Scott Bilas (scott@gaspowered.com)
Senior Mechanic
Gas Powered Games
25 Central Way Suite 210
Kirkland, WA 98033 USA

GDC Game Objects Talk
Created 6.27.2001
Due 7.3.2001
Tel: 425.576.8746x123
Fax: 425.576.8746

A Data-Driven Game Object System

Speaker Biography (100 words)

Please include a 100-word biography that describes your professional experience and background, with emphasis on the areas that qualify you to speak on the subject(s) you have chosen. Include a credit list and indicate your role or function on the titles you've developed. (Please write in third person, present tense.)

Scott works at Gas Powered Games where he builds back-end gaming systems, such as scripting engines, parsers, databases, application foundations, and type systems. Shipped titles include Dungeon Siege (engineer/handyman), Gabriel Knight III (engineering lead), iCat Commerce Online (project lead/system architect), and Mighty Math Cosmic Geometry (engineer). Scott has been published in *Game Developer* magazine and Game Programming Gems, and edited a section of Game Programming Gems II. He also thinks lectures are fun, and has given several GDC talks in past years.

Level (select one)

Beginner	The beginner level is intended for audiences who have cursory or fleeting experience with the subject matter.
Intermediate	The intermediate level is intended for audiences who are well versed in the subject matter.
Advanced	The advanced level is targeted to audiences who are accomplished, experienced and expert in the subject matter.

Intended Audience & Prerequisites

Please describe what section of our audience would best benefit from this session? Is prerequisite knowledge necessary for understanding the content of the session? (Please write in third person, present tense.)

All engineers can certainly benefit – it's nearly impossible to work on a game without coming into contact with a game object system. However, the engineers who focus on system architecture and game databases are the intended audience, and will get the most out of the lecture. Attendees should be experienced in building systems using traditional C++ object-oriented methods.

What is the idea takeaway from this presentation? (5 sentences)

Please describe, in five sentences, what tangible ideas will be taken away be the attendee? Or, in other words, what will they learn? (Please write in third person, present tense.)

An attendee will have learned why the traditional approach to game object management is inadequate, and how to design and build a better system: component-based, hierarchical, data-driven, and tuned to their game's specific needs.

Abstract (150 words)

Please attach a description of your presentation, as you would have it appear in the conference program, in 150 words or less. (Please write in third person, present tense.)

In a typical game, nearly all dynamic, interactive content is managed by “game objects”, which must perform complex tasks like animating, speaking, pathfinding, persisting, triggering, and networking. The textbook method to designing this system is to construct a hierarchy using the C++ type system, yet designers want their objects to perform as many interesting and varied tasks as possible. As engineers we must then struggle to make their desires fit into the constraints of our strict type system, recompiling over and over again to keep up with even slight changes in the design (not to mention keeping the level editor up to date).

In this talk we will cover a better way of building and managing game objects. Throw away the hierarchy, break the tasks down into C++ and script components, and assemble the components into objects. Data-drive the assembly process and now you have a powerful and quick system for building new types that requires no engineering time to manage! The design given in this talk is the same as that used in Dungeon Siege, which manages hundreds of unique object types.

Syllabus or Expanded Abstract (500 words)

Your chance to describe your class in greater detail to the Advisory Board. Please limit your expanded description to one page or 500 words (we have to read hundreds of these). ***Absolutely no proposal will be considered without a syllabus.***

Introduction

Introduce talk with brief background on the speaker. Define a game object system, as the term will be used in the talk, and what elements it includes. Also note the application of this design (content-heavy role playing game, 2800 total object types, 50,000 objects in the game world, continuous streaming engine, etc.).

Quickly cover the traditional solution. Mention the classic textbook "employer/employee" example, and why it falls over with more than a handful of types (reference the Age of Empires postmortem as an example).

Implementation

Next cover the system used in Dungeon Siege:

- Explain the need for a general-purpose hierarchical data language such as XML (give examples from Dungeon Siege's GAS language). Also need a simple query system for this database.
- Detail the organization and classes used in the system – hierarchical data configuration with instancing and override support, the component-based game object system that relies on it, and the object databases that manage all of this. Answer the question, "what is a component?"
- Show how to build components out of scripts and seamlessly integrate them into the system so that there is no apparent difference between C++ and script components to the rest of the system.
- Finally, detail how the level editor fits into all of this. The editor must save/load object instances for a level and be able to override properties of components, keeping up to date with changes in those components. Show how to gain this functionality nearly for free.

Advantages

- Easy to use with a fast turnaround. Designers can very quickly have their ideas prototyped, usually without engineering intervention. They can create new types and easily insert them into the game. New functionality can be added by a scripter, with engineers only providing occasional new function support.
- Memory efficient. As components are groups of related functionality, it is easy to remove components from types that do not need them. Better yet, in a multiplayer, server-only components can simply not be loaded on clients.
- Self-organizing. The types are organized based on required functionality, not programmer convenience, and so naturally organizes a game's object types into an easy to understand specialization tree.
- Self-documenting. One of the problems of engineering is documenting and maintaining the schema and rules of the engine – this system makes it easy to see how objects may be constructed at a glance by encoding the rules and constraints into data that the designers can see and even tune.

Disadvantages

Any complex system comes with disadvantages. These may be avoided by actively managing the design and implementation of the components.

- More prone to spaghetti because components very often need to talk to each other, and so become interdependent. Various operations may become order-dependent on the components. Assigning each component a priority can help.

- Must be careful about how the data templates are structured to make sure that components aren't added arbitrarily (increasing memory usage and CPU load). Should regularly evaluate the components to make sure they are slim, though the necessity of this completely depends on the game being built.