GAMASUTRA
The Art & Business of Making Games

PRINT

# Faculty Postmortem: Cal Poly Pomona's Game Development Course

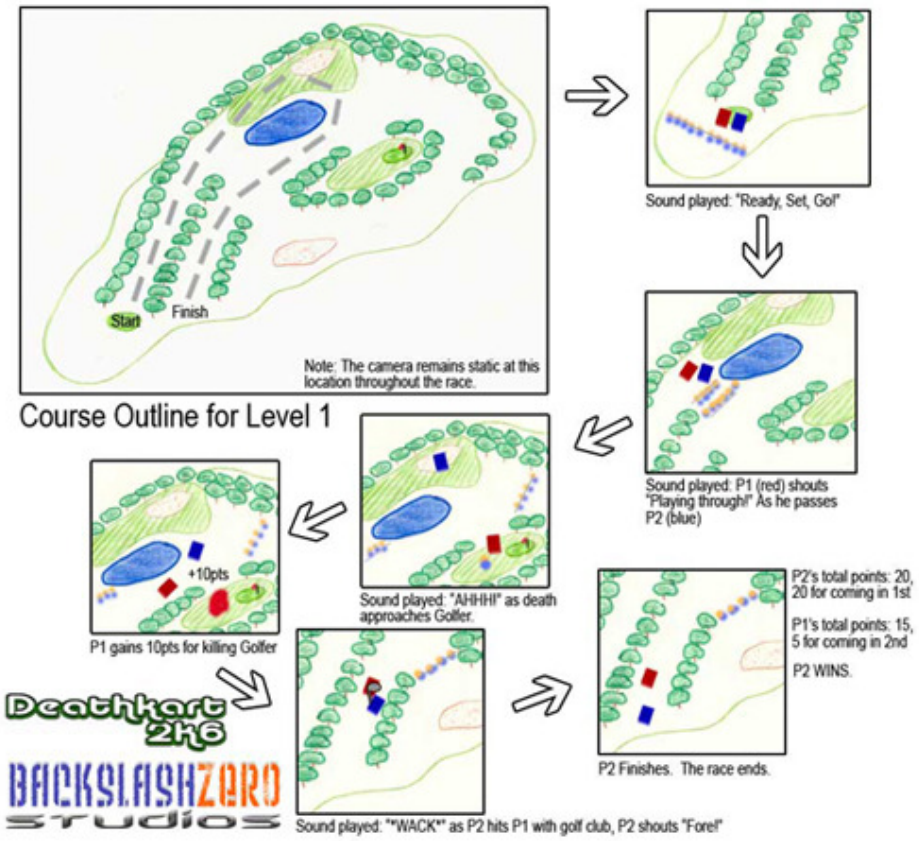
By Robert W. Kerbs

**Introduction**

Over the last five years, I have worked as a faculty member in Cal Poly Pomona's Computer Science (CS) department. Having formerly worked as a game programmer for Kronos Digital Entertainment, I frequently assign programming projects with a slant towards solving game development problems. Our students especially appreciate the opportunity to program applications that interest them. About a year ago, students finally persuaded me to offer a one-term game development course.

I created a 10-week course that would allow senior-level CS students the opportunity to conceive, design, and implement a 3D PC game. This course, Intro to Game Development, was offered for the first time in Spring 2006. Each game was required to have three different levels, each with increasing difficulty. I selected OpenGL for the graphics pipeline, GLUT for event handling and models, OpenAL for sound effects, Microsoft's MCI API to play back CD audio tracks, and Lua to script game-state information. C and C++ were used to implement the game engine and to put it all together.

Although offered as a senior-level course, students had differing strengths. Some had taken the 2D/3D computer graphics course (where we teach OpenGL), some had taken the AI course, some had taken the operating systems course, and so on. Consequently, students selected their teammates not simply based upon who they knew, but also based on their perception of student skill bases. The prerequisite I required for all students was completion of the C++ programming course we offer.




**Student Storyboard for Level 1**

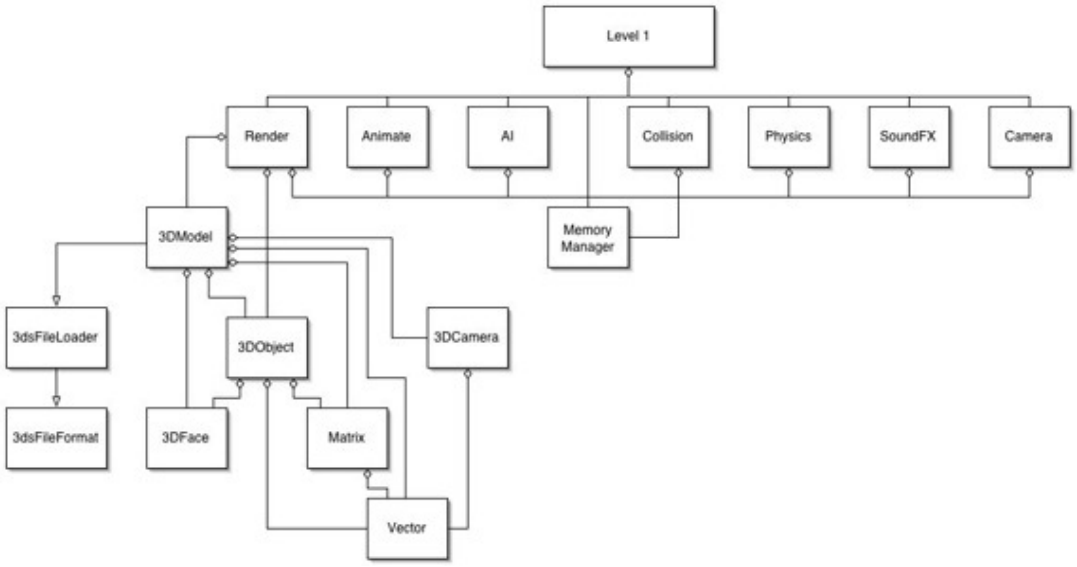
**Milestones**

To help ensure that students would be successful in the ten-week period of time, I created a milestone chart that included deliverables due at the end of each of the ten weeks. I typically lectured once per week presenting theory and examples while students utilized the rest of their time to work within their groups. For the *final* (week 11), each team presented their game and performed a critical stage analysis.

**Development**

Students assembled themselves into 14 four-person teams (two teams had five students) and assigned themselves a studio name. Each team developed a story for their game and three storyboards, one for each level of their game. Students were given the latitude to implement their storyboards in any format they wanted—Flash, a paint program, by hand—some did a mix of all of these.

The software architecture of each game was accomplished via a software modeling language frequently used in software engineering called Unified Modeling Language (UML). There are 13 different types of UML diagrams that can be constructed. Due to time constraints, I had each team construct one UML class diagram. This diagram served as the foundation for each team's software architecture.




**Student UML Class Diagram**

At this point, each team began work assembling the software modules that would eventually become their game. The milestone chart helped direct the order of construction of the modules as deliverables were due each week. For example, after the UML class diagram was due, the next week the memory manager needed to be completed, and so on. It took about four weeks before most teams could begin to see what their own unique virtual worlds looked like.

Simultaneously, students worked on the visual and aural elements of their games. Although I encouraged students to construct very simple models (and to not try to duplicate their favorite game characters), some took the opportunity to learn about Maya, Milkshape 3D, 3D Studio Max, Blender, and so on, and tried to build and import models from these software packages into their games. Many groups recorded their own sound effects, while a couple of groups recorded their own sound tracks.

The game state was set via Lua scripts. Collision detection and handling was eventually implemented as was Non-Player Character (NPC) AI and physics. I required that all parameters for these items be set via Lua scripts.

The final deliverable was an Auto-Run CD that students placed in a computer in our Software Engineering Laboratory. This CD started up in kiosk-mode demonstrating all screens, levels, models, sound effects, and songs for the game. Once interrupted, the game exited kiosk-mode and the main game screen came up for the player to make a selection.

While each team's game was demonstrated in kiosk-mode, each team member described their experience and contribution. The last task was for each team to present a critical stage analysis (something that would normally occur at each stage of the game's development). In this case, I had each team present five items that went right, five things that went wrong, and five things they would change if they were to continue their development. Students' comments serve as a basis for the remainder of this postmortem.

---

**What Went Wrong**

**Time management on the overly-ambitious ambitious project.** While a couple of teams were able to fully implement their story ideas and meet all milestones, most team's games were overly-ambitious for a ten-week course. A few started off with the intention of building and importing models from Maya, 3D Studio Max, and other modeling software packages. When they realized that a great deal of work would be needed for actual implementation of these models in their own games, most teams dropped the idea, and went ahead with simple models of their own. From the beginning, I encouraged

the teams to use very simple geometric shapes at first, and if time-permitted, they could then consider utilizing the more advanced models. Only one team (out of 14) was able to nail down every milestone so well that they had extra time before the next milestone was assigned to explore modeling packages.

**Technical problems.** Each game had three modes: player-against-machine, player-against-player, and kiosk. By far, the most difficult mode for the teams was with kiosk mode. In this mode, all screens, models, animations, AI, collisions, physics, sound effects, and songs for the game would automatically be demonstrated for each level of the game. Once the final level was traversed, then the game would start over at level 1. This challenge tested all aspects of their games and pointed out the difficulty of handling AI, collisions, and physics correctly.


**Student-Created Main Selection Screen**

A major problem with most groups was the use of a source code control system whereby the team's code could easily be merged together. Magnifying the problem was the fact that many students had different software packages and versions than the computers in the Software Engineering Laboratory. Frequently, students worked on something at home and found that it did not work at school on a laboratory computer. At least 30% of the student in-lab time was ill-spent with both of these problems.

Another technical problem involved the systems teams demonstrated their games on. While all of the computers were 2Ghz P4s with 1GB of RAM, the graphics cards were not up-to-date. Consequently, some teams experienced slow-downs when using the school's computers.

**Design.** Some teams had a difficult time progressing from their original UML class diagram to actual game implementation. As problems became identified in their original designs, some modified their UML diagrams to better represent what they actually ended up with. While this iterative approach is a good software engineering principle, some teams did not close the design loop by updating their UML diagram.

---

**What Went Right**

**Teamwork.** This was clearly the number one cited area students talked about when describing the success of their games. They realized that they could not have completed their projects on their own in the time given without splitting up tasks and meeting frequently. An important point that our industry partners would like to see in new CS graduates is in fact the ability to work with and communicate clearly with others. CS students frequently fall in a more stereotypical model of the programmer who works alone and only communicates when absolutely necessary. Socially, students seemed to enjoy the support structure of the team environment.

**Deadlines.** Overall, students were satisfied (some very surprised) at how much they were able to accomplish in such a short period of time. Many students were taking three or four other classes at the same time. Consequently, a majority of the teams frequently pulled all-nighters to get the milestones fulfilled.

**Technologies.** CS students typically do not have experience utilizing different programming language APIs in a single program. CS-type programming projects usually fall along the lines of: "using Java, implement a program that reads a file of integers, stores them in a binary tree, and then prints out the number of steps to reach each integer in the tree." Consequently, this project proved to be an opportunity to combine OpenGL, OpenAL, Lua, and other interfaces into a program that they invented themselves. As a result, students felt very good about successfully designing, implementing, debugging, and finally enjoying their games as they saw the various elements come together.

**Student Gameplay Screen**

**Game architecture.** Since the UML class diagram (described above) had to be created before any programming started, the teams were able to conceptually understand what was expected of them before partitioning specific programming tasks. This resulted in relatively simple and straight-forward designs—and made implementation and debugging easier. This modular approach also enabled students to understand that if given more time, they would have the opportunity to replace these modules with more sophisticated versions (i.e. physics, AI, collisions, memory manager, etc).

---

**Next Development**

**Architecture.** Some teams said they would stick to more of a C++ object-oriented design approach. It turned out that C-style coding fit in more directly with OpenGL and that the object-orientation approach took more contemplation during the design-stage. It was pointed out that a careful C++ design would be beneficial due to its inheritance and composition features and speed up later development stages. A few groups did indeed take this approach for this reason.

**Technical Aspects.** There were a lot of technical aspects that teams would address—the gamut was large. One team, for example, discovered they were drawing the entire world instead of just the viewable world, resulting in rendering slow-downs. Another team had some ideas on how to improve the game's AI. Other teams wanted to redesign their game engines to be constant-rate driven rather than message-driven so that they would run at similar rates regardless of processor speed.

**Requirements and planning.** All the teams agreed that Cal Poly Pomona's 2D/3D computer graphics course should be made a prerequisite for this course. Also, a source code control system should be set up so that students can more easily manage their development. Another improvement might be to allow students to present their games on their own laptops and not be required to run on the Software Engineering Laboratory's computers.

**Time.** The biggest time-related issue is that students underestimate the amount of time and effort to complete a task. Like many elements comprising a 3D game, one change can affect other things in the design—even if it was designed in a textbook object-oriented C++ fashion from the onset.

**The Team Celebrates!**

**Conclusion**

Students had a very challenging yet rewarding time in this class. Most had a lot of experience playing games but did not have a solid foundation of game complexity or construction. Many students informed me that they spent more than twice the amount of time in this class than they would have imagined. The excitement surrounding the design and construction of a team's game of choice continually motivated students.

Our industry partners (i.e., developers in the greater Los Angeles area) have been advising us that students need to be gaining experience working with others and on larger-scale projects. In response, we have introduced 56 students to this environment and approach. Before the end of the term, a couple students successfully landed jobs in game development companies.

Due to the positive outcome of this course, I am currently developing a game engine design course that will serve as a prerequisite for the course described here. The game engine course will be offered Winter 2007. A revised version of the course described here will be offered in Spring 2007.

---