

SM – A scripting language for game engines

Ola Frid

19 February 2008

Short summary

- What has been done:
 - A scripting language for game programming
 - Designed
 - Implemented
 - Evaluated

Background

- Why use a scripting language for games?
 - Separation of game engine and data
 - Different requirements of engine code and game code
 - Easy reuse of engine
 - Safe user modifications
 - Less need for frequent compiles
 - Makes some things easier
 - Saving game state
 - Configurations

Background

- Examples
 - Lua, an extensible extension language
 - For configurations and user macros
 - Used in many games
 - Grim Fandango, Escape from Monkey Island, Far Cry, World of Warcraft UI, and many more
 - Dynamic typing
 - Global state
 - Nothing exactly specific for games

Background

- Examples
 - UnrealScript
 - Basically a language and VM for games
 - Used in many games
 - Unreal series, Deus Ex (1&2), Harry Potter games, America's Army, Gears of War, and many more
 - Similar to Java
 - Curly braces, object oriented, static typing, no pointers etc
 - Provides many features helpful for games
 - Networking, states, timing
 - Tied to the Unreal Engine

Background

- Problem with existing, roughly
 - Either
 - The language is general; for example, has no helpful ways of doing game centric networking
 - or
 - The language is tied to a certain game engine, or even a certain type of games

Task

- What do we want?
 - A scripting language for game programming
 - Solves common problems for game programming
 - Not tied to any specific game engine
 - Instead, easily bound to any engine
 - Familiar to game programmers
 - Preferably also provides good organization and bug prevention

Task

- Primary points
 - Networking
 - Organization
 - Couplings
 - Security
- Secondary points
 - States
 - Concurrency
 - Save games
 - Configurations

Language

- Overview
 - Curly braces style (like C/C++/Java/UnrealScript etc)
 - Object oriented (in C++/Java sense)
 - Multiple inheritance (like C++ virtual inheritance)
 - Uses modules, has free functions
 - Strongly and statically typed

Language

- Why statically typed?
 - Helps with networking
 - Catches some bugs at compile/load time
 - Forces organization
 - Might by some be considered bad, limiting
 - Can be helpful on large projects with many programmers
 - Alternatives?
 - Strong unit tests
 - Very accurate programmers

Language

- Types
 - Basic types
 - bool, string, int, float
 - Small vectors
 - float2, float3, float4
 - Object references
 - ClassName etc
 - Not pointers

Language

- Variables and functions
 - Looks like in C++/Java
 - `int var = 3;`
 - `int fib(int n) ...`
 - Slight difference on functions
 - Function body can be a single statement
 - `int add2(int a) return a + 2;`
 - Concise, consistency with `if / while` etc.

Language

- Statements and expressions
 - Most like in C++/Java
 - foreach (ClassName obj)
 - Iterates over all existing objects of a certain class
 - Needs more capabilities
 - Lacking normal for
 - String concatenation operator
 - . . . like in Lua
 - Casting operator like in D
 - cast(Type) expr

Language

- **Classes**
 - Mostly like in Java, but
 - Inheritance is specified a bit differently
 - 'override' keyword needed to override functions
 - No constructors
 - Has default properties block (from UnrealScript)
 - Has variable replication block (from UnrealScript)

Language

- Example of syntax
 - Inheritance
 - extends ParentClass;
 - In class body
 - Can have multiple, order can matter
 - Default properties block
 - default { var1 = val1; var2 = val2; etc }
 - Uses 'defaultproperties' in UnrealScript

Language

- Why multiple inheritance?
 - Prevents large base classes
 - UT2004 Actor class, almost 2000 lines (UnrealScript)
 - Can write small modular components
 - Easily select what behavior an object should have
- Problem free?
 - No
 - Other solutions were considered

Language

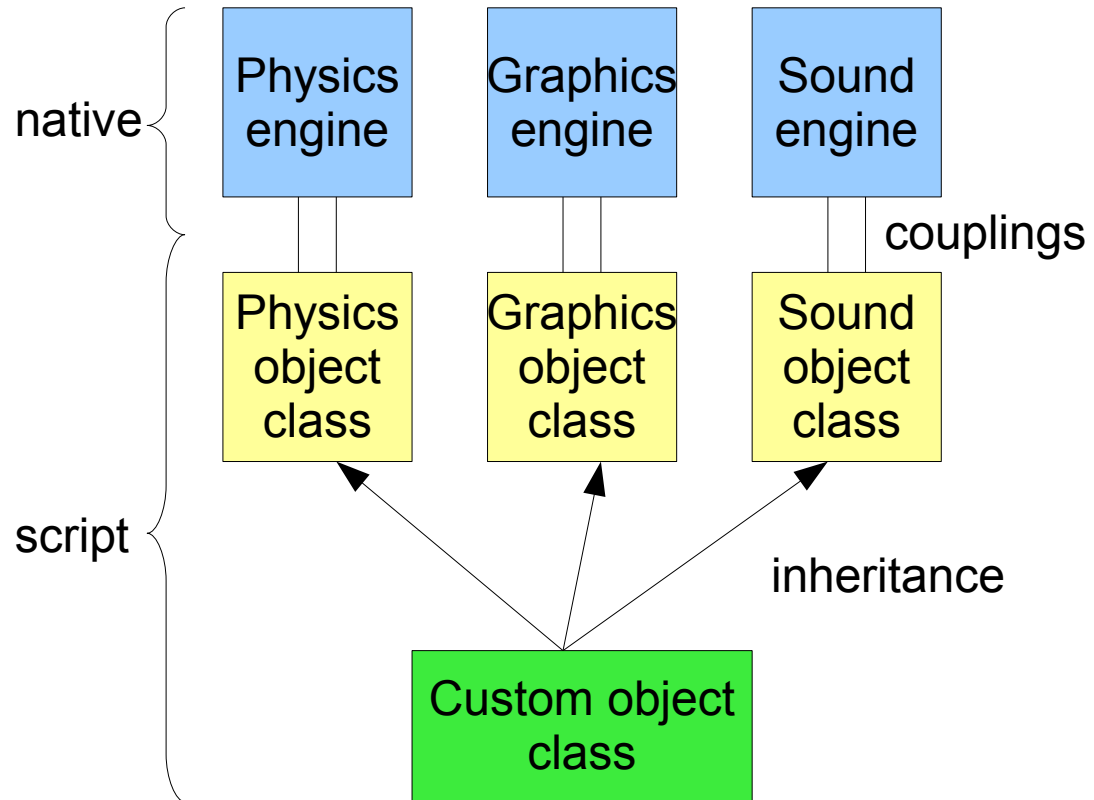
- Networking
 - Based on UnrealScript
 - For functions, specified with 'client' and 'server' modifiers
 - Calls only replicated to/from client owning object
 - For variables, specified in replication block
 - replication { if (cond) var1, var2; if (cond2) var3, var4; }
 - Syntax not finalized

Language

- Couplings to native code
 - Specifies how script code interacts with the game engine (or other system code)
 - Uses the 'native' modifier for functions, classes and member variables implemented in native code
 - native void playSound(Sound sound);
 - native string model;
 - native class Object ...
 - Uses the 'event' modifier for functions called from native
 - event void collidedWith(Collider c) ...

Language

- Strategy
 - Create base classes for game engine components
 - Physics, Sound, Graphics
 - Use base classes
 - Inherit from the components needed for the object



Implementation

- Type checker, interpreter
 - Visitor pattern over syntax tree
- Uses
 - C++
 - BNF Converter
 - Generates code for parser and syntax tree representation

Implementation

- API in C
 - So it can be used from many languages
 - Incomplete
- Command line tool
 - Can type check and test modules
 - Can write glue code for native couplings

Implementation

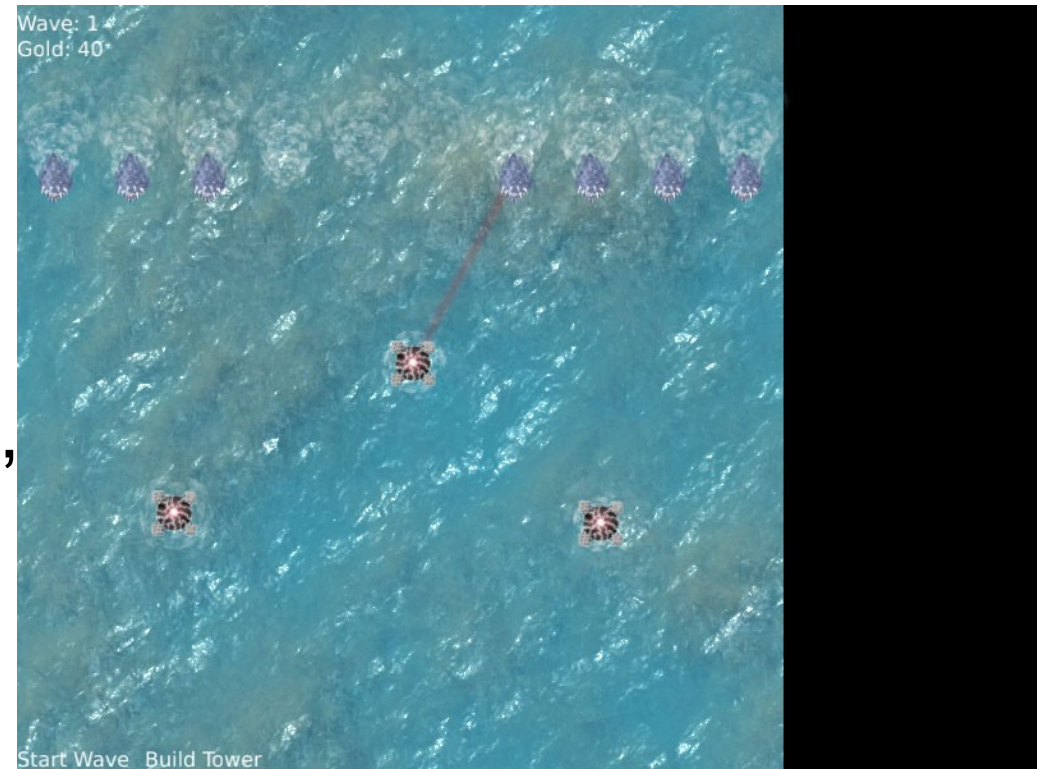
- Networking
 - Quake3 Networking model
 - Compares last known client state with the current, sends changes
 - No requirements of reliable transport layer protocol
 - Resends all function calls since last known client state, client ignores already executed
 - Incomplete, no client to server replication

Implementation

- Couplings
 - Glue functions generated from script code
 - Checks argument types, converts values etc
 - Calls certain functions expected to be implemented
 - A bit bothersome, could use improvements
 - Function to bind glue available to native code only

Evaluation

- Beginning of test game, Nerfling Rush
 - Using components was handy
 - Doing couplings was repetitive
 - Overall straightforward, but more work is needed



Evaluation

- Goals reached?
 - Primary points
 - Networking
 - Not completed
 - Organization
 - Fairly reached. Module system, not fully implemented lookups
 - Couplings
 - Works, but could be improved, more automated
 - Security
 - Language is safe, implementation is not (inf. iteration/recursion)

Evaluation

- Goals reached?
 - Secondary points
 - Time was not enough to look into them
- Could it be used?
 - It shows promise, but more work is needed
- Why wasn't it more completed?
 - More time consuming than was imagined
 - Maybe better had it used some existing system

Future work

- Completing implementation
 - API, networking, some of type checker/interpreter
- Looking into secondary points
- Adding general language features
 - Arrays, delegates, etc
- More test games
 - Especially a network game

Questions?

- More info, source and report will be available at
 - <http://jolle.se/cth/thesis/>
 - Scheduled to be up within a few days