ABSTRACT

In the past 3-4 years there has been a significant interest in computer games in University and college curricula, as a way to teach early computer science, to attract more students into the program, and to teach advanced concepts and lend vocational weight to a curriculum. In this article we discuss many ways that games can contribute to an undergraduate CS program, and illustrates specific ways that the use of games has influenced the students, the faculty, and the institution. Our claims are supported by numbers based on actual observation and study. We also show how the inclusion of games can add to research aspects and the reputation of a computer science department.

Categories and Subject Descriptors

K.3.2: Computer and Information Science Education - computer science education, curricula, information systems education. **J.m MISCELLANEOUS.**

General Terms

Algorithms, Management, Measurement, Design.

Keywords

Computer science curriculum, education, pedagogy.

1. INTRODUCTION

Although courses and programs related to computer games and video game programming and design have been taught for a decade or so, it is only relatively recently that there in any significant interest from Universities. The perception used to be that many of the skills needed to construct modern games are vocational ones. However, the synergistic effect of integrating a large number of diverse computer science topics/concepts into one framework, the inclusion of pertinent portions from Fine Arts disciplines, the software engineering practice that accrues from the group effort to build a non-trivial game, and the potential positive recruiting effect to help with enrolment deficits are strong positive arguments that are being recognized by administrators and teachers.

When a course on games programming/design is proposed for a university program, there can be a significant degree of opposition from faculty. It is true that a game programming course does not teach a significant new set of computing skills that cannot be taught in more traditional graphics, algorithms, and data structures classes. However, a game is a very complex piece of software that combines advanced concepts from a wide variety of areas within CS, and the study of game programming and development allows each individual component to be placed in a very practical context. In addition, students appear to be strongly motivated by the subject material, and spend a great deal of time working on their assignments.

This paper will describe four specific aspects of the use of games in the undergraduate CS curriculum and their impact on the students, faculty, and school. The use of game assignments in programming courses even in the first year will be discussed, and evidence will be given that students work harder and perform better when games are used in that context. The design and implementation of a senior CS course in game programming, the capstone course of a concentration in game development, will be described, along with some observed results. We'll also discuss the incorporation of arts disciplines into the CS program, and look at how success in these enterprises has led to a successful research program in game technology. This program often uses undergraduate students as key players, doing programming, design, acting as subjects, and co- authoring research papers.

1. GAMES AS ASSIGNMENTS - YEAR 1

Casual polls of students enrolled in the introductory computer science class at the institution where the authors teach indicate that two

thirds to three quarters of all freshmen in computer science became interested in the subject because they play computer and video games. Current wisdom implies that learning is most effective when we build on what the learner already knows, and using situations they are familiar with [18, 19]. Students understand games far better than they understand employee records keeping, and 'widget' manufacturing, both of which are sources of favorite entry-level programming problems.

For these reasons, computer games have been used as assignments in the first year programming courses in Calgary since the 1990's. We've been using classic arcade games with great success because students understand the problem, classic games make use of older and simple technology, and they are recognizable cultural objects for that generation []. However, rather than just discuss the utility of games for teaching programming in the abstract, we have some actual quantitative data.

0.1 An Opportunistic Quantitative Study

In 2002 an opportunity arose that permitted us to measure the effect that game assignments have on the students. A last minute change of policy had the first year classes diverge in their assignments sets, one set being game based to some extent and the other set being a fairly standard set in the behaviorist tradition. The assignments were handed in using an on-line system, so we had access to all of the source code.

Our intention was to use standard software engineering complexity metrics to determine how complex the student solutions were, and to compare this between the two distinct assignment sets, which were in fact two concurrent lecture sections. There appears to be very little previous work [][] that involves a complexity analysis of assignments in the way that is being done here. The method proposed is to use perceived effort as a measure of success. Assuming that the assignment grades in both classes are more or less equivalent, as determined by review of the assignments by experienced teachers, then the assignments that involved the largest amount of work or effort should be an indicator of how much was learned [][].

The actual assignments, beginning with the standard set are:

1. Read percentage grades and print corresponding letter grades.
2. Create a class that represents a point in a two dimensional Cartesian coordinate system.
3. Create subclasses **shape**, **rectangle**, **circle**, and **test** from the **point** class defined in assignment 2.
4. Swing-based mortgage calculator.
5. Simulate a greenhouse. Has sensors and effectors, uses threads and a simple GUI.

The Game-based assignment set is:

1. An implementation of a simple calculator.
2. First class - integrate a *BigNum* class into the calculator.
3. **Write an ACSII-graphics version of the Four Seasons Solitaire game.**
4. Design and write a recursive parser for expressions.
5. **Design and implement an ASCII-graphics version of the Centipede arcade game.**

To assess the student assignments for complexity, we used the commonly encountered *Halstead* metrics [], precisely because they are commonly encountered and understood. We computed select metrics automatically from the code submitted and calculated an average for each group. The values selected for calculation were: Number of Tokens (N), Vocabulary (v), Length (L), Lines of Code (LOC), Effort (E), Time to Code (TC).

We also computed the *cyclomatic complexity* (CC) []. The results, tabulated in Table 1, show a significant advantage to the game-based assignment set over the other in terms of every complexity measure.

Table 1: Assignment Metrics

	Set A	Set B (games)	Set A: 3&5 only	Set B 3&5 only
N	369	481	412	636
V	315	534	301	648
L	2556	4846	2410	5978
LOC	117	153	131	203
E	13,623	24,910	15,321	34,740
TC	4.2	7.7	9.5	10.7
CC	6.84	4.75	1.70	4.18

The average values for the actual game assignments are computed too so that they can be compared directly against the corresponding non-game assignments at the same point in the course.

Conclusion: *In all cases the game assignments are more complex and require more effort than the other assignments, and this is amplified when only the two game assignments are considered.* Our observation was that the students were quite willing to spend the extra effort required by the more complex game assignments. Following up the students in successive courses showed a tendency for the students who did the game assignments to get slightly higher GPAs in second year courses, including theory. The students in the game group achieved an average grade 0.7 higher than did the behaviorist group in the successor programming class;

Grades are computed on a 4-point scale, so this would represent an 18% improvement. This improvement deteriorates with time if the instructional methods are not continued. Taking the successor course 4 months later reduces the improvement to 0.1 from 0.7, or 2.5%. Improvement is higher for the 'A' students []

0A GAME PROGRAMMING COURSE

The University of Calgary has offered a course on computer game programming since January, 2000. We have an existing body of graduates, connections to industry, and the beginnings of a research program to show for our efforts, as well as whatever reputation one gains from being the first.

The game programming course at the is a regularly timetabled fourth year class. It is taught in the Winter term (second semester). This course has been designed and is taught largely by staff from a successful professional game development company - Radical Entertainment of Vancouver BC. Because there is a thousand kilometers between Vancouver and Calgary, and because Radical could obviously not afford to lose valuable staff for so long a period, all of the lectures are provided during one single week, 7 hours per day, at the beginning of January before regular classes begin. This has the advantage that all essential lecture material is complete before the students begin their project.

During the lecture periods the students are organized into groups of five, each group being responsible for creating one game as a course project. We have chosen a driving or racing game as a project each year for many good reasons. First, such a game has the basic structure of most video games sold today in terms of graphics, sound, and play. It will be fun for the students to play the game resulting from their project. Next, the graphics can be simple while maintaining playability. There’s no real need for hard to render objects like trees and animals, and no need at all for character animation, which can be quite difficult. Finally, in spite of those simplifying aspects, a driving game involves some quite difficult physics, and is sufficient challenge for a single semester.

> The students have time during the first week of instruction to create a basic concept design for their game, and to discuss it with the professionals from Radical. After that a regularly scheduled lab occurs each week during the semester, and students with problems can address them either there or with the instructor. Further design documents will be supplied on a timetable and assessed, giving the students feedback. A news group is created for the course, and students can communicate with Radical or the

local instructor relatively quickly.

Students work on a game of their design in a small ‘game studio’ containing a few high-end PCs and the software they need for game creation: 3D modelers, an audio editor, compiler, and so on. We use Maya for modelling and have access to the Viewpoint 3D object model library; we also have audio software (Sound Forge) and a collection of effects, and recently have allowed them to use the ODE physics engine. The instructors take on the role of publisher, essentially contracting out the game to the groups of five students. There is a random visit by the publisher, where each group is given 24 hours notice that a live demo will be required. The idea is to model the actual situation faced by game development teams.

When the course is complete, the students will have a valuable addition to their portfolio - a complete game. This course is rated quite highly by the students when they conduct their assessments, with numerical values between 6.1 and 6.9 on a 7 point scale. Moreover, students are more pleased overall with this course than with others, spending time on their work voluntarily. One or two students find jobs each year based on this course, and many are still in the game industry. Discussions with the Faculty of Science Recruitment Officer have revealed that about 30% of students admitted to CS during his four year tenure came to our school because of the game course and./or program.

0.1 An Example Project - MiniMayhem

The MiniMayhem game was one of the first and one of the best projects in the course so far. It is a combat driving game: the cars are equipped with missiles and mines, which can be picked up by driving through a rotating floating cube (Figure). Game play consists of attempting to destroy your opponents, three cars controlled very effectively by the game’s AI module.

Action takes place on what looks like the floor of a family room. Cars are about 1-2 inches high, and the floor is littered with toys, books, and Lego blocks these serve as both obstacles and places to hide. There is a very modern sound track, good sound effects, and multiple camera positions. Physics is good, not excellent, but the game allows the player to adjust things like the friction on surfaces and the accelerations of the vehicles. The game is wickedly difficult to beat, but huge fun. One of the game’s builders moved immediately to a Vancouver game developer upon graduation.

FIGURE 1. A screenshot from the student created game MiniMayhem.

0.1 Objectives and Outcomes

Many computer science departments specify their courses, and therefore their entire programs, as a list of topics to be covered. This is usually referred to as *coverage*, and is a less than optimal approach because it is a *content standard*, and specifies inputs; that is, what the content is that should be covered. *Performance standards* specify the desired *output*, such as what the students must do, and how they demonstrate success []. One alternative to coverage, and one that has many advantages, is to specify the *objectives and outcomes* of each course [][]. Objectives guide the activity, lesson, or unit of instruction, and are directly related to assessment and evaluation. Generically, objectives are goals, and it is plain that there are many possible kinds and levels of goals.

We have specified objectives for real-time rendering, simulation, animation, human/computer interfaces, artificial intelligence, software design, real-time systems, information retrieval, parallel and distributed programming, audio/signal processing, networking, and elements of algorithm analysis []. There is no other course of which we are aware that includes this very diverse set of concepts, and which has such a disparate set of outcomes. Of course, not all game programming courses will include all of these outcomes - each course must be designed to fit into the program and system environment that is available. A complete list

of the objectives and outcomes can be found in [], but an example is:

> *Upon completion of the course, the student should be able to explain what real-time scheduling algorithms do, and how they enable the response time requirements of real-time systems to be achieved; demonstrate an understanding of their design and implementation issues.*

0COMPUTER SCIENCE AND ARTS

Due to the variety intrinsic with the artistic experience, and the quite distinct natures of the different artistic media, computer science has not traditionally been a cooperative or understanding partner of the artist, and in turn artists have been hesitant and awkward about adopting digital media as a part of their daily work. Computer scientists tend to be more closely aligned with mathematics than with acting, set design, or painting (although a significant fraction of them either draw or play and instrument), and are more accustomed to processing numerical data than drawings and music. Effective joint efforts between artists and computer scientists require a comprehension of the opportunities and problems in both sets of disciplines, and video game projects offer a unique opportunity to present the students with a working context (a *situated* environment).

Our practical experience is that music students are easier to attract into game collaborations than are art or drama students, possibly because musicians see a clearer career path. We have had music students involved in three different groups, always with success. The music students not only perform music but compose new music for the game. They are often registered in a project course in composition in the music department, and are not members of the game class per se. This kind of collaboration between departments rarely requires external approval, and so can be arranged by the instructors themselves.

The collaborations so far have results in students learning about these difficult interactions, in better game audio, and in explicit collaborations between the computer science and the music department on research activities.

0.1 A Game Class in Fine Arts

We designed a senior seminar/studio course jointly by professors from the Computer Science and Drama departments. This permitted us to offer game related material from a different perspective. The class consisted of seven students, a mix of computer science students fulfilling an option requirement, and drama students (design) with a technical orientation. In addition, there were students at Carleton University in Ottawa working on a similar project, and the two groups remained in touch through teleconferences. We were also affiliated with the Banff Centre through the New Media Institute, who provided information, facilities, and seminars for the students [][].

The idea was discussed first in a research group, which was attempting to deliver high school physics programs to remote rural communities using the internet. The use of interactive computer games as one mode of delivery was proposed, and the course was born. The class decided on a role-playing game (RPG) that could be extended to cover the entire high school physics curriculum by using multiple levels.

0.1 Results of the Arts Collaborations

In the short term, two of the students from the fine arts game class ended up as game developers for the Banff Centre for the Arts, creating educations games for a funded research project. Effective joint efforts between artists and computer scientists require a comprehension of the opportunities and problems in both sets of disciplines. We are suggesting that should offer computer science students the opportunity to work with artists in artistic endeavors at a relatively early stage in their careers. We will, in this way, create a new kind of graduate, one possessing an amalgam of the skill sets and an appreciation of the working styles from both fields. Working together on a large multimedia project like a game gives them an understanding of how the two distinct groups can work together effectively. Where this may lead is impossible to tell, except that it opens a range of possibilities that can't be completely anticipated.

0GAME-RELATED RESEARCH

The Digital Media Laboratory (DML), currently situated within the Computer Science department at the University of Calgary, has been the focus of digital game based research at the that school. Statistics will tell an interesting tale - since 2000, the genesis of the undergraduate course and games concentration, the DML has published 34 scholarly works on the subject of computer games. This includes papers, books, panels, keynotes - all aspects of scholarly activity. Of these, 18 (or 53%) had the explicit participation of an undergraduate student, and 8 (or 24%) listed at least one undergraduate as a co-author (E.G. []). This is a very significant contribution by students, and the student population becomes aware of the possibilities for research from their peers.

This research has been the basis for external and internal funding as well, to a total of between $50-$100 thousand dollars, depending on exactly what is counted.

FIGURE 2. A screen capture from the OceanQuest game, externally funded and built by students.

An excellent example of funded research using undergraduates was the OceanQuest game project [], funded by SciQ (http://www.sciq.ca/oceanquest/qzine.html) to the tune of $15,000. The idea was to devise a game that would be used to teach grade 7-8 students about ocean floor ecology, in particular the volcanic vents on the ocean floor (Figure).

The game was designed and built by the authors and students in the computer game programming course in a period of about six months. It is a 3D game with sound in which the player pilots a submarine to the vents to collect biological samples. The students worked for two months before the course began during which time they were on a salary, and an artist was paid to develop models and a trailer. When the class began, the students finished the project for course credit alone.

The game was a success, and was unique in that we developed an additional Flash version of the same game for online deployment. It was updated a year later by another student, and was the subject of a paper in the Digra conference in 2005 [].

0CONCLUSIONS

We are in a position to offer some quite specific conclusions on the impact of the use of games in the curriculum, especially game programming classes. First is the recruitment potential - the roughly 1/3 of our first year class that has been attracted by the game courses and concentration is quite significant. The fact that we have shown, at least in a preliminary fashion, that game