Game Construction as a Learning Tool

Shri Rai
School of Information Technology
Murdoch University
Western Australia
(08) 9360 6090
s.rai@murdoch.edu.au

Kok Wai Wong
School of Information Technology
Murdoch University
Western Australia
(08) 9360 6100
k.wong@murdoch.edu.au

Peter Cole
School of Information Technology
Murdoch University
Western Australia
(08) 9360 2918
p.cole@murdoch.edu.au

ABSTRACT
The conventional way of teaching IT is increasingly not attractive to students of today. For example, covering the various topics in Computer Science is not exactly exciting when students are taught the theory and then asked to practice by coming up with solutions to "made-up" problems. These solutions then get discarded when they move on to a new topic. A computer game can involve most and possibly all topics in Computer Science. When working on computer games, instead of discarding solutions that students worked on earlier, students learn to incorporate solutions to previous problems into newer solutions. The final product is a lot more sophisticated and students can actually feel proud of what they have achieved. They can also have enjoyed themselves along the way. They would have learnt that topics they have studied had a purpose. This is useful, as quite often, students cannot figure out why they need to study a particular topic, especially if they are finding it hard. In this paper, we suggest that game construction can be an effective learning tool.

Categories and Subject Descriptors
K.3.1 [Computer Uses in Education]: Computer-assisted instruction (CAI).

K.3.2 [Computer and Information Science Education]: Computer science education.

General Terms
Experimentation.

Keywords
Computer Games, algorithm animation, algorithm visualization, learning Computer Science, software construction

1. INTRODUCTION
The computer games industry is the fastest growing sector in Information Technology. According to the Game Developers Association of Australia (GDAA), an estimated amount of $A40.9 billion dollars of interactive video games were sold in 2002 [7]. This figure is expected to rise above $50 billion.

Games are not just used for entertainment but along with games used for education and training, games are used in rehabilitation, strategy development, "what-if" scenarios, public policy development, warfare simulation, etc. Such games are called "Serious Games" [16, 17, 19]. We show in this paper that playing a game is not the only way to learn. Learning can also happen when attempting to build a game.

Like other courses, Computer Science (CS) courses (or units) have assignments and projects to test students' understanding of the subject matter. Traditionally, CS assignments and projects are very often "made-up" to test only the material that students have just learnt. Once the students have completed the assignments or projects (which very often involves programming), the code that the students have created are not used again. As far as the students are concerned, they have completed the assignment or project and they give no real thought to what lessons have been learned – other than they got their assignment or project to work.

Our experience shows that the demands placed on Games Technology and Simulation (GTS) students are significantly higher than conventional CS students. In spite of the short time frames available, students enrolled in GTS are consistently producing better quality software that is more complex. The approach taken to teach our GTS students is not the same as the traditional approach. We spent some time designing our games courses to fit in better with what we wanted our students to achieve. This paper also describes the design of the first games unit that students have to take as part of their degree. Students enrolled in this unit are in the second semester of their second year of a three year Bachelor of Science degree.

There are a number of topic areas which are normally studied by students studying towards a degree in CS. These would typically include:

- Data encoding and storage (e.g. bits, bytes, number and text representation, binary operations, storage devices)
- Machine architecture and machine language (e.g. CPU, machine cycle, device/peripheral communication, assembly language)
- Operating Systems (e.g. OS architecture, resource management – memory, process, .. etc., security)
- Networking, Internet and the Web (e.g. topologies, protocols, applications, security)
- Algorithms (e.g. problem solving, control constructs, algorithm representation, efficiency)
- Data Abstractions (e.g. lists, trees, .. etc., object orientation)
- Data Storage and data retrieval (e.g. files, databases)
- Programming Languages (e.g. various languages with: programming constructs – variables, constants, parameters, data types, assignment and control
System, testing, documentation)
• Artificial Intelligence (e.g. autonomous machines, various techniques like state machines, Artificial Neural Networks (ANNs), Fuzzy logic, Genetic Algorithms, Expert Systems, ..etc)
• Theory of Computation (e.g. computability, complexity)
• Computer Graphics

GTS students need to know all of the above as well as:
• Mathematics including some Calculus
• Physics where they need to understand and implement laws of conservation of energy and momentum;
• Art – the artwork needed for their characters and their scenes
• Audio – the physics of sound as well as the manipulation and management of sound resources
• Animation
• Story-telling – including how narratives are structured and techniques for getting emotional involvement
• Artificial Intelligence (AI) has increased coverage, e.g.
  o Exhibition of emotional and other behaviours
  o Chasing and evading behaviours which get exhibited by Non-Player Characters (NPCs)
  o Path finding and navigation by NPCs
  o The use of Fuzzy Cognitive maps and Affordance theory to model the behaviour of environments.

GTS students also need to be aware that games usually serve some purpose. For example, people playing computer games also pick up some skills that are not necessarily computer related. These need to be taken into account when developing games. Diablo requires skills like mapping, understanding risk/benefit and trade offs. Playing “The Sims” properly leads to an understanding that aesthetics matter and players need to understand how to live with social skills. Quake/Unreal teaches 3D spatial awareness, world modelling and prediction. Chess players need to understand forward planning, minimisation of risks and maximising of opportunities [20].

GTS students also need to be aware that they would be working in teams. However, unlike conventional teams working on Information Technology projects, games teams have members from various backgrounds and educational levels. Some team members may not even have a degree. Examples of some team components/members include programmers, game play designers, multi-play designers, storyboard developers, level designers, graphics artists, graphics designers, animators, networks, tool designers, AI, audio designers, dialog creators, voice talent, play testers, marketing, technical writers [20].

GTS majors would have only completed one and one-half years of their three-year degree before they start on their first games unit. As they are only half way through their degree programme, not all the CS topics have been covered. Therefore, certain topics are covered in the first games unit, as these topics are needed for the students to be able to build anything useful. This coverage is done “just-in-time”.

Students starting the first games unit would have been exposed to Object-orientation and algorithms in earlier units. Unfortunately, this exposure is not sufficient to build anything substantial as the students are still grappling with the concepts that they had learnt previously. We take the approach of Huni and Metz [9] who have demonstrated the usefulness of the “just-in-time” approach and others (e.g. [18], [11], and especially [6]) who have studied the issues associated with algorithm animation and visualisation as a teaching tool. This will be discussed later.

The next section provides an overview of the design of our first games unit called Games Design and Programming (GDP). It is included to show what students need to achieve and in particular to emphasise that students need to learn about computer algorithms (amongst other things). Section 3 presents the results of our approach. Section 4 discusses why our approach is more in line with previous findings on the issues associated with visualisation tools for understanding algorithms.

2. GAMES DESIGN AND PROGRAMMING STRUCTURE

The Games Design and Programming (GDP) unit is a core unit in the Bachelor of Science (Games Technology) degree. It is also a pre-requisite to subsequent units in the Games Technology degree. GDP covers Computer Graphics, Mathematics, Newtonian Physics, Artificial Intelligence and elements of media studies. All these are indicated in the IGDA’s Curriculum Framework [10]. Along with GDP, students also need separate units called Intelligent Systems and Computer Graphics Principles and Programming for the subsequent advanced units in the Games Technology degree program. It was felt that from the very first games related unit in the degree program, students would have to develop games as part of their final project. This requirement determined the content of the Games Design and Programming unit. The final description of the unit in the handbook is stated below:

“This unit covers introduction to games design and programming from a theoretical and practical points of view. The unit includes game theory, data structures, and play testing, as well as the physics of games programming. Further work covers non-linearity, level-design and intelligent agents. Practical rules for writing games programs are developed and applied.”

The declared aims of GDP are [15]:

• To introduce students to the field of Games Design and Programming and to appreciate the multidisciplinary nature of this field;
To introduce students to the essential concepts and techniques through practice work based on developing programs that create interactive visual imagery;

- To get students to acquire independent self-learning skills.

The listed learning objectives of the unit are [15]:

- To find out about current applications of computer graphics in games;
- To learn about the techniques and algorithms used for developing games applications involving both 2D and 3D objects;
- To learn the essential theory behind games design;
- To be able to design and implement simple computer games in C/C++ including the use of library functions from various APIs (Application Programmer’s Interface);
- To acquire some ability to extend current skills unaided.

Students enrolling in GDP would have done two CS units at the first year level. This would have given them exposure to the Java programming language. C++ is used in GDP as the implementation language. Like other units, GDP was taught in a thirteen-week semester. The table below shows the list of topics covered.

<table>
<thead>
<tr>
<th>Topic No.</th>
<th>Topic Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Games in Context</td>
</tr>
<tr>
<td>2</td>
<td>History of Games</td>
</tr>
<tr>
<td>3</td>
<td>Game Graphics</td>
</tr>
<tr>
<td>4</td>
<td>Game Design</td>
</tr>
<tr>
<td>5</td>
<td>Virtual Worlds</td>
</tr>
<tr>
<td>6</td>
<td>Game Physics/Mathematics</td>
</tr>
<tr>
<td>7</td>
<td>Game AI</td>
</tr>
<tr>
<td>8</td>
<td>Game Engines</td>
</tr>
<tr>
<td>9</td>
<td>Network Games</td>
</tr>
</tbody>
</table>

Staff with various backgrounds was called on to teach the unit. Staff specialising in literary and media art forms taught “Games in Context”, “History of Games” and part of Game Design where narrative and narrative structures are examined. A person from the games industry (game developer) was employed to teach the rest of “Games Design” as well as “Virtual Worlds, “Game Engines” and “Network Games”. Staff with a Computer Science background taught “Game Graphics”, “Game AI”, and “Game Physics/Mathematics”. The finer points of object orientation using C++ were covered on demand – when students needed to know so that they could do the implementation. As it can be seen, the topics are interdisciplinary and therefore the staff teaching the unit came from different disciplines.

The practical work that student did was a mix of individual work as well as group work. The individual assessments were needed to ensure that each student would have the necessary skills. Group work was encouraged as students needed to learn how to work co-operatively and therefore acquire group-work skills. This group-work was needed because, as indicated earlier, games are not designed and built by individuals but teams of people. Although group-work was strongly encouraged when students were working on the projects and assignment, group-work was not made mandatory.

Assessment in the unit was such that to pass the unit, students needed to have an understanding of what was taught in the unit. However, to get distinction grades, students needed to go beyond what was covered in the unit. They had to find out additional tools and techniques to use to build their games.

3. RESULTS

Students who completed the practical work and sat the exam passed the unit. The number of Distinction and High Distinction grades awarded was very high. This was not expected when the unit was designed. The results, however, showed that students were able to demonstrate understanding of the subject matter. As an example, Figure 1 shows part of the design of one of the robots in a game called “Robowar”. The object of this game is to destroy the robots guarding a city and capture the city. The AI in the robot enables the robots to chase the player if the player comes too close. Students implemented the AI that they had recently learned. It was pretty obvious which AI technique actually worked well when the NPCs were moving around. To get the NPCs to behave more intelligently, students had to analyse their AI generating algorithms in a lot of detail. After all, it was the AI in their own NPC that would look silly if they did not get the AI to work correctly. The end result of all this was that they got to understand the AI techniques a lot better than they would have if they studied the AI techniques the conventional way. In fact, GTS students’ understanding of AI was a lot better than CS students.

In addition to the 3D environment and graphics, the students have put in a fair amount of effort in designing the robots. As an example, the following is some of the description (including some backstory):

“The Prototype Jofus Mk1 (PJM1) was initially designed as an all-purpose – all-terrain construction robot. However the military saw its potential due to its heavy metal plating and sturdiness. They took the original blueprints and modified it in secrecy. Before the Virus struck, initial tests showed that the PJM1 was airtight with its own air supply and with enough metal plating was able to withstand radiation and airborne dangers. After the Virus had struck the population, our protagonist wakes up in the military bases hospital ward with no recollection of who and where he/she was. He/she finds the PJM1 and discovers that he is able to pilot it. Later on he/she discovers that he/she was part of the tests which involved the use of biological enhancers injected into him/her so that his/her strength and reflexes were improved to make him/her able to pilot the PJM1.”
The students understood that they could not build an NPC that is omnipotent in a game, as this would enable only one outcome for each game. Instead, this robot had the following characteristics that were trade-offs: “The fastest of all bot types; lowest health of all bot types; lowest armour of all bot types; weakest weapons but fast firing; short to medium range attacks”. Explosions in the game were built using particle systems.

Figures 2 and 3 are screen shots of an assignment which involved construction of virtual tours. Figure 2 shows a person inside the library looking out. The interactive tour starts at the car park and ends in the library. Participants in the tour can move around in any of the public areas of the North wing of the library. All the books, shelves, pillars, bricks, lights, benches, etc have been implemented as objects in C++. Many of these were developed for use in weekly exercises. The C++ classes were then incorporated into the assignments and projects. The re-use of classes and frameworks was emphasised. As the size of the software projects was large, students would not have been able to complete construction without designing abstract classes that could be re-used in many situations. A number of students understood this quite quickly when they were doing the preliminary work for the project and the scale of the project dawned on them. Others learnt the hard way. At the end of the project, all students were “converts”. They had first hand knowledge of what it would be like if they did not get their object orientation correct early.

Although not immediately apparent by just looking at figure 3, the models are to scale. Even the number of bricks, number of rafters, size of the rafters and size and placement of the concrete benches are accurate. The models were built after considerable time spent collecting data about size and placements. Because of the level of detail, polygon counts were high. This would have meant that the virtual tours would run very slowly. To resolve this problem, schemes had to be implemented where the polygons were only drawn if they were within view of the virtual camera.

4. Discussion and Conclusion
The pre-requisite knowledge required for GDP was not sufficient preparation to design and write the object oriented (OO) programs and algorithms needed. Most of the important aspects of OO had to be taught when it was needed.

There are various reports in the literature that OO concepts and algorithms are not easy to teach. Beck and Cunningham report on issues dealing with teaching OO concepts [1]. They reported success teaching object oriented concepts to experienced and inexperienced procedural programmers by getting them to participate in role-playing exercises using cards which represent objects. Brown developed an algorithm animation technique [5]. He pointed out that visual display of algorithms can aid teaching in various areas of CS - data structures and algorithms as well as programming.

A number of other studies later showed that algorithm animation or visualisation was not as useful for learning about algorithms as
previously believed. An empirical study by Stasko, Badre, and Lewis [18] suggest such animations would benefit more "advanced students" but have not much value for beginning students. They advise that for algorithm animations to be useful, students should have tools to construct their own algorithm animations. Jarc and Feldman’s [11] empirical study, also, could not find significant differences in performance between those who used interactive algorithm animation and visualisation and those who did not. Byrne, Catrambone and Stasko [6] were also unable to convince themselves that algorithm animation would help students learn algorithms. But they made the interesting observation that students may have been learning by predicting the algorithms behaviour and this could be done using "static means". We have also observed that when students actually engage with the algorithms manually, they get to understand the algorithm.

Hundhausen [8] report similar results, and in particular, support findings by Stasko et al. (1993). Hundhausen notes that conventional algorithm visualisation software "can actually distract students from concentrating on activities and concepts relevant to an undergraduate algorithms course". But if students construct their own animations, they are able to focus on the course and learn.

Mulholland [13] conducted a more detailed study of the effectiveness of visualisation tools for learning. He showed that the misunderstanding of what the visualisation tool was doing contributed to the tools ineffectiveness. Recommendations were then made as to how this misunderstanding could be minimised.

What the researchers are showing is that understanding algorithms (and other difficult concepts) cannot be done passively. The learner has to actively engage with these concepts. Other studies do point out that "active engagement on the part of students leads to higher motivation and better integration and retention of content. Some visualization tools lend themselves to active student engagement more readily than others." [3]. We argue that the most amount of engagement is obtained when students have to implement the algorithms as part of a game and can visually observe the performance of their algorithms.

Our experience suggests that it is more difficult to learn concepts in isolation. When concepts are learnt separate from their actual application, students very often do not see the point of learning these concepts even when they are told what the applications are. Games are the best vehicles for applying these concepts. Students get to see the application of the concepts in action. If a search algorithm is not efficient, students immediately get a feel for the problem as they can see the problem occurring before their eyes. They know immediately if the AI that they built into their Non-Player Character (NPC) is actually exhibiting "intelligence". It is not a "toy" problem anymore. They also get to know which AI technique gives more intelligence to their NPCs. Students also immediately see the potential for reuse of useful algorithms and solutions. Well tested working code classes do not get discarded but instead are reused. We have actually seen this happening – students creating their own code libraries and reusing them. We have not observed this desirable behaviour in CS classes. Our findings in these regard support what Huni and Metz [9] reported.

Many others have reported the usefulness of appropriate games (and simulations) for learning [2, 4, 12, 14, 21]. The nature of games permits active engagement with the content and increased enthusiasm levels. Learners can explore and experiment. Designing and building a game, can therefore, be a very useful way towards a more thorough understanding of the algorithms, data structures and other topics in Computer Science.

5. ACKNOWLEDGMENTS
Our thanks to ACM SIGCHI for allowing us to modify templates they had developed.

6. REFERENCES


