milestone involved moving their solution to Windows Forms. It is at this stage that students begin to see the value of software design principles. Students recognize the worthiness of the general object-oriented principle that one should separate that which varies from that which stays the same, when they begin porting their first milestone solution to the second. Almost without exception, students have intertwined user interface logic within their domain model. As they rewrite their original design to fit the new user interface requirements, the students begin (perhaps for the first time) to become really receptive to the idea that a proper design will save them time and effort.

Wick [13] has noted that by "starting students with a design that they find reasonable and understandable and then refactoring that design to introduce design patterns, students get a better appreciation for value-added by the design pattern." The two additional final milestones for these game projects followed this approach. Additional requirements were introduced (such as navigating around an XML-based map, adding animated sprites, or modifying the game to work on a Mobile PC device). To help implement and manage these changes, various design patterns were introduced as possible solutions to the problems raised by these new requirements. In general, students were much more receptive to these patterns and appeared to have significantly fewer problems implementing them then in my previous non-game attempts (in other courses) at teaching design patterns. The next section provides some details on how certain design patterns were introduced and integrated into the game projects.

3. THE DESIGN PATTERNS USED

One of the real benefits of game projects is the ease at which complexity can be introduced into that project. The typical business application that can be implemented within a semester has a very typical workflow: retrieve data, transform and present data, validate changes to it, and then save it. The architecture and design of such an application is also quite stereotypical, in that one usually architects the project into three logical layers (such as presentation, business, and data access layers).

Games potentially have a much more open-ended workflow and as a consequence are less amenable to the typical business application architecture. A game can force students to construct their own architecture (for the world and its maps, for the actors, for the sprites, for the combat system, for the artificial intelligence, etc.) and their own process (e.g., when should the map files be read, when should the images be read for the actors, should all the images be stored in memory, etc).

This open-endedness provides the perfect canvas for the instructor to create the appropriate game project based on the design patterns he or she wishes to "paint" into the students' minds. Depending upon which project was used, a variety of different patterns have been covered in this course. These have included the Singleton (for creating a single repository of all images), Memento (for implementing an undo system), Mediator (for coordination between different user controls), Factory (for creating GDI+ images based on tile keys), State (for handling each game actor's state), Composite (for implementing a hierarchical inventory), Observer (for handling the game events caused by the actors, Strategy (for implementing a run-time configurable combat system), and Command (for handling different user-specifiable game actions). We do not have the space to cover all these patterns; the remainder of the paper will describe three of these in more detail.

Just before covering these three patterns, two important pedagogical beliefs that informed the author's approach need to be mentioned. The first of these beliefs is that students learn the best path by strolling down, for a while at least, mistaken paths. This means that the students are guided into a solution to a problem by first encountering the problem in their own programming. Thus, a design pattern is covered only after the students have already written some messy code as a solution to a problem in an earlier milestone; this way, the students are more likely to see the benefit of the design pattern in contrast to their own solution. The other pedagogical belief is that students learn best about the utility of proper design when requirements shift. That is, a good design pays most of its dividends as a project's scope changes and widens. It is relatively easy to do this via game projects. Students would be forewarned that some very particular aspect of the assignment was going to be modified close to the due date; the students thus needed to design with this adaptability in mind. The students were often appreciative of how certain design patterns made these changing requirements easer to manage.

3.1 Singleton Pattern

The Singleton pattern is perhaps the most straightforward and easy to implement of the Gang of Four patterns. This pattern is used to ensure that only one instance of a particular class ever exists and that this instance is globally available to other classes.

While the Singleton is fairly easy to implement, students are often unclear as to when it should be used. One of the few principles that students pick up from their first year introduction to programming class is that global variables are usually to be avoided. Perversely, students embrace the global-creating Singleton pattern like a smoker returning to tobacco after an unpleasant enforced absence. That is, students initially suffer from "Singletonitis" [8], and make use of it in way too many classes.

The important point about the Singleton pattern is that it is helpful when multiple instances of a class would consume too much memory or slow the system too noticeably. It is often difficult for relatively inexperienced programmers to know in advance what kind of class might cause such a problem. With a game project, on the other hand, it is often much easier to make the students *see* the effect of a class that consumes too much memory when instantiated multiple times. Games make this possible since the game will often become too slow and visibly sluggish.

The effect of a memory-intensive class is particularly evident with those that use graphical resources. For instance, a 2D map representing the world in which the character moves might be implemented by a 1000x1000 element array, with each element in the array containing a map tile that is 32x32 pixels in dimension and about 20K in size. The memory consumed by such a map would be prodigious if each map cell had a unique tile. Luckily, all we generally need to create a visually interesting map are three or four dozen unique tile image files, which will make the map significantly more memory-friendly. Nonetheless, the map still consumes a lot of memory and it is important to ensure that there is only ever one instance of the map in memory and that each tile image file used in the map only exists once as well.

Discussing this problem in class, students very quickly comprehend why only one instance each of these two things should ever exist, but are unsure of how to do it. The Singleton pattern, once introduced, is typically quickly integrated into the students' solutions. Most students end up implementing the pattern similar to that shown in Figure 4.

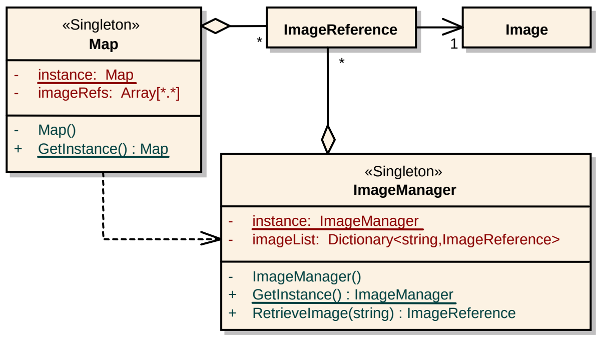


Figure 4: Sample Singleton patterns

3.2 Strategy Pattern

The Strategy pattern is used to define a family of algorithms that are encapsulated as separate classes and which can be used interchangeably. This pattern makes it easier to dynamically configure an application at run-time to use different swappable algorithms for a given single task. The vast majority of students have played video games in which the difficulty level is configurable by the user. As a result, the value of a design in which an algorithm can be varied independently of the program that is using it, can make intrinsic sense to the students when described in the context of a game.

In the game projects used in this course, one of the common requirements has been the need to allow the user to specify which combat system the game is to use. The first milestone uses a reasonably simple combat system (player rolls a dice to inflict damage, and then the computer rolls a dice to inflict damage back). In subsequent milestones the students must implement progressively more complex combat systems based on different business rules; their programs must always, however, allow the user to choose at run-time the combat system that will be used.

Following the usual approach, the best solution to this problem was addressed after the students had created their second milestone (i.e., after they had implemented two combat system algorithms and needed to support two more). At this point the students typically had a number of parallel conditional structures that were located in two or more classes.

When the Strategy pattern was covered, most students immediately saw in it a solution to their existing "ugly" and hard-to-maintain configurable combat system from their second milestone. After covering the Strategy pattern, most students ended up with a solution similar to that shown in Figure 5.

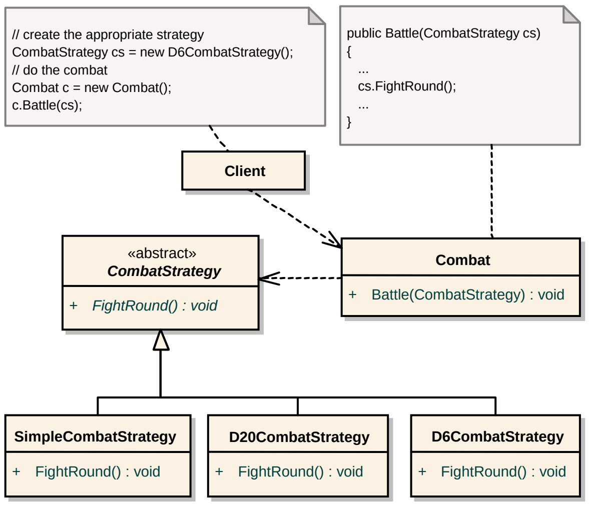


Figure 5: Sample Strategy pattern

3.3 Observer Pattern

The Observer pattern is used to implement a one-to-many dependency between objects, so that when one object's state changes in a particular way, all of its dependencies are notified. This pattern is also sometimes referred to as the Publish-Subscribe pattern in that one object (the subject) "publishes" events that subscribers (the observable subjects) are interested in. This pattern is often used when constructing graphical user interfaces that follow the MVC (Model-View-Controller) model. C# has built-in support for the Observer pattern via event delegates.

In one of the game projects, the different game actors (both the computer's and the player's) could perform different actions such as fight a combat, die, leave a combat, pick up treasure, move to a different location, and use a treasure item. There were typically several different classes that were "interested" in these events, such as the control that displayed the game map, the control that displayed the different commands that could be performed at any given time, and the control that displayed the status and statistics of the actor. This last control was a purposely late addition to the requirements of the project.

Once again the students and the instructor discussed in lectures the different possible approaches to dealing with the complexities of these requirements. After struggling through this problem in their third milestone, the students were positively excited to learn about the Observer pattern, and most students were able to implement something similar to that shown in Figure 6.

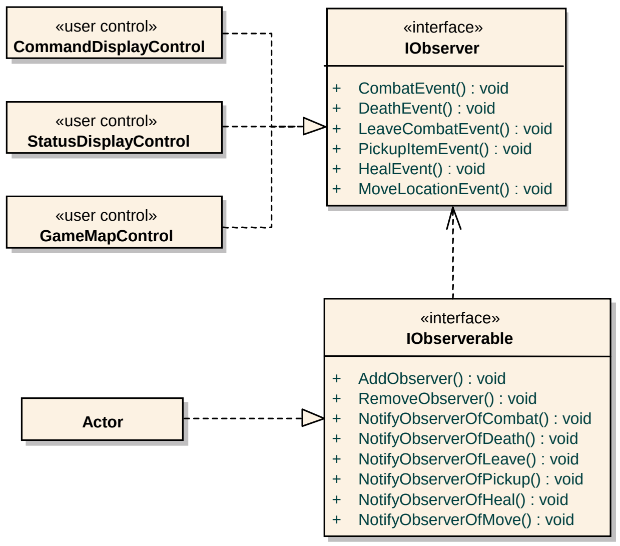

«user control»
CommandDisplayControl
«user control»
StatusDisplayControl
«user control»
GameMapControl
«interface»
IObserver
+ CombatEvent() : void
+ DeathEvent() : void
+ LeaveCombatEvent() : void
+ PickupItemEvent() : void
+ HealEvent() : void
+ MoveLocationEvent() : void
«interface»
IObserverable
+ AddObserver() : void
+ RemoveObserver() : void
+ NotifyObserverOfCombat() : void
+ NotifyObserverOfDeath() : void
+ NotifyObserverOfLeave() : void
+ NotifyObserverOfPickup() : void
+ NotifyObserverOfHeal() : void
+ NotifyObserverOfMove() : void
Actor

Figure 6: Sample Observer pattern

3.4 Making the Students Believe in Interfaces

Design patterns require knowledge of more advanced object principles such as abstract classes, interfaces, and polymorphism. This author has always struggled in particular in convincing students about the necessity and utility of interfaces. Students can typically be convinced that abstract classes and polymorphism are both good and necessary things. They typically remain, however, quite agnostic towards interfaces. They recognize that interfaces might exist but they certainly do not plan on changing their (programming) behaviors and use them since they can not see why one would use an interface instead of an abstract class.

Through the use of game projects, however, this author has managed to turn the students (or at least some of them) into true interface believers. The Observer pattern as shown in Figure 6 does use interfaces; one could however implement it in C#

without interfaces by using delegates. The introduction of animation into their projects has been a particularly fortuitous way to reiterate the utility of interfaces.

In the game, animation is introduced in the final milestones. An animated game object must be able to start animating itself, stop animating itself, play itself just once, and choose its animation speed. The problem is that the students must introduce animation into several different classes in their domain. Players and monsters must animate, combat effects must animate, inventory items must animate, and map items (such as trees or lava) must animate. Adding animation to a common base class is not too attractive a solution since these different things will have quite different object hierarchies, and it is too late in the semester to redesign the entire object model. Another reason why an abstract class is not too attractive here is that the animation implementation varies depending upon which type of thing is being animated (e.g., some objects are animated via animation strip images, while others are animated via GDI+).

Interfaces provide a ready made solution to the students' animation problem. After discussing interfaces again in the lectures, most students come up with a solution similar to that shown in Figure 7.

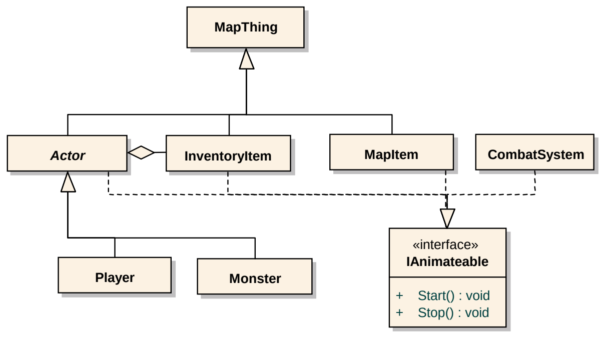


Figure 7: Interfaces at work

4. CONCLUSION

In this author's experience, game development is an ideal way to teach design patterns. Games provide a rich context in which students are able to recognize the utility and appropriateness of the original Gang of Four design patterns along with the higher-level design principles that the patterns encapsulate. Along with the benefits of learning design patterns, games are as well an ideal capstone-style project for upper-level students. As Jones as noted [7], the "integration of concepts and techniques required to design and build computer games covers many of the topics offered in an undergraduate computer science curriculum, allowing students concrete application of much of the theory, concepts, and skills they have been exposed to."

5. REFERENCES

[1] Becker, K. "Teaching With Games: The Minesweeper and Asteroids Experience." *The Journal of Computing in Small Colleges*, 17, 2 (December 2001).

[2] Dewan, P. "Teaching Inter-Object Design Patterns to Freshmen." *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, 37, 1 (February 2005).

[3] Freeman, Er., Freeman, El. *Head First Design Patterns* O'Reilly, 2004.

[4] Gamma, E., Helm, R., Johnson, R., Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[5] Giguette, R. "Pre-Games: Games Designed to Introduce CS1 and CS2 Programming Assignments." *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, 35 1 (January 2003).

[6] Hansen, S. "The Game of Set – An Ideal Example for Introducing Polymorphism and Design Patterns." *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, 36, 1 (March 2004).

[7] Jones, R. M. "Design and Implementation of Computer Games: A Capstone Course for Undergraduate Computer Science Education." *Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education*, 32, 1 (March 2000).

[8] Kerievsky, J. *Refactoring to Patterns*. Addison-Wesley, 2005.

[9] McLuhan, M. *Understanding Media: The Extensions of Man*. The MIT Press, 1994 [reprint from 1964].

[10] Metsker, S. J. *Design Patterns in C#*. Addison-Wesley, 2004.

[11] Mungai, D., Jones, D., and Wong, L. "Games to Teach By." *Proceedings of the 18th Annual Conference on Distance Teaching and Learning* (Madison, Wisconson, 2002).

[12] Shalloway, A., Trott, J. *Design Patterns Explained: A New Perspective on Object-Oriented Design, 2nd Ed*. Addison-Wesley, 2004.

[13] Wick, M. R. "Teaching Design Patterns in CS1: a Closed Laboratory Sequence based on the Game of Life." *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, 37, 1 (February 2005).

Creativity in the Cane Fields: Motivating and Engaging IT Students Through Games

Colin Lemmon, Nicola J. Bidwell, Marion Hooper,
Chris Gaskett, Jason Holdsworth, Phillip Musumeci

James Cook University, Cairns Campus
McGregor Rd. Smithfield
Queensland, Australia
+61 7 40421233

colin.lemmon@jcu.edu.au

ABSTRACT
In this paper we discuss the influence of the unique local environment and culture on students and teaching styles in the IT degree at James Cook University Cairns Campus. In this degree program games are used to motivate self-directed study and increase student engagement in first and second year programming subjects, and also to generate interest in learning new technologies such as programming for mobile devices. We discuss the use of a mixed reality location based game to improve attitude to teamwork by integrating students in a games subject and a general IT software engineering subject. Students learn the value of community engagement through links to a local primary school for design and evaluation of games, to ensure a balanced approach to user requirements, game design and implementation. Students have explored niche applications of games through the development of a game for children with disabilities.

Categories and Subject Descriptors
K.8.0 [**Personal Computing**]: General - Games, K.3 [**Computers And Education**]: Computer Uses in Education - *Collaborative learning*, H5.2 [**Information Interfaces And Presentation**]: User Interfaces - *Prototyping, Theory and methods, User-centered design*

General Terms
Management, Design, Human Factors, Theory.

Keywords
Games, Community, Collaboration, User interaction.

1. INTRODUCTION
The Cairns campus of James Cook University in Far North Queensland, Australia, is 200 miles from the main campus and 1000 miles from the closest major city. The economy is built on traditional industries such as agriculture and mining, and more recently industries such as tourism and creative industries. Cairns hosts a diverse range of cultures—traditional sugar cane and dairy farmers, those seeking an alternate lifestyle, indigenous inhabitants, and an artistic community attracted by the reef and the rainforest.

The regional characteristics produce a unique student profile. As the only university to run courses locally the university has a social responsibility to service the needs of all types of students. This results in classes containing students with a broad range of abilities, generally resulting in a bimodal distribution of academic achievement in first year IT subjects. The student base presents considerable challenges when compared to universities in large cities with entry requirements to target students with high academic achievement. Thus, subject delivery and teaching style must be adapted to get the best out of each student, encouraging them to higher levels of commitment and achievement so that the academic integrity of the degree program is not compromised.

First year students include those with varied attitudes towards self-directed study and commitment to academic achievement. This commonly arises from lack of familiarity with the larger world outside of the local community. At universities in larger cities the draw of high salaries and the understanding of the levels of expertise required for these positions is a significant motivational factor for students. Possibly because of their limited knowledge of the rewards of top level IT positions in larger companies in capital cities and overseas, our beginning IT students are often lured away from study by jobs that require only limited expertise. In response we have developed techniques to motivate our first year programming students until they develop an understanding of the rewards for high achievement and the relevance of the material being taught; and the commitment required for self-directed study.

There are other social issues that effect motivation, commitment and performance in isolated regions. The Cairns campus of James Cook University is a little over 10 years old. There is little family history of university education and many students (>50%) are the first in their family to attend university. Their families have little or no understanding of what is required to obtain a university degree, especially the time commitment needed.

Games can provide motivation for self-directed study and increase student engagement, especially in first year programming subjects. In later subjects gaming is employed to generate additional interest when teaching technical principles. In the group projects the shared interest and experience of students in games is employed to facilitate group cohesion and commitment to the project. In addition game projects provide an avenue for development of social responsibility through projects, for example, developing games for children with disabilities.

It has been said that Generation Y, or the Millennials, have a greater focus on self satisfaction than preceding generations. Whether or not this is true, there is no denying that university education is currently competing with entertainment for the attention of students. Entertainment for Generation Y and later is dominated by gaming on devices from phones to networks of desktop PCs. Games as part of a teaching and learning strategy provide an effective tool for focusing student attention on learning.

2. BACKGROUND

James Cook University in Cairns offers IT degrees with a range of strands including eBusiness, IT Professional, Networking and Multimedia Games Development.

The development of the Multimedia Games Development strand was driven a number of years ago by a recognition of the strength and importance of creative industries in the local community and economy, and in response to the demand from potential students in year 12 that was identified by local high school promotional talks and university open days. Providing a degree based on the general interests and activities of high school students helps promote the opportunity and interest in future tertiary education.

The core programming languages in the general computing subjects include C/C++, Java, Python, Perl, and PHP. The games strand also includes training in Maya and the Torque games engine.

Game subjects allow introducing and developing complex human-centered design approaches to an otherwise technically focused degree. In three subjects running in second and third year, students tackle and extend themes frequently addressed in traditional human-computer-interaction (HCI). The difficulties often encountered in motivating technically focused students to place the user at the centre of their system design do not seem to be a problem when students are able to relate this to the centrality of the player experience to the success of gameplay. For example, one subject explores games from multiple perspectives of rules, play and culture and uses Salen and Zimmermann's opus [1] as the core text. The students rise to the challenge of tackling the advanced ideas from semiotics to sociology. Human-centred design principles relating to user/player experience and usability can be introduced across multiple contexts and tackle multiple issues. It is quite difficult to imagine how we could extract the same extensive debate about emotions from a large group of young people (mostly men) in other contexts! Play gives all people a safe place for self-expression: which we believe is at the heart of innovative, human-sensitive design.

The following sections describe the place of games in our course structure chronologically, from first year undergraduate study to graduate research.

3. GAMES AS A TOOL FOR INTRODUCING PROJECT DEVELOPMENT TECHNIQUES

In the first year multimedia subject which is taken by both IT and non-IT students are required to develop a multimedia project that includes interactive elements. Multimedia projects are developed using the Adobe/Macromedia Director MX 2004 environment. Students are encouraged to use the built-in Director features as well as writing their own scripts using the Lingo scripting language. The interactive elements required include real time text entry, data storage and access, keyboard and mouse control. The incorporation of these elements into a game-like artifact is a natural extension of the students own experiences as game players and also allows them to experience the favourable response of others to an interactive multimedia experience that they have created. For many of these students this is their first experience of computer programming and the use of game-like examples during practical and tutorial sessions provides an effective motivation for them to struggle with the complex task of getting their own game to work as planned. During the development process many are also inspired by the responses of their beta testers to correct faults in their game design and add elaborations to enhance the game's appeal to the end user. In this way students gain first hand experience of the need for testing and refinement during the project design and development process in a context where the main driver is their own motivation to create something pleasing to the end user. The resulting projects show a very high level of development in the students' programming and project development skills in a very short period of time and it is evident that the experiences gained in this process inform their later work in other games and project development subjects.

4. GAMES AS A MOTIVATIONAL TOOL IN FIRST YEAR NON-GAMING PROGRAMMING SUBJECTS

In the first year first semester procedural programming subject (based on C/C++) the assignment completion rate is improved by prescribing the development of a game for the first programming assignment. Over the past four years the assignments have been a cricket game, a tennis game, a number guessing game and a golf game. The first assignment is a critical piece of assessment as it establishes the students' confidence in their ability and their subsequent attitude to programming. Good academic performance in the first programming subject is essential as students who perform poorly in this subject have significant problems recovering from a lack of basic programming for the remainder of their degree.

The use of a games based assignment has been introduced into the subsequent second semester C/C++ object oriented programming subject to ease the transition between procedural programming and object oriented programming. Last year the first assignment was an object oriented, text mode adventure game which introduced students to object oriented programming in an enjoyable and motivational way. This resulted in improved commitment to assignment completion.

Games are also used in the second year Java programming subject. Here games are used to teach and motivate students to learn graphical and GUI programming. Games used in this subject in the last few years have been GUI controlled single player games, including a game of Pool, the Snake game, and the Game of Life which used a mobile phone emulator.

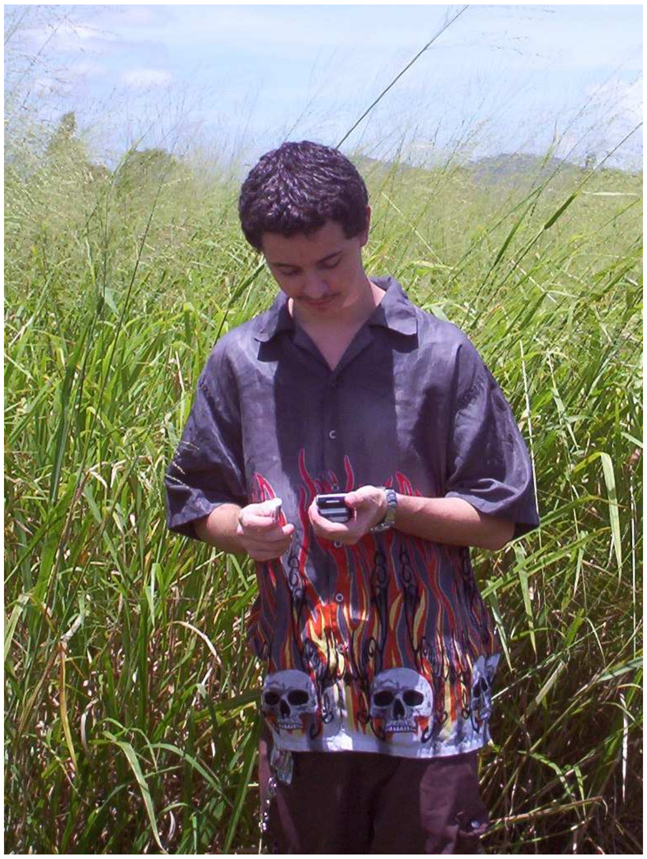


Figure 1. Gameplay design of mixed reality location based game using mobile devices

5. GAMES AS A TOOL TO IMPROVE ATTITUDES TO COLLABORATION

Games are used extensively in Java and software engineering subjects to generate interest in learning new technologies. They also provide a good way to illustrate the relationship between design and development processes and different communication forms. Recently, an innovative mixed reality *location based game* (LBG) has been used to integrate a project into the work of students in two subjects. Games students design a location-based multi player game and the IT students implement the game. The activities are summarized in Table 1.

The first step in the process involves the games students investigating the constraints of the world and the technology using smartphones and GPS in situ and bodystorming game ideas. They then iteratively refine a game concept by lo-fi prototyping and evaluating it using the Wizard of Oz protocol before presenting the game concept as a set of rules to the students in the Games strand [2].

The IT students take the game design and develop a distributed games engine to implement the game. This process of games implementation teaches the IT students concurrency, synchronization, socket programming, wireless networking, GPS and midlets using Bluetooth to enable cell phones with GPS capabilities.

The IT students are required to develop the games engine and system framework including login, interface and game framework. This framework is then used to implement the mixed reality LBG. The current game is a version of battleship in Java using the Bluetooth enabled location aware mobile phones. The code is developed firstly using midlet emulation and execution then later ported to physical devices where the games are played between cell phones as illustrated in Figure 1.

The use of games technology to teach programming on mobile devices maintains interest and motivation for group members who do not choose programming as the first priority in their studies. This allows them to benefit from being able to relate project work to real world applications in an interesting way. Having input into the project increases their tolerance to the frustration inherent in learning how to program.

Table 1. LBG activities in cooperative game development between IT students and Game students

2nd Year Games Subject	3rd Year IT Subject
Observing the basic gameplay of paper-based battleship	Speculative project development, and generation of inception artifacts, of a basic Non-LBG Battleship game for a mobile phone
Bodystorming and capturing LBG game ideas using Smartphones and GPS in situ	
Visualisation and abstraction of LBG concepts	Elaboration iteration based on technical development of
Evaluating lo-fidelity LBG prototypes using Wizard of Oz	Non-LBG Battleship game on phones
Refining and communicating LBG design concept	Programming Java Midlet & Servlet Connectivity
	Major change to project requirements by incorporating LBG concepts into the Non-LBG mobile game
	Demonstration of Client-side Java Bluetooth programming
	Elaboration Iteration 3 & 4
Play and evaluate LBG software components	

Using a real world multiplayer platform which can be run on real phones further enhances student interest and participation in the project. Object-oriented software design techniques allow students to adapt to changing requirements, concentrating on the inception and elaboration phases of agile UP. This engenders a specific kind of procedural reasoning and systematic thinking about large-scale software problems using UML diagrams, and software design patterns. In previous offerings of the subject students developed a non-location based game on mobile phones which engaged their interest in applying patterns for system design and specification. The LBG project addresses the changing requirements aspect of the subject's learning objectives and industry's need for graduates to be responsive to dynamic and temporally evolving contexts. Thus, the subject links theoretical perspectives directly to the project as a concrete framework for students to clearly and concisely express software specification design ideas. This is compatible with requirements of Industry and professional organisations (e.g. Australian Computer Society) that graduates be able to design effectively for an increasing breadth of technologies. The project also enabled students to practice working with cross-functional teams.

A noticeable benefit of designing learning activities compatible with students own game experience is their frequently improved attitude to teamwork. Experienced lecturers remark that students who have extensive Massively Multiplayer Online Game experience are better able to recognize the importance of all individuals to a team and that good team outcomes result from combining individual skills and strengths rather than indulging in 'lone ranger' behaviour.

6. GAMES & COMMUNITY

Links to a primary school provide second year students in the games directed project subject with an ideal opportunity to design for people apart from themselves. In this way teaching of games technologies is firmly linked to the critical importance of player-centred game design. Students learn and practice the skills needed to develop prototypes using the Torque game engine in a series of 11 structured practical workshops which incorporate 3D modeling tools. Focus on the player is developed early in the project and maintained throughout the project so that the technical skills of the Torque game engine are integrated with powerful design techniques based on the Fullerton and Swain textbook [3]. Students brainstorm, create a physical (paper-based) prototype, develop and evaluate concepts, prepare preliminary design documents, and then prototype and test software with the 11 year old children who are the target audience. Students visit the primary school every 6 weeks, firstly to evaluate a large number of physical prototypes (see Figure 2) and then to test a smaller number of refined software prototypes. In the process students encounter implementation issues fully embedded in the design process and are motivated to create an innovative, internally complete and balanced game. This approach discourages a view of games design and development in which the player becomes almost a subsidiary issue.

Changes in student behaviour and attitudes as they evolve their group prototypes in games directed project subject strongly suggest that our human-centred approach to teaching IT supports both a sense of community and a more effective approach to design. The degree to which students demonstrate a central concern for gameplay, playability and the fun of their games for junior play-testers is palpable. For example, an interesting aspect of links to the primary school through the Directed project this year was the significant environmental and socio-economic impact on the community of Cyclone Larry, a category 5 cyclone. The catchment area for the target primary school is closer to the centre of destruction than the university and the students' ideas for games with elements compatible with the children's potentially unsettling experience was quite heart-warming.

Links to past students who work in school IT departments or as IT teachers in local schools has provided the opportunity for these classes to become involved in the evaluation of beta versions of games developed by students in the games strand. This provides the school students with an understanding of the game

Figure 2. Physical prototyping game concept ideas with school children

development process (design as opposed to merely coding), a practical example of what they may learn at university and also allows them to talk personally with current students and ask questions regarding the possibility of enrolling in an IT/Games degree in an informal and non-threatening atmosphere. School-based testing of beta versions of games provides valuable knowledge about human behaviour and the characteristics of their target audience for the games students. The feedback received in the testing process also highlights the importance of the design process and develops a deeper understanding of human computer interface issues. Students also learn the importance of an iterative design process.

It is important that students completing the degree are able to apply their knowledge and skills to obtain employment and fulfill the needs of the community. Apart from the obvious mass market for games we encourage students to understand the wider applications of their skills to niche game markets and non-game markets.

The third year project subjects require that students develop a software system for a real client over a year. The project subjects have been highly successful and received a national citation from the Carrick Institute for outstanding contributions to student learning in higher education.

This year three games students completing the project subjects worked as a team to develop a game for children with disabilities. Working with the *Switched on Learning Project*[1] gave the students access to a niche market and required them to deal with unusual requirements: The game players interact with the game through two large switches rather than a mouse, keyboard, or joystick. The switches control the actions of the player in a 3D simulation of a shopping centre. Tasks include finding appropriate shops to purchase items from a shopping list, interacting with shop staff, and making a purchasing including receiving change.

Beyond the practical activities described above, a major point of the game is to explore independence. The children that play the game have restricted mobility and few opportunities to choose

[1] http://www.switchedonlearning.org/

their own activities. Even in the early stages of development when the game included only one activity and so many bugs that it seemed hardly playable the children were overjoyed with it. Although they had often seen their siblings play 3D games with high quality sound, they had never had the opportunity to play one.

Fulfilling a real client's needs or wants is a significant step in the transition from student to practitioner, and a highly rewarding experience for our students.

7. GAMES AS AN INTRODUCTION TO RESEARCH

As a consequence of student catchment the option of pursuing post-graduate studies is not obvious to students. Now deemed to deserve both serious academic research, as well as major industry and business uptake, techniques used in computer entertainment translate well into advances in research and enable us to leverage student interest in the processes of research. For example, several students' observations of multi-player behaviour in one assignment fed into a paper on the efficacy of different visualisation forms for wayfinding large-scale simulated worlds [4]. Second and third year student development activities for mixed reality LBG also formed the basis of another paper [2] in which students were eager to be involved. This year an Honours student's project was based on games. An interesting aspect of games as a research area is that students will so willingly adopt good research practices in order to pursue a pet theory they have developed in their own gameplay. It appears that because proving or disproving their theory has so much relevance to their gameplay and, for some students, game design, they will engage in protracted and detailed observation sessions and workshops to explore and articulate their experiences and develop sound theories [5].

8. CREATIVE INDUSTRIES

Growth in the creative and technology industries of remote Australia requires cultivating an entrepreneurial spirit amongst students in local universities to inspire them to develop their own enterprises. We endeavour to encourage views of technology futures that are sensitive to our locality; while also exposing students to technologies that are encountered in large or metropolitan universities but not, contemporarily, in our region.

In the games strand students are encouraged to draw upon our unique natural environment and locational context to inspire their game designs. For example, in one second year subject they have to consider design constraints created by the mountains and rainforest surrounding our campus for their mixed reality LBG. This provides a direct and engaging way for students to attend to the environmental demands and constraints associated with pervasive computing. In a third year subject one assignment requires them to create visual and audio representations that capture specific qualities of nature and use these to convey particular emotions and themes. Through immersion in their environment students learn to realize the special opportunities for creativity afforded by our visually spectacular scenery and the advantage this may afford local creative industries.

The ethos of creative volunteer work associated with game culture seems to instill students with a particular sense of entrepreneurial spirit. As well as contributing to the community through multimedia presentations and games created for volunteer groups several students currently or previously involved in games subjects have created their own local businesses, for example, www.redbackgames.com.

9. CONCLUSION

In addition to motivating and engaging students to learn to program, game design helps stimulate students to develop human centered design skills. Through community engagement with school children, students develop games using an iterative development process resulting in an appreciation of the balance between the needs of the user and the requirements for coding. Students are also encouraged to learn technical skills through development of location based games on mobile devices and to apply gaming skills to niche applications as well as the mainstream games market.

10. ACKNOWLEDGMENTS

We wish to acknowledge the cooperation of staff and students of Woree State School and the Switched on Learning project, and JCU Bachelor of IT students.

11. REFERENCES

[1] Salen, K., and Zimmermann, E. *Rules of Play. Game Design Fundamentals*, MIT Press, MA, 2004.

[2] Bidwell, N. J., and Holdsworth, J. *Battleship by Foot: Learning by Designing a Mixed Reality Game*. Joint International Conference on CyberGames & Interactive Entertainment (CGIE06), Perth 2006.

[3] Fullerton, T., and Swain, C. *Game Design Workshop: Designing Prototyping & Playtesting Games*, CMP Books, 2004.

[4] Bidwell, N. J., Poyner, D., Irving, J., Putna, L., and Wold, A. *"Make it through with another point of view": Landmarks to Wayfind in Gameworlds*. Australasian Conference on Interactive Entertainment (IE05), Sydney 2005.

[5] Browning, D., Stanley, S., Friar, M., and Bidwell, N. J. *Emplacing Experiencee.* Joint International Conference on CyberGames & Interactive Entertainment (CGIE06), Perth 2006.

A Soft Approach to Computer Science: Designing & Developing Computer Games for and with Senior Citizens

Vero Vanden Abeele, Jelle Husson, Luc Vandeurzen, Stef Desmet
e-Medialab

Group T – Leuven Engineering School
Vesaliusstraat 13, B-3000 Belgium
+ 32 (0)16 30 10 30

vero@groept.be, jelle.husson@groept.be, luc.vandeurzen@groept.be, stef.desmet@groept.be

ABSTRACT

This paper introduces a soft approach to programming and especially game development, based on the inclusion of the end-user (or player) in the development process and on the rigorous use of a scaled-down version of the Unified Process: software development is more than only coding.

Students were assigned to develop a computer game for and *with* senior citizens. The project started off by observing senior citizens in their 'natural habitat', researching what 'passions' occur in their daily lives. These observations then became the input for brainstorm sessions. Seniors and students generated game-ideas and, consequently, co-designed the selected ideas into game concepts. Based on a voting by the senior council, one game concept was chosen to be developed: 'Petanque', a multi-player *jeux de boule* game.

During the actual development of this game, seniors continued to provide inspiration. But besides this user-centred approach to game development, other 'soft' skills were addressed as well such as software engineering, team dynamics and project management. During five weeks, the students worked full-time, in team, on the game creation, taking on different roles. We encouraged using existing game engines and development platforms to ensure that within this limited amount of time a playable demo could emerge. This demo was finally presented to, played by and tested by a senior audience.

This user centred, team-oriented project resulted in an inspiring computer game, directly grafted on the passions and desires of the senior. Students gained insight in and empathy for their player audience and became aware of the diverse management and 'soft' skills that are necessary in the creation of successful software applications.

Categories and Subject Descriptors

H.5.2 User Interfaces - User-Centred Design
K.6.3 Software Management, Software development, Software process

General Terms

Design, Human Factors, Management

Keywords

Seniors, Elderly, Game Development, Game Design, Co-design, Participatory Design, Human-Centred, User-Centred Design, Software Engineering, Unified Process.

INTRODUCTION

1.1 The master in e-Media

The master in e-Media engineering is a graduate program for students that have a bachelor in computer science or similar. The program specializes in the development of rich media application and has a special focus on higher level programming skills, software engineering, computer graphics, digital media and human-computer interaction. Recently we saw a rising interest with our students in game development. Therefore the program started to incorporate games as a specific topic, e.g. a course on user-centred design was shifted to player-centred design. But more fundamentally, we oriented the subject of our 'multidisciplinary project' to game development[1]. This project is a five-week, full-time course in an attempt to mimic a more real world job situation. Students work together in team, taking on different roles and managing the development process. In this multidisciplinary project, we follow a Unified Process to software development, together with a user-centred approach. This hands-on experience helps our students in their future career.

1.2 The I-Methodology

Students in computer science and thus game development courses prefer to tap into their (self-perceived) unlimited creativity and come up with their own ideas [8,6], referred to as the I-methodology [5]. This self-centred design process often results in 'boys' designing for 'boys' [2] or to put it in other words hard-core gamers designing for hard-core gamers. With this project on game development, however, we did not only want to address the need for development and management skills. As the computer (game) sector is looking to gain maturity and a larger target group [1,10] with this course we also wanted to address the need for a more inclusive, more mature approach to user-interaction.

Combining the I-methodology with the fact that computer science programs (and consequently the game development industry) are mainly populated by (young) males, the difficulties are obvious when trying to develop games aimed at a wider audience. As a

[1] The years before there was a stronger focus on distributed web applications and augmented reality applications.

result, a widespread critique is that the game industry has difficulties in addressing non-traditional player groups [7]. Because of this consideration we choose to develop games for a senior audience. This audience made it necessary for our students to do research away from their desks and clearly necessitated a user-centred design process.

Furthermore it has been demonstrated that linking computer science to societal merits and giving programming a more humane face can increase the attractiveness to female students. [4]. Thus, by choosing this senior audience we also had the noble goal of closing the digital divide, and increasing cooperation between older and younger generations. And we foster the hope to create a stronger appeal to female programmers.

PLAYER-CENTRED GAME DEVELOPMENT FOR AND WITH SENIORS

We deemed a player-centred approach necessary to innovate gameplay for a senior audience. Therefore, we conceived a project in which students developed games for and with senior citizens. The project activities encompassed five different phases, distributed over a course on player-centred game design and a multidisciplinary project on game development. We started out with an ethnographic inquiry of senior citizens[2]. Consequently, in the second phase seniors and researchers brainstormed for ideas and converted selected ideas into game concepts. During the third phase, the game concepts were presented to a broader audience of seniors and one game concept was selected. This game concept was developed, in the fourth phase. During this phase we used a Unified Process (UP) approach [1] to develop a playable demo in only five weeks. In the fifth and final phase the game was played and evaluated by the seniors. We now discuss the different phases and clarify the steps with some illustrations.

Phase 1. Ethnographic observations

We started out by conducting ethnographic observations. During the time span of one week, seniors were observed, interviewed and ‘probed’ by our students at their homes. Seniors were asked to record all ‘enjoyable activities’ or passions. It was stressed that a passion is something that makes the time fly, but really can be anything. Seniors were asked to write down all passions on post-it notes and stick these notes in a passion logbook and take photographs of any artifacts, surroundings or people related to these passions. If possible ‘show & tells’ of the passions were asked for. The students directly analyzed different factors that were important for a better understanding of these passions: What is the nature of the passion? What exactly makes it enjoyable? How is that passion situated in time and space? Are other people involved? Is there technology that facilitates the passion? Finally students asked the senior to create a top five of the most important passions to him or her.

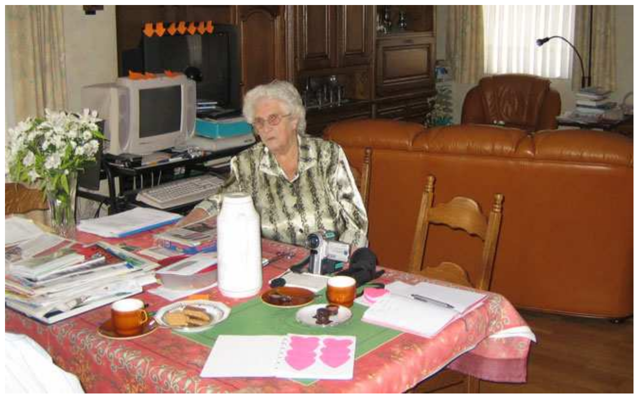


Figure 1: A picture of a senior at her home. Notice the orange post it notes on the television and the computer, as well as the heart-shaped notes in the logbook on the table.

In total, this not only gave a list of 50 passions[3] in elderly life, but also ensured that students gained insight in and empathy for the lives of senior citizens.

Phase 2. Participatory design

Approximately one month after the ethnographic observations we started with the participatory design sessions. For this phase, we constructed design teams consisting of one student and one senior citizen. A social scientist and an interaction designer were present to moderate and facilitate the design processes.

Seniors and students first brainstormed for possible ideas inspired by a small contextual story. In total 399 ideas[3] were generated. Not surprisingly, many of the passions that were listed in the top five during the ethnographic inquiries also ended up as ideas on the wall during this brainstorm. After the idea generation phase, the teams evaluated them on their attractiveness. In the end, each team chose one idea to elaborate upon. This idea was then co-designed into a game concept.

Design teams were also encouraged to create paper prototypes and visualize their vision. For each of the 10 teams, the end result of this participatory design process was a rough ‘game design (concept) document’ and if possible a paper prototype.

Figure 2: A brainstorm session with students and a senior.

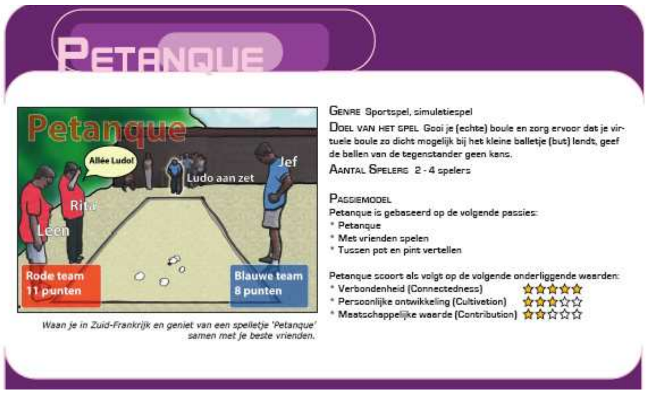
[2] Ten senior citizens (seven male and three female) participated in the research project. The age varied from sixty-eight to eighty years. All seniors were in good health, living independently.

[3] For a detailed overview of the passions, ideas and game concepts we like to refer to the website at http://sbox.groept.be

Figure 3: Students and seniors are co-designing a game-concept and constructing a paper prototypes.

Phase 3. Presentation and selection of the Game Concepts

From the ethnographic observations, we knew that seniors spend a lot of time on activities such as playing cards, solving puzzles, watching television, etc. But when listing a top five of passions, these activities fell short and did not show up. Neither did these activities make it into the brainstormed ideas and game concepts. Instead, game concepts were (although unpolished) surprisingly rich. Most games had a strong multi-player component and mixed several game genres, such as adventure, quizzes and role-playing games. Most games offered the possibility to enrich knowledge, such as cultural or travel quizzes, and many game concepts could also attract a non-senior audience. This raw material was turned into eight attractive game concepts[4] by the interaction designer. (two game concepts were too much alike and converted into one, one game concept was still too vague after phase 2)


PETANQUE
Petanque
Allée Ludo!
Ludo aan zet
Jef
Rita
Leen
Rode team
11 punten
Blauwe team
8 punten

Waan je in Zuid-Frankrijk en geniet van een spelletje 'Petanque' samen met je beste vrienden.

GENRE Sportspel, simulatiespel

DOEL VAN HET SPEL Gooi je (echte) boule en zorg ervoor dat je virtuele boule zo dicht mogelijk bij het kleine balletje (but) landt, geef de bellen van de tegenstander geen kans.

AANTAL SPELERS 2 - 4 spelers

PASSIEMODEL
Petanque is gebaseerd op de volgende passies:
* Petanque
* Met vrienden spelen
* Tussen pot en pint vertellen

Petanque scoort als volgt op de volgende onderliggende waarden:
* Verbondenheid (Connectedness) ★★★★★
* Persoonlijke ontwikkeling (Cultivation) ★★★☆☆
* Maatschappelijke waarde (Contribution) ★★☆☆☆

Figure 4: Petanque, a multi-player, jeux de boules games was chosen to be developed.

During a bimonthly meeting of the senior city council, the game concepts were presented on posters and seniors were asked to bring out there vote. Unfortunately, there was no clear indication or consensus about one winning concept. Therefore, upon practical considerations, the Petanque game was chosen to be developed further.

[4] For an overview of the game concepts, posters, the methodology and results we like to refer to http://sbox.groept.be.

Phase 4. Development of 'Petanque'

The actual development of the Petanque game started during a five-week full-time course in which a team of 6 students contributed to the development. In a multi-disciplinary, team-oriented way students developed the actual game. A strong focus was on software engineering, using a iterative approach (UP) to carry the project to a successful end. We want to stress that although the three previous phased lead to a inspiring game concept, requirements were still uncertain or even unknown. Also students had no substantial experience with the tools or the application domain.

Of course we did not aim for a big-budget game within the reality of an educational setting; however the multi-disciplinary project had enough complexity to define clear roles for each student. We distinguished between a project leader, a software architect, a lead programmer, a character artist, an environment artist and a 2D interface artist. Student's responsibilities however were not limited to these predefined roles; each student was expected to work on different aspects if necessary. Following the UP-approach, the project went through the phases of Inception, Elaboration, and Construction.

1.2.1 Inception

During Inception, team roles were assigned, along with some process management agreements. Students made decisions on the development tools and the environment used to create the game. Because of the short development time students were encouraged to use tools and environments that would support ultra rapid development. As an environment, the team chose to investigate the Virtools Framework [11]. For content creation the team chose to use a combination of e-frontier Poser, Autodesk 3ds Max, Adobe Photoshop and standard sound editors.

After the decisions on tools and frameworks were made, the original Petanque idea was revised. Although the game concept was already defined during the second phase, students were encouraged to do some more exploration with seniors and make adaptations to the game concept if needed. Getting a deeper understanding of Petanque was certainly necessary, since not all international students were aware of the characteristics of this typical activity for Belgian people. After some real-life games and discussions with seniors, all students got a better understanding of

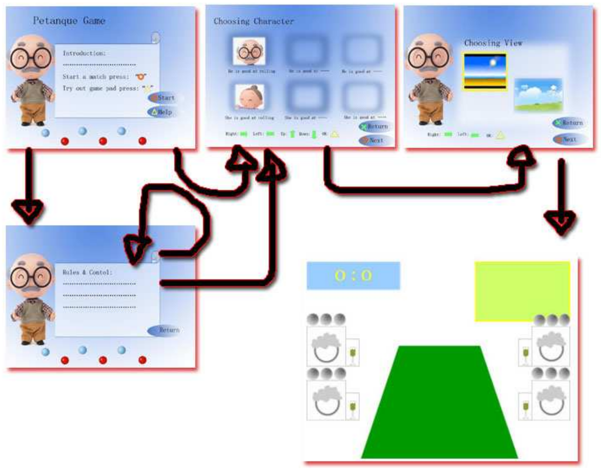


Figure 5: A first draft of the structure and style of 'Petanque'.

the rules and the joy of playing the game. The team created a new game concept document in which they listed the key elements of their game as well as a first draft of the structure and style of the game.

Students gave the following (abbreviated) description of the concepts: *Petanque is a simulation – sports game, with 2 players, and intuitive, consistent controls. We focus not on the competitive, but on "feel good" and social interaction between players. There is a Practice Mode and Play Mode. There are always visual and auditory explanations of functionalities. There is a possibility to compose a team of unique characters with individual skills. There should be a consistent style in 2D interface and 3D content. We should create immersive 3D characters the player could relate to. Finally, we need matching music and sound effects.*

1.2.2 Elaboration

Next in line was the Elaboration phase. Students worked out their ideas in a more detailed concept document. In this and the previous phase the focus was not only on programming skills (all students had a background in computer science) but also on the diversity of skills that are necessary in game design & development. Students had to keep in mind that they were developing for senior gamers, which implicated certain choices for the development of the user interface such as large fonts, auditory explanations, etc.

Although the students were not unfamiliar with the tools they had chosen, they certainly were no experts. Previous projects were quite small in numbers and complexity and so the team's experience with the tools was rather poor. Since a good plan was needed for knowing how to tackle this project, the team scheduled a short period of technical scouting with the tools, keeping the ultimate goal and their individual sub-goals in mind. Especially the Virtools Framework was something they needed to learn more about in order to create a piece of software with the necessary complexity.

As soon as the students had a precise idea of what to do and how it could be done with the chosen tools, the architecture was designed and a basic framework was implemented, allowing them to have a working demonstrator at all times.

Near the end of the elaboration phase, the students had an intermediate review of their project, together with a professional game designer (Swen Vincke, CEO Larian Studios).

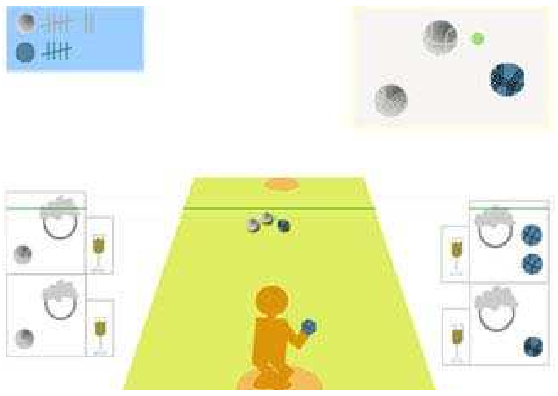


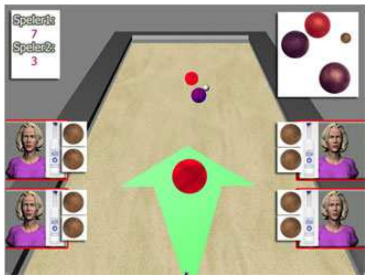
Figure 6: The throwing mechanism.

An important lesson they learned there was to keep an eye on the overall style of the game because different people were working in parallel on different parts of the project. Swen Vincke stressed that content creation often takes up more resources than the actual development and that there was a strong need for style guides when outsourcing this work. When analyzing the student's game, the lack of style guides was showing in inconsistencies within the visual style and certain gameplay components.

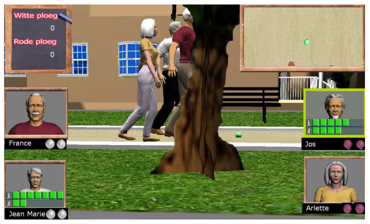
The team was also told to focus first on the most critical components of the game mechanisms. For this game this involved the throwing mechanism of the balls. First the team needed to implement a satisfying way of throwing the balls before elaborating other parts of the gameplay.

1.2.3 Construction

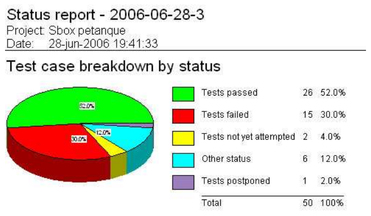
Filling in the initial framework with content and game rules happened in iterations, which continued in the Construction Phase, alongside testing.



The construction phase turned out to consume most of the time that was available for the project (and more than was originally planned). Along the road, many nice ideas and cool additions popped up. To keep the project manageable, decisions were made about which features should be considered *top priority* and which features would be *nice to have*.



One particular example of a nice-to-have idea was the implementation of a sub game where the players could drink Pastis (an alcoholic beverage, associated with playing the game in real life). Drinking Pastis would influence the throwing skills of the players. This feature did not make it into the final version of the game; neither did any of the other nice-to-have features. Because the team had no substantial technical experience, most implementation parts were underestimated and in the end the realization got delayed.



🟩	Tests passed	26	52.0%
🟥	Tests failed	15	30.0%
🟨	Tests not yet attempted	2	4.0%
🟦	Other status	6	12.0%
🟪	Tests postponed	1	2.0%
	Total	50	100%

Testing and debugging took more time than first thought and often repairing one bug resulted in other problems arising elsewhere. An accumulation of these small delays resulted in the project going over its schedule. Fortunately, the students did implement a safety net by setting their personal deadline largely in advance of their ultimate presentation deadline. Furthermore, it has to be said that some motivated students put in more than the 200 hours (or five-weeks full-time) that were originally scheduled.

Part of the management process required students to keep track of the time they spend on the project. We found that for some students a better estimation of the actual workload was about 280 hours or an equivalent of seven weeks of full-time working. In the end, because of the safety net, the prioritization of components, and the good-will of some of the students, the team managed to present a fully working prototype.

Phase 5. Evaluating the game

In a final stage, the Petanque game was demonstrated to and played by one senior, who gave a overall positive evaluation. In the overall context of the project, the team stressed that they would have definitely liked to have tested and evaluated the game more profoundly and with a larger senior audience but in the scope of this course “multidisciplinary project” there simply was no time left over.

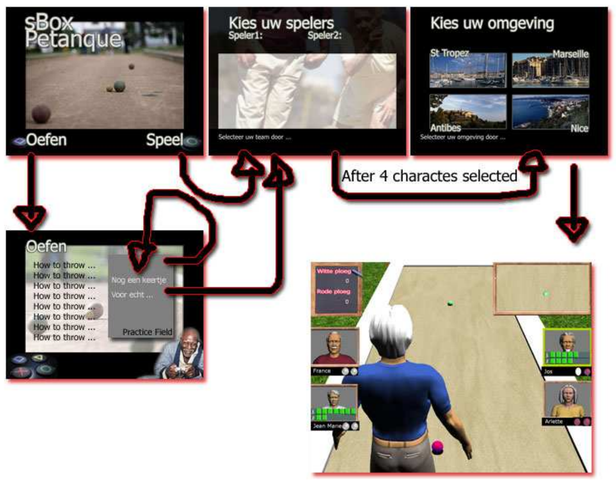
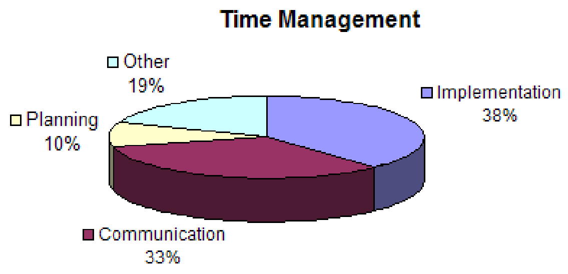


Figure 7: Playing the final game!

Students also evaluated their process and gave the following remarks. From the beginning they felt that the clearly defined roles and tasks for everyone were an advantage that allowed for parallel working schemes. Reviewing the work of every team member and formally documenting every step in the development process felt initially somewhat as ‘overhead’. But after the first iteration, templates existed for all documents and they learned that reviewing improved the overall quality of everybody’s work. The iterative software development approach forced them to first develop a core system demonstrating all critical features that made up the ‘heart’ of the game. Each additional iteration resulted in a new executable system that improved the previous one in quality and number of features. They realized very quickly that this process was a safe approach to bring a project with so many uncertainties to a good end. They also experienced that defining the activities per iteration is a dynamic and non-trivial activity in itself since the option list seemed to become longer and longer: new requirements, always more and more ‘nice-to-haves’, bug fixes, ... Near the end of the project, they worked with a task list sorted on criteria as ‘task priority’ and ‘task workload’ to make the best decisions in defining the next iteration activities. Clearly, it didn’t make sense to work on low priority tasks with a high workload when almost no time was left over.

From the beginning of the project, the students kept track of all time spent on every project activity. After processing these time sheets, they clearly saw that ‘coding’ was only a small part of the project.



CONCLUSIONS

The choice of computer games for seniors was a valuable exercise to learn students about the intricacies of programming on a large-scale in a multidisciplinary environment and about the importance of a user-centred process. The ethnographic observations and co-design sessions provided a valuable input for game ideas and led to creative and non-stereotypical game concepts. Students gained insight in and empathy for their player audience.

The Unified Process approach to game development helped students to organize and prioritize their time and work, which is critical to the complex process of game development. The Virtools framework along with other rapid prototyping tools helped students to deliver a working prototype within a limited amount of time. Also it helped students to focus on higher level programming skills. Students became aware of the diverse and versatile aspects that are part in the creation of computer games.

But the most important skill they learned was to work together, to manage and to communicate so that in the end all the pieces would fit the puzzle.

The results of this approach resulted not only in an inspiring computer game, directly grafted on the passions and desires of the senior. Students also gained insight the ‘soft’ skills that are necessary in the creation of computer games. And most important, they had fun!

ACKNOWLEDGEMENTS

A special thanks goes to all students of e-Media, 2005-2006 contributing to this project. Furthermore we are very grateful to the senior city council of Louvain and especially, Jerome Verhoelst, the president, who made this project possible. Finally we like to thank Swen Vincke for his useful remarks to this project.

REFERENCES

[1] Ermi, L. & Mäyrä, F.(2004) *Player-Centred Game Design: Experiences in Using Scenario* Study to Inform Mobile Game Design, Game Design Research Symposium and Workshop, Copenhagen

[2] Gansmo, H., Nordli, H. & Sorensen K. (2003). *The Gender Game. A study of Norwegian Game Designers*. SIGIS.

[3] Larman, C. *Applying UML and Patterns – An Introduction to Object-Oriented Analysis and Design and Iterative Development.* Pearson Education, 2005.

[4] Margolis, J. and Fisher, A. (2002). *Unlocking the Clubhouse: Women in Computing.* Cambridge: MIT Press.

[5] Oudshoorn, N.E.J. & T.J.Pinch. (eds) (2003) *How Users Matter. The Co-construction of Users and Technology*. Massachusetts: MIT Press

[6] Oxland, K. (2004). *Gameplay and Design*. Addison Wesley. p. 24-43.

[7] Ray, S.G. (2004). *Gender Inclusive Game Design: Expanding the Market*. Charles River Media. USA.

[8] Rouse, R. (2005). *Game Design: Theory and Practice,* 2nd edition, Wordware, Texas

[9] Sannella, M. J. *Constraint Satisfaction and Debugging for Interactive User Interfaces.* Ph.D. Thesis, University of Washington, Seattle, WA, 1994.

[10] Sykes J. & Patterson A. (2004) *Considering the User in Video Game Design*, Workshop on Player-centred Design. ACMCHI 2006

[11] Virtools. http://www.virtools.com

Middle-to-High School Girls as Game Designers – What are the Implications?

Magy Seif El-Nasr Ibrahim Yucel Joseph Zupko Andrea Tapia Brian Smith

College of Information Sciences & Technology

Penn State University
University Park, PA

{magy,iyucel,jzupko,atapia,bsmith}@ist.psu.edu

ABSTRACT

The percentage of young women choosing educational paths leading to science and technology-based employment has been dropping for several years. In our view, the core cause for this phenomenon is not a lack of ability, but rather a combination of low self efficacy, misconception of the IT field, and lack of interest and social support from families and peers. The specific aim of this paper is to discuss a case study – a class named *Gaming for Girls*. This class was offered to middle and high school girls three times from Fall 05 to Summer 06. In these classes, female students assumed the role of designers and developers engaged in developing their own games using commercial game engines. Based on this experience, we assert that through the activity of designing games using game engines, girls can understand the process of game development, acquire computer science skills, and increase their confidence level with regards to computing.

Categories and Subject Descriptors
K.4 [**Computers and Education**], J.5 [**Arts and Humanities**]

General Terms
Performance, Design, Experimentation.

Keywords
Education, game modding, learning, gender

1. INTRODUCTION

The gender imbalance problem in the information technology (IT) and game industries has been a topic of interest for many years. In ′02 women accounted for 24.6% of the IT profession compared to 25.4% in ′96 [2] (not counting administrative jobs). A parallel can be found in the game industry where, in 2005, women accounted for only 11.5% of the game development workforce [3].

We regard this problem as a "pipeline" issue. Women earn significantly fewer undergraduate degrees in computer science and engineering than men. This may be traced back to middle and high school education, where women students continue to track out of math and science classes that provide the foundations for IT careers [4-6]. American cultural expectations and influences often convey the message that women are unsuitable for the IT world (e.g., [7]). In years prior to college, research studies have revealed effects of such social norms and expectations; for example, research showed that some girls exhibit low self-efficacy regarding computing and small amounts of informal and voluntary computer exploration (e.g., [8, 9]). It has been suggested that this is related to young women's negative perceptions of the IT field, e.g., it is male-oriented or anti-social [5].

Academic Days '07, Feb, 2007, USA.

As a result, there is a need for interventions that are aimed at increasing middle and high school girls' exposure to design and programming, thus demystifying the technology profession and promoting computer literacy. In this paper, we discuss a case study of such an intervention—*Gaming for Girls*—a game design course aim at engaging middle and high school students in design and programming activities through building games using existing game engines. Our premise is that this activity will (1) increase students' comfort level with technology, (2) demystify game and software development careers and underlying development processes, (3) increase students' self-efficacy with computing, and (4) promote the acquisition of programming and design skills.

In this paper, we will describe three different offerings of the *Gaming for Girls* course during Fall 05, Spring 06, and Summer 06. We present these offerings as case studies, detailing the curricula, engines used, problems encountered, and lessons learned. We collected surveys, interviews, and observational data during each offering. However, due to the small sample size (25 students in each offering) and the constant refinement of the curricula and game engines, it is difficult to provide generalized results, thus we will keep the analysis at an individual level. We will discuss learning outcomes based on our observations and interactions with students as well as their perception of what they learned based on survey data. In future work, beginning with the current offering (Fall 06), we will conduct and experiment with several assessment methods to measure learning outcomes.

2. Games and Learning

Many researchers have argued that games provide suitable environments for learning [10]. Several techniques have emerged from such studies: 1) learning through game playing, and 2) learning through game design, which has three flavours: creating games a) from scratch, b) using tools created by researchers, or c) using commercial game engine, i.e. game modding.

Modding is defined as the process of changing an existing game, thus it generally requires the use of a game with built-in development tools, e.g., *Unreal Tournament* and *Warcraft III*. Game modding demands an understanding of the underlying engine and game mechanics in order to use and modify the game, which is mostly learned through the use of the engine itself or by research on forums. Game modding has the advantage of offering content and mechanics, thus providing an architecture for creating complex and aesthetically pleasing games which are otherwise difficult to build given students' skill and time constraints.

Learning through design can occur in many domains with different types of development activities. Since the development of the Logo language in the 1960s, educational researchers have investigated ways that programming computers can facilitate learning about mathematics, computation, and more general problem solving skills [11-13]. Many researchers have devised approaches to engage students in learning through designing and developing their own games. For instance, Harel's work in elementary schools demonstrated children working for prolonged periods on the creation of educational games using the Logo programming language [14]. Kafai [15] noted similar engagement as students developed their own games, and she also tracked their abilities to incrementally create, evaluate, and revise their designs over time. Hooper's longitudinal study of software development in schools showed students expressing notions of cultural identity in their programs—ideas that were not likely to be expressed had students just played existing games [16].

These studies used Logo as the primary programming language, but a number of programming environments have been created to help novices learn by designing and implementing working computer programs [17-21]. The courses described in this paper are also examples of using games to teach computer science skills where students use commercial game engines instead of research-based languages or tools. Time, cost, and expertise are significant barriers to experimenting with video game design in educational settings, but customizing existing games may reduce the difficulty and make it possible for learners to create credible and aesthetically pleasing prototypes. The time commitment to return is important for middle and high school students since they generally lack the time to devote months to a game project but still desire 'commercial' aesthetics quality. In addition, using commercial game engines provides a robust infrastructure that students can use and a realistic environment that students can learn from (thus teaching them realities of game systems).

3. Gaming for Girls Courses

3.1 Engines used

A different game engine was used for each course offering. The choice of a game engine is critical, as it fundamentally promotes (or hinders) the course's learning objectives. As we previously argued, different engines promote different learning objectives [22]. Therefore, when choosing an engine, an educator needs to consider class schedule, size, style, student skills and age, in addition to the course's learning objectives. We chose three engines: *Warcraft III*, *Game Maker*, *RPG Maker XP*.

Warcraft III was used in the Fall 05 course. *Warcraft III* includes a visual programming tool, the *Trigger Editor*, which allows students to program using dialogue boxes and point-and-click rather than writing code. However, it also allows students who are interested in writing code to do so through the same interface. Its programming environment includes notions of event-driven programming, Boolean logic, and parallel execution. Its art and design tools facilitate 2D map design, terrain design, and the creation of character behaviors. Additionally, it includes in-engine documentation in the form of tool tips and help text. These features may help students focus on semantics rather than syntax.

These features also had drawbacks, however. Semi-complex structures, such as deeply-nested conditionals, are tedious to specify in the visual programming tool. Additionally, the in-engine descriptions assume intermediate to advanced programming knowledge and make assumptions that may not be obvious, such as the fact that an expiration timer on floating text is dependant on the floating point "permanence" being off.

In the Spring 06 course, we used *Game Maker*, an engine that allows students to build 2D games. *Game Maker* is designed with the flexibility to build any type of game, and thus is not associated with a specific interaction model. Unlike *Warcraft III*, which embeds a real-time strategy interaction model, *Game Maker* can be used to produce a side-scroller as easily as a top-down role-playing game. While this greatly increases students' freedom and creativity, it can also be imposing as students needed to develop their own interaction model in addition to building a game. Using an existing game engine for game modding (*Warcraft III*) seemed to increase students' comfort with the tool when compared to creating a game from scratch as is necessary in *Game Maker*.

Game Maker offers a visual programming tool similar to *Warcraft III* with some differences. In *Game Maker*, programs are part of game objects but are also event driven. For example, a ball object can be programmed to reverse direction when a collision event occurs between the ball and a wall object. While the visual programming tool is simple to understand and use, it too becomes tedious to use with semi-complex structures.

Game Maker requires students to understand event-driven programming and a weak concept of object-oriented programming. Variables proved to be more important in *Game Maker* than in *Warcraft III* and parallel processing less important, although both concepts are present in both engines. Students must also understand geometry in 2D, sprites (pixel editing), and collision-detection, as *Game Maker* relies heavily on "object-collides-with-object" events.

In Summer 06, we used *RPG Maker XP*. Like, *Warcraft III* and *Game Maker*, it provides a visual programming tool on top of a scripting language (*Ruby* in this case). Code is event-driven, although the number and types of events is significantly smaller than in the other two engines. Although *RPG Maker XP* is not embedded in a game, it is defined by a 2D "Japanese-style" RPG interaction model, and thus is constrained by that model. This proved beneficial since students reacted favorably to the model, and thus it was easy for them to construct their games using this model as a base.

The visual programming tool of *RPG Maker XP* is conceptually different than its underlying scripting language. In fact, the tool is actually a "mini-language" that is implemented within *Ruby*. As a result, it was very difficult for students to move from the visual programming environment into *Ruby* when necessary, particularly when compared with *Warcraft III* or *Game Maker*.

Students working with *RPG Maker XP* deal with 2D map editing, layers (transparency), and event-driven programming. Switches, which are basically Boolean variables, are used extensively. Students will most likely need to deal with editing stats such as health and mana points to use the engine's combat system.

Table 1 summarizes the concepts that students are required to know in order to work with the engines.

Table 1. Programming concepts required for each engine

Game Engine	Programming Concepts Promoted
WarCraft III	Variables, Boolean logic, event-based programming, parallel execution, 2D map design, terrain design, and character behavior scripting
Game Maker	Variables, Boolean Logic, weak notion of Objects (as entities), sprites, collision detection, 2D geometry and coordinate systems
RPG Maker XP	Variables, Boolean logic, event-driven programming, concept of layers, 2D animation, 2D map design, and basic math for battle stats

3.2 Curriculum

The first offering used lectures to present knowledge and lab-time for developing games to deepen and solidify understanding. Later offerings nearly eliminated lectures all together, presenting knowledge through building mini-projects or other activities with the game engines. The last few days of all classes focused on providing students with an environment to finish and polish their game projects. Instructors concentrated on providing feedback, help, and facilitating discussion and critique.

The first course was offered over a 5-week period during Fall 05 using *Warcraft III*. The class met on Saturdays, once a week, for around 4 hours. Each week focused on a specific topic and students were given homework on that topic. The first week's topic covered map design. Students were asked to design and implement an environment for their games, motivated by a provided short story. The second week focused on characters and object design. Students were given a lecture on creating interesting characters in a narrative sense and asked to flesh their characters out on paper before implementing them in their game. Week three focused on character behavior and plot. This was their first exposure to programming, where they needed to make characters move, talk, and carry objects. The last two classes were spent providing students with more programming knowledge as needed and helping them debug. During the final class, students presented their games to parents and other educators.

The second course was offered over six weeks during Spring 06 using *Game Maker*. The class met on Saturdays, once a week, for 4 hours. The first class introduced *Game Maker* by asking students to build a simple game of *Breakout*, from start to finish. Students used existing art content, but many also created their own sprites for the project. During the second class, students focused specifically on designing environments and the collision of objects in environments. Week three introduced students to programming, and it was at this point that students decided whether they wanted to build a completely new game for the rest of the class, or to build on the *Breakout* game they had created during the first class. The last three weeks of the course were spent drawing and animating characters, polishing and critiquing, and providing students with programming concepts and debugging help as needed to complete their games.

The third course was offered as a 1-week camp during Summer 06 using *RPG Maker XP*. It should be noted that this particular offering engaged only middle school girls, while other offerings included both middle and high school girls. Similar to the second class, students built an entire game from scratch during the first day, this time a tale of King Arthur. Students were asked to bring a fable or myth to the class and spent the next four days telling that story in the game engine. The topics covered included map design and how to make interesting characters, as well as variables, flow control, and parallel execution.

4. Evaluation

We ran a study during each offering to evaluate the impact of the course on increasing self efficacy, engaging girls in design activities, promoting programming and design skills, and enhancing their perception of the IT field. In these studies, both quantitative and qualitative methods were used. Three types of data were collected: (1) surveys conducted at different time periods during the course sessions, (2) observations of student performance and questions during class periods, and (3) analysis of projects and assignments completed by the students.

Survey methods are often subjective, rely on perception, and to a large extent rely on the participant's judgement. However, they can be effective in measuring certain qualities, such as motivation and self-efficacy. The changes in the curriculum and engine prevent us from generalizing our findings.

The analysis presented in this section will be use the survey data taken from the summer 06 course. These surveys were comprised of both closed and open ended questions. In the case of the closed ended questions made of discreet categories, we present the data in numerical form, providing descriptive statistics. In the case of the open-ended questions, we provide the data in textual form, as illustrative quotes. These quotes are used both as stand alone qualitative data as well as supporting evidence for the numerical descriptive data, adding richness and meaning.

4.1 Capture and Motivate

On the first day of class, we asked students to talk about their motivations and hopes for the course. Most students expressed some excitement for creating a game. One student said, "After this morning's class, I'm excited to start working on more RPGs and perhaps even buy the program and make my own RPGs later." Other students expressed a desire to creatively bring their stories and characters to life. For example one student said she was most interested in, "…making my characters talk, building a world, and making an interesting story." When asked why they decided to take the course, they stated they liked computers (68%) and games (68%), and thought the class would be fun (61%). When asked how they felt about computers, 83% said they "loved them."

Parents were asked to complete a survey one week after the end of each course. When asked what long term effects the class had on their daughters, slightly more than half of the parents said they had noticed some change. A parent stated, "She learned the math she has been studying in school can have a real application. She learned programming can be fun."

It seems that game design motivated and captured the interest and attention of the students in our classes. The positive opinions from students must be tempered by the limitations of this study. This population was self-selected: students already had an interest in computers and gaming before they enrolled in the class or they would not have been interested. While the data says nothing about the effects this class might have on a truly general population, it obviously had some positive effect on this narrow, self-selected sample. This question demands further research.

4.2 Self Efficacy and Perception of IT

On the first day of class, the students were asked several questions to determine their confidence level with computers and their perceived self-efficacy. 24% felt they knew a lot about computers, 48% felt they knew [somewhat] a lot about computers. Fewer claimed they knew a lot about computer games. Fewer still felt they were confident with programming. Several expressed concern managing the programming aspects of the course. Another group of students expressed concerns being able to finish the project in the allotted time. One student said, "I don't know if I'll be able to finish a whole video game in 4 more days."

On the same day, we asked the students what they hoped to learn. The most common answer was to build video games. However, about a third of the students responded with the desire to learn more programming or computer skills. One student said she would like to learn, "… how to make an awesome video game. I want to learn everything about technology or at least more than I did." Another student stated that she simply wanted to learn, "how to be able to fix minor problems on my family's computer."

On the last day of class we asked the students similar questions about competency and self efficacy. 64% of students responded that they felt more confident about their abilities than they had on the first day, with 36% more stating they felt somewhat more confident. 96% felt they had learned a lot from the class. 48% felt they understood more about computer programming than on the first day with an additional 40% stating they felt somewhat more confident in their programming abilities. 52% felt they clearly understood how a computer game is built with an additional 48% giving more cautious assent. 60% felt very confident they could build a computer game in the future with an additional 24% feeling somewhat confident. Perhaps most importantly, 76% said they would like to take a more advanced programming class.

Before the course began, parents were surveyed on the impact of the *Gaming for Girls* class on their daughters. The majority of the parents hoped that their daughter would learn how to make a computer game (32%) or how to program a computer (28%). When the parents were asked what they imagined their daughter would be doing in the class, they unanimously answered learning how to create computer games using programming tools.

Parents were surveyed a second time one week after the end of each course. 88% of parents felt that the camp may have influenced their daughter's perception of working with computers, and confidence level with computers. When asked what long-term effects the class had on their daughters, slightly more than half of the parents said they had noticed some change. A mother stated that her daughter, "…has always been fairly comfortable with computers but she talks more about getting a Dell or converting one of our Macs with a PC emulator. The camp was clearly a confidence booster—something immeasurably important to girls of this age group." The parents also felt their daughters had gained technical skills. A mother of a student said, "she learned the basics of how games are made. She learned about various applications of computer technology and how computers are used in various areas."

Some parents have also expressed the impact of the course on their daughters' technology related activities and career choices. For example, a parent stated, "she was extremely enthusiastic about pursuing technology as a possible career choice. This is something that I will need to follow-up on to ensure that she is given the opportunity to explore. Additional classes would be of great interest." A mother of one of the students said about her daughter, "She wears her tee-shirt with confidence and talks often about her camp experience. She also talks more about enrolling in the College of [Information Sciences and Technology] and would like to explore possible scholarships, grants, and/or funding for that program." Another mother stated that her daughter, "… has purchased the software and is making new games already."

5. Discussion

Each offering resulted in many lessons that helped us reshape future offerings. These lessons were collected through student comments, discussions with individual students, observations, as well as the surveys and interview data collected.

The first two courses were offered during the school year (Fall and Spring). Our collected survey data indicated that girls were very busy and involved in many activities that competed with our course, including clubs, social activities, and of course, school. Spring was particularly busy, during which we had the lowest retention rate of the three classes. In general, many girls responded that they needed more time or would have liked to devote more time to their projects outside of class. Additionally, due to time commitment during the first two classes, it seemed that it was harder for students to assimilate the design and programming techniques. While all students demonstrated some understanding of the basics, such as Boolean logic, flow control, variables, and events as demonstrated by their project work, we felt that some left the class with holes in their knowledge or didn't fully understand some of the basics. On the other hand, some students demonstrated advanced knowledge beyond what was taught, such as using the scripting language in *Game Maker* to manipulate low-level parameters of game objects.

In the third class, all students understood most or all of the basic concepts we targeted, e.g., variables, Boolean logic, map design, mathematical manipulation to balance fight mechanics. Indeed, we entered this class severely underestimating their abilities and needed to add a great deal of material. For example, we presented a tutorial on creating sprites, including how to add highlights and shadow to sprites. Our initial plan did not include any discussion about creating sprites at all, which we added. Further, several students explored the *Ruby* programming language underlying the visual programming environment of *RPG Maker XP*, adding new features such as a visible timer. From analysis of projects and interactions with the students, we didn't note any student who intentionally avoided an idea or gave up on an idea because she found it too hard or lacked confidence that she would figure it out. Every game created had interesting game-play, map design, usage of music and sound effects, and a well told story. Student self-efficacy was very high.

The selection of the engine was also a very important choice. For example, students were constantly fighting *Warcraft III's* default interaction model as they tried to create their games. *Game Maker* posed the exact opposite problem. Students were presented with a blank slate, with no built-in interaction model to anchor their ideas and very little art content. *RPG Maker XP* seemed to strike the best balance, providing a great amount of content and a solid interaction model that was flexible enough to let students control their narratives.

In the summer course, we had time for polishing and critiquing, but we found that the girls had very little interest in revisiting their games. It seemed that the girls had little interest in reflecting on their games once they were completed.

Students were interested in drawing or otherwise creating their own characters throughout all of our classes (character modeling was a topic that came up unanimously as something the students would be interested in learning in the future). Sprite animation was covered in detail during the Summer class. However, once we showed them the steps involved, they generally lost interest, and very few students actually created their own characters. Instead, students tried to find the best fit among the provide content; for example, they would edit the sprites provided in terms of changing color hue, such changes involved much less time and effort than creating characters and sprites from scratch. A challenge for us in the future is to bring novel forms of visual control over characters into our classes that give our students the desired freedom without introducing a deterring time investment.

6. Future Work

The work presented here discussed three *Gaming for Girls* course offerings where we used a different engine for each offering. The courses provided a great environment for learning computer science skills. We have seen students apply basic programming concepts, such as variables, loops, and conditionals, and more advanced concepts, such as parallel and event programming as discussed in [23]. We can say with confidence that the majority of projects across all classes demonstrated the knowledge we targeted in the courses. However, to what degree students actually learned these concepts is unknown. This problem requires further research work. Future courses will include different assessment methods to gauge learned knowledge.

7. Conclusion

Over the three offerings of the *Gaming for Girls* course, data collected suggests that engaging girls in game design and development using commercial game engines can be used as a vehicle for (1) increasing students' self efficacy, (2) acquiring design, programming, and artistic skills, while (3) engaging them in the activity. However, more work is needed to generalize this assertion. We are continuing to run this class, and thus will continue to gather survey and observation data, which will help us generalize this assertion. In addition, we are currently exploring several assessment techniques to measure learning outcomes in future courses.

8. REFERENCES

[1] ITAA, "Adding Values: Growing Careers, ITAA's 2004 Workforce Study," Arlington, VA 2005.

[2] ITAA, "ITAA Blue Ribbon Panel on IT Diversity," 2003.

[3] IGDA, "Game Developers Demographic Report," 2005.

[4] T. Camp, "The Incredible Shrinking Pipeline," Communications of the ACM, vol. 40, 1997.

[5] J. Margolis and A. Fisher, "Greek Mythology and Attracting Undergraduate Women to Computer Science," presented at Joint National Conference of the Women in Engineering Program Advocates Network and the National Association of Minority Engineering Program Administrators, 1997.

[6] J. Margolis and A. Fisher, Unlocking the Clubhouse: Women in Computing. MA: MIT Press, 2002.

[7] E. M. Trauth, "Odd Girl Out: An Individual Differences Perspective on Women in the IT Profession," Information Technology and People, vol. 15, pp. 98-118, 2002.

[8] C. Beise, "IT Project Management and Virtual Teams," presented at Proceedings of the 2004 SIGMIS Conference on Computer Personal Research, Tucson, AZ, 2004.

[9] S. Nielsen, L. Von Hellens, and S. Wong, "The Game of Social Constructs: We're Going to WinIT," presented at Proceedings of the 2000 International Conference on Information Systems (ICIS), 2000.

[10] J. Gee, What Video Games Have to Teach Us About Learning and Literacy. NY: Palgrace Macmillan, 2004.

[11] S. Papert, Mindstorms: Children, Computers, and Powerful Ideas. New York: Basic Books, 1980.

[12] M. Resnick and S. Ocko, Lego/Logo: Learning through and about Design. Norwood, NJ: Ablex Publishing, 1993.

[13] R. D. Pea, D. M. Kurland, and J. Hawkins, "Logo and the Development of Thinking Skills," in Mirrors of Mind: Patterns of Experience in Educational Computing, R. D. Pea and K. Sheingold, Eds. Norwood, NJ: Ablex Pub., 1987.

[14] I. Harel, Children Designers. Norwood, NJ.: Albex, 1991.

[15] Y. Kafai, Minds in Play: Computer Game Design as a Context for Children's Learning. Mahwah, NJ: Erlbaum, 1994.

[16] P. K. Hooper, "They have their own Thoughts: Children's Learning of Computational Ideas from a Cultural Constructionist Perspective." Cambridge, MA: MIT, 1998.

[17] M. Conway, S. Audia, T. Burnette, D. Cosgrove, and K. Christiansen, "Alice: Lessons learned from building a 3D system for novices," presented at Proceedings of SIGCHI Conference on Human Factors in Computing Systems, 2000.

[18] A. Repenning and J. Ambach, "The Agentsheets Behavior Exchange: Supporting Social Behavior Processing," presented at CHI 97, New York, 1997.

[19] M. Resnick, Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds. Cambridge, MA: MIT Press, 1994.

[20] D. Ingalls, T. Kaehler, J. Maloney, S. Wallace, and A. Kay, "Back to the Future: The Story of Squeak, a Practical Smalltalk Written in Itself," presented at Proceedings of the 12th ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications, 1997.

[21] D. Smith, A. Cypher, and J. Spohrer, "Kidsim: Programming Agents without a Programming Language," Communications of the ACM, pp. 54-67, 1994.

[22] M. Seif El-Nasr and B. Smith, "Learning through Game Modding," presented at Games, Learning, and Society, Wisconsin, 2005.

[23] I. Yucel, J. Zupko, and M. Seif El-Nasr, "Education, IT, Girls, and Game Modding," International Journal of Interactive Technology and Smart Education, vol. 3, 2006.

The Effects of Games in CS1-3

Jessica D. Bayliss
Rochester Institute of Technology
Rochester, NY 14623
585-475-2507

jdb[at]cs[dot]rit[dot]edu

ABSTRACT

Last year, the RAPT (Reality and Programming Together) program ran CS1-3 course sections using games as a motivator in teaching traditional outcomes. Indicators such as grades on common finals, retention of students, and performance in CS4 were used to compare regular students with RAPT students. No significant retention differences were found in CS1 and CS2, but student success rates changed in CS3 (100%) and CS4 (95%) for RAPT students when compared with regular CS3 (84%) and CS4 (73%) students. We discuss the RAPT courses and course materials along with course differences between regular and RAPT sections that could have led to different success rates for students.

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

General Terms
Algorithms, Design, Languages

Keywords
Games, CS1, Video Games

1. INTRODUCTION

Gaming is a popular pastime for students from all walks of life and is often a common link between students from very different backgrounds. It is now one of many extracurricular reasons that students look at when choosing a particular university [1]. Games may be motivators for learning in a variety of situations [2][3][4]. They have recently been popular in coursework, although studies with formal control groups remain rare. The general consensus seems to be that games can motivate students and provide positive experiences [7][8].

The reasons for bringing games into a course are varied. Some courses actually have students play games constructed to help them learn materials as diverse as history and programming [5][7][10]. Some courses study the design and programming of games themselves [9][12]. There is even a study on using the massively multiplayer game Everquest II in order to learn and practice a foreign language [6].

The RAPT program concentrates on using games as an application area on top of traditional Computer Science curriculum. The RAPT program courses maintain the same outcome goals as non-RAPT CS1-3 courses, and retention rates and final exam results are compared as a measure of effectiveness. The CS1-3 courses teach core topics in the Computer Science discipline such as programming skills, algorithm design, data structures, and software engineering principles. These courses are taught on a quarter system and are the equivalents of CS1-2 courses from a semester system.

The RAPT program introduces traditional topics and curricula through using games as an application area. Computer games are complicated pieces of cross-disciplinary software that draw from fields such as art, english, physics, and biology. As such, games can provide a rich set of examples and materials for coursework. Games also allow creative student expression. RAPT students designed and programmed their own games in CS3 and some student projects contained original art and music.

Forty percent (approximately 80 students) of the Computer Science entering class applied to be in the RAPT program. Current results from RAPT CS1-3 indicate that grades from common final exams are slightly higher, but not significantly different from exam scores coming from students in regular (non-RAPT) course sections. Furthermore, students find the RAPT materials to be subjectively more satisfying than non-RAPT materials. More interestingly, the retention rates for RAPT sections are not significantly different from regular sections for CS1-2, where students are often discovering what computer science is about. The results are different for CS3 where RAPT students had no individuals with D's, F's, or W's and regular sections had 16% in these categories.

In order to try and determine if this difference was due to instructor bias, these students were tracked through the next course (CS4), which they had with a different instructor. CS4 had no emphasis on games and is a course where students learn the C++ language as well as pointers and more advanced software design. The RAPT students were unsuccessful 5% (n=20) of the time whereas the regular CS4 students (n=34) were unsuccessful 27%. They were unsuccessful since they obtained a D, F, or W in the course.

These results appear to indicate that while students want to engage in using games to promote learning (indicated by the 40% application rate), this may not translate directly into increased retention in the first two courses. This does not mean that games do not provide a powerful motivator however, as retention and grades from CS3 seem to indicate that students may have learned more and that this helped them in the CS4 course. We discuss this and other reasons for the potential success of the RAPT students below.

2. The RAPT Program

The RAPT program was started during the summer of 2005 in order to teach traditional CS1-3 concepts using computer games as an application area. The program successfully ran through the 2005-2006 school year and is entering its second year. We continue to collect statistics on the students in this program.

2.1 RAPT CS1

RAPT CS1 was first taught over the summer in 2005 and was taught again in 2006 due to the success of the first program. Results from 2005 indicated that it helped students “get ahead” in college and meet each other before the start of the school year [11].

The course itself was taught as a synchronous, online course consisting of 2 hours of lecture per week and 2 hours of lab. As a program rather than a course, it carried no credit and there was no fee to be a part of the program. In order to keep the course linked with real industry practices, game developers presented their work in on-line chat sessions over the summer.

The only material benefit of this program for students is that they could gain the knowledge to place out of a traditional CS1 course upon entering in the fall quarter. Placement into CS2 frees up one course slot for students. Students are self-selected: they must apply to the program. During both 2005 and 2006, approximately 40% of the entering CS freshman class (around 80 students) applied to be a part of the ten week RAPT summer program.

In 2005, 48 students were accepted and in 2006 this number was decreased to 28 due to a decrease in available slots for the fall RAPT CS2 course. Six extra students from the 2006 class were allowed to “sit in” on the RAPT sessions without an expected seat for the CS2 follow on course. These individuals were people who did not have any previous programming experiences and in some cases were people who were not necessarily convinced that they wanted to major in Computer Science.

Of the total number including the sit in students, 5 were women (~15%). This is somewhat higher than the percentage of women entering Computer Science at our university where the numbers have been as low as 5%. Of note, three of these women were among those “sitting in” on the course, because they had no programming background. All three are still enrolled in Computer Science as their major. This appears to indicate that females are not “scared off” by the games nature of the RAPT program.

The summer program followed the outcomes of the university’s regular CS1 course:

1. Students will be able to identify the basic elements of program syntax, semantics, and computer language concepts.
2. Students will understand the fundamentals of object-oriented design, including classes, objects, and reference vs. primitive data types.
3. Students will describe the phases of the software development life cycle.
4. Students will be able to distinguish between various control structures (e.g., repetition and selection).
5. Students will compile, run, and test Java programs, and apply error recovery strategies for the verification of programs.
6. Students will understand and be able to discuss the ethical responsibilities of a computing professional.

Java was used as the main programming language in this course. An exam at the end of the program tested knowledge of the expected outcomes, and this was further highlighted with students taking and passing the placement exam. The placement exam does not test outcomes 3 and 6. Outcome 3 was met by quiz results from the course as well as lab performance on labs where students had to answer questions on the software development life cycle. Outcome 6 was met during week 10 when students actively discussed the EA Spouse letter [13] in terms of the ACM Code of Ethics.

All of the 9 weekly programming labs involved either playing or coding on a game [15]. Games were discussed in lecture as sophisticated examples of software design and implementation. Examples from games were used to discuss testing programs, expressions, modulus, objects, algorithms, loops, and arrays among other topics. Please see our paper on the CS1 course materials for more information [11].

2.2 RAPT CS2

RAPT CS2 is the first course students in the RAPT program take upon entrance to the University in the fall. This course focuses on expanding the concepts from CS1 with more advanced programming concepts including threads, exceptions, and event driven programming. This is the first point in the program where it becomes possible to compare regular CS2 with the RAPT version of the course. RAPT CS2 is taught in the studio format as are most of the regular CS2 sections.

The common outcomes for students in all sections of CS2 are:

1. To translate requirements for small software problems into object-oriented designs and working programs.
2. To apply the collection classes from a collection framework library to software designs. Included are collections such as linked lists and arrays, along with supporting classes such as iterators and interfaces used for comparing collection elements.
3. To able to design simple multithreaded programs and follow their execution.
4. To be able to design simple networking programs and follow their execution.
5. To be able to design and implement software for File I/O.
6. To be able to design and implement software applications that use graphical user interfaces, are designed around an event-driven paradigm, and employ common visual components such as windows, buttons, text fields, and menus.
7. To be able to understand how exceptions work, explain their uses and employ them in their own designs.

8. To be able to discuss ethical issues that arise in taking on potentially dangerous or infeasible software development assignments.

Additionally, RAPT CS2 had an extra outcome: to learn C#. The main reason for switching languages at the beginning of CS2 is to encourage students to learn about different languages, while at the same time allowing them to leverage their existing knowledge of Java-like syntax. The course discusses some of the differences between C# and Java including delegates and checked exceptions.

The course involved a quarter long project that used the Gamebots mod for Unreal Tournament [18] in order to have students practice the concepts they learned in class. The project had them create game (ro)bots that played against each other based on developing a client side program that connected with the Unreal Tournament game running the server side Gamebots mod.

The project involved parsing commands sent by the Unreal Tournament server to each client bot, constructing the networking code for communication, creating a class hierarchy for bots with a common interface, and programming the AI for an individual bot. Students were motivated to search the web for a variety of algorithms to use to make their bots smart. The most complicated bot used a fuzzy state machine in order to make decisions when competing against other players.

There was a two-hour lab every week, with parts of the lab not finished during the time serving as homework that was due the next week. Please see the course web site for more detail [16]. The labs involved either playing games or writing them. Some lab examples were:

- Students wrote stories while using complicated exception handling in order to demonstrate their understanding of exceptions. These stories involved multiple endings in a “Choose Your Own Adventure” style.
- Students wrote programs to procedurally generate sounds such as could be heard on a nature tape using threads and simple sounds. The most complicated student work was an homage to Homer Simpson, which used 18 threads and was written by a student musician.
- Students learned about data driven programming through altering a tetris game to read in files containing blocks rather than having to define all blocks inside the program. They then wrote a generic block rotation algorithm.
- Students changed the graphical user interface for the Gamebots project in order to include important information about student bots locations and items.

2.3 RAPT CS3

RAPT CS3 continued where RAPT CS2 left off and concentrated on data structures and algorithms. The defining feature of CS3 was the ability of students to work in groups of size 4-5 in order to design and create a game of their choice. There were multiple landmarks for the quarter long project and students were required to both use and construct a variety of data structures and algorithms for their game. RAPT CS3 was taught in the same studio environment as RAPT CS2.

The outcomes of RAPT CS3 matched regular CS3 with the addition of learning C# and learning how to work in a group:

1. Students will be able to explain the properties of stacks, queues, lists, trees, and graphs, implement them, apply them to common software problems when appropriate, and analyze their performance.
2. Students will be able to analyze the efficiency of algorithms, including those used for sorting and searching.
3. Students will be able to translate requirements for small software problems into algorithmic solutions and working programs.
4. Students will be able to describe hashing techniques and apply them to appropriate data organization problems.
5. Students will be able to discuss ethical issues that arise in taking on potentially dangerous or infeasible software development assignments.

All outcomes were met through using games as an application area with the exception of outcome 5. This outcome was met through student discussion of the Star Wars program and subsequent test questions on the ethical issues involved.

There was a two-hour lab every week, with parts of the lab not finished during the time serving as homework that was due the next week. Please see the course web site for more detail [17]. The labs primarily involved algorithms and data structures in games. Some lab examples were:

- Constructing and using a binary space partition (BSP) tree in order to draw a graphical room correctly from back to front
- Constructing the midpoint line and circle drawing algorithms
- Constructing a solution to the Towers of Hanoi problem both recursively and iteratively to show how the same problem can be solved using different techniques
- Constructing a graph of bot node points in Unreal Tournament and writing/using breadth first, Dijsktra’s algorithm, and A* to search for a bot path to a particular location (2 labs).

3. Results

RAPT sections of CS1, CS2, and CS3 outcomes were measured against regular CS1-3 courses through similar final exams and retention numbers. Additionally, RAPT students were tracked through CS4 in order to look at longer term retention and student grades after RAPT is finished. Table 1 shows the basic retention percentages on a course-by-course basis.

Of the 28 students accepted into CS2 in 2006, two never attended class, one withdrew after finding out he obtained a 5 on the AP AB exam, and three withdrew from the course for various reasons. Of the 22 completing the distance program, 2 failed the placement exam.

This close to 10% failure rate for those taking RAPT CS1 in 2006 is very similar to the 8% failure rate seen in the RAPT course from the placement exam in 2005. Of those taking the placement exam in 2006, none of them would have automatically placed into CS2 or CS for AP students due to Computer Science AP exam scores. All were slated to take CS1 in the fall before taking the placement exam. The traditional failure rate for CS1 students from the regular sections is 10% in this first class and so no extra retention of students was shown with this course.

Table 1:Lack of success due to D's, F's, and W's. The percentages and numbers across RAPT sections are compared with regular CS1-3 sections.

	Normal Sections*	RAPT*
CS1	10% (n=443)	8%(n=37, Pass/Fail^)
CS2	19% (n=360)	22% (n = 32)
CS3	16% (n=319)	0% (n = 21)
CS4	27% (n=34)&	5% (n=20)

* Lack of success indicates that a student had a D, F, or W and does not indicate any loss due to students choosing not to take the course on the normal course track.

^ Students in the summer RAPT program had Pass/Fail due to the placement exam in the fall being the measure of success in the course.

&The measure was obtained from one section of CS4 rather than all.

As a potential side effect of the distance nature of the course, students were shown to be less intimidated by their peers than regular CS1 students (see Table 2). It is possible that the distance nature of the CS1 RAPT course makes students less likely to be intimidated. Students who may not know any programming are still able to compete with others in playing games.

Of the 34 individuals from the 2005 RAPT CS1 summer program who placed out of CS1, 2 around the border of passing the placement exam decided to take a regular CS1 course to solidify CS1 knowledge (they were both successful in passing CS1-3), 1 placed into CS for AP students (the course above CS2), and 1 decided to take a regular CS2 course since he didn't care for the games emphasis of RAPT CS2. It is important to note that not all students like games and a course emphasizing games should be one of a few potential courses that a student can take to satisfy their CS1-3 requirements.

Three students withdrew in the middle of the quarter with two of them switching to the Information Technology major. Four individuals also failed the RAPT CS2 course and all of them suffered from overall university GPA's below 2.0. This represents a 23% loss in students, which is fairly similar to the 19% of students receiving D's, F's, and W's among the regular CS2 sections from the fall of 2005. RAPT CS2 retention was not significantly different from regular CS2.

Since the CS2 final measures basic understanding of common programming concepts, the regular CS2 final could be translated into C# for RAPT. Thus, the regular and RAPT finals that were given were very close to each other. The average on this final was 85%, which compared favorably with final exam averages from the same teacher in a previous regular CS2 section where the average was 81%. The similarities between the averages imply that the RAPT students are not being hurt by the emphasis on games in the course.

Slightly more students received A's in the course (44%) in RAPT when compared with the average student in a regular CS2 section (37%). The instructor's average percentage of A's from previous CS2 sections was 30%. Students from the CS2 course compare favorably with students in regular sections.

Table 2: Student perception of intimidation due to their peers. While the numbers are significantly smaller due to the size of the RAPT course, intimidation percentages look different from a normal CS1 course.

	Always	Frequently	Sometimes
CS1 (n=369)	7%	15%	23%
RAPT 2005 Survey Week 4 (n=33, 2005)	0%	3%	30%
RAPT 2005 Survey Week 10 (n=21, 2005)	0%	5%	19%
RAPT 2006 Survey (n=30)	0%	10%	13.3%

Of the 20 students entering RAPT CS3, there were no D's, F's, or W's from the course. This is the first point where there is a fairly large difference from the regular CS3 course. In regular CS3, D's, F's, and W's were obtained by 16% of the people taking the course.

In order to show the effects of the RAPT sequence on students, all students passing CS3 were tracked through CS4 during the spring quarter and will be tracked through Software Engineering I and Operating Systems. None of these courses are taught by the professor teaching the RAPT courses.

CS4 introduces students to the C++ language and more sophisticated software design. Of the 20 RAPT students entering CS4, all passed the course but one. The one that did not pass failed multiple courses during the spring quarter. Of students taking the course, 60% of the students passing CS4 received a grade of A in the course, 35% received B's, and 5% (1 person) received a C. None of these students have changed majors.

The grades obtained from a regular CS4 section during the spring quarter (n=34) were 38% A's, 24% B's, 12% C's, 3% D's, and 24% F's (rounded percentages). The trend of RAPT students obtaining higher grades with less of a likelihood of dropping out or failing CS classes continues with CS4.

4. DISCUSSION AND FUTURE WORK

The higher success rates of RAPT students in CS3 and CS4 could be due to several factors. The main factors most likely affecting the success of these students are the games emphasis of the course and the self-selected nature of the students themselves.

Students themselves think that the games emphasis is a very positive experience with subjective comments like the following as indicative of the majority of the class: "The one thing I really like about this course is that it's programming focused on video games. Personally, when I first started this program, I thought it was going be average CompSci class, just with like one or two references to video games. Then, probably after the second day, I realized that this program was new and demanding, yet exciting, fun and way different then regular CS classes." One hundred percent of those going through the program wanted RAPT to continue for future students.

CS4 was taught to RAPT CS4 students in a studio learning environment and could have contributed to the higher number of A's seen between the RAPT and regular students. The course materials, projects, and outcomes are the same between the sections.

The self selected nature of the RAPT students means that they could be more highly motivated to achieve than regular students. Several occurrences outside of RAPT indicate the nature of the RAPT program may have picked out these kinds of students, even though they are not generally students who would have been chosen for the university honors program and they did not have CS AP exam credit in general.

As an example, the regular CS4 course is normally taught in a lecture/lab delivery style. Upon hearing this, the RAPT students in CS3 put together a petition in order to ask the Computer Science Department to consider allowing a RAPT CS4 section in the studio delivery model. All students in the course signed this petition and after meeting with the CS department chair, the course was allowed to be in the studio delivery model as opposed to the normal lecture/lab model.

RAPT students have also been very active in the department. The Computer Science Department hires student lab instructors to help teach CS1-4 labs. Of the 42 lab instructors for fall of 2006, 6 are from the 20 that went through RAPT CS3. This represents 30% of the RAPT section. It is much more normal to see around 10% of students from a CS3 section apply to be lab instructors.

In the future, more results from RAPT will be collected as the program is in its second year. Original RAPT students will also be tracked through Software Engineering I as well as Operating Systems in order to look at any grade/retention differences.

5. ACKNOWLEDGMENTS

This program was supported by a Microsoft Computer Gaming Curriculum grant.

6. REFERENCES

[1] Ali, M.,GGL's First Annual Gaming Survey, Retrieved September 8, 2006 from http://www.ggl.com/news.php?NewsId=3886

[2] Sweedy, E., deLaet, M., Slattery, M. C., and Kuffner, J., Computer game and CS education: why and how, Proc. of the 36th SIGCSE technical symposium on Computer science education (Feb. 2005), 256-257.

[3] Parberry, I., Roden, T, and Kazemzadeh, M. B.., Experience with an industry-driven capstone course on game programming, Proc. of the 36th SIGCSE technical symposium on Computer science education (Feb. 2005), 91-95.

[4] Clua, E, Feijo, B., della Rocca, J., Schwartz, J., Das Gracas, M., Perlin, K., Tori, R., and Barnes T., Game and Interactivity in Computer Science Education, SIGGRAPH Educators Program, 2006.

[5] Schrier, K., Using Augmented Reality Games to Teach 21st Century Skills, SIGGRAPH Educators Program, 2006.

[6] Rankin, Y. A., Gold, R., and Gooch, B., Playing for Keeps: Gaming as a Language Learning Tool, SIGGRAPH Educators Program, 2006.

[7] Schwartz, J., Stagner, J., and Morrison, W., Kid's Programming Language (KPL), SIGGRAPH Educators Program, 2006.

[8] Lewis, M. and Massingill, B., Graphical Game Development in CS2: a Flexible Infrastructure for a Semester Long Project, Proc. of the 37th SIGCSE technical symposium on Computer science education, 2006.

[9] Parberry, I., Kazemzadeh, M., and Roden, T., The Art and Science of Game Programming, Proc. of the 37th SIGCSE technical symposium on Computer science education, 2006.

[10] Chamillard, A.T., Introductory Game Creation: no Programming Required, Proc. of the 37th SIGCSE technical symposium on Computer science education, 2006.

[11] Bayliss, J.D. and Strout, S., Games as a Flavor of CS1, Proc. of the 37th SIGCSE technical symposium on Computer science education, 2006.

[12] Coleman, R., Krembs, M., Labouseur, A., and Weir, J., Game design & programming concentration within the computer science curriculum, Proc. of the 36th SIGCSE technical symposium on Computer science education (Feb. 2005), 545-550.

[13] The EA Spouse Letter, Retrieved August 10, 2006 from http://ea-spouse.livejournal.com/274.html.

[14] Bayliss, J., RAPT CS1 Course Home Page, Retrieved September 8, 2006 from http://www.cs.rit.edu/~cs1.

[15] Bayliss, J., RAPT CS2 Course Home Page, Retrieved September 8, 2006 from http://www.cs.rit.edu/~cs2.

[16] Bayliss, J., RAPT CS3 Course Home Page, Retrieved September 8, 2006 from http://www.cs.rit.edu/~cs3.

[17] The Gamebots mod, Retrieved September 8, 2006 from http://www.cs.rit.edu/~jdb/gamebots.

XYZZY: Finding New Magic in Text Adventure Games

Brian C. Ladd
State University of New York, Potsdam
Potsdam, NY 13676
1-315-267-2944
laddbc@potsdam.edu

ABSTRACT

. Computer game development entices, entertains, and engages students of the "Nintendo" generation. Many instructors have taken this to mean that students must develop games as graphically intensive as those recently released by Nintendo. The current work describes the use of text adventure games to engage students at much lower cost. The focus on text means students work with standard C++ and the fundamentals of computer science rather than being distracted producing eye candy. Student enthusiasm, evaluations, and outcomes attest to the success of our approach.

Categories and Subject Descriptors

K.3.2 [Computer Science Education], K.8.0 [Games]

General Terms

Design

Keywords

Text adventure game, computer science education, C++, game studies

1. INTRODUCTION

Before the recent slump in interest in undergraduate computer science, computer games were well nigh invisible in the computer science curriculum. Now, as the computer games industry grows and new cohorts of undergraduates come to college having been raised on game consoles of ever increasing capability, computer science educators look to computer video games to entice, engage, and educate new computer scientists. Computer science educators look to bring technologies from the computer games industry into the classroom in capstone projects, game development concentrations, and even introductory courses.

Unfortunately, the use of industrial strength solutions in introductory courses can add to introductory students' frustration as they must surmount the learning curve for both a new programming language *and* a complex and powerful video game engine. This paper reports on success in harnessing the engagement and entertainment aspects of computer games for introductory students by reaching back into the history of computer games, back to the era of *text adventure games*. This approach has been applied as both a final project in an objects-early CS1 course as well as a project-based CS1.5 course.

These projects also lend themselves to the use of reinforcing parallel development of both a game and a game engine, the introduction of current literature in game studies, the use of prose and oral design presentations, and student designed assignments. All of this while maintaining a focus on computer science fundamentals clearly wrings new magic from a venerable game genre.

2. DIGITAL GAMES IN CSE

Soloway and Guzdial [24, 6] both discuss how many students now entering college were exposed to computers through 16-bit consoles. Both discuss ways to use games in the computer classroom to raise interest and engagement in programmers of the "Nintendo generation." The Oblingers make similar observations across the curriculum, also noting that increased engagement correlates with improved performance[17].

For all the reasons discussed in the above, the use of digital games in computer science education has grown in recent years. Bayliss[1], Ladd[13], Ladd and Harcourt[12], and Parberry *et. al.*[18.19] all discuss using game development assignments in different level courses. Several different approaches were discussed in a panel by Woltz *et. al.*[27].

Except for Ladd, the above examples all focus on the use of graphical computer games in the classroom. Even descendants of Karel the Robot use graphics and sound to reward students learning to use them [20,4].

Recent work in game studies has looked past graphical digital games to a simpler, text-based age. Jerz has described scholarship in *interactive fiction* or text adventure games[8] and Monfort's *Twisty Little Passages* is a deep literary study of the genre, examining it as a modern expression of the timeless art of the riddle[14].

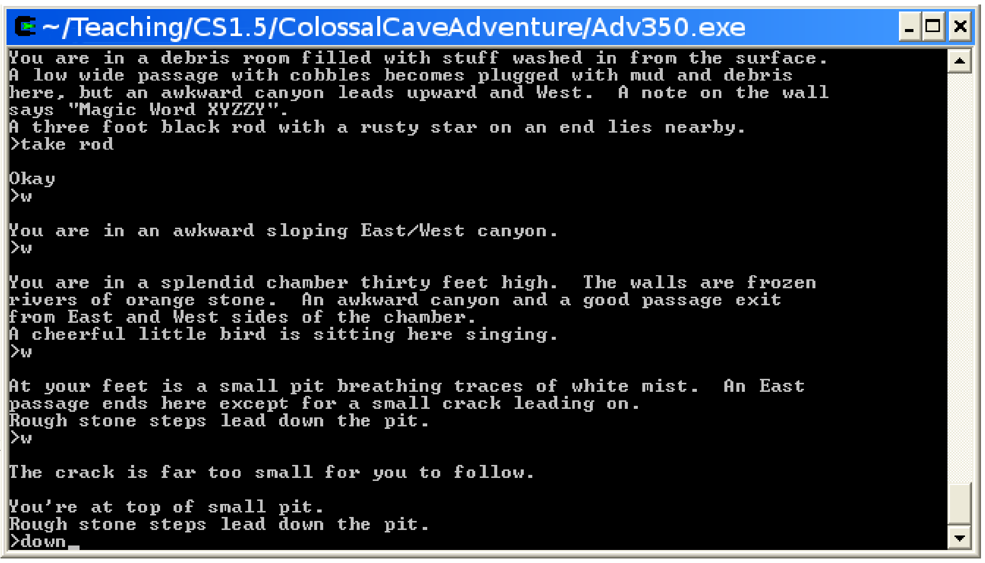
Text adventure games have been used in computer science education: Moser looked at making a text adventure game to teach programming concepts[15] and Summit's class notes teach C using a text adventure game project[25].

Using games to entice, entertain, and engage introductory programmers has been successful; many of the approaches, however, use graphical programming. This either requires a complex game engine or toolkit for students to program or a lot of additional effort for students to get the game looking the way they want. Taking a page from the game studies literature, the current paper reports on gaining the benefits of games without the graphics, of traveling back in time to the days of the great text adventure games.

3. EXTADVENTURE GAMES

Our Goals

Having historically taught a CS1.5 project course with each professor designing their own semester long project, the computer science faculty sought a single project that met five goals: use standard C++; include opportunities for student communication in written and verbal form; require creative, student-designed content; provide a context for the introduction of a wide range of software engineering topics; and engage our students.



```
You are in a debris room filled with stuff washed in from the surface.
A low wide passage with cobbles becomes plugged with mud and debris
here, but an awkward canyon leads upward and West.  A note on the wall
says "Magic Word XYZZY".
A three foot black rod with a rusty star on an end lies nearby.
>take rod

Okay
>w

You are in an awkward sloping East/West canyon.
>w

You are in a splendid chamber thirty feet high.  The walls are frozen
rivers of orange stone.  An awkward canyon and a good passage exit
from East and West sides of the chamber.
A cheerful little bird is sitting here singing.
>w

At your feet is a small pit breathing traces of white mist.  An East
passage ends here except for a small crack leading on.
Rough stone steps lead down the pit.
>w

The crack is far too small for you to follow.

You're at top of small pit.
Rough stone steps lead down the pit.
>down_
```

Figure 1: Screenshot: Colossal Cave Adventure

Limiting ourselves to standard C++ meant that students would focus on fundamental concepts such as classes, functions, and containers rather than loading a different animated image on every button. Success in CS2 requires a mastery of the syntax and semantics of simple data structures, not a particular GUI or toolkit.

Communicating across the curriculum is important, particularly in a small liberal-arts institution with a required capstone experience. Students have a tendency to compartmentalize educational topics, keeping writing for English classes and oral presentation for public speaking class. Explicitly including them in the projects goals communicates to all instructors how important they are.

Student-designed assignments protects students from plagiarism while still permitting them to talk about their projects. Being able to help one another improves all students' learning. Further, by designing their own content, students are masters of their own destiny; it increases their buy-in to the project and thus their engagement.

A semester-long project could lead to workman-like lectures. To keep up instructor interest a good project would serve as a starting point for introducing a wide variety of topics.

Designing a text adventure game along with a text adventure game *system* to display the game fits these five criteria marvelously. Our success in using student-designed content, in particular, mirrors that reported in in Toothman[26].

Operational History

Current students were born after the golden age of the text adventure game; they have little or no knowledge of *Adventure*, the first text adventure game[5] (see Figure 1). A *text adventure game* is a simulated world in which players use text commands to control their character's actions. Input and output limited to text fit very well with the capabilities of early home computers and programming languages. Purely text adventure games enjoyed a commercial heyday through the early 1980's. Companies such as Infocom, Level 9, Magnetic Scrolls, and Adventure International enjoyed financial success with games like *Zork*, *The Hitchhiker's Guide to the Galaxy*, and *Amnesia*. As the power of home computers grew, consumers and producers of computer entertainment began to move from text-based interactive stories to more and more graphically intensive games.

As the commercial viability of text adventure games waned, a crop of community-produced *interactive fiction* rose. Free interactive fiction authoring kits were developed and deployed including *Inform* and *TADS*. Over the past decade the quality of interactive fiction produced by the on-line community has surpassed the best commercial games and the Interactive Fiction Archive [7] holds the winners of the **Interactive Fiction Awards** (for winners of *the Comp*, an annual competition for short works) and the **XYZZY Awards**.

It's All Writing

Students are introduced to the field of interactive fiction and then sent off to select and play a game. They must write up their experience and also give a short (5 minute) presentation on the plot and interface of their game. This acquaints the class with the conventions of text adventure games much more broadly than playing just their one game.

The course project is about constructing both a *game* and a *game engine* in parallel. This introduces students to the idea of data driven programming and permits each type of writing to support and reinforce the other.

Data driven programming, the idea of creating an "engine" with performance that is changeable through the use of different data files, is something that many students have been exposed to without even realizing it. Many have played game "mods" or modifications which demonstrate that the games are data driven.

They are, in many ways, amazed that they, too, can write games that can be modded.

The idea that programming is just a specialize form of writing has been explored by the author (see Ladd[11] and its references) and used to good effect. The successful translation of writing assignments to programming assignments and the utilization of students' familiarity with writing essays to enhance beginning programming skills led to thoughts about how writing a game and writing a game engine are similar.

Consider the execution of a single-threaded program. The point of control begins in **`main()`**and "visits" lines sequentially unless directed otherwise. Thus an executing program can be compared directly to a single player moving into and through the world of a text adventure game. So designing a program is akin to designing a game; designing a method is akin to designing a room; the calling stack is akin to the history of path from where the avatar is back to the beginning of the game. These similarities are explained as each portion of the game project unfolds. This permits students to see that groups of nodes can provide a useful abstraction (so that they can talk about "the mansion" without having to specify exactly which nodes are in it) just as defining classes can provide abstraction for groups of methods.

In the classroom (and on-line discussions with students) the author has seen the light come on for some struggling students when the explanation from the "other" strand of writing is given to them. Work to tease out the exact relationship between the two design realms remains for another day.

4. PROJECT-BASED CS1.5

CS1.5 begins with an introduction to C++ (CS1 is taught in Java). Thus it begins with a series of review exercises to teach students the new syntax. The remainder of the semester uses the text adventure game. (Portions of this section are modeled on Section 4 of [13].)

Review Assignments

Prior to settling on a text adventure game project, review assignments were *ad hoc*, designed by each instructor as the semester began. This meant that there was sometimes a disconnect between the review exercises and the project.

When the text adventure project was adopted, a set of review assignments were designed that take students from zero to being able to parse a string into a vector of strings. This assignment is similar to that found in Koenig and Moo's fifth chapter[10]. The final review assignment is to read files containing attribute=value pairs. These simple files then serve as the basis for all configuration files for the text adventure game engine.

Phases of Adventure

Introduction to Interactive Fiction

The first day of class students are introduced to the genre of text adventure games. As described above they are guided to the Interactive Fiction Archive and to the ***Comp*** winners in particular. The ***Comp*** entrants are designed to be finished in two hours by experienced judges; the short story lines, limited number of puzzles, and generally high quality make these very good introductions to interactive fiction for the students.

At the end of two weeks, most students have completed their game and give a good presentation. This phase focuses on communication and engaging students; it runs in parallel with the review assignments. Starting with the history of *Adventure* leads to discussions of the history of games, history of the Internet, and even a history of computers.

Game Design

Students hear about all the games played in the class and then the design their own. While designing, students read Nelson's *The Craft of Adventure*[16]. This document takes them inside the head of an author of great interactive fiction and a great interactive fiction *system*. It provides students with insight into the job of a game designer.

The design assignment specifies as little as possible so that students make their creative contribution to the assignment at this stage. They select *who* the player is, *what* the goal of the game is, and *how* the avatar interacts with the game world and its inhabitants (combat, conversation, other). This last bit is crucial. With a decade of experience in using digital game assignments in CS1-CS2, the author has observed that gratuitous violence turns women away from computer science; research bears this out[9]. This has led us to give students as much leeway as possible to determine the level and types of conflict in their games. Research also suggests that customization has positive effects on student outcomes[23].

This assignment now culminates in both a written and oral presentation of the design. The oral presentation, added after the first time the course was taught, has greatly improved student engagement and made the entire class aware of all the different games.

This assignment mixes customization of the game with communications skills practice. Topics presented during this phase include the difference between a "choose your own adventure" book where there is no memory of previous actions and a good adventure game where choices have consequences.

This is also where faculty guidance is most important. Student game designs are typically far too ambitious. Rapid turn around is necessary to help students scale-back their designs and move some of the harder ideas into the "Extended Game" phase at the end of the semester.

Location Class

Students design and code their **`Location`** class. They also build a temporary driver program that loads 10% of their game world. Though it is never described in these terms, the locations of a text adventure game are the nodes of a digraph. Students must determine how to serialize this structure in a self-describing text file (as per Bentley[3]); the file format must be field-order-independent with support for comments.

This is the first coding phase, the first trip from design to code and back again. They learn that designs are not static. They are also introduced to the Factory design pattern (though we do not use that terminology at this point). This phase teaches and uses standard C++. It is a perfect opportunity to discuss what serialization is and how data driven programs work.

Detailed Game Design

The game design seldom survives the first engagement with the code. Using the guidance from the instructor students fill in blanks in their game, specifying the gameplay and attributes for critters and items.

While working on this design, students are introduced to different views of games. Lectures and readings from Salen and Zimmerman's *Rules of Play*[22] present games as simulations, entertainment, and stories. Monfort's analysis of text adventure games is also presented at this point.

Students played text adventure games and then stepped back to present what made them fun or not fun; this is the first step into game design. While this assignment focuses on learning C++, the "fun factor" in the game is critical to engagement so students must learn something about game design. This phase also works on written communication skills.

Game Class

As the next phase begins, students are presented with a suggested architecture for the game engine. Presenting a suggested architecture pushes all of the programs to have similar structure; this runs counter to the idea of student customized project. This compromise is a necessary evil: CS1.5students lack the design maturity to create a moderately complex multi-class solution.

Students construct the **`Game`** class, a Singleton that brings together all of the resources for a game. This replaces the driver program from the Location phase. It leads to the discussion of the "game loop" which displays the game state, processes user input, and updates the game state. Students are sometimes surprised to find that this loop is at the core of all interactive programs.

Critter, Item, Inventory, and Playable Game

At the end of the Game phase, students have a navigable map without any interaction. It would be possible to make the development of a playable game a single phase yet with five weeks to implement three major classes (**`Critter`**, **`Item`**, **`Inventory`**) internal milestones serve to motivate students and keep them on track.

The **`Inventory`** class is the one non-STL container constructed during the class. Students learn about linked lists in this course and don't really understand it until they have implemented it for themselves. Students practice using pointers, reading different kinds of configuration files, and implement all additional commands designed in previous phases.

Extended Game

At the end of the previous phases students have a working text adventure game. As with the grading described in Becker [2], a working text adventure game with locations, items, and critters, constitutes a threshold. Completing it by the end of the semester (while it is "due" two weeks *before* the end of the semester) is worth 2.5/4.0. The final phase of the project, due at the end of the semester, is to extend the game using some technique presented in class: inheritance, templates, conversation trees (really just ``mini-maps" with different labels on the movement directions), combat, saving/restoring games in progress, etc.

We also present MS Windows specific GUI programming at this point and offer students the chance to extend their project by adding an MFC-based GUI to it. The change from having control of the game loop to an event-driven program is a real challenge but many students want to have a "pretty" program when they are done. Student feedback on this led us to bend our rule to use only standard C++.

During the final exam period, all of the students return to the classroom for one last time, each having prepared a slightly longer presentation on their completed game. They highlight the extensions they completed and lament the extensions they could not get working. These presentations are well received by everyone in the class. They are the capstone on the course, showing students just how far they have all come.

5. EVALUATION

Ours is a small liberal arts college with a long history in computer science and a short history with a computer science major. The CS1.5 course is designed to help smooth the learning curve for new students while permitting staffing with our two tenure-track computer scientists. Developing a text adventure game assignment is a lot of work; writing both a text adventure game and a game engine in one semester is a whole lot of work. Is this assignment worth staff and student investments?

The assignment has fulfilled its five primary requirements very, very well. Except for the optional GUI extension, student game engines use nothing but standard C++. Written and oral communication assignments are built into the assignment and enhance student engagement when the present both their game design and their final game. Student game designs are not always prefect but they do increase student commitment to their project and again, enhance engagement. The game motivates discussions of computer history, interactive programs, software design patterns, and an oblique introduction to the graph data structure.

Student engagement is obviously improved. Graduating seniors have been known to talk about their text adventure game in front of their parents or siblings. They also talk about them at departmental social events, commiserating with sophomores who are taking the course and mentioning that most programs they write in later courses had parts they had first written in their text adventure game.

Students who finish CS1.5 are obviously prepared and motivated to go on to CS2; in four years only one student has chosen not to go on (a greatly improved retention rate) and only one who went on failed CS2. Anecdotally, poorer students have done better with this assignment than before. Another measure of our success is anonymous student evaluations. Comparing semesters with the text adventure game with the authors evaluations in the same course *before* the text adventure game, every measure of student satisfaction is better. The only negative in the student evaluations is student perception of workload in the course.

This project is large even with high-level architecture and a great deal of code presented in class, students work very hard in this class. Since this serves as an introduction to software engineering in our curriculum, this is currently acceptable. Student buy-in to the project is the only reason they will work this hard.

6. CONCLUSION

Four successful years of using the text adventure game assignment shows that it has met its goals. Student-designed content increases student motivation. The use of threshold grading also motivates them with a feeling of control over their academic outcome. The use of simple text I/O helps students focus on the fundamentals.

The author hopes, in future courses, to extend this success in several ways. The addition of more software engineering tools such as on-line discussion boards, wikis, and software version control will enhance the introduction to software engineering. It will also provide useful base knowledge for later courses. This assignment might also lend itself to the use of agile development methods. Finally, the addition of more game design and game studies material would help students place their creations in a broader context and give them a deeper understanding of games so they can decide whether or not to take our upper-level Computer Games and Simulations course.

The use of a project-based CS1.5 has proven its usefulness over the years. The text adventure game project has, also, proven its worth. It has met all of our pedagogical goals *and* eased the preparatory burden for teaching CS1.5 while still keeping the project fresh for both the Atari and the Nintendo generations.

7. ACKNOWLEDGMENTS

The author wishes to thank Dr. Collen Knickerbocker for the idea of using text adventure games in introductory courses as well as his invaluable support of my tenure case.

8. REFERENCES

[1] Bayliss, J. D. and Strout, S. 2006. Games as a "flavor" of CS1. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education* (Houston, Texas, USA, March 03 - 05, 2006). SIGCSE '06. ACM Press, New York, NY, 500-504.

[2] K. Becker. Teaching with games: the minesweeper and asteroids experience. In *The J. of Computing Small Coll.*, 17(2):23–33, 2001.

[3] J. Bentley. *More Programming Pearls*. Addison Wesley Publishing, Reading, MA, USA, 1988.

[4] Bergin, J., Roberts, J., Pattis, R., and Stehlik, M. 1996 *Karel++: a Gentle Introduction to the Art of Object-Oriented Programming*. 1st. John Wiley & Sons, Inc.

[5] W. Crowther, D. Woods, and K. Black. *Colossal cave adventure.* ftp://ftp.gmd.de/if-archive/games/pc/adv350kb.zip. MS-DOS Executable.

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

[7] Interactive fiction archive. http://ifarchive.org/.

[8] D. Jerz. *Interactive fiction: An introduction to scholarship*. http://jerz.setonhill.edu/if/bibliography/intro.htm, August 2001.

[9] S. Kiesler, L. Sproull, and J. S. Eccles. Pool halls, chips, and war games: women in the culture of computing. *SIGCSE Bull.*, 34(2):159–164, 2002.

[10] Koenig, A. and Moo, B. E. 2000 *Accelerated C++: Practical Programming by Example*. Addison-Wesley Longman Publishing Co., Inc.

[11] Ladd, B. C. 2003. It's all writing: experience using rewriting to learn in introductory computer science. *J. Comput. Small Coll.* 18, 5 (May. 2003), 57-64.

[12] Ladd, B. and Harcourt, E. 2005. Student competitions and bots in an introductory programming course. *J. Comput. Small Coll.* 20, 5 (May. 2005), 274-284.

[13] Ladd, B. C. 2006. The curse of Monkey Island: holding the attention of students weaned on computer games. *J. Comput. Small Coll.* 21, 6 (Jun. 2006), 162-174.

[14] N. Monfort. *Twisty Little Passages: An Approach to Interactive Fiction*. MIT Press, Cambridge, MA, USA, 2003.

[15] R. Moser. A fantasy adventure game as a learning environment: why learning to program is so difficult and what can be done about it. In *ITiCSE '97: Proceedings of the 2nd conference on Integrating technology into computer science education*, pages 114–116, New York, NY, USA, 1997. ACM Press.

[16] G. Nelson. *The craft of adventure*. Web, http://www.ifarchive.org/ifarchive/info/Craft.Of.Adventure.T1.letter.pdf, 1995.

[17] D. Oblinger and J. Oblinger, editors. *Educating the Net Generation*. Educause, June 2005. http://www.educause.edu/educatingthenetgen/.

[18] Parberry, I., Roden, T., and Kazemzadeh, M. B. 2005. Experience with an industry-driven capstone course on game programming: extended abstract. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education* (St. Louis, Missouri, USA, February 23 - 27, 2005). SIGCSE '05. ACM Press, New York, NY, 91-95.

[19] Parberry, I., Kazemzadeh, M. B., and Roden, T. 2006. The art and science of game programming. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education* (Houston, Texas, USA, March 03 - 05, 2006). SIGCSE '06. ACM Press, New York, NY,

[20] Pattis, R. E. 1981 *Karel the Robot: a Gentle Introduction to the Art of Programming*. 1st. John Wiley & Sons, Inc.

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

[22] K. Salen and E. Zimmerman. *Rules of Play : Game Design Fundamentals*. MIT Press, Cambridge, MA, USA, October 2003.

[23] G. Sindre, S. Line, and O. V. Valvåg. Positive experiences with an open project assignment in an introductory programming course. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 608–613, Washington, DC, USA, 2003. IEEE Computer Society.

[24] E. Soloway. How the nintendo generation learns. *Commun. ACM*, 34(9):23–ff.,1991.

[25] S. Summit. Intermediate c programming assignments. http://www.eskimo.com/scs/cclass/asgn.int/index.html, 1999.

[26] B. Toothman and R. Shackelford. The effects of partially-individualized assignments on subsequent student performance. In *SIGCSE '98: Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, pages 287–291, New York, NY, USA, 1998. ACM Press.

[2 7] Wolz, U., Barnes, T., Parberry, I., and Wick, M. 2006. Digital gaming as a vehicle for learning. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education* (Houston, Texas, USA, March 03 - 05, 2006). SIGCSE '06. ACM Press, New York, NY, 394-395.

Using XNA-GSE game segments to Engage Students in Advanced Computer Science Education

G. Michael Youngblood
The University of North Carolina at Charlotte
College of Computing and Informatics, Dept. of Computer Science
Charlotte, NC 28223
youngbld@uncc.edu

ABSTRACT

This paper presents the notion of using *game segments* which are developed solution packs providing the full code for a segment of a game with a clear element left for implementation by a student. In conjunction with a course teaching targeted principles and techniques, *game segments* provide reinforcing homework assignments. Seven guiding principles for design, and five example *game segments* used in an upper-division undergraduate/mid-level graduate class are described and lessons learned discussed. The *game segments* are targeted at XBOX 360 deployment using Microsoft's XNA Game Studio Express for the online XNA Creators Club.

Keywords

Artificial Intelligence Games, Games in Computer Science Education, game segments

1. INTRODUCTION

Since the introduction of the first game console computer science students and game enthusiasts have desired to write programs and games for these machines. There is a large homebrew movement based on hacking and writing programs for game consoles; however, the legality of these endeavors is at a minimum questionable since they often infringe on security protections and use unreleased development tools (sources not provided to protect those involved, but they can be easily found on the Internet). In particular, the XBOX—since it is based on common PC hardware—has been a popular platform to modify. To their credit, Microsoft Corporation has realized the desire by many current and aspiring developers to develop games and applications on the latest generation of console platforms—the XBOX 360[1].

In August 2006 Microsoft released the first beta of the XNA (originally an acronym for XBOX Network Architecture, but now symbolizing a game development suite of tools) Game Studio Express (GSE) [9, 8]. GSE is built upon the freely available Visual C# 2005 Express Integrated Development Environment (IDE) [6]. This is a scaled down version of the professional Microsoft Visual Studio IDE and provides a similar professional feeling development platform. Using the XNA GSE, anyone can develop games for the Windows platform and starting on December 11, 2006, they will be able to deploy their games on a special developer version of XBOX Live called the *XNA Creators Club* ($99/year subscription fee) for the XBOX 360 [7]. This product and system for XBOX 360 signifies the first time that it will be possible for anyone with the desire, a little money, and modest access to a PC to develop for a game console platform.

Computer science students who, like most individuals, spend a large part of their entertainment time playing computer games are particularly interested in game development for a game console. This is unchartered territory for many students and represents the cutting edge that many students seek. Using XNA GSE also satisfies their desire to use tools similar to those they will use in industry and many report that when using professional quality IDEs that they feel their work is more relevant and closer to what professionals do.

Utilizing these student interests, I incorporated XNA GSE into my upper-division undergraduate/mid-level graduate *Artificial Intelligence for Interactive Computer Games* class (Game AI) this Fall (2006) at the University of North Carolina at Charlotte (UNCC). The goals of this paper are to share my experiences in developing challenging homework assignments for the class, provide a number of guiding design principles, relate my observations from the student work, and overall forward my ideas of creating *game segments* for use in computer science education.

This course was originally designed to delve deep into the specifics of agent-based AI for controlling the environment, characters, and elements specifically for interactive computer games. Since it was a first-time offering at UNCC and the prerequisites were open, all of the students that signed up for the class had no prior AI classes and only 2 had considered taking an AI class before this course was offered. From the onset, the game aspect had already drawn in 82% of the class to take a course of study that would not have prior—introducing them to the field of artificial intelligence. However, this class was originally intended to be a more advanced agents class, but because of the lack of prior AI experience this class ended up providing more of an introduction to AI—so, I strongly believe that everything discussed in this paper is equally applicable to an *Introduction to Artificial Intelligence* class. I also believe that games could also be applied to most upper-level and masters specialty classes (e.g., networking, computer graphics, databases) in a similar manner.

2. GAMES IN CS EDUCATION

Using games incorporated into computer science education is by

[1]XBOX and XBOX 360 are registered trademarks of Microsoft Corporation

no means something new or novel, but most of the focus has been in introducing game development classes, incorporation into the CS1 and CS2 curriculum for lower-division undergraduates, or allowing a game as a capstone project [15, 10, 4]. Games are a popular topic in education at the moment.

This paper specifically discusses games as applied to homework assignments in an artificial intelligence class for the upper-division undergraduate/mid-level graduate student providing another data point of game incorporation into collegiate education. It does not promote the use of a single game adapted through the semester as in [4], but rather a different mini-game or *game segment* for each assignment providing a focused and relevant assignment for the desired learning task.

3. GAME SEGMENTS

One of the biggest problems facing students with any assignment is how to get started. In assignments where the object is to apply the learned concept from class to a computer game, the student needs to be isolated from everything that does not pertain to the knowledge they are to apply. This means that they should not program any aspect of the game not related to the specific objective. The method that I use to focus the students starts with creating project scaffolding for what I call a *game segment*. A *game segment* is not a full game, although it could be, but merely a section of a game where it would be feasible to apply the technique that the student should program into the game. It should make sense from a broader perspective that a game segment could fit into a larger setting becoming a part of a much larger game—this helps engage the student and reduces the burden for developing scaffolding. All of the game segment elements that do not pertain to the learning objective should be programmed and the areas where the student is to insert their application of the desired technique should be clearly indicated. Getting started is not so scary to the student now since they can build the project (which should initially compile and run without their additions) and have clear areas for them to contribute.

The game segments should include the following elements:

1. Clear and concise assignment instructions
2. A solution pack for XNA Game Studio Express or similar construct using other middleware with the following:
   (a) Project file that maps all content into the IDE
   (b) Well-documented object-oriented code files making the game
   (c) A working solution or video/class demonstration to illustrate the desired action of the game when properly developed
   (d) Art and audio assets to support the game
   (e) Any data files necessary
   (f) Development notes
   (g) Clearly defined areas for students to perform their programming tasks
3. Source material for any incorporated elements (e.g., code libraries)

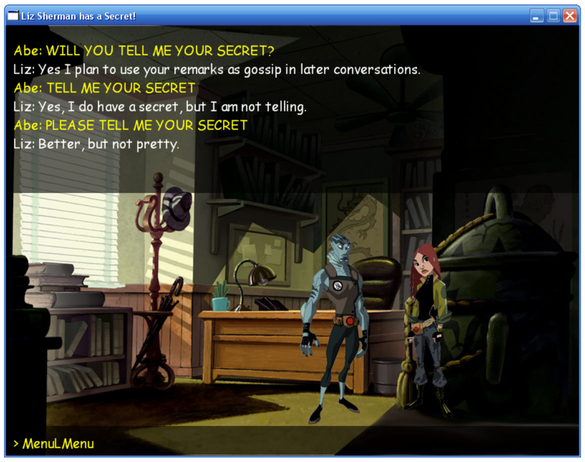
The key idea is to get the student to focus on the main learning objective—the application of the learned technique and not all of the other aspects of the game. However, students do also desire to adjust the parameters of the game, examine the code to understand it, and change the art and sound assets to evolve the game to their own liking. Game segments should support these activities and may incorporate them into the assignment.

In the Game AI class, five home work assignments in the form of *game segments* were developed. At the time of the writing of this paper only four have been currently assigned, but we can examine the basic elements of each to gain a better understanding of what they provide and what they test. All of these game segments are designed for XBOX 360 deployment and make use of the XBOX 360 game controller which can also be used on a PC. In some cases the computer keyboard is used, which is consistent with XBOX 360 deployment since a USB-based keyboard can be plugged in and will work with the XBOX 360.

3.1 Chat Bot

The first *game segment* assignment is an exercise on developing a reflex agent without state through the creation of a conversation agent. This *chat bot* will engage in discourse and interact via the screen with a character controlled by the player using game controller for movement and the keyboard for text entry. Wallace's AIML (Artificial Intelligence Markup Language) [13],used in ALICE [14] and winner of the Loebner Bronze Prize [5], is used in conjunction with Nick Tollervey's open source C# AIML engine [2].

The student only needs to create the AIML (developed in XML) file for the chat bot and the bot profile that creates some background for the agent. The AIML establishes a set of conversation patterns and corresponding responses. As an exercise in state overlap, the students were also asked to include the default knowledge base and two others of their choosing from a set of 28 provided. The conflict from their conversation elements with ones from the incorporated conversation knowledge base is inevitable and requires the student to explore and modify all of the knowledge bases used in order to create a seemingly intelligent agent.



Liz Sherman has a Secret!

Abe: WILL YOU TELL ME YOUR SECRET?
Liz: Yes I plan to use your remarks as gossip in later conversations.
Abe: TELL ME YOUR SECRET
Liz: Yes, I do have a secret, but I am not telling.
Abe: PLEASE TELL ME YOUR SECRET
Liz: Better, but not pretty.

> MenuLMenu

Figure 1: *Liz Sherman has a Secret* game segment.

The example provided with the base code is called "Liz Sherman[2] has a Secret". This involves two characters, Abe Sapien[3] (played by the human player) and Liz Sherman, in which Abe is supposed to determine Liz's secret (i.e., that she is a firestarter) through verbal discourse as shown in Figure 1

In order to promote more interest and engagement of the class with the exercise, they were also tasked to exchange the charac-

[2]After the Mike Mignola character from the Hellboy oeuvre.
[3]Ibid.

ter and background images in order to change the premise of the game which is summarized in the following statement: *You are [the]* {*blank*} *of* {*blank*}*, and your job is to ask* {*blank*} *about* {*blank*} *in order to* {*blank*}.

3.2 Motion Planning

The second *game segment* assignment is an exercise on developing a reflex agent using state and planning in a three phase exercise involving a foraging task. The agent takes the form of a chipmunk, Zippy, in a 32×24 discretized field with a river as shown in Figure 2 in which he must find an acorn and then return to his burrow.

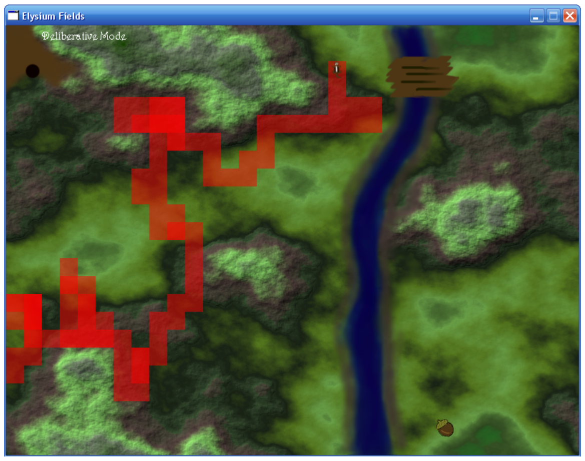


Figure 2: *Elysium Fields* game segment.

The game segment has been setup to accomodate an agent chipmunk with a percept interface and a control interface in two modes. The percept interfaces allows the agent to *see* in a 5×5 grid centered around the agent. Separate percepts for the acorn smell and capture are provided. Acorn *smell* is reported true if the acorn is within a 5 grid radius of the agent. If the acorn is picked up by the agent using the *pickUp* action when the agent and acorn occupy the same grid area, then the *haveAcorn* percept will be true. The agent is also provided two distance sensing sensors when in reactive mode which are placed 2 grid squares forward and diagonal left and right in front of the agent. These sensors relay the Euclidean distance of the acorn to each sensor.

The control interfaces is broken into two modes of operation. In deliberative mode the agent can move in one of five possible ways (moveUp, moveRight, moveDown, moveLeft, burrow). The burrow action can only occur on the entry location to the agent chipmunk's home (a known location) and will allow the agent chipmunk to enter its home. The second mode involves agent moves of rotateRight, rotateLeft, moveForward, moveReverse, and pickUp for reactive control of the agent chipmunk.

The goal of this game segment is to locate the single acorn in the level map, pick it up, and then return to the chipmunks burrow with the acorn. Successful entry into the chipmunks home with the acorn results in a *win*. Goal execution is performed in 3 distinct phases:

1. The first phase is for the agent to explore the map using a search method and one of the following collision avoidance methods: Contour tracing, Collision avoidance tracks, Waypoints from the centroids of decomposed cells, or Voronoi pathways or other potential field methods other than pure gradient [3, 12].
2. The second phase is initiated upon smelling the acorn and consists of a shift from a deliberative search to a reactive *focus and grab* behavior. The agent shall focus on the acorn and move close to the acorn center and issue a pickUp command. Capture of the acorn signals the end of the foraging task and the beginning of the next phase.
3. The third phase is to plan a path and execute traversal from the acorn location to the home burrow location using A* search [12]. When the agent chipmunk enters its home burrow and has the acorn then the game segment is completed.

3.3 Adversarial Search

The third *game segment* assignment is an exercise on developing an adversarial search [12] game segment revisiting the Elysium Fields of *game segment* 2. Using that grid discretized world, consider a game where player one, Zippy the chipmunk who starts in the lower left corner, is trying to cross over the bridge to the land of acorns. Player two is a group of three trees who occupy four grid spaces each in a square shape. The trees are randomly started on the left side of the board and may not cross the bridge. This is a turn-based game where the first player moves one square in either a North, East, South, or West direction. The second player is the three trees who must also all move one square each under the same motion constraints. Player one's goal is to get to the bridge and cross over. Player two's goal is to trap player one so that they cannot move. The rock and water obstacles may not be crossed by either player. The student is to design a Static Board Evaluation (SBE) metric programmed as a method for this game to be used in adversarial search where player one is MAX and player two is MIN [12].

3.4 Neural Networks

The fourth *game segment* assignment is an exercise on training a back propagation neural network [12] to control a game character. This game segment is called *Brainatars Revenge!* It features *Brainatar* in the upper, center of the screen who is trying to stave off numerous apprehenders who approach from one of five directions and take six turns to approach him directly (five slots, touching Brainatar on the sixth). Some of the apprehenders have super powers and travel twice as fast, but are recognized as *supers* during their approach. The apprehenders appear randomly in each vector with only one apprehender allowed in an approach vector at a time. Brainatar can only be touched a maximum of 10 times before defeat. In order to prolong his freedom from his inevitable capture, Brainatar has equipped himself with a laser stun gun which can move one position at a time or fire in this turn-based game. Stunning the apprehenders sends them back to an offscreen pool of individuals.

The exercise for the student is to either play the Brainatar character or write some simple if-then logic (examples provided) to control him successfully enough to capture play data. Then using the generated data, it is split into a training and test set as determined by the student and used to train a C# ported libneural-based [1] back propagation network of the student's encoding to play Brainatar. The trained network can be easily saved and loaded to control the character. A provided timer displays how long Brainatar survives in order to provide a basis for class competition.

3.5 Flocking

The last *game segment* assignment is an exercise on developing flocking and steering behaviors in agents. This game segment is called *Herding Cats*. There are 20 cats running stray in a field that like to travel together in a loose herd and will follow certain other animals. One animal they like to follow is the trained Basenji, Sambuka, who is controlled by the human player. Sambuka must lead the stray cats into a pen where they stay to feast on cat nip.

The students provide the *localized leader following* and *flocking* behaviors [11] for the cat object. If the behaviors are implemented correctly the character motion will appear realistic (maybe not for cats necessarily, but other herding animals) and will allow the player to guide them into the pen. A timer is added to allow competition in herding.

3.6 Availability

All of the *game segments* described in this paper and others in development are compatible with the XNA GSE version 1.0 released on Dec. 11, 2006, and are available for free download at http://serenity.uncc.edu/GameSegments. I also plan to make these available in some form on the XBOX Live XNA Creators Club which will make them available for play on the XBOX 360. These game segments can be incorporated into classes, used as examples, or modified in any way compatible with the *GNU Public License*.

4. LESSONS LEARNED

The *game segments* described in this paper have been used in the Game AI class previously mentioned up through assignment four which is due near the time of writing this paper, but the student questions on this assignment also provide some insight. Throughout the design, development, assignment as homework, teaching, and grading of these game segments a number of **design principles** have become evident and guided this process.

My first experience with teaching a game-related course was at The University of Texas at Arlington (UTA) where I assisted as a co-instructor in an *Introduction to Software Engineering for Computer Games* course. This was the first course of its kind offered at UTA, and as with most first-time courses had a few bumps along the way—the most noticeable being that it takes some time to develop a game, and if you want to develop several games in a semester the basics needed to be provided. There are options for middleware that greatly simplify game-making, but most do not provide the professional level IDE or full expandability. They are also usually limited in deployment to a single platform (the development one) and certainly not to a game console. From these lessons learned, I knew that XNA GSE offered the right platform, but in order to focus on learning objectives the games would need to be completed to the point that the only elements remaining were the programming artifacts to be provided by the students. When I first developed the notion of putting together *game segments* to scaffold the student's work and have them focus on the application lessons and not necessarily the game two initial rules guided me. These were the following:

Design Principle 1:. *The areas where students are to modify, incorporate their code, or examine for debugging should be very clear in the game segment.*

Design Principle 2:. *Allow the student to easily change the cosmetics and/or behavior of the game so they can customize it and take ownership.*

DP1 (Design Principle 1) was simply my main goal, but in discussions with many game students it became clear to me that they wanted to be able to make the games their own. In examining the art asset incorporation in XNA GSE, it became clear that *DP2* was possible by using simple graphics with clear directions for replacement. If designed correctly changing the entire game should be simple and not time consuming. This principle was tested in game segment 1 with great success as can be seen in Figure 3 where the students focused on the chat bot AI, but easily modified the segment into a character setting of their own. All of the game segment's art assets can be easily changed creating an entirely different basis for each.

While developing the game segments I was often pressed for time in getting them prepared for the class, so the game segments usually only contain the scaffolding needed to setup the game segment and support the programming assignment. As such, some *user friendly* aspects or improvements are left as an exercise for the students. For example, in game segment 1 which requires a conversation the backspace ability for correction was not implemented. 89% of the students did implement backspace in their assignments (one additional line of code) and spent some time exploring the entire code base. This led to *DP3*, and since students look through the entire code base—*DP4*. The desired element in game segment 2 (and 3) that was left out was the ability of the chipmunk to move diagonally in the planning phase—67% of the students implemented those extra methods though not required.

Design Principle 3:. *Always leave an element of the base code that the students would like to change but is not required to be changed.*

Design Principle 4:. *Document all of the game segment source code well. It serves as an example student will refer to for additional learning.*

When game segment 2 was due many students underestimated the amount of work it would take to accomplish the assignment. Only 44% completed the full assignment. Upon reviewing their work and reimplementing the solution myself, *DP5* became evident. The remainder of the class was given a chance to revisit game Segment 2 for redemption and with another week 100% had completed the assignment.

Design Principle 5:. *Keep the game segment simple by testing only one major application of a learned technique at a time with the addition of at most one minor applied technique.*

While reviewing student code for the game segments, one thing became quite clear—that despite the style and manner that the game segment was coded, the students would use whatever style they were used to for programming and not follow the provided examples. In the case where they would change the entire game many would leave all of the original variable names which usually made no sense in the changed context of the game. Although *DP6* captures an important essence of the game segments, an accompanying (well-matched) coding standard and the requirement to follow it may better help the students—particularly those bound for industry.

Design Principle 6:. *The game segment should encourage the desired proper coding style and paradigm (e.g., object-oriented).*

Throughout the semester of using game segments the students have reported very positive feedback on using them (except for the initial game segment 2, which was too long). They all report that it has helped them understand XNA GSE and C# better than they could have done on their own—only one student was a C# programmer prior to the class. Game segment 2 could be split into two parts giving effectively three game segments from the same base as game segment 3, which was also based on GS2. The students also liked being able to focus on their programming objective while having the control to improve and modify the existing code base. All of the students are involved in a larger class game project of their choosing and are reusing parts of provided game segments even

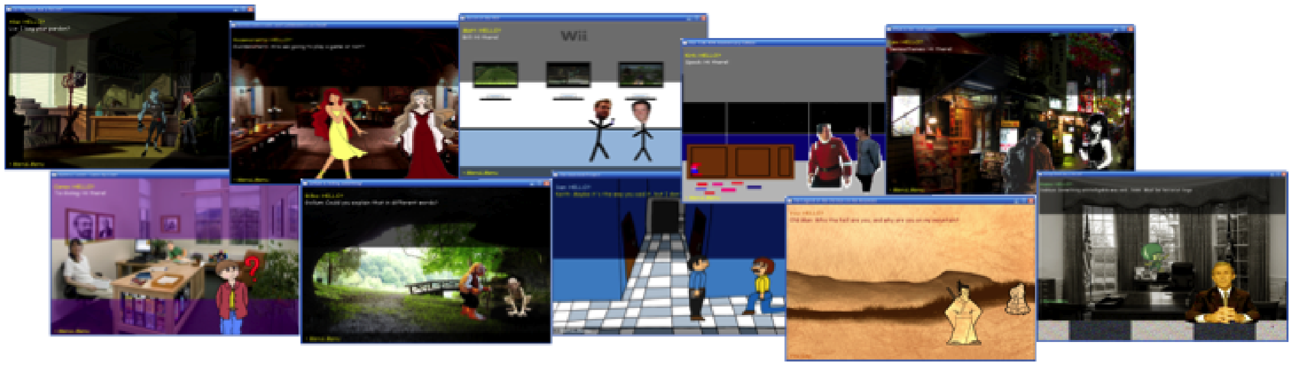


Figure 3: The unique games developed from a game segment. The original, provided game segment is in the upper left.

though not required. One group of students had converted a game segment (#2) into a game tile engine, and another group has started a local game company and used some parts of game segment 1 for their initial prototype.

During the Game AI course we have often paused to share the work of the class and at the student's urgings have setup the ability to have competitions starting with game segment 4. Gaming seems to bring out the competitive nature of the students, but even though they all do not win they seem to be learning and are all enjoying the experience–no bruised egos experienced so far. *DP7* captures the student's desire to share their work with each other and engage in friendly competition.

Design Principle 7:. *Design for competitions or diversity to help show off the student's work in class incorporating it back into the lessons.*

These design principles capture the guiding elements for the game segments I have worked on so far, but as more become evident they will be posted on the website and incorporated into future segments.

5. CONCLUSIONS

While most of the gaming attention in computer science is either focused on the CS1 & 2 curriculums or game design and development courses, their incorporation into advanced computer science courses can help engage students in those classes as well. Students want to work with professional level tools, work with new technologies, and deploy applications to new frontiers. XNA GSE offers them this opportunity using a modern IDE with game deployability to the XBOX 360 platform. Instructors and students both want to focus on the application of learned principles and techniques allowing for some customization and therefore personal ownership but not taking focus away from the primary task. In this paper, I introduce the concept of a *game segment* created using seven *design principles* and successfully used in a Game AI class taught in the Fall of 2006 at UNCC. Using *game segments* allows students to learn and do more in a focused manner.

I plan to incorporate *game segments* into more classes, make more available on the Internet, and further develop design principles for creating them. Their effectiveness as a teaching and learning mechanism needs to be evaluated. As fully tested suites mature for courses and are adopted, their impact on learning will be assessed.

6. REFERENCES

[1] P. Crochat and D. Franklin. libneural home page. http://ieee.uow.edu.au/~daniel/software/libneural/.

[2] N. H.Tollervey. An AIML Chatterbot in C#. http://ntoll.org/article/project-an-aiml-chatterbot-in-c.

[3] A. LaMothe. *Tricks of the Windows Game Programming Gurus*. Sams, Indianapolis, IN, USA, 2002.

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

[5] H. Loebner. Home Page of The Loebner Prize in Artificial Intelligence. http://www.loebner.net/Prizef/loebner-prize.html.

[6] Microsoft Corporation. Microsoft Visual C# 2005 Express Edition. http://msdn.microsoft.com/vstudio/express/visualcsharp/.

[7] Microsoft Corporation. XNA Frequently Asked Questions. http://msdn.microsoft.com/directx/xna/faq/.

[8] Microsoft Corporation. XNA Game Studio Express. http://msdn.microsoft.com/directx/xna/gse/.

[9] Microsoft Corporation. Microsoft Game Technologies Center, Nov 2006. http://msdn.microsoft.com/directx/XNA/default.aspx.

[10] I. Parberry, M. B. Kazemzadeh, and T. 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.

[11] C. Reynolds. Steering behaviors for autonomous characters, 1999. http://www.red3d.com/cwr/papers/1999/gdc99steer.html.

[12] S. Russel and P. Norvig. Artificial intelligence, 2002.

[13] R. Wallace. AIML: Artificial Intelligence Markup Language. http://www.alicebot.org/aiml.html.

[14] R. Wallace. The Anatomy of A.L.I.C.E. http://www.alicebot.org/anatomy.html.

[15] U. Wolz, T. Barnes, I. Parberry, and M. Wick. Digital gaming as a vehicle for learning. In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 394–395, New York, NY, USA, 2006. ACM Press.

Using a simple MMORPG to teach multi-user, client-server database development

Greg Wadley
Department of Information Systems
The University of Melbourne
Australia 3010
+613 8344 1586
greg.wadley@unimelb.edu.au

Jason Sobell
Philology Pty Ltd
111 Barry Street
Carlton, Australia 3053
+613 9349 4735
jason@philology.com.au

ABSTRACT

Applications built for undergraduate programming assignments are typically single-user systems, of which the programmer is also the sole user. Real-world information systems differ from this scenario in a number of ways. In particular, they are usually client-server systems within which many users concurrently access the same data. In order to illustrate for our students the benefits and pitfalls of multi-user systems based around a shared database, we asked them to build a simple massively-multiplayer online role-playing game (MMORPG) which stored game-world and player state in a relational database. We provided students with a graphical client written in Visual Basic. As players moved about the game world, interacting with objects and other players, their client programs called procedures in the central database to update game state accordingly. The students' task was to implement database tables and procedures that allowed the clients to work. The system's client-server architecture resembled that of commercial information systems and often occasioned concurrent access to data. In this paper we describe the system, the students' experience of building it, and our perception of its pedagogical pros and cons.

Categories and Subject Descriptors

K.3.2 [Computing Milieu] Computer and Information Science Education – *Information systems education.*

General Terms

Design, Human Factors, Languages, Theory

Keywords

database, multi-user, education, project, MMORPG, RDBMS

1. INTRODUCTION

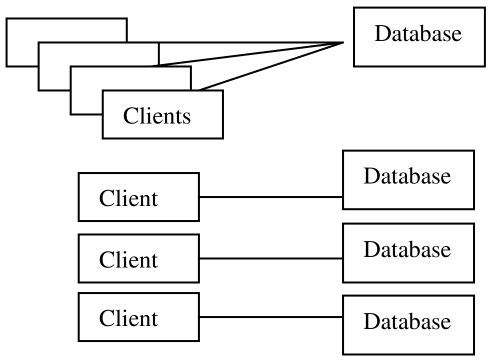
Computer games are used as undergraduate programming assignments for a number of reasons. Many students play games, understand this class of application well, consider games fun to develop and use, and are interested to find out how they work. Games emphasize the user interface of an application, and are well suited to real-time visualization of system state. They can readily illustrate a number of problems from simple program logic to graphics, usability and artificial intelligence. Game development expertise can lead to a career that many students regard highly.

Different genres of games are suited to different courses and competencies. For example, students in introductory programming courses are often asked to build simple puzzle games, while advanced students in game development courses might build 3d games or graphics engines.

The authors teach an advanced database course in a Bachelor of Information Systems degree. Our aim is to expose final-year students, who have completed an introductory course in SQL and E-R modeling, to issues concerning the development, physical implementation and administration of database systems. Lab exercises in our course involve hands-on use of both Oracle and Sql Server database management systems and related tools.

Information systems in industry typically use a client-server architecture, by which many users connect to a central database. Yet projects in database courses are typically single-user systems, of which the student is both the programmer and the only user. While this architecture is easy to program, teach and administer, it obscures the main purpose to which most relational databases are put, which is to store a representation of entities and events that are significant to a group of people, who read and write the data in an ad-hoc way and effectively communicate through it. An understanding of this function of multi-user databases cannot easily be gained by building single-user applications in which the database is simply a convenient disk-based data store.

Figure 1. Typical I.S. architectures used in Industry (above) and Education (below)



Nor do single-user systems readily illustrate the problems, such as lost updates, uncommitted dependencies, and inconsistent reads, that can arise when several users concurrently access the same data. Without experience of these problems it is difficult for students to understand the techniques that database and DBMS designers use to solve them, such as transactions and locks, or the problems that these techniques in turn can cause, such as deadlock.

We reasoned that in order to better understand client-server database programming, our students should build a multi-user system in which an update to the database by any user affected everyone else's view of the data in an obvious way. Taking into consideration also the benefits of using games in programming assignments, we concluded that building a multi-user game which stored the state of the game-world and players in a relational database would be an informative and motivating project.

Our students were not expert GUI programmers and our course does not focus on writing client programs, especially the kind of graphics-intensive clients used in commercial multi-player computer games. Also, because of the limited power of our server, lab computers and network, a fast-paced game was inappropriate. These restrictions excluded some genres of games from consideration, such as team-based 'shoot-em-ups'.

Inspired by the popularity of massively multiplayer role playing games such as World of Warcraft, we decided to use a simple MMORPG as the project theme. To preserve the course's focus on database rather than GUI programming, we wrote a client program using Visual Basic and gave it to students. We provided also a list of the tables and stored procedures that the client would look for in the supporting database. The students' task was to implement these tables and procedures.

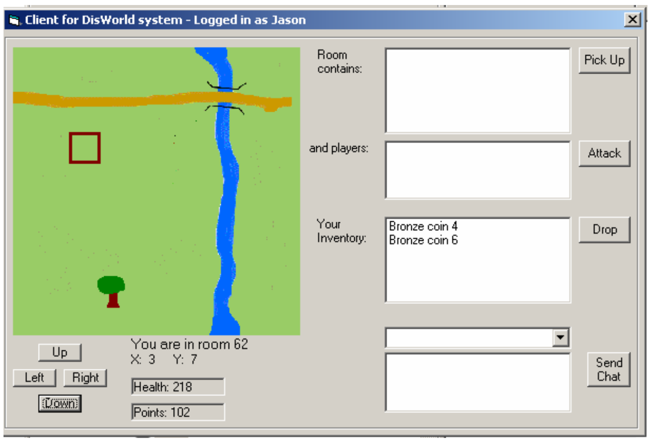
The essence of an MMORPG is that all players act within the same, single virtual world. Each player's actions affect, and are visible to, the other players. Furthermore, the game-world is persistent: world and player state are maintained independently of any particular individual's login sessions. These criteria require properly managed, central, disk-based storage.

We hypothesized that this function of an MMORPG – to maintain a single, persistent representation of a (in this case, virtual) world, which is read and written to by many users – is essentially the same function we wanted to illustrate for RDBMS-based information systems generally.

2. HOW THE GAME WORKED

To simplify the task of programming the client, and to make the game rules and mechanics clear, we designed a simple game that took place in a two-dimensional 10 x 10 'grid world' shaped like a chess board. Players navigated around this grid by moving up, down, left or right, one square ('room') at a time, using buttons on the game client. To illustrate an avatar's location, the client simply highlighted the appropriate square. As well as the players' avatars there were items scattered about the world which players could pick up, hold in their inventory as they moved about, and drop again in a different location. These were displayed to the player in listboxes.

Figure 2. the Game Client (GUI)



Bronze coin 4
Bronze coin 6

You are in room 62
X: 3 Y: 7
Health: 218
Points: 102

Each individual player and item was represented by a row in a table in a central relational database. As the game progressed, the database kept track of players' locations, health and scores, and the locations of items. Location had low resolution, and could be stored as an integer. Items had to be located either in one of the rooms or in a player's inventory. To pick up an item, a player's avatar needed to be located in the same room as the item. The player then pressed a 'pick up' button on his/her game client.

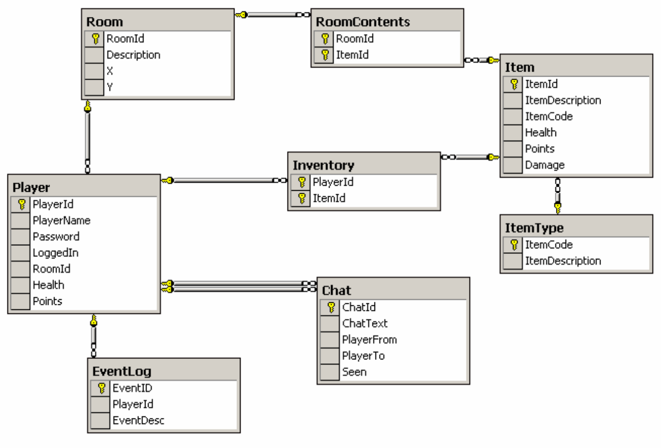
Some items ('weapons') could be used to attack other players. Attacks could only occur when two players occupied the same room. Players accumulated points by picking up items, and lost health when attacked. Health could be restored by finding a health pack. These game actions were effected by the player by pressing a button in the client, triggering execution of a stored procedure which read and wrote data in the database.

Each room in the game-world was represented by a row in a database table. A room was marked 'out of bounds' by omitting that row from the table (we represented non-traversable rooms graphically in the client as a river and a tree). Movement by a player required that the client perform a Select against the database to check whether the desired destination room was traversable, and if it was, to read which items and players were in the room: these were then listed in the client. Finally movement required an Update to change the player's recorded position.

Players could send text messages to each other. The messages were displayed in a list box in the game client. The sending player's client inserted a row in the Chat table, and the receiving player's client eventually selected it for display.

Players had to identify themselves to the game (ie log in) by supplying a name and password already recorded in the Player table. Thereafter a player's game client was able to supply the player's identifier to the database when sending or retrieving data. Following the 'persistent world' approach used in MMORPGs, a player's location and state was preserved between any logout and the next login. Logged-out players did not appear in game clients and could not be attacked. Logins, logouts and other game events were recorded by inserting rows into an event-logging table.

Figure 3. the Database Schema



The database schema is illustrated in figure 3. Several of the kinds of tables typically implemented in relational databases are present in this schema, including:

- *entities* (people, objects, places) - the Player, Item, and Room tables
- *associative entities* (relationships between other entities) – the RoomContents and Inventory tables
- *logged events* - the EventLog and Chat tables
- *lookup tables* – the ItemType table

The individual tables were:

Player: contained one row for each individual person who was registered to play the game (ie students and teachers taking the course). Like a typical 'person' table, it recorded identifying information (a player's name and password), some dynamic properties (location, health, points), and whether the player was currently logged in.

Room: contained one row for each of the game world's rooms (except for non-traversable ones), identified by the integers 1 to 100 and by x-y coordinates. Players and items in the game world were situated in exactly one room at any given time. We programmed the client in such a way that it would be relatively straightforward to re-implement the game with more or fewer rooms.

Item: contained one row for each individual object in the game world. Items were classified into four categories - Points, Health-packs, Weapons and Fixed - using the lookup table *ItemType*. While health items were consumed when picked up, players could carry weapons and points items in an Inventory, implemented as a table pairing items with players. We arbitrarily restricted the size of an inventory to three items, to reduce hoarding by players. Players could not interact with Fixed items. At any given time an Item had to belong either to a player's inventory or to a room. This method of representing items made it relatively straightforward for game administrators to insert new items into the game.

The *Inventory* and *RoomContents* tables associated Items with Players and Rooms respectively.

EventLog: recorded actions by players for (fictitious) auditing purposes. For simplicity we described events with a string of text rather than categorizing them with a lookup table.

Chat: contained one row for each text message sent by a player to another player. When the receiving player's game client fetched a message for display, it marked the message as 'seen' rather than deleting it from the table.

When a player carried out a game action, their game client made one or more calls to the database. The students' task was to enable these calls using stored procedures, according to the specifications listed in table 1. For each procedure we gave students the procedure name, a description of its behaviour, the inputs that a client would provide, and the outputs a client would expect. The client was programmed to call the appropriate procedure in the database when the user pressed a button. The 'getter' procedures were also called every few seconds by a timer in the client, to check whether the room's contents had changed, check incoming messages, and allow the screen to be refreshed.

Table 1. Stored procedures that supported game actions

Procedure	**Inputs**	**Outputs**
spLogin: See if there is a row in Player that matches this name and password. If there is, change that row's LoggedIn field to 1, write a row to the Event table, and return the row from Player. Otherwise, return nothing.	PlayerName, Password	1 row from Player table
spMovePlayerTo: Select from Room to check that desired room exists (is traversable). If it is, update player's location.	PlayerId, X, Y	'Success' = 1 or 0
spGetPlayer: Join Player and Room tables to select all information about this player and current room	PlayerId	Player and room details
spGetItems: Join Item and RoomContents tables to get information about each item in this room.	RoomId	list from Item table
spGetPlayers: Return info about all players in this room, except this player	PlayerId, RoomId	list from Player table
spGetInventory: Join Item and Inventory tables to get information about each item in this player's inventory	PlayerId	list from Item table
spPickUpItem: Delete item from RoomContents and add it to Inventory. Update player's point score.	PlayerId, ItemId	(none)
spDropItem: Delete one row from Inventory, add one row to RoomContents.	PlayerId, ItemId	(none)

spAttack: Check that selected item is a weapon and how much damage it inflicts. Reduce victim's health accordingly.	PlayerId, VictimId, ItemId	Message about how much damage was inflicted
spGetAllPlayers: Return a list of all players in the Player table, to populate chat dropdown.	(none)	list of PlayerId, PlayerName
spAddChat: Write one row to Chat table	Text, PlayerId1, PlayerId2	(none)
spGetChat: Select all chat messages addressed to this player and not yet seen. Mark them as seen.	PlayerId	list of PlayerName, ChatText
spAddEvent: Write one row to Event table	PlayerId, EventDetail	(none)
spLogout: Change this player's LoggedIn field to False.	PlayerId	(none)

3. THE STUDENT EXPERIENCE

We provided each student with a Sql Server database in which to implement his/her tables and procedures. We encouraged students to check the behaviour of their databases by using both the game client and Sql Server *Query Analyzer* to execute procedures. The latter made it easy to display several different SQL statements and their outputs on the same screen, such as 'select *' before and after a procedure execution, which was useful when debugging procedure code.

While each student implemented their own project database, they could, when desired, allow another student's client to log into their database, in order to play their game with the other student.

We also provided a working 'black box' solution to the assignment. This was a database with tables and procedures already implemented by us. It allowed students to play and observe a working version of the game, to better understand their project requirements. We used server permissions to ensure that students' game clients could execute the procedures in this database without being able to read the procedure code or table structures. During a number of lab classes we asked all students to log into the provided database and play a game together. During these sessions we displayed the contents of some tables on a screen in the lab, using the *Enterprise Manager* and *Query Analyzer* tools. Students were able to simultaneously observe their own screen, the screens of other students nearby, and the changing table data on the projection screen, in order to understand how game clients were interacting with and within the shared database.

To submit their work, each student placed their 'create table' scripts and stored procedure code into a text file, and emailed it to the teachers. Code was assessed according to correctness of client behaviour, with bonus marks for elegance and efficiency.

4. WAS THE PROJECT SUCCESSFUL?

In using a multiplayer videogame as a database project our aim was not to teach game design, but to utilize a type of application which we believed undergraduate students would find interesting and intuitive, and which would successfully illustrate the benefits and pitfalls of developing systems based on multi-user access to shared data. Therefore in analyzing whether our project was successful, we need to ask whether students acquired a better understanding of how systems are built around relational databases. That is our 'general' question. We can also analyze our detailed decisions. For example, was the project too easy or hard? Was the database schema too simple or too complex? Was it reasonable to give students a pre-programmed client and ask them to develop the server?

We did not conduct a formal experiment to measure the educational impact of the project. However we can give brief answers based on informal feedback received from students during the project, the University's 'Quality of Teaching' feedback received after the project was over, the work submitted by students, and our subsequent reflections on the course. Overall, we felt that the project was successful in engaging and educating students. However there were some problems, and these are listed below.

4.1 Pros

Our game emphasized user interaction via shared data. The game was multi-user in a way that was easily understandable by students. It emphasized 'computation-as-interaction' over 'computation-as-calculation'; the client-server, networked, interaction-based view that Stein [1] suggests is the best metaphor for understanding modern information technology use.

The project utilized an architecture which is common in business contexts: a client written in Visual Basic, running on a Windows PC, accessing a Sql Server database.

The game client helped students to monitor the changing values stored in the underlying database, realizing some of the pedagogical advantages of visualization [2]. The visual client encouraged students to frequently compare their client screen with the underlying database tables and see more clearly the effect of their procedures on the database.

The system afforded communication among users. Other than tools such as email and instant messaging, it is difficult to think of an application class that lends itself as readily as multiplayer games to having several users communicate through a shared database. The collaborative and visual nature of the system helped to motivate students.

The project emphasized that the core function of an information system is to represent some interesting subset of the world; in particular some entities, their properties and relationships, categorized into classes, which are relevant to a business problem [3]. Although a game-world is fictitious, using a relational database to represent the properties and behaviour of a game world emphasized the representational function of databases.

Some common database project applications (such as order entry) tend not to excite students. The MMORPG was an unusual project for a database course and was more interesting for many students. They were happy to explore the system in class and even after hours. While the system was presumably not as engaging as a commercial MMORPG, its relative simplicity made the underlying mechanisms clear to students curious about how MMORPGs might work.

Although this project did not use a business-oriented theme, many of the design issues exposed here are relevant also to business systems. The most important tables in a business database typically represent classes of entities and events. Event tables tend to become large over time. Miscellaneous tables are needed to implement sub-classes and many-to-many relationships. These phenomena were present in our game database.

In common with many real-world applications, our game had to adequately handle user identity (established through a login) to allow the system to work in a meaningful way, and to allow communication between users.

A number of game actions lent themselves to demonstrating problems of concurrent access to data. For example, to pick up an item required a Delete from the RoomContents table, an Insert to the Inventory table, and an Update of points in the Player table. Getting this code to work correctly when two players simultaneously tried to pick up the same item required careful ordering of these statements, and the use of transactions. Game scenarios such as these may demonstrate problems of concurrency in a more visual and dramatic way than do commonly-used teaching examples such as "transfer funds".

4.2 Cons

While many students are interested in games, some are not. Our assignment was not a typical business application. While this was a plus for some students, many of them were business-focused, and some felt a game to be relevant only to recreation.

There were some problems fine-tuning our approach of giving students a finished client and asking them to implement a database to support it. In typical system development the server would be developed before, or at least in conjunction with, the client. It took some redrafting before we specified the inputs and outputs of the client clearly enough for students to write their procedure code. Without access to client code, it was more difficult for students to debug database code (they could not, for example, display client variables), and students were not exposed to methods of data validation in the client. It is probably better for students to program both client and database in an application: however this was not feasible in our project.

4.3 Relevance to game development courses

The aim of our project was to illustrate the design and development of client-server databases rather than games. Our game-world and game-play were not to commercial standards. We did not intend that the game closely resemble a commercially viable MMORPG. However some comments can be made on the relevance of this project to game development.

It is possible that for performance reasons some commercial MMORPGs do not use a relational database to store game-world and player state. However our research indicated that several do, and at least one open-source MMORPG does so [4]. Our two-tiered architecture was much simpler than the complex multi-tier, multi-server, distributed architectures needed to operate a large-scale MMORPG. However we feel that we captured the essence of multiplayer game design in this project. A course oriented to game design or development might make use of a system like this as a starting point to discuss multiplayer game design.

5. CONCLUSION

We found that development of a relational database to support a simple MMORPG worked well as a project in an advanced database course. Multiplayer games are well-suited to illustrating issues of identity, interaction, and concurrent access to data in multi-user information systems. In this paper we have described the game mechanics, the database tables and stored procedures that our students produced, and our experience as teachers using this project in an undergraduate course.

The game is easy to simplify or expand, allowing it to be easily tailored to different pedagogical goals, course levels and class sizes. For example, students could be assigned a simple version of the game in an introductory database course, and a more complex version in an advanced course. Students in a game design course could be given this game and asked to improve its game play, game world, or interaction design. A web interface for game administration could be added. The authors would be pleased to discuss the application with educators interested in using it in their courses.

6. ACKNOWLEDGMENTS

The authors would like to thank the students who undertook the project, and to Graeme Simsion and Darren Skidmore at the University of Melbourne for reviewing a draft of this paper.

7. REFERENCES

[1] Stein, L. *Challenging the computational metaphor: implications for how we think,* in Cybernetics and Systems 30:473-507, 1999

[2] Hundhausen, C. *Integrating algorithm visualization technology into an undergraduate algorithms course: ethnographic studies of a social constructivist approach,* in Computers and Education, 39:237-260, 2002

[3] Weber, R. *Ontological Foundations of Information Systems,* Coopers and Lybrand, 1997

[4] Riddoch, A. and J. Turner. *Technologies For Building Open-Source Massively Multiplayer Games*, Worldforge.org, 2005

Educating Game Programmers

Timothy E. Roden
Center for Advanced Computer Studies
University of Louisiana at Lafayette
Lafayette, LA, USA
troden@cacs.louisiana.edu

James Etheredge
Department of Computer Science
University of Louisiana at Lafayette
Lafayette, LA, USA
jne1390@louisiana.edu

ABSTRACT

The University of Louisiana at Lafayette is now in its third year offering a game programming concentration within the degree of Bachelor of Science in Computer Science. We describe this new curriculum and our experience so far.

Categories and Subject Descriptors

K.3.2 [**Computing Mileux**]: Computers and Information Science Education [**Computer Science Education**]

General Terms

Design, Experimentation.

Keywords

Game programming, Graphics, Undergraduate education

1. INTRODUCTION

Sales of console and PC games in the U.S. exceeded 7 billion dollars in 2004. Worldwide, total sales of game hardware and software exceeded 23 billion dollars in a market expected to grow at an annual rate of approximately 20%. A record 12 games in 2004 sold more than one million units and 52 console games sold more than 500,000 units. In addition, gaming platforms are increasingly moving away from the traditional single computer or console. In a survey by the Electronic Software Association, nearly half of frequent game players reported playing games online, while over a third used a wireless device, such as a cell phone, to play games [7].

In terms of jobs, the industry is growing commensurate with the market. Game industry jobs represent very lucrative employment opportunities for newly graduating Computer Science majors. A recent survey reflected both the youth and vitality of the industry [5]. More than half of game industry programmers (58%) responding reported less than seven years experience with an average salary of $82,107. For non-lead programmers with less than three years experience, the average salary was $52,989. Lead programmers with less than three years experience averaged $76,848.

Universities across the U.S. have responded in varying measure to the growing video game phenomenon [3,4,6,9,14,15]. Despite the obvious need for trained workers, especially programmers, few models exist that describe a comprehensive undergraduate curriculum tailored especially for those entering the game industry. What follows is a description of the curriculum developed by the Department of Computer Science at the University of Louisiana at Lafayette. The curriculum targets undergraduate computer science majors interested in game design and programming.

2. THE CURRICULUM

The Computer Science Department at the University of Louisiana at Lafayette (ULL) offers an ABET-accredited, well-rounded undergraduate degree. In addition to the core curriculum, the department requires students to choose an area of concentration. The concentrations represent 15 hours of study in one of five areas. These 15 hours include computer science courses as well as relevant courses in other disciplines. The newest concentration area is Video Game Design and Development. Students are required to take two game programming courses, one at the junior level and one at the senior level. For the remaining nine hours students choose among approved electives from Visual Arts, Theatre, English (creative writing), and Communications (production and editing). In addition, students are required to take two Visual Arts courses and a Physics course as part of their Arts and Sciences electives.

In designing the new concentration, there were few academic models to follow. Most universities with computer science game programming courses had only established their courses within the last few years. A notable exception is the game programming curriculum developed by Parberry at University of North Texas (UNT). Parberry was an early pioneer in game programming education, offering the first course in 1993. While not specifically having a concentration area, the UNT Computer Science Department offers two game programming courses and maintains a dedicated game programming laboratory for the students taking these classes [14]. Parberry identifies an important feature of the courses is a close collaboration with the UNT School of Fine Arts. The chief motivation for the content of the courses is based on what the game industry wants in new college graduates. The success of the UNT program is evident by the long list of game companies now employing former UNT students. Another, more recent, example of a two-course undergraduate curriculum within the traditional computer science degree is the recent adoption of such a program at Marist College as reported by Coleman [4].

In looking to industry for advice on creating a game development curriculum, perhaps the best example is the International Game Developer's Association's (IGDA) website [8]. The IGDA publishes a list of topics useful for developing game-related educational courses. It is notable, however, the IGDA does not recommend specific courses despite the first draft of their Curriculum Framework was published in 2002. This is likely due to the lack of curricula with an established track record and the changing nature of game development.

3. THE GAME DEVELOPMENT LAB

We established a dedicated game programming laboratory with 20 PCs. Our reasons for this are similar to Parberry [l4] and include:

1. General access computer labs are not equipped with the hardware and software we need. To develop the caliber of PC games currently marketed to consumers we require the same hardware required by these games.
2. We require graphics cards that are fairly expensive and could not be purchased for general purpose labs.
3. We prefer to have the latest audio cards in our game development computers. While this is not strictly a necessity, as compared to graphics cards, having high quality audio cards opens the possibility for our students to create games with sophisticated audio.
4. We want dual monitors at each workstation. Our students typically create full-screen games. During development games often crash. With one monitor this often results in the crashed game remaining on the screen. Using two monitors, a crashed game can be cleared away without rebooting the computer by accessing the desktop on the second monitor.
5. We want a separate lab where students can both develop and play games.
6. Having students playing games in a general purpose lab would be too distracting for other students.
7. We want to have a single lab where our game development students can meet with other students sharing similar interests.
8. One of our instructors schedules some game development classes in the lab so the lab needs to function as both an open lab and a teaching lab.
9. We want the lab computers on a separate network to facilitate student development of networked games in a controlled setting.
10. We have a comprehensive suite of game development-related software for which we have a limited number of licenses.

The lab is connected with a dedicated 48-port gigabit ethernet switch. A server hosts 1.2 terabytes of disk space for student accounts. We have both color and B/W laser printers, color scanner, digital camcorder, a digital camera and web cameras. The current hardware configuration of each workstation in the lab is listed in Table 1. An ongoing challenge is keeping the hardware updated, given gaming hardware continues to improve rapidly. We estimate the computers need a yearly upgrade consisting of at least a new graphics card. In addition, all the computers will likely need replacement every two to three years.

Table 1. Hardware Installed on Each Workstation

3.2 GHz Intel Pentium 640 Processor
2 GB Dual Channel DDR2 SDRAM, 533 MHz
512 MB PCI Express graphics card
128-voice audio card
160 GB Serial ATA hard disk
16x CD/DVD burner (DVD+/-RW)
Dual 20" LCD monitors
Desktop speakers
Headphones with microphone

The software installed on each workstation is listed in Table 2. In addition we have several copies of FaceGen from Singular Inversions and one copy of Alias MotionBuilder. FaceGen is used to create 3D models of human heads while MotionBuilder is used to generate animation data for 3D models created in Maya. We also have several collections of sound and texture libraries available for use in the lab. We installed the Maya 3D modeling software since the Visual Art Department at ULL uses Maya exclusively and we want art students to use the lab. We also installed Lightwave because we felt it was a simpler program to use and would allow our computer science students to create simple 3D models if needed. To convert 3D models between different file formats, we added a file conversion utility.

Table 2. Software Installed on Each Workstation

Software	Purpose
Microsoft Windows XP	Operating system
Microsoft Office	Word processing, spreadsheet, etc.
Microsoft Visual Studio .NET	Compiler
Adobe Photoshop	2D image editor
Adobe Audition	Sound editor
Adobe Premiere	Video editor
Alias Maya Complete	3D modeling
NewTek Lightwave	3D modeling
SnagIt	Screen capture program
GameMaker	2D game creation editor
Okino NuGraf / PolyTrans for Maya	3D file conversion

Our department system administrator and several graduate students maintain the lab. Our administrator spends an average of three hours per week on lab administration and the graduate students spend an average of ten hours per week working in the lab. We provide lab accounts to both computer science and art students registered in the game courses in addition to students simply interested in learning more about game development. Students with accounts can access the lab via an electronic card reader that scans their student ID card.

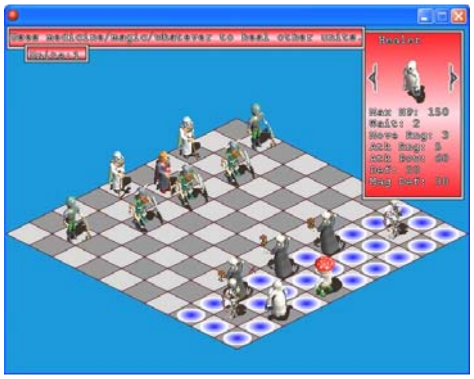


Figure 1. A student-created game from the intro course.

4. INTRODUCTORY COURSE

The first game programming course is a junior level course aimed at students who have successfully completed a sequence of introductory programming courses. We also require students to have completed Calculus I. This requirement is imposed to ensure an appropriate level of mathematical maturity that we have found to be a good predictor of success in programming-intensive courses. An example of a game created by a student in the introductory course is shown in Figure 1.

4.1 Course Content

Since our first game programming course is aimed at juniors, we focus on 2D game development. Limiting the first course to 2D game programming offers several distinct advantages:

1. The task of learning the material is more manageable.
2. Students are able to begin developing viable games very early in the course.
3. The concepts underlying game engines are easily demonstrated using object-oriented 2D game engines.
4. Many concepts encountered in 3D video game development have a 2D counterpart that is simpler to understand. Collision detection is a good example of this.
5. Most of the software needed is readily available.

The content of the 2D game programming course has the following three components. First, there is a design component. Students are exposed to the methodologies and issues involved in the design of games. Topics include game scope, playability, level design, artificial intelligence, art, sound, and design documentation. These topics are presented as in-class discussions based on material in a game design text. Throughout the course, students are taught game design principles via lectures and in-class discussions. The textbook used for the course includes game design document examples and a discussion of each element of the design [2]. The text also presents each major type of game genre and explains the unique aspects and design issues associated with it.

The second component of the course is the rapid development of complete 2D games using a high-level game development authoring tool. The tool we use is Game Maker [13]. It uses an object-oriented approach to creating games. Levels, sprites, characters, sounds, music, and control structures are defined as objects. The creation, destruction, behavior, and interaction of objects are defined within the context of the game. Other features such as timing, scoring, level change, game start and game end conditions, and particle systems, can be easily incorporated into the game. Game Maker provides an integrated development environment to allow development, testing, and the creation of an executable version of the game. Students use Game Maker to develop two games during the course. One game is done individually and the other is developed as a team effort. All games are required to have supporting design documents.

The final component is the development and use of a simple game engine written in C++ using the Microsoft DirectX 9 Software Development Kit (SDK). The game engine is the subject of another required text by Jones [10]. The text also includes 3D topics such as 3D modeling, texture mapping and 3D lighting. These topics prepare students for the advanced course.

4.2 Lessons Learned

Aside from determining the overall design of the game, the hardest part of game development is the creation of the artwork and game audio (sound effects and music). While it is relatively easy to find a great deal of art and sound effects on the Internet, it is not easy to find good quality art assets that are in the public domain.

We found it an advantage to get students creating games as soon as possible. It provides students with a great feeling of accomplishment and keeps them enthusiastic. It also allows them to start developing a portfolio that will be invaluable when they begin to look for a job after graduation.

We gave students time to play games during class. Students are interested in game development because they are interested in playing games. We wanted to maintain that enthusiasm. We alternated between playing the games and looking at the underlying code. Another way to keep the course enjoyable is to let the students play each other's games in class. We were impressed at how good students are at evaluating other student's games and how much they enjoyed seeing what others had accomplished.

We let the students know that any form of vulgarity or excessive violence will not be tolerated. This is stated in the course syllabus including a clear penalty for failure to comply. We placed a requirement that all games developed in the class had to be rated PG. The students offered a compromise rating of PG-13, which we accepted.

We gave the students time to complete projects. There is an enormous time commitment involved in designing and implementing a game. Even taking into account the Game Maker software and working in teams, for most of the students this course is the most time-consuming course they have had to date.

5. ADVANCED COURSE

The advanced game programming course is for computer science students who have completed the introductory game course. The course is also open to upper level undergraduate art students who have completed three to four semesters of computer animation.

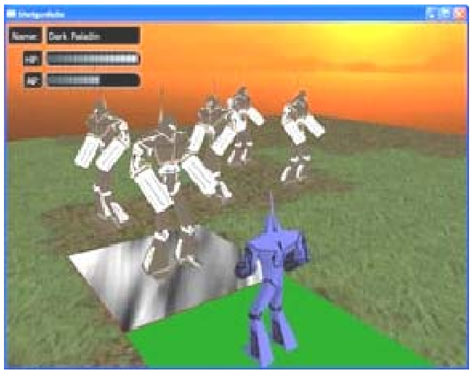


Figure 2. A student-created game from the advanced course.

The course is project-oriented with students working in teams of two to three programmers plus one or two artists to develop a 3D game over the course of the semester. A team-based semester game project has also been used by Parberry [14] and Sindre [16]. An example of a game created by a student in the advanced course is shown in Figure 2.

5.1 Course Overview

There are three texts: one required book for programmers [11], a second recommended book [10], and a required book for artists [12] that the instructor also recommends for programmers. The class meets once per week for three hours of lecture. Generally, the first hour of class is devoted to art issues with the remaining two hours covering programming. Multiple topics may be covered during one class with some topics spanning two sessions.

The first class is devoted to introducing the course, the game lab, and describing the project. Students are asked to complete a survey indicating their particular interests. Each student is given a copy of the survey from all other classmates. Using this information, students are asked to contact their classmates outside of class and form teams for the remainder of the semester. During the second class the teams are finalized. During the first few weeks of class each team is asked to complete a preliminary and later a final design document describing their game. The final design document includes the game concept, genre, summary of gameplay, major features, setting, story, target audience, a list of art assets to be created and a weekly work schedule. The project consists of five milestones during the semester. The milestones are team formation, the two design documents, a presentation demonstrating the game late in the semester, and a final demo and presentation of the game.

5.2 Art

The art component of the course follows closely with the art textbook by Omernick [12]. Multiple chapters from the text are grouped together into six topics. The text does not cover character animation so the seventh topic is an addition made by the instructor. The seven topics are:

1. Creating concept art – the use of reference art such as photographs.
2. Creating 3D models – basics of 3D modeling for games including typical constraints such as low polygon count and limited texture usage.
3. Creating textures – file formats, resolutions, seamless and tileable texture sets.
4. Applying textures to 3D models – UV mapping.
5. Special effects – particle systems, billboards, environment mapping and texture animation.
6. Game lighting – precomputed versus in-game lighting.
7. Character animation – modeling, rigging, animation methods.

5.3 Art/Programming Issues

Early in the course students are encouraged to create a simple program that loads and displays a 3D model. The instructor's expectation is this activity should take at least a month with ongoing modification made throughout the course of the semester. This exercise is important to get programmers and artists working together as soon as possible. There is typically a significant disconnect between what artists can create in a modeling program and what a game engine can import. This is one of the most difficult problems faced by artist-programmer teams. The problem is not so much that 3D file formats are difficult to understand. With some degree of effort, students can create a 3D file parser. Instead, the problem centers around what artists put into these files. A 3D modeling program allows the artist great flexibility in creating and texturing a model whereas a game engine typically requires a model to be created using a very specific subset of functions available in modeling software. For example, modeling programs may allow the artist to add procedural textures while game engines may only be able to import simple UV-mapped 2D textures.

We have had success using the Lightwave file formats including the 3D object file format and scene file format. The scene file contains bones and key frame animation data. Students are encouraged early in the semester to work with and refine their model viewer to discover any problems in their content creation pipeline.

5.4 Programming

Students are expected to use the C++ programming language and Microsoft DirectX 9 as the underlying Application Programming Interface (API) for their game. DirectX is the most widely used low-level API in commercial games and there are many books available that describe its use. Like Adams, we also provide students with a basic code framework to get started [1]. A set of demo projects based in part on the DirectX SDK provides basic game engine functionality including graphics, input device support, audio playback, and networking. The most important part of the code is support for loading Lightwave object and scene files. Other specific components include a file-level texture manager, integrated event processing for multiple input devices, particle systems, skeletal animation, visibility queries, collision detection, and streaming audio support. The multi-threaded code abstracts details of the DirectX SDK which enhances the maintenance of the code since each new version of DirectX has had changes to the API. Hiding the low-level DirectX implementation of a game engine was also reported by Coleman to be beneficial to an undergraduate game programming class [4].

Unlike the art component of the course, the programming textbooks serve as reference material. Programming lectures are supplemented with simple C++ demonstration code and are grouped into the following 11 topics:

1. The compiler – presentation of the Microsoft Visual Studio integrated development environment.
2. DirectX SDK – primary interfaces including graphics, sound, and input.
3. Game graphics – basic 3D concepts, hardware and software, 3D transformations, 3D object representation, file formats and 3D camera control.
4. Windows programming – OS architecture, SDKs, event-driven programming and multi-threading.
5. Game architecture – major software components of a 3D game engine.
6. Thread programming – thread creation, termination, prioritization and synchronization.
7. Advanced 3D graphics – visibility, bounding volumes, and scene management.
8. Socket programming – TCP and UDP application protocols.
9. Texturing - texture coordinates, texture filtering, texture animation, multi-texturing, cubemaps, and volume textures.
10. Terrain and water.
11. Procedural game level generation – scene management for indoor levels and automatic generation of indoor geometry.

5.5 Lessons Learned

The advanced course was taught for the first time in spring 2006. Those students who had previously taken the undergraduate graphics course did better than those who did not. As a result we intend to add undergraduate graphics as a prerequisite for the advanced game course.

The art students were not accustomed to creating models for games and working under the constraints that game engines place on textures and material properties. The art textbook, while presenting good information in general, did not provide practical methods for many texture-related tasks such as creating UV maps. To remedy this we have added another software package to the game lab specifically designed for creating UV maps and adding textures to 3D models including painting directly onto a model.

6. THE FUTURE

In spring 2007 we will establish a motion capture laboratory for use by students in the advanced course. Commercial games rely heavily on motion capture to generate human animation. As an added benefit to the Visual Art Department, the motion capture laboratory can also be used by art students for non-game animation projects.

7. REFERENCES

[1] Adams, J. Chance-It: An Object-Oriented Capstone Project for CS-1. In *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education (SIGCSE ’98)* (Atlanta, Georgia, Feb. 26 – March 1, 1998) ACM Press, 1998, 10-14.

[2] Bates, B. *Game Design*, Second Edition. Thomson Course Technology, 2004.

[3] Becker, K. Teaching with Games: The Minesweeper and Asteroids Experience. *The Journal of Computing in Small Colleges*, 17, 2 (Dec. 2001), 23-33.

[4] Coleman, R., et. al. Game Design & Programming Concentration within the Computer Science Curriculum. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education (SIGCSE ’05)* (St. Louis, Missouri, February 23-27, 2005) ACM Press, 2005, 545-550.

[5] Duffy, J. Game Developer’s 5th Annual Salary Survey. In *Game Developer* (April 2006), CMP Media Group, 2006, 11-17.

[6] El-Nasr, M. S., and Smith, B. K. Learning Through Game Modding. *ACM Computers In Entertainment*, 4, 1 (January 2006).

[7] Entertainment Software Association. Essential Facts about the Computer and Video Game Industry: 2004 Sales, Demographics, and Usage Data. Entertainment Software Association. `http://www.theesa.com`, 2005

[8] IGDA Curriculum Framework. Report version 2.3 beta, International Game Developer’s Association. `http://www.igda.org`, 2003.

[9] Jones, R. M. Design and Implementation of Computer Games: A Capstone Course for Undergraduate Computer Science Education. In *Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education (SIGCSE ’00)* (Austin, Texas, March 7-12, 2000). ACM Press, 2000, 260-264.

[10] Jones, W. *Beginning DirectX 9*. Thomson Course Technology, 2004.

[11] Moller, T., Haines, E., and Akenine-Moller, T. *Real-Time Rendering*, Second Edition. AK Peters, 2002.

[12] Omernick, M. *Creating the Art of the Game*. New Riders Games, 2004.

[13] Overmars, M. Game Maker 6.0. `http://www.gamemaker.nl`, 2006.

[14] Parberry, I., Kaxemzadeh, M., and Roden, T. The Art and Science of Game Programming. In *Proceedings of the 2006 ACM Technical Symposium on Computer Science Education (SIGCSE ’06)* (Houston, Texas, March 1-5, 2006). ACM Press, 2006, 510-514.

[15] Pleva, G. Game Programming and the Myth of Child’s Play. *The Journal of Computing in Small Colleges*, 20, 5 (Dec. 2004), 125-136.

[16] Sindre, G., Line, S., and Valvag, O.V. Positive Experiences with an Open Assignment in an Introductory Programming Course. In *Proceedings of the 25th International Conference on Software Engineering (ICSE ’03)* (Portland, Oregon, Ma3 3-10, 2003). ACM Press, 2003, 608-613.

Design Issues for Undergraduate Game-Oriented Degrees

Michael Mateas
Dept. of Computer Science
Univ. of California, Santa Cruz
Santa Cruz, CA 95064
michaelm@cs.ucsc.edu

Jim Whitehead
Dept. of Computer Science
Univ. of California, Santa Cruz
Santa Cruz, CA 95064
ejw@cs.ucsc.edu

ABSTRACT

The paper describes the most significant design issues concerning the development of game-oriented undergraduate degree programs. These issues fall into two broad categories, those that concern the organization of the degree, including its framing and naming, as well as issues concerning the degree's content. Content issues include the amount of computer science content, use of digital media content, game design and game projects, ethics requirements, breadth requirements, and the impact game degree programs can have on the existing computer science curriculum.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education

Keywords

Curriculum design, computer game degree, game education

1. INTRODUCTION

Motivated by several factors, many universities are considering or implementing degree programs at the graduate and undergraduate level focused on computer games. One reason these programs are being instituted is the goal of increased enrollment, leveraging the widespread cultural interest in computer games into student involvement in game-oriented degree programs. Computer games are also intellectually exciting, sitting at the nexus of computer science, film, digital media, theater, art, literature, economics, and social science, thereby offering new opportunities for dramatic expression, nonlinear storytelling, social commentary, and interactive education. This intellectual interest in the expressive and interactive potential of computer games is also driving the creation of degree programs.

Like any complex design activity, the creation of a new degree program involves the simultaneous consideration of a wide range of issues. Some of these issues are universal across all kinds of degree program creation—gathering resources, building political support—and some are distinctive for game-oriented programs. The authors of this paper have been involved in the design of game-oriented undergraduate degree programs at the University of California, Santa Cruz (Bachelor of Science in Computer Science: Computer Game Design) and the Georgia Institute of Technology (Bachelor of Science in Computational Media). Within, we discuss many of the core design issues involved in the development of new undergraduate degree programs focused on computer games, drawing upon our personal experience, as well as other notable degree programs. While much of this discussion is useful for the creation of any game-oriented degree program, due to the authors' backgrounds the discussion will be more focused on technically focused and interdisciplinary degrees, with less discussion of art-focused game degrees.

The sections below begin by discussing issues concerning the framing and naming of undergraduate game-oriented degree programs (Section 2). Following is an exploration of a series of curricular design issues (Section 3) that affect the academic content of game-oriented degree programs.

2. CURRICULUM ORGANIZATION

The following two sections present issues concerning the overall emphasis of a gaming oriented degree program, and the naming of such programs.

2.1 Degree Emphasis

Perhaps the most important issue in designing a computer game degree program is determining its emphasis. Reflecting the broad and interdisciplinary nature of professional computer game creation, degree programs have emphases along a continuum from very art focused programs to very technically focused programs, with a number of broadly interdisciplinary programs exploring different niches in between. Undergraduate computer game degree programs in the United States tend to fall into one of three categories:

Art focused:

These programs emphasize the artistic and graphic design aspects of computer games, with only a small number of programming courses. Students graduating from these programs are well suited to join the art track of a computer game company. Example: BA in Game Art and Design, Art Institute Online.

Evenly interdisciplinary:

These programs have strong computer science foundations, but do not go into computer science topics with the same depth as technology focused programs. Instead, they offer a broader mix of courses on game design topics, and tend to emphasize game design. These degree programs can also provide students some degree of choice as to whether to focus on technology or arts. Example: Georgia Tech: BS in Computational Media; Worcester Polytechnic Institute: Interactive Media and Game Development.

Technology focused:

These programs are strong computer science degrees, with additional courses adding depth in computer game design. Students graduating from these programs can enter the technical track in computer game companies. As compared to the other kinds of programs, the technically focused programs provide greater depth in computer science topics. Examples: UC Santa Cruz BS Computer Science: Computer Game Design, Univ. of

Southern California, BS Computer Science (Games); Univ. of Denver, BS Game Development and Animation.

The difference between the evenly interdisciplinary and technically focused programs can be subtle. One way of thinking about the difference is that the evenly interdisciplinary programs emphasize technology-inflected design, while the technically focused programs have curricula that emphasize design-inflected technology.

The choice of emphasis will often be strongly determined by the kind of organization developing the degree. An art-focused department or school, such as the Art Institute Online, will naturally develop an art-focused degree program. In a similar vein, computer science departments will tend towards technically focused degree programs, since it permits building upon their existing strength.

Interdisciplinary programs are more challenging to attempt, since they require more new courses to be developed. Most colleges and universities in the United States do not have dedicated game studies departments or research groups, and hence do not have a ready base of game design or game studies courses to draw upon when creating a new major. Prospective undergraduate students are typically very motivated by degree programs that permit them to engage in game design. As a result, evenly interdisciplinary programs tend to better match this desire.

One concern when developing programs is the career prospects for students after graduation. For students in art focused or technology focused programs, there is a straightforward story to tell, with art focused students going into the art track of game studios, and technology focused students going into the game development track. Additionally, art-focused students could perform a wide range of digital art work, and technology focused students are sufficiently well trained that they can take almost any kind of information technology job. With evenly interdisciplinary programs, the story is more complex. Many game development studios are unlikely to hire freshly graduated students to perform game design, since this is typically a senior role. However, a job as a level designer is certainly a reasonable expectation. Additionally, students in the interdisciplinary programs do receive a solid technical background, though it is not as deep as the technically focused degrees. As a result, they are likely qualified for some game development tasks. Since these degree programs are very new, they do not yet have enough graduates to fully understand their career trajectories.

While many game-oriented degree programs focus on vocational outcomes for their students, it is important for programs to ensure their students are adequately prepared to embark on graduate study. There are an increasing number of graduate programs focused on computer games. As well, students who complete a game-oriented degree in their undergraduate studies may very well choose a different type of degree for their graduate studies.

2.2 Naming Game-Oriented Programs

At present there is no consensus on what title is granted to students after completing a game-oriented degree program. An extensive listing of game degree program names can be found in Davidson [Davi05], which is current as of 2005. We see a few emerging trends in program names.

Game Development

Several programs have game development in their title. This title is directly descriptive, since students are taught how to develop computer games. The title gives the impression of being more vocational than a science or engineering degree, similar to the term “programming”. Examples:

Game Development (FullSail)
Electronic Game and Interactive Development (Champlain College)

Computer Science plus

Technically focused programs that are emerging out of computer science departments are choosing to use the BS Computer Science title, and then tack on additional descriptive terms. These titles carry the gravitas of the existing BSCS, itself an upstart scant decades ago. These titles fuzz the issue of whether these are specialized computer science degrees, or completely novel degrees. Examples:

BS Computer Science (Games) (Univ. of Southern California)
BS Computer Science: Computer Game Design (UC Santa Cruz)

Media oriented

Titles using the term “media” recognize that computer games have created a new form of computational media. This ties computer games into ongoing work in digital media, and creates a broader intellectual space for the degree program, as it can expand beyond an initial focus on computer games into different forms of computer afforded media. The drawback is that the general population currently does not have a strong understanding of interactive/digital/computational media, and does not necessarily equate that with the creation of computer games. Examples:

Computational Media (Georgia Institute of Technology)
Interactive Media and Game Development (Worcester Polytechnic Institute)

Simulation oriented

Some degree titles focus on the virtual world simulation aspect of creating computer games. This is an interesting move, evoking aspects of Simon’s *Sciences of the Artificial* [Simo68] or Gelernter’s *Mirror Worlds* [Gele91], which emphasize the simulation science aspect of computer science. These degree titles also broaden the programs, opening up the possibility of covering more simulation-focused content in the future. Like degrees with media in their title, it is not clear that prospective undergraduate students equate simulation with computer games. Examples:

Real Time Interactive Simulation (DigiPen)
Digital Simulation and Gaming Engineering Technology (Shawnee State University)

3. CURRICULUM CONTENT

3.1 Computer Science Content

Technically focused programs have to strike a balance between how many traditional CS courses are required, which provide students with a firm CS foundation, vs. how many game specific courses to offer (both design and technology), which give students depth in games. This is an opportune moment for CS programs to create game specializations, as many computer science departments are already reconsidering their core curriculum. Significant downturns in CS enrollments have caused CS departments to question why current CS curricula fail to attract students [Denn05].

One way of characterizing the resulting curricular debates is in terms of a big-kernel vs. small-kernel understandings of what it means to be a computer scientist. Big-kernel approaches assume that, in order to be a computer scientist, all students must acquire a large shared body of knowledge consisting of "core" CS topics before taking more specialized courses. The large "core" generally consists of the union of the research specialties of the department faculty (everyone thinks their specialty is something everyone should know), skewed towards specialties that were already well established in the 1970s and 1980s. Thus, for example, systems, compiler and theory topics may comprise a large portion of big-kernel curricula, while software engineering, human-computer interaction and ubiquitous computing will not, precisely because the former were well-established subfields before the later. Obviously there are only so many topics that can be fit into a four year degree. In a union-of-research-interests big-kernel model, early subfields will naturally dominate more recent subfields for the simple reason that, in a curriculum that grows by accretion, the "core" is already full before the arrival of more recent subfields. Small-kernel approaches acknowledge that CS has grown into a large and unruly confederation of often only loosely related research areas. The core of topics every computer scientist should know is therefore small, with students branching into sub-areas of computer science more quickly, and consequently more choice available earlier in the curriculum. A number of highly-ranked programs are making the move towards small-kernel CS, the most publicized being the Georgia Institute of Technology's Threads program [Furs06].

Within this climate, the creation of a technically-focused game specialization can energize the departmental discussion and debate around curriculum reform. A game degree puts pressure on CS departments to offer more specialized classes, and to make those classes available earlier in the curriculum. The high-level of interest in such programs among entering freshman guarantees that game specializations will quickly have a large number of students, further putting pressure to reduce the number of "core" courses, as the number of students in the game specialization equals or exceeds the number of students in the traditional major. Curriculum changes necessary to accommodate a game degree have the happy side-effect of setting up departments to be able to quickly establish additional specializations in the future, making the department more nimble in its ability to address future technological and social changes.

While the argument above has counterpoised big-kernel CS vs. small-kernel plus specialization, big-kernel CS is actually a specialization itself. Since the sub-disciplines that were established early (and which dominate big-kernel curricula) tend to focus on computers and computing as ends in themselves, while more recent sub-disciplines tend to connect computing to specific social and cultural contexts, big-kernel CS can be characterized as the specialization focusing on computing for computers, while more recent sub-disciplines can be characterized as computing for people. Interestingly, this directly relates to the much decried lack of women in computer science programs. The seminal, multi-year Women in Computer Science study run at Carnegie Mellon University [Marg03] found that women attracted to computing, more so than men, tend to care about the context and connections of computing to other arenas; it is these connections that make computer science meaningful to them [Marg99]. Thus, traditional big-kernel CS programs, with their focus on "computing for computers," are almost perfectly designed to chase women from the program. Games, as the emerging and potentially dominant expressive medium of the 21st century, are poised to connect to every facet of cultural life, from politics to public policy, from education to entertainment. The creation of a game degree, and the resulting structural changes within the CS department that ease the creation of future specializations, create opportunities for students to connect computer science to other disciplines, and to broader social and cultural concerns, addressing the female retention issues that have plagued traditional computing curricula.

3.2 Digital Media Theory

Any game program has to decide how much of the design and theory side of the curriculum should be explicitly focused on games vs. on situating games within the broader framework of media studies, particularly looking at digital and interactive media. While at first blush it may seem appropriate for game degree programs to focus entirely on game design and theory, so as to maximize the students' depth of knowledge in games, we feel that this would be serious mistake. This is for a quite simple reason: what it means for something to be a "game" is not stable. An over-focus on the design elements, approaches, and rules of thumb that are used to create what is currently understood as a game will leave students unprepared to track the ongoing evolution of the medium as the very definition of game continues to change and morph.

As an example of the fluidity of the concept of "game", in [Ward05] Wardrip-Fruin examines a number of interactive media work that have been explicitly declared "not games" by their creators, and yet have strong game-like qualities that, in some cases, have resulted in the definition of "game" broadening to accept the new type of interactive experience. For example, the website for Electronic Arts declares *The Sims* to be "The #1 best selling game of all time", yet both academic commentary as well as the commercial game press have described *The Sims* as more of a software toy than a game. Creator Will Wright has also referred to his sim games (from *Sim City* on) as software toys. *The Sims* has no winning condition, no score, and no explicit game goals. Yet now it is fairly commonplace to refer to such open-ended "playable simulations" as games; the concept of "game" has expanded to include such experiences. As another example, alternative reality games (ARGs) explicitly smudge the boundaries of the magic circle, creating conspiracies for groups of players to uncover through a mixture of real-world and web-based sleuthing. ARGs purposefully obscure the distinction between fiction and fact, mixing players' everyday lives with the game world. Such games often explicitly declare "this is not a game" as a way to help players maintain a belief that the events in the game are really happening. But ARGs, as the name implies, are now considered a standard game genre; the ever-plastic concept of "game" has again expanded to encompass this new form.

In order to prepare students to not only participate, but hopefully play a leadership role in this continuing innovation in the nature of games, they must not only understand the current state of game design, but how games participate in the broader media ecology. A firm foundation in the history of interactive media prepares students to understand how new media forms come into being, and how communities of practice develop conventions that simultaneously realize and constrain the technical possibilities of the medium. By investigating media phenomena such as networked communities, mobile communications and geo-positioning technologies, ambient media, and electronic literature, students become familiar with how different combinations of social and technical infrastructures function as expressive forms. A broad foundation in digital media, in addition to depth courses in game design, prepares students to create the games of today as well as invent the new game genres of tomorrow.

3.3 Game Design Content

Even technology-focused game design programs must have significant game-design content in the curriculum. The game industry as a whole is moving towards interdisciplinary work practices, as exemplified by the organization of the team working on *Spore*, Will Wright's "Sim Everything" game. As Wright has described in his Game Developers Conference talks on *Spore*, designers and programmers are tightly coupled in the design process, with designers knowing enough programming to write lightweight prototypes that demonstrate design concepts, and programmers knowing enough design to iterate tightly with the designer as they write robust production code. The UC Santa Cruz degree program aims to create design literate programmers who have the CS chops to design the architectures and write the engines necessary to realize the game, while being first-class participants in the design process.

The most basic element of game design is the notion of rule systems, formal systems that govern player interaction and the evolution of game state. Students learn how to think about rule systems, and learn standard rule patterns found in many games, such as the rock-paper-scissors pattern governing relationships between units in RTSs and spells and equipment in RPGs. Additionally, students learn how to use rapid prototyping, including paper and pencil prototyping, to understand emergent interactions between rules. Concepts from psychology, such as flow, cognitive understandings of problem solving, and affective response, prepare students to think about player response during game playing. Concepts from sociology provide students with tools to think about community design for MMOs. Students learn about the game design process, including the typical roles found on game teams, milestones, resource management, and lessons from design science. In addition to focusing on design for the interactive entertainment industry, students learn about serious games, including political and policy games, training games, and games for health. Finally, students learn about emerging game design topics, including alternative reality gaming, and mobile and casual gaming,

3.4 Project Based Learning (Game Projects)

One area on which there is rough consensus is the need for at least one substantial game development project in the curriculum. Typically this project is viewed as a final year capstone, in which students work as members of a team to create a large computer game. Like most project-based learning activities, the senior game project is intended to allow students to synthesize knowledge developed in the classroom by applying it to the construction of a computer game. Game projects also allow students to experience all aspects of a typical game development lifecycle, from initial game concept through coding and art asset creation, to testing and deployment. Additionally, the project provides students with a completed game they can add to their portfolio and demonstrate to prospective employers or graduate schools. As an example of one such project, in the UC Santa Cruz degree program, students take the Game Design Studio sequence (3 quarters) during their entire senior year. Many other programs have similar project sequences.

Game project classes can also be used to energize students early in the program. Incoming students are very eager to begin learning the primary focus of their chosen degree in their first year. However, this is the year when most degree programs focus on background courses, such as an introduction to programming, calculus, physics, and institution-specific general education requirements. This leads to a mismatch between student interest and excitement to engage in game design and development, and the coursework they are required to take. One way to address this is to have students engage in game project development early in their curriculum. The DigiPen BS in Real-Time Interactive Simulation (RTIS) degree program is an excellent example, with students taking a game project class in each semester, starting the second semester of their freshman year. Another approach taken by the UC Santa Cruz program is to have a freshman year game project course, called the Game Design Experience. This course combines lecture material introducing game design with a small team development project.

3.5 Ethics Content

There are several negative aspects of computer games that appear in mainstream media. Some games have graphic violence, and there are concerns that repeated exposure to this violence leads to more aggressive behavior. Other games, especially massively multiplayer online games, can be extremely engaging, resulting in extensive gameplay hours that can lead to loss of relationships or job loss. These concerns are a mixture of both real issues and general societal concern with any new and powerful medium that is adopted by youth. Addressing these concerns is an aspect of designing a game-oriented degree program, since they will likely be raised by faculty examining any new proposed curriculum, and may affect the degree of enthusiasm among faculty members towards the degree. In addition to this internal consensus-building concern, the parents of prospective students, who are often quizzically amused by the game-playing activities of their children, are happy to hear that ethical issues, rather than being ignored or side-stepped, are dealt with squarely within the curriculum. Finally, for the students themselves, an ethics course ensures that students are familiar with some of the societal debates swirling around games.

One way to address these concerns is to introduce an ethics requirement. An ethics course provides students with a framework for critically examining a wide range of ethical issues, including those that are associated with computer games. We do not anticipate that students taking an ethics course will, for example, immediately stop using violence as an interaction mechanism in their games. Depictions of violence may, in fact, be an entirely appropriate design choice for a particular game. The goal is instead to ensure that inclusion of violence, or creation of extremely compelling gameplay, will be done after considering the ethical implications, and with full knowledge of any tradeoffs that are being made. Furthermore, if a new game ends up creating unforeseen negative issues, students will have the ability to reason about the ethical implications.

3.6 Breadth Content

One tension in creating a game-oriented degree program is the need to provide sufficient depth in either the artistic, design, or game development areas so that graduating students are competitive in applying for game development jobs, while at the same time providing a broad education across a wide range of topics so that students can draw upon this rich background knowledge when designing games. Job advertisements for computer game professionals focus on specific skills, such as specific programming languages, graphics libraries, AI techniques, etc, and say nothing about broad backgrounds. In contrast, game designer Chris Crawford lists a very broad set of books, movies, and periodicals in his, "Education of a Game Designer." (Chapter 9, in [Craw03]).

One solution to this tension is to require students to take some number of breadth courses that are not directly related to computer games. Colleges and universities with a wide range of degree programs tend to have a variety of courses covering history, literature, psychology, music, film, theater, and so on, which students can draw upon for this breadth. Most 4 year colleges and universities have some kind of formal breadth requirement for all students. At UC Santa Cruz, 80 credits out of a 180 credit degree program must satisfy campus General Education requirements, which are split across a wide range of subjects, and include a writing intensive course. The UC Santa Cruz game degree has a series of "Art and Social Foundation" requirements in which students select three from a list of five categories: Art, Film, Theater Arts, Music, and Economics. In each category, students select a single course from a list of approved courses for the major. In this way, students have part of their general education requirements targeted towards courses of particular relevance for game design, while still giving students substantial freedom in picking additional breadth courses.

More specialized institutions need to explicitly create courses to provide some of this depth. FullSail has required courses on Historical Archetypes and Mythology and Media and Society, and DigiPen has courses on Mythology for Game Designers and Society and Technology. One of the clear tradeoffs is that specialized institutions offer fewer breadth courses, thereby allowing them to offer more game design courses, while traditional colleges and universities can draw upon a broad array of breadth courses, but with fewer slots available for focused game design courses. The lack of required breadth courses also permits more compressed schedules, with FullSail's program taking just three years.

3.7 Impact of Computer Game Topics on Existing Curricula

The creation of a game degree program is not only an opportunity to add new game-specific courses to the curriculum, but also an opportunity to add gaming related content to existing CS courses. Games are a great student motivator and, due to the breadth of topics CS topics that play a role in games, can be incorporated into almost any class in the CS curriculum. As an example of organizing a CS course around games, one of us (Mateas) developed a version of the sophomore introduction to hardware systems course at Georgia Tech organized around the Game Boy Advance (GBA) portable game system.

The Media Device Architectures course covers the basics of binary representation, memory and processor architecture, memory-mapped I/O, interrupts, and low-level C programming (bit masking, bit shifting, etc.), but using the GBA as the reference architecture. Though a fairly contemporary game system, the GBA is organized like many classic consoles, with no OS, a memory-mapped architecture for I/O (requiring explicit memory manipulation to, say, draw on the screen), and interrupt-based input handling. Thus, students can gain real experience coding "close to the silicon", while having the motivation of programming a real game console. Further, unlike many introductory hardware systems courses which make use of an artificial simplified reference architecture for teaching assembly language programming, GBA programmers have a real motivation to embed assembly blocks in their C code, as assembly may be needed to optimize critical paths within the game code, or to write interrupt handlers that can require the use of opcodes that aren't generated by the C compiler. Finally, GBAs can be networked together with serial cables, giving students the opportunity to play with simple network protocols.

4. CONCLUSION

This paper has presented a series of design issues that confront any group seeking to develop a new game-oriented undergraduate degree program. Our goal in presenting these issues is to make these design considerations explicit, so that future designers can more easily create new, compelling degree programs by understanding the issues involved in the creation of existing ones.

References

[Craw03] Chris Crawford, *Chris Crawford on Game Design*, New Riders, 2003.

[Denn05] Peter Denning and Andrew McGettrick. Recentering Computer Science. *Comm. ACM*, 48 (11), Nov. 2005, 15-19.

[Davi05] Drew Davidson, "Games by Degrees: Playing with Programs," *On The Horizon. Special Issue. Second Generation E-Learning Part 2: Serious Media.* 13(2), 2005, pp. 70-74.

[Furs06] Merrick Furst and Richard DeMillo. Creating Symphonic-Thinking Computer Science Graduates for an Increasingly Competitive Global Environment, The Georgia Institute of Technology, 2006, http://www.cc.gatech.edu/images/pdfs/threads_whitepaper.pdf.

[Gele91] David Gelernter, *Mirror Worlds*, Oxford University Press, 1991.

[Marg99] Jane Margolis, Allan Fisher and Faye Miller. Caring about connections: gender and computing. *IEEE Technology and Society Magazine*, 18 (4), 1999, 13-20.

[Marg03] Jane Margolis and Allan Fisher. *Unlocking the Clubhouse: Women in Computing*. MIT Press, 2003.

[Simo68] Herbert A. Simon, *The Sciences of the Artificial*, MIT Press, 1968.

[Ward05] Noah Wardrip-Fruin. Playable Media and Textual Instruments. *Dichtung Digital: Journal on Digital Aesthetics* (1). 2005. http://www.dichtung-digital.com/2005/1/Wardrip-Fruin

SAGE: A Simple Academic Game Engine

[Extended Abstract]

Ian Parberry
Jeremiah R. Nunn
Joseph Scheinberg
Erik Carson
Jason Cole
Department of Computer
Science & Engineering
University of North Texas
Denton, TX, USA
ian@unt.edu

ABSTRACT

SAGE is a simple academic game engine for use in a game programming class in the undergraduate Computer Science curriculum, designed specifically as a core onto which students can add their own game engine features. SAGE consists of a sequence of demos written in C++ using Microsoft DirectX, each extending its predecessor in a process called *incremental development*. Incremental development is a proven pedagogical technique used for the education of game programmers at the University of North Texas since 1997.

Categories and Subject Descriptors

K.3.2 [**Computing Mileux**]: Computers and EducationComputer and Information Science Education[Computer science Education]

General Terms

Design, Experimentation

Keywords

1. INTRODUCTION

In 1993 we introduced a game programming course to the undergraduate computer science program at the University of North Texas. At the time this was a difficult task, both because there were no course materials, books, or web pages available, and because the industry-driven focus of the class and the perceived trivial nature of entertainment computing made the subject matter controversial. Interestingly, the objections came from faculty - both the students and the administration were in favor of the class. Since 1993 the initial game programming class has evolved with the fast-moving game industry, and spawned a second, advanced game programming class. After more than a decade of operation, our game programming classes have positioned our alumni for employment in companies including Acclaim Entertainment, Ensemble Studios, Gathering of Developers, Glass Eye, iMagic Online, Ion Storm, Klear Games, NStorm, Origin, Paradigm Entertainment, Ritual, Sony Entertainment, Terminal Reality, and Timegate Studios. For more information about these classes, see [11, 12].

Despite a rocky beginning, game programming is now gaining acceptance in academia (see, for example, Adams [1], Becker [2], Faltin [4], Feldman and Zelenski [5], Jones [7], Moser [8], and Sindre, Line, and Valvåg [13]), resulting in a proliferation of new classes and programs both internationally and nationwide and a move towards a professionally recommended curriculum in game studies [6]. In contrast to institutions such as Digipen, Full Sail, and SMU's Guildhall that offer specialized degrees or diplomas in game programming, UNT offers game programming as an option within a traditional computer science curriculum.

The students in our game programming classes are usually seniors in the computer science program, who are technologically savvy and experienced programmers. They are usually quite capable of reading the documentation for game APIs, such as Microsoft DirectX, themselves. For them, the biggest road-block is picking the small subset of techniques that they actually need from the wealth of options available. The lectures focus on getting started, and leave exploration of options in the more than capable hands of the students. Our game programming classes have a positive effect on undergraduate enrollment in the Department of Computer Science and Engineering at UNT. Out of almost 200 students from the prerequisite classes surveyed in 1993, 49% of students intended to take the introductory game programming class, and 39% said that the class was a contributing factor to their presence in the Computer Science program at UNT (for full figures, see [12]).

Selection of a game engine is a major decision that can make or break a game programming class. Students learning game programming in academia need an engine that is flexible, extensible, stable, and well-documented. Industry game engines such as Vertigo's Quake II .NET, Unreal Technology's Unreal Engine, and Valve's Half Life 2 engine, are large, complex, and relatively complete. An academic game engine should in contrast be small, simple, and incomplete. It should be suitable as a foundation on which students can build, and above all be easy to understand and modify, especially by relatively inexperienced students. It should illustrate new concepts in enough detail for students to get started, but should avoid "completeism". It should obey the educational principle "proceed from the known into the unknown".

The main part of this paper is divided into five sections. The first section lists the requirements for a simple game engine, and the technology necessary to implement them. The second section gives an overview of the SAGE project. The third section describes the seven SAGE demos in more detail. The fourth section describes our experience with SAGE in the classroom in Spring 2006. The fifth section discusses our choice of DirectX and Visual C++ for this project.

2. MINIMUM REQUIREMENTS

SAGE is designed to provide the minimum requirements for a game, which are a 3D world that a player can explore in real time, with interesting objects in it, with which the player can interact. The key adjectives in the preceding sentence are *real time* and *interactive*. The technology necessary for this includes:

- A graphics renderer, using pixel shaders and HLSL. It is essential for student morale that the rendering engine be close to cutting edge, and to provide the latest shader technology.
- Objects, including a method for importing 3D models created by artists, and an object manager that takes care of object creation, behaviour, rendering, and destruction.
- A 3D world, consisting of terrain and some method for level-of-detail to increase rendering speed.
- Input from the keyboard, mouse, and joystick to enable the player to interact with the world and the objects in it.
- Collision detection to enable interaction between the player, the objects, and the world.
- A particle engine to enable visual effects that follow from that interaction.

3. SAGE OVERVIEW

SAGE is a 3D game engine developed as a sequence of demos, each built on its predecessor, in a process called *incremental development*. Incremental development has been used in the construction of a billboard demo in a simple 3D world with limited camera movement (*Ned's Turkey Farm*, see Figure 1) for introductory game programming classes at UNT since 1997 (see [11]). Earlier versions have been published in two books [9, 10]. The aim is not to teach this game per se, but rather to teach the development of games in general using this engine as an example. It is designed to have many of the features of a full game in prototype form so that students can use code fragments from it as a foundation on which to build their own enhancements. The students are graded on the basis of a project, which is to create a sprite-based game in groups together with art students from the concurrent game art and design class.

SAGE brings this experience to a fully 3D game engine, based on an educational pedagogy that has a proven track record. SAGE includes a sample game, *Ned's Turkey Farm 3D*. The code consists of a sequence of game demos, each showcasing a new feature. The feature is demonstrated in rudimentary form, leaving room for students to enhance it. The trick is getting it complex enough to convey the fundamental principles, yet simple enough for students to understand. SAGE has Doxygen generated documentation, and approximately 200 pages of tutorials.

SAGE is developed in C++, uses DirectX 9.0, and is accompanied by Visual Studio project files. It is released under a BSD open source license, and is available on the first author's website and in the Microsoft Developer Network Academic Alliance Curriculum Repository.

Figure 1: Screen shot of *Ned's Turkey Farm*.

SAGE is organized as follows. The following description applies to Demo 6, the complete fully-featured project. The top-level folder contains two subfolders, `Ned3D` containing game-specific code, and `SAGE` containing engine code. The `Ned3D` folder consists mainly of game-specific classes derived from the basic SAGE classes, which we will not describe further here. The `SAGE` folder contains two subfolders, `SAGE Resources` containing resources for the console and effects files for the pixel shaders, and the `Source` folder containing SAGE source code.

`SAGE\Source` contains the following subfolders.

- `Common`: Low-level code, which will be described in more detail below.
- `Console`: The game console.
- `DerivedCameras`: A free camera and a tether camera.
- `DerivedModels`: An animated model using animation frames and linear interpolation, and an articulated model.
- `DirectoryManager`: A directory manager, which manages the organization of resources in subfolders.
- `Game`: The GameBase class, which contains game logic code.
- `Generators`: A name generator and an identifier manager.
- `Graphics`: Graphics related code, including vertex buffers, index buffers, and effects.
- `Input`: Input using DirectInput.
- `Objects`: Game objects and the object manager.
- `Particle`: The particle engine.
- `Resource`: The resource manager.
- `Sound`: The sound manager.
- `Terrain`: The terrain code, including height map and LOD.
- `TinyXML`: TinyXML code.
- `Water`: Code for water animation, including use of the reflection pixel shader.
- `WindowsWrapper`: An abstraction layer for Microsoft Windows specific code.

The `Common` folder is of particular interest, since it contains the low-level code for SAGE. SAGE is based on the freely available low-level code from Dunn and Parberry [3]. `Common` includes the following utilities:

- `AABB3.cpp`, `AABB3.h`: Axially aligned bounding boxes.
- `Bitmap.cpp`, `Bitmap.h`: Bitmap image reader.

Module	Code
Common framework	13,729
SAGE	13,469
tinyXML	4,883
Ned specific	2,750
Total:	34,831

Table 1: Number of lines of code in SAGE.

- `CommonStuff.h`, `CommonStuff.cpp`: Common stuff that doesnt belong elsewhere.
- `EditTriMesh.cpp`, `EditTriMesh.h`: Editable triangle mesh class.
- `EulerAngles.cpp`, `EulerAngles.h`: Euler angle class.
- `MathUtil.cpp`, `MathUtil.h`: Basic math utilities.
- `Matrix4x3.cpp`, `Matrix4x3.h`: Homogenous transformation matrix code.
- `Model.cpp`, `Model.h`: Simple class for a 3D model.
- `Quaternion.cpp`, `Quaternion.h`: Quaternion class.
- `Renderer.cpp`, `Renderer.h`: Rendering engine (modified somewhat from its original form in [3]).
- `RotationMatrix.cpp`, `RotationMatrix.h`: Rotation matrix class
- `TriMesh.cpp`, `TriMesh.h`: Triangle mesh class.
- `Vector2.h`, `vector3.h`: vector class.
- `WinMain.cpp`, `winmain.h`: Windows dependent code.

The following low-level code was added to Common:

- `camera.cpp`, `camera.h`: Base camera class, from which the free camera and the tether camera are derived.
- `fontcacheentry.cpp`, `fontcacheentry.h`: Encapsulates the Direct3D font class.
- `plane.cpp`, `plane.h`: Math plane class.
- `random.cpp`, `random.h`: Pseudorandom number generator.
- `rectangle.h`: Rectangle class.
- `texturecache.cpp`, `texturecache.h`: Texture cache class.

Phase 1 of SAGE consists of approximately 35,000 lines of C++ code (including header files, code, and comments). The code is distributed into four parts, the Common framework (described above), SAGE code, tinyXML, and code specific to the sample game, *Ned's Turkey Farm 3D*. The number of lines of code in each of these modules is given in Table 1. The code architecture is described in Figure 2, with the foundation being code from Microsoft DirectX and the Windows API, the Common framework being layered on top of that, supporting the SAGE engine, with code specific to the particular game supported by SAGE layered on top of that.

4. SAGE DEMOS

SAGE consists of seven incremental demos, as follows:

- Demo 0: Model importation and display
- Demo 1: Terrain input and rendering
- Demo 2: Shaders using HLSL
- Demo 3: Game engine architecture
- Demo 4: Collision detection
- Demo 5: Particle engine
- Demo 6: 3D sound

4.1 Demo 0

Demo 0 demonstrates the code for reading and displaying a model. The code for Demo 0 shows the programmers how to import a

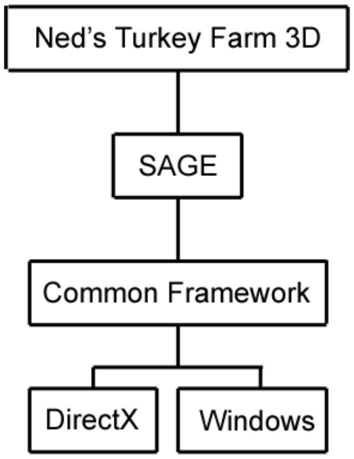


Figure 2: SAGE architecture.

model, render it, and perform simple operations such as rotation and camera motion under user control. In addition, the executable is a useful tool for artists and programmers to check for correct export of models, which can be created using a 3D modeling tool such as Maya or 3D Studio Max (see Figure 3).

Each modeling program has a proprietary file format that changes with each version, the updating of which can cause previously used models to become unusable. Each has facilities for plug-ins to export to a different file format. Some file formats are text, some are binary. Direct3D has a native file format (.X). Other popular file formats exist, eg. Quake II, Quake III models. Managing the input of art assets is one of the biggest startup hurdles in making a game demo. File format converters exist, but our experience with them has in general been less than positive, often resulting in the introduction of degenerate triangles, sliver triangles, missing triangles, detached triangles, and the mangling of origin, axes, normals, and scale.

To help avoid these problems, SAGE uses the S3D format from [3], and includes an S3D plug-in for Maya. S3D is a simple text format that enables the programmer to view the model data directly in a text editor to check for simple errors.

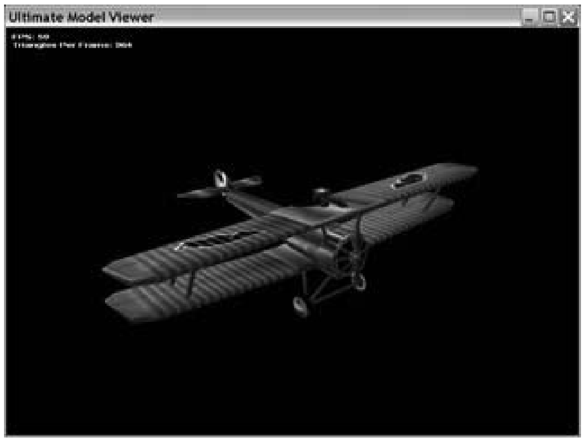


Figure 3: Demo 0 showing the plane model.

4.2 Demo 1

Demo 1 covers terrain input and rendering. It reads a height map from an image file and renders an island surrounded by a small finite area of ocean (see Figure 4). Simple grid-based level of detail is provided. A free camera can be used to explore the terrain. A simple console allows the user to modify game properties easily.

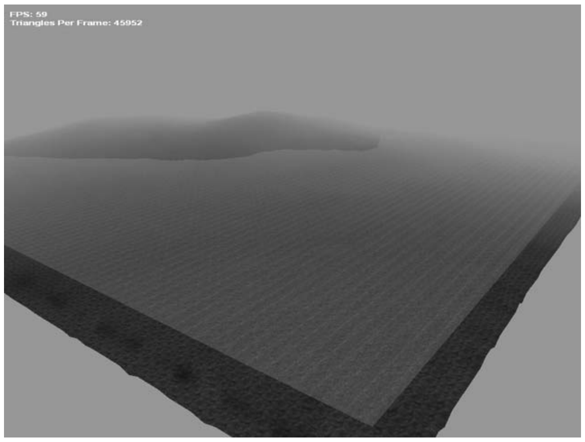


Figure 4: Demo 1 showing ocean and island.

4.3 Demo 2

Demo 2 covers shaders using HLSL. Shaders are provided for texture blending (demonstrated on textures that change with terrain height), and for reflections in water (see Figure 5). A triangle of water that moves with the camera gives the illusion of ocean extending to infinity. We particularly avoided the temptation to create a large number of shaders, preferring to leave that for students. Since shaders are an intricate subject the shader tutorial is the longest of our tutorials, consisting of approximately 50 pages.

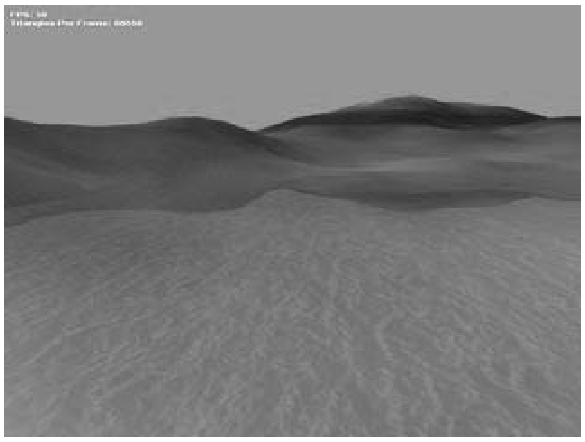


Figure 5: Demo 2 showing terrain reflections and texture blending.

4.4 Demo 3

Demo 3 covers game engine architecture, including objects, an object manager, a tether camera, and DirectInput. Types of objects supported include rigid objects, articulated objects, and animated objects. Articulated objects consist of separate hierarchically organized parts that may be moved or rotated independently, such as the propellor on the airplane and the blades on the windmill in *Ned's Turkey Farm 3D*. Animated objects consist of key frames created by the artist as a set of rigid objects. The SAGE animated object provides in-betweening using linear interpolation. *Ned's Turkey Farm 3D* has crows implemented as animated objects.

4.5 Demo 4

Demo 4 covers collision detection using axially aligned bounding boxes (AABBs). Collision of objects with terrain, objects with objects, bullets with objects are detected. Object-terrain collision is implemented by interpolating terrain height within a triangle, object-object collision is implemented using AABB-AABB intersection, and bullet-object collision is implemented using ray-AABB collision detection. In a real game, AABB collision detection would be only the first or primary level of collision detection, designed to quickly eliminate noncolliding objects. Subsequent levels of collision detection, including bounding boxes and bounding spheres at the secondary level, and triangle-triangle collision detection as the tertiary level, are left as possible projects for the student. For educational purposes, SAGE will render AABBs in real time for classroom demonstrations (see Figure 6).

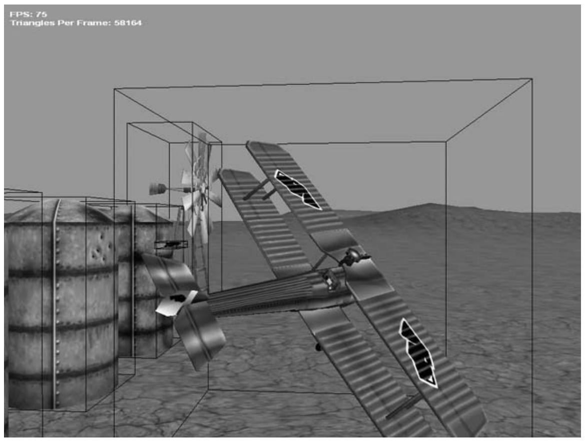


Figure 6: Demo 4 showing AABBs.

4.6 Demo 5

Demo 5 covers particle engines and provides a general purpose particle engine that is used in *Ned's Turkey Farm 3D* for explosions, clouds of feathers (see Figure 7), smoke, gunfire flash, and dust raised by a bullet hitting the terrain.

4.7 Demo 6

Demo 6 covers stereo 3D sound using DirectSound.

5. SAGE IN THE CLASSROOM

SAGE was used for the first time in the classroom in Spring 2006 in the first author's CSCE 4220 (Advanced Game Programming) class while the code and tutorials were still under development. The resulting student games were of a higher quality than in previous years, and included the following:

- Duck Hunt: A medieval first-person shooter in which the played floats across a lagoon at twilight in a canoe, shooting flaming arrows at ducks.
- Sink This: A third-person submarine game in which the player attempts to torpedo other submarines.
- Fury Mallard: A third-person shooter in which a duck attempts to kill men in black suits.

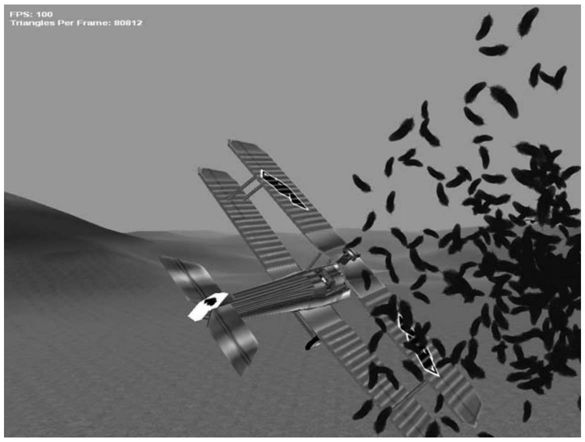


Figure 7: Demo 5 showing cloud of feathers created using particle engine.

- Ghost Hunter: A third-person shooter in which the player attempts to kill zombies.
- Galactic Battlefield: A third-person space shooter.
- Great Space Race: A third person space racing game where the player must navigate between portals against the clock.
- VertiGo: A 3D puzzle game in which the player attempts to navigate a marble through a 3D array of cubes.

6. ON DIRECTX AND VISUAL C++

The authors of this paper have attracted a substantial amount of criticism from academics over their choice of DirectX for the graphics API and Visual C++ for the compiler supported in this project, over OpenGL and g++ respectively. In response, the authors wish to make the following observations:

1. The DirectX SDK (Software Developer's Kit) can be downloaded and used free of charge. Visual Studio Express can be downloaded and used free of charge, which with the addition of the Windows Platform SDK (also available for free) and the DirectX SDK can be used for game development under Windows.

2. DirectX is updated every two calendar months. This means that bug fixes are applied quickly. Unlike OpenGL, there is little or no trouble supporting available video cards. A major version of DirectX is released regularly (DirectX 10 will be available within a year), which ensures that the API keeps up with the latest in graphics technology.

3. We believe that students benefit from using in class the same tools and techniques used by a substantial fraction of the game industry.

4. We strongly believe that students should be exposed to as many different compilers and APIs as possible during their academic tenure. Our students are already exposed to open source software including g++ and OpenGL in other Computer Science classes. DirectX and Visual Studio add to this experience, and are in no way intended to supplant it.

7. CONCLUSION

SAGE Phase 1 was completed in June 2006, and can be downloaded from `http://larc.csci.unt.edu/sage`. SAGE is funded by a grant from Microsoft Research.

8. REFERENCES

[1] J. C. Adams. Chance-It: An object-oriented capstone project for CS-1. In *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education*, pages 10–14. ACM Press, 1998.

[2] K. Becker. Teaching with games: The minesweeper and asteroids experience. *The Journal of Computing in Small Colleges*, 17(2):23–33, 2001.

[3] F. Dunn and I. Parberry. *3D Math Primer for Graphics and Game Development*. Wordware Publishing, 2002.

[4] N. Faltin. Designing courseware on algorithms for active learning with virtual board games. In *Proceedings of the 4th Annual Conference on Innovation and Technology in Computer Science Education*, pages 135–138. ACM Press, 1999.

[5] T. J. Feldman and J. D. Zelenski. The quest for excellence in designing CS1/CS2 assignments. In *Proceedings of the 27th SIGCSE Technical Symposium on Computer Science Education*, pages 319–323. ACM Press, 1996.

[6] IGDA. IGDA Curriculum Framework. Report Version 2.3 Beta, International Game Developer's Association, 2003.

[7] R. M. Jones. Design and implementation of computer games: A capstone course for undergraduate computer science education. In *Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education*, pages 260–264. ACM Press, 2000.

[8] R. Moser. A fantasy adventure game as a learning environment: Why learning to program is so difficult and what can be done about it. In *Proceedings of the 2nd Conference on Integrating Technology into Computer Science Education*, pages 114–116. ACM Press, 1997.

[9] I. Parberry. *Learn Computer Game Programming with DirectX 7.0*. Wordware Publishing, 2000.

[10] I. Parberry. *Introduction to Computer Game Programming with DirectX 8.0*. Wordware Publishing, 2001.

[11] I. Parberry, M. Kazemzadeh, and T. Roden. The art and science of game programming. In *Proceedings of the 2006 ACM Technical Symposium on Computer Science Education*. ACM Press, 2006.

[12] I. Parberry, T. Roden, and M. Kazemzadeh. Experience with an industry-driven capstone course on game programming. In *Proceedings of the 2005 ACM Technical Symposium on Computer Science Education*, pages 91–95. ACM Press, 2005.

[13] G. Sindre, S. Line, and O. V. Valvåg. Positive experiences with an open project assignment in an introductory programming course. In *Proceedings of the 25th International Conference on Software Engineering*, pages 608–613. ACM Press, 2003.

Designing Shape-shifting Collaborative Laboratory Spaces to Facilitate Game-Design Education

David I. Schwartz
Cornell University
Department of Computer Science
5137 Upson Hall
+1 607-255-5395
dis@cs.cornell.edu

Tony Cosgrave
Cornell University
Cornell University Library
109 Uris Library
+1 607-255-7148
ajc5@cornell.edu

Steve Weidner
Cornell University
Cornell Information Technologies
215 Computing & Communication Ctr.
+1 607-254-7403
sw275@cornell.edu

ABSTRACT
In this paper, we describe a novel approach to computer laboratory design for multidisciplinary education, including game design. The Cornell Library Collaborative Learning Computer Laboratory (CL3) is a shape-shifting workspace in which students and instructors can move semi-circular, dual-workspace computer tables to fit a wide-variety of group needs and sizes. We demonstrate that this concept facilitates game-design and development education. Early studies indicate that CL3 does indeed work, though the concept needs a few refinements with respect to training and demonstration.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management – *teams*.

K.3.1 [**Computers and Education**]: Computer Uses in Education – *collaborative learning*.

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *ergonomics*, *evaluation–methodology*, *interaction styles*.

H.5.4 [**Information Interfaces and Presentation**]: Group and Organization Interfaces – *collaborative computing*, *computer-supported cooperative work evaluation–methodology*.

General Terms
Management, Design, Experimentation, Human Factors.

Keywords
Collaborative Learning, Computer Laboratories, Game Design, Multidisciplinary, Education.

1. INTRODUCTION
Game-design programs involve multidisciplinary work, often involving artists, writers, musicians, engineers, and so forth. However, typical computer laboratories, especially those for traditional face-front instruction and course needs, do not serve the wide range of game-oriented team work. When the Game Design Initiative at Cornell University (GDIAC, [1]) began in 2001, our own labs were limited to individual workspaces. As the multidisciplinary component grew, GDIAC realized the need for flexible, collaborative space. By merging with the Cornell University Library's *CreationStation* multimedia development facility [2], a cross-departmental team (Computer Science, University Library, and Academic Technologies) jointly developed the Cornell Library Collaborative Learning Computer Laboratory (CL3). CL3 opened in August, 2004 [3].

CL3 currently hosts GDIAC's core game-design courses, academic excellence workshops for introductory programming, and the new CreationStation laboratory. The facility can fit upwards of 30 students and has supported approximately 10K users each year in the past two years of operation. In this report, we first explain the background components that drove CL3's design. Next, we highlight elements of the implementation that can assist others in building similar collaborative space. Finally, we summarize the results of the current analysis and propose future work.

2. Background
2.1 Collaborative and Cooperative Learning
Collaboration and cooperation have become essential elements of modern educational pedagogy [4-6]. The CL3 project did not seek to justify the importance of teamwork, but instead, to determine *how* to facilitate collaboration with hardware and software. To explain this facilitation, we provide the following definitions:

- *Collaborative learning*: general team-based education.
- *Cooperative learning*: a specific form of collaborative learning that requires team interdependence, different skill sets, final product, and individual accountability.

We describe CL3 as *collaborative* because of the availability for public use. We describe how Cornell's game courses use cooperation in Section 2.3.

2.2 Collaborative Programming
When CL3 was first conceived, the original model addressed the need for collaborative programming space. Cornell University's College of Engineering introductory courses often provide *academic excellence workshops*, which are pass/fail classes in which students work collaboratively on extra course material [7]. For introductory programming, the insufficiency of typical face-front computer labs drove the original plans for CL3. A common model involves *pair programming*, in which a pilot and co-pilot program as a pair, which has shown excellent results in education and practice [8]. Manufacturers have even begun to provide pair programming computer desks [9], and various programs have researched how computer desks can facilitate collaboration [10-12].

2.3 Game Design and Development Education

Game design and development education has flourished in the past few years, leading towards a call for an understanding of best practices [13]. Although pedagogy and content still vary (notwithstanding the wide range of courses and program names), one common aspect is the need for multidisciplinary teamwork. Although game design is an interdisciplinary field [14], students from music, art, writing, engineering, and more work together to produce original works. This collaboration drives tremendous interest in such education, which offers excellent team-skill development, appealing to many students.

As noted in Section 2.1, working on a game often involves cooperation. At Cornell, groups receive individual and group grades, whereby the individuals also rate themselves. This need for close collaboration and multidisciplinary work necessitates a collaborative learning space.

2.4 Learning Spaces and Location

One fundamental aspect of CL3 is the location, an issue perhaps often not addressed. By situating CL3 in a university library, we provide "neutral ground." Whereas not all schools may have this extreme separation, but the computer science and art departments are at literal ends of the campus. Because game design requires joint effort and mutual respect, identifying central and neutral areas is key to facilitating collaboration.

3. LABORATORY DESIGN

This section explains how CL3 addresses the fundamental issues and ideas expressed in Section 2.

3.1 Table Design

CL3's core design starts with a pair-programming computer desk, two of which are illustrated in Figure 1:

- Curved, one eighth-circle to allow for semi-circles in a classroom arrangement.
- Seating on the inside curve to use classroom space more efficiently. Note that an inside curve helps to alleviate lines of sight that aim away from partners.
- Table rollers—each table forms a moveable unit to create larger collaboration groups, perhaps even an entire class.

To determine the dimensions shown in Figure 1, we worked with Cornell's laboratory guidelines. We also developed a full-scale mockup to test user-responses to the environment with informal surveys.

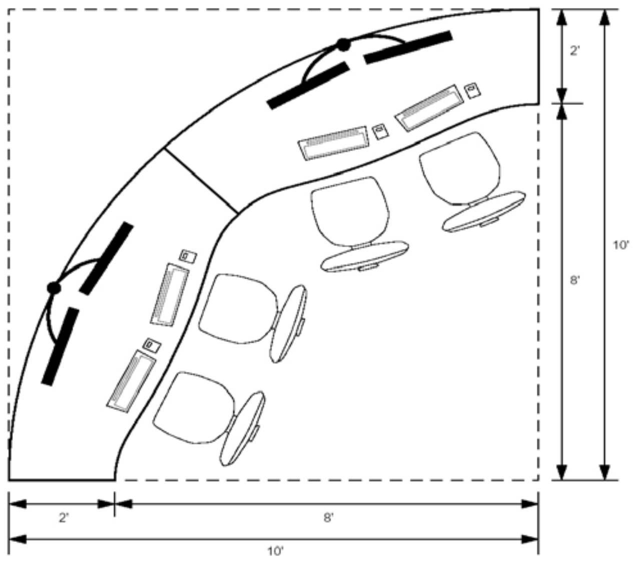


Figure 1. Two of CL3's pair programming desks.

3.2 Workstations and Input Devices

A commonly asked question is the choice against laptops. For example, Stanford's "Teamspace" lab [12] uses a shared large-screen monitor with individual laptop inputs. However, given the interest in building flexible space, portable large-screen monitors proved too costly.

The fundamental design seeks to provide a space in which neither the pilot nor co-pilot at a workstation would have an advantage, driving the project. So, CL3 workstations are dual input: two mice and two keyboards. Both users at a single workstation must negotiate for control. Given the use of Windows XP for maximum software flexibility for public use, support for simultaneous dual-cursors/pointers was limited. In Section 5, we will address this notion further, due to upcoming collaboration software releases.

To allow for rapid table movement and shape-shifting, each table has a UPS unit that provides "plug-and-play" capability to each workstation. Albeit not a standard use of a UPS, our units have lasted over two years.

3.3 Networking

The original conception for CL3 had a completely shared network, where any group could send their project to any other machine, real-time. Due to the demands of games, the software at the time did not suffice. For schools with limited budgets our solution provided a suitable alternative. CL3's computers all connect to a common network, both wired and wireless. So, students can share files, moving files to machines (including the instructor's). Technologically, this option required relatively little extra cost and still afforded the inter- and intra-group collaboration that we sought.

3.4 Instructors and Facilitators

When designing CL3, we accounted for multiple instructors, especially with the needs for game-design education. Co-instructors, peer facilitators, and teaching assistants all need to weave from group to group. Two catch phrases in education nicely summarize teaching styles: "chalk-and-talk" and "guide-by-the-side." As students develop their game, allowing group time is often more productive than the traditional lecture style. Thus, a classroom space that allows for guided group time greatly assists project development. In CL3, we provide two locations for an instructor podium (shown in Figure 2), portable wireless keyboard, and operator workstation, all linked together. Not only do co-instructors have an ability to demonstrate and communicate examples, but student groups can split duties during presentations (e.g., play a prototype and give PowerPoint slides).

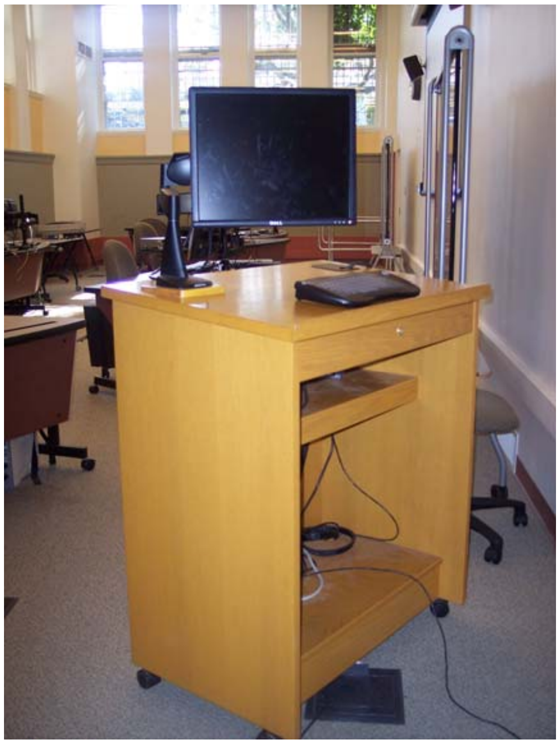


Figure 2. Portable instructor podium

3.5 Layouts

For mobility and flexibility, we endeavored to place as many power outlets in the raised floor as possible. To avoid breaking the budget with a completely electrified grid, we used cutouts of the tables to determine a large variety of table configurations for the given space. Figure 3 shows three configurations for the room. Given a choice of space, we would have preferred a square room, though the rectangular space sufficed.

The configurations in Figure 3 demonstrate three kinds of collaboration for game design groups. From top to bottom,

- Parallel order: provide an approximate traditional environment for lecturing.
- Distinct order: provide semi-private workspace.
- Shared order: provide larger-scale group space or inter-group review areas.

In each of these cases, the configurations facilitate common activities for game-design groups. Moreover, allowing the students to shift table configurations "on-the-fly" provides a degree of fun to the class time, in keeping with the focus on games.

3.6 Resources

One key aspect of the design is the involvement of an independent organization outside of game creation. For Cornell, the University Library provides several services that match the needs of game creation:

- Storage of digital-arts tools (e.g., musical keyboards, recording equipment, drawing tools).
- Storage of games, systems, and accessories.
- Staffing and oversight.

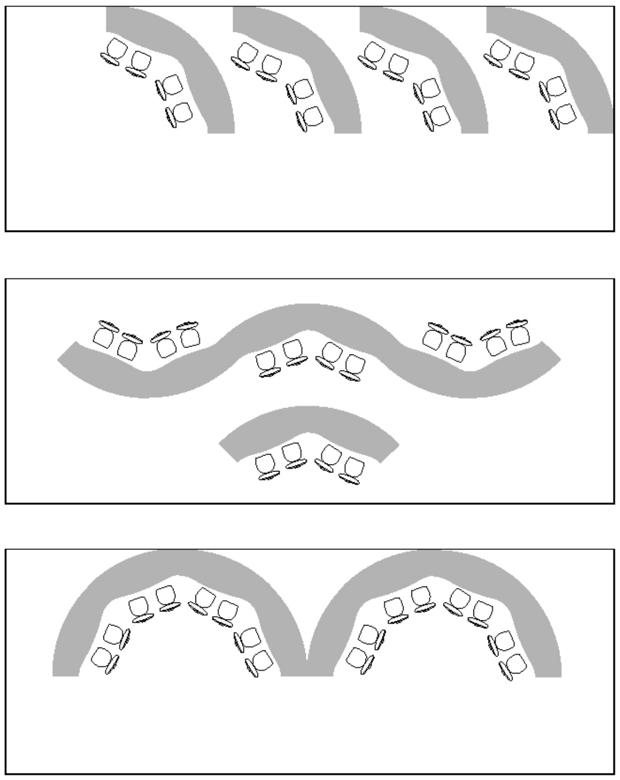


Figure 3. Example table configurations (to scale)

Besides finding space, designing the tables, and finding resources, accounting for constant monitoring was crucial. Allowing students to move tables breaks common conceptions. An organization set up for monitoring, staffing, and loaning—a library—provides an ideal partner for game creation [3].

4. EVALUATIONS

From Fall 2005—Spring 2006, two Cornell courses involved in human-computer interaction and ergonomics offered to use CL3 as an example to study. We present the general findings of both studies in this section [15].

4.1 Human-Computer Interaction

In Fall 2005, the first course project sought to investigate whether or not CL3 facilitates collaboration and suggest improvements to the design. The study involved a questionnaire given during class times. About 55% of the 38 responses came from the introductory game-design students. Key findings that the evaluation reported include the following for the total number of respondents:

- 30% moved tables.
- 46% struggled over the mouse.
- 39% struggled over the keyboard.

In terms of moving the tables, common comments from students included the following:

- "no need" (as the instructors or other students have already picked a suitable arrangement).
- Being unaware of mobility (lack of instruction or demonstration of CL3 tables).
- Fear of breaking something

The report provides further details. Given that 74% reported preferring collaboration, and 83% reported satisfaction, the surveyed students seemed genuinely interested in a collaborative facility. The survey team concluded that while collaboration does indeed occur in CL3, there are weaknesses that need addressing, based on the above findings. One key issue that the team related is the need for communication concerning CL3's mobility and assuaging fears of damaging the equipment.

4.2 Ergonomics

In Spring 2006, an entire advanced course in ergonomics used CL3 as an experimental project to test. This study expanded upon the first team's work, delving into the specifics of the table design, instruction on lab use, measurement of collaboration, and constructive suggestions.

The team surveyed 55 CL3 users and gathered the following data:

- 43% move the tables, with about half of the responses indicating that table movement helps to facilitate collaboration.
- 37% of non-movers were unaware of table mobility.

The survey team points out another interesting notion in terms of conflicting understanding of collaboration during public hours. Outside of "trained" game-design students, other students would sometimes consider the space strictly as quiet, despite CL3's name. Although the library offers neutral ground, it carries this other preconception.

Whereas this survey also concluded that CL3 does facilitate collaboration, they did offer several constructive suggestions to improve the concept:

- Educate users about posture to improve seating and use of input devices.
- Educate users about table adjustments and mobility, especially to improve collaboration. For example, visual/hardware "cues," such as handles would help.

5. FUTURE WORK

In both studies, the surveys reached a small group of students. Our next step is performing a large-scale study with questions focused on table movement and collaboration. In the interim, we intend to focus on educational material (e.g., signs, login screens, lab operator training, instructor training, and workshops) to help demonstrate CL3's capabilities. The subsequent results should prove interesting to see if our proposed efforts will help to break down preconceptions on lab use.

6. ACKNOWLEDGEMENTS

We would like to acknowledge the Cornell University Libraries for funding, ideas, staffing, and continuing support; the Faculty Advisory Board on Information Technology for a Cornell Faculty Innovation in Teaching grant; the Cornell University Academic Technology Center for funding and continuing support; Greg Bronson, David Davies, and Daisy Fan for collaborating on the original design; The Intelligent Information Systems Institute for software donations; and Electronic Arts for additional financial support.

7. REFERENCES

[1] Game Design Initiative at Cornell University (**GDIAC**). http://gdiac.cis.cornell.edu.

[2] Cosgrave, T. Completing the Learning Cycle: Managing the Cornell CreationStation Experiment. In *Managing Library Instruction Programs in Academic Libraries, Selected Papers Presented at the Twentieth-Ninth National LOEX Library Instruction Conference*, pages 41-45, 2001.

[3] Steele, B. New computer lab can morph to fit its users' collaborative needs, *The Cornell Chronicle*, October 7, 2004, http://www.news.cornell.edu/Chronicle/04/10.7.04/CL3.html.

[4] O'Donnell, A., Hmelo-Silver, C., Erkens, G. (Eds.). *Collaborative learning, reasoning, and technology*, L. Erlbaum Associates, 2006.

[5] Bosworth, K. and Hamilton, S. *Collaborative learning : underlying processes and effective techniques*, Jossey-Bass, 1994.

[6] Sharan, S. (Ed.). *Cooperative learning : theory and research*, Praeger, 1990.

[7] Cornell Engineering: Academic Excellence Workshops. http://www.engineering.cornell.edu/student-services/learning/academic-excellence-workshops/index.cfm.

[8] Pair Programming, an Extreme Programming Practice. http://www.pairprogramming.com.

[9] Woodware Designs -- Pair Programming Desks. http://www.charm.net/~jriley/pairall.html.

[10] Scott, S., Sheelagh, M., Carpendale, T., Inkpen, K. Tabletop design: Territoriality in collaborative tabletop workspaces, *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, ACM Press, 2004.

[11] Ryall, K., Forlines, C., Shen, C., Morris. M. Tabletop design: Exploring the effects of group size and table size on interactions with tabletop shared-display groupware, *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, ACM Press, 2004.

[12] Shih, C., Fox, A., and Winograd, T. Teamspace: An Innovative Workspace for Collaborative Academic Computing. *Learning Technology publication of IEEE Society, 6,4* (Oct. 2004), 29-31.

[13] IGDA - Curriculum Framework. http://www.igda.org/academia/curriculum_framework.php.

[14] Hoetzlein, R and Schwartz, D. Computer Game Design as A Tool for Cooperative Interdisciplinary Education, *Proceedings of The American Society for Engineering Education St. Lawrence Section Conference*, Queens University, October 2003.

[15] Cornell Library Collaborative Learning Computer Laboratory (CL3). http://www.cs.cornell.edu/dis/cl3.