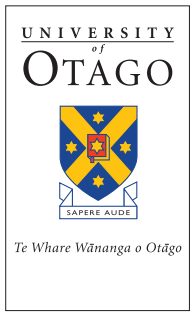
Otto-von-Guericke-University Magdeburg
University of Otago, Dunedin
Department of Computer Science



Research Report

Co-Development, Delivery and Structural Analysis of a Computer Game Course

Lennart Nacke

Supervisors:

Professor Dr Maic Masuch (Otto-von-Guericke-University Magdeburg),
Mr Simon McCallum (University of Otago)

Nacke, Lennart:
Co-Development, Delivery and Structural Analysis of a Computer Game Course
Research Report,
Otto-von-Guericke-University
Magdeburg, June 2004.

Internship Task

Computer game design is still a young, growing subject in computer science education. Due to the industry's need for qualified personnel, who appreciate the complexities of computer game design, many new academic courses are being created.
As computer game design is a new topic there is still much research required into pedagogy and delivery. The focus of this internship is to assist in the development and delivery of an undergraduate course on computer game design. There are three phases to this internship:

1. Development
   This includes the research on suitable game engines, taking into account the limited budget available for the course. It also includes

   - the analysis of proposed lecture topics corresponding to the IGDA Curriculum Framework [IGD03]
   - research and study on software engineering topics, such as version control systems like CVS and documentation with LaTeX

2. Delivery
   Including assistance in lectures for the course as well as tutoring. To further help the students we also include:

   - an introduction to interactive fiction using the Inform environment and construction of the first assignment
   - the creation and maintenance of an interactive website accompanying the course, which includes a student feedback system, a forum and a maintenance system for lecture topics and textbook errata

3. Analysis
   Reporting on the success and/or failure of the course. (A post-mortem in game design terminology)

The three phases are meant to be carried out in a chronological order given the following timeframe.

Timeframe September 2003 - February 2004

Preamble

Practical semesters or internships for computer science/computational visualistics students generally involve, on a large scale, research on classical topics like simulation, algorithmic geometry or information systems. There is no doubt that these fields of study provide an interesting area of research. What about modern and more popular areas of computing? What about complex software products that are mainly designed for children and young adults? - What about computer games?
Many people chuckle when they hear about scientists conducting research on their children's favourite pastime. Nevertheless this new field of research is getting more attention globally and as it rises, so does the need for a field-tested curriculum.
This report will show and analyse the composition of such a curriculum for a computer game design course taught to computer science students at the University of Otago in New Zealand.

Why Should You Read This?

If you are part of this paper's target audience, you are most likely to be interested in computer games. Since I will focus on the teaching of this subject you might gain interesting insights into the area by looking at the tools we evaluated for use in education. Or you might benefit from reading our attempts and the lessons which we learned. Even if you are not part of the target audience you will hopefully find this report an interesting review of the research required to create a computer game course.

How This Report is Structured

This report is divided into three parts - Development, Delivery and Analysis. Part one will look at the process of developing a computer game curriculum. Chapter 1 will give an overview of computer games and the approach to the teaching of this subject in academia: which aspects are important and how you can tailor the general curriculum to your audience. The next two Chapters will show the efforts and pre-considerations made at the University of Otago for teaching game design in the available format and for the expected audience. Chapter 4 extends this by looking at a diversity of tools that can be employed for teaching computer game concepts to university students.
The second part will elucidate how the considerations described in part one were used in the delivery of the computer game course. It shows the technical realisation of a course assistance system and gives an overview of the teaching topics used in the course.
Lastly, the third part gives an analysis of the course and its creation process. This features a post-mortem and personal remarks of the author. It also gives an outlook into the future and the development of New Zealand's first game developers conference.
The Appendices include an overview of the games that were created during the course, some of the author's assignments for the laboratories and some code.

Acknowledgements

Many thanks to all the wonderful people who contributed to the success of my internship and this report. Mr Simon McCallum and his teaching team (especially Mr Jayson Mackie and Mr Tim Nixon) made me feel comfortable in New Zealand instantly. Their academic help, their criticism and the inspiring discussions I had with them, made it a pleasure to work at the computer science department of the University of Otago. I am pleased to say, that I have not only gained a colleague of international calibre but also a close friend in Mr Simon McCallum.
Lots of thanks to Professor Dr Maic Masuch for his supervision, constructive insights and for showing me how to structure this report to make it scientific. Thank you also for proof reading the draft of this thesis.
A lot of people from the computer science department have also contributed to the success of my internship in various ways. Especially the system administrators Mr Allan Hayes and Ms Cathy Chandra - Great support! Special thanks to the additional spell-checker Ms Sui-Ling Ming-Wong.
Thanks to all the students of the game design course - I will never forget those nights we spent on research at a LAN evening with lots of pizza. I miss you all. (Especially Aidan, Grant, Brandy, Billy, Brett, Matthew and Andrew - you are a very friendly bunch!)
And of course thanks to my family especially my parents, Wolfgang Nacke and Anne Nacke, for their never-ending support, love and encouragement - and my girlfriend Ellen Zacher, who even though we were locally separated from each other for six months always gave me all her love, trust and encouragement. Thank you.

Lennart Nacke, June 2004

Contents

List of Figures

List of Tables

List of Abbreviations

AGDC	Australian Game Developers Conference (2003)
AI	Artificial Intelligence
API	Application Programming Interface
COSC	Computer science
COSC 360	Name of the computer game design summer school course
CTO	Chief Technology Officer
CVS	Concurrent Version System
FAQ	Frequently Asked Questions
GML	Game Maker Language
GUI	Graphical User Interface
IDE	Integrated Development Environment
IGDA	International Game Developers Association
LINUX	Linus Torvalds UNIX
Maya PLE	Maya Personal Learning Edition
MOCAP	Motion Capturing
MS	Microsoft
MSDN	Microsoft Developer Network
MySQL	My Sequential Query Language
NDA	Non Disclosure Agreement
NES	Nintendo Entertainment System
NPC	Non-Player Character
NPR	Non Photo Realistic
NURBS	Non Uniform Rational B-Splines
NZD	New Zealand Dollars
NZGDA	New Zealand Game Developers Association
NZGDC	New Zealand Game Developers Conference
PHP	PHP Hypertext Preprocessor
RAM	Random Access Memory
SDK	Software Development Kit
SDL	Simple DirectMedia Layer
UI	User Interface
UNIX	Uniplexed Information and Computing System

Part I

Development

Chapter 1

Computer Games in an Academic Context

Due to the popularity of home computers, consoles and portable devices (like the Nintendo Game Boy); computer games have become an essential part of modern society. In the still expanding entertainment industry, computer games have by now far exceeded the revenue of their sister, the movie industry. However, whereas movies are today an accepted art form, which is taught and analysed in academic institutions worldwide, computer games are still not widely accepted.
Academia and industry are slowly realising the potential of computer games, not only as an art form of itself (e.g. NPR Games [FMS01]) but also as a topic for academic research. Still, this industry is in need of highly trained young professionals.
The games industry of today consists mainly of self-trained graphics programmers as well as a few professional people who have gotten in with just their passion and commitment to games. Few teams have professional structure; most of them are young motivated start-ups with the lack of a general education. That is why the IGDA has brought up an example curriculum for use in academia [IGD03].
This curriculum gives an idea of the topics that are relevant to the production process of computer games. The curriculum gives only the outline with which the academic institution, in this case the computer science department of the University of Otago, has to decide on the scope of these suggestions for available course formats.
During the regular semester computer science students generally have a variety of subjects which make up their courses. Thus the time spent on a computer game course is only as much as they can spend on their other courses, but the amount of work that computer game production requires is usually much higher than that of other computer science subjects. This is because the complexity of computer game software includes knowledge from other classical computer science areas such as artificial intelligence (AI), computer graphics and human computer interaction. As well as this it incorporates non-computer-science areas like educational science, visual arts and design, philosophy, marketing and business.
With that in mind a format different from a regular semester course might be more

suitable to emulate a game production process. If such a format is not available, it is better to pick from the range of computer game topics just one aspect that fits into the study course of the targeted students.

1.1 Characteristics and Constraints of the Otago Course

The course at the University of Otago was co-developed by staff from the design and computer science departments. This originated from the idea to give the broadest scope on computer game topics that was possible at this university. Since almost all attending students came from computer science, the focus of the course could easily have just been on computer game programming. But since this was the first course of this kind, its aim was to be propaedeutic.
The course name "COSC 360 - Computer Game Design" implies that the endeavour of this course concept was to cover processes outside of programming, too. The term "Computer Game Design" itself is very hard to define. According to [RA03] it is a process of "Imagining a game", "Defining the way it works", "Describing" its elements and "Transmitting" that idea to the game production team. This had to be part of the teaching.
The subject itself was completely new to the university so no resources from other courses could be used. The department of computer science at the University of Otago decided, that the course was to be introduced as a summer-school paper in 2004. These papers offer the equivalent of a complete 13-week semester course carried out in just half the time (i.e. 6 weeks). This gives you the advantage that students will focus on the subject "around the clock" and you can cover more than just one aspect of computer games. This also means that work is going to be more intensive and therefore a good infrastructure must be provided. The computer science department would provide the students with a computer laboratory rightly tailored for their needs. The installation of this lab and its computers will be described later in this report.
Mr Simon McCallum wanted to use the subject to additionally introduce the students to a software production workflow, that is similar to the one used in the industry. Therefore all the concepts of the course needed to focus on actual usability in practice.
The following chapter will show by means of which criteria we assembled lecture topics and course ideas from the IGDA curriculum framework.

Chapter 2

General Computer Game Course Layout

2.1 General Course Topics

For the construction of a curriculum the IGDA Curriculum Framework was used as a consulting paper [IGD03]. It breaks down the major topics of computer game education into the following categories "Critical Game Studies", "Games and Society", "Game Design", "Game Programming", "Visual Design", "Audio Design", "Interactive Storytelling", "Game Production" and "Business of Gaming". These core topics had to be analysed with regard to their suitability for the available course format and target audience. The relevance of each topic and its use in the Otago computer game course is analysed in the following.

2.1.1 Critical Game Studies

According to [IGD03] this comprises "Criticism, analysis & history of electronic and non-electronic games". Out of this wide area it was decided to discuss definitions of game play and game flow. Another discussion would seek explanations for the term "willing suspension of disbelief" [RA03] with the students. The student would be encouraged to find some texts about "Ludology" on the web.
One of the first course lectures would need to deal with history of computer and video games. This would give a synopsis of console design and development over the last decades as well. Another decision was to include definitions of a game and "meaningful play" as it is described in [SZ03] as an important lecture topic.
Lastly the film-and-media-studies department was asked to contribute a lecture that would deal with games environments.

2.1.2 Games and Society

In [IGD03] this is defined as "Understanding how games reflect and construct individuals and groups". The "experience of play"[IGD03] in everyday life and its impact on players is well suited for a student discussion in tutorials. With a look at the audience this category would sadly need to be covered in brief only. Computer science students are usually not very interested in sociological aspects and theoretical discussions. This is a fact that should be changed in the future by changing the general computer science curriculum from covering very technical topics to also include training of soft skills. As far as I know, soft skills are already being included into computer science curricula. The "Otto-von-Guericke" University of Magdeburg is one example for this.

2.1.3 Game Design

"Principles and methodologies behind the rules and play of games" [IGD03]. This is certainly one of the largest categories in computer game development. It is also the most complex and most difficult to teach.
Game theory, game rules and game balancing would expand the lecture topics from the "Critical Game Studies" category. The user interface and interaction with the computer were put on the lecture schedule as well as a topic to discuss in the tutorials.
It seemed desirable to at least touch on game culture and gender specific aspects in one of the tutorial sessions.

2.1.4 Game Programming

As it says in the definition [IGD03] this category covers "Aspects of traditional Computer Science - modified to address the technical aspects of gaming". Knowing from experience it was concluded that an audience of computer science students would appreciate this topic the most.
Most of the basic physics, maths and algorithmic background for computer games need to be available in something similar to a manual. Therefore the decision was made to use a technical game programming book as a textbook for the class [Só3]. With this, the students would always have a quick reference at hand when they were going to implement games.
This main part of the course would be a programming project, in which the students would try to finish a computer game prototype by the end of the course. For this, programming teams would need to be formed.
The iterative model for software development was going to be a lecture topic among formalism like the creation of design documents, C++ and Windows programming, 2D and 3D graphics programming (general computer graphics). Software engineering topics would also be included as part of game programming and mainly consist of documentation, version control and testing. Principles of artificial intelligence would need to be a topic as well.

Other points of interest in this area included the component architecture of a game engine, security issues of network games and basic principles of collision detection. Modularity of code was a principle the students needed to stick to during the development of their games.

2.1.5 Visual Design

The design department was responsible for "Designing, creating and analysing the visual components of games" [IGD03]. The relevance of visual design in this course would not include traditional art and design fundamentals as much as practical 3D modelling and animation. Computer graphics programming is considered as a part of the preceding category of programming. Visual design is more concerned with the creation of game art and designs. The format of the course would not allow going into too much depth here and to cover areas like "character design" or "lighting and effects". The tool the students would use for their games would need to take care of 3D effects as well.

2.1.6 Audio Design

Although sounds are important for the user experience, the "Designing and creating sound and sound environments" [IGD03] would only be a small part of the course. Of course a sound creation lecture or lab would be necessary, in which the students would be introduced just to the basics.

2.1.7 Interactive Storytelling

Mr McCallum thought that "Traditional storytelling and the challenges of interactive narrative" [IGD03] would be very suitable for an introduction to the course. Thus the students would need to create an interactive story by working in an interactive fiction environment called Inform [Nel01]. The English department would also provide a lecture about narrative. Labs and a graded assignment on the work with Inform would also make up a part of the course.

2.1.8 Game Production

"Practical challenges of managing the development of games" [IGD03] was a major part of the students projects. Student projects would be completed in teams consisting of members responsible for different roles (which will be described later). The importance of coordinating their work was something that the students would need to learn during the course.
A preliminary lecture on group dynamics was also considered.

2.1.9 Business of Gaming

"Economic, legal and policy aspects of games" [IGD03] is a wide field. To get a little bit of this interesting topic close to the student audience help from the business department was needed. Fortunately Mr Phil Osborne, a lecturer of the marketing department, was happy to be involved in the course and to give a lecture on the business of gaming. Because of Mr McCallum's connection to the New Zealand Game Developers Association (NZGDA), it was also possible to have Mr Mario Wynands of Sidhe Interactive[1] to give a talk about business practices in the industry. Another industry related lecture was planned with Mr Jon Labrie, former Chief Technology Officer (CTO) of Weta Digital[2].

2.2 General Course Concepts

For the creation of this course several brainstorming sessions with course coordinator Mr Simon McCallum took place to consolidate basic ideas and concepts of the course. These concepts originated from talks with people from the computer game industry as well as suggestions from other lecturers about things that worked well in their courses. The main criterion was suitability of features for application in the computer game industry. The training of skills in writing and software engineering seemed to fit this criterion and seemed to be a good addition to the soft skills needed in game design. Learning how to express oneself, which is another important soft skill, would be practised in the tutorials. Another criterion which would influence the choice of tools that will be presented in the next chapter was the price of software. The budget for the course was not large. The following Subsections present the final choice of concepts.

2.2.1 Writing Skills

Firstly the course would start with a major assignment in interactive fiction worth 10% of the students' grades. For this assignment the students had to become familiar with the Inform language and learn how to program, compile and run Inform files. They would have to use Inform to write a little text adventure game, where they had to challenge generating interactive narrative.
The next major assignment due after three weeks of the course would be the game design document or game script worth 15% of the final grade. Those documents would consist of approximately 30 pages and the students would need to analyse their ideas and transform them in an interesting script. Writing a design document before starting to write the game is a common practice in the computer game industry. Thus it serves the main criterion of industry suitability.
A large part of the final exam would be the post-mortem of the games, created by the student teams. This means analysing the work that they have done in the course and

[1] A major game development company in New Zealand
[2] Company responsible for the Special Effects in the "Lord of the Rings" movies

evaluating how many of the ideas from the design document have made it into the game prototype which they would hand in at the end. A tutorial would need to be offered to show how to write a post-mortem in one of the last laboratories. Writing post-mortems after finishing a game has also become very common in the computer game industry. Many interesting post-mortems of major productions can be found in Game Developer Magazine and on Gamasutra[3].
The reason for including these documents is also to give the students the possibility to be creative before and after touching the code. By doing this they always have the chance to evaluate the work that they are doing.

2.2.2 Software Engineering Skills

Software engineering is the stepchild of many young computer game development companies. Version control software is becoming more popular but good code documentation is still not a standard in the industry. COSC 360 would introduce the students to many interesting topics in software engineering while always keeping the focus on using the tools for game development. Version control software and its proper use would need to be taught to students.
Another software engineering focus includes code documentation and commenting. Tools for this purpose include the MS Office Suite as well as free tools like LaTeX and Doxygen. The training of industry-related tools was an important aspect for Mr McCallum. Documentation tools will be evaluated in the next chapter.

2.2.3 Team Skills

Responsibilities for different areas of computer game production need to be split among an average of five students per student team. The teams should be formed in the first week of the course and each group has to assign the following positions to each team member: graphics designer, chief software architect, audio engineer and project manager.
By taking a role the person will have the responsibility for the outcome of that certain part of the project. This does however not exclude members from helping each other throughout the different areas. Most of the work will be the code implementation involving all of them.
With the summer school format in mind this promotes the idea that the students need to learn time and stress management along the way. Daily meetings will need to be organised and milestones would give a control over group process, which group managers need to report about in weekly meetings.

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

2.2.4 Practical Knowledge

This is used as a generic term to describe the training in the use of specific tools. As already mentioned in subsection 2.2.2 the training of certain tools was an important aspect for Mr McCallum. The tools that are used in the industry also need to be taught in the course. This includes integrated development environments (IDEs) for programming languages, graphic design tools, 3d modelling tools and of course a game engine or other middleware. The overview and evaluation of such tools is done in the following chapter.

2.3 Additional Ideas

In the following a few additional ideas are listed, that came to mind during the planning of this course together with Mr McCallum. A basic teaching concept to accelerate the workflow for the students was to be available in the computer laboratory around the clock to answer all upcoming questions.
For managing suggestions and keeping track of recent information a dynamic website had to be programmed, which will be explained in further detail in Part II of this report.
One of the things that were considered to be important for a good learning atmosphere was to establish a course culture. The computer science Linux lab, consisting of 50 workspaces would be converted into a Windows XP lab.
When enrolment finished 40 students were listed, so every student was going to have his own designated machine and all the groups were going to sit together in workspaces. With that knowledge the idea was brought up, that it might be good motivation if the students brought along tools that would help them be creative around at their workspace [e.g. Action Figures for character animation]. Another plus for the creation of an industry atmosphere was the support of local computer stores, which gave away video game posters. The lab was decorated with them afterwards. The whole idea behind these environmental settings was to help students focus on computer games and be creative and productive along the way.
People are usually more productive if they can identify with an idea and feel comfortable with their working environment. The logo in Figure 2.1 was designed to complete the corporate identity of the course. During the course students requested to have a T-Shirt

Figure 2.1: The COSC 360 course logo

logo design contest. The best design would go on a T-Shirt, which would be made available to all the course members at the end. If no logo would win, the course logo would go on the shirts. To keep the students informed about daily news from the game industry, a subscription of Gamasutra[4] was recommended to them and game related news would be posted on the whiteboard in the computer science lab.

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

Chapter 3

Tools and Game Engines for the Course

3.1 Tools

For the course the University of Otago a default computer environment was predetermined by Mr McCallum. The few tools that were chosen as defaults are listed in the following. For each tool the field of application as well as a short evaluation of its usability for the course will be given.

3.1.1 Microsoft Windows XP and Visual Studio

As research on the game industry shows most games are developed for a Windows platform, because the main market uses this platform and typically the game industry tries to target the largest market available. For the Windows environment there is basically no good alternative to the Windows compiler of Visual Studio and its very intuitive Integrated Development Environment (IDE).
Since some of the students would already be familiar with the Visual Studio IDE from other computer science courses this solution is both, student-friendly and industry-related[1]. It also would not bring about any extra costs, because the licenses were already available at the computer science department.

3.1.2 LaTeX

This "document preparation system" [Lam94] was chosen as means of documentation for the process of the student projects. Since it is widely used in computer science for research papers and such, Mr McCallum thought that students would benefit from a

[1] Although the author wants to make a little annotation that an open source project called Dev-C++ also provides a very good, alternative IDE for Windows programming

basic introduction to the use of this system.
The students would have to learn to document in LaTeX but would also be given the opportunity to work with a basic office installation. A good Windows distribution of LaTeX is the open source project MikTex which is developed on Sourceforge and works well together with TeXnicCenter. The latter is an Integrated Development Environment (IDE) for creating LaTeX documents in the Microsoft Windows environment.
LaTeX is good tool for scientific reports but the use of it for the writing of design documents is quite poor. When it comes to brainstorming and keeping track of ideas on the fly, a full word processor[2] offers a much better service. Because ease of use is essential, most students would also prefer Microsoft Word for writing their design documents.

3.1.3 Version Control Software

When multiple users work on the same files it is quite hard to keep track of changes manually so nothing gets deleted by accident. Therefore the use version control software in even a small development group is essential.
After talking to several companies at the Australian Game Developers Conference (AGDC), Mr McCallum knew that the Concurrent Version System (CVS) was very popular in the computer game industry. CVS is a tool that enables the user to track changes to a set of files over time. It allows multiple developers to coordinate changes, track versions and permit simultaneous development of different versions of the same code. It merges the code.
Despite other tools like Microsoft Visual Source Safe, which basically offers the same functionality, CVS is freely available without any cost. This certainly adds to the popularity of this software. Since Visual Source Safe was not available at the computer science department the students would be working with CVS. Additionally the use of popular Windows clients like WinCVS[3] and TortoiseCVS[4] would need to be explained to the students.

3.1.4 Inform

Inform was invented in 1993 by Graham Nelson and is a framework, which allows the creation of text adventure games. Once Inform is compiled, it will run on almost any platform, given it provides an interpreter[5]. Inform is a scripting language, a library and a compiler that generates the code to a platform independent .z* file format[6].
The library provides a basic "world model" [Nel01], that defines rules and sets a basic list of verbs a player can use. The library parser interprets those verbs and calls

[2] like OpenOffice.org or Microsoft Word, which was available on the student computers
[3] http://www.wincvs.org/
[4] http://www.tortoisecvs.org/
[5] One can find interpreters at: http://www.inform-fiction.org/zmachine/interpreters.html
[6] files for the "Z-Machine" [also used in early Infocom adventures Zork, etc.] .z5 for version 5 up to .z8 for version 8

appropriate functions. The game designer only needs to worry about the functions or reactions that need to be implemented to make use of the world provided by the framework. In this sense Inform is a very simple version of a game engine, which programs can be easily constructed for.

3.2 Evaluation of Game Engines

The choice of the programming tools to develop a game with is a very important one, as it will influence the workflow and outcome of the production. For the COSC 360 computer game course, a tool suitable for beginners and medium level programmers had to be found. The first choice was whether to use a game engine or let the students write a game (and possibly a little engine) in their favourite programming language from scratch.
Since the course was going to have a mix of students from second year up to postgraduate students, Mr McCallum decided to have both opportunities available. Therefore a variety of available game engines had to be evaluated. The criteria had to be met to be suitable for the course at the University of Otago are listed in 3.2.1. The list is not close to a complete overview of the computer game engine market but rather is narrowed down to the ones that the author considered most interesting for use in the computer game design course.

3.2.1 Criteria for an Evaluation

All the game engines are analysed particularly with regard to the beginner assistance they provide. This includes the availability of pre-made assets that ease start requirements for beginner level students. The possibility of scripting events and therewith a simple programming language is another aspect of beginner assistance. This way everybody enters assignments with the same level of knowledge.
Another main criterion is the support provided by the manufacturer of the tool and, ideally, an online community. If a tool has steep learning curve this is particularly necessary. Furthermore the interaction of a game engine with other commercial packages plays an important role. This especially concerns 3D modelling packages, because such tools are generally very expensive. The latter brings up the last and probably most important criterion in this evaluation: The cost of the tool.

3.2.2 Game Maker

Game Maker was developed by Mark Overmars, a professor at Utrecht University, Netherlands in 1999. The program is written in Delphi and is available for Windows above version 98. Assistance for beginners is included in so far as that the tool has

an accompanying website[7] which provides downloadable online documentation, a huge number of well-written tutorials, many resource packs (like sprite libraries), a general "Frequently Asked Questions" (FAQ) section and a large forum[8].

An overview of Game Maker's Graphical User Interface (GUI) can be seen in figure 3.1.

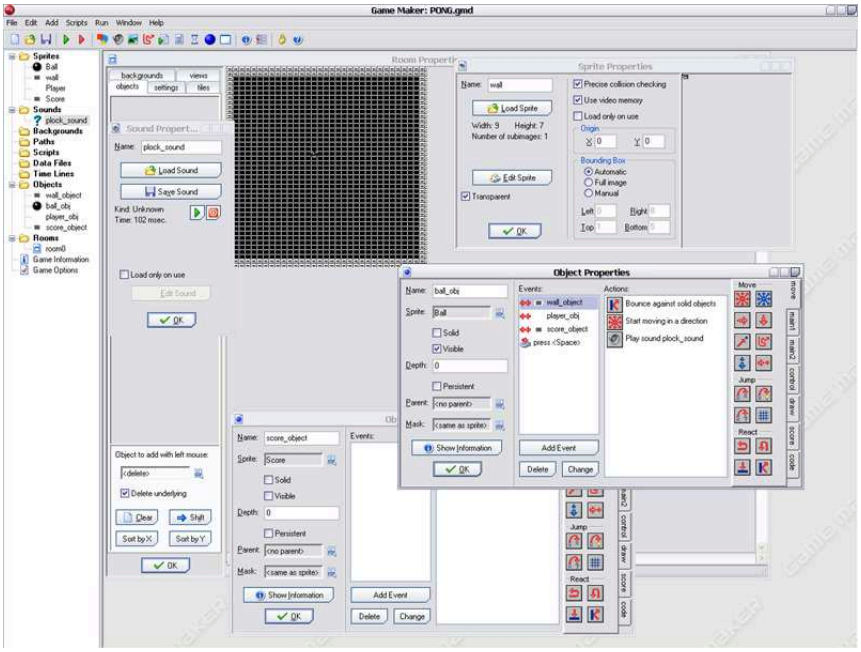


Figure 3.1: Game Maker environment overview

Especially the process of learning this tool is sped up by the many tutorials and the extensive documentation that comes with Game Maker. Small two-dimensional games can be developed in no time. To create a game, sprite graphics and sounds can be loaded and attached to objects. Properties (including events, actions, flags) like "Bounces against solid objects" can be assigned to objects and sprites. Objects are placed in rooms, which contain sprites and have properties on their own. Once the room is finished, Game Maker can compile the file and will create an executable.

But nevertheless the tool is also suitable for programmers, which want to have more control than just click their scripts and assets together. Game Maker comes with an own scripting language called "Game Maker Language" (GML) that allows more control over the assets in the game program. The language is well documented and many questions are answered in the online forums.

Unfortunately there is no real extensibility for Game Maker into the three dimensional world. The engine can emulate 3D, but is designed for sprite based games. Thus it does not interact with any commercial modelling packages.

[7] http://www.cs.uu.nl/people/markov/gmaker/

[8] It counts about 7,200 registered members and more than 300,000 posts at the time this report was written

The program is available on the Internet as a free and a full version. The free version reminds of registration every time it starts up, does not support a particle system and cannot be extended (as for the full version, one can program extension dlls). Registration fee is 15 Euro.
Technically Game Maker is a good introductory system for non-programmers into the world of computer game design, but for computer science students it might be too limiting in terms of experience with the game world.

3.2.3 Alice 3D Authoring System

Alice [Con97] is a project in the area of animation scripting in the Stage3 Research Group at Carnegie Mellon University, Pittsburgh, USA. One of the goals of Alice is to make computer graphics programming easier and more accessible to computer inexperienced users. It uses the means of interactive storytelling to give special appeal to girls.

It assists the user with a default model library which contains prefabricated models

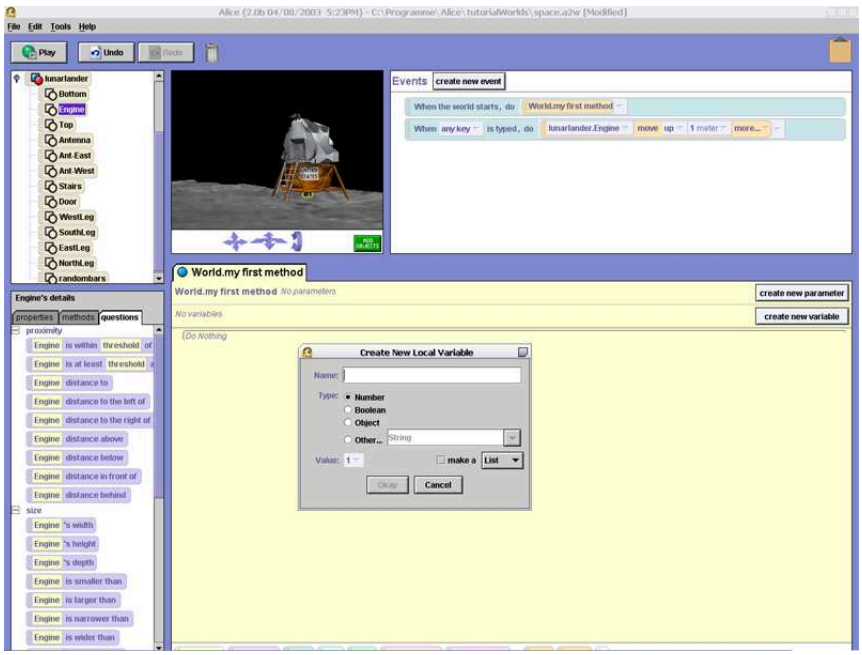


Figure 3.2: The Alice 3D scripting environment

and also provides a link to a large online model library. The main feature is scripting of events, which is done with default script snippets that belong to the objects available in Alice's 3D World. These script snippets refer to object oriented programming. Java's programming semantics have been wrapped completely, since Alice itself is programmed in Java. Thus the tool does not come with a scripting language of its own.

The tool itself is accompanied by a number of tutorials and there is a website[9] which contains a tiny FAQ section, some workshops and a small forum section[10]. An overview of Alice's GUI (as seen in figure 3.1) shows how the scripting of a simple event can look like. For novice programmers the tutorials in Alice are a good introduction into the world of object oriented scripting. Simple events and actions can be clicked together.
Creation of the 3D models cannot be done inside Alice. It is only concerned with the scripting of the model events. However it allows the import of "ASCII Scene Export" (*.ASE) files, which can be produced using Discreet's modelling software 3D Studio Max. Unfortunately this is the only way to create 3D objects for Alice.
The Alice tool is created for use in computer programming education and available completely free of charge via download but the additional cost of 3D Studio Max as a model creation tool is very high. Also the tool does not seem to provide good assistance to team-work since the merging of files into a scene leads to problems such as the total crash of the program. The program itself turns out to be quite unstable since it crashed several times during this analysis.

3.2.4 Shark 3D

In the computer game course which is taught at the University of Magdeburg, Shark 3D by Spinor GmbH is the engine of choice. It is a big, yet flexible, engine which enables programming for PCs and consoles. Shark's main difference to other game engines is that it is based on a component-architecture rather than just C++ libraries or pre-compiled tools.

The assistance that it offers for beginner level students is not extensive. The engine comes with three default projects (template_walk[11], template_camera and test[12]) which serve as a basis for first implementations of own levels. The documentation is rather technical and not really useful for beginner level game students. It is composed of three sections for programmers, designers and developers (each section consisting of a manual and reference).
The website[13] does not provide any kind of additional documentation, FAQ section or user forum. Only an overview of the technical structure and features is given. The support via email is good. Once a bug is discovered, Spinor's employees[14] are very eager to help and find the source of the error. Although Shark comes with some models in the default projects it provides no modelling solution itself. It ships with a 3D Studio Max Exporter Plugin which exports Shark 3D files out of the modelling tool. A support for Maya is planned but was still at development stage when the course started. Support

[9] http://www.alice.org/
[10] This forum counts 292 registered members and only a few more than 300 posts at the time this report was written
[11] Consists of basic components to move a player around
[12] A simple test level with basic interaction scripts and animation events
[13] http://www.shark3d.com/
[14] Spinor GmbH is the company behind the technology of Shark 3D

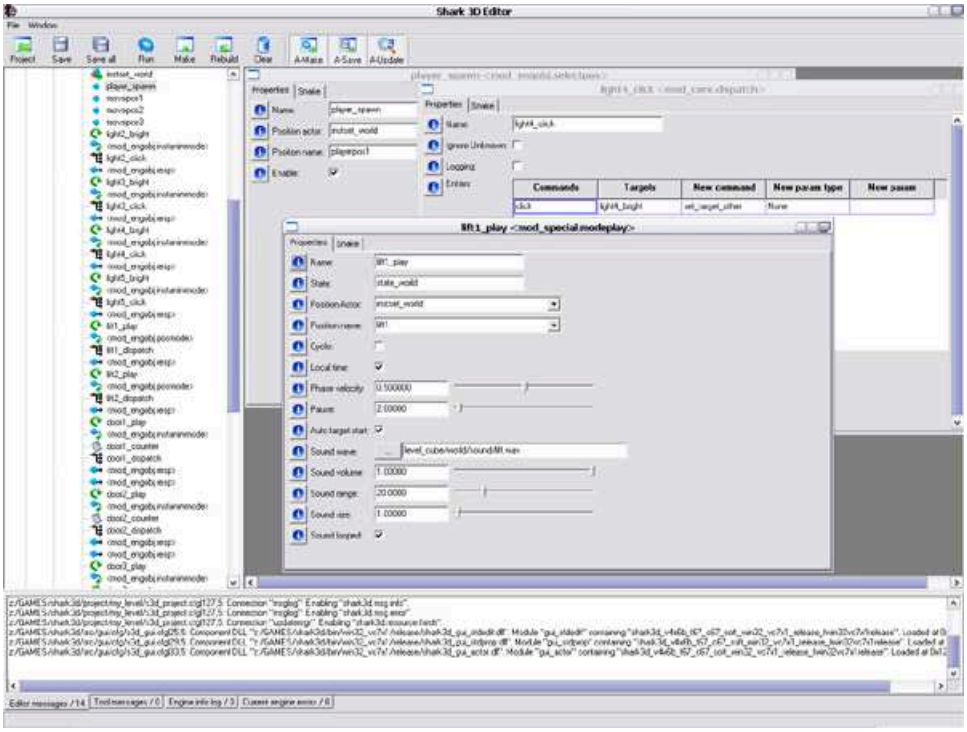


Figure 3.3: The new Shark3D editor GUI allows configuration of component files

for other modelling tools was not evident.
The component architecture allows programmers, designers and developers to work on different levels with the same engine. For example the designer will never need to touch any of the C++ code that runs the engine. Basic scripting of events can be done in component description files which control the actor objects in the engine. The engine also supports an interface to Java and C++ code.
Shark 3D also features a dynamic resource update (DRU), which enables the programmer to change component scripts while the engine is running. The changes in the code are instantly shown in the game.
Shark 3D is a commercial game engine but also provides the possibility for educational institutions to use it free of charge when a Non-Disclosure Agreement (NDA[15]) is signed. The version that was looked at was v4x6b and it featured a GUI for editing the game projects' component files, which it did not have in older versions.
Nevertheless Shark 3D Version v4x6b was only compatible with 3D Studio Max for importing 3D models. Due to the fact that the University of Otago did not have a licence for 3D Studio and for the course not enough funding was available. Thus Shark was considered for use at first but dropped later to the benefit of 3D Game Studio.

[15]Where you assure not to distribute the code to third party or use it commercially.

3.2.5 3D Game Studio 6

3D Game Studio by Conitec Corporation is a tools package. The three editors for models, levels and scripts interact together to create a game engine which is well suited for beginners and a nice consideration for advanced programmers as well.
The Manual is huge and features references and step-by-step tutorials for all three editors and a detailed introduction to the scripting language C-Script of Game Studio. Some tutorials allow even the absolute programming novice to create a simple shooter game in almost no time at all. The website[16] contains a FAQ section, developer contests, a huge resource site, a user magazine and a massive forum[17]. Additionally Game Studio comes with many pre-fabricated assets such as 3D models, textures and whole games.
Customer support is done via the forum and email. The reaction to occurring problems

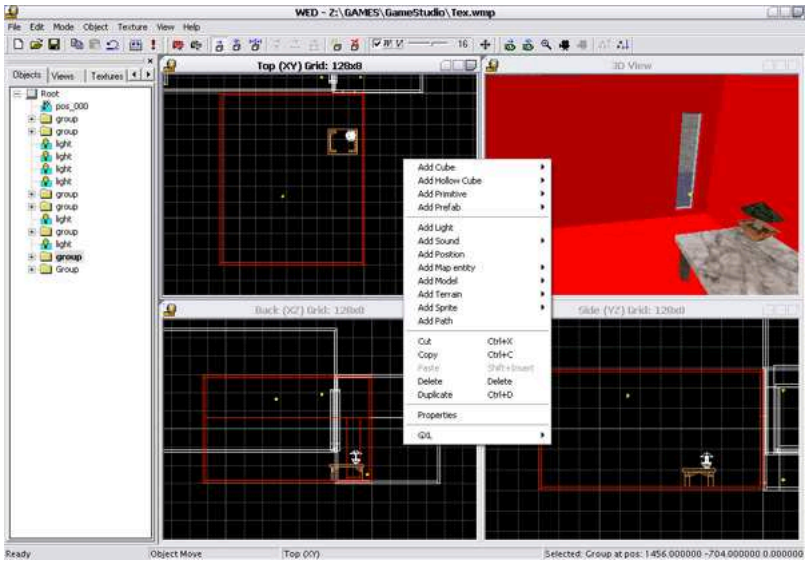


Figure 3.4: Game Studio level editor environment

is very fast. Furthermore Game Studio supports almost all known modelling formats (i.e. Worldcraft, Milkshape, 3D Studio Max, Maya) and interacts without problems with all tools. It also features its own model editor called MED.
The actions events and some of the physics inherent in the game studio engine "Acknex" can be scripted with the help of C-Script. The syntax closely resembles the C programming language but it limits the user to function calls within the game studio environment. All program scripts are compiled before running the game. Import of pre-made Quake or Half-Life levels can be done using the level editor (called WED)

[16] official website: http://www.3dgamestudio.com/ and the accompanying resource site http://www.acknex-unlimited.de/

[17] The official forum counts 7,928 registered members and more than 300,000 posts at the time this report is written

which make the tool very accessible for the intermediate game developer who generally works for game mods and suchlike. Finally the Game Studio SDK enables professional programmers to extend Game Studio with Plugins (DLLs) to suit their special needs.
Game Studio is a commercial game engine, but the pricing options for educational licences are low. A thirty-student licence costs US$ 399 and a special deal that was offered for the course was US$ 599 for 100 students. Thus the first option comes to a single cost of US$ 13.30 or NZ$ 21.74 per student[18]. The second option only costs US$ 5.99 (NZ$ 9.80) per student.
Thus Game Studio is an affordable solution for a smaller budget and due to the huge community very suitable for beginners. For a complete overview of the pricing options please refer to table I in Appendix A.

3.2.6 Auran Jet

Auran Jet by Auran Pty Ltd. is a professional development platform for 3D computer games. Auran itself states that the engine can be used for more than just game ap-

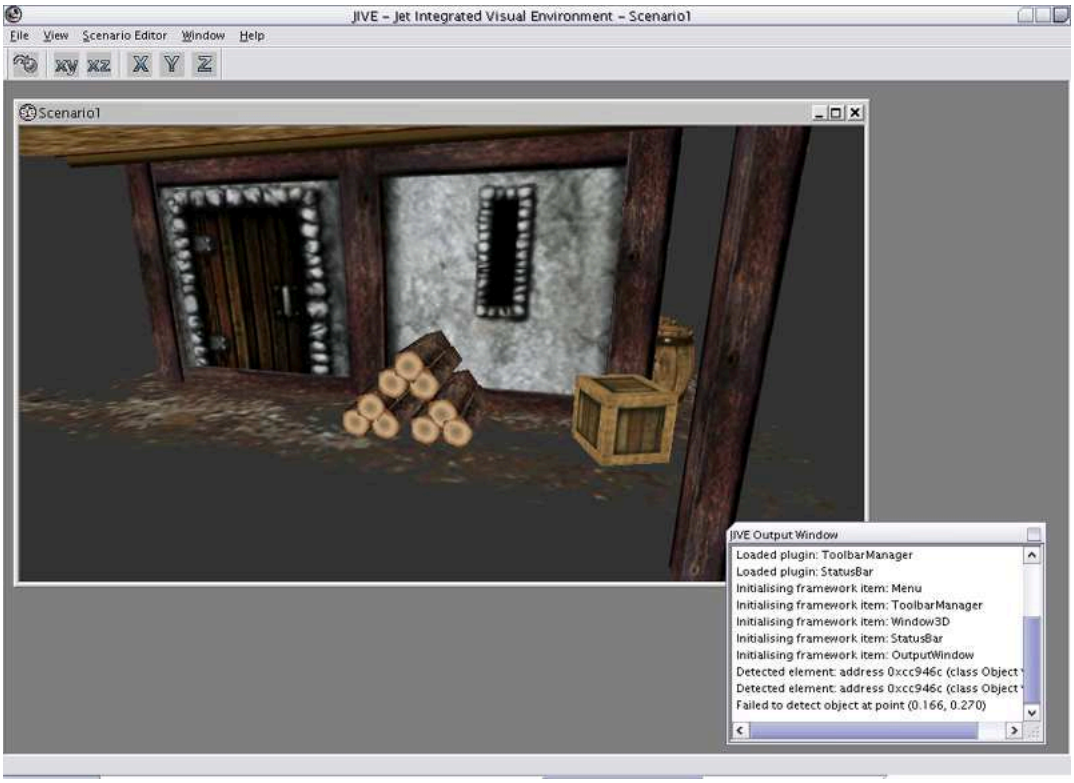


Figure 3.5: The Auran JIVE tool

plications. The engine provides tools for the development of interactive 3D products

[18]Exchange rate 0.611889, 28.10.03

without a restriction in the style of application. Of course this is a feature that most game engines provide.
When licensed, Auran provides a community for "Jet" developers consisting of a large online forum, a Newsletter and CVS access. Auran Jet is being developed in Australia which was interesting for the fact that one of the course lecturers had contacts to some of the developers at Auran. Support for Auran Jet is increased for commercial licences. A "dedicated support programmer" [Auran.com] provides email and telephone support. Exporter Plugins are provided for 3D Studio Max and Maya up to Versions 4. Meshes and Animations can be exported directly from those packages.
A student licence for Auran Jet would cost AU$ 99 (NZ$ 114) per student. The licence includes the access to a booklet, manual and access to the developer forum. The price and the additional cost of the modelling package were too expensive. A list of the pricing options can be found in table I in Appendix A.

3.2.7 Torque

The Torque Game Engine is a product of GarageGames, Inc. who affiliates to with the independent game developer community. It has been used for a dozen titles, the most

Figure 3.6: Torque demo application

famous being Tribes 2 (published by Sierra). The engine's focus is on good network support.
It features engines for scripting (with an integrated compiler), GUI, 3D, meshes, particles, terrain and interior rendering. Scripting is done with C++ like syntax and given functions. The GUI engine allows constructing scriptable interfaces and features a default control set. It also features a GUI Builder. All common modelling formats can be imported.
The mission editor lets you construct objects and script missions, edit terrain and a lot more. Sound support is done with the help of OpenAL, a freeware audio library. Lots of tools like a terrain editor and generator (also for fractal based terrain generation) come with the engine.
The commercial product is used by a lot of professional programmers and a licence for 50 students would cost US $ 2.250. That makes US $ 45 (NZ $ 74) per student. The licence grants access to developer forums and the official CVS. Nevertheless the cost is too high and the features are too professional for a beginner course. For an overview of the pricing options of all the commercial tools refer to Appendix A, table I.

3.2.8 Freeware Alternatives

There is a large variety of game engines available as open source projects. Irrlicht[19], OGRE[20], Crystal Space[21], NeoEngine[22] and Genesis 3D[23] are just a few of them. Due to the limited time available for research on game engines suitable for a beginner course, it was impossible to analyse all the engines mentioned above. Some of them are more attractive for professional programmers; some of them simply lack good documentation. In the end none of the engines were recommended but some general information about these possible alternatives was given to the students.
Some of the projects that were looked at here, might become possible tools in the future.

OGRE (Object-Oriented Graphics Rendering Engine) is an object oriented approach to a game engine in C++. The engine is designed to work on different platforms like UNIX, OSX and Windows. It is independent of a 3D implementation like Direct3D or OpenGL (even though those are supported). Like most open source engines, it consists of C++ classes that manage common graphical requirements like culling and rendering. It allows implementation of shaders (e.g. written in Cg or Assembler) and supports most mesh and texture formats. It also offers plugins for the most common modelling tools.
All in all, the engine is a collection of multiple classes and although they are well documented, this approach is more code based [in this case C++] than most other engines.

[19] http://irrlicht.sourceforge.net/
[20] http://www.ogre3d.org/
[21] http://crystal.sourceforge.net/
[22] http://www.neoengine.org/
[23] http://www.genesis3d.com/

This makes the learning curve very steep for beginners. One of the key features of most commercial engines is to keep the learning curve gradually low. All of them offer working demos or default levels to get started quickly without having to read the manual intensely.

Irrlicht is the name of another advanced engine. The focus of the engine is to be high performing for real-time 3D applications. It is a cross platform engine that is easy to use and fast. It uses Direct 3D and OpenGL for its rendering process. It supports a character animation system, particle effects, billboards, light maps, environment mapping, stencil buffer shadows and a software renderer. Another feature is the 2D GUI System which can import common mesh file formats and textures.
Thus it offers almost all of the features of a professional engine and also includes good documentation and some tutorials. This may definitely be worth a look for an intermediate computer game design course if not the next COSC 360.
The other open source game engines mentioned above will not be described in this text,

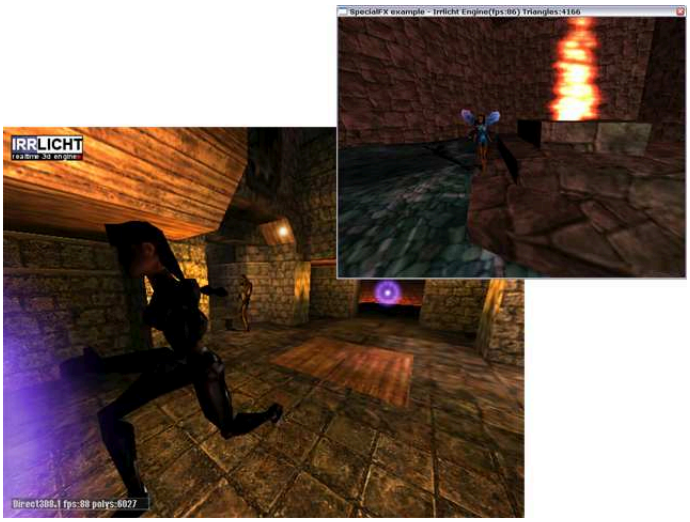


Figure 3.7: Irrlicht engine technology demos.

but their websites provide good details about their current development. Their features are similar to those of OGRE and Irrlicht although they have their differentiating factors. In a nutshell freeware computer game engines are becoming a good alternative to the commercial ones and should be considered for use in the education of computer game design.

3.2.9 Conclusion

For the final decision on which tools would be used for the course, negotiations about the pricing packages took place with several distributor representatives of the mentioned distributing companies. In addition to that some costs that are connected (e.g. if an engine needed a special modelling suite) were considered. Table 3.1 illustrates the separate costs of such modelling packages. Blender 3D was included in this research after the course started so it could not be installed on the computers. However the most attention was given to the way that beginners are assisted when learning to use a game engine. If the assistance was not extensive, the tool needed to at least have a good support to be suitable for the course.

After discussions with Mr McCallum it was agreed that Shark 3D and Game Studio

3D Software	**Cost Options**
Maya Complete	US $ 7,500 for a node-locked licence, available in the Otago graphics lab
Maya Unlimited	US $ 16,000 for a node-locked licence (includes Maya Fur, Maya Live and Maya Cloth)
Maya PLE 5	Completely free and still powerful w/ watermarked rendering, 133 MB download, Game Studio supports Export
3ds max	US $ 3,495 for a full retail licence Educational licences start at US $ 600
Gmax	Completely free, lacks some features (small download)
LightWave 3D	Educational Version 7.5 (5-User Lab Pack) for US $ 975.95 not support by most engines
Softimage XSI	Student Versions for Essentials US $ 495 and Advanced US $ 795 free version EXP for Half-Life 2
Blender 3D	Free open source project[24], big community Has some bugs but supports most export formats

Table 3.1: Cost of 3D modelling packages.

looked to be the most likely to be chosen, even though the more professional tools like Auran Jet or Torque were still in talks about pricing. When all pricing offers were collected the final decision was made.

The benefit of the commercial engines was the good support, although the open source engines have a large user community that tries to assist with questions. What speaks against most commercial engines is the very expensive price. For advanced programmers they supply good tools to create games, but for students the relation between cost and

Game SDK	**Category**	**Beginner Assistance**	**Support**	**Inter-action**	**Cost**
Game Maker	Educational	Documentation, tutorials, resources, FAQ, large forum	Good, via forum and community	None with modelling packages	Free or 15 Euro
Alice	Educational	Model library, few tutorials, tiny FAQ and forum	Via forum, not tested	3ds max	Free
Shark 3D	Commercial	3 default projects, large documentation, no community	Good, via email	3ds max	Free for education /with NDA
Game Studio	Commercial	Documentation, tutorials, user magazine, resources, FAQ, large community, editors for modelling and scripting	Good via forum and email	3ds max, Maya	US $ 599 per 100*
Auran Jet	Commercial	Editors, newsletter, CVS, forum	Via forum, email and phone	All	AU $ 99 each*
Torque	Commercial	Editors, tutorials, community	In the community, via email	All	US $2.250 per 50*
O.G.R.E.	Open source	Tutorials, community, documentation	Not official, community	All	Free
Irrlicht	Open source	Tutorials, FAQ, forums, documentation	Not official, community	All	Free

Table 3.2: Comparison of game engines on the basis of certain criteria

effect is bad. The table 3.2 shows the criteria that the game engines were compared with.
The differences between the commercial and educational licences for approximately 50

students (which is the proposed number of course attending students) are significant. Not included in this research are the times the licences last for the different engines. The course budget did not allow purchase of a professional modelling tool nor of a high-end professional game engine. Comparing the cost of tools in table 3.1 that are compliant with those in table 3.2 the decision was made to go for a cheap solution that provides the most assistance and the largest community.
In the end it was decided to use Maya PLE and Gmax as 3D modelling tools for the course and Game Studio as the primary game engine. Since Game Studio supports a plugin for Maya PLE it became the modelling tool of choice. The decision for Game Studio was made because it was the most complete of the packages for the cheapest price. To publish the student games an additional professional licence was purchased because the student version does not support publishing.

3.3 Game Programming Languages

For the creation of games not only the choice of tools plays a major rule, but also the selection of the programming language. Two popular programming languages are presented and their use in game programming is discussed.

3.3.1 Programming in C/C++ and Java for Games

The most common languages used in the computer game industry today are C, C++ and Java. While the latter is more often used for the development of small internet game applets, C++ has become the industry standard for developing computer games. It is a bit more flexible than C and utilises a lot of libraries for game development (most of them focussing on graphics, sound and input). Java is generally used for Internet and Mobile games (on cell phones together with special SDKs).
C can be used for programming games, but C++ is more popular among programmers because of its object orientation, class structure and integration into MS Windows. Nevertheless programming with C++ also depends on the libraries that are available for easy multimedia integration. Some of them most common libraries used in COSC 360 for game development are presented below.

DirectX

DirectX is an SDK (consisting of a set of low-level APIs) designed by Microsoft for multimedia applications and games. It integrates very well into the Windows compiler and Visual Studio. Version 9 features the following Application Programming Interfaces (APIs):

- DirectX Graphics
  bundles DirectDraw and Direct3D into one API, including the Direct3D extensions (D3DX) utility library.

- DirectInput
  supports different input devices (keyboard, mouse, joystick, controllers) and force-feedback technology.
- DirectPlay
  network applications, especially multiplayer games, includes session management.
- DirectSound
  used for playing and capturing waveform sounds.
- DirectMusic
  similar to DirectSound but preferred for soundtracks, also handles MIDI sounds.
- DirectShow
  handles Video/Audio capture and playback, can be also be used for streaming video
- DirectSetup
  enables easy DirectX installation.
- DirectX Media Objects
  COM-based data-streaming components, handles conversion from input to output data.

The core communication between the different APIs demands the most practice. The initialisation of the APIs can then be stored in functions. Especially with the help of Direct 3D (now DirectX Graphics), sprite movement and reaction can be controlled with some skill. Thus, the possibility of creating a game engine is given that allows control over the graphics with functions that utilise DirectX, while the game itself uses the written functions.
Another API library for C coding is **SDL**[25] (which works with C++). It is a multimedia library, which runs on several platforms. In the Windows Environment which was used for the course, SDL accesses DirectX and makes its handling much easier. SDL is an open source library that would be provided to the students along with DirectX.
Java is used in many academic courses as an introductory language to programming. In Java, instances of MediaTracker and Threads can be used to move sprite images and to program internet games. Initially, the use of Java for simple programming examples was considered but later dropped in favour of teaching C++ to the students. Thus Java was not really used in the course. For one of the tutorials, which was concerned with testing state-based AI behaviours with Robocode[26] students had the option of using Java to play around with that tool. As it turned out most students were in the crunch phase of their game development and did not spend extra time with the tutorial.

[25] Simple DirectMedia Layer
[26] available at http://robocode.alphaworks.ibm.com/home/home.html

Part II

Delivery

Chapter 4

Putting the plan into practice

After developing key concepts for the course and deciding on a working environment (including the game engines and work machine installations) further work on lectures and lab content[1] had to be done as the course progressed.

4.1 Ordering Course Literature

Game design is a young subject of research where it is almost impossible to get adequate books at a University library. Therefore a lot of book orders were placed a month before the start of summer school to have a good pool of information ready on close reserve. A lot of sample chapters were read by the staff and some of the very new computer game books were browsed through with safari IT. After having read a few the following titles were ordered "Chris Crawford on Game Design"[Cra03], "Macromedia Flash MX Game Design Demystified"[Mak03], "3D Games, Real-time Rendering and Software Technology"[WP01], "3D Games: Animation and Advanced Real-Time Rendering: 2"[WP03], "Game Programming Gems"[DeL00], "Java 2 Game Programming"[Pet03] and "Developing Online Games. An Insiders Guide"[MP03]. It turned out the students were happy with the selection as an addition to their textbooks[SÓ3, RA03].

4.2 Making the Course Website Dynamical

A static HTML Layout and site existed for the course[2]. An upcoming task was to make this website dynamic to finally feature a forum and mailing system. To achieve this, permissions needed to be granted from the department's system administrator Ms Cathy Chandra to transfer the web account to a web server that featured PHP and a

[1] This refers to the content of the practical exercise classes
[2] http://cosc360.otago.ac.nz/

MySQL database. After several discussions about security issues a database was created and the course staff were given web space on an application server.
First the navigation system of the website was redesigned to be modular (see Figure 4.1).

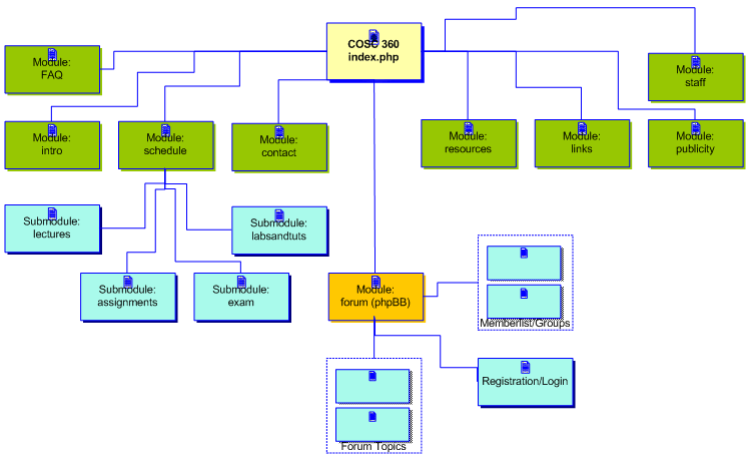


Figure 4.1: Modular website structure

The layout remained static, but the content was entered into a database or read from remaining files. A main file index.php included the modules that can be called via a navigation function. If those modules are further subdivided the navigation function automatically includes a submenu. Except for schedule.inc, resources.inc, staff.inc and publicity.inc the files remained static. These four files used the database to get their content. The schedule and resources needed to be updated as the course progressed since some lecture topics and times changed and not all the exercises had been written beforehand. A small insert and update script allowed to make changes on the fly.
Later a script was included to mail students proposals to the lecturers as well as an errata page for our textbooks. The open source script phpBB was used to supply the students with discussion boards.
Most of the functions and scripts written for the course page were also used for the New Zealand Game Developers Conference (NZGDC) website.

Chapter 5

Lectures

5.1 Course Scheduling

As mentioned in section 2.1 the course curriculum tried to orientate its content close to the ideas from the IGDA framework [IGD03]. But as it things progress, some things need to be changed and so some of the original ideas from chapter 2 needed to be adjusted. The first thing that was assured about a month before its start was the weekly schedule for the course. The schedule is illustrated in Table 5.1. The course

Time	**Monday**	**Tuesday**	**Wednesday**	**Thursday**	**Friday**
10:00-10:50	Lecture	Lecture	Lecture	Lecture	Guest Lecture
...	*break*				
13:00-13:50	*"placeholder"*	Tutorial 1A	*"placeholder"*	Tutorial 2A	
14:00-14:50	*"placeholder"*	Tutorial 1B	*"placeholder"*	Tutorial 2B	
15:00-16:50	Lab	Lab	Lab	Lab	

Table 5.1: Weekly time schedule of COSC360

involved lecture times from 10:00 to 10:50 a.m. Monday to Friday, with Friday being an exception insofar as the lecture was voluntary and given by a guest from an university department (other than the design or computer science department) or from a company. Furthermore the schedule shows four lab times from 3:00 to 4:50 in the afternoon, which refer to practical exercises in the computer laboratory[1]. Labs were scheduled every afternoon from Monday to Thursday.
To complete the schedule two 50-minute tutorials were offered twice a week. For these tutorials the students would split up in groups of around 15-20 people. In those smaller groups it was planned to discuss game related topics. The "placeholders" were planned

[1]For convenience these practical exercises are here referred to as Labs

to eventually move tutorial times from one day to another if needed.

5.2 Lecture Plan and Changes

With the schedule at hand lectures were realigned and the final flow of topics was fixed. The first week would start with an introduction to computer games via game history and feature topics from gameplay, groupwork and documentation to narrative. An assignment in Inform would be the students' first contact with game programming. In the second week more formal topics such as creating design documents, 2D sprites, collision detection and game engines would be the focus of the lectures.
The third week would focus more on visual things like design, storyboarding, animation and game environments. During this week an excursion to the Motion Capturing Lab, which will be discussed in Chapter 7, was planned as well.
The fourth week would concentrate on interfaces, like UI, tools for games, audio interfaces and design. In the fifth week the focus would be on 3D graphics, online games, AI and marketing. The last week of the course was planned very loosely and tentatively, because Mr McCallum was considering sacrificing some lecture time so that the students could focus on finishing their games.
Some lectures which were planned at first like a music lecture or a lecture real time computing were cancelled either because the lecturer could not keep the promised target date or because other lectures (like 3D graphics) were extended. The exact lecture plan with the actual lectures of COSC 360 and the tutorials and exercises can be found in Appendix A.
Now when comparing the final timetable with the original suggestions from the IGDA framework, it is evident that not all of its categories could be accounted for. "Games and Society", "Audio Design" and "Business of Gaming" were only covered in brief. More attention was given to formal processes like design documents, project work and game programming.
Through his connection with the NZGDA Mr McCallum also managed to get a few people from the industry involved. Peter Ashford, who is working for a local games company was the one to make the start with a lecture on groupwork and production workflow. Later in the course Mario Wynands[2] and Jon Labrie[3] were going to give additional lectures. This relaxed the learning atmosphere and through the feedback from the industry students can see the actual possibilities where their work can take them in the future.

[2]Managing Director of Sidhe Interactive - who just finished working on Rugby League for Electronic Arts

[3]Former CTO at Weta Digital and managing director of Blister

Chapter 6

Exercises and Tutorials

The tutorials and labs[1] needed to cover a wide scope of topics. The content of the lectures was deepened but also the labs provided the students with the knowledge they needed to create their own game. Since this part of the course was concerned with the student project, the finishing of a self crafted game, it slowly introduced all of the techniques necessary for the creation of computer games.
The first week concentrated on the use of Inform and writing a little text-adventure (see next section), while the second week then focussed on project management, CVS and design documents. The following week introduced the students to all the features of their tools. Most important was the introduction to Microsoft Visual Studio, 3D Game Studio and Maya, because they would need to decide on the tool they would use for their game afterwards.
This meant that from there on resources in the course were split up. As it turned out, half of all students were willing to use 3D Game Studio to develop their game and the other half preferred programming sprite based games in C++ with the help of the Visual Studio IDE. The idea to give the students the freedom of choice with their tool did not turn out to be completely satisfactory. If the students would all have used the same tool they could have discussed similar problems among each other and staff members would only have needed to focus on one development platform. Thus the preparation of teaching would have been much easier. The result was that sometimes the students ended up with problems that no one could find immediate solutions for. One thing that should be learned out of this experience is to always have a test-run with a tool of choice among several staff members for a few months. This way one can gain important information to find a way through the problematic scopes of the tool. It is in my opinion a much better approach to decide on just one development tool for the whole course. Nevertheless the use of two tools for game creation was quite pleasantly handled by the course staff and all the teams could be motivated to finish their projects to a certain level.
The other labs focussed on training the skills necessary for a computer game developer

[1] Labs describes the practical exercises held in the computer science laboratory

with respect to certain aspects similar to those of the lectures. Due to huge range of skills required it was only possible to scratch the surface with each topic.
Mr McCallum was happy to give a broad intense overview of all topics a game developer has to face rather than narrowing down the schedule and concentrating more on certain aspects. Thus a broad overview was more important than a deeply specialised discussion of a certain aspect. As this course progresses in the future years it may very well be possible that certain topics will be picked out and intensified in other courses. All these courses may one day make up a computer game degree but that is still quite a long way to go.
Two exceptional labs should be mentioned here. One lab session took place at a Motion Capturing Laboratory, which is explained in further detail in Chapter 7, and another one was contributed by the theatre studies department. The students had to meet in a gym and explore ways of interacting with each other. One group would try to play and mimic a situation, while the others had to interpret what was going on. The students then got a feel of how to display certain situations and find out which emotional stereotypes are used in play and games as well!
One of the labs would teach the basic use of LaTeX for documentation reasons. The students got an introduction on how to format a basic document together with some information on how to comment their source code. LaTeX is certainly a good tool for scientific documents, but as practice has shown people prefer working with an Office Word Processor. It is much easier to quickly write down ideas and merge documents from team members with such a tool. Therefore LaTeX cannot be recommended as a design document tool in game design. For some students the tutorials, which mostly featured questions and ideas to get into a discussion with the students about the current lecture topic, seemed a bit too philosophical and less forward than they were used to from other computer science areas. After a while they came to appreciate this format of a tutorial.
It was an interesting incident that the game "Manhunt"[2] was the first video game ever to become banned in New Zealand almost at the same time that the impact of computer games on the player was discussed. The mere possession of this game can result in a fine of approximately US $ 1,250 (NZ $ 2,000), and the exposure of it to a person under the age of 18 leads to one year imprisonment or a US $ 12,210 (NZ $ 20,000) fine, the same for retailers and advertisers of the game. This of course sparked discussions if the banning really keeps people from playing the game. The law and the restrictions are different in New Zealand and this topic had been discussed in the computer game classes at Magdeburg University. So the attempt was made to explain how censorship and indexing is dealt with in Germany.

[2]developed by Rockstar Games [http://www.rockstargames.com/manhunt/]

6.1 Introduction to Interactive Fiction Using Inform

The first week of the course would introduce the students to Inform and they would need to create a simple text-adventure for their first assignment. Figure 6.1 shows the final output of a compiled Inform file in an interpreter. The tool looks much like a command line, which works with typed-in commands.

The similarity to a command line environment and the simplicity of the language led to

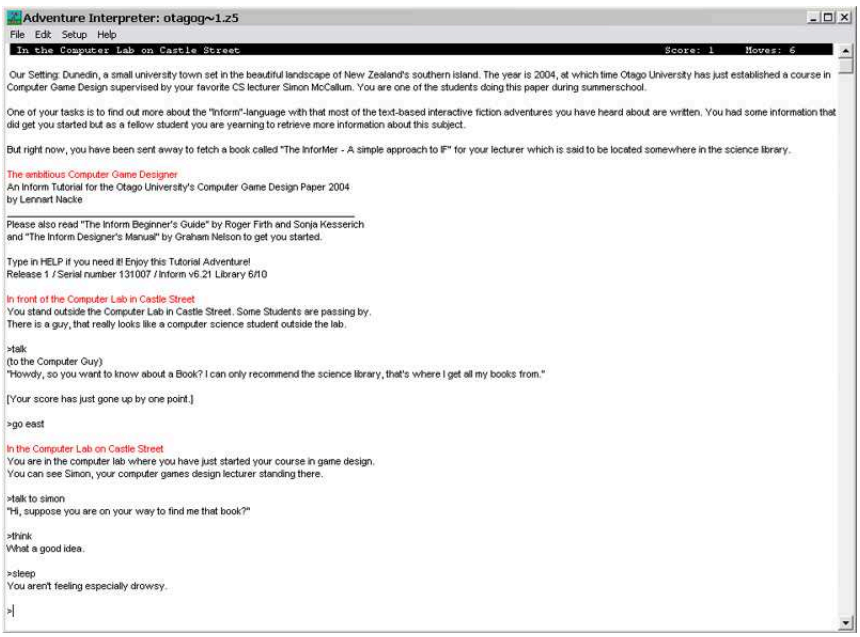


Figure 6.1: Inform assignment in the WZIP interpreter

the decision to use Inform as an easy introduction to computer game design. By doing this the focus is moved away from high end computer graphics. When programming a game in Inform, interaction and creativity come together as very important factors.

With the help of Roger Firth's Tutorials [FK03] a text-adventure template was created that the students had to expand and modify as part of their assignment. The complete code-listing can be found in Appendix D; the assignment in Appendix C.

Inform has C-like syntax but provides object orientation. The language lets you create a story-world in which you can define objects and actions that the player can perform upon objects. Inform recalls the early days, when computer games like "Adventure" or "Zork" were industry standard. Since then games have grown bigger and more graphical up to the point where some games lack story and focus mainly on the graphics. This should not be confused with a number of other games that do not rely so much on story. These games are more focused on skills and player reaction than the immersion of the player.

6.2 Preparation of the Lab Computers in Otago

The computer science department of the University of Otago had reserved a computer science student lab for COSC 360, which normally features Unix Systems, to be reconfigured into a computer games lab. The machines all worked with an Intel Pentium III processor at 797 MHz with 256 MB RAM attached to them. As mentioned earlier the course was designed to be very close to the real industry, so when possible and affordable a configuration that is close to the computer game industry was used.
One of the tools a game programmer has to be familiar with is Microsoft Visual Studio and DirectX. The DirectX SDK can be downloaded from Microsoft's company internet site for free. DirectX integrates very well into Visual Studio. The native system for this compiler is Windows. Thus Windows XP Professional (Version 2002) with Service Pack 1 was chosen as the development platform to work on.
When configuring the computers the attempt was made to include the best freeware available and to get an educational licence for all the tools needed.
The Audio Tools featured Audacity, Cool Edit 2000 and Oggdrop. For graphics Adobe Photoshop 7 with Image Ready 7 and Gimp were installed. The modelling tools described earlier included Gmax and Maya PLE 5. Only the latter was actually used in the course. After the Motion Capturing Session Deep Exploration (Demo with MoCap add-on) was used to easily display the captured data files.The editing and coding environments were TextPad, TeXnic Center, Visual C++ 6,Gnu Emacs and the Inform compiler. MS Office Suite 2003 was also installed (featuring Excel, Word and Project). The game engines that we made available to the students were 3D Game Studio, DirectX 9.0 SDK, Torque Engine Demo, Shark 3D and Game Maker 5.
Last came useful accessories that are needed for general purposes: PUTTY, SSH Secure Shell, Tortoise CVS, WinCVS, FreeMind, MikTex, Ghostscript and GSView, Acrobat Reader and Power Archiver2001.
Some software configuration had to be done with TextPad and the Inform compiler so that it integrated seamlessly.
Problems came up when the settings of the default user profile needed to be saved. Gmax would either quit working or request a new activation for every user (which is free but a lot of hassle). TextPad could not save Syntax and configuration settings in the profile but rather in the TextPad program folder, which ended in writing a tutorial for the inclusion of the Inform compiler into TextPad (it could also be compiled with a batch call, so this did not turn out to be big problem).

Chapter 7

MoCap as an Animation Device in Computer Games

Motion Capturing (MoCap) has become very popular and is appreciated by professional animators because of the realism of the captured movements. It is used for Cut scenes and Animation of 3D models in computer games and graphics. For the course a session at the MoCap Lab of the physiotherapy department was booked. The goal of using MoCap for the course was to show the students the usage of this important tool, the resulting file formats and their processing. The possibility of using MoCap as an alternative to keyframe animation was also considered, but neglected due to the amount of work that would need to be put in to get the animations correctly integrated into the student games.

7.1 Preparations

The MoCap Laboratory contains 12 infra-red cameras arranged in a circle, which are able to track reflective markers. These markers are attached to the body and the computer reconstructs the image from each camera in three dimensions. The more markers attached, the higher the accuracy of the captured movement. The system is very accurate, even with higher speeds, as it allows the adjustment of the frame rate.
For the setup of the test scenes, a storyboard had to be constructed. The first scene was just a test of regular walking through the captured area. The next scene was walking with slightly more intensity to the action, displaying a certain mood. Another scene focussed on a real play situation and involved walking with sudden disturbance of the actor causing him to fall over backwards. This latter scene was quite complicated to arrange as mattresses were needed to cushion the fall and on impact some of the markers would be hidden.
To see the different marker setups a body actor was engaged to do some of the movements, while others were acted out by the author of this report. This just served as a training session for the actor, who was used later in the course to display all the motions

that each student group wanted to capture.

Figure 7.1: Motion Capturing session. Embodiment of a walking ogre.

The picture 7.1 shows an example of a movement capture. During a certain time frame the markers attached to the actor are recorded by the infra-red cameras of the system. After the capture, the data is saved as vertices in three dimensions over time. With these 31 vertices recorded, one still has to figure out which vertex represents which part of the body. The markers are therefore labelled before the capture. The raw data is saved into a text file and contains only a list of sets of numbers together with some header information.

7.2 The TRC File Format

The header of the file contains information about the frame rate, number of markers, etc. A basic example of a trc file header is shown below:

```
DataRate           60.0
CameraRate         60.0
NumFrames          600
NumMarkers         31
Units              mm
OrigDataRate       60.0
OrigDataStartFrame 1
OrigNumFrames      600
```

Following that are columns that display an integer frame number, a float representing the time in seconds from the start of recording and float point x-, y-, z- coordinates for each vertex in combination with their label. The disadvantage of MoCap data is that it is often very unclean after the output and needs to be polished to make it useful for further processing. After recording the raw files contain numerous unregistered points defined by the analysis program as a default model.
The clean up process after motion capturing involves stepping through each frame in the Motion Analysis program, identifying the vertices and making sure they are labelled correctly. This ensures there are no unregistered points left in the files. Because the time at the MoCap lab was limited by the cost of its use (US$ 122 [NZ$ 200] per hour) the decision was made to just use the unpolished raw data in the class session.

The analysis of this data could be done by loading it into a matrix. Dropped frames

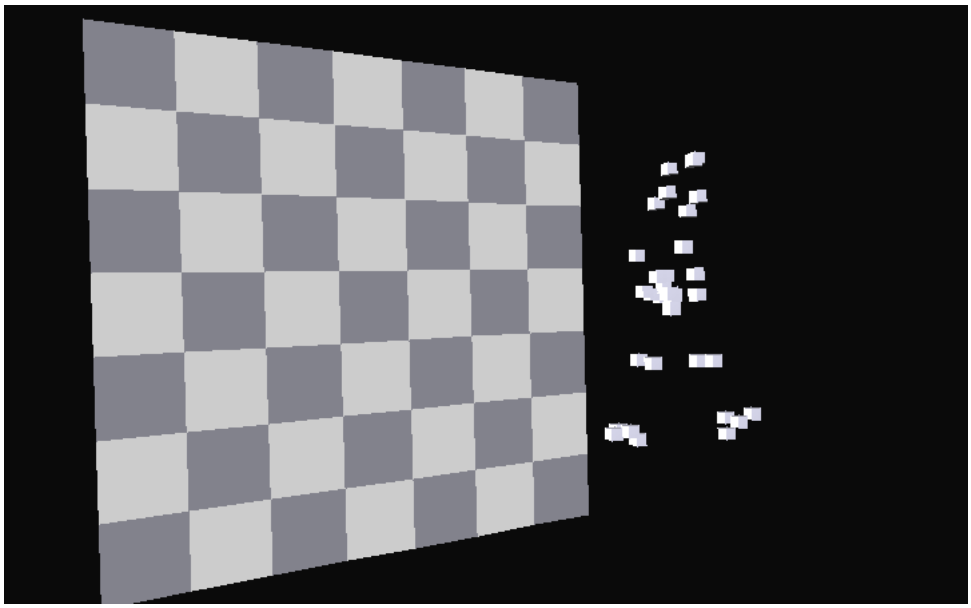


Figure 7.2: View of motion capturing data in Deep Exploration

and unidentified vertices can then be identified more easily. There are several strategies to identify dropped vertices:

1. Minimum distance (Zero order) The basic idea behind this is that vertex A is at time t+1 most likely to be the closest to where A was at time t. This is done for several frames. Now we have a list of minimum distances for each point. A loop goes through this list assigning the lowest distance points first. When this process is complete any points left over are likely to have been missed.

2. Velocity (First Order) This approach assumes constant velocity. Define a time step, then take the difference of a point from its predecessor and divide the value

through the time step. The estimated point is the actual point plus the product of velocity and time step.

3. Acceleration (Second Order) A better estimate than constant velocity is constant acceleration. The change in velocity is important. Given three identified points, we look at the change between points at t-2 and t-1 compared with t-1 to t. Once we calculate the two velocity changes for t-1 and t-2 we can use this to estimate the new velocity for position t+1. With second order equations it is possible to model gravity.

4. Jerk/Cubic functions (Third Order) This refers to a constant change of acceleration. This allows smoothing out human motions. For good results you need four points. It works like the example above but you calculate the rate of change of acceleration. Calculations are three dimensional.

5. Structural integrity Humans have structural integrity in their body parts. If we identify certain body parts in the 3D space we can give estimates on the flexibility of a connection. With these constraints that specify a range where points can go over a certain amount of frames we can clean up the MoCap data.

7.3 A MoCap Manipulation Tool

Mr McCallum's plan during the course development was to let the students write a tool that could visualise and manipulate the MoCap data. It could even have a function to clean up the MoCap data using the methods mentioned above.
During the course it was apparent that the students were quite busy with the work on their game projects. Since this was taking most of their time, one of the tutors provided a java applet that could read the MoCap data and display simple dots in a selected dimension. The students could extend this tool if they wanted to.

7.4 Keyframe Animation and MoCap

One group that was working on a "stickmen" game managed to use some of the MoCap data to apply it to their stickmen in 2D. Regular keyframe animation of painting each sprite frame by hand would have cost them a lot more time. This way they could just attach their geometry to the MoCap data and after cleaning it up a bit, they had very realistic animation.
Not all groups needed this of course. Another group used just regular keyframe animation to render an animation sequence in Maya and then imported all the rendered frames into their game. The games created in Game Studio did not use MoCap data.

Part III

Analysis

Chapter 8

Conclusion

In the following the different components that this course was built with are analysed. From the basic idea of using a framework and available resources to the final course layout it was a long refinement process. COSC 360 was an ambitious project from start to finish.

The idea of giving a very broad overview over all topics of computer games was only

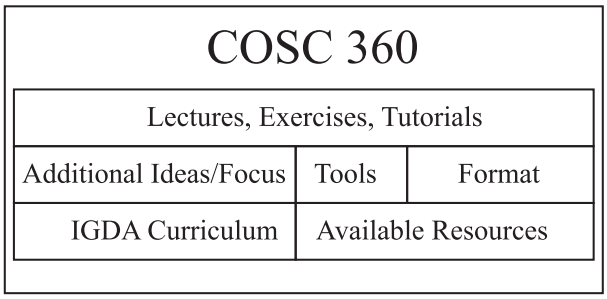


Figure 8.1: The components that constructed COSC 360

partially achieved, as a lot of time had to be spent of teaching programming and group working concepts. The final lectures and labs reflect the focus on computer game programming topics. Although COSC 360 still involved a lot of other areas it was still a characteristic trait that the course audience was made up mostly of computer science students. Thus the focus on the more technical areas of game design was not planned but evolved with the progression of the course as students requested to deepen certain areas more than others.

The format brought along general course concepts that were discussed in Section 2.2. Most of the general course concepts that were planned were put into practice in the end. The working atmosphere that was achieved was to my mind very close to the real computer games industry. Everyday all of the staff had to push their limits to help the students and make them work hard to finish their work in time. Most of the time this included working until late hours. Because of the compact format of summer school it

was better to work in the lab with the students. This way the staff was available for guidance and help around the clock. During the course approximately 11000 working hours were used for 34 students. At the time the students finished their games they had written about 6000 lines of C-Script code and 9000 lines of C++ code.

8.1 Choosing the Right Tools

"Give us the tools and we will finish the job" [Churchill[1]]. The choice of tools is important with regard to their possible field of application. A lot of the tools that were used in the course supported the workflow of game production (like code editors, Office and CVS). Other tools like LaTeX for example were in my opinion not necessary for the course. The introduction to Inform was a good idea, but alternative tools for creating small adventure games could be considered (e.g. Hugo, HTML TADS) for use in the future.
One criticism is the usage of the game engine. As I concluded earlier it is a much better approach to use just one tool throughout the course. Student teams can then be of more help to each other because they are facing similar problems at the same time. Also, the game engine should not only be evaluated but also be field tested for a few months before the start of the course. Sample tutorials to work with the engine need to be selected. 3D Game Studio is an appropriate tool for teaching game design to beginners. Thus, it needs to be assured that all students taking the course have only beginner knowledge of programming. On the other hand 3D Game Studio can be employed as a tool for professionals as well. When the knowledge levels of the students are unequal the more skilled could all be assigned to one team with the task to extend the Game Studio engine via a plugin.
The scalability of a tool is very important. If a professional programming language is used then the risk of inexperienced students not catching up on topics is high. Therefore an introductory course should always provide a tool from which different aspects can be used. In Game Studio some tasks can only be done by experienced users, others are no problem for novice programmers. For the future teaching of this course I would suggest focussing on one tool (e.g. Game Studio) to teach the aspects of game design. For other areas that concentrate more on game programming and where the target audience are experienced programmers (e.g. students in higher semesters) the focus can be on programming tools. But a kind of omnium-gatherum of game engines is to my mind a far too complicated approach to teaching and detracts from the main objective.
On a more positive note, the existence of a user forum, website and CVS to assist student communication is extremely helpful. To my mind this greatly aids the creative process among students and gives immediate feedback to the staff, so problems can be adjusted and unclarities can be dissolved. This kind of communication platform is recommended for any academic course that features student project work.

[1]Quote from Winston Churchill at a BBC radio broadcast in February 9, 1941

8.2 Course Format and Team Projects

A last word about the format of the course: The use of summer school for teaching a subject with a wide scope like computer games is ideal. The division of students into teams assigned to different projects is to my mind also a good approach to handle work as it is done in the "real" industry. Due to the lucidity of summer school a clear deadline is always in sight. Nothing diverts the students' attentions from working on their project and they remain focused on the project until the end of the course.
However, this also brings along the disadvantage of immense pressure when the deadline comes closer and things are not working correctly. Therefore methods to relieve stress need to be thought about and used.
The compact schedule also leaves little time for the course coordinators to prepare the lectures because much of the time is spent on working with the students. It would be best if all lectures would be present and prepared before the start of the course.
In the student projects it is important to assign one person to be responsible for a group (as a project manager or team leader); this makes communication between students and lecturer much easier. In combination with weekly meetings with project managers it offers a good overview of the task progress.
To my mind, this general format is suitable for teaching computer game development in an academic environment.

Chapter 9

Post mortem of the course

9.1 What went right?

9.1.1 High Motivation Among Students

First of all, the most important thing to mention is that all the students were highly motivated when they applied for this course and stayed enthusiastic along the way. In most computer science subjects the application of knowledge remains unseen by most of the public and if a student has solved complex equations to write a great command line tool that can calculate vector maths, people probably will not be as interested as when he shows them a computer game. Computer games, especially those that are programmed in 3D, feature complicated mathematical equations to work correctly. That is a fact, which most people do not realise. Thus it is more rewarding for a programmer to work on game that even his little brother can play to show his abilities.
The same goes for students who may have never programmed interactive graphics before. They find it so rewarding to take part in game programming that they do not mind working late hours, enduring a higher workload and more lab time than in other courses. This seems staggering but computer games is a subject that people are either totally keen on or not interested in at all. Many of the enrolled students were members of the local student chapter of the International Game Developers Association (IGDA).
As a method for stress relief during crunch periods especially close to the end of the course, some game related events were provided aside from the course. The students could join in for LAN games in the evenings, watch movies and have pizza dinners together with some of the staff. They could discuss their ideas and problems if staff was present in the lab. They were also able to reflect upon their game experience in the tutorials and lectures to analyse certain aspects and discuss features of game play and graphics. To give better support, regular project manager meetings were scheduled on Mondays after lectures and students reported the progress of their work and asked for advice.

9.1.2 All Students Finished Their Projects

After the experience from other computer game courses (e.g. the ones at the University of Magdeburg), it seemed like a tough task to get the students to complete a game by the end of the course, which lasted only for 6 weeks during summer school. Luckily almost none of the students attended other courses parallel to this one, which meant they could devote most of their time to work on their game and take the part assigned to them in their teams.
After all, the students decided quite quickly on the way they wanted to produce their game (half of them chose 3D Game Studio [Acknex Engine] 6 as their tool of choice, while the other half programmed a 2D sprite game in DirectX, some of them also used SDL). In the end all teams had at least one playable level done, while one team produced a platform game that reached a very professional stage. Admittedly this was the only team consisting of PhD and honour students.
It was good to see that all of the students in the end had the ambition to finish the product that they had thought about in their design documents at the beginning of the course.

9.1.3 Involvement of Industry People and Guest Lecturers

As the structure of this course was interdisciplinary, it was important to get different departments involved as well as people from the games industry. Jon Labrie, former CTO at Weta Digital and Managing Director of Blister[1], and Mario Wynands, Managing Director of Sidhe Interactive[2], travelled to Dunedin to add value to the course by giving advice from the computer game industry.
Guest lecturers from various departments from the University of Otago talked about specific topics like Game Narrative (English Department) or Game Business (Marketing Department). This extra content for the course was scheduled and the extra value was appreciated by lecturers as much as by students. The fact that all guest lecturers happened to be very interested in this collaboration was not expected.

9.1.4 Teams Stayed Together

When the first decision on members of the teams was made, the approach was to try to fit students with same interests and same skill level together. But as past experiences have shown, most teams run into conflict during crunch periods and when it comes to being credited for common achievement.
The approach to mix students according to their skills rather than to their personal preference was good. As far as possible personal interests were accounted for. Ensuring that all team members put in the same amount of work was the team manager's task

[1] http://www.blister.co.nz
[2] http://www.sidhe.co.nz

and it was well done in all cases. Although some teams experienced difficulties along the way, most of them worked well together and all of them managed to solve problems and stay together until the end of the course.

9.2 What went wrong?

9.2.1 Only Male Students

Although it may sound peculiar at first, one would like the course to include both genders. Computer science, and with that computer games, is still completely dominated by male developers but the industry needs fresh ideas and explore new market areas. As one article on the "Game Girl Advance" Webzine[3] describes the possibilities that lie in computer games that appeal to females as well as males. Making computer game development attractive to the few girls that study computer science should be considered a major issue in future years.

9.2.2 No Optimal Support on Game Studio

Game Studio and its programming language C-Script was new to the staff and although we managed to find a solution for all student problems, it will certainly be an advantage in future years that most of the staff have now already worked with Game Studio and know some of its distinctive features.
If middleware is used for a course for the first time it will always be a problem initially because of the uncertainity of upcoming difficulties among students.

9.2.3 Lecture Plan Too Ambitious

Even though the weekly schedule was already tight, Mr McCallum often could not manage to stay inside regular New Zealand University lecture hours [which last 50 minutes]. The amount of content that was put in the course was perhaps a little excessive given the amount of time available. In general, it was estimated that about twice the amount of time was needed to cope with all the information. Since everything that was considered unnecessary was already left out the scope was possibly a bit too wide for a beginner course.
The focus on certain aspects of computer game development beforehand could help to solve this problem in the future, I think. This could also be solved by maximising the lecture hours but then it is questionable if all given information can be processed by the students. The topics have to be cut to actually offer the right amount of work scheduled for the course.

[3] http://www.gamegirladvance.com/archives/2003/11/07/getting_grrl_gamers_the_future_of_gaming.html

9.2.4 Unplanned Focus on Game Programming

Because this course was designed for students of computer science the average amount of programming problems and software engineering was higher than first considered or known from the courses at the University of Magdeburg. Computer game development is primarily a programming task. This is not a real disadvantage but for a good overview of all areas splitting the course with the design studies department should be considered.

Chapter 10

Personal Remarks

Having spent half a year working on this course, I became very attached to it in the end. I have never experienced so much appreciation for what I was doing before in my life. Working with the students everyday, refining and forming the course to fit all expectations and the curriculum was a stressful task but also a great fulfilment for me. One time we spent until midnight with the other staff to correct and grade the students' design documents. For the correction of the Inform assignment I spent a complete weekend figuring out people's code and trying to find the bugs or the fun factor in their games.

The two months the course was running meant giving up the private life and inhaling a breeze of what the real industry must feel like. It is a lot of work but the work is very satisfactory and enjoyable. This is probably the best way to describe the feeling.

I spent two months of the development phase just researching on the web and telephone about game engines and their costs. This was not pleasant at first because it seemed like it would never end. Every time I had thought I had looked at enough engines more of them appeared. In the end and for this report I decided to only include a few of them because differences were only marginal, especially among the freeware engines.

When I started my internship at the University of Otago the goals and definitions of the computer game course were not precisely clear and merely ideas. After Mr McCallum clarified what he needed to have done by the start of the course it was up to me to decide what my focus would be on.

Setting the focus on the course development was the right choice for me in the end, because after the more theoretical development phase I was glad to be a part of the teaching team and working with students. I really enjoy team work and academic work with students. Being a tutor allowed me to do both. In my opinion this is the great advantage when you do an internship in academia rather than industry.

Since the course was designed to mirror certain industry features I also faced some of the typical challenges such as working overtime, multitasking, time management and stress. As a consequence I felt a bit worn out by the end of the course but also found out how to handle situations where teamwork has to be done under pressure. Thus, for the delivery of the course it felt a bit like working in the industry. Additionally I think I gained a lot

of knowledge about computer game education and the acceptance of this new research and teaching subject.
All in all I would recommend an internship in a foreign country and a young discipline like computer games. You have more freedom than in established sciences. The following section will look closer at the cultural differences in the research that I carried out.

10.1 Game design education - A comparison of New Zealand and Germany

After having experienced education in the area of computer game design in Germany as well as in New Zealand I will try to compare the aspects that are distinctive to the countries. I also have to mention that I experienced two different sides of the same subject. In Germany, I attended computer game classes as a student whereas in New Zealand I was co-developing and delivering the course as part of the staff. This little difference aside, I think I can compare the education in both countries.
In Germany computer game design education is already a departmental study group and offers several courses on the subject, while the course I developed is one of the first game courses in New Zealand. The University of Magdeburg introductory course deals a great part with licensing, legal issues, censorship and violence in computer games because these subjects are part of German computer game culture. In contrast, COSC 360 focussed a lot on software engineering techniques and only scratched the surface of censorship discussions.
The second computer game algorithms course at the University of Magdeburg is more practical but concentrates on analysing and using the features of a game engine [Shark 3D]. It is more important to learn how things are working together than finishing a final product. The Otago course teaches more programming and was directed entirely to the goal of having a game prototype in the end. The students had time to think about their ideas and write a design document which they were going to produce. The division into projects and groups that worked together for the whole course with different tasks assigned to all team members was a key feature here.
Unfortunately the Magdeburg course is a regular semester course and the students participating in it often take concurrent courses of equal or more importance during the same semester. This format makes it harder for the students to constantly work on the game. They have to write a design document, but since they have just a portion of time in the week for it and no assurance if they will work on the game that they are proposing, these documents are only a few pages and not really useful for the creation of a computer game.
The course in Magdeburg offers an additional modelling tutorial course before it starts, which gives the students good insights into a modelling program and allows them to learn about a lot of 3D features. This is a great addition to the regular game algorithms course. On one hand the Otago course just quickly gave an overview of the most important Maya functions but did not really go into the depth of 3D modelling. On the

other hand much better tutorials were offered on key features of MS Visual Studio and DirectX. This underlines that the Otago course concentrated more on applied computer science aspects rather than artistic features.
It is quite hard to differentiate which approach is the best. Often the decision on what key aspects a computer game curriculum focuses on, and depends on the teaching already available at the University. In the case of the University of Otago this was definitely computer science. Thus, COSC 360 has its main focus on programming.
In contrast, the computer graphics focus in Magdeburg is the one which contributes most to the departmental computer game study group. This focus is more on the graphics aspects of games, which is slowly changing because more people are joining the group with interests in other topics like AI. Mr McCallum, who developed the Otago course, is an AI lecturer which explains his major interest in non-graphic-related game areas (especially AI).With the lecturers from the design department the graphics part was taken care of from a non-computer-science point of view. This not the best approach but worked well for the Otago course.
Both courses are good, even though I think a format similar to summer school could be a great addition to German academia, because students then have the time to intensely focus on completing a computer game and organising themselves to achieve that goal.

10.2 Future prospects: The New Zealand Game Developers Conference

During the time that we spent working on the course, Mr McCallum had the idea to take computer game development and education a step further than just teaching a course at the University of Otago. During intensive talks with members of the NZGDA, especially with Mr Mario Wynands, he began developing New Zealand's first game developers conference. Computer game experts across New Zealand directed their interest to this idea and soon we began meetings with University staff to plan the conference. I was glad to help out a little bit in the planning of the conference.
While we were all working together on a corporate layout for this conference, we had to bring a website with information online and start to confirm keynotes. Surprisingly, both authors of the text-books that we used in the course soon confirmed as keynotes. With big names like Ernest Adams[RA03], Jon Labrie and Daniel Sanchez-Crespo[SÓ3] the conference grew more attractive. I programmed a PHP Newsmailer for the NZGDC website and used most of the PHP functions from the COSC 360 website to program an easily modifiable navigation.
The conference was soon named fuse and will take place at the University of Otago in Dunedin from the 26th - 29th of June 2004. We managed to get some publicity and were even mentioned on the front newspage of Gamasutra. The IGDA and NZGDA said they would support it and the focus was clearly on the gaming industry in New Zealand, which has an excellent reputation in other fields of creativity like film and animation. The development of local content and establishing New Zealand games as a brand of its

own is a major intent of the conference.
The conference is designed to feature lectures, workshops, an IGDA student room and extra services. It was very exciting to attend all the preparations that were made for this big event and help search for funding possibilities. After I left New Zealand I received the news that the New Zealand government will provide a substantial amount of funding for the conference.
This conference will put a spotlight on game development in New Zealand and make the area for future students more attractive which, in turn, will hopefully solidify the courses at University of Otago and ideally lead to the development of degree program for computer games.

Bibliography

[Con97] Matthew J. Conway. *Alice: Easy-to-Learn 3D Scripting for Novices.* PhD thesis, University of Virginia, School of Engineering and Applied Science., 1997.

[Cra03] Chris Crawford. *Chris Crawford on Game Design.* New Riders Publishing, Indianapolis, Indiana, USA, first edition, June 2003.

[Cur92] Pavel Curtis. Mudding: Social phenomena in text-based virtual realities. In *Proceedings of the 1992 Conference on the Directions and Implications of Advanced Computing*, Berkeley, CA, 1992.

[DeL00] Mark DeLoura, editor. *Game Programming Gems*, volume First. Jenifer Niles, Charles River Media, Inc., 2000.

[FK03] Roger Firth and Sonja Kesserich. *Inform Beginner's Guide.* Interactive Fiction Library, second edition, February 2003.

[FMS01] Bert Freudenberg, Maic Masuch, and Thomas Strothotte. Walk-through illustrations: Frame-coherent pen-and-ink style in a game engine. In A. Chalmers and T.-M. Rhyne, editors, *EG 2001 Proceedings*, volume 20(3), pages 184–191. Blackwell Publishing, 2001.

[IGD03] IGDA Education Committee. Curriculum framework, February 2003. IGDA Curriculum Framework - The Study of Games and Game Development, Version 2.3 beta, February 25, 2003.

[Juu99] Jesper Juul. A clash between game and narrative. Master's thesis, IT University of Copenhagen, Copenhagen, Denmark, February 1999. A thesis on computer games and interactive fiction, http://www.jesperjuul.dk/thesis.

[Ker00] Isaac Victor Kerlow. *The Art of 3-D Computer Animation and Imaging.* John Wiley & Sons, Inc., second edition, 2000.

[KR88] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language.* Prentice Hall PTR, second edition, 1988.

[Lam94] Leslie Lamport. *LaTeX - A Document Preparation System - User's Guide and Reference Manual.* Addison Wesley, 1994.

[LaM02] André LaMothe. *Tricks of the Windows Game Programming Gurus*. Sams Publishing, Indianapolis, Indiana, USA, second edition, June 2002.

[Lis02] Konrad Lischka. *Spielplatz Computer - Kultur, Geschichte und Ästhetik des Computerspiels*. Verlag Heinz Heise GmbH & Co KG, Hannover, Germany, 2002.

[Mak03] Jobe Makar. *Macromedia Flash MX Game Design Demystified*. Peachpit Press, first edition, 2003.

[MP03] Jessica Mulligan and Bridgette Patrovsky. *Developing Online Games - An Insider's Guide*. New Riders Publishing, first edition, March 2003.

[Nel01] Graham Nelson. *The Inform Designer's Manual*. Interactive Fiction Library, fourth edition, August 2001.

[Ove89] Mark H. Overmars. Computational geometry and its application to computer graphics. In *Advances in Computer Graphics*, pages 75–107, 1989.

[Pet03] Thomas Petchel. *Java 2 Game Programming*. Premier Press, first edition, 2003.

[RA03] Andrew Rollings and Ernest Adams. *Andrew Rollings and Ernest Adams on Game Design*. New Riders, Indianapolis, Indiana, USA, first edition, May 2003.

[Ruc03] Rudy Rucker. *Software Engineering and Computer Games*. Pearson Education Limited, first edition, 2003.

[SÓ3] Daniel Sánchez–Crespo Dalmau. *Core Techniques and Algorithms in Game Programming*. New Riders Publishing, first edition, September 2003.

[Sal00] Marc Saltzmann. *Game Design*. X-Games, München, Germany, second edition, 2000. German translation the book Game Design, Secret of the Sages.

[SZ03] Katie Salen and Eric Zimmerman. *Rules of Play: Game Design Fundamentals*. MIT Press, 2003.

[WP01] Alan Watt and Fabio Policarpo. *3D Games Real-time Rendering and Software Technology*. Addison-Wesley, first edition, 2001.

[WP03] Alan Watt and Fabio Policarpo. *3D Games: Animation and Advanced Real-Time Rendering: 2*. Addison-Wesley, first edition, 2003.

Declaration

I declare hereby, that the present work was created independently and only with allowed appliances.

Magdeburg, the June 9, 2004 Lennart Nacke

Appendix A
Tables and Schedules

Game SDK	**Pricing Options**
Shark 3D	free for educational use but must sign NDA[1]
Game Studio 3D	US $399 per 30 students Special deal US $599 for 100 students First option: US $13.30 per student NZ$21.74 (rate 0.611889, 28.10.03) Second Option: US $5.99 per student (NZ$9.80)
Auran Jet	AU $99 per student (NZ$114) Booklet & Manual, developer forum access Connection to one of our teaching fellows
Torque	US $2.250 for 50 students CVS access, developer forum access Comes down to US $45 (NZ$74) per student

Table I: The pricing options for commercial game engines

Days	**Lecture Topics**	**Tutorials**	**Lab Exercises**
Monday	Preliminary lecture		
Tuesday	Introduction and history	Discussing your favourite game	Lab introduction
Wednesday	Definitions - What is a game	ZORK vs. PONG, Inform	Inform Assignment
Thursday	Meaningful Play		CVS Instructions, LaTeX
Friday	Moved to 16th		

Table II: First week - topics overview (January 5 - 9, 2004)

Days	**Lecture Topics**	**Tutorials**	**Lab Exercises**
Monday	Group work, CVS, doxygen and communication skills [Peter Ashford]	Group dynamics, roles	Project Management
Tuesday	Development cycle - Introduction to design documents		Design Document - Brain storming
Wednesday	2D graphics, sprites and collision detection		2D Game Programming in Visual Studio
Thursday	Middleware and Game Engines	Middleware Good or Bad?	
Friday	Narrative [English department]		

Table III: Second week - topics overview (January 12 - 16, 2004)

Days	**Lecture Topics**	**Tutorials**	**Lab Exercises**
Monday	C++, Windows, DirectX	Selecting features of language	Game Studio 6
Tuesday	Visual design		Introduction to Maya
Wednesday	Storyboarding and prototyping		Storyboarding
Thursday	Animation		Game Script Assignment 2, MoCap Lab
Friday	Games environments [Film and media studies]		

Table IV: Third week - topics overview (January 19 - 23, 2004)

Days	**Lecture Topics**	**Tutorials**	**Lab Exercises**
Monday	User interface design	Possibilities of immersion (Dangers and Opportunities)	Writing MoCap clean up and using animation previewer
Tuesday	Immersion	UI and Immersion	MoCap Manipulation [including JAVA Parser and Viewer]
Wednesday	Tool Design		Interface design in Game Studio
Thursday	Audio Design	Hand back marked assignments, discuss your game	Audio editing, Internal Project Management
Friday	Mobile Games [Jon Labrie]		

Table V: Fourth week - topics overview (January 26 - 30, 2004)

Days	**Lecture Topics**	**Tutorials**	**Lab Exercises**
Monday	3D graphics (part 1)		Open GL, Direct X
Tuesday	3D graphics (part 2)	Evil 3D Math	Work on Game
Wednesday	Networks, online games and communities		Online Games - Virtual Communities
Thursday	AI for games	AI and Online games issues	Simple AI agent- state machine - hunting
Friday			

Table VI: Fifth week - topics overview (February 02 - 06, 2004)

Days	**Lecture Topics**	**Tutorials**	**Lab Exercises**
Monday	Advanced AI		Network games - hang man, java gamelets
Tuesday	Game Business [Mario Wynands]	AI, Work on Game	Client server MUD
Wednesday	Marketing computer games [Phil Osborne]	Work on Game	Work on Game
Thursday	How to write a post-mortem	Work on Game	Work on Game
Friday	Work on Game	Work on Game	Hand in Game

Table VII: Sixth week - topics overview (February 09 - 13, 2004)

Appendix B
Student Games

This Appendix gives a short overview of the games that were produced during the computer game design course COSC 360 at the University of Otago. All produced games featured a documentation, a high concept and a design document. A post-mortem was written on all games during the final examination of COSC 360. Most of the projects were very ambitious but the final outcome was overall very pleasant.

1. **Trash Wars**

Genre:	First Person Shooter / Turn Based Strategy
Number of Players:	Single
Tool:	3D Game Studio
Graphics:	3D Models (textured)
Extra Tools:	3D Game Studio
AI:	No
Team:	Thomas Harwood-Stevenson,Jason Clutterbuck, ShiChang Hou, Suhel Mangera and Peter Suggate

This game's idea is to combine first person shooter action with turn based strategy. The story is set up in a dark urban environment, where the mafia is omnipresent and the crime rate is high. The backstory had a few inconsistencies in the development of the players. But the group planned to have multiple players available. One of the player goals was to go and collect the garbage in the city and deal with occurring crime. Sadly one of the main programmers became ill during production which slowed down the development process so that in the end the game did not really make a finished impression. The idea of the game was good and the team spent a lot of time on the backstory, but they were also too ambitious with that. It is probably more suitable to be developed in a longer time span and maybe with a different tool than Game Studio.

Figure I: Trash wars

2. **Waldrick**

Genre:	First Person Adventure
Number of Players:	Single
Tool:	3D Game Studio, Photoshop
Graphics:	3D Models (textured)
Extra Tools:	3D Game Studio
AI:	Yes
Team:	Cliff Hawkins, Lincoln Johnston, Josh Garner and Marcus Ford

In this action game, the player becomes a fish named Waldrick. On his journey through the ocean Waldrick has to fight with other fish, acquire special skills and solve some puzzles. The game is set up to be a first person action adventure from not-so-regular point of view. The adventure elements are mainly the character development and improvement of skills that are available to the fish avatar after fights. The fights are turn-based which reminds more of role-playing games. But the fighting system is implemented precisely and also features a combo fighting mechanism which immediately results in fun gameplay. The graphics are very cute and the game features a great in-game interface. It was one of the first student games that reached a playable phase.

Figure II: Waldrick - a fish game

3. **Revolutions**

Genre:	Real-time Strategy
Number of Players:	Single or Multiplayer / Networking not finished
Tool:	Microsoft Visual Studio, SDL, C++
Graphics:	Sprites, Partially rendered in Blender
Extra Tools:	None
AI:	Yes, very simple
Team:	Tim Elder, Jeremy Folland and Dale Smith

This game is set up to be a space-based strategy game in the future. One of the game's goals is to coordinate resource management in between planets and the player fleet. All space travels feature a timing mechanism which is important for the real-time gameplay. The game also features a model for space combat where fights take place between fleets. A very ambitious plan was to allow the game to be real-time in a network environment, so that different human players can join together in a local area network. This feature took more time to program than the group had estimated and was not finished by the end of the course. It was programmed with use of SDL and DirectX.

Figure III: Revolutions - a space strategy game.

4. **Haggard War**

Genre:	Third Person Turn-Based Strategy
Number of Players:	Single, but Networking planned
Tool:	Maya, 3D Game Studio
Graphics:	3D Models (textured)
Extra Tools:	3D Game Studio
AI:	Yes, very simple
Team:	Chris Gillum, Jeff Laird, Jeffrey Lloyd, Josh Manning and Brett McConachie

The idea behind the game was to program a third person, turn-based strategy shooter-like game. The story evolves around an alien invasion on planet earth and of course the player has to control the last line of defence of human soldiers (or contrary the alien invaders). For good gameplay both races and their units would need to be carefully balanced and the team planned to have many different units available for the player.
The player is involved in the story of defending the earth against the aliens and as the game progresses the story evolves further. Four levels were finished (more or less). The game adds gory effects such as big guns and cartoon-like blood splatter. But dead units disappear due to performance issues. The overall play feeling is like a pen-and-paper-based war game.

Figure IV: Haggard war - a combat strategy game.

5. **Monkey Commandos**

Genre:	Platform Game
Number of Players:	2-3
Tool:	Microsoft Visual Studio, Direct X, C++
Graphics:	Sprites, Pre-rendered in Maya, High Quality
Extra Tools:	Level Builder
AI:	None
Team:	Billy Chang, Phil Mcleod, Andrew Rowse and Liam Tze Vui

This game is a classic two-player platform jump-and-run, that also involves puzzle elements. It needs a minimum of two players to run and does not work in a single player mode or without a network. There are also not enemies or AI characters in the game. But the game features a co-operative mode, where two avatars (controlled by human players) can interact to solve a puzzle. Most of the puzzles consist of stepping on switches to open doors and climbing and jumping through different areas to reach the end of the level. The group managed to have a level builder ready for the game, so that they could spend the end of the course focusing on gameplay and finishing some puzzle levels. The graphics were pre-rendered in Maya (most of them feature key-frame animation), so that they looked quite professional. The game was of the ones that were very complete by the end of the course.

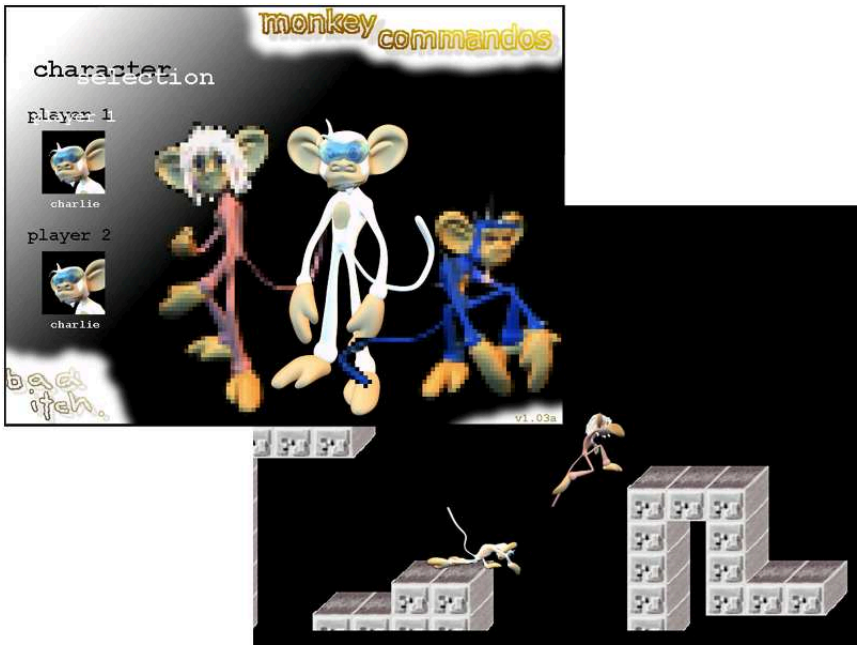


Figure V: Monkey commandos - a jump-and-run game.

6. **Pharmacon**

Genre:	Isometric Action Game
Number of Players:	Single
Tool:	Microsoft Visual Studio, Direct X, Photoshop
Graphics:	Isometric Pixel Sprites
Extra Tools:	Level Editor (incomplete)
AI:	Yes
Team:	Grant Mark, Aidan Fraser, Brandel Zachernuk and Elliot Pahl

A very different idea for a game was created by this group. Their game bases on the idea that the player needs to cure the sick. It is very much an action game, because the player is operating in a town of sick people where the disease spreads fast among the citizens. The approach to heal instead of kill people is an interesting basis for this colourful isometric shooter game. The very bright colours give high contrast to the action that takes place. The project was very ambitious and even though the game featured all needed components it was not quite finished at the end of the course. To a final note, the core team has joined together in game development start-up company called Straylight Studios in Dunedin and they are still working on this title.

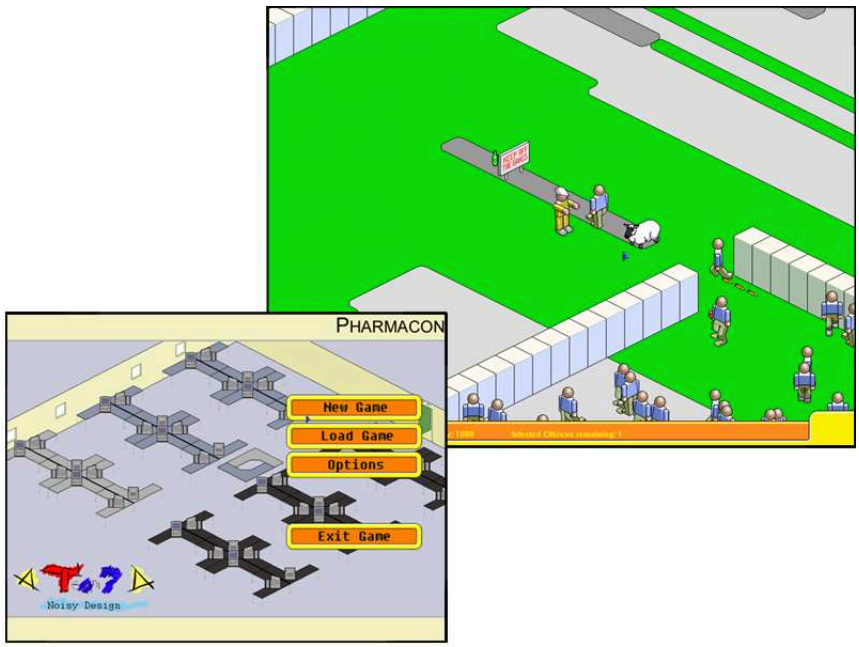


Figure VI: Pharmacon - a colourful yet dark isometric world.

7. **Stickmen 2146**

Genre:	Platform Game
Number of Players:	Multiplayer
Tool:	Microsoft Visual Studio, SDL, C++
Graphics:	Sprites, Pixels
Extra Tools:	None
AI:	None
Team:	Thomas Chan, Andrew Cowan, Matthew Jenkin and Tim Jones

A simplistic approach to a multi-player shooter in the tradition of Quake 3 Arena. No 3D models but some MoCap data was used for the completion of the stickmen animations. The game is more of a jump-and-run shooter, because all of the action is 2D. The cartoon animations are quite an interesting diversion from the photo-realistic shooters on the warehouse shelves. An interesting feature is the chat on click, where the stickmen can exchange little messages while shooting. The game is designed to only work as multi-player, so it features no additional AI. The network support was finished by the end of the course and the state of the game was also very playable.

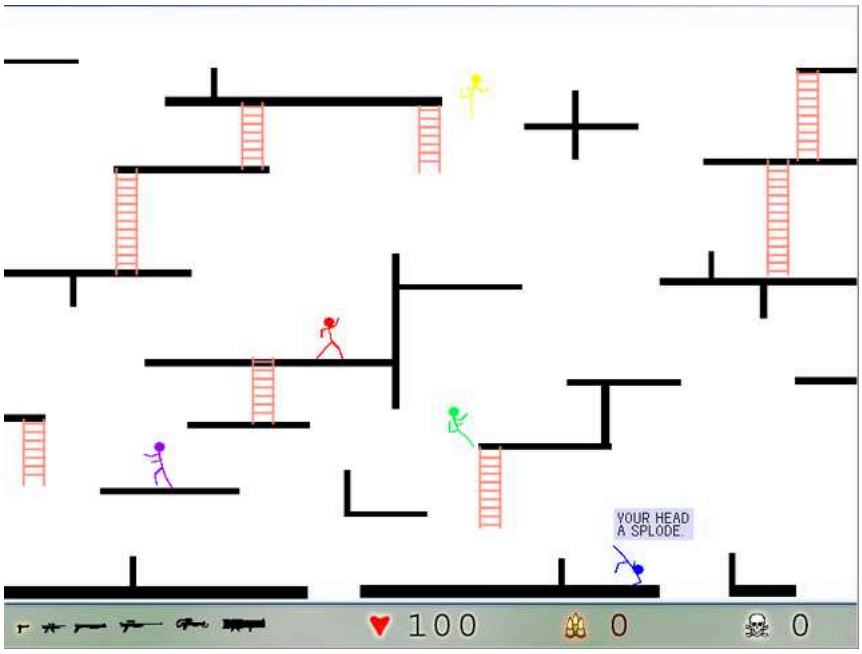


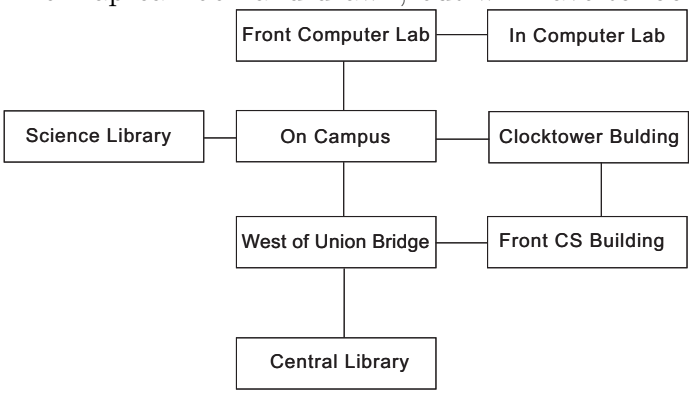
Figure VII: Stickmen 2146 - a simple but very enjoyable shooter.

Appendix C
Assignments

1. **Inform Assignment [worth 10% of the final grade]**

   (a) Take a look at our inform game file *cosc360.inf* [download from http://cosc360.otago.ac.nz/download/cosc360.inf]. Draw a map of the game's locations. In Inform, Locations are referred to as rooms. Rooms again are just Objects. An Inform game therefore is just a series of objects.
   Write down, how an object definition in infom looks like, especially explain what the keywords *with* and *has* declare.

Solution (as given to Tutors):
The map can be hand-drawn, but will have to look at least like this:



Object definition looks like this:

```
Object object_id "reference_name" parent_object_id
  with property value, ...
   has attribute_1 attribute_2 ... attribute_n;
```

with is for object properties
has is for object attributes

(b) Take our predefined class `Room` here

```
Class   Room
  has   light;
```

This class is used to set up cells or rooms (places to go) on a map, that are open air and therefore have the attribute `light` enabled. Use this class to expand the map with at least four more cells, which fulfill the following prerequisites:

- two cells are connected to each other
- you should be able to walk into the central library
- the unavailable exits have a suitable description why one cannot go there

---

Solution (as given to Tutors):
Could be like 4 rooms this:

```
!At least one room connected to the library
 Room  central_lib_inside "Inside the Central Library"
    with  description
          "You are inside the Central Library.
           Here the students should elaborate the interior.",
        n_to central_lib,
        cant_go "You cannot go here, because something prevents
          you from going this direction (be creative)";

        !Three more rooms like the above but two have to
        !be connected via n_to, w_to, etc.
```

(c) Look at the section:

```
Include "Grammar";
[ TalkSub;
    if (noun == player) print_ret " ... ";
    if (RunLife(noun,##Talk) ~= false) return;
    print_ret " ... ";
];

Verb 'talk' 't//' 'converse' 'chat' 'gossip'
    * 'to'/'with' creature          -> Talk
    * creature                      -> Talk;
```

Write a similar routine called `RetrieveSub` for use with the verbs *retrieve, detect, ferret out, trace* or other ones with similar meaning that you can think of. This routine has to do whatever you think would be adequate for this verb (the further usability will be judged by your tutors however). If you use retrieve with your book object, it has to provoke a certain reaction with the player. What this reaction looks like is left to you.

Solution (as given to Tutors):
This would be an adequate Subroutine:

```
[ RetrieveSub ;
      if (noun in player)  print_ret "It is in your inventory
                                        already.";
      if (RunLife(noun,##Retrieve) ~= false) return;
      if (noun==player) print_ret "Why would you try to
                                   retrieve yourself?";
      if (noun has animate) print_ret "Impossible!";
      else if (noun in location) move noun to player;
         print_ret "You have retrieved the item.";
];
Verb 'retrieve' 'detect' 'ferret out' 'find'
    * 'from'/'with' multi       -> Retrieve
    * noun                      -> Retrieve
    * noun 'from' creature      -> Retrieve;
```

(d) Use the class `Prop` or define your own class for at least two different books the player should be able to borrow from the central library. Implement a new NPC named `librarian` using our `NPC` class, who the player should be able to talk to and get the book objects from. Locate this NPC in the central library. For each book the player receives the score has to be incremented by two.

Solution (as given to Tutors):
Best is really to define an own class for the books:

```
Class  Book
with   times_read 0,
has    ;
```

Then a book would be minimally implemented like this:

```
Book  book_id   "Inform Book Template"   central_lib_inside
with  name      'inform' 'template' 'book' 'infb'
                'infbook' 'tempbook' 'ibt',
      describe  [;  if (self in central_lib_inside) rtrue;
                ],  !no separate listing

has;
```

Our NPC would use the NPC class and look like this:

```
NPC   librarian "Librarian" central_lib_inside
with  name 'librarian' 'lib' 'libra' 'booky' 'bookworm'
           'assistant' 'receptionist' 'staff' 'l' 'person',
      description
          "This is normal Librarian as the ones you know from
           real life",
      describe [; if (self in library) rtrue;
               ],
      tsp_to 0,    !times spoken to
      life [;
             Talk:
                  self.tsp_to = self.tsp_to + 1;
                  switch (self.tsp_to)
                  {
                     1:  print_ret "~What is it?~";
                     2:  move book_id to player;
                         score = score + 2;
                         print_ret "~There is your first book!~";
                     3:  move book_id2 to player;
                         score = score + 2;
                         print_ret "~There is your second book!~";

                     default:  print_ret "~How can I help you?~";
                  }
             !Retrieve: !you could define some retrieve actions
                        !here as well
      ],
has   ;!either female or male or neuter, can also add proper
```

(e) Change the game's goal. When the player scores more than 5 points he/she should instantly win (setting `deadflag` to 2).

Solution (as given to Tutors):
One way to do it, is to check every time player moves to a room

```
Class  Room
with   each_turn [; if (score >= 5) deadflag = 2;
                 ],
has    light;
```

A much better (and more accurate way) of doing this is to check TimePasses:

```
[ TimePasses; if (score >= 5)  deadflag = 2;
  return false;
];
```

(f) Write down a storyline with an idea for your own inform game. Clearly define, what situation the player is in and what goals he/she wants to achieve. What obstacles are there? How many rooms and NPCs do you need? How much do you want to extend the grammar library to suit your needs? Think about these issues and write down your concept. It should at least be two pages long and clearly give an idea about a story-based game that makes a user experience last about 20-30 Minutes. Do not get into too much detail.

Solution (as given to Tutors):
This is set up as a creative task for the students to think about the story that drives a game. It is important to create interesting NPCs as much as achievable and attractive goals. They should also consider interactive extensions (such as new verbs) and what functions are behind those verbs.

(g) Make a party mod for our tutorial game. One should be able to explore the area in north of castle street and participate in a student party. You choose the objectives but have to provide a solid documentation to your modification as well. This task is on a total non-obligational basis.

Solution (as given to Tutors):
This is a playful task for students interested in practicing their skills in Inform. Points are awarded for creativity and programming style.

---

2. **Tutorial: Introduction to LaTeX**

(a) Look at our LaTeX tutorial on the COSC 360 Webpage. You need to write TeX Commands into a .tex file with a plain text editor (it just works like a programming language) and compile that file to get your final document. Our Machines are setup so that you can easily access TeXnicCenter from the programs menu under windows. This IDE provides a very good help function as well (you should use it!).
You can render LaTeX files into device independent file (dvi) files as well as postscript (ps) or portable document (pdf) files.

(b) A basic file looks like this:

```
\documentclass{article %if you want it to be an article
\usepackage[dvips]{graphicx} %for using graphics in dvi,ps
\begin{document}
   \author{NAME} %insert your name here
   \maketitle %creates a title
              %here are the contents of your file
\end{document}
```

With the `\documentclass` you define a style for your document. There are quite a few options to format your text with LaTeX that way. The keywords `\begin{document}` and `\end{document}` define your document environment. With `\maketitle` , you basically quickly write a title to your document. There are multiple arguments that can define your title.

(c) To include graphics into your document, you need to include `\usepackage[dvips]{graphicx}` before you start your document and then in your document, you will need to create a figure that contains your image file (looks much better). The following code shows you how to include a graphics file:

```
\begin{figure}[htbp]
  %You need to know WIDTH and HEIGHT
  %of your image file for the bounding box
     \includegraphics[bb= 0 0 WIDTH HEIGHT]{PATH_TO/FILE}
       \caption{WRITE CAPTION HERE}   %optional
       \label{fig:figurename}         %optional
\end{figure}
```

This gives you a nice overview of an image figure that is labelled and can be referenced in your text.

(d) Now look on the webpage and the LaTeX links there. Try to create the following:

- A text with sections
- A reference in the text
- An image figure, which should be referenced in the text
- A 2x2 table

Appendix D
Code Listings

1. **Abridged contents of index.php**

```
<?
require("inc/dbc.inc.php");   //database settings
require("inc/functions.inc.php"); //functions for the site
//define title
$TITLE = "COSC360 - Computer Game Design | ".ucfirst($_GET["show"]);
?>
<!-- DOCTYPE DECLARATION LEFT OUT HERE --->
<html xmlns="http://www.w3.org/1999/xhtml">
<html><head><title><?echo $TITLE;?></title>
<? //function to draw meta tags I left out here
 //drawMetaTags();  ?>
<link rel="stylesheet" type="text/css" href="360.css" title="360 Style">
</head>
<body>
<div id=titlebox><!--Title Image Box-->
<table width = "100%">
<tr>
<td align="left" rowspan=2><a href="http://www.otago.ac.nz">
 <img src="images/unilogo.gif" alt="Otago University Logo"></a></td>
<td align="center" valign="bottom">
 <img src="images/cosc360.gif"
      alt="COSC360 - Computer Game Design"></td>
<td align="right" rowspan =2>
<a href="http://www.cs.otago.ac.nz/">
<img alt="Computer Science logo" src="images/penguinsmall.gif"></a></td>
</tr>
<tr>
<td><?  //THIS IS WHERE MY NAVIGATION SCRIPT STARTS:
//==print==menu==================
$r=selectFromDB("links","*",NULL);
```

```
while($tmp=mysql_fetch_array($r)) {
if(strlen($tmp["linkname"])>3) $linkz[]=ucwords($tmp["linkname"]);
else $linkz[]=strtoupper($tmp["linkname"]);
$id[]=$tmp["sublink_id"];
}
mysql_free_result($r);
menu($linkz,$id);
//==print==submenu===============
if($_GET["submenu"]) {
//select only the assigned links from the sublinks table
if($_GET["submenu"] == 3) $adder="";
else $adder=" ORDER BY sublinkname DESC";
$r=selectFromDB("sublinks","*","WHERE sublink_id =".$_GET["submenu"].$adder);
while($tmp=mysql_fetch_array($r))
$link[]=$tmp["sublinkname"];
mysql_free_result($r);
submenu($link);
} //END OF NAVIGATION SCRIPT
?></td>
</tr>
</table>
</div><!--End Title box-->
<div id=bodybox><!--Main Content-->
<?//This inclued all items as an inc.php file
//Makes finding the files all easy
if($_GET["show"]) {
if($_GET["subItem"])
include("inc/".$_GET["show"]."/".$_GET["subItem"].".inc.php");
else
include("inc/".$_GET["show"].".inc.php");
}
else  include("inc/intro.inc.php");
?>
</div><!--End Main Content-->
<div id=imagebox><!--University Logo--></div>
<!--footer left out here-->
</body>
</html>
```

2. **Abridged contents of functions.php**

```
<?
//==WALKS==THROUGH==ARRAYS==FOR==THE==DB==============
function printWalkThroughArray($ARRAY,$DECIDE) {
for($i=0;$i<count($ARRAY);$i++)
if($i==count($ARRAY)-1)
if($DECIDE=="VALUES")
if ($ARRAY[$i]=="NULL") $txt.="NULL";
else  $txt.="\"".$ARRAY[$i]."\"";
else
$txt.=$ARRAY[$i];
else
if($DECIDE=="VALUES")
if ($ARRAY[$i]=="NULL") $txt.="NULL, ";
else  $txt.="\"".$ARRAY[$i]."\", ";

else
$txt.=$ARRAY[$i].", ";
return $txt;
}

//==WALKS==THROUGH==2==ARRAYS==FOR==THE==DB==UPDATE=======
function printUpdateSet($ARRAY_A,$ARRAY_B) {
for($i=0;$i<count($ARRAY_A);$i++)
if($i==count($ARRAY_A)-1) {
if($ARRAY_B[$i]=="NULL") $txt.=$ARRAY_A[$i]." = NULL";
else $txt.=$ARRAY_A[$i]." = \"".$ARRAY_B[$i]."\"";
}
else {
if($ARRAY_B[$i]=="NULL") $txt.=$ARRAY_A[$i]." = NULL, ";
else $txt.=$ARRAY_A[$i]." = \"".$ARRAY_B[$i]."\", ";
}
return $txt;
}

//==BASIC==SELECTION==OF==DATA==FROM==THE==DATABASE=======
function selectFromDB($dbtable,$colHeaders,$W_condition) {
//the column header names of the dbtable
if(is_array($colHeaders))  $what=printWalkThroughArray($colHeaders,NULL);
else $what=$colHeaders;

$select = "SELECT ".$what." FROM ".$dbtable;
```

```
if($W_condition) $select.=" ".$W_condition;
$rslt=mysql_query($select)
      or die ("Codedump: ".$select."<br> Unknown Command: ".mysql_error());
return $rslt;
}

//==================DELETE==FROM==DATABASE=========================
function deleteFromDB($dbtable,$deletingCondition) {
$delete = "DELETE FROM ".$dbtable." WHERE ".
           $deletingCondition;//e.g. ID = \"10\"
$rslt=mysql_query($delete)
      or die ("Codedump: ".
              $delete."<br> Unknown Command: ".mysql_error());
return $rslt;
}

//==BASIC==INSERTION==IN==THE==DB==================
//this also creates a codedump if not working
function insertInDB($dbtable,$colHeaders,$insertVals) {
$what=printWalkThroughArray($colHeaders,"");
$this=printWalkThroughArray($insertVals,"VALUES");
//the insert values
$insert_values = "INSERT INTO ".$dbtable." (".
                  $what.") VALUES (".$this.")";
$rslt=mysql_query($insert_values)
     or die ("Codedump: ".$insert_values.
             " Unknown Command: ".mysql_error());
return $rslt;
}

//==BASIC==UPDATE=OF==THE==DB==================
//this also creates a codedump if not working
function updateDB($dbtable,$colHeaders,$updateVals,$W_condition) {
if(is_array($colHeaders))
$what=printUpdateSet($colHeaders,$updateVals);
else
$what=$colHeaders." = \"".$updateVals."\"";
$update = "UPDATE ".$dbtable." SET ".$what." ".$W_condition;
$rslt=mysql_query($update)
     or die ("Codedump: ".$update." Unknown Command: ".mysql_error());
return $rslt;
}
```

```
//===MAKES==A==MENU==FOR==YOUR==TOP==SITE==PART=========
function menu($LINKS,$IDS) {
print("<div id=navbox><p class=separator>\n");
for($i=0;$i<count($LINKS);$i++) {
if($_GET["show"]==strtolower($LINKS[$i])) $classname="current";
else $classname="nav";
if($LINKS[$i] =="Forum Index")
$url="bb/index.php";
else
$url=$_SERVER["PHP_SELF"]."?show=".strtolower($LINKS[$i]);
if($IDS[$i] != 0) $url.="&submenu=".$IDS[$i];
//======print==out==array====
if($i==count($LINKS)-1)
print("<a href=\"".$url."\" class=".
       $classname.">".$LINKS[$i]."</a>\n");
else
print("<a href=\"".$url."\" class=".
       $classname.">".$LINKS[$i]."</a> ~ \n");
}
print("\n</p></div>");
}


//===THIS==CREATES==THE==SUBMENU==FROM==A==LINKS==ARRAY====
//link array comes from the database in this case
function submenu($LINKS) {
print("<p class=separator>\n");
for($i=0;$i<count($LINKS);$i++) {
$sIt=substr($LINKS[$i],0,3);
$sI=strtolower($sIt);
//check if your link is a part of the subitem
if($_GET["subItem"]==$sI) $classname="current";
elseif(!$_GET["subItem"] && $LINKS[$i] == $LINKS[0]) $classname="current";
//otherwise draw the links regular style
else $classname="nav";
//specify the URL you want to point to
$url=$_SERVER["PHP_SELF"]."?show=".
strtolower($_GET["show"])."&submenu=".$_GET["submenu"]."&subItem=".$sI;
//======print==out==array====
if($i==count($LINKS)-1)
print("<a href=\"".$url."\" class=".
        $classname.">".ucwords($LINKS[$i])."</a>\n");
else
```

```
print("<a href=\"".$url."\" class=".
        $classname.">".ucwords($LINKS[$i])."</a> ~ \n");
}
print("\n</p>");
}

//========THIS==CREATES==FORM==FIELDS==WITHIN==A==FORM==TAG==============
//currently doing radio, select, text, textarea
function drawInputField($formname,$classname,
                        $label,$input_type,$options,
                        $_VALUE) {
if($_VALUE == "NULL") $_VALUE=$_POST[$formname];
if($input_type!="radio")
   echo("<p>\n<label for=\"".
          $formname."\"><b>".
          $label."</b></label><br>\n");
//The dropdown list
if($input_type=="select") {
echo("<select name=\"".$formname."\" id=\"".$formname."\">\n");
for($i=0;$i<=count($options)-1;$i++) {
if($_VALUE == $options[$i])
echo("<option value=\"".$options[$i]."\" selected>".
       ucwords($options[$i])."\n");
else
echo("<option value=\"".$options[$i]."\">".
       ucwords($options[$i])."\n");
}
echo("</select>\n");
}
//a textarea (big textfield)
elseif($input_type=="textarea") {
echo("<textarea name=\"".
       $formname."\" rows=\"10\" cols=\"50\" class=\"".
       $classname."\">".$_VALUE."</textarea>\n");
}
//regular textfield or radiobutton
else {
echo("<input type=\"".$input_type."\" class=\"".
       $classname."\" name=\"".$formname."\" id=\"".$formname."\"");
//radiobutton
if($input_type=="radio") {
echo(" value=\"".$label."\"");
if($label == "Other")
```

```
echo(" checked=\"checked\" ");
echo("/>\n");
echo($label."<br/>\n");
}
//no radiobutton
else
echo("value=\"".$_VALUE."\"/>\n");
}
echo("</p>");
}
?>
```

3. **The Inform Assignment Framework**

```
!=============================================================
!Created on the basis of Roger Firth's Wilhelm Tell Tutorial game
Constant Story "The ambitious Computer Game Designer";
Constant Headline
      "^An Inform Tutorial for the Otago University's
        Computer Game Design Paper 2004
       ^by Lennart Nacke
       ^-----------------------------------------------------
       ^Please also read ~The Inform Beginner's Guide~
        by Roger Firth and Sonja Kesserich
       ^and ~The Inform Designer's Manual~
        by Graham Nelson to get you started.^
       ^Type in HELP if you need it (disabled)!
        Enjoy this Tutorial Adventure!^";

Constant MAX_CARRIED 1;     !Maximum items a character can carry
Release 1; Serial "031218"; !Serial = date of the programm: YYMMDD

Include "Parser";     !need it, header file
Include "VerbLib";    !need it, header file
!=================================================
! These files were added additionally but are left
! out for this assignment
!=================================================
!Include "Menus";     !optional
!Include "Howtoplay"; !help function, if you use "sos" in the game.

!=============================================================
! Object classes can be used as Game Objects later
! Set up a suitable environment.

Class   Room         !Setting up a Regular Cell in our Adventure Pattern
  has   light;       !Set attribute, no properties defined yet.

Class   Prop         !Properties of Cells, Objects in Cells
  with  before [;
            Examine: return false;
            default:
                print_ret "You don't need to worry about ", (the) self,
                          ".";
        ],
```

```
  has   scenery;     !Cannot move it

Class   NPC          !implementing behaviour for a non-player character
  with  life [;
            Answer,Ask,Order,Tell:
                print_ret "Just use T[ALK] [TO ", (the) self, "].";
        ],
  has   animate;

!=============================================================
! The Rooms/Landscapes/Sights that you enter during the game:

  !This is where you start:
  Room  before_computerlab "In front of the Computer Lab in
   Castle Street"
  with  description [;
          print "You stand outside the Computer Lab in Castle Street.
          Some Students are passing by.^";
          if (computer_guy in self) print "There is a guy, that
          really looks like a computer science student outside
          the lab.";
          new_line;
          ],
          n_to "The northern part of the street is the heart
          of ~Studentville~ and you do not feel it would be
          a good idea to go there now. You have got work to do.",
          w_to "You can see a wall and not much space to go through.
          You better choose a different direction!",
          e_to computerlab,
          s_to crossing_castle;

  Room  computerlab "In the Computer Lab on Castle Street"
  with  description [;
          print "You are in the computer lab where you
          have just started your course in game design.^";
          if (simon in self) print "You can see Simon, your
          computer games design lecturer
          standing there.";
          new_line;
          ],
        w_to before_computerlab;

  Room  crossing_castle "On Campus: Crossing Castle St. with St. David St."
```

```
with  description
      "You are north of Campus, east from you flows the
       ~Water of Leith~, across which
       you can see a very nice clocktower building.
       West of you are university buildings.
       Glimpsing through the walls you see a sign but
       cannot read it from here.",
      n_to before_computerlab,
      w_to science_lib,
      e_to clocktower,
      s_to crossing_union_bridge;

Room  clocktower "In front of the Clocktower building"
with  description
      "You hear the river flowing behind you as a this beatiful
       building ascends before your eyes. There is pathway on
       the side that leads south. A lot of students are
       walking around, while a japanese couple has their
       wedding pictures taken under a
       tree by the riverside.",
      s_to cs_dept,
      w_to crossing_castle,
      n_to "No, that is probably leading off campus,
            you do not want to go there. You have got work to do.",
      e_to "No, there are more buildings, but you do not want to go there.
           You have got work to do.";

Room  cs_dept "East of the ~Water of Leith~
  in front of the computer science building"
with  description
      "Yes, you know this door, it leads right up
      to the computer science department where you
      enrolled for this course earlier. The door seems locked.
      To the west you see a brigde.",
      n_to clocktower,
      w_to crossing_union_bridge;

Room  crossing_union_bridge "West of Union Bridge"
with  description
      "Union bridge leads east from here.
      There is a pathway down to south across the lawn from
      here.",
      e_to cs_dept,
```

```
        n_to crossing_castle,
        s_to central_lib,
        w_to "If you go west from here, you probably
              lose focus on your task, so you decide to go another
              direction.";

  Room  central_lib "In front of the Central Library"
  with  description
        "Right in front of you rises the huge modern building,
        that has a sign ~Central Library~ above
        the entrance. You notice some students travelling around.",
        n_to crossing_union_bridge,
        w_to "If you go west from here, you probably
             lose focus on your task, so you decide to go another
             direction.";

  Room  science_lib "In front of the Science Library"
  with  description [;
        print "There is a sign above the entrance reading
          ~Science Library~. It makes you curious,
          if you could possibly find your book in here.";
        if (lenny in self) print "Great there is Lenny standing
                             out there. He might just be the right person
                             to talk to and find out about that book that
                             you are searching for.";
        ],
        e_to crossing_castle;


!=============================================================
! Non-Player Characters in the Game:

  !Simon can be found in the Computer Lab
  NPC   simon "Simon McCallum" computerlab
  with  name 'simon' 'lecturer' 'games' 'game' 'professor' 'sim'
             'design' 'computer' 'guy' 's',
        description
            "Simon has long hair and a nice short goatee. He wears a
             T-Shirt with a Otago Computer Science Penguin Logo
             on it and a blue-yellow jacket.
             He has a visionary look on his face.",
        !Now we use this because we do not want Simon
        !to be listed separately.
```

```
        describe [; if (self in computerlab) rtrue;
                "^Surprisingly you see Simon standing there.";
                !This works for any other room
                 ],
        times_spoken_to 0, ! for counting conversation topics
        times_touched 0,
life [;    !here we override given methods
                !for verbs during the life cycle
            Mild:   print_ret "~What is your problem?~";
            Kiss:   print_ret "~Urg! Really, what is your problem?
              I am quite happy with my wife!~";
            Attack: score = score - 1;
                    print_ret "You would not have a tiny
                      chance against him!
                      Besides why would you do that crap?!";
            Give:   move noun to self;
                    print "You give Simon ", (the) noun;
                    if (noun == inform_book) {
                      deadflag = 2;
                      print_ret "^^Finally I have got it back!
                      Thank you very much!^
                      You can start the inform assignment
                      now and rewrite this adventure a little.";
                     }
            WakeOther:
                self.times_touched = self.times_touched + 1;
                if(self.times_touched > 1) {
                   score = score - 1;
                   print_ret "~You really don't seem to get it?
                     Do you now?~";
                }
                else print_ret "~I am not asleep,
                  what is the problem?~";
            Talk:
              if (location == computerlab) {
                  if (inform_book in simon) {
                      print_ret "Yeah you won!";
                  }
                  else if (inform_book in player) {
                      print_ret "Hey cool, you want to
                      give me that book now?!";
                  }
                  else{
```

```
                    self.times_spoken_to = self.times_spoken_to + 1;
                    switch (self.times_spoken_to) {
                        1:  print_ret "~Hi, suppose you are on your way
                          to find me that book?~";
                        2:  print_ret "~Better hurry up and get it now!~";
                        default: print_ret "~Did you already find
                          that book, which I wanted to have?~";
                    }
                  }
              } else print_ret "~I don't belong here!~";
        ],
        before [;     !here we override given methods for verbs general
              Strong: deadflag = 3;
                  print_ret "You die instantly! Never mess
                    with your lecturer!";
        ],
has     male proper;

!The computer guy stands outside the lab waiting for you:
NPC     computer_guy "Computer Guy" before_computerlab
with    name 'guy' 'computerfreak' 'person' 'male' 'boy'
             'student' 'computer' 'science' 'cg',
        description
            "He really looks like these guys who are totally
             into computers, actually if you are
             male, he looks quite like you.",
        !Now we use this because we do not want the computer
        !guy to be listed separately.
        describe [; if (self in before_computerlab) rtrue;
         !This causes the computerguy to be in the clab desc
                  "^There is computer guy standing there.";
                 ],
        times_spoken_to 0, ! for counting the conversation topics
        life [; !here we override given methods for verbs.
            Talk:
                self.times_spoken_to = self.times_spoken_to + 1;
                if(self.times_spoken_to > 5 &&
                   self.times_spoken_to <= 15) {
                  score = score - 1;
                  print_ret "~HEY MAN! Quit Buggin me!!~";
                }
                if(self.times_spoken_to > 15) {
                  deadflag = 3;
```

```
                print_ret "~I told you stop annoying me, MAN!
                If you can't listen, I might as well teach you
                some bad guy tricks!~ Although he seemed quite
                friendly at first, the computer guy comes up to
                you and starts hitting you.
                You quit talking instantly, but he doesn't seem
                to understand! You feel horrible pain, while
                you are beaten into a big puddle of blood.";
        }
                switch (self.times_spoken_to) {
                    1:  score = score + 1;
                        print_ret "~Howdy, so you want to know
                        about a Book? I can only recommend the
                        science library, that's where I get all
                        my books from.~";
                    2:  print_ret "~Like I said, you should
                        really take a look at the Science Library.~";
                    default: print_ret "~Yeah, it really
                    might be in that library...~";
                }
        ],
has     male;

!Lenny stands in front of the science library:
NPC     lenny "Lenny" science_lib
with    name 'guy' 'len' 'l' 'lenny' 'mr' 'nacke' 'person'
    'man' 'boy' 'german' 'student' 'intern'
    'computer' 'science',
        description
              "This is the german intern, that brought
              you this weird little tutorial, he wears
              a shirt reading ~WILL CODE FOR FOOD~ and
              wearing a summer hat. He looks a little stressed.
              Maybe he can help you out.",
        describe [; if (self in science_lib) rtrue;
                    "^You can see Lenny standing there.";
                    !This works for any other room
                  ],
        times_spoken_to 0, ! for counting conversation topics
        life [; !here we override given methods for verbs.
             Talk:
                self.times_spoken_to = self.times_spoken_to + 1;
                switch (self.times_spoken_to) {
```

```
                    1:  print_ret "~Hi, what are you doing here?
                        Oh, I think I know you from the
                        Computer Game Design Class? How can I help
                        you? What are you looking for?~";
                    2:  move inform_book to player;
                     score = score + 1;
                     print_ret "~Oh yes, the inform book,
                     I have heard about it. Actually I have
                     just borrowed this one from the library
                     right here. You can sure have it!~
                    ^YOU RECEIVED THE INFORM BOOK!";
                    default: print_ret "~What a nice day.
                    I will go surfing this afternoon...~";
                }
        ],
        before [;  !here we override given methods for verbs general
               Kiss:
               print_ret "If you are a girl: Lenny
               is very enchanted and gives you his phone number.
               If you are a guy: He laughs and turns away";
        ],
  has   male proper;
!=====================================================================
! Game Objects and Properties in Rooms, also know as 'the other stuff'


  Object  inform_book "Book: ~The InforMer - A simple approach to IF~"
    with  name 'inform' 'book' 'writing' 'manual' 'informer' 'if',
          times_read 0,
          description [; !!Unnecessary
              if (location == science_lib)
                  print_ret "Great, this is the book you have been
                  searching for.
                  Good, that Lenny gave it to you. You will now
                  have to take the book up
                  to Simon in the computer lab";
              else
                  print_ret "It is the inform mania";
          ],
          !if(self.times_read > 0)
          before [;
            Consult,Examine:
            self.times_read = self.times_read + 1;
```

```
            switch (self.times_read) {
                1:  print_ret "This is the Inform Manual,
                 it will tell you everything you will need
                 to start programming inform.
                 You should turn the page to find useful
                 internet links!";
                2:  print_ret "
                    http://www.inform-fiction.org/links/tutorials.html";
                 default: print_ret "If you want to know more about
                 inform and the secrets behind
                 it you should look for ~The Inform Beginner's
                 Guide~ by Roger Firth and Sonja Kesserich
                 and ~The Inform Designer's Manual~ by Graham
                 Nelson. They are really great resources
                 of information.";
            }
            print_ret "This is the right thing to do!.";
          ],
  has ;

Prop  "Students"
with  name 'students' 'scarfies' 'people' 'crowd',
      description "The cold has made them though,
      you are one them usually and you all study in
      this beatiful place. Still they might not be able to
      help you with your task!",
      before [;
          Talk:  print_ret "~Aye, man. We can't help
          you with that. Sorry!~";
          Attack: score = score - 1;
          print_ret "~Don't you dare going there!~ You would
          have a tiny chance against them!";
          Consult: print_ret "~Hmm, that kind of book would
          probably be in the science library!~";
      ],
      found_in before_computerlab clocktower central_lib science_lib,
has   animate pluralname proper;

Prop  "Couple"
with  name 'japanese' 'couple' 'jap' 'lovebirds' 'wedding',
      description "A beatiful wedding set! The poeple seem
      to be busy though.",
      before [;
```

```
            Talk:  print_ret "They are minding their own business.";
            Attack: score = score - 1;
            print_ret "You would have a tiny chance against them!";
        ],
        found_in clocktower,
  has   animate pluralname proper;

!=============================================================
! Entry point routines

[ Initialise;
    location = before_computerlab;  !Set the starting location of the game
    lookmode = 2;                   !like the VERBOSE command
    !move quiver to player; give quiver worn;
    Amusing("CHEERS! You have just successfully
        won your first computer gameassignment!");
    player.description =            !given after examining yourself
        "You are wearing a sweater with a hood, washed-out and patched jeans.
         Your hair is fuzzy and you are glad not be wearing those shell-rimmed
         glasses anymore. They gave you the creeps. Now you are on the edge of
         becoming a computer game programmer and the nerd-times are over.";
    print_ret "
        Our Setting: Dunedin, a small university town set in the beautiful
        landscape of New Zealand's southern island.
        The year is 2004, at which time
        Otago University has just established a course in Computer Game Design
        supervised by your favorite CS lecturer Simon McCallum.
        You are one of the students doing this paper during summerschool.^^
        One of your tasks is to find out more about the ~Inform~-language
        with that most of the text-based interactive fiction
        adventures you have heard about are written. You had some
        information that did get you started but as a
        fellow student you are yearning to retrieve
        more information about this subject.^^
        But right now, you have been sent away to fetch a book called
        ~The InforMer - A simple approach to IF~ for your lecturer
        which is said to be located somewhere in the science
        library.^";
        !the first introductory lines printed at program launch
];

[ DeathMessage; print "You have screwed up your first computer
games design tutorial you must have done something really stupid!"; ];
```

```
!=============================================================================
! Standard and extended grammar

Include "Grammar";
!=============================================================================
! Here you have the chance to include special
! not pre-defined verbs into this game:
!-----------------------------------------------------------------------------

[ TalkSub;
    if (noun == player) print_ret "Nothing you hear surprises you.";
    if (RunLife(noun,##Talk) ~= false) return;  !consult life[;Talk:]
    print_ret "At the moment, you can't think of anything to say.";
];

Verb 'talk' 't//' 'converse' 'chat' 'gossip'
    * 'to'/'with' creature          -> Talk
    * creature                      -> Talk;
!-----------------------------------------------------------------------------
```