Challenges and Opportunities in the Design of Game Programming Classes for a Traditional Computer ......................................................... 4  
   Ian Parberry

Teaching Students to Make Alternative Game Controllers ........................ 18  
   David I. Schwartz

Evolving Interdisciplinary Collaborative Groups in a Game Development Course ........ 25  
   Hollie Bourdreaux, Jim Etheridge and Ashok Kumar

Designing an International Curriculum Guideline for Game Audio: Problems and Solutions ................................................................. 38  
   Ufuk Onen, Richard Stevens and Karen Collins

Studying Commercial Games: Justifying Choices ........................................ 48  
   Katrin Becker

Skins: Designing Games with First Nations Youth ..................................... 54  
   Beth Aileen Lameman and Jason Edward Lewis
We’re thankful for our sponsors’ support; Alice and Klaus Peters at AK Peters gave us a good start, good guidance and allowed us to move from a print journal under their imprimatur to an electronic one when it made sense to do so.

The Golisano College of Computing and Information Sciences generously funded the editorial and publication costs of this effort.

The Game Education Summit of North America is acting as a co-promotional partner and the next issue of JGDDE will publish the “Best Paper Award” winner from the Summit.

Visit our website at: http://www.rit.edu/gccis/gameeducationjournal/ for more information or to submit articles for our next issue.

This work is licensed under a “Creative Commons Attribution-Non Commercial-NoDerivs 3.0 Unported” License. You are free to share the work under the following conditions: You must attribute the work in the manner specified by the author or licensor; You may not use this work for commercial purpose; You may not alter, transform, or build upon this work. See http://creativecommons.org/licenses/by-nc-nd/3.0/ for full details.
Dear Readers,

Welcome to the first issue of the *Journal of Game Design and Development Education* (JGDDE).

The JGDDE is a peer-reviewed journal that covers all aspects of teaching the art, craft and science of game design and development to students in and out of a higher education setting. We plan to cover a range of work in our emerging field, as well represented by the content in our first issue. For example, Ian Parberry’s piece covers the emergence and growth of his pioneering work in the field teaching game programming in a university setting over the better part of two decades while Beth Lameman’s article looks at an evolving effort to teach game design and development to pre-university indigenous youth as a way of recapturing and owning their heritage while David Schwartz’s effort chronicles the early design of a mixed hardware and software course to create new types of game controllers.

Games are a multidisciplinary medium and those of us who teach their creation come from as broad a set of backgrounds as you can find. JGDDE will celebrate that diversity of skills and backgrounds. We look forward to having you as readers and as contributors.

Stephen Jacobs
Assistant Professor
Editor-in-Chief
Challenges and Opportunities in the Design of Game Programming Classes for a Traditional Computer

Ian Parberry
University of North Texas
ian@unt.edu

ABSTRACT
Game programming classes have been offered at the University of North Texas continuously since 1993. The classes are project based, and feature collaborative coursework with art majors in UNT’s College of Visual Arts and Design. We discuss the design that enables them to provide training for students intending employment in the game industry without sacrificing academic educational depth or the educational needs of mainstream computer science students.

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 controversial, much-challenged, and difficult move. There were no course materials, books, or web pages available. Interestingly, the only objections were from faculty — both the students and the administration were in favor of the class. Objections were raised about the industry-driven focus of the class and the perceived trivial nature of entertainment computing. Since 1993 the initial game programming class has evolved with the fast-moving game industry, and spawned more classes and an undergraduate certificate in game programming. After more than a decade and a half 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. Our early alumni are moving upward in the game industry to programming director and producer level positions.

In sharp contrast to their early history, game programming classes are now gaining acceptance in academia (see, for example, Feldman [7], Moser [11], Adams [1], Faltin [6], Jones [9], Becker [3], Alphonce and Ventura [2], and Sindre, Line, and Valvè [15]), resulting in a proliferation of new classes and programs nationwide, and a move towards a professionally recommended curriculum in game studies [8]. Unlike institutions such as DigiPen, Full Sail, and SMU’s Guildhall that offer specialized degrees or diplomas in game programming, we offer game programming as an option within a traditional computer science curriculum. Keeping in mind that many institutions are starting game programs, and that many of them are designing their curricula in an ad hoc manner, the purpose of this paper is to share some of what we have learned from experience over the last decade and a half by describing our game programming classes, the design philosophy behind them, and some of the potential pitfalls to be avoided.

We begin by discussing game industry needs in Section 2, and some important issues in the design of a game programming class in Section 3. We will discuss our introductory class in more detail in Section 4, and our advanced class in Section 5. Section 6 describes two new classes and a recently introduced undergraduate Certificate in Game Programming at UNT. Section 7 discusses the need for separate lab space for use in game programming classes. The Conclusion contains some data about the impact of game programming on the computer science program at UNT. A preliminary version of the material in this paper appeared in [12, 13, 14].

2. WHAT GAME COMPANIES WANT
Game companies want C and C++ programmers with general competence in technical subjects typically found in an undergraduate computer science program such as
programming, computer architecture, algorithms, data structures, graphics, networking, artificial intelligence, software engineering, and the prerequisite math and physics classes. In addition, they usually require evidence of the following skills and experience:

1. Work on a large project, that is, larger than the typical “write a program for a linked list” kind of programs that are typically used as homeworks in programming courses.
2. Creation of a game demo or two, something nontrivial that plays well and showcases the programmers’ ability. This shows that the applicant is devoted enough to have spent their own time to create something, and has the perseverance to see it through to completion.
3. That the applicant is a “team player”, somebody who can work in a multidisciplinary team with other programmers and nontechnical people such as artists.
4. That the applicant can learn independently, because the game industry continues to push the boundaries of what can be done using new computer technology.
5. That the applicant is well-versed in game technology.

While our undergraduates can technically learn enough about the game industry in general and game programming in particular from books to satisfy most of these requirements, our game programming classes are designed to help students achieve them more effectively than they could alone, and encourage them to higher levels of achievement. The requirements listed above are similar to the “Ideal Programmer Qualities” listed by Marc Mencher [10]:

- Team Attitude: The ability to work in a team to get the job done, without unnecessary friction.
- Self-starters: The ability to work without constant supervision.
- Follow-through: The ability to see tasks through to the end.
- Communication: The ability to communicate with programmers and nonprogrammers.
- Responsibility: The ability to take responsibility for things that have been done, and are to be done in the future.

In addition to satisfying the needs of aspiring game programmers, we quickly found that the game programming classes are attractive to general students as a capstone experience, paralleling the experience of Jones [9]. Indeed, our class projects meet most of the requirements of the capstone project CS390 in [4].

Other employers are also attracted to students who have experience with a group software project with nontechnical partners. Feedback has suggested that game demos created with artists tend to show better in interviews than the typical project created by programming students alone.

3. DESIGNING A GAME PROGRAMMING CLASS

There are a number of key decisions in the design of a game programming class that affect the outcome in a fundamental way:

1. Should the classes be theory based, or project based?
2. What software tools should be used?
3. Where do programming students find art assets?
4. Should students be free to design any game in any genre, or should their choices be limited?
5. Should students write their own game engine, or work with a pre-existing engine?

On the first question, the options were either a theory class with homeworks and exams, perhaps augmented with small programming projects, versus a project class in which the grade is primarily for a large project programmed in groups. We chose the project option, understanding that students would come out of the classes with two substantial game demos that will play a major role in their first job interview in the game industry.

On the second question, the Department of Computer Science and Engineering at UNT was until recently almost exclusively Unix based, with g++ being the compiler of choice and graphics programming taught using OpenGL. We chose to use Windows, Visual C++, and Microsoft DirectX instead for the game programming classes. We have in the past attracted a substantial amount of criticism from academics over our choice of DirectX for the graphics API and Visual C++ for the compiler over OpenGL and g++ respectively. In response, the author wishes 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. Students benefit from using in class the same tools and techniques used by a substantial fraction of the game industry.

4. Students should be exposed to as many different compilers and APIs as possible during their academic tenure (this is encouraged in Section 10.2.2 of [4]). 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.

On the third question, that of art assets, the obvious choice is to have students take advantage of the free art on the web. Our experience is that students benefit substantially from working with art students. Not only do they benefit professionally in learning to collaborate across disciplinary lines, but they are strongly motivated to produce better code when they see better art. We will describe more of our collaboration with the College of Visual Arts and Design at the University of North Texas in Section 4.

On the fourth question, on whether students should be allowed to design and implement a game in any genre, our experience is similar to that of Sindre, Line, and ValvÊ [15]. Constraints on the type of game being created (as in [1, 2, 3, 6, 7, 9]) may seem attractive from a managerial point of view because, for example:

- It allows for a more shared experience, enabling students to learn and collaborate across group lines.
- It gives the flexibility to reassign group membership in response to late drops and overheated group dynamics.
- It allows the art class to streamline their process by using a pipeline art production line where necessary.

However, we have found that the element of creativity, student morale, the quality of the resulting games, and the outcomes all suffer when any kind of constraint is placed on the game being developed.

On the fifth and final question, whether to teach with a pre-existing game engine and tools or to have the students create their own custom game engines, the pre-existing game engine option may seem the most attractive at first for several reasons:

- It allows the students to “stand on the shoulders of giants”, that is, to achieve more than they can on their own by leveraging existing code.
- It prepares them for the game industry, where they will likely find themselves working on an existing engine, or at least with an existing code base.
- It is easier for faculty to teach from an existing game engine than to teach students to create their own game engines.

However, we have found that the arguments for not using a pre-existing game engine are more compelling in practice:

1. Teaching students to use a single game engine simply trains them in its use. The learning curve in a single 15-week class is typically so steep that they run the risk of spending their time wrestling with code rather than developing general skills.

2. Existing game engines for educational use tend to be poorly documented, low in features, and unstable. Students find that they spend most of their time trying to force a recalcitrant engine to do what they want it to do, or coding around obscure bugs. They are often resentful of the fact that their grade depends on somebody else’s ability to write code, particularly when it is obvious that somebody “else” writes bad code.

3. The code for existing game engines is generally production code, code that is designed to run fast and be maintainable, rather than teaching code, which is further designed to teach basic concepts.

4. Students who are exposed to the internal code of a simple game engine are able to more quickly pick up the details of the proprietary game engine at their first job.

5. Students entering the game industry will most likely spend the majority of their professional lives modifying and making additions to somebody else’s code. This is the last opportunity that they will have to devote major slices of their time on their own game engine.
For these reasons we opted to teach game engine programming with the class project being to create a game engine using some standard utilities, rather than modifying a free or proprietary game engine.

4. GAME PROGRAMMING 1

Our introductory game programming class, now called Game Programming 1, was introduced in 1993 as a special topics class. Despite some initial resistance from faculty curriculum committees, it received its own course code CSCI 4050 and catalog entry in 1997, effective in Fall 1998. It is offered once a year in Fall semesters.

The intro game programming class started out in 1993 as a 2D game programming class for DOS, changed to DirectX 3, and has been updated annually to keep pace with each new release of DirectX, from DirectX 5–9. It is a project class. Students must attend lectures, but the final grade is for a game programmed in teams. To make this as real-world as possible, the students are given an ill-defined objective, as recommended in Sections 10.3.2 and 10.4 of [4]. In the first class meeting, the students are shown a slide that describes the grading system as follows:

A "It really knocks my socks off."
B "It's a pretty cool game."
C "It’s an OK game."
D "It’s not there, but at least you tried."
F "You really blew it off, didn’t you?"

Two kinds of points are awarded: completeness points and techno points. Completeness points are awarded for things such as:

- Does it run without crashing?
- Are there few (preferably no) bugs?
- Does it have an intro, a title screen, a credits screen, a menu screen, help screens?
- Does it play with the keyboard, mouse, and/or joystick?
- Does it have sound support?
- How is the game play? Is it fun?

Techno points are awarded for implementing technology not covered in class. Examples include, but are not limited to:

- MP3 instead of WAV format sounds
- Showing video clips using DirectShow
- Lighting effects (eg. directional light, sunset, shadows, lens flare)
- Pixel and vertex shaders
- Network play using TCP/UDP/DirectPlay

Game Programming 1 is taught in parallel with a game art class taught to art students in the College of Visual Arts and Design at UNT. Part of the art students’ grade is to produce the art work for a game programmed by our students.

To encourage group synergy we teach both the art and programming classes at the same time in different rooms in the same building. Classes run for 3 hours in the evening, and the final hour is reserved for group meetings between the artists and programmers. We have experimented with running the classes at different times, and at the same time in different buildings, resulting in both cases in a massive drop-off in meeting attendance, and a corresponding decrease in the quality and number of completed games at the end of the semester.

Allowing students to form their own groups based on common interests has proved to be the best way of maintaining interest and excitement about the projects. At the end of the first class we take the students in both classes — typically 30–35 programmers and 15–20 artists — into a large classroom and have them stand up sequentially and introduce themselves to the class, asking them specifically to talk about what kind of games they like to play, what kind of game they would like to create, and any prior experience. We then allow them to wander around at random, and come to the front of the room when they have formed a group of two programmers with one artist. We have found that the amount of artwork required by a simple sprite game is within the ability of a single art student to create in a single class. However, we always have one or two groups of odd sizes, which are handled in a case-by-case manner.

Final projects are presented to the instructor in a series of 30-minute slots over two days during Finals week. They are graded on the final executable only, the instructor does not look at source code. After demonstrating the game and allowing the instructor to play, the students are quizzed on their individual contributions to the game, to ensure that they actually did what they claimed to have done. Grading on the executable only is a radical departure from other classes that the students have taken in the computer science curriculum, but is an important real-world constraint.

Starting in the Fall 2002, we instituted a game contest for students in CSCI 4050 and the associated game art class. Entry is strictly optional, and does
not contribute to grades. The contest is judged by a panel of 4 or 5 local representatives from the game industry. Prizes are donated by Texas game and publishing industries, ranging from the more expensive books and games to less expensive T-shirts and posters. The contest lasts 2–3 hours, and is open to the general public.

Holding the contest in the final week of classes, approximately one week before the deadline for turn-in of the final projects, encourages students to start coding early. Previous attempts at getting students to get started early were focused on checkpoints and documentation. Preliminary progress reports and play testing dates proved to be positive up to a certain point, after which insistence on more checkpoints and documentation took up valuable time that could more profitably be spent creating the actual game. The game contest is a much more positive way of reinforcing the final deadline.

Our proximity to the DFW metroplex with its high density of game development companies makes it easy to attract guest lecturers. We encourage visits by teams from development houses including artists, programmers, and designers, and have them speak to the combined class of artists and programmers. Rather than technical presentations, we have guest lecturers speak about what it is like to work in the game industry, what it takes to get their first job, and what educational paths the students should pursue. Typically, we have two or three presentations per semester.

We have used a teaching technique called incremental development. Rather than going through the DirectX documentation or tackling a single monolithis game engine, we teach using a basic game called Ned’s Turkey Farm, a simple side-scroller in which the player pilots a biplane and shoots crows (see Figure 1). 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.

The code is currently organized into a sequence of eleven code demos. Each demo is built on top of its predecessor. A file difference application, such as windiff is used in class to highlight the changes in code that must be made to add the new features. An average of one demo is presented per week.

Figure 1: Screen shot of Ned’s Turkey Farm.

A typical class begins by running the demo and pointing out the new features, followed by a powerpoint slideshow describing the new demo, its new features, the theory or principles behind them, and any implementation details, but at a high level without getting bogged down in the code. This is followed by running windiff and going through the code changes in more or less detail depending on the complexity and difficulty of the code. Often, we run Visual C++ to show students in real time the effects of minor code tweaks.

The code is organized into twelve incremental demos as follows:

- Demo 0: Getting started
- Demo 1: Introduction to Direct3D
- Demo 2: Scripting and debugging
- Demo 3: The sprite
- Demo 4: Managing objects
- Demo 5: AI
- Demo 6: The game shell
- Demo 7: Sound
- Demo 8: DirectInput
- Demo 9: The joystick
- Demo 10: Playability
- Demo 11: Persistence

4.1 Getting Started: Demo 0

Demo 0 is, for many students, their first Windows application. It fires up a black full screen window with a text prompt and waits for user to hit the ESC key to exit. Students learn how to register a full screen window, create it, draw graphics on it using the Windows GDI, respond to user keyboard
input, and shut down the application gracefully. They are introduced to the concepts of Windows messaging and the message pump.

4.2 Introduction to Direct3D: Demo 1
Demo 1 is our first Direct3D application. It starts Direct3D and displays a shoebox background consisting of a floor and a backdrop. It may be run either fullscreen or windowed by changing a global variable.

4.3 Scripting and Debugging: Demo 2
The executable for Demo 2 looks the same as Demo 1, but there is a lot more under the hood. We add XML scripting using TinyXML. Now the students can change the behaviour of the demo by editing Ned.xml in an XML editor instead of having to recompile the code. This allows the art students to change some of the settings in the game easily.

Debugging may seem like a moot issue until students seriously start creating their own game. The problem with DirectX fullscreen debugging is that DirectX takes complete control of the screen. The Visual Studio debugger is the first line of defense, but some bugs will crash the debugger. The debug code in Demo 2 will let the programmer read debug output in real time in a client console application on a second monitor, or on the screen of a second computer. It will also save the debug output in a file, and display it in the Visual Studio debugger’s Output window. It accepts printf-like parameters, and it can be disabled with two keystrokes by commenting out a single #define.

4.4 The Sprite: Demo 3
Demo 3 is our first attempt at simple real-time animation. A plane sprite moves across the background. F1 tabs between game view and eagle-eye view, in which the camera pulls back so the programmer can see what is happening “behind the scenes.”

4.5 Managing Objects: Demo 4
Demo 4 has more objects, managed by a sprite manager and an object manager. The object manager is a first draft only in that it can create objects but not yet delete them. Sprites are now animated with multiple frames of animation. There is now a continuous, infinite scrolling background with the camera in motion to keep the plane in the center of the screen.

4.6 Artificial Intelligence: Demo 5
In Demo 5, crows now have simple rule-based artificial intelligence with some randomness thrown in to make them behave slightly differently. They try to avoid the plane as much as they can given a limited attention span. Flocking can be seen as emergent behaviour. The plane fires a bullet when the player hits the space bar. Bullets have a fixed lifetime. When a bullet hits a crow, the crow explodes and turns into a falling corpse, which disappears when it hits the ground. The object manager now has full functionality, in that it can now delete objects and recycle their space.

4.7 The Game Shell: Demo 6
In Demo 6 there is a game shell wrapped around the game engine, consisting of an intro sequence (a logo screen, a title screen, and a credits screen), and a menu screen. The player can click out of any of the intro screens. From the menu screen one can play a game or quit by pressing the appropriate key on the keyboard. After each game, the player is returned to the menu screen after they kill the last crow, or preemptively by pressing ESC. From there the player can re-enter the game engine. We show how to gain direct access to the back buffer to blit the intro screens there directly.

4.8 Sound: Demo 7
Demo 7 introduces sound, managed by a sound manager class. Since DirectSound will not allow a sound to be played multiple times simultaneously, the sound manager keeps multiple copies of each sound, sharing the sound data, and automatically selects the first copy that is not currently playing. The plane engine sound loops.

4.9 DirectInput: Demo 8
Demo 8 uses DirectInput to give faster access to input hardware. We start by using it for the keyboard and mouse instead of using the Windows messages like in previous demos. We also add some 2D animation for clickable buttons on the menu page. The mouse is used to press menu screen buttons, as a joystick, and to fire the gun. The buttons on the main menu are drawn as 2D sprites. There is a custom DirectInput mouse cursor.

4.10 The Joystick: Demo 9
Demo 9 adds joystick control to the DirectInput code, and adds a device selection screen with radio buttons.

4.11 Playability: Demo 10
Demo 10 adds more complexity to the game, and introduces the drawing of text in screen space. We add multiple levels, with more crows as level number increases, and a success screen in between levels. Player can now be hit by crows, which reduces health
The player’s health and number of lives are managed by a score manager. We add text on the screen showing the level number, number of crows, health, lives and score.

4.12 Persistence: Demo 11
Demo 11 stores in the Windows registry the game settings that should persist from one execution of the game to the next. We start with the high score list and the initial input device. Checksums are used to detect tampering. New code is added to display the high score list, enter a new name typed in by the player into high score list, and manage the stored high score list.

5. GAME PROGRAMMING 2
The advanced game programming class, now called Game Programming 2, was introduced in 2000 as a special topics class. It received its own course code and catalog entry in 2003, effective in Fall 2004. It is offered once a year in Spring semesters. The introductory game programming class is a prerequisite.

Game Programming 2 covers interactive real-time 3D animation. The grade for the class is for a 3D game created in groups, a typical group consisting of two programming students and two or more art students from the concurrent art class taught in the College of Visual Arts and Design. An increase in the number of art students per group over Game Programming 1 is required because of the increase in work needed to produce 3D models. Programming students are also permitted to use art work from the web, but this has a number of disadvantages, including:

1. Quality: Models often have inappropriate triangle count (too high or too low) and topological defects (degenerate, detached, or sliver triangles, for example).
2. Post-processing: Models often require significant post-processing, for example, they may not be located at the origin, and may have triangles listed in the wrong order for back-face culling.
3. File format: Models are often posted in various formats, for which loaders must be written or adapted from other code. File format converters exist, but they are typically expensive, buggy, or produce low-quality results that require post-processing.
4. Motivation: Programming students are more excited about their games, and hence better motivated, if they can have custom art created on-the-fly in response to group design decisions. Our experience has shown that downloading art from the web typically results in dissatisfied students and lackluster games.

The biggest drawback to having students create a custom 3D game engine is the amount of work involved. It is imperative that students use some basic utilities, for example, the D3DXUtil library provided in the DirectX SDK. We have used a set of improved utilities (including a basic rendering engine, a model importer, and implementations of Euler angles, matrices, vectors, quaternions, axially aligned bounding boxes), published in Dunn and Parberry [5]. We have had the greatest success from using a simple game engine which we call SAGE. Students who wish a different experience are permitted to download and use any free game engine.

As with Game Programming 1, one of the key elements of this class is the excitement generated by having programming students work with art students. In the advanced game programming class, however, there is a significant barrier to communication between the artists and the programmers: the model file format problem. The art students can work with sophisticated 3D modeling tools such as Maya, Lightwave, and 3D Studio Max, but unfortunately the native file formats generated by these programs are proprietary and difficult to load. The programming students work with Microsoft DirectX, which has strong support for its own file format, called “.x.” We have found this disconnect between the file formats used by the artists and the programmers the most difficult gulf to bridge. All of the plug-ins, exporters, and file converters we have tried have the disadvantage of being expensive or unreliable, or both. Worse still, they fail annually when the College of Visual Arts and Design upgrades its subscription to the 3D modeling tool of choice, leading to a last-minute scramble to provide software tools in time.

To help mitigate these and other problems, we created a simple academic game engine called SAGE. SAGE is designed to provide the minimum requirements for a game, which are:

1. A 3D world
2. That a player can explore in real time,
3. With interesting objects in it,
4. With which the player can interact.
The key adjectives in the preceding list 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 reasonably 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.

SAGE brings the experience of incremental development 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.

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 [5]. Common includes the following utilities:
  - AABB3.cpp, AABB3.h: Axially aligned bounding boxes.
  - Bitmap.cpp, Bitmap.h: Bitmap image reader.
  - CommonStuff.h, CommonStuff.cpp: Common stuff that doesn’t 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 [5]).
  - RotationMatrix.cpp, RotationMatrix.h: Rotation
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.

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

<table>
<thead>
<tr>
<th>MODULE</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common framework</td>
<td>13,729</td>
</tr>
<tr>
<td>SAGE</td>
<td>13,469</td>
</tr>
<tr>
<td>tiny XML</td>
<td>4,883</td>
</tr>
<tr>
<td>Ned specific</td>
<td>2,750</td>
</tr>
<tr>
<td>Total</td>
<td>34,831</td>
</tr>
</tbody>
</table>

Table 1: Number of lines of code in SAGE.

5.1 Model Importation and Display: 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 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).

5.1 Model Importation and Display: Demo 0
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 [5], 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.

5.2 Terrain Input and Rendering: 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.

5.3 Shaders: 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.

Direct-Input. 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.

5.5 Collision Detection: 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 boxes 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 4: Demo 1 showing ocean and island.

Figure 5: Demo 2 showing terrain reflections and texture blending.
5.6 Particle Engine: 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.

5.7 Sound: Demo 6
Demo 6 covers stereo 3D sound using DirectSound

6. THE GAME PROGRAMMING CERTIFICATE

In 2009 we added two more classes in game
programming. The first is a class on game math
and physics, created in response to student
complaints that Game Programming 2 had too
much math content. This class was taught for the
first time in Fall 2009. Material covered includes:

1. Intro to 3D math
2. Vectors
3. Matrices
4. Intro to game physics
5. Particle motion
6. Newtonian mechanics
7. Multivariate calculus
8. Floating point numbers
9. Ordinary differential equations
10. Quaternions
11. Rigid body mechanics
12. Collision detection
13. Deformable bodies
14. Lagrangian mechanics

The second new class is a Topics class in which the
students specialize in the advanced topic of their
choice, working in groups. This helps students to fulfill
Requirement 4 of Section 2. Topics undertaken recently
include water animation, procedural clutter, procedural
quest generation, shaders, shadow rendering, and
cell processor programming. Undergraduates are
encouraged to work with graduate students, which
exposes them to academic research and increases the
probability that they will return for a graduate degree.

Completion of the four classes, Game Math and
Physics, Game Programming 1, Game Programming
2, and Topics in Game Development entitles
students to an undergraduate Certificate in Game
Programming, starting 2010.

7. THE GAME PROGRAMMING
LABORATORY

Providing a dedicated game programming laboratory
proved to be an important requirement when
developing the game programming classes. The
standard open laboratories provided by universities
are unsuitable for game programming classes for
several reasons.

- The process of updating software is typically
  slow and cumbersome, since open labs catered
to a wide range of students from various
disciplines.
- The hardware, in particular the graphics cards
  are not up to expected standards.
- Students developing and playing games
  are distracting to other lab users, and game
development students typically run afoul of any
rules against game playing, despite the argument
that it is required for a class assignment.
- Since game students are required to work in
teams with other programmers and artists,
a substantial amount of team meetings
during development and debugging need to be actually at the keyboard. Open labs are typically designed for students who work alone, and in general have a policy of silence.

- Game development students are excited about what they do, and in consequence tend to be rowdy and loud.
- A dedicated game development space provides a place where students can meet and work with other students who share their interest. The area becomes a crucible for independent learning and experimentation that inspires students to greater efforts and achievements.

The location of the lab is important. Currently it is across the hall from the author’s office, which since the office and lab doors are usually open, facilitates the author’s interaction with students. The space is located away from the other faculty offices where the noise generated by the students will not cause a problem for more sober and sensitive colleagues.

To ensure that the lab software is kept up-to-date and that the hardware does not get stolen, we hire a student as lab monitor. He or she is required to keep the lab open for 20 hours per week. The job usually goes to one of the alumni of the game programming courses so that he or she can provide help to the current crop of students. In addition to this paid position, several trusted students also have lab access on the understanding that they provide additional informal open hours by allowing other students to use the lab while they are there. The lab door is fitted with an electronic card-swipe lock that monitors and records entry. Figures 8 and 9 show the lab layout.

We started with a small room with three computers in 1993. As space became available, we moved to larger quarters in 1994 and 2001, finally moving into the current location in which we have approximately 570 square feet, 12 computer workstations, and a file server. The computers are on a special 2-year upgrade cycle, as opposed to the standard 3 or 4-year upgrade cycle, paid for by course fees from the game programming classes. Dual monitors are a must for on-screen debugging.

We found that the best organization for the lab is the “bull pit” model, with computers around the outside of the room. The center of the room has tables arranged for conferences and meetings and for laptop use. The computer workstations are connected to the wired network through a router that serves as a firewall to prevent packet floods generated by rookie game programmers from swamping the building’s network. One corner of the room has been set up with a sofa, a TV, and several game consoles to foster interaction between students and provide an inviting club-like atmosphere. A typical day will find people playing a game on the console, playing networked PC games, writing code for their own game engines, and engaged in vigorous discussions on subjects ranging from linear algebra to graphics using one of the many whiteboards.

Figure 8: Current laboratory layout.

Figure 9: Planned laboratory layout. Unlabelled rectangles are tables.
8. CONCLUSION

We have had success over the last decade and a half with a two-course sequence in game programming in a traditional computer science undergraduate curriculum. The classes are project based, and feature collaborative work with art students in the College of Visual Arts and Design. In addition to training aspiring students for the game industry, the classes also provide a capstone style project experience for all computer science students.

Our experience with game industry involvement is that while companies are, with a few notable exceptions, reluctant to provide any sort of concrete support for game development programs in academia, individuals are much more positive. Requests for guest lecturers from industry almost always results in great presentations from motivated, knowledgeable, and experienced game programmers and artists.

The author believes that game programming classes at UNT have had a significant effect on student enrolment and retention. Student numbers are currently dropping in computer science and engineering programs nationwide, which is mirrored at UNT (see Figure 10). Figure 11 shows enrollment figures for the introductory and advanced game programming classes from to present compared to the total number of Bachelor’s degrees awarded by the Department of Computer Science and Engineering. The introductory class was capped at 35 students in 2002. We see that a substantial fraction of graduates have taken game programming.

SAGE can be downloaded from http://larc.csci.unt.edu/sage, and was funded by a grant from Microsoft Research.

9. REFERENCES


Teaching Students to Make Alternative Game Controllers

David I. Schwartz
Rochester Institute of Technology
dis@mail.rit.edu

ABSTRACT

This paper describes a preliminary approach to teaching students about game design and development by making a custom game controller in a single course. RIT’s first seminar in alternative game controller design introduced students to game input mechanisms and encouraged innovation. By working with hardware, students explored physical design issues to improve their overall game design skills. The course demonstrated early success, as evidenced by the 2009 Game Developers Conference in which a banjo controller and game created by the students garnered accolades. To help other instructors develop a similar course, this paper addresses the motivation for, and components, of the course design. By combining elements of reverse engineering, interfaces, and game design, students can indeed design and successfully make custom game controllers. The paper contains a summary of one of the two final student projects: a deformable surface. The postmortem notes that despite the course success, future versions will need more time for projects. The paper concludes with a discussion of future directions.

1. INTRODUCTION

1. Game Design and Development Education

Not so long ago, the idea of forming a unique academic field devoted to games seemed folly, and to some, even heresy. However, a variety of fields, programs, and courses spanning game studies, design, development, and others have blossomed [1]. A flurry of academic papers has explored defining/refining course topics with recent works exploring connections with education, simulation, and more, e.g., [2, 3]. Theory and concepts continue to emerge, sometimes supplanting industry focus, resembling similar “splits” seen in the sciences and engineering between theory and application. With students trying to enter the competitive game industry, academic programs that focus on skill building and creativity can retain their popularity. To stand out, students need new ways to explore game design and development.

2. Game Controllers

The game input device, or game controller has a fascinating history mostly documented in a variety of websites, academically it has only been examined in some papers, and very few books [4-9]. The controller occupies a paradoxical position in computer game studies. Although it is central to gameplay experience—it marks physically the difference between play with a game and merely watching a screen—it goes largely unreflected on by gamers and in gaming literature. [10] But, game controllers have roots as a variety of mechanical input systems and interfaces, like knobs and buttons. Console controllers essentially evolved with ergonomic improvements and additional functionality [11].

With respect to game input and player control, modern controllers involve a combination of abstraction and mapping [4]. For example, consider how games frequently assign the A-button to jump [12]. Although press-and-release simulates a vertical action, jumping does not necessarily imply crouch (the press) and jump (the release). Thus, a button infers an abstraction, triggering a loosely connected game action. On the other hand, a mapping (or natural mapping) implies a direct analog from controller to action. For example, pushing upwards
on an analog stick would directly move an avatar up on the output device—even with axis inversion, the action still maps logically, albeit in the opposite direction.

3. Alternative Game Controllers
Although the term alternative game controller seems relatively accepted, alternative implies that a regular game controller exists [13-16]. One could contend that the PS3/Xbox360 style controllers (and recent predecessors) are standard. Each new console seems to alter previous controller designs, sometimes starting anew, as with the Wii. In fact, a term to signify a standard seems elusive, though some suggest “pronged.” Still, a definition of standard schemes has emerged:

- Console games: directional pad, analog stick(s), shoulder buttons, face buttons.
- PC games: keyboard and mouse.

Thus, an alternative controller deviates from these common components and forms, tending to provide greater (or different) mapping fidelity. For example, numerous steering wheels and pedals have facilitated driving games. Dance pads are another early successful example, though other genres, like RPGs, can use the pads [17]. Two modern examples have produced tremendous attention for alternative input: Guitar Hero (and successive music/rhythm games with their controllers) and motion control via Wii, Kinect, and Playstation Move.

Cost is the greatest deterrent to alternative controllers, ranging from development to manufacturing to consumer. Until recently, alternative controllers seemed a niche market, catering to specific genre, like racing. However, the advent of motion control and interest in “exergaming” has led to a variety of products, as listed above. As seen below, the question of acceptance of controller seems to tie to how it facilitates new gameplay, creating new experiences that more easily map to natural motion.

4. Background
Game design and development students can take risks that industry might avoid, leading to innovation. By exploring new ideas without commercial pressure, many student projects have helped introduce very influential games, e.g., Portal and others. Likewise, motion control has spurred on much of the recent interest in alternative controllers. For example, numerous “cracks” of the Wii Remote have shown that many students and people outside of academia have already tackled the challenge of adapting motion-based game controllers [18, 19]. In fact, Jonny Lee’s WiiMote Head Tracking project predates recent advances in new motion controllers and seems to continue to inspire students [20].

Given the recent popularity in alternative game controllers, game companies continue to innovate their controllers. Many research projects explore design innovations, often involving areas of tangible interaction and haptics. For example, one attempt to recreate the EyeToy for PC [21] foreshadows later development with Microsoft’s Kinect. In fact, one of the student projects in the experimental course used an EyeToy (Section 3). Other innovations include touch, especially for magnification [22] and body movement [23]—the innovations are ongoing.

5. Motivation
Alternative controllers offer excellent opportunities to explore game design innovation. Students can either refine interfaces to facilitate established genres or discover new kinds of games. Addressing a common complaint of limited appeal, students might expand the range of games that the alternative controllers serve. Ultimately, students can gain greater understanding of the user experience by designing for players.

Providing a hardware experience can also expose software-oriented students to more low-level techniques, “looking under the hood,” so to speak. Greater experience with drivers, C and assembly programming, signal processing, USB, networking, and other technology can enhance a student’s resume, especially those looking for development positions. For those looking outside of the game industry, knowing how to analyze and design a controller will provide even greater opportunities, e.g., consumer electronics, medical, manufacturing, robotics, and military.

This work can also provide excellent multidisciplinary opportunities. Apart from game design and development fields, entertainment engineering and entertainment technology continue to grow [24]. What better way to gather a variety of students than with a game controller, which embodies a mechanical, electrical, software, design, and business product? Since the game industry now uses a variety of controllers, all of these students can greatly benefit from learning to work with each other, mirroring industry practice. With excellent technical abilities, inspiring engineers to consider games in later courses could boost the variety of prospective graduate students.
2. COURSE DESIGN

1. Objectives

In RIT’s Winter quarter (December 2008-March 2009), the course ran as a preliminary, experimental seminar with eleven students. The exploration posed these questions:

1. What kind of new game controller would facilitate a new game experience for which that controller most logically abstracts and maps player input?
2. Which game controllers can students most easily dissect, analyze, modify, and use?
3. Which sensing technologies (e.g., accelerometers, gyroscopes, and other devices for embedding computing and sensing) most easily connect to game controllers?
4. Can teams of students appropriately scope their ideas?

The interdependent problem of game and controller design defined the core challenge for students to overcome.

3. Culminating Project

To address the course outcomes, the project presented a three-fold design challenge to all of the students:

- Game: Are there game mechanics that a current controller does not naturally facilitate?
- Controller: What unique input device would serve those mechanics?
- Interface: How will the controller work?
- All three design components involved interdependent thinking—a deliberately challenging problem requiring significant cycles of iteration. At a minimum, students needed to adhere to the following constraints for the controller:
  - Uniqueness: It must not already exist;
  - Interaction: It must be good for a particular type of game/primary mechanic;
  - Reality: It could be constructed of off-the-shelf parts, including other controllers;
  - Platform: It must connect to a PC;
  - Durability: It must last during an all-day institute showcase; and constraints for the game:
    - Gameplay: It must exhibit as much interaction as an early game prototype;
    - Length: It must have at least one level;
    - Interaction: It must exhibit a genre/primary mechanic that the controller facilitates;
    - Platform: It must play on a PC;
    - Elements: It did not need to have narrative, plot, or other elements except when required to demonstrate the controller.

For the interface, the students needed to ensure deliberate and useful abstractions and mappings to the controller. Due to existing game design courses and experiences, the students could start with controller design, though the game required roughly parallel development. Ultimately, the project still involved providing compelling gameplay, which emphasized that the controller design must somehow serve the game design.

4. Example

Thus, the class split roughly into two groups around specific projects: a banjo controller [41] and a surface controller. Student final reports, press releases, and pictures are all available [42]. This section describes the “spandex garbage can”: a tactile, multiplayer, force-feedback, 3-D surface controller, as shown in Figure 1.

What’s inside? Starting from the notion of interacting with a visual environment and faced with the need to make something physical, the students suggested a fabric-based controller. Through a series of experiments and more brainstorming, the students used an intriguing idea: place a light source inside a dark, opaque rubbered trash can with a camera—an EyeToy, in fact. After melting the first barrel with holiday lights, the students opted for a “strip light” and a piece of diffuser used in ceilings, as shown in Figure 2.

---

Figure 1. The “garbage can” controller

---
How does the controller actually work? Wherever a player touches the spandex, the light underneath becomes essentially more intense, creating “bumps” that the camera registers. The GPU can very quickly render the height of each section of the surface. As noted by the students, although the edges do distort, the majority of the surface response is quite accurate.

The team essentially split into controller and game sub-teams to maximize the dwindling project time. The game team opted to exhibit the multiplayer and 3-D aspects via Sand Havens, a casual RTS game (Figure 2). Players push down on the surface to raise land so that bad creatures avoid eating good creatures. But since the game conserves mass, raising land simultaneously lowers other areas. Lands that sink below sea level fill with water and threaten the good creatures.

3. POSTMORTEM

1. What Went Right

A number of factors in the course design and a certain degree of luck of student skill sets greatly contributed to the course success:

- Early analysis: By requiring students to do a quick prototype of a game involving an old controller, the class generated a number of ideas and introduced ways to innovate.
- Individual pitches: With such a large, interdependent design “space” (games, controllers, interfaces), students needed a way to find “common ground.”
- Guest lectures: Bringing in an electrical engineering professor greatly helped explain a variety of electronics, especially the mouse, shows the detail in what students may consider mundane.
- Group work: Game design and development programs have relied on team building and group dynamics. With an extremely complex design problem, students needed support. Besides splitting work, the mix of students across fields helped to create mentoring.
- Interdependency: Students can resolve an interdependent hardware/software/interface design problem in a variety of approaches. Once they resolved either the core gameplay (“play Banjo Hero”) or interaction (“touch a 3-D surface”), the teams could then more quickly iterate on the remaining aspects.
- Hardware: As originally planned, by providing a hardware experience, students learned about an entirely new aspect of game design.
Students found NES, Atari, Sega Genesis, and other older game controllers relatively easy to dissect and study.

- Cost: Each controller ran the student teams about $60, though the banjo ultimately got a nice strap (and still needs a case).
- Motivation: The students insisted on building an actual controller despite the choice to simulate inputs.
- Feasibility: Time and resources greatly worried everyone. However, both teams successfully created games and controllers within seven weeks due to some students having prior hardware experience.

2. What Went Wrong

However, as with any first run of a course, there were various surprises, synergies, and problems:

Time: The final project should have started from the beginning, especially to focus the design of the game and controller.

- Faculty: More multidisciplinary support would have solidified the topic flow.
- Space: Via multidisciplinary effort, faculty connections might help with lab issues.
- Milestones: Because any aspect of design could become entangled or hang, not all groups had sufficient balancing.
- Bulk/Quantity: Without copies or smaller prototypes, choosing who should own the final controller created problems. Ideally, each student should have something to demonstrate.

3. Future Work

The initial run of the course showed promise and demonstrated that students can indeed make original and interesting alternative game controllers. However, given the frenetic pace and design complexity, future versions require more time, perhaps with one or two weeks of starting the term. A true multidisciplinary effort would greatly broaden the scope, help flesh out missing material, and improve student mentoring. Connecting with researchers in HCI, physical computing, haptics, and a variety of related fields would also improve the educational direction—and even produce students for their research. Such work has already commented with an Augmented Reality Golf project [42].

Another open area involves the availability of peer-reviewed technical schematics and explanations to formalize the numerous websites created by Makers who seem to have adapted their own system of peer review. Although the problem is more about pedagogy instead of theoretical research, educators who synthesize this material would provide a tremendous resource (and oversight) for a multitude of students. Because of the close interaction with a variety of engineering, computation, and design fields, a properly evaluated source would serve many courses.

Subsequent courses can build upon a common framework of controller design, perhaps forming an accepted track within many game development programs. Future efforts could include social networking and augmented reality, tapping into networked and multiplayer applications. Many avenues await innovation in game controllers.

4. ACKNOWLEDGEMENTS

I am deeply grateful for all the hard work, initiative, and motivation of all my alternative game controller students: Dominick D’Aniello, Sela Davis, Ben DeLillo, Mike Ey, Alex Lifschitz, Aury McClain, Tom Merrill, Joe Pietruch, Anthony Reese, Michael Tangolics, and Andy Zickler. Jay O’Leary and Ben Kalb ActionXL were extremely generous in donating many controllers (and other components) for this class and numerous other projects. My former department of Information Technology at RIT provided faculty development funds, which paid for all of the used controllers we disassembled. I am also grateful for three summers’ worth of visiting research faculty fellowships at the Air Force Research Lab in Rome, NY (AFLR/RI) with Dave Ross and Alex Sarmacki—as part of that work, we investigated aspects of citizen science, which I applied to this course work. I also appreciate “OCAI’s” public-domain joystick from [http://www.clker.com/clipart-13249.html]. Finally, I would like to thank the reviewers for providing great suggestions for improving this paper.

5. REFERENCES


Evolving Interdisciplinary Collaborative Groups in a Game Development Course

Holliie Bourdreaux  Jim Etheridge  Ashok Kumar
Univ. of Louisiana at Lafayette  Univ. of Louisiana at Lafayette  Univ. of Louisiana at Lafayette
hollieboudreaux@gmail.com  jne@ull.edu  axk1769@louisiana.edu

ABSTRACT
Computer science departments at many universities are now beginning to offer video gaming courses in response to the growing need for skilled programmers in the video game development industry. This paper describes the evolution of a senior level game programming course over a four year period. The course described is the culmination of a curriculum which offers students a concentration in video game design and development as part of a rigorous computer science undergraduate degree program. In addition to the core computer science coursework, students are required to complete two video game programming courses and have the option of taking two more as electives. To satisfy the remainder of the concentration requirements students can choose from electives in visual arts, theater, and creative writing. The current incarnation of the senior level game programming course puts great emphasis on large interdisciplinary team development projects. The course also incorporates software engineering principles, open source software, and periodic group and individual status reporting. The addition of a final project presentation to an audience of peers, industry professionals, and the media has proven to be an effective incentive for the students.

1. INTRODUCTION
The Computer Science department at the University of Louisiana at Lafayette (ULL) added a new concentration area of video game design and development to its curriculum in 2004. A concentration is a set of classes, consisting of approximately 15% of the total coursework, that focuses on an area of specialization. ULL currently has five concentrations, one of which is video game design and development. The core of the video game design and development concentration is several semesters of game development classes for several different platforms. The course discussed in this paper is the final one in that sequence.

1.1 Curriculum Overview
In addition to university general education requirements, the computer science curriculum consists of several areas of coursework, beginning with introductory and data structures courses then branching off into the following categories:

• Core Computer Science Courses - The courses expected in any accredited computer science curriculum that do not fall into another category below.
• Gaming Relevant Courses - This group consists of courses that are especially relevant to video game design and development, such as Artificial Intelligence, Graphics, Networking, Human Computer Interaction, Software Engineering, and Physics.
• Artificial Intelligence familiarizes students with the background and foundations of AI, intelligent agent-based representation, problem solving and search algorithms, game playing, introduction to LISP, knowledge representation and knowledge-based systems. It also includes an introduction to other subareas such as natural language processing, connectionist models, and evolutionary algorithms.
• Physics prepares students with the classical and relativistic mechanics: heat and mechanical waves. The first level of calculus is a prerequisite for this course. The course helps game design students to understand concepts such as straight line and rotatory motion, force and impact, particle physics, and springs which they
will use in their game designs.

- Gaming Courses - The concentration courses fall under this category. Currently available courses include Introduction to Video Game Design and Development, Video Game Design and Development (the subject of this paper), Programming Handheld Devices, Java for Small Devices, and Game Algorithms and Game Engine Architecture.
- Gaming Electives - Students are required to select from approved courses in gaming related areas including creative writing, Communications, and Visual Arts.

Visual Arts prepares students with an introduction to the use of a computer as a tool for artistic expression. The projects employ scanners, video digitizers, and printers, together with editing software for drawing, painting, image manipulation, and 2D animation.

In conjunction with the senior level game development course students get hands-on experience in motion capture and related simulation in the Computer Science department’s motion capture lab. More information about the concentration can be found at the computer science department’s website [1]. This paper will discuss the senior level game development course, which has been offered every spring semester starting in 2006, for a total of five offerings as of this writing.

2. CLASS ORGANIZATION

2.1 Learning outcomes

The departmental accreditation process dictates that objectives be set for the program as a whole. In order to determine the department’s ability to achieve these goals, each course in the program has a set of learning outcomes that reflect the content of the course and relate to the overall program objectives. Periodic assessment of the learning outcomes for each course provides a means for the department to determine the extent to which it is achieving its goals. The learning outcomes for the senior level video design and development course are as follows:

- Work in a large group to design and develop a non-trivial 3D video game
- Develop and implement a leadership hierarchy within the group
- Analyze and present to the class periodic individual and group status reports on the progress of the project
- Work effectively in conjunction with group members from other disciplines (Visual Arts, Music, Creative Writing, etc.)
- Develop and implement procedures necessary to integrate components of the project produced by disparate software tools (game engines, 3D modeling tools, sound editors, etc.)
- Analyze the strengths and weaknesses of software tools relative to the requirements of the project
- Evaluate honestly and fairly the commitment and productivity of all members of the group
- Define and maintain a development timeline for the project
- Coordinate the creation of a substantial presentation of the status and features of the final product to classmates and other interested individuals

In general, the learning outcomes address the issues involved in the design and implementation of large software systems by relatively large (by academic standards) interdisciplinary teams. The main factor that makes the team approach in this course unique is the fluidity of the project requirements. The development teams determine the game design based on a consensus among team members regarding the type of game they want to create. While the major components of the design are fixed, the details of the design can, and often do, change on a regular basis as a reaction to new ideas, software integration issues, art and audio asset development, and time constraints. The purpose of the learning outcomes is to provide a means of assessing the student’s ability to function in this type of development environment.

2.2 Decisions

Several decisions had to be made with regards to the structure of the course. Table 1 provides a brief description of these decisions and the section in which a more thorough explanation is provided.

<table>
<thead>
<tr>
<th></th>
<th>2006</th>
<th>2007</th>
<th>2008</th>
<th>2009</th>
<th>2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmers</td>
<td>2-3</td>
<td>2</td>
<td>6</td>
<td>9-10</td>
<td>5-6</td>
</tr>
<tr>
<td>Artists</td>
<td>0</td>
<td>0</td>
<td>3 per Group</td>
<td>3 shared</td>
<td>4 shared</td>
</tr>
<tr>
<td>Musicians</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2 shared</td>
<td>1 shared</td>
</tr>
<tr>
<td>Development Time</td>
<td>3-4 weeks</td>
<td>3-4 weeks</td>
<td>16 weeks</td>
<td>16 weeks</td>
<td>16 weeks</td>
</tr>
<tr>
<td>Rendering Engine</td>
<td>custom</td>
<td>custom</td>
<td>Ogre</td>
<td>Ogre</td>
<td>Ogre</td>
</tr>
<tr>
<td>Version control</td>
<td>none</td>
<td>none</td>
<td>SVN</td>
<td>SVN</td>
<td>SVN</td>
</tr>
</tbody>
</table>

Table 1: Overview of Major Decisions
2.3 Larger, Interdisciplinary Groups
As of this writing, the course has been offered five times with several varying parameters. These parameters are outlined in Table 2.

In the 2006 and 2007 offerings of the class, the students were all computer science majors divided into groups of two to three students. Although the class was open to artists, few chose to participate. Those who did felt overwhelmed by the workload and quickly dropped the course. In 2008 more artists were attracted with the promise that their workload would be manageable. Word of mouth led to the recruitment of more artists for 2009 and 2010.

<table>
<thead>
<tr>
<th>Issue</th>
<th>Relevance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group Size</td>
<td>What group size students with the best learning experience?</td>
</tr>
<tr>
<td>Artists</td>
<td>How should artists be allocated to the groups?</td>
</tr>
<tr>
<td>Musicians</td>
<td>How should musicians be allocated to groups?</td>
</tr>
<tr>
<td>Rendering Engine</td>
<td>What rendering engine provides the best combination of availability, power and ease of use?</td>
</tr>
<tr>
<td>Version Control</td>
<td>What is the best way to manage a large code base?</td>
</tr>
<tr>
<td>Demo</td>
<td>How to evaluate the finished project?</td>
</tr>
<tr>
<td>Peer Evaluation</td>
<td>How to evaluate each student’s contribution to the project?</td>
</tr>
<tr>
<td>Motion Capture</td>
<td>How best to integrate current animation technology into the course?</td>
</tr>
<tr>
<td>Project Size</td>
<td>What is the best balance between complexity and manageability?</td>
</tr>
<tr>
<td>Software Tools</td>
<td>How does software selection affect student productivity?</td>
</tr>
</tbody>
</table>

Table 2: Differences in Course Structure over the Five Offerings of the Course

In 2008, there were eighteen students enrolled. Of these, twelve were programmers and six were artists. Rather than break up the class into many smaller groups, it was decided that having only two groups of six programmers and three artists would allow for larger, more complete games to be developed. The groups met twice a week during class meetings and were also required to meet at least once a week outside of class. Using the success of this class as a basis, the 2009 class also used large groups, but pooled the three artists as a shared resource used by both teams of programmers. This was done because there was no way to easily split three artists between the two groups. This class also featured the introduction of musicians. Two musicians chose to participate and were also used as a shared resource. In 2010 there were 11 programmers divided into two groups of five or six, four shared artists and one musician. Creative writers will be integrated in future offerings of the course.

2.3.1 Challenges Faced
Working in interdisciplinary groups provides students with a taste of real world development. In a survey conducted of all students that had taken the course, the computer science students were very satisfied with the abilities of artists and musicians when they were available and appreciated the chance to work with them.

For many, if not all, of the students, this was the first time they had worked on a large project with students from other disciplines, which presented some unique challenges. When the students first started working together, there was a lack of understanding between the members of different disciplines as to the challenges they faced. For example, programmers had no concept of how long it takes to make a model suitable for use in a game. However, to give the programmers a taste of the artist’s side of game development, all of them were required to make at least one model to be used in the game. These models were generally small things like bullets and crates, but it helped aid in understanding the artists’ challenges in the project.

The artists had to learn to make simpler character models with a much lower polygon count than they were accustomed to. In their art classes the focus is on making very highly detailed models but in games efficient use of resources is more important. Highly detailed models consume too many resources and impact the performance of a game. To resolve this issue the artists would make simpler models but use a detailed texture to give it a more visually appealing appearance.

The main problem faced by the musicians in 2009 was that the two teams did not give music requests until later in the semester and were largely ignored before that. In 2010 the teams were encouraged to decide what music will be required early on and
2.4 Software Tools

The course places an emphasis on free and open source tools so students can use these tools after graduation and continue to work on their created games. If the course used proprietary software the students would not have that freedom. The only exception is Maya [2], which the artists are required to use by their curriculum. The programmers are encouraged to use Blender [3] instead, since it is free and they do not require the power of Maya to make simple models.

2.5 Ogre

The first two times the class was taught, students used a graphics engine written by the professor teaching the class. One of the main complaints about this custom engine was the lack of documentation and lack of features. The third offering of the course saw a new professor and therefore a new engine needed to be selected. The Object-oriented Graphics Rendering Engine (Ogre) was chosen [4]. Ogre is a free, open source graphics engine. It is not a game engine, although many people, including those in this class, have used it for that purpose. It is written and maintained by a small core group and contributed to by a growing online community which also maintains a thorough wiki [5].

The power of Ogre is that it can be combined with other libraries, such as sound and physics, to create a customized game engine. Ogre does not include these libraries, but it does provide an interface to include these libraries in an application. In order to promote maximum flexibility, Ogre does not force users to use specific libraries. This use of additional external libraries is good practice for real production situations because graphics or game engines will rarely have all the required functionality built in.

In the classes where Ogre was used, each group took advantage of several of these external libraries. Most of the libraries and tools used worked very well, but some proved to be more difficult to integrate with Ogre. Students took this as a challenge to overcome. In addition, there were issues that could not be easily solved with a convenient library. One student spent many weeks developing a practical solution to fog of war, the obscuring of areas of a game world that have not been explored yet with fog, then posted his findings on the Ogre forums to share with the online community [6].

Many tools for content creation developed by the online community are also available. The Ogre Particle Accelerator is an example of one such tool [7]. It is written using Ogre and allows a user to create custom particle systems and see the results of parameter changes in real time as shown in Figure 1(a). When the particle system is completed, the editor can export a file that can then be loaded into the user’s application. An application that uses a particle system created in the Particle Accelerator is shown in Figure 1(b).

Figure 1: Using the Ogre Particle Accelerator.
(a) Creating a particle system with the Particle Accelerator.

Figure 1(b): A student created application using a particle system created in the Particle Accelerator.

In a survey given to all students that have taken the course, the comments about Ogre were very positive as compared to the custom engine used in previous years. Table 3 lists general student feelings about both engines.
Due to the size of the projects being developed in this course, each team was required to use Subversion and Trac [8, 9]. Subversion (SVN) is a command line open source version control system. Since development was done on Windows the students used TortoiseSVN instead [10], which is a Subversion client that easily integrates into the Windows environment. Trac provides a wiki and ticket system to allow students to keep track of bugs, tasks, and more. It can also act as an interface to an SVN repository and allow browsing of the source code and track changes made over time.

Using SVN also emphasizes the team cooperation aspect of the course and fosters a collaborative environment. When students are not assigned tasks or given any direction on what part of the code to work on, check-in conflicts, where multiple programmers are attempting to change the same part of the code simultaneously, will occur often. Coordinating which student is working on what section of the code base, via the Trac ticket system, reduces these conflicts. SVN and Trac were also used as part of the evaluation process in order to track student involvement in the projects. Since SVN and Trac activity is tied to a student’s school ID, if the student’s ID does not show up in the activity logs the instructor knows it is likely the student is not pulling his or her weight.

2.7 Content Creation
Characters used in the student games were modeled using a variety of modeling programs including Maya, Blender, and Milkshape [11]. Several models from one of the student games are shown in Figure 2. Most models were animated by hand, but a few were animated using the computer science department’s motion capture lab. Plugins for all three programs are available on the Ogre website to enable exporting of the character models in a format readable by Ogre.

The motion capture lab, running Vicon software, features eight cameras. It is an undergraduate lab open to any student with a valid reason for using it. In the 2008 offering of the course the lab was just being set up so only one group took advantage of this resource and only animated two characters with it. In 2009, however, one group animated all of their characters using motion capture. Figure 3 shows a student using the motion capture lab to record his movements on the left and the character from the game using those movements on the right. The other group used insect characters in their game and chose to do the animations by hand.

Maya was used by some groups to create the game environments, but others used a program called Earth Sculptor [12]. Earth Sculptor is a user friendly terrain creation tool that can make terrains suitable for inclusion in games. It is available in two versions: free with limited features and a full version for a small fee. For the purposes of this class, the free version was sufficient. After sculpting the terrain, the user can paint the desired textures onto it before exporting the terrain as a set of files that can be imported into Ogre. Figure 4 shows a terrain being created in Earth Sculptor (a) and imported into an Ogre application (b). For groups using Maya, the Ogre website provides an plugin for most versions of Maya to export the created terrain in the proper format for Ogre. Neither Earth Sculptor nor Maya is better than the other. Earth Sculptor has a shallower learning curve and works well when creating simpler environments. Maya has a very steep learning curve but is able to create more complex environments that Earth Sculptor cannot.

Table 3: Summary of Student Comments on Custom Engine and Ogre

<table>
<thead>
<tr>
<th>Feature</th>
<th>Custom Engine</th>
<th>Ogre</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation</td>
<td>Almost none</td>
<td>Extensive Wiki and full API reference</td>
</tr>
<tr>
<td>Ease of Use</td>
<td>Very difficult to use</td>
<td>Tutorials on the Wiki make it easy to become familiar with the code</td>
</tr>
<tr>
<td>Understandability</td>
<td>Lack of documentation makes it hard to follow the code</td>
<td>Generally easy to follow after completing tutorials</td>
</tr>
<tr>
<td>Adding/Custon Functionality</td>
<td>Very difficult</td>
<td>External libraries make adding physics, sound, user interfaces, etc, simple in most cases</td>
</tr>
<tr>
<td>Support</td>
<td>Original creator</td>
<td>Extensive online community forums</td>
</tr>
</tbody>
</table>
Sound effects and music were created using Adobe Audition [13] and Audacity [14]. Some teams recorded custom sounds for their game. The server in the game lab contains over 40GB of royalty free content that includes audio files that can be used to create music and sound effects, in addition to textures and models. All content created for the course is saved on the game lab server and is made available to all students in the video game concentration.

2.8 Development Time

In the 2006 and 2007 offerings of the course, the students were only given a few weeks at the end of the semester to work on their games. The early part of the term was devoted to lectures and demonstrations of various aspects of video game creation. As a result, the games produced were very small. Starting in 2008 the groups were formed by the end of the first week of classes. The groups began planning and brainstorming right away, with implementation beginning by the third or fourth week. In place of regular lectures students took advantage of Ogre’s extensive and detailed tutorials to get up to speed on Ogre and become familiar with game development topics. Special lectures on selected topics are given as interest warrants and time permits. Weekly show-and-tell updates are required, in addition to examination of SVN and Trac logs, to ensure that everyone is making progress. As a result, the groups are able to get a great deal accomplished. Sixteen weeks is only a fraction of the time spent on development in the game industry, where games require years and hundreds of people to complete, so it is understandable that none of the teams have been able to fully implement their game design. However, every team has made significant progress in that short time. All the games have been playable even if some features are missing or incomplete.

In the survey, students were asked to give a percentage representing how much of their original plan was implemented at the end of the semester. From 2006-2008 students gave responses generally ranging from 50-75%. In the 2009 class, where students were encouraged to start with a small plan, the responses were higher, ranging from 60-90%. The plans devised by the 2009 teams started with a minimum core game design and included a wish list of other features that would be added if time permitted. As a result, the teams were able to focus on the most important aspects of their respective games instead of incomplete features that would end up being removed from the game due to time constraints. The 2010 groups followed this same
design structure, with 72% believing that they completed 70% or more of their original plan.

2.9 Internal Group Dynamics
As the size of a group grows, proper management of that group becomes even more important. Below are several measures taken to assist the students with managing their groups.

2.9.1 Team Leader
Every team was required to decide on a team leader. This person was responsible for assigning tasks to team members, dealing with conflicts, and making sure implementation was going according to schedule. In the survey, 85% of respondents felt that their team leader was an effective or very effective leader, with the remaining 15% stating their leader was only somewhat effective or not effective. Many of the poor responses were from students working on the game Carny Roll, which was plagued with leadership issues (see section 4.1).

2.9.2 Task Assignment
Before 2010, where the teams were required to use Trac for task management, the teams did not keep detailed records of who was assigned to a task. As a result, some students were able to slip through the cracks and do very little work. In an effort to prevent this from occurring again, in 2010 Trac tickets were used to assign tasks and the instructor was able to quickly identify students that were falling behind in their duties.

2.9.3 Trac Leader
Starting in 2010, the Trac leader was appointed by the team leader to handle the creation and assigning of tickets, as well as the managing of the weekly milestones. This allowed the instructor to easily see which students were completing their assigned tasks on time.

2.9.4 Artist/Musician Liaison
One student per group was appointed to be the liaison with the artists and musicians, if present. This student was responsible for communicating what art or music assets are required for their group’s game and ensuring that the artists or musicians are creating the assets in a timely manner.

3. Student Games
This section will give a brief overview of the games developed by the student groups in the last two iterations of the course.

3.1 Carny Roll (2008)
One team of the 2008 class decided to make a first-person shooter (FPS) game. In an FPS, the user sees from the perspective of a character and wields some sort of weapon such as a knife, sword, or gun. In Carny Roll, the user’s character is trapped in a dream of an evil carnival with other characters. Defeating the characters in the area earns tickets which are used to escape when enough have been collected. Figure 5 shows a screenshot of Carny Roll being played.

In a departure from FPS tradition, defeated characters do not collapse to the floor in a pool of blood. Instead, they explode into rubber ducks and candy. The students did this because they wanted to try an alternative from the traditional rag doll physics and bloody violence commonly seen in this genre of game. Carny Roll also features networked play with support for multiple servers. The use of the physics library NxOgre [15] allowed environment features such as barrels and crates to behave realistically when disturbed. Raknet, a cross-platform C++ game networking engine [16] was used to allow multiple players to compete over a network.

Figure 5: Screenshot of Carny Roll gameplay

Unfortunately, leadership issues plagued this team. The team mutinied against two leaders due to their ineffectiveness and poor organization before finally settling on a third team leader. The constant changing of the command structure and lack of organization caused the overall quality of the game to suffer. Nearly all students on this team rated the leaders as ineffective in the survey.

3.2 Internal Conflict (2008)
The other 2008 team elected to make an educational game based on the real time strategy (RTS) genre. The basic premise of an RTS is to build up an army to defend against and defeat one or more enemy armies.
While most RTS games have the player commanding an army of humanoid characters, in Internal Conflict, the user directs an army of beings based on cells in the human immune system. Using these cells, the player must defeat various diseases infecting the body in which the game takes place. Figure 6 shows a screenshot of Internal Conflict.

The characters are, for the most part, not based upon how they look in the human body, but the functionality of the cells is very similar. Because of this, Internal Conflict can potentially be used to teach people about the human immune system. The game features cel-shaded graphics, a style that makes objects rendered on a computer appear to be hand drawn. This allowed for many simpler models to be created. The game also includes voice acting for the immune system characters, fog of war to obscure unexplored areas of the map, and a simple artificial intelligence controlling the enemy army. All units use a path-finding algorithm to navigate around obstacles.

Also included with Internal Conflict is a stand-alone level editor. It was developed alongside the main game during the first half of the semester and then used to create the levels used in the game. After selecting a terrain and placing the initial setup of all the characters and buildings, the user can choose to export the level as a file that can be loaded by the main game. The level editor uses CEGUI [17] for its user interface. CEGUI uses XML scripts to describe the user interface layout and objects. The functionality of these objects is determined by the application code.

This team included several programmers who also had experience with modeling and drawing. As a result, some of the characters and art were created by these very enthusiastic programmers instead of the dedicated artists. Maya was used to create the game environment and some of the characters, with Milkshape used for the rest. CEGUI was used for the user interface with a custom style created by one of the programmers with artistic talent.

3.4 Steam Bout (2009)

Unlike the other groups, this team did not come up with the graphical style first and then tell the artists what was required. Instead, they used the idea of “psychographic profiles” of players developed by Wizards of the Coast [18], which uses the personality and behavioral traits of a person to determine their motivation for playing. In this way, developers can design content to satisfy each type of player. The artists were asked to design the characters to conform to those profiles using the art style of their choice. The resulting art style was steampunk, a blending of a setting where steam power was still used, such as the Victorian era, with elements from science fiction or fantasy. The game itself is a networked third person fighting game. The networking was done via Raknet. Each character has a certain amount of “energy” with which to perform their fighting moves. A gameplay screenshot is shown in Figure 8.
3.5 Path of Cleansing: the Dark Horde (2010)

Path of Cleansing: The Dark Horde is a real time role playing game set in a medieval fantasy world. The game starts with the protagonist helping her mother tend to their farm. At some point, a giant meteor crashes into the farm, destroying the family farm house and unleashing a terrible plague upon the kingdom. People and animals become infected with the plague and turn into zombies. The player is tasked with finding the source of the plague and discovering a way to stop it. The game features a very detailed underlying battle system, similar to the Advanced Dungeons and Dragons (AD&D) table-top role playing adventure gaming system. Everything from player attributes such as intelligence, strength, charisma, constitution, and dexterity, are all meticulously factored into the damage calculations. A screenshot of the game is shown in Figure 9.

3.6 N-Stal (2010)

The team creating N-STAL decided to make a game that was styled and inspired by games that featured more exploration and less outright combat or enemy encounters. They wanted to focus on one or two large encounters that would be fantastic, rather than several enemies that would be less interesting or exciting to fight, feeling that the player would be more impressed or challenged by something that was physically imposing. Their main character was based not on a soldier, as is common in this genre, but on an explorer. Taking a hint from Steam Bout, the team created a psychological profile of their main character and passed that to the artists, who styled his appearance off of the class professor in a form of homage. The other character that was given the most focus by the artists was the boss, a large, quadruped robot, shown in Figure 10.

4. POST DEVELOPMENT/PROJECT ASSESSMENT

4.1 Performance Evaluation

At the end of the semester, teams are required to present their games in a seminar open to faculty, peers, industry professionals, and the media. Since it would not make sense for computer scientists to judge artwork or music, assessment of those parts of the game was done by faculty members from the appropriate departments where possible. A student’s grade for the course was based on a combination of factors:

4.1.1 Critiques of the Game by Peers

Each student is required to play the other teams’s game and evaluate it on several criteria, as well as
give an overall impression:

• Originality of concept
• User Interface
• Artwork and animation
• Sound
• Game Design
• Balance Other students that have previously taken the course also volunteer to play and critique the games.

4.1.2 Status Updates

Students are required to show proof of their progress in order to verify progress is being made towards completing their assigned tasks instead of procrastinating until the end of the semester. SVN and Trac logs are mainly used as proof of progress, although a student may sometimes be working on a task offline and will be required to provide an appropriate proof.

4.1.3 Individual Cooperativeness and Productivity

Real world development is rarely done solo. Students are encouraged to collaborate with their team members to complete tasks. At the end of the semester all members of a team critique their teammates in regards to their cooperativeness and productivity.

4.1.4 Instructor’s Assessment of the Final Product

This includes evaluation of several factors:

• Code review and demo
• Gameplay
• Features implemented
• User interface
• Percentages of original design implemented
• Peer evaluations

4.1.5 Performance at the Seminar

Each team member is required to participate in the presentation. Students are graded based on their enthusiasm, preparation, and presentation.

4.1.6 Review by Professional Developers

Professional game developers attend the seminar and offer their input as to the quality of the final product, while taking the short development time into account.

4.2 Evolving Documentation and Resources

All artwork and music created for the games is collected at the end of the semester and saved on the game lab server. These assets are made available to future groups of students that take any of the game concentration classes. In addition, students that find and use a new Ogre plugin library are required to write a handout, in a standardized format, and provide example source code, if applicable, about the library to assist students that may use it in the future. The document covers how to integrate the plugin into Ogre, any pitfalls to watch out for, an example of how to use the library, and other helpful tips. In this way, students can use their experience to teach future students and reduce development time in upcoming semesters.

5. RELATED WORK

Bayliss and Bierre [19] describe a newly designed and in-place curriculum named Game Design and Development (GD&D) at Rochester Institute of Technology, whose entire focus is on game design and development. The curriculum does overlap with the traditional Computer Science (CS) and Information Technology (IT) curriculum; however, the common courses deal mainly with advanced programming. The idea in the GD&D curriculum is to appeal to a broader range of students, with possibly more female students than in traditional CS and IT curriculums. It is reported that a smaller percentage of incoming GD&D students are familiar with basic programming than in the CS and IT curriculums, when they enter. However, the GD&D curriculum provides 3 innovative initial programming courses for GD&D students via usage of Wii-controlled games and a supplied code base. It has been observed that GD&D students go on to take the regular course in C++ programming and do well in terms of performance as well as retention. Furthermore, upon completing the curriculum, the majority of GD&D students wished to become software developers in gaming industry (44%) or game designers (25%). In other words, the curriculum inspired most GD&D students to become software developers and designers.

The work by Bidarra, et al. in [20] describes a particular course on game projects, taught in the second year, at Delfts University of Technology with a focus on game development in large teams consisting of students from different disciplines. The proposed courses learning outcomes include demonstrating proficiency in the following:

1. Applying media and programming techniques within the context of computer games, and in relating them to particular game effects;
2. Striving for the balance between the effectiveness of a programming technique and the desired quality of a game effect;
3. Describing the main modules of a game engine.
and purposefully using their functionality;
4. Deepening object-oriented programming skills while building a complex and large software system; and
5. Developing and contrasting teamwork skills within the context of a realistic interdisciplinary team.

The deliverables for the proposed project-based course include working source code, game design document, and technical document. The survey results, reported in the paper, indicate that the students were highly motivated upon completing the course and were largely happy with the projects.

Brown, Lee, and Alejandre [21] report introducing a game design project in the department of computer science at Drexel University with an emphasis on developing team development skills as well as other soft-skills in students. The goals of the course were to:

- Provide students with an opportunity to develop math games for middle school students that can be immediately available for use;
- Enable students to receive and respond to feedback from multiple sources;
- Teach educational game design and development concepts to Drexel University students by providing them with a real world application for the knowledge gained,
- Provide students with an opportunity to express their ideas using multiple media such as online-only discussion board exchanges, storyboards and class presentations, and
- Provide students with an opportunity to work in teams of students from technical and non-technical disciplines.

A team consisted of students from computer science, education, and digital media. The end users for the product (game) were K-12 students and teachers. The course entailed not only planning, design, and development, but also demonstrating the product to the end user and taking their feedback into account for further improving the product.

Kessler, van Langeveld, and Altizer [22] describe a new interdisciplinary program at the University of Utah, entitled Entertainment Arts and Engineering (EAE), which brings together students from the School of Computing and the Division of Film Studies in an effort to teach both video game development and computer animation. The key characteristic of the program is the shared classes where students from both Computer Science and Fine Arts study together and cooperate on game and animation projects. The program is highlighted by a yearlong capstone course in which the students work together to make a video game or animated short from scratch. The interdisciplinary program is reported to attract students and better equip them for careers in video games and animation.

The report from a forum in [23] states that gaming is an interdisciplinary field with a wide range of topics encompassed by the acts of game creation, analysis and criticism. It states that game development students must be exposed to analytical, practical and contextual materials, and it points to the International Game Developers Association (IGDAs) curriculum framework.

One common factor in most of these programs is the focus on getting the task done (i.e., a game designed and developed) in mid to large size teams, mostly interdisciplinary. This is a significant departure from the traditional computer science curriculum, which was mostly devoid of such experience.

6. CONCLUSIONS AND FUTURE WORK

Developing a game curriculum is a challenging task that requires cooperation and expertise from several diverse areas of knowledge, as articulated in [24]. Further, there could be many ways of designing a curriculum depending on the emphasis of the curriculum. The video game design and development curriculum was established as a concentration within the framework of an existing, fifty year old nationally accredited program that awards the Bachelor of Science degree in Computer Science. It meets or exceeds the minimum guidelines set forth in [24] for starting a game curriculum.

A criticism of many college courses is that they do not represent a “real world” experience. Within the constraints of time and personnel, the current format of the course described here attempts to provide students with as realistic a game development environment as possible. Under the current format, 90% of students felt that the course was either “useful” or “very useful” in preparing them for a career in video game development. The professors have noticed that students taking this course have gained broader knowledge that can be taken to other courses and the workforce. This includes but is not limited to:

Evolving Interdisciplinary Collaborative Groups in a Game Development Course
• Code efficiency – In the offerings of the course where artists were not present, the graphics used in the game were very simplistic and did not tax the hardware. The assets provided by the artists in later offerings were complicated enough to cause slowdown in the games unless the students optimized their code.

• Modular design – Due to interfacing with Ogre’s prebuilt libraries, the students had to use a more modular design and work around what was already written. This also gave them experience working with other people’s code that will be useful for all future development projects.

• Interdisciplinary cooperation – The students now have experience working with people from other disciplines. This will be beneficial when they graduate and work on real development projects.

• Self sufficiency – Students quickly discovered that for many tasks, such as physics or networking, one or more Ogre libraries existed. However, there were also many features planned for the games where no such library existed and the students wrote their own application or code. One of the programmers for Internal Conflict created a custom level editor in order to design the levels for the game with a convenient click and drag interface. On the same game, another student created a custom implementation of fog of war.

As of this writing, the senior level game development class has been taught five times, three times in its current version. Student response to the last three iterations has been overwhelmingly positive. Many students from the first two semesters of the course have expressed a desire to take the class again. The survey asked students how satisfied they were with the course overall, and the results indicated much higher satisfaction levels for all aspects of the course under the new format. Students that had taken the course under both formats were very pleased with the new format and felt it provided for a much better learning experience.

In the future the class will be expanded to include students from performing arts and creative writing. Also, the department will begin encouraging students to consider more educational themed games like Internal Conflict in all the gaming courses.

7. ACKNOWLEDGMENTS
The authors would like to thank the anonymous reviewers for their constructive comments.

8. REFERENCES


Evolving Interdisciplinary Collaborative Groups in a Game Development Course


Designing an International Curriculum Guideline for Game Audio: Problems and Solutions

ABSTRACT

In this paper, we present an overview of some of the issues and questions encountered in developing an international game audio curriculum, and outline some of our solutions. In doing so, we discuss the interdisciplinary needs of video game instruction, the industry’s desire for key soft skills in addition to technical skills (based on our informal and formal querying), and the constraints faced in terms of institutional and international differences in curricular structure. While our curriculum is specifically concerned with game audio, our solutions and approach can be adapted across other disciplines facing similar difficulties.

1. INTRODUCTION: WHY A CURRICULUM GUIDELINE FOR GAME AUDIO?

A common question faced by both industry and academic practitioners and theorists of games is ‘how can I get a job in the video game industry?’ While video games have, in the past decade, been making in-roads into training programs in both private and public sector post-secondary education, there are still significant gaps in the overall curriculum, especially when it comes to audio. Some degrees in video game design or development may offer a few hours of instruction on audio, and some degrees in sound design or composition may over a few hours of instruction on games, but there are very few options for students who want a dedicated knowledge of game audio that will meet the needs of the game industry. Brian Schmidt, an industry veteran who has recently created a workshop dedicated to helping train game audio specialists, describes, ‘Although many schools teach music composition and sound design, few if any teach the additional skills needed to create music and sounds for interactive games. . . There are numerous issues that traditional composers or sound designers ignore when faced with working on their first game. For example, what is a ‘parameterized’ sound effect, how do I create interactive music? What is XMA compression? Why can’t I just use Pro Tools to create game audio? These issues are second nature to top game audio professionals, but they overlooked in traditional music or sound design courses. In fact, the need for better game audio education was one of the reasons for the formation of the Game Audio Network Guild, a non-profit group aimed at organizing and educating creators and audience, and remains an ongoing mandate of the Interactive Audio Special Interest Group, an organization primarily aimed at developing tools and standards for the audio industry.

Although there are conferences and workshops with ‘boot camp’ style introductions to game audio, the need for a more comprehensive education is certainly seen as critical by those active in the video game industry. Working in conjunction with the Interactive Audio Special Interest Group (IAsig) and liaising with members of the Audio Engineering Society (AES) and the Game Audio Network Guild (GANG), several academics and industry partners have been collaborating to create an international Curriculum Guideline for teaching game audio at the post-secondary level. The Guideline began from this basic industry need for better trained employees in game audio (music, sound design, dialogue, audio implementation, production, direction and audio programming). The IAsig felt that a Guideline
coming from a conglomeration of academic and industry personnel was the best way to ensure that an appropriate curriculum is created that meets the needs of industry as well as the rigors of post-secondary education, and therefore was the ideal organization to spear-head the development. The aim of the Guideline is to provide information, assistance and suggested teaching topics to institutions wishing to implement more game audio into their curriculum, whether as an entirely new program, or just as one or more additional courses added to an existing program. While the focus is on audio for video games – at this time the largest market for interactive audio – it is assumed that interactive audio for other media (educational products, interface sounds, etc.) will be a growing area, and skills are aimed at students learning the most important aspects of interactive audio that can be applied to a number of industries. The experience of designing an international Curriculum Guideline raised interesting questions and problems in regards to how to develop a curriculum that can be implemented in a variety of different educational structures, while at the same time ensuring that the guidelines will be adopted, as well as meet the needs of both industry and education in vocational-based and more traditional post-secondary institutions. In this paper, we describe some of the difficulties that we have encountered, and illustrate our approaches to solving these difficulties, with the aim of providing readers the opportunity to adopt ideas as they seek to develop their own programs or curricula. We divide the paper into three significant areas of difficulty, with each section describing solutions, although there is considerable overlap in both the difficulties and our solutions in each section. Finally, we summarize what we have learned from the process, and outline areas for future development.

1.1. A Note on Terminology

Although there was involvement from people in several countries and areas of the world (with the majority of representation from North America, Australia and Europe), the largest proportion of the IAsig Educational Workgroup was U.S. American, and the USA remains one of the major centers in video game development, and so we have chosen to use American terminology with regards to the curriculum. A semester refers to a 16-week block of classes (what in many Commonwealth countries is referred to as a ‘term’). Typically about 4 weeks of the semester is reserved for exams. It is assumed that a student will take four or five courses per semester. We define a course as a one-semester block of classes that works towards building a qualification (in the UK, this is referred to as a ‘module’). We assumed that courses would meet for at least three hours in-class per week. A degree program (major) is the collection of courses that makes up the complete qualifications, and a stream is a specialization within that program. In fact, this confusion over terminology was one of the first and easiest problems to overcome. Discussing typical in-class time and program structures amongst our countries and systems and agreeing on terminology to describe the structure was one of our very first steps.

2. THE INTERDISCIPLINARY NATURE OF GAME AUDIO

Game audio is highly interdisciplinary in nature. We define interdisciplinarity as any work involving more than one discipline, and the integration of these disciplines into a distinct, new area of inquiry. [25] The interdisciplinarity of game audio is both a result of the structure of the game industry, and its needs, as well as institutional divides that tend to segregate the Arts and Humanities from the Sciences.

2.1. Jack-of-All-Trades Roles Within the Industry

Game audio practitioners often find themselves having to wear many hats. While in big-budget game productions and in large production companies the job descriptions for employees are often clear, in mid and low-budget productions and in smaller companies the border-lines between work descriptions of composers, sound designers, audio engineers, programmers and the like often tend to blur. Today, a large section of the industry (not only those that develop games but also the others that provide applications and content for websites, customer service systems, mobile phones and such) still has the ‘one-person’ audio department and that one person must work across all audio disciplines, in other words s/he has to be a ‘jack-of-all-trades’ that composes, manages sound design and implementation. This blurring of the boundaries of job descriptions is also true (though to a lesser extent) for video, television and the film industries as well. However, the implementation of sound into a game is a far more complicated process than is usual in the film or television market. It is often impossible, in other words, to implement audio into a game without some form of programming knowledge, and even composition must often be thought of in terms of programming-like logic.

The blurring of boundaries described above calls
for professionals who are not only competent in one area (e.g., expressly in music composing, or in audio recording), but in other areas of audio as well. For example, a composer working on a low or mid-budget game, besides composing, recording and mixing music, usually ends up recording and editing sound effects, designing sound, and even recording and editing dialog and voice-overs. Moreover, the skills required are the same as those for traditional linear media (e.g. film, video), with an additional skill-set related to the requirements of the interactive, non-linear, programming-based media (responding to in-game parameters, creating dynamic, interactive compositions, and so on). Any curriculum for game audio should include these additional needs of interactive media while at the same time providing enough of a grounding in traditional, linear, non-interactive methods that the student will be competent entering either field.

2.2. The Need for an Institutionally Interdisciplinary Approach

Not only must the students be able to demonstrate broad skills in both creation and implementation, and cover skills from more than one form of audiovisual media, in most schools game audio creators fall into a disciplinary valley surrounded by the Arts and Humanities (music and sound programs) on one side, and Computer Science (programming) on another. Game audio undoubtedly encompasses music and sound or recording departments, but also requires an understanding of programming in order to understand implementation or audio engine programming (the domain of Computer Science departments), acoustics (often falling under Physics or Engineering departments), real-time sound generation (requiring a foundation in Mathematics and Physics), voice talent directing and managing (the domain of Drama and Communication departments), and a general foundation in game/film/media studies to provide theoretical, analytical and critical contexts (which can be in a variety of Media Studies, Fine Arts, Film Studies, English departments, and so on). It was necessary to develop a curriculum that could draw from this interdisciplinary and cross faculty nature and ensure that, for example, even students with a concentration in music composition would have an understanding of the work that is undertaken to turn their compositions into actual music productions into a game, in order to foster communication between disciplines.

Moreover, having a basic understanding of these areas at an early stage in their learning cycle would allow students to explore possibilities beyond their immediate perceived interests into other areas. The integration of audio into a game requires many considerations and choices that should be made between the composer(s), sound designer(s), implementation person(s) as well as audio programmer(s). Decisions regarding real-time mixing for instance, and the various needs for audio compression as well as other technical limitations often need to be made as a group to prioritize needs at each point in the game. Clearly, then, understanding the tasks of others in the group and the technical needs of the game are a vital part of the job.

2.3. Additional Career-Based Requirements

Besides these specific interdisciplinary skill needs, considering that many of the available jobs are freelance contracts for both music and sound design, students also require a demonstrated understanding of marketing and business, especially the video game and music industry, budgeting, and copyright law. When working as freelance, contract (which can be per game or per a specific time period), or on an in-house basis, apart from their particular audio specialty, students also need to wear other hats such salesperson, accountant, self-promotion and marketing expert, and the like. Trying to squash too many needed skills into a program is a common problem faced by many disciplines, of course, and so the difficulties we faced here were not necessarily new, but were compounded by the institutional divisions that would often mean that these skills are taught in a variety of faculties. In addition to our solutions described below, we strongly suggest that students work, where possible, in some form of internship as part of their program. These internships, of course, not only help the student to develop career skills and networking, but also help to bring fresh ideas and the latest trends back into the classroom.

2.4. Our Solution: A Prioritized Common First Year

Taking into consideration the specific skill needs of game audio professionals, we set about prioritizing the skills that were most important to each of the typical jobs available. This involved several stages. First, we collected a range of job advertisements that listed skill requirements and collated the information. We then surveyed members of the game audio community to come up with lists of skills that they undertake in their own jobs. Together these lists formed a set of skills that became part of a shared Google Document. Members of the IAsg were invited to rank skills on an online survey according to their perceived value (critical skills, important skills, desirable skills, and unimportant skills) for each of the main streams of our focus (composers, sound designers, implementers and programmers). This ranked list became the foundation
of a series of skill blocks that focused around related skill-sets. Due to the over-burdened nature of the curriculum, we eliminated those skills that were deemed unimportant, and in some cases, desirable skills that were not deemed critical or important.

For example, our skill-block of ‘Microphone and Recording Skills’ included the objectives to be able to direct a basic recording session using microphones, mixers and editing systems; to understand microphone choice and placement for various types of studio recording; to select, use, and appropriately place microphones in field recording; to understand static and live/motion capture dialogue recording; and to conduct an ADR (automatic dialogue replacement) recording session with an awareness of recording facial movements for post recording animation lip-sync.

A more theoretical selection included a ‘Game Music Analysis’ block. These skills included understanding the aesthetics and practices of interactive music; understanding how different musical styles can be used in game and media production; the ability to critique contemporary game scores; understand the evolution of game music and its relationship to technological advances; and to understand the parallels between non-linearity in concert music and game music. The skills for recording were rated as high priority in each of the streams, whereas the music analysis skills were rated as lesser importance for sound designers and programmers than for composers.

From this list of skill blocks, then, a set of core blocks that were deemed critical or important to everyone were highlighted and were collected as belonging to a ‘core’ series of classes. Our final core set of skill blocks included an understanding of the game industry and various project and time management skills, understanding games and game design, studio and session management, recording skills, editing and mixing skills, acoustics skills, implementation skills, synthesis and sampling skills, programming skills and analysis skills. In other words, students would all begin with a broad base of common courses that they shared in order to foster teamwork and understanding on upper-year projects. Students in each of the streams would take courses that offered these skills in a common first year. That way, students may also adjust their expectations and have a greater understanding of related but different potential areas of study. Taking this list of core blocks and more stream specific skill blocks, we then attempted to anticipate how many hours were required to learn each of the skills to a degree appropriate for a graduate. Based on these suggested hours, we created larger sections of core blocks into courses of about 30 hours instruction time. In this way, we could see how many courses would be required for each stream and if the plan was feasible.

We then created an example ‘blue sky’ curriculum based around these courses for a four-year US American degree program as a suggested example of how the program could be implemented. We called this a ‘blue sky’ curriculum because this would be an ideal implementation of the curriculum, integrating all of the skills deemed important for a professional. We also recognized, however, that due to many factors (see below), the implementation of such a curriculum for many institutions would be impractical, and so set about creating alternative ways to implement the highest priority skills into courses.

3. SKILLS-SPECIFIC LEARNING: SOFT SKILLS, OBSOLESCENCE AND AUTOIDACTICISM

One of the key findings of our discussions and surveying of industry professionals was the strong need for soft skills, particularly adaptability and the ability to independently upgrade skills. We have separated these needs here, although clearly there is some overlap.

3.1. The Importance of Soft Skills

Many of those involved in the industry stated that what they thought was more important than training technical skills was soft skills or emotional intelligence, and what might be referred to as self-management skills.[24] According to Paul Lipson, president the Game Audio Network Guild, candidates wishing to enter the field of game audio should be ‘flexible, personal, and professional’ and remember that ‘You work with and for a team.’ [1]. In the development of the curriculum within such a technologically driven discipline such as game audio the focus has inevitably been on technical skills and knowledge. However there is significant evidence that success in finding and keeping work in any industry (particularly technologically-based and keeping work in any industry (particularly technologically-based industries) is not simply dependent upon technical ability but also on a student’s so called ‘emotional intelligence’ [8] ‘soft’, or ‘interpersonal’, skills [10]. These are sometimes assumed to be a part of the ‘hidden curriculum’,
defined by Dutton as ‘those unstated values, attitudes, and norms which stem tacitly from the social relations of the school and classroom as well as the content of the course’ [6]. The annual report in 2008 from Project Bar-B-Q 2008, an interactive audio think-tank conference, rated interpersonal skills as being very important for every position [18]. Personal and interpersonal skills, referred to in the UK as ‘Employability’ skills [15], also feature strongly in our analysis of 40 recruitment advertisements for jobs in games audio. We found that interpersonal skills could be grouped into three main areas:

1. Interpersonal skills such as the ability to develop and sustain productive working relationships: ‘good interpersonal skills’; ‘able to support the development of relationships’; ‘Build lasting relationships’.

2. Teamwork skills: ‘Great teamwork skills’; ‘Be a Team player who is able to share his/her ideas with others’; ‘ability to work collaboratively with the other members of a team’; ‘Demonstrate the ability to work in a team environment’; ‘able to mentor and support team members’; ‘Is able to work well under guidance and direction’; ‘Knowing how to work under pressure, for long hours, as part of a close-knit team.’

3. Communication skills: ‘Great communication skills’; ‘excellent communicator (spoken and written)’; ‘pro-active communicator’; ‘able to share ideas and import new ideas and/or processes’. We also found a recurring theme in terms of the personal attributes of the students: ‘Proactive, self-motivated person who can multitask effectively’, ‘working without direct supervision’, ‘the ability to both take direction well and make strong, self-directed decisions’, ‘Demonstrated ability to be proactive and self-motivated’, ‘Well organized, rigorous and autonomous’.

Questions arose as to how to ensure that these necessary soft skills became incorporated into the IAAsig’s Curriculum Guideline, to guarantee that students would not only develop these skills, but be able to demonstrate these skills to potential employers through their portfolios, through internships, and in job interviews. How to teach these skills, therefore, became a question related to the wider aims of education in general and the structuring of course content in higher education.

3.2. Changing Industry and Skill Obsolescence

Useful to our conception of a curriculum guideline was a summary by Hutchings and Saunders, who state ‘In the highly competitive environment all higher education institutions face, the ability to conceive, design, market, deliver and reengineer curricula that meet the diverse and rapidly changing vocational, disciplinary and artistic aspirations and expectations of their clients, is critically important’ [12]. In other words, skill obsolescence can go beyond merely the tools of the trade [22], and it is necessary that an allowance for aesthetic, industrial and technological obsolescence is built into a curriculum. The nature of the game industry requires an ongoing, evolving approach to course design that can rapidly adapt to changing technology and tools. For universities, the bureaucratic structure of approving changes to courses can be difficult and very time consuming (often up to as much as two years). Therefore, the nature of course descriptions as well as the proposed content had to be vague enough to allow for what may be significant changes to occur.

Moreover, the epistemological divide that exists between traditional university and more vocational centric learning (what might be called the difference between education and training) means that those coming from the traditional university background and its emphasis on broadness and soft skills may be less qualified in terms of hard skills and specialization immediately required by the industry, but potentially better able to adapt to a changing industry as well as changing job positions later. There was a constant consideration, therefore, to find a balance between necessary training in hard technical skills as well as rounding out these skills with theoretical understanding and critical thinking [26].

3.3. Autonomous Learning

The need to develop autonomous learning is clear not only from our research into the job market discussed above but also from the fast moving nature of the game industry itself where the market is driven by technological development and innovation [21]. Within the proposed curriculum the emphasis is on teaching principles, not packages, as by the end of a four year program it is very likely that current software and even the fundamental technologies of the game consoles they are built on will have entered a new cycle. Professional schools and private colleges may be better able to adapt and provide specific programs, but as discussed, the students do not merely require technical skill-sets, but also the ability to adapt quickly, to contextualize their work,
and to adjust their skills to changing aesthetics. The pedagogical model requires what Derry and Murphy define as an ‘embedded curriculum’, as opposed to a ‘detached curriculum’. They describe that a detached curriculum provides isolated instruction in a particular skill-set, like mathematics or music, whereas an embedded curriculum teaches students about learning-to-learn within their discipline. They describe, ‘If students are, right from the beginning, instructed not only about the subject but also about how to learn it, then they will, in the long run, acquire a considerable number of specific learning techniques. Equipping the students with a reservoir of subject-related efficient learning techniques is a main step in enabling them to acquire new knowledge independently.’[13] Our courses, therefore, were organized around the concepts and principles behind the design and development of audio for games, rather than around any hardware or software specific skills [14], and would strongly encourage a constructivist approach [3] that may prove more effective in producing the kind of auto-didactic who can continue to refresh their skills on their own. As Mike Rawson describes, ‘learning to learn’ has become a critical part of the skills agenda, and is an essential ‘package of skills’ to ensure future employability [20].

3.4. Our Solution: Problem Based Learning With an Experiential Focus

Criticism of the concept of learning outcomes has come with fears that complex learning cannot take place, and that attention would be focused on those things that can be described in terms of objectives and outcomes [16]. We recognized, therefore, that there had to be opportunities in the curriculum for exploration, for in-depth study and personal growth, and that the curriculum ‘should not be so crowded that “surface” learning is encouraged at the expense of understanding’[16]. Describes Rawson, the learner needs to be ‘involved in a self-reflexive process of learning: a conscious examination of his or her learning processes’ [20]. Analysis, criticism, and self-reflexive thinking, journal and essay writing and discussion, along with self-directed but guided learning are seen as essential in fostering this learning to learn. Peter T. Knight describes that time for strategic thinking, portfolio-making, along with ‘mindfulness’ and reflection on practice needs to be written into a program [16].

One way of promoting the development of such skills is simply to make students aware of the importance placed on these skills by potential employers such as the results of our job analysis survey. Although highlighted in the curriculum as specific learning outcomes, we feel that modules viewed as ‘generic’ are traditionally undervalued by students and that these should be embedded into the context of the students’ projects. Skills development most effectively takes place through a process of ‘active participation, feedback and reflection’ [19] and so we would recommend that these are developed within an experiential learning cycle [17] that involves simulation and role play within a team project. This is addressed particularly in the recommendation for an interdisciplinary final major project that should involve students in a five stage process of awareness, practice, feedback, reflection, and further practice post-feedback [11].

This interdisciplinary project would also allow them to learn about, experience and reflect up on different team roles [2] and if we are going to prepare students to work in a globalized industry, students should also develop an understanding of cross-cultural capability [5]. Moreover, we would recommend ongoing journal writing throughout the entire program in order for students to reflect on their own progress and learning. As part of a final-year thesis project, students are required to create a portfolio that not only demonstrates technical competence, but is also accompanied by written analyses of their own work that explains their thought processes, self-criticism, and research methods for the projects, design philosophy and approach taken. In this way, we are creating reflective practitioners that have the skills to analyze and discuss their work with others.

4. INSTITUTIONAL AND INTERNATIONAL BARRIERS

In addition to consideration of the difficulties described above, it was clear from the start that any curriculum guideline that would be truly useful would have to be extremely flexible and adaptable. The reasons for this became immediately apparent in our discussion of our own international and institutional differences. A summary of the difficulties in more detail will explain the challenges more clearly before we describe the approach we took to combat these difficulties.

4.1. International Issues

One of the initial difficulties encountered in discussions arose from the issue of internationalization and language. Wanting to create a curriculum guideline that can be used by a variety of undergraduate systems
around the world meant dealing with different terminology even within the English language described in the Introduction. Confusion amongst the group was quickly sorted out by defining a set of terms at the outset that we could use for the duration of the project. Language was a fairly easy problem to tackle, therefore, but more difficult problems in terms of internationalizing the curriculum arose. A Bachelor of Arts degree commonly lasts three years in most of the European Union, whereas it lasts four years in most of North America, for instance. Many countries have different systems and structures of organization for undergraduate teaching. Even within some countries, such as the USA, there are differences in terms of in-class hours per semester, making the structuring of a curriculum difficult. Tied in with this are the ranges of existing skills expected of students upon entering a degree program. In the UK, for instance, students have typically finished their general education program and have taken a two-year A Level (Sixth Form) series of specializations in their areas of interest. Planning for a degree program that can be implemented in many different countries, therefore, involves planning for a considerable degree of flexibility with regards to the amount of time that may be spent on a subject/topic area.

4.2. Institutional Issues

Not entirely divorced from the issue of internationalization are the many institutional differences, not only in terms of structure, but also in terms of requirements, cultures, and levels of bureaucracy that can make a full implementation of the guideline unrealistic. Many institutions have their own distinct degree requirements. At the University of Waterloo in Canada, for instance, BA students in all subjects must take a minimum of six courses from the Humanities (English, History, Languages, Philosophy, Fine Arts and Religious Studies) and four courses from the Social Sciences (anthropology, economics, geography, political science, psychology and sociology). Each major has an additional set of requirements that must be met. A 3-year BA degree consists of 30 total courses. A full year’s worth of courses, therefore (10 courses) is already taken up by meeting the compulsory general BA requirements, leaving only 2 years worth of courses for specialist training in the chosen major. A four-year Honours program requires an additional year with more individual research work, and a five-year Co-operative BA program requires work terms between study terms where students undergo paid internships. Pennsylvania State University’s Music program, on the other hand, requires a general composition course, 45 ‘general education credits’ in addition to its 79-88 credits for the major (where approximately three credits make up a course as we have defined it). Another significant related issue that arose from the planning stages is the political realities of higher education and how the curriculum can be implemented into a variety of different styles of program (fit around existing programs, become an entirely new program, etc.). An ongoing dichotomy throughout the development process meant the blue sky curriculum had to confront the reality of disciplinary boundaries, university administration, accreditation procedures and various associated bureaucratic barriers to quick adoption.

As we planned out our curriculum, we were quickly confronted by the inter-disciplinary needs of the curriculum described above. We are not just dealing with sub-divisions within a single faculty, but are crossing faculties in numerous cases between the Sciences and the Arts and Humanities. This raises the question of whether graduates should obtain a BA or BSc, for instance, which can require different processes for approval, assessment, and so on. It also means that co-operation must take place between faculties (which can be difficult when Universities fund faculties differently based on enrolment, etc.). Moreover, to have projects to work on (i.e. games), students must collaborate with departments who design and develop games —these, too are currently divided into very different disciplines, coming from both the Arts (game studies/digital media departments) and Sciences (computer science departments), which can have quite different faculty cultures [7]. These interactions can become significant challenges in terms of managing course credits, teaching, funding, and so on.

Many universities are, of course, recognizing the need for cross-faculty collaboration and programs that span both the Arts and Sciences. It has even been speculated that such collaborations are now instrumental to institutional survival. Notes Garry Brewer, ‘Longer-term survival, though by no means guaranteed, will depend increasingly on the ability of such schools to tear down barriers limiting access to one another and between themselves and the real world where problems abound’ [4]. He suggests that until a significant restructuring of the entire system occurs, the most reasonable solution may be to ‘take root in the interstices between existing programs, without threatening the dominant paradigm’ [4]. This need for flexibility at the institutional level is further complicated by the fact that curriculum
guidelines are more likely to be used if they allow for flexibility on behalf of not just institutional strengths and weaknesses, but also the strengths and weaknesses of the individual educators, who need the ability to bring their own skills, research, beliefs, frames of reference, methods and cultures of learning, theoretical preferences, operational objectives and training to their teaching [9].

4.3. our solution: creating an adaptable, flexible framework

With the difficulties discussed above, it was clear that the curriculum would need to be developed as small modular units that could be reconfigured into a variety of structures. Collaborative mind-mapping web software such as Thinkature was useful in our initial development and agreement on content and streams of specialization. A Scaled Curriculum Grid was developed that enabled a variety of programs to draw from: one semester technical programs, one or two-year college programs, four year university programs, and a six-year multi-track post-graduate program. The more traditional syllabi we developed for our blue sky curriculum and the Scaled Curriculum Grid was then adopted by international educators and ‘re-skinned’ into skills and learning outcome-based blocks that could be shuffled and adapted to fit around existing programs, to develop new programs, and to fit a variety of course time structures. These building blocks were prioritized based on the voting of the group members as to their importance described above, and so schools that were unable to implement all of the blocks could prioritize requirements.

5. CONCLUSIONS, SUGGESTIONS, AND NEXT STEPS

A summary of our process will be useful before we outline what we have learned from creating this Curriculum Guideline.

- Agreed on terminology
- Agreed on streams of learning based on typical division of labor in the workforce
- Collected job ads, highlighting skills listed
- Queried industry professionals about their job skills
- Collated skills information and ranked according to priority for our streams
- Chunked the skill-sets into skill-blocks that make sense to be taught together
- Chunked skill-blocks into core blocks that were deemed critical or important for everyone and set these as a common first year requirement Estimate hours to learn skills in-class
- Created larger sections of blocks into courses of about 30 hours instruction time
- Created a blue-sky model organization of courses as an example curriculum (USA structure)
- Developed syllabi that describe learning outcomes
- Returned to skill-blocks and reconfigure based on priority
- Developed real world tasks and case studies that can be incorporated into syllabi
- Created a list of resources educators can draw on to teach skills.

5.1. Suggestions

Based on our experience, we have created a list of suggestions for others faced with a similar task of creating an international curriculum guideline.

- Agree on a set of terminology at the start: This sounds fairly self-evident, but in many cases we came across misunderstandings based on terminological differences.
- Have an online collaborative space: We used Google Docs spreadsheets, Thinkature mind-mapping software, and SurveyMonkey to create surveys. This not only allowed us to see progress, but enabled us to understand other members’ thought processes, comment and adjust collaboratively without having to send around many versions of a document through email, which quickly became unmanageable with the many members of our group. We also had VOIP conference calls to discuss matters that became contentious.
- Get industry involved early: While there are often differences of opinion and understanding with regards to the purpose of higher education, by bringing in industry professionals as members of the group, and using real-world job advertisements, we ensured that the skills we would be promoting were the skills that would enable students not just to obtain employment, but to maintain employment. We use combinations of qualitative and quantitative detain analyzing industry needs.
- Prioritize learning outcomes/skill-sets: Not all schools will be able to ‘drag and drop’ a curriculum guideline into a new program. Many schools will need to know what the most important skills are, as there may be limited time available to teach in the area.
• Have a blue sky model: We created a four year curriculum plan fairly early in the process, and although this ended up being changed considerably by the end, it helped to have a full document to look at to see structure and direction.

• Plan for change/flexibility: We recognized that even by the time the curriculum guideline was finished it would have been out of date if we included reference to specific software titles, and so on. By leaving open-ended descriptions that can adapt to changing technology, and focusing on skill-sets that are more time-resistant, we can leave time and space for specific tools be taught.

5.2. Future Plans
The implementation of the curriculum guidelines is an issue that we are faced with currently, due to the fact that there are few educators available in the area of game audio. It is our hopes that with growing interest in games in general, as well as with the increased opportunity to attend game audio ‘boot camps’ (at the Game Developers Conference and at GameSoundCon, for instance) as well as packages of teaching materials (see below), that this lack of available educators will change soon. Another barrier we face is the current lack of pedagogical materials available to instructors. The creation of these materials requires considerable input from those currently practicing in the games industry – an industry whose employees are often overworked already. When textbooks do get written by those in the industry, they are sometimes created without a complete understanding of the needs of the classroom.

Educators need to collaborate as much as possible, therefore, when and where possible, to ensure the timely creation of teaching materials. This need has been partly met via the IAsig online interactive audio WIKI, and the growing number of books available in the area of game development. However, specific materials are still required, and as the technology is rapidly changing, a system of constantly updated material needs to be created and shared. As such, a wish-list of resources is being compiled with the aims of developing documentation and research in the areas that are still sparsely documented.

The success of an interdisciplinary program such as game audio requires a reconsideration of the ongoing separation on campuses of the Arts and Sciences faculties. Moreover, it is necessary that departments and faculties be open to cross-faculty collaboration and research, as well as practice-based research for the curriculum to be successful. Currently, in many universities it is still the case that faculty are assigned to a department, and that publication and/or teaching outside of one’s immediate discipline is at best misunderstood, at worst ignored and/or not given credit towards tenure, merit raises, and so on. Many new interdisciplinary areas such as game audio require faculty to maintain technological skills that change rapidly, in addition to disciplinary knowledge from a vast array of areas (for example, video game development and areas of computer science and programming, music technology, industry trends, sound design and film and game theory). A further consideration is to obtain ongoing feedback on the curriculum, in order to evaluate its success. This curriculum auditing is a necessary process of quality control. While the IAsig initially proposed the possibility of creating a certification system or having a standardized certified testing system to ensure that students were obtaining the skills that are required, ultimately we decided against these solutions because they would institute yet another barrier to rapid adoption, as well as inhibit the flexibility needed (as discussed above). Instead, we propose to evaluate curriculum through qualitative and quantitative research, including documents, interviews and/or surveys and observations and on-site visits, as well as statistical data regarding attrition and graduation rates, as well as post-graduation job placements.

6. ACKNOWLEDGMENTS

The authors would like to thank the Interactive Audio Special Interest Group (particularly the Education WorkGroup), as well as members of Project Bar-B-Q and the Audio Engineering Society for their input on the curriculum guideline and on this paper. A copy of the curriculum guideline will be made available through the Interactive Audio Special Interest Group (IAsig.org) in 2010.

This paper was written by the collaboration of Ufuk Onen of Bilkent University, Richard Stevens of the Centre for Creative Technology at Leeds Metropolitan University, and Karen Collins of the Canadian Centre of Arts and Technology at the University of Waterloo.

7. REFERENCES

[1] 2009 Audio Engineering Society Convention


Karen Collins, CCAT, University of Waterloo, 200 University Ave W. Waterloo, ON Canada N2L3G1 (collinsk@uwaterloo.ca)

Richard Stevens, Centre for Creative Technology, Leeds Metropolitan University, 209 Caedmon Hall, Headingley Campus, Leeds, UK LS6 3QS (r.c.stevens@leedsmet.ac.uk)

Ufuk Onen, Faculty of Art, Design and Architecture, Department of Communication Summit, and Design, Bilkent University, TR-06800 Bilkent, Ankara, Turkey (mail@ufukonen.com)

Designing an International Curriculum Guideline for Game Audio: Problems and Solutions 47
Studying Commercial Games: Justifying Choices

Katrin Becker
Simon Fraser University
becker@minkhollow.ca

ABSTRACT
Before researchers can perform studies using commercial games, they must choose which game or games to study. The manner in which that choice is made and justified is the focus of this paper. Ideally, research informs pedagogy and when looking at game education it is important to be able to justify and defend conclusions drawn from game studies so they can inform best practices in design and development. As the number and sophistication of titles released in a given year continues to rise, it becomes even more important to look more seriously at how we are choosing the games we study, the criteria we use for those studies, how we support our claims about the suitability of the game for our purposes, and how generalizations to other games should be limited or qualified. This paper is a report on a qualitative meta-analysis of the methods used in choosing games for study and the implications that holds for both researchers studying games and educators teaching about games and game development.

1. INTRODUCTION
Digital games have been around for about 45 years now (Williams, 2006) and game studies as a recognizable discipline has been around for a little over ten years (Wolf, 2001). Digital games continue to grow both in popularity and variety and the discipline of game studies as well as programs in game design and development are growing right alongside them. Methodology in game research is largely still an ‘undiscovered country’ and although many useful approaches from other disciplines are used, the nature of the artifacts themselves (i.e. digital games) have sufficient unique elements to warrant the development of some game-study-specific approaches. A first step in the process of developing such approaches is to begin to classify the kinds of research done using games. The kinds of research conducted with and on games can be subdivided into several broad categories that necessarily affect the both what kinds of approaches are appropriate as well as what kinds of conclusions can be drawn. One category involves examinations of one or a small number of specific games, which is the focus of this article.

Examinations of specific games can produce various insights in the same way that examinations of specific literary or other artistic works can, and while there remains an interest in examinations of certain specific games for specific detailed purposes, such as Kurt Squire’s doctoral study using Civ III (2003), as the number and sophistication of titles released in a given year continues to rise, it becomes necessary to look more closely at how we are choosing the games we study, the criteria we use for those studies, and how we support our claims about the suitability of the chosen game(s) for our purposes. Often, studies of individual games are conducted with the intent to generalize at least some of the conclusions to other games and/or other players. Sometimes studies seek to address questions about entire genres. Given the number and variety of games with no cleanly defined delineations of genre, how can we increase the confidence that it is even possible to examine one game in order to make generalizations to other games?

Many games are no longer trivial or frivolous so the question of generalizability is not a straightforward question. In an established discipline, claims that a particular artifact or work meets certain criteria critical to the analysis are usually supported by something beyond the author’s say-so. As studies on, with, and of games become more accepted
and common in mainstream academic research, it will also become more important to justify the choices of subjects. It did not appear as though such justifications were the norm, so a meta-analysis was conducted to determine the reporting frequency of explanations of subject choices in the existing literature. The analysis made no assumptions about the validity of the results reported in those papers, the purpose was specifically to find out whether authors explained or justified their choice of game(s). Several sources of game study literature were consulted (such as the DiGRA & FuturePlay conferences and the author’s own reference library collected during five years of doctoral research), along with a more general Google Scholar search in order to gather literature on games from various disciplines. Only studies that specifically mentioned commercial games were considered, and of those, only those studies and reports were selected where the question could legitimately be asked, “Why was this game chosen?”

2. THE ACADEMIC STUDY OF GAMES

Why is it important to justify the choice of game being used as an example in a scholarly article or for the purposes of study? In the early days of games studies there seemed little call for careful scrutiny of one’s game choices. We studied what we had handy and wrote about the games we happened to be playing. However, if we want to make the case that the game in question can lead to some broader insight or that it is in some way representative then we really should have some evidence to back this up. When a single game or a small number of games are chosen as the subject(s) of study they form part of the bounded system that is the case being examined, which in turn implies that there are identifiable aspects of the game(s) that makes the case of special interest (Stake, 1995). If we are proposing the use of a game for some serious purpose such as education or social change, or if the study of some specific game is intended to uncover some insight applicable to our agenda, then whether that agenda is the examination of its educational potential or the discovery of something that can inform design or development, then as academics we have a responsibility to explain why that game is suitable for our purpose.

One compelling reason for putting thoughtful effort into justifying the choice of a game used in a study is that it helps to make the study itself more credible. This has implications for the increased acceptance of game studies academically as well as for helping to improve relations between academia and the games industry. In a recent article offering suggestions for how the Academy could build stronger ties with the Games Industry, John Hopson argues that we should “(u)se examples from bestsellers. A good example from a popular game is more effective than a great example from something they’ve never heard of. Industry people often suffer from an ‘if-they’re-so-smart-, why-ain’t-they-rich’ attitude towards smaller titles. Even if the small title is a perfect example of how the theory works, they’re going to be less likely to listen if they haven’t heard of the game ahead of time. Commercial success is one way of making sure that the audience will respect your examples, but you can also use titles that are well known or critically acclaimed but which weren’t necessarily huge blockbusters. It’s also important to keep your examples as current as possible, because many industry folks will see a three-year-old example as ancient history” (Hopson, 2006).

The field of game studies and game education includes a unique combination of academic and industry literature rarely seen in other disciplines. This synergy has many advantages but also causes some difficulties, especially in academic circles. In games, much useful information and insight comes from non-academic sources that are not normally subject to the kinds of peer review to which we in the Academy are accustomed and thus are not typically recognized as carrying the same weight as those that come from more traditional peer-reviewed sources. One way to help address this discrepancy is to ensure that the defensibility of the academic studies can stand up to close scrutiny, and one way to ensure that is to make sure that the specific games chosen for study can be objectively justified as meeting the goals of the study.

3. OBJECTIVE JUSTIFICATION

While the field of game studies is rapidly gaining momentum with more journals and conferences being offered every year, it is still in the process of building academic credibility and rigor, and defensibility of research as well as methodology remains critical. Based on a cursory examination of over 1000 recent scholarly publications that mention ‘digital games’ it was determined that the games described there can be broadly categorized into three groups:

1. Non-commercial games that have been designed
or developed by the authors,
2. Non-commercial games (including but not limited to serious games) that have not been designed or developed by the authors, and
3. Commercial titles.

The classification scheme proposed in the following sections can be applied to studies involving any of these three broad groups, but it is the last group, namely studies involving commercial titles that are of interest here. For the purposes of this article, the last group includes commercial games designed or developed by the authors of the publication. A case in point is that of studies of violence in games, and highlights the importance of careful justification of the choices of games to verify their “fitness for purpose”. One of the criticisms of many ‘media-effects’ studies is that these studies commonly include a broad variety of games and treat them as though they are essentially interchangeable (Freedman, 2001). In a longitudinal study of violence in an online videogame, Williams and Skorik raised questions about the generalizability of games which have implications far beyond their own study. “The online database www.allgame.com lists descriptions of more than 38,000 different games across 100 platforms. To collapse this wide variety of content into a variable labeled ‘game play’ is the equivalent of assuming that all television, radio, or motion picture use is the same” (Williams & Skoric, 2005).

As Dill and Dill (1998) have noted, “This is akin to lumping films like The Little Mermaid with Pulp Fiction, and expecting this combined ‘movie viewing’ variable to predict increases in aggressive behavior” (1998, p. 423).

By not providing careful rationales for our game choices, we are not paying sufficient attention to the great variety of games available, and in doing so we risk nullifying any results that come from such studies. The existence of such a large number of games means that we cannot assume that one game is as suitable as any other for the purposes of study. In other words we cannot collapse all ‘adventure games’ into one category and assume that what we discover about one adventure game will apply to some other adventure game. Both Grim Fandango and God of War are listed as adventure games by Mobygames.com and both rank among the top 10 in that genre, yet it would be inappropriate to study the design of one for insights into why the other is successful. Similarly, studying ONE game does not necessarily allow us to generalize our findings to any other games. This not only has implications for results involving the players, but also for any conclusions we attempt to draw about the game’s design. Without careful qualification of the choice of game for a given study there can be no generalization to other games.

While a suggestion to force all games researchers to apply some sort of ‘scientific’ approach to their choice of games is clearly unreasonable, paying closer attention to how we choose games and making a point of explaining those choices can certainly help address legitimate questions about a game’s fitness for purpose in the context of a given study. We may not feel the need to justify choosing Shakespeare’s Hamlet if our purpose is to make some generalization about tragedies, but video games are not in the same category as classic literature. Games have not yet attained the level of acceptance, nor of unarguable classification that classic literature has and until they do we should still be explaining our decisions – especially if we hope to realize some findings that could be applied to some other game or some other population. On the other hand, providing defensible rationales for game choices does not preclude the possibility of choosing a game because it is one we personally like, but we still need to address how that makes that game a worthy candidate for study. If we choose a game because it is popular, then we should be able to support that claim with facts or citations that can stand up to scrutiny. Further, we should be able to explain why a game’s popularity is germane to our study. Similarly, if we claim that a specific game is representative of an entire genre, then it is reasonable to back that claim with further references.

4. META-ANALYSIS OF GAME STUDIES

Since the question of how games are selected by researchers has not previously been examined the author conducted a qualitative meta-analysis (Delgado-Rodríguez, 2001) of what methods researchers reported using in choosing games for study. Papers and reports published over a five year period between 2003 and 2008 were examined with the goal of determining the reporting frequency of explanations of game subject choices.

Since there is no way to verify that a lack of information about the selection criteria applied to the choices of games indicated that none existed, it should be noted that a lack of explanation in the publication does not prove a lack of consideration for the study. It is certainly possible that carefully considered reasons motivated the game choices in
many of the studies presented here, but that these were simply not included in the publication. The worthiness of the choice that was made was also not being examined here, and indeed many well-known game scholars were included in the list of papers examined. In many cases there would be little controversy over the claim that the chosen game has the characteristics described in the paper. In some cases there would also be no dispute that the particular type of game is a suitable choice (and perhaps even the most suitable choice) for the study as reported. Many of the reports have contributed to the body of knowledge in games studies in important and significant ways. The concern has to do with verification.

5. APPROACH

In order to report as widely as possible and since this study sought to discover why researchers chose the games they did, multiple publications by the same authors using the same game were avoided, unless that game was being used for a different purpose in each study. Two separate analyses were performed and the data were combined. In the first, a variety papers from scholarly publications released between 2003 and 2006 that reported on research involving at most five distinct games were chosen. In the second papers from 2006 that had missed the cut-off date from the first analysis through papers published in 2008 were included. Both sets of papers were examined to discover which games were chosen and whether an explanation or justification of the game choice was included. Studies featuring games like DDR (Dance, Dance, Revolution) were not included in the first analysis since there were so few commercial kinetic games available at the time, the rationale is understood UNLESS the study was looking at some aspect of the game other than its interface. Similarly, studies that focused on a characteristic aspect of a specific game (such as effectiveness of recruitment in America’s Army) were also excluded. A distinction was made in the meta-analysis between the description of the game (including gameplay and any noteworthy features of the game) and a rationale for the choice of the game. Virtually all studies described the games that were being used but these descriptions were rarely connected with the reasons for choosing that game.

6. STUDY CLASSIFICATION

In order to be able to classify studies in a manner that would permit some kind of comparison it was necessary to group the studies and the following classification was devised for this purpose. All papers were examined to determine the purpose of the study and five groups were identified:

1. Specific studies were ones where a specific game is used that had no identifiable substitute. In other words, if the questions were asked, “Could some other game have been used?” the answer would have been no. An example of this is Squire’s early study using Civ. III (Squire, 2003). It could be argued that at the time, there really was no other game that could have been used in this study.

2. Typical studies were ones where the game was claimed to be a representative example, such as an MMO, or that it supported an in-game economy that has a real world value. In some cases, the reason given was far more nebulous, such as that it was “interesting”.

3. Apparatus studies were studies where the game was used as an apparatus rather than the focus and the object of the study was something else, such as the game was being used as the basis of a writing assignment, and it was the writing assignment rather than the game that was being studied.

4. Mod studies were ones that made use of some commercial game or engine but where the study focused on the mod rather than the original game.

5. Other: this was the ‘catch-all’ category for studies that could not be placed into one of the other groups.

7. FINDINGS

The meta-analysis included 89 papers that were examined in detail. 131 games were identified comprising 93 distinct titles (some studies used more than one game but numerous studies used the same games such as World of Warcraft). Three quarters of the studies reviewed indicated that the game(s) were in some way representative. While most authors made some attempt to explain their choice of game, most of those explanations were effectively unsubstantiated opinions such as claiming that the game is highly successful or popular. The claims are rarely supported with other data. Only one paper out of the 89 examined reported having applied some systematic technique to identifying candidate games for study. Most of the papers examined included a synopsis of the game being studied, but only about 30% offered an explanation for how or why this game met the needs of the study, and fewer still (9%) supported that explanation with citations. For example, popularity is given as a rationale in 3% of reports, but none explain how the game’s popularity is germane to the study. Some included explanations that are either difficult to verify or substantiate in any
objective way, such as that the game is interesting. Obvious rationales such as “we needed an MMO and this game is one” still beg the question, “Why THAT MMO?” In a third of the studies the explanation for the choice of game included a statement of a requirement for the study. For example, the game needed a strong story-telling component or an ability to play the game from multiple perspectives. In most cases nothing is offered to support claims that the chosen game meets the specified criteria other than the authors’ assertion that it does.

While there may not be much controversy over claiming that Midtown Races is a driving game and that a driving game is needed for a study on fear of driving after an automobile accident (Walshe, Lewis, Kim, O’Sullivan, & Wiederhold, 2003) it may still be appropriate to ask in what way this particular driving game fits the need. Other claims, such as choosing Doom because it is easily “mistaken” for violent (Molesworth, 2007) deserve objective support. Only one study described a rationale for the exclusion of one or more games from study (Warnes, 2005) and one other report actually explained the methodology used to select the game for the study. Henderson (Henderson, 2005) allowed the study participants to vote on a game, citing prior research that suggested participant interest was an important factor in the study’s success. It is suspected that for many several of the studies many game choices were, at least in part opportunistic, as the researchers had access to or were already playing this game. Only one researcher actually stated that they were already playing the game as their explanation for choosing it (Chen, 2005). In three other cases, the researcher states that they have prior experience with the game but it is not made clear whether the study began before or after that individual began to play that game, nor how much influence the author’s own game playing preferences had on the choice. Comments such as, “I’ve been playing this game for years” places game studies in a somewhat unique position as both casual and avid gamers draw on their own playing experiences to help inform their studies. This kind of connection places many game studies in the realm of what Glesne has called “Backyard Research” which can make separating researcher roles from pre-existing ones complicated and difficult (Glesne, 1999). Support for claims about the game’s popularity does not necessarily speak to a game’s fitness for purpose, and yet one in ten of the studies used this as their primary justification for choosing the game. In other words this MMO was chosen because it is popular when the focus of the study itself was something other than popularity.

If a claim is made that a particular game was chosen because of its open-ended gameplay, it would be helpful to cite other sources that agree with this claim. By far the most common attribute supported by other references is the claim about the game’s popularity and the most common outside reference is to sales figures. Simply stating that a first person shooter was needed for this study is no longer sufficient to justify the choice of genre – it should be possible to explain what qualities of this genre are important. Was the particular game chosen because it was a representative example or because it was the best or worst example? What evidence is there to support these claims? Fitness for purpose “equates quality with the fulfillment of a specification or stated outcomes” (Harvey, 2004). If a researcher claims that a particular game is an appropriate

8. CONCLUSIONS

One of the things that many aspiring researchers learn early in their careers is that they must support the claims they make in their writings. A common note written by supervisors in the margins of almost every thesis draft states something to the effect of: "Says Who?", or “Where does it say that?” We are often told that if we cannot support a claim then we should not make it. Should we expect less of studies that use games? The results of the combined meta-analysis indicate that a very small minority of game researchers currently report on the methodology used for the choice of a game in a study, or use examples of excluded games to support their choices. Very few explain how or why their stated game requirements support the goal of the study. While some cite references to support at least some of their claims about why this kind of game is needed for this study, almost none cite any references supporting their claim that the chosen game actually meets those requirements.

If the study is looking at a game’s economy or some aspect of its interface design, then the fact that the game was popular does not necessarily imply that it is a representative choice. In the author’s own work examining player learning support in highly successful commercial games, several of the top choices had to be eliminated because they did not offer sufficient support (Becker, 2008). In other words justifying the choice based on the notion that the game’s popularity implies some requisite level of quality does not always work.
choice for a particular study then it is appropriate to offer justification for that claim. Given the great number of games available, it is no longer sufficient to claim that a particular game meets certain criteria without supporting that claim in a verifiable way.

Even though critical and commercial success are both recognizable and accepted measures of a game’s popularity, and popularity in turn gives some indication of that game’s perceived quality as judged by players, developers, and game critics, these are also highly subjective measures.

Katrin Becker, PhD, Adjunct, School of Interactive Arts and Technology Simon Fraser University, Surrey BC

9. REFERENCES
Skins: Designing Games with First Nations Youth

Beth Aileen Lameman  
Simon Fraser University  
beth@bethaileen.com  

Jason Edward Lewis  
Concordia University  
jason.lewis@concordia.ca

ABSTRACT
Aboriginal Territories in Cyberspace (AbTeC) conducted the Skins Pilot workshop to explore a pedagogy that integrates North American Indigenous cultural frameworks into the design of digital games and virtual environments. Skins provides instruction in digital design, art, animation, audio, and programming within a context of Aboriginal stories and storytelling techniques. In the pilot workshop with Mohawk youth at the Kahnawake Survival School, students developed a digital game based on traditional stories from their community. Students were interested in integrating stories from their communities in digital games, respected but modified or expanded the stories where appropriate, and were capable of translating those stories through the complex means for developing a video game. Encouraged by these outcomes, AbTeC is adapting the Skins Pilot yearlong curriculum into Skins Intensive, a two-week intensive curriculum for a summer workshop that offers college credits through Concordia University. This paper describes the motivations behind revisions to the curriculum and considers possible advantages and disadvantages.

CATEGORIES AND SUBJECT DESCRIPTORS
General Literature [GENERAL]; Conference proceedings

GENERAL TERMS
Design, Human Factors

KEYWORDS
Game design, education, curriculum, workshops, youth, First Nations, Aboriginal, American Indian, Native American, Indigenous

1. INTRODUCTION
Aboriginal Territories in Cyberspace (AbTeC), based out of Obx Labs at Concordia University in Montreal and directed by Lewis and Skawennati Fragnito, is a network of academics, artists, and technologists that encourages Indigenous participation in online culture and exploration of new media technology. The main objective of the AbTeC research network is to discover, define, and implement methods by which Indigenous people can use networked communication technology to strengthen our cultures.

In an effort to overcome the economic, social, and cultural factors that influence the low rate of Indigenous participation in the making of new media and encourage Indigenous representation in digital games and virtual worlds specifically, AbTeC proposed to conduct Skins—a game/virtual world development workshop for Aboriginal youth that teaches them design, programming, art, animation, and audio.

In developing the curriculum for Skins, AbTeC brought together diverse academics, artists, activists, and technologists in workshops that involved discussions around the role of new media technologies in North American Indigenous cultural production, outlining curriculum for teaching First Nations youth how to use such technologies, and testing curriculum. Some of the participants included Nacho Nyak Dun storyteller Louise Profeit; Mohawk new media artist and curator Steve Loft; Dogrib writer Richard Van Camp; Cree comic book artist Steve Sanderson; Cree filmmaker Myron Lameman; game researcher, designer, and curator Celia Pearce; game and human computer interaction researcher and designer Katherine Isbister; digital culture, games and virtual worlds researcher Bart Simon; game animator and designer Ken Finney; and Microsoft Canada’s Christian Beauclair.
Simultaneously, AbTeC created a studio/research outpost in the virtual world Second Life (SL). Using SL, AbTeC made TimeTraveler™, a machinima series and alternate reality game which won Best New Media at the imagineNATIVE Film + Media Arts Festival 2009.

We drew upon what we had learned via research, workshops, and projects to design curriculum and launch Skins. The Skins Pilot took place over one year (2008-2009) in the senior level art course at Kahnawake Survival School—the high school of the Kahnawake Mohawk First Nation in Quebec, Canada. We found that 1) students were interested in integrating stories from their communities in digital games, 2) they respected but modified or expanded the stories where appropriate, and 3) they were capable of translating those stories through the complex means for developing a video game. Due to these positive outcomes, we are adapting the curriculum for the Skins Pilot (yearlong workshop) to Skins Intensive (a two-week intensive workshop that offers credits at Concordia University).

In this paper, we first discuss the motivation behind Skins. Secondly, we compare other game/virtual world development workshops for youth to Skins. Then we discuss the curriculum and findings of the Skins Pilot workshop (2008-2009) in relation to influencing the forthcoming Skins Intensive workshop (Summer 2011). Lastly, we consider future work for Skins Intensive based on lessons learned from the Skins Pilot.

2. MOTIVATION

In conducting Skins, our goal is to encourage First Nations youth to be more than consumers of digital media; rather, we wish to show them how they themselves can be creators who can approach games with a critical perspective and from within their own cultural context. We are motivated by the possibilities of digital games and virtual environments for Indigenous peoples as well as correcting or adding to representations of Indigenous peoples in commercial games.

Indigenous peoples’ survival, recovery, development, and self-determination hinges on the preservation and revitalization of languages, social and spiritual practices, social relations, and arts [1]. Digital games and virtual environments, with their unique combination of story, design, code, architecture, art, animation, and sound [2], provide a rich medium though which to explore different strategies for pursuing such preservation and revitalization. For example, Thornton Media’s RezWorld is a virtual environment for learning the Cherokee language. It has even been argued that the fluid, open, and networked characteristics of modern digital media make it particularly useful as a tool for Aboriginal storytelling, with Loretta Todd, Cree/Métis filmmaker and Director of the Aboriginal Media Arts Lab, suggesting “the experience of cyberspace offers the reversal of narrative as derived from storytelling, a return to oral tradition” [3]. Furthermore, due to the radical decrease in the costs of the means of production and distribution, digital games and virtual worlds present Indigenous people with a powerful opportunity to widely (or narrowly) communicate stories in which we shape our own representation.

Unfortunately, most Indigenous representations in commercial games and virtual environments are lacking. Lameman (né Dillon) has conducted extensive investigations into Indigenous representation in both game content and production teams. Her research has shown that, while North American Indigenous youth are known to be avid video game players [4], they rarely appear in commercial games, and when they do, they are misrepresented in aspects including culture, behavior, and language [5, 6, 7]. For example, GUN—a Sandbox Third-Person Shooter in which the player character Colton White fights his way through the West—includes missions to kill Apaches and scalping as a killing move. The Association for American Indian Development (AAID) consequently ran a boycott against GUN. Additionally, although Age of Empires III: The WarChiefs (Fig. 1) treats American Indian/Alaska Natives positively as allies to European colonists, the game has been criticized for its revisionist approach to history [8] and its lack of incorporating Indigenous thinking in the gameplay [9]. For example, in The WarChiefs, players have to gather resources like wood, but are never given the ability to replenish resources by planting trees. Players experience land as a map with borders to be discovered, fought for, and claimed. Overall, where North American Indigenous characters are playable, they are stereotyped, and where they are side-characters, they are targets of violence [10].

Figure 1. Icons for Character Units from Age of Empires III.
These representations are not surprising when considering that there are few North American Indigenous employees in the game industry. Despite the wide range of opportunities in game and virtual worlds development teams, which are made up of designers, programmers, artists, animators, sound engineers, and others, there is still a clear lack of diversity in employment. In the most recent survey by the International Game Developers Association, 83% of respondents stated they were “White” and 88% stated they were male [11]. American Indian/Alaska Native and First Nations representation was so minor—totaling only 41 respondents, most still in school—that statistics were not reported.

In order to address concerns in diversity, enabling communities with technology, education, and experience is paramount. American Indian/Alaska Native and First Nations representation in the game industry is likely low because there are so few in Computer Science—the percentage of American Indian/Alaska Natives earning Computer Science bachelor’s degrees in the U.S. only increased from 0.4% to 0.5% during 1985 to 2005 [12].

Interestingly, Katherine Isbister suggests fostering training and production environments in which people of a specific group can produce stories, characters, and gaming scenarios that are imbued with their aesthetic, concerns, and interests as a community [13]. Skins speaks to this approach.

3. YOUTH WORKSHOPS

Prior to Skins, Lameman conducted a game workshop for Anishinaabe First Nations youth at Algoma U in Sault Ste. Marie, Ontario. However, few game/virtual world development workshops reach out specifically to Aboriginal communities. There are several workshops offered to youth in general, such as Activate!, Emagination Computer Camps, and Game Dev Camp. The overall hope for all workshops is for youth to take the knowledge learned—whether practical skills or developmental life lessons—into the future, by accessing higher education and possibly entering media industries. The structures and findings from these comparative workshops offer insights into Skins.

Although game/virtual world development workshops are often short—ranging from a few hours to a few months—they rely on long-term layered learning to retain and enhance the varying skills needed to develop a game/virtual world. The Skins Pilot benefited greatly from the ability to stretch curriculum out over one school year, but even so, needed more time. The Skins Intensive workshop faces an even greater time restraint.

Like Skins, most game/virtual world workshops for youth bring in experts from the industry to give students hands-on experience with people who work day to day with the technology. Researchers are brought in for conceptual elements, such as the importance of characters in creating emotionally driven games. There are also numerous resources online for students to access during the workshop or afterward, ranging from tutorials for using software to articles about how to break into game industry. All of these additional resources for knowledge are essential in Skins.

AbTeC was motivated to select the virtual world Second Life and the game engine Unreal (Fig. 2) as platforms because of the benefits offered by “modding”. “Modding” involves modifying an existing game/virtual world by changing code and assets using an existing engine rather than developing a game/virtual world from scratch. Modding workshops help students adapt a complex system and critically reflect on commercial games made with the same technology [14], since youth work with games that they have played, but with the opportunity to change environments, stories, characters, and objectives to suit their creative goals. Notably, modding costs less and is less time consuming than commercial virtual environment and game development [14].
game development workshops task youth with creating educational games that will appeal to their peers [15, 16, 17]. Other workshops focus on teaching skills like programming or engineering through the process of designing a game [18]. Skins differs in that the curriculum integrates Aboriginal stories and storytelling techniques.

Similar to Skins, there are workshops geared toward non-gamers. Much attention in this area has gone to girls due to the lack of female participation in game culture and the game industry. These workshops tend to start from the basis that girls play games less than boys, or play games only from certain genres. Regardless of prior experience with game play, game/virtual world development workshops are accessible pathways to learning communications technologies, in part because of the shared knowledge amongst youth [19].

Jacob Habgood, who hypothesizes that anyone can and should learn how to develop games, adapts Bloom’s cognitive learning to his workshops [20]. Students participate in six kinds of engagement: 1) knowledge, in which they observe and recall factual information, 2) comprehension, in which they understand the meaning of knowledge, 3) application, in which they apply knowledge in new situations, 4) analysis, in which they identify and extract patterns in knowledge, 5) synthesis, in which they use old ideas to create new ones, and 6) evaluation, in which they reflect on the ideas. Skins follows this structure, but with unique considerations for Aboriginal storytelling and acknowledging processes used in the game industry.

In Yasmin Kafai’s workshops [16, 17, 18], youth designed games with math and science topics for other youth to play and simultaneously learned programming; contemplated interface designs; designed visual elements; came up with stories, dialogue, and characters; wrote instructional strategies; and created fraction representations. Youth were put in the active role of constructing strategies; and created fraction representations. Youth became active participants by having team or personal direction in workshops. The Girls Creating Games Program found that, when girls were provided the skills and support to design choose-your-own adventure games, they resisted gender stereotypes by offering players the chance to win or lose the game and providing more opportunities for personal triumph than opportunities to help others [21]. They also found that when youth are given the option to express themselves, they are more likely to retain knowledge [21]. Similarly, in the Skins Pilot, youth excitedly engaged with traditional stories and reworked them unexpectedly.

In all of these workshop examples and in Skins, participatory design is paramount in education. Participatory design projects need clear definitions on the roles of participants and how they contribute [22], much like game/virtual world development, which requires a team with a variety of very specific roles. Youth can learn design, programming, art, animation, writing, communication, sound design, and a myriad of other skills in the process of developing a game. When youth take on individual responsibilities, they feel accomplished as a member of the team. Instructors and mentors can help each student with their role, but it is important that each youth have a sense of individual participation and learn specific skills [23]. It is the hope that this participation, in turn, encourages youth to pursue higher education and helps shape their career interests.

4. SKINS

The curriculum, dynamics, and results of the Skins Pilot held in 2008-2009 inform Skins Intensive, a two-week version of the Skins workshop to be offered in 2011. The Skins Pilot took place at Kahnawake First Nation’s high school, Kahnawake Survival School, in the Senior Graphics Arts course taught by Owisokon Lahache. Ten Mohawk students participated.

4.1 Curriculum

The Skins curriculum teaches First Nations youth a multitude of skills related to game production while at the same time encouraging them to develop game concepts, characters, and mechanics based out of their own cultural experience [24]. The curriculum materials can be used in numerous settings, whether intensive or long-term workshops, extracurricular programs, or in-class projects. Skins can also be adapted into a regular class schedule, as its content could fit well in Culture, History, Art, Design, and/or Technology classes.

The curriculum contains several aspects that are uniquely oriented towards First Nations students. One is the emphasis on traditional stories and storytelling.
techniques, which serves to both encourage youth to reflect on how stories are transmitted in their community and how they themselves can participate in the preservation, evolution, and future transmission of those stories. A second such aspect is the inclusion of a community partner who plays a central role in mentoring the youth and ensuring that cultural elements, such as language and stories, are represented in ways that reflect the history and values of the community.

The curriculum covers traditional storytelling as well as topics central to game and virtual environment production, including: Aboriginal storytelling traditions, Aboriginal storytelling techniques, Aboriginal storytelling across media, concept development, interactive narrative, level design, art direction, 3D modeling, 3D animation, digital audio, and project management.

4.1.1 Modules
Lessons are broken down into 4 modules—Play, Storytelling, Game Design, and Technical. Play involves playing digital games, board games, and viewing films by Aboriginal filmmakers. Storytelling helps students reconnect with their culture and reflect on choices for their game design. The Game Design module walks students through the pipeline process without needing to use technology. Technical lessons break down the practical skills needed to develop a virtual world or video game.

The Play module was created to provide opportunities for students and mentors to get acquainted with each other at the beginning of the workshop series by playing digital games, board games, and viewing movies made by Aboriginal filmmakers. These sessions can also be scheduled throughout the workshop series to break up long work periods.

The lessons in the Storytelling module are designed to give students a chance to listen to stories told by elders as well share their own stories. Students learn about the different ways that stories can be told from traditional oral storytelling to comic books, films, and narrative in digital games. The lessons at the end of this module are meant to help students work as a team to decide on a story they would like to tell in their own video game. Students will also learn how to write the narration that will be featured in the introduction and cut scenes for their game.

The Game Design module helps students take plan their game from concept through implementation via group work. The lessons concern how to create a basic layout and design for game creation through paper-prototyping, how to test the game idea without even turning on a computer, and designing the concept art for the game.

The lessons in the Technical module introduce students to the hands-on work required for implementing a successful game, including Modeling, Textures, Animation, Unreal, and Sound. Lessons involve the production pipeline from beginning to completion and focus on animation, texturing, and modeling. Notably, the module employs Second Life as a preliminary step before using the game engine Unreal.

4.2 Skins Pilot
We learned several must-haves during the Skins Pilot that help inform Skins Intensive: 1) flexible curriculum, 2) a dedicated instructor connected to the community, 3) defined roles, and 4) creative freedom.

The Skins Pilot resulted in iterative design of the curriculum—meaning designing, testing, and revising curriculum during the workshop. The on-site research team (Lewis, Fragnito, and research assistants) held weekly meetings where Lahache joined when possible. The curriculum constantly adjusted as the workshop developed. Causes for adjustment included the students’ rate of progress, input from guest lecturers, and changes in the school’s general academic schedule.

Lahache’s participation was central to the success of the Skins Pilot. As a respected teacher, artist, and traditionalist within the Kahnawake community, she brought a deep understanding of both storytelling traditions and creative expression within it. She fully embraced the curriculum, to the point that she herself learned all of the technical skills involved, and her enthusiasm was palpable to the students. When the teacher is willing to devote a pedagogical and a Saturday every month—in addition to the bulk of regular class-time—the students understand that she believes what they are doing is important. We feel that the workshop would not be repeatable without the same amount of commitment that Lahache provided.

In the third month of instruction, we began the game design process. Lewis, Fragnito, and Lahache acted as producers while the students chose roles suited to their interests and previous skills. Roles included designers, artists, programmers, or hybrids
such as technical artists. We gave students the choice between modding a virtual environment such as Second Life, or modding in a video game engine, thinking that some students might be more interested in one than the other. The students unanimously decided to create a game mod. Using what they learned in the other modules, the students selected stories and brainstormed how to adapt them to a game environment and gameplay. They were encouraged to stand out as individuals with their skills while maintaining a sense of teamwork and group integrity.

We were encouraged by the degree of participation and intensity of commitment shown by the students. In addition to the normal class hours, they spent considerable amounts of their free time at the workshops. We asked them to undertake a rigorous technical curriculum that included programming, art, design, writing, audio and image production; that demanded both teamwork and leadership from each participant; and that required them to improve their time management, critical thinking, and cultural reflection skills. Though a few students’ participation waned towards the end of the year, a core group of six persisted through the entire process.

The core group of students successfully implemented an Unreal level of the game Otsì!: Rise of the Kanien’kehá:ka Legends [25]. Otsì!: is a rich representation of their story, and, by extension, themselves and their community. The landscape reflects that of the Kahnawake area, and the longhouses in the village are modeled after traditional Iroquois structures. The Flying Head came (Fig. 3), as one of them said: “Straight from the nightmares I had when my Auntie told me stories.”

Anecdotal evidence such as this suggests to us that the workshop met most of its objectives. Stories from the community came alive for the students in both the telling and discussions about them, and, ultimately, in the game itself. They were then able to synthesize their own original story, and furthermore, transform that narrative into a gamespace and gameplay. They learned the technical skills necessary to then implement the game to a point that it was playable (if not entirely finished).

After the Skins Pilot, three of those students joined Lewis’ research lab as Junior Research Assistants while starting their studies at Concordia University. They continued working on the Otsì!: game level created in the workshop, and, along with Obx Lab’s other research assistants, developed it to the state where it won Best New Media at imagineNATIVE Film + Media Arts Festival 2010. Our goal is to offer similar junior research assistantships to the students who participate in the forthcoming Skins Intensive workshop in Summer 2011.

4.3 Skins Intensive
Encouraged by the degree of participation and intensity of commitment shown by the students, we are working on adapting the yearlong Skins Pilot to a two-week workshop called Skins Intensive.

4.3.1 Technology
In the Skins Pilot, we chose to start with Second Life, and, while that environment has several pedagogical advantages (shared workspace, easy object creation and avatar customization, etc.), the students’ healthy ability to absorb instruction indicated that we could start with more complex tools. Since we quickly moved on to the animation software Blender (Fig. 4) in the Skins Pilot, we will begin with Blender in Skins Intensive.
Although using Unreal was successful, we became interested in the game engine Unity3D after seeing it used frequently at the International Game Developers Association’s Global Game Jams—a 48-hour collaborative event where teams rapid prototype games [26]. Further, Unity3D is Mac-friendly and supports mobile game development. AbTeC has been exploring mobile app development and we are very interested in the possibility of focusing the intensive workshop on mobile game development. Thus, we looked closer at comparisons between Unreal and Unity3D with our curriculum needs in mind.

Overall, Unreal and Unity3D compete in terms of rapid prototyping—both engines can produce prototypes at the same pace as long as we have instructors who are comfortable with the technology. However, Unity3D is more flexible for making different genres of games, while Unreal is strongest at making a handful of game genres. Breaking out of Unreal’s mold is difficult, which is an issue when making games based on Aboriginal storytelling. For example, in the pilot workshop, students still had to fit the mold of a First-Person Shooter game and levels. We appreciated Second Life for its sandbox play, but this didn’t translate well to Unreal later in the workshop.

Unity3D’s greatest strengths, relative to Unreal, include: 1) community, 2) documentation, 3) development language, 4) web deployability, and 5) extensibility.

The Unity3D community is vast and provides support as well as many utilities and tools available for free on the web. Unity3D’s user base varies from beginner to expert, educational to commercial, and produces games for platforms from console to mobile. UNITE, a yearly conference, offers presentations, discussions, and bootcamp sessions specific to Unity3D [27].

With such a prominent community, documentation is also in-depth. Resources such as Unity’s official documentation [28], Unity3D Student [29], and Mixamo for Unity animation [30] have extensive written and visual (often video) tutorials as well as forums for community trouble-shooting.

We can choose a preferred development language and also have access to a variety of strong languages. For instance, the Javascript language, a language more known for web development, provides a gentle introduction to programming. For more serious programming, we can use Mono, a language popular for cross-platform development that offers more robust interfaces to Unity3D components. Unity3D also uses the PhysX engine, the physics engine developed by Nvidia, which is very popular and used in many commercial games. Games made with Unity can be easily played online by being deployed using the Unity Plugin, which is as simple to install as Flash.

Further, it is feasible and even recommended to make our own engine or art tools with Unity’s extensibility in its engine and editor [31]. Although this will be more time consuming, we have the opportunity to develop a specific toolset for our curriculum, relying on help from the vast Unity3D production community.

Unity3D’s greatest weaknesses, relative to Unreal, include its 1) graphic prowess and 2) graphic tools. Some graphic tools we will need, such as decals and blend shapes, are not available and require us to write our own toolset or look to community resources. Fortunately, there are many options currently available or in-development that we can carefully select and plan for before the Summer 2011 intensive workshop. Also Unity3D works with Blender, which allows us to retain our existing art and animation lessons.

Based primarily on community and documentation resources, we have decided to change the game engine from Unreal to Unity3D. Fortunately, since there are many existing lessons, we can easily adapt and incorporate resources into our curriculum structure and focus on our workshop dynamics.

4.3.2 Curriculum

In the Skins Pilot, the main challenge we faced had to do with the sheer quantity of content that we needed to cover, coupled with the need—created by the substantial production hours necessary for 3D modeling, animation and level-building—to proceed with all four modules in parallel. We staged the modules so that eventually they were all running in parallel, though our original preference had been to conduct the Play and Storytelling modules first and then move on to the Design and Technical components. The result was that the third through sixth month were probably too heavy, and too fractured between the modules.

We are faced with an event greater time crunch in the two-week intensive workshop (including only around 80 hours of workshop time), but also with an opportunity to adjust the curriculum structure to
match the development pipeline of rapid prototyping. The sessions in the first week will be subject-oriented day by day in the following module order: Play, Storytelling, Design, and Technical. Game concepts will be decided early so that each session contributes assets to the final games as students learn the technology components of game development. The second week includes intense work time for developing the game that walks students through the development pipeline.

There are two major factors to consider that will likely require iterative design of the curriculum during Skins Intensive: 1) the number of total students and teams, and 2) the game genre. Students may want to be divided into more groups either because they want to explore certain roles (e.g. if several students desire to be the lead designer) or they have different game concepts. Genre will be influenced by what excites the students, as well as our interest in exploring mobile gaming platforms.

4.3.3 Workshop Dynamics

Throughout the process, beginning with the first planning meetings two years before the workshop was conducted, a central concern was the issue of what stories—if any—were appropriate for remediation. The genre chosen in the Skins Pilot—a First-Person Shooter—raised even more questions. The most difficult aspect here is that it is virtually impossible to establish who has authority, or even simple legitimacy, to make such decisions. We relied heavily on Lahache, Fragnito (herself from the Kahnawake community), our guest lecturers from other Aboriginal communities, and even the students themselves to make these judgments. In Skins Intensive, each new story will have to undergo similar inspection. Since the workshop will take place at Concordia University in Montreal as opposed to the Survival School in Kahnawake (thus “off-reserve”), we must be especially mindful of protocol. Lahache will again be active in the designing and execution of the workshop, and will represent and advocate the community’s concerns. We also plan to involve guest lecturers such as storytellers from the community.

Depending on the number of students, we are considering dividing the youth into two teams so that two games can be developed in parallel. In this way, students will have more freedom to work on the game design they prefer, but will also experience working in a team structure with flexible but identified roles, including variants of designer, writer, artist, animator, programmer, sound, and producer.

A long-term goal is to offer college credit for such workshops through Concordia University. This simultaneously offers a direct payoff for participating in the workshop, and introduces students to how the university environment operates. We are conducting entrance and exit interviews with students to investigate their perceptions of higher education in order to understand their educational and career aspirations before and after the workshop.

Since the workshop takes place in Montreal, we are providing transportation for the students for the 20-minute ride from Kahnawake to Concordia University. Skins Intensive, unlike the Skins Pilot, has to include travel time and for settling in after arriving at Concordia. However, since we do not need to fit into their secondary school hours, we are able to include several evening events for team building and debriefing after long workshop days.

5. FUTURE WORK

Skins Intensive will be the site of a much more in-depth data collection and analysis effort. The Skins Pilot resulted in useful but limited findings in regards to the perception of the curriculum, the technology, and higher education and/or professional choices [32]. We need to obtain concrete data on these aspects in order to fully validate the curriculum. To that end, we will interview youth about the curriculum as the workshop progresses, as well as conduct entrance and exit interviews to verify their experience and determine their perspectives about technology and higher education and identify any changes.

We will also conduct extensive videotaping and ethnographic observation. The students will be employing Photovoice whereby they take photos to capture their experience, perspective, and concerns, and then write captions for the images explaining what the images represent and why they are important [33]. All of this will be done in a context that integrates indigenous and Western methodologies for understanding knowledge acquisition and cultural context [1].

Following the intensive workshop we hope to find other Aboriginal communities who wish to try the workshop. The curriculum is available for free from the AbTeC website [24], and can be reconfigured so that even communities which may face technical resource challenges can still conduct an interesting
workshop with only the Play, Storytelling, and Game Design components. The intensive version of the curriculum will also be made available for free at the AbTeC website when it is completed.

6. ACKNOWLEDGEMENTS

We would like to thank the Social Sciences and Humanities Research Council, the Hexagram Research Institute, and Office of Research at Concordia University for their support. A special thanks is given to Owisokon Lahache and the Kahawake Survival School administration for hosting the Skins Pilot workshop. We look forward to the Skins Intensive in 2011. We also wish to thank the full AbTeC research team for their invaluable assistance in preparing the curriculum.

7. REFERENCES


