Scripting

Christine Alspach
Paul Betts
Boonth Nouanesengsy
Eric Stegemoller
Overview

- Introduction & Motivations
  - Christy
- Issues and problems
  - Paul
- Language Options
  - Boonth
- Wrappers
  - Eric
What is scripting

- By definition, a language which doesn’t need to be compiled
- Game world has other interpretations
  - Use the same language as the main game. Compile to .dlls and link at run time (Quake II)
  - Have a compiler & virtual machine to read bytecode scripts (Unreal)
- Main code calls external functions of some sort
Why use scripting

- Way to change game without recompiling
  - Gameplay content
    - Change prices, weapon specifications, NPC behavior
  - Testing
    - load mid-game scenarios
  - User content
    - Changes can be made without releasing source
- Some languages allow dynamic linking
Factors to consider

- How much do you need
  - Simple content may best be served by XML
  - If you script the entire game you may be missing the point
  - Many languages have little error-checking
  - In general, script the things which you think will change the most

- How fast does it need to go?
  - Tradeoff between run speed and edit speed
  - Testing setups don’t need to run quickly
  - Most scripting doesn’t occur in time-sensitive areas
  - Languages with garbage collection may run at inopportune times

- Is it ok if the users see it?
  - Many languages can be compiled or encrypted for shipment
Language choices

• Roll your own
  – Time consuming

• Scripting languages you may already know
  – Python (*EveOnline, Backyard Sports*), Perl, JavaScript…

• Some common in the game industry
  – Lua (*WOW, Escape from Monkey Island, Homeworld 2 …*)
  – UnrealScript -$$- (*Deus Ex, America’s Army …*)

• Other Open-Source Options
  – Squirrel, Ruby, ICI & many many more
Issues with Embedded Scripting

- Type Marshalling
  - Converting between C++ types and script types
- Multi-threading
  - Turns out you can't have 3 threads calling the same interpreter
- References / Garbage Collection
  - How the interpreter keeps track of memory allocation
Type Marshalling

- Data types between your application and the interpreter probably aren't the same
  - Integral types may be of different formats (UTF-8 vs. ASCII, etc..)
  - Types may exist on one that don't exist on another (Python and Ruby both have a “hash table” integral type)
  - Structures and Classes aren't the same either; neither are function calls
The solution?

- Three approaches:
  - Manually convert data types as needed between C++ and the interpreter
    - Quickest for small projects, but very error-prone
  - Create a parser to read C++ files and auto-generate the bindings
    - Approach used by Swig and Boost.Python
    - Sometimes hard to set up
  - Create a scheme to add metadata to types, then generic type marshaling code can be written
    - Approach used by several UI frameworks (GTK+, Qt)
    - Makes C++ code difficult to read
Threading issues

- Just like in native code, the interpreter must be treated in a thread-safe manner
- Interpreter will usually handle threads that it creates correctly
- Python includes a mechanism to serialize access to the interpreter via a global interpreter lock
  - Not as scary as it seems, it yields periodically to avoid deadlock
References / Garbage Collection

• Scripts often keep track of memory allocation implicitly
  – Python does this by reference counting
  – Ruby uses a mark-and-sweep garbage collection algorithm
• All bets are off when writing bindings though!
  – References must be explicitly kept track of; difficult to track down bugs (Decrement a reference one too many times and your program segfaults somewhere)
Lua

- Multi-paradigm language
  - Small set of features that can be extended
- Used extensively in games
  - World of Warcraft
  - Neverwinter Nights
  - Baldur’s Gate
Lua

- Comes in many flavors
  - LuaPlus
    - Simplifies embedding process
    - Has debugger
  - LuaBind
    - Support for inheritance
    - More powerful, more complicated
  - toLua
    - Automatically generates the binding code to access C/C++
Lua

• **Advantages**
  - Popular
  - Well-documented
  - Light
  - Fast

• **Disadvantages**
  - Not as powerful
Squirrel

- High level imperative/OO programming language
- Lightweight, designed for video games
- API similar to Lua
Squirrel

- SqPlus
  - Easy to bind
- Squadd (similar to LuaBind)
  - Boost-based
- SQDev (Eclipse)
  - Debugger
  - Syntax highlighcher
Squirrel

- Mainly differences over Lua
  - Syntax closer to C++ then Lua
  - Classes\Inheritance built in
  - Has fast arrays
Python

- Multi-paradigm language
- Popular for scripting
- Used in some games
  - EveOnline
  - Freedom Force
- PyOgre
  - Python wrapper for Ogre
  - Currently being rewritten with SWIG
Python

- **Advantages**
  - Powerful
  - Robust
  - Libraries to do just about anything (regex's, Xml parsing, network I/O...)

- **Disadvantages**
  - Slow
  - Larger memory footprint
Generating Wrappers

- The most popular tool to generate wrappers is Swig (http://swig.sourceforge.net)
  - Generates bindings for: Allegro-CL, C#, Chicken, Guile (Scheme), Java, Modulo-3, MzScheme, OCAML, Perl, PHP, Python, Ruby, TCL
  - Parses all of ANSI C and almost all of ANSI C++ (except for nested classes)
- Other wrapper generators include Boost.Python/Pyste and Gobject.
  - The people who write Boost are way too good at C++
How does Swig work?

- It reads an interface file
- This file lets you select which classes/methods you want to create bindings for
  - %module example
    %{
      #include "example_header.h"
    %}
    %include "example_header.h"
- Swig reads this file and generates the appropriate code
Baked Goods!

- Eric's bringing some baked goods for you, like cookies and cupcakes!