WPI

IMGD 4000
Technical Game Development II
Scripting

Robert W. Lindeman
Associate Professor
Interactive Media & Game Development
Human Interaction in Virtual Environments (HIVE) Lab
Department of Computer Science
Worcester Polytechnic Institute
gogo@wpi.edu

WPI

Scripting

- Two* senses of the word:
  - Scripted Behavior
    - Having NPCs follow pre-set actions, rather than choosing them dynamically
  - Scripting Language
    - Using a dynamic language to make the game easier to modify
- The senses are related
  - A scripting language is good for writing scripted behaviors (among other things)

** also "shell scripts," which are not today's topic*

WPI

Scripted Behavior

- One way of building non-player character (NPC) behaviors
- What's the *other* way?
- Versus simulation-based behavior
  - E.g., goal/behavior trees
  - Genetic algorithms
  - Machine learning
  - etc.

Scripted vs. Simulation-Based Behavior

WPI

- Example of scripted behavior
  - Fixed trigger regions
    - When player/enemy enters predefined area
    - Send pre-specified waiting units to attack
  - Doesn't truly simulate scouting and preparedness
  - Easily found "exploit"
    - Mass outnumbering force just outside trigger area
    - Attack all at once

Scripted vs. Simulation-Based Behavior (cont.)

WPI

- Simulation-based (non-scripted) version
  - Send out patrols
  - Use reconnaissance information to influence unit allocation
  - Adapts to player's behavior (e.g., massing of forces)
  - Can even vary patrol depth depending on stage of the game

WPI

Advantages of Scripted Behavior

- Much faster to execute
  - Apply a simple rule *versus* run a complex simulation
- Easier to write, understand, and modify than a sophisticated simulation
- Fits well into our mental model
  - If this happens (trigger), then do that (action)

Disadvantages of Scripted Behavior

WPI

- Limits player creativity
  - Players will try things that "should" work (based on their own real-world intuitions)
  - Will be disappointed when they don't
- Allows degenerate strategies
  - Players will learn the limits of the scripts and exploit them
- Games will need *many* scripts
  - Predicting their interactions can be difficult
  - Complex debugging problem

Stage Direction Scripts

- Controlling camera movement and "bit players"
  - Create a guard at castle drawbridge
  - Lock camera on guard
  - Move guard toward player
  - etc.
- Better application of scripted behavior
  - Doesn't limit player creativity as much
  - Improves visual experience
- Stage direction can *also* be done by sophisticated simulation
  - E.g., camera system in God of War

Scripting Languages

- *You can probably name a bunch of them:*
  - Custom languages tied to specific games/ engines
    - UnrealScript, QuakeC, HaloScript, LSL, ...
  - General purpose languages
    - tcl, Python, Perl, Javascript, Ruby, Lua, ...
    - The "modern" trend, especially with Lua

Often used to write scripted behaviors.

Custom Scripting Languages

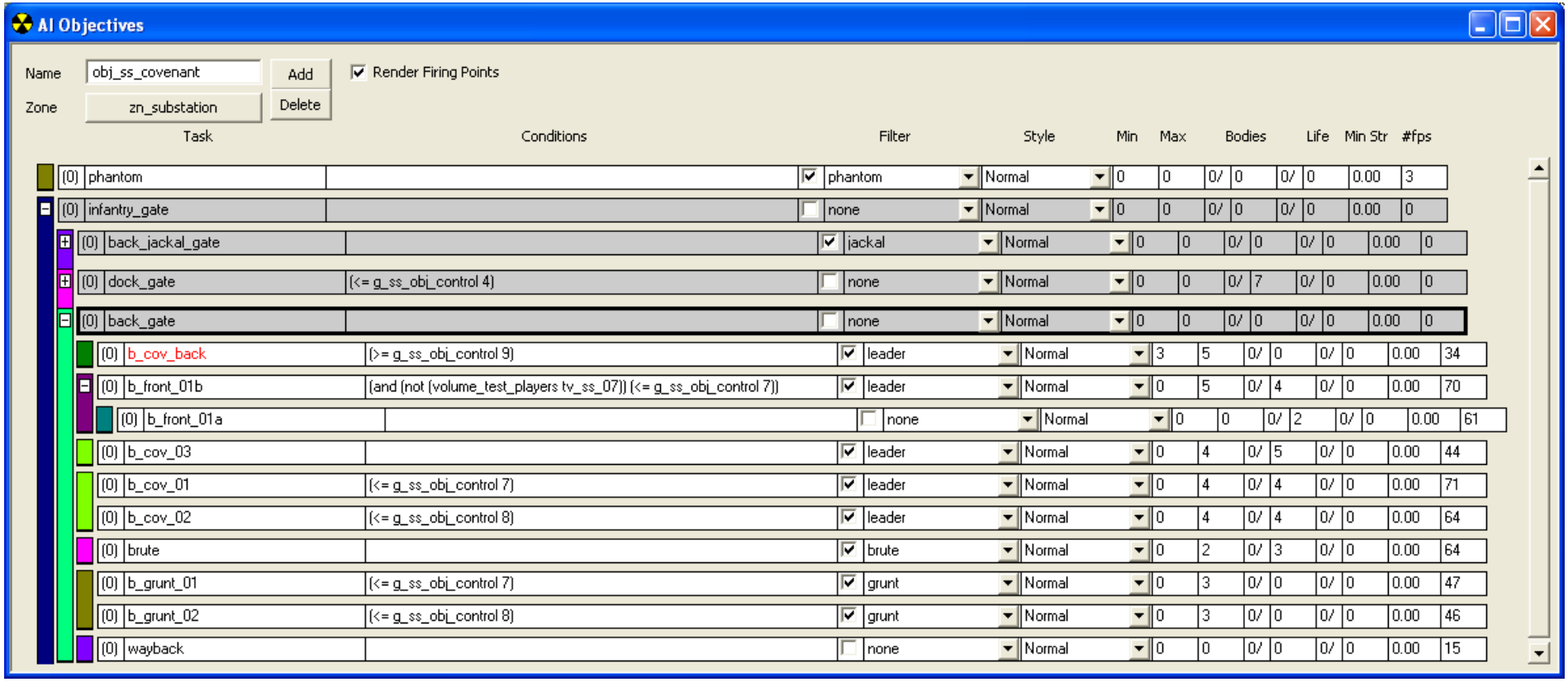
- A custom scripting language tied to a specific game, which is just idiosyncratically "different" (e.g., QuakeC) doesn't have much to recommend it

- However, a game-specific scripting language that is ***truly natural*** for non-programmers can be very effective:

```
if enemy health < 500 && enemy distance < our bigrange
        move ...
        fire ...
    else
        ...
    return
```

(GalaxyHack)

Custom Languages and Tools



"Designer UI" from Halo 3

General Purpose Scripting Languages

WPI

What makes a general purpose scripting language different from any other programming language?

- Interpreted (byte code, virtual machine)
  - Technically a property of *implementation* (not language per se)
  - Faster development cycle
  - Safely executable in "sandbox"
  - Recently JIT native compilation also
    (see http://www.mono-project.com/Scripting_With_Mono)
- Simpler syntax/semantics:
  - Untyped
  - Garbage-collected
  - Built-in associative data structures
- Plays well with other languages
  - e.g., LiveConnect, .NET, Lua stack

General Purpose Scripting Languages

WPI

But when all is said and done, it looks pretty much like "code" to me....☺

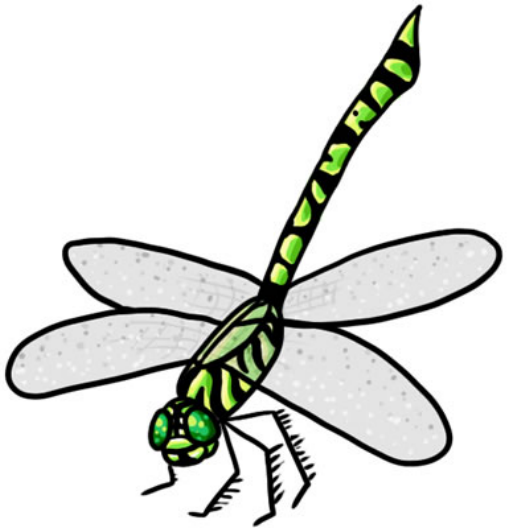
e.g., Lua

```
function factorial(n)
   if n == 0 then
     return 1
   end
   return n * factorial(n - 1)
end
```

So it must be about something else…

Now go back in time...

To the world of C++ engines....



Scripting Languages in Games

So it must be about something else...

Namely, the game development process:

- For the technical staff
  - Data-driven design (scripts viewed more as "data," not part of codebase)
  - Script changes do not require game recompilation

- For the non-technical staff
  - Allows parallel development by designers
  - Allows end-user extension

WPI

A Divide-and-Conquer Strategy

- Implement *part* of the game in C++...
  - The time-critical inner loops
  - Code you don't change very often
  - Requires complete (often very long) rebuild for each change
- ...and *part* in a scripting language
  - Don't have to rebuild C++ part when scripts change
  - Code you want to evolve quickly (e.g., NPC behaviors)
  - Code you want to share (with designers, players)
  - Code that is not time-critical (can migrate to C++ later)

General Purpose Scripting Languages

WPI

But to make this work, you need to successfully address a number of issues:

- Where to put *boundaries* (APIs) between scripted and "hard-coded" parts of game
- Performance
- Flexible and powerful debugging *tools*
  - Even more necessary than with some conventional (e.g., typed) languages
- Is it ***really*** easy enough to use for designers!?

Most Popular Game Scripting Language?

WPI

- **Lua**
- Has come to dominate other choices
  - Powerful and fast
  - Lightweight and simple
  - Portable and free
- See http://lua.org

117 Lua-scripted Games

(Wikipedia)

A
- Angry Birds
- Aquaria (video game)

B
- Baldur's Gate
- The Battle for Wesnoth
- Bet On Soldier: Blood Sport
- Bitfighter
- Blitzkrieg (video game)
- Blossom (video game)
- Brave: The Search for Spirit Dancer
- Brütal Legend
- Bubble ball
- Buzz!
- BZFlag

C
- Civilization V
- Company of Heroes
- Cortex Command
- Crackdown
- Crowns of Power
- Crysis

D
- Demigod (video game)
- Digital Combat Simulator
- Diner Dash

E
- Empire: Total War
- Enigma (video game)
- Escape from Monkey Island
- Etherlords series
- Eufloria
- Evil Islands: Curse of the Lost Soul

F
- Fable II
- The Fairly OddParents: Shadow Showdown
- Far Cry
- FlatOut (video game)
- FlatOut 2
- Foldit
- Fortress Forever
- Freeciv
- Freelancer (video game)

G
- Garry's Mod
- Grim Fandango

G cont.
- The Guild 2

H
- Tom Clancy's H.A.W.X
- Hearts of Iron III
- Hedgewars
- Heroes of Might and Magic V
- Homeworld 2
- Hyperspace Delivery Boy!

I
- Impossible Creatures
- The Incredibles: When Danger Calls

K
- King's Bounty: The Legend

L
- Lego Universe
- Linley's Dungeon Crawl
- Lock On: Modern Air Combat

M
- Mafia II
- MDK2
- Metaplace
- Minions of Mirth
- Monopoly Tycoon
- Multi Theft Auto
- MUSHclient

N
- Napoleon: Total War
- Natural Selection 2

O
- Operation Flashpoint: Dragon Rising

P
- Painkiller (video game)
- Plants vs. Zombies
- PlayStation Home
- Psychonauts

R
- Rail Simulator
- RailWorks
- Ratchet & Clank Future: Tools of Destruction
- Regnum Online
- Requiem: Memento Mori
- Richard Burns Rally
- RigidChips
- Rolando 2: Quest for the Golden Orchid
- Room for PlayStation Portable
- ROSE Online
- Runes of Magic

S
- S.T.A.L.K.E.R.: Shadow of Chernobyl

S cont.
- Ryzom
- Saints Row 2
- Shank (video game)
- Silent Storm
- SimCity 4
- The Sims 2: Nightlife
- Singles: Flirt Up Your Life
- SpellForce: The Order of Dawn
- Spring (game engine)
- Star Wars: Battlefront
- Star Wars: Battlefront II
- Star Wars: Empire at War
- Star Wars: Empire at War: Forces of Corruption
- Star Wolves
- StepMania
- Stratagus
- Supreme Commander (video game)
- Supreme Commander: Forged Alliance

T
- Tales of Pirates
- Tap Tap Revenge
- Teeworlds
- There (virtual world)
- Toribash

U
- ÜberSoldier
- UFO: Afterlight
- UltraStar
- Universe at War: Earth Assault

V
- Vegas Tycoon
- Vendetta Online

W
- Warhammer 40,000: Dawn of War
- Warhammer 40,000: Dawn of War II
- Warhammer Online: Age of Reckoning
- The Witcher (video game)
- User:Jinouk9500/World of Warcraft
- User:WilliamSewell/Xsyon
- World of Warcraft

X
- X-Moto

Y
- You Are Empty

Z
- User:ZukaVSD/Exoterra

WPI

Lua Language Data Types

- Nil – singleton default value, nil
- Number – internally double (no int's!)
- String – array of 8-bit characters
- Boolean – true, false

  Note: *everything* except nil coerced to false!, e.g., "", 0
- Function – unnamed objects
- **Table** – key/value mapping (any mix of types)
- UserData – opaque wrapper for other languages
- Thread – multi-threaded programming (reentrant code)

Lua Variables and Assignment

- Untyped: any variable can hold any type of value at any time

```
A = 3;
A = "hello";
```

- Multiple values
  - in assignment statements

```
A, B, C = 1, 2, 3;
```

  - multiple return values from functions

```
A, B, C = foo( );
```

"Promiscuous" Syntax and Semantics

WPI

- *Optional* semi-colons and parens

```
A = 10; B = 20;
A = 10  B = 20
  A = foo( );
  A = foo
```

- *Ignores* too few or too many values

```
A, B, C, D =  1, 2, 3
A, B, C  = 1, 2, 3, 4
```

- Can lead to a debugging *nightmare!*
- *Moral:* Only use for small procedures

Lua Operators

- Arithmetic: + - * / ^
- Relational: < > <= >= == ~=
- Logical: and or not
- Concatenation: ..

... *with usual precedence*

WPI

Lua Tables

- Heterogeneous associative mappings
- Used a lot
- Standard array-ish syntax
  - Except any object (not just int) can be "index" (key)

```
mytable[17] = "hello";
mytable["chuck"] = false;
```

  - Curly-bracket constructor

```
mytable = { 17 = "hello", "chuck" = false };
```

  - Default integer index constructor (starts at 1)

```
test_table = { 12, "goodbye", true };
test_table = { 1 = 12, 2 = "goodbye", 3 = true };
```

Lua Control Structures

- Standard if-then-else, while, repeat & for
  - with break in looping constructs

- Special for-in iterator for tables

```
data = { a=1, b=2, c=3 };
for k,v in data do print(k,v) end;
```

produces, e.g.,

```
a   1
c   3
b   2
```

(order undefined)

WPI

Lua Functions

- Standard parameter and return value syntax

```
function (a, b)
    return a+b
end
```

- Inherently unnamed, but can assign to variables

```
foo = function (a, b) return a+b; end
foo(3, 5)  ➔  8
```

Why is this important/useful?

- Convenience syntax

```
function foo (a, b) return a+b; end
```

WPI

Other Lua Features ...

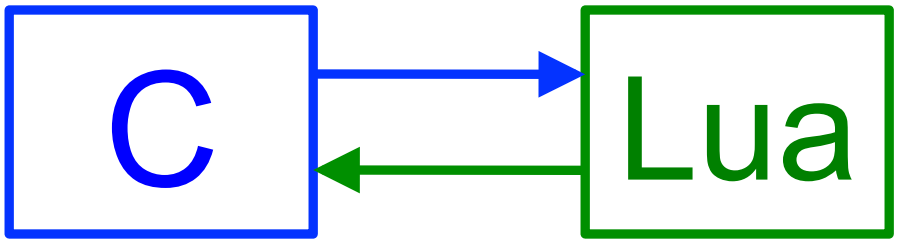
- Object-oriented style (alternative dot/colon syntax)
- Local variables (default global)
- Libraries (sorting, matching, etc.)
- Namespace management (using tables)
- Multi-threading (thread type)
- Bytecode, virtual machine
- Features primarily used for language extension
  - Metatables and metamethods
  - Fallbacks

See http://www.lua.org/manual/5.2

WPI

But Lua cannot stand alone...

- **Why not?**



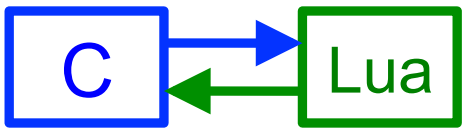
- Accessing Lua from C++
- Accessing C++ from Lua

WPI

Connecting Lua and C++

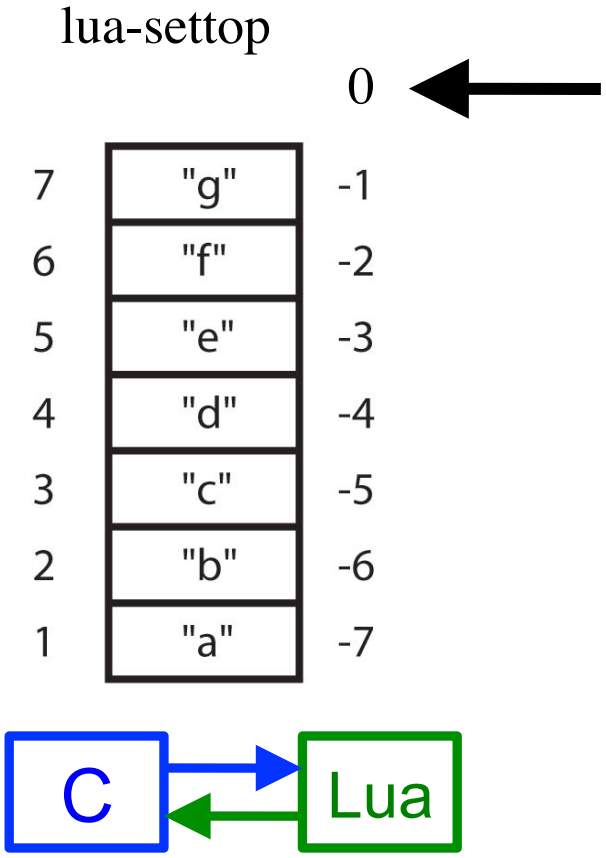
- Lua virtual stack
  - Bidirectional API/buffer between two environments
  - Preserves garbage collection safety

- Data wrappers
  - UserData – Lua wrapper for C data
  - luabind::object – C wrapper for Lua data



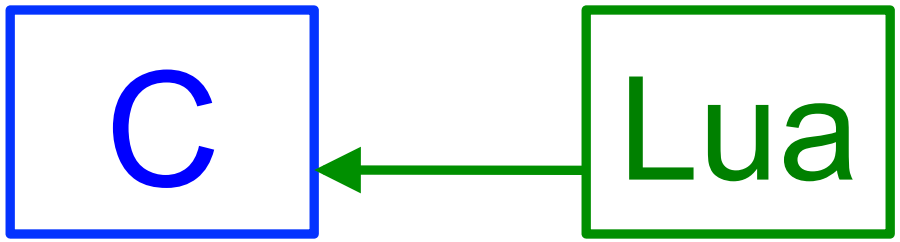
Lua Virtual Stack

- Both C and Lua env'ts can put items on and take items off stack
- Push/pop or direct indexing
- Positive or negative indices
- Current top index (usually 0)



WPI

Accessing Lua from C



Accessing Lua Global Variables from C

WPI

- *C tells Lua to push global value onto stack*

```
lua_getglobal( pLua, "foo" );
```

- *C retrieves value from stack*
  - *using appropriate function for expected type*

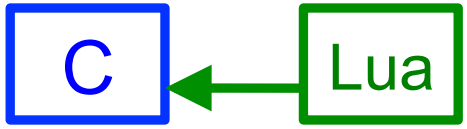
```
string s = lua_tostring( pLua, 1 );
```

  - *or can check for type*

```
if ( lua_isnumber( pLua, 1 ) )
  { int n = ( int )lua_tonumber( pLua, 1 ) } ...
```

- *C clears value from stack*

```
lua_pop( pLua, 1 );
```



Accessing Lua Tables from C (w. LuaBind)

WPI

- *C asks Lua for global values table*

```
luabind::object global_table = globals( pLua );
```

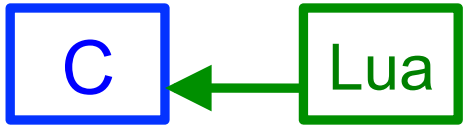
- *C accesses global table using overloaded [ ] syntax*

```
luabind::object tab = global_table["mytable"];
```

- *C accesses <u>any</u> table using overloaded [ ] syntax and casting*

```
int val = luabind::object_cast<int>(tab["key"]);

tab[17] = "shazzam";
```



Calling Lua Functions from C (w. LuaBind)

WPI

- *C asks Lua for global values table*

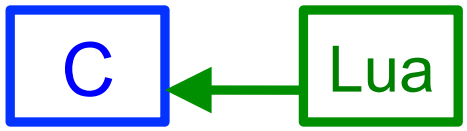
```
luabind::object global_table = globals( pLua );
```

- *C accesses global table using overloaded [ ] syntax*

```
luabind::object func = global_table["myfunc"];
```

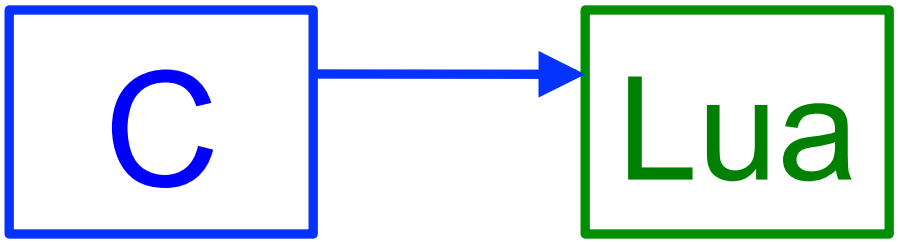
- *C calls function using overloaded ( ) syntax*

```
int val =
  luabind::object_cast<int>( func( 2, "hello" ) );
```



WPI

Accessing C from Lua



Calling C Function from Lua (w. LuaBind)

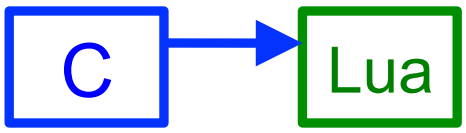
WPI

- *C "exposes" function to Lua*

```
void MyFunc ( int a, int b ) { ... }
module( pLua ) [
        def( "MyFunc", &MyFunc )
];
```

- *Lua calls function normally in scripts*

```
MyFunc( 3, 4 );
```



[See more details and examples in Buckland, Ch 6.]

So what's all this got to do with Unity?

WPI

- The game engine core of Unity is coded in C++...
- Javascript (a close cousin of Lua) is provided as a "scripting language"
- So this is the same paradigm we have been discussing, except that you never have to (get to ☺) recompile the C++ part!

WPI

Thanks Chuck!

- Thanks to Chuck Rich for this material!