

Sol 2

COMS 6156

Topics in Software Engineering

Iris Zhang - iz2140
JeanHeyd Menaïd - jm3689

I. Intro

Motivation

In early 2014 a Github user by the name of Rapptz, an oncologist with an interest in C++ released Sol 1.0. It was first used because he was helping out a computer science professor with a lightweight wrapper for the “Ninja” build system, an alternative to “Make.” It was necessary that the entire compilation fit on a thumb drive.

JeanHeyd’s motivation for making the project stemmed from his curiosity towards Lua and how it is used in game programming. He already had an interest in C++ and searched for an opportunity to learn Lua and put his skills in library development to use. He saw that many applications need to use Lua from their C++ applications, and are thus forced to use the Lua C API which demands a person understand how to manipulate the Lua stack and get used to a number of implementation details. Thus he took over the Sol 2 project from Rapptz.

Iris joined the project in mid-March as she had already been using Lua in game development. In addition, she was taking Language and Library design in C++ and wanted a way to combine the type safety and classes of C++ and the ease of use of Lua. She wanted to see if Sol was the right fit.

Overview

In a nutshell, a Lua binding library for C++. You can easily import a sol.h header into a C++ file and immediately begin using Lua. Sol eliminates the boilerplate of using the C API to run/ use Lua code in C++, and offers powerful features to let C++ be used in Lua. It also provides flexibility with its compatibility: it doesn't matter which version of Lua you use. Performance-wise, Sol is one of the best among its competitors. This library would be useful for any existing C++ developers who need access to the Lua API. In particular, anyone who has need for a similar product such as luawrapper, lua-intf, luabind, Selene, oolua, SLB, SWIG, etc. would probably be better off switching to Sol instead.

Deliverables

- Sol2 Library and source files: <https://github.com/ThePhD/sol2>
- Sol2 Documentation and website: <http://sol2.readthedocs.org/en/latest/>
- Sol2 Demonstration Project / Sidescrolling game: <https://github.com/iz2140/Sol2Test>
 - branch “master”: SFML project only
 - branch “UsingSol”: Sol and SFML project

To build Sol2 for any machine, follow the instructions here: <http://sol2.readthedocs.io/en/latest/tutorial/getting-started.html>

To build the Demo Project, you must have Xcode installed with SFML, Lua, and set the linker flags, build settings, architecture, etc as outlined in the Appendix.

1. Using SFML with XCode: <http://www.sfm-dev.org/tutorials/2.3/start-osx.php>
2. Install Lua locally: <http://www.lua.org/manual/5.2/readme.html>
3. Place the lua directory containing the install folder that was produced from this step 2 into the root directory of the Xcode project.

4. Install Sol2 and place sol.hpp and Sol source files into the same directory that main.cpp will run in.

II. Features

Outline

In a nutshell, a lua binding library for C++. You can easily import a sol.h header into a C++ file and immediately begin using lua. The following is a simple outline of features that Sol supports:

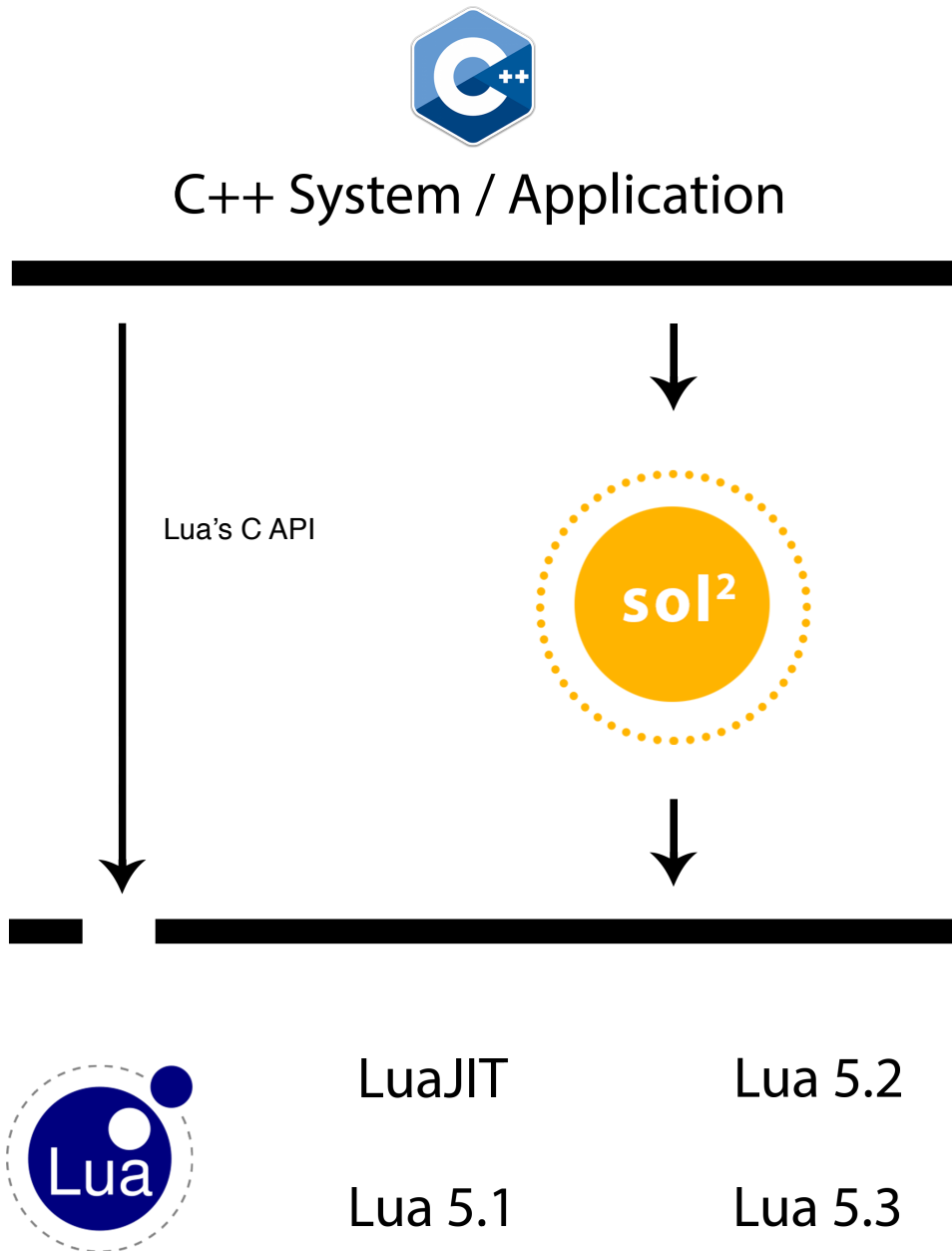
- Lua 5.1, 5.2, and 5.3.
- Table support: setting values, getting values of multiple (different) types
 - Lazy evaluation for nested/chained queries
 - Implicit conversion to the types you want
- Support for callables (functions, lambdas, member functions)
 - Pull out any Lua function with `sol::function`
 - Can also set callables into `operator[]` proxies
 - Safety: use `sol::protected_function` to catch any kind of error
 - *Advanced*: Overloading of a single function so you don't need to do boring typechecks
- User-Defined Type (`sol::usertype` in the API) support:
 - Set member functions to be called
 - Set member variables
 - Set variables on a class that are based on setter/getter functions
 - Use free-functions that take the Type as a first argument (pointer or reference)
 - Support for “Factory” classes that do not expose constructor or destructor
 - Modifying memory of userdata in C++ directly affects Lua without copying, and
 - Modifying userdata in Lua directly affects C++ references/pointers
 - If you want a copy, just use value semantics and get copies
- Thread/Coroutine support
 - Use, resume, and play with coroutines like regular functions
 - Get and use them even on a separate Lua thread
 - Monitor status and get check errors
- *Advanced*: Customizable and extensible to your own types if you override getter/pusher/checker template struct definitions.

For full feature documentation, see: <http://sol2.readthedocs.org/en/latest/features.html>

Architectural Design

A C++ application can sit on top of Sol to interact with Lua. Sol was written as an interface on top of Lua's C API. Note that as a header only library, the Lua C API can still be

interacted with directly (it is not hidden as an implementation detail in a compiled library) if the user desires. Sol is API-agnostic, so covers all versions of Lua and works with all of them equally well.



III. Results

Documentation

While before there was barely any documentation, now there is plenty. In addition, the documentation that existed was better organized and formatted. Documentation can be roughly split into three portions: Tutorial, Feature details, and API documentation. In addition more support and attention was given towards writing more beginner-friendly documentation aimed at complete novices. A “quick n’ dirty” tutorial and “getting started” portion was added with more detailed instructions for installation and building. While the library itself is powerful and has great features, an objectively worse library (Selene) was getting more attention. We really improved documentation in the hopes it would spur more users to contribute and use our library: <http://sol2.readthedocs.io/en/latest/index.html>

Sol 2.5

a fast, simple C++ and Lua Binding

When you need to hit the ground running with Lua and C++, **Sol** is the go-to framework for high-performance binding with an easy to use API.

build passing

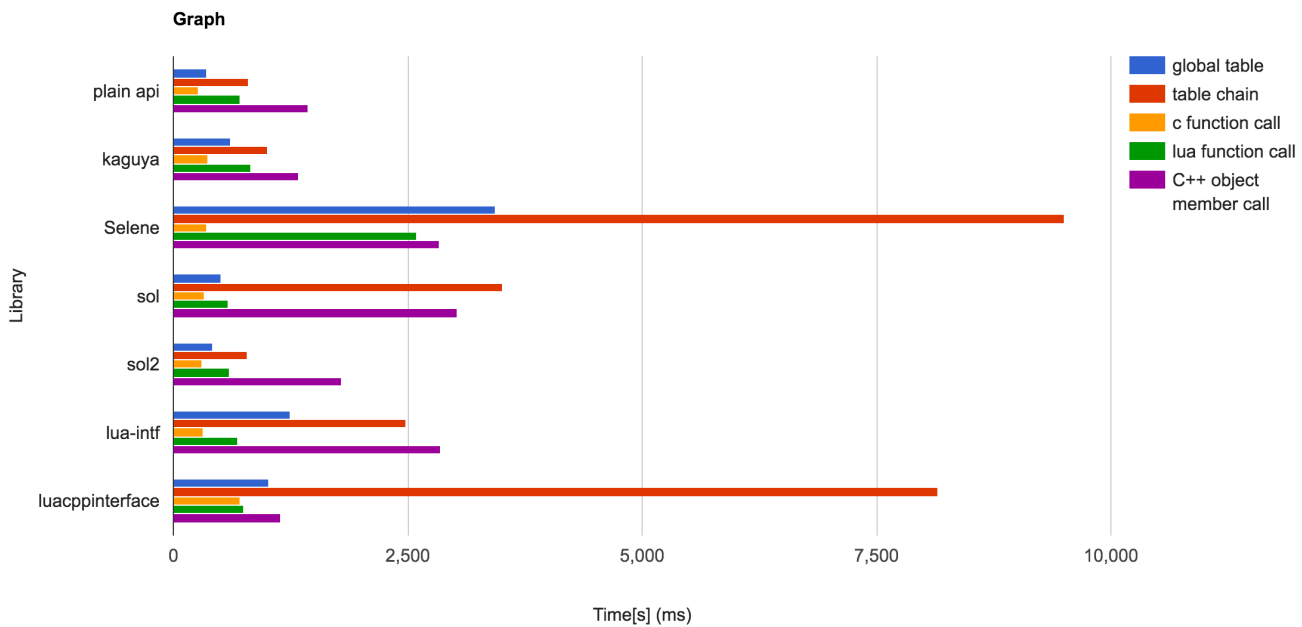
get going: ¶

- tutorial: quick ‘n’ dirty
- tutorial
- api reference manual
- features
- benchmarks
- safety
- exceptions
- run-time type information (rtti)
- licenses
- origin

Benchmarks

Performance-wise Sol increased its rank by various measures. The first shows a graph showing benchmark tests provided by a third party, Github name satoren, taken from this following site: http://satoren.github.io/lua_binding_benchmark/. Sol’s performance can be measured against Sol2’s performance, which markedly improved in global table, table chaining, and C++ object member call.

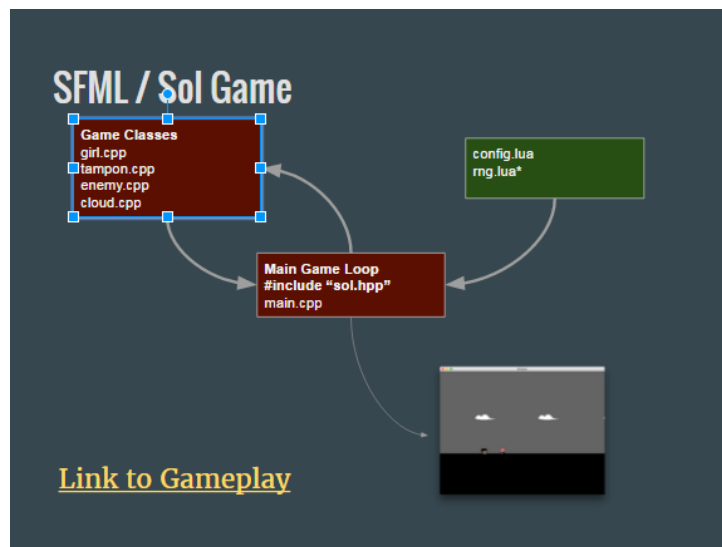
For all benchmarks, see <http://sol2.readthedocs.org/en/latest/benchmarks.html> for the direct source code, data, and all the specifications for the test.



Demo Project

For the demonstration project, Iris coded a side-scroller based on a volunteer project she did for a client that was based in Lua using CoronaSDK. The basic idea of the game is that the player would control a protagonist on the screen that can shoot bullets. Enemies will randomly generate and move in different speeds towards the protagonist. When a bullet hits an enemy, that counts as collision detection and both enemy and bullet disappear. This project is sufficiently complex enough to demonstrate C++ classes, object lifespan, 2D sprite animations, and UI adjustments to user input while simple enough to code for a student project.

For external tools, the C++ game framework chosen was SFML (<http://www.sfml-dev.org/>). Sol needed a C++ binary so any game for this demo project must compile to C++. For development environment, Xcode was used as it is the only known tutorial for how to compile SFML on Mac OS X. Compilation and building issues took up a significant portion of time, so Iris' perception of Sol's ease of use definitely went down a notch.



The project's architecture is as such: one main game loop that kept track of user input to the keyboard and updated the GUI accordingly. Every object on the GUI was controlled by a C++ class related to that object, such as the girl, the tampon bullets, the enemies, and the clouds. These made up all the SFML-related files, represented in red in the above diagram. These were coded up first solely in SFML, and then Sol was incorporated later as to show a comparison of the code. The Sol files are represented in green in the above diagram, and were used to cut down on the use of magic number constants in the game.

```
config.lua
1 game = {
2     initialScore = 10,
3     tampons = 10,
4     enemies = 10,
5 }
6
7 window = {
8     width = 1600.0,
9     height = 1200.0,
10    frameRateLimit = 60
11 }
12
13 girl = {
14     img = "girlWalk.png",
15     width = 59,
16     height = 64,
17     animFrames = 2,
18     startingPosition = { x = 800, y = 600}
19 }
20
21 tampon = {
22     img = "ammo.png",
23     width = 73,
24     height = 16,
25     frames = 4
26 }
27
28 enemy = {
29     img = "enemy.png",
30     width = 55,
31     height = 60,
32     animFrames = 2,
33     startingPosition = { x = 1600, y = 600}
34 }
35
36
```

Iris discovered that while magic numbers are easily tucked away in a config file, retrieving them in the C++ game required the user to still use magic strings that corresponded to the Lua keys used. (Meaning, in the C++ file in order to access the girl's X starting position, it must be accessed using something like `girl["startingPosition"]["x"]` which would require one to know those strings from the start. However there is really no way around this.) The type safety offered by C++ however is unparalleled compared to Lua.

Iris also suggests that for future projects, SFML should not be used for rapid game development as the C++ language itself is hurdle enough without trying to wrangle GUI code on top and do heavy lifting of memory management. If Sol were written as a Lua header that could easily bring C++ classes

into Lua projects, she would recommend it more readily. As of now, Sol is more fit to be brought into projects already written in C++ that requires Lua integration. If there was time, Iris would write an Unreal game that demonstrated extensive use of Sol.

Tools & libraries: Xcode, Git, SFML, Lua 5.2, Sol

Future Development

Sol is still receiving a lot of new interest from users across the globe. This is an open source project that will continue to grow as long as JeanHeyd maintains interest and remains committed in improving the project. As of the time this report was written, this is a snapshot of the current stars, watchers, forks, and open issues on the Github repository:

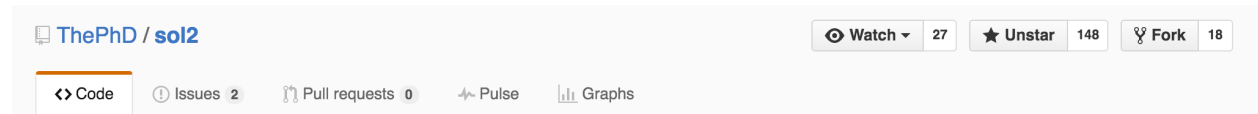


Figure: 148 Stars!

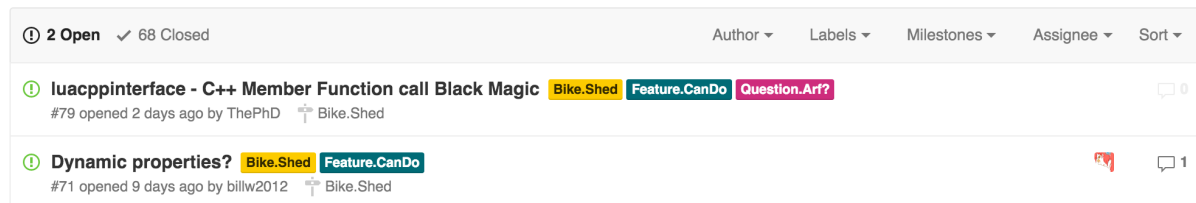


Figure: Open issues

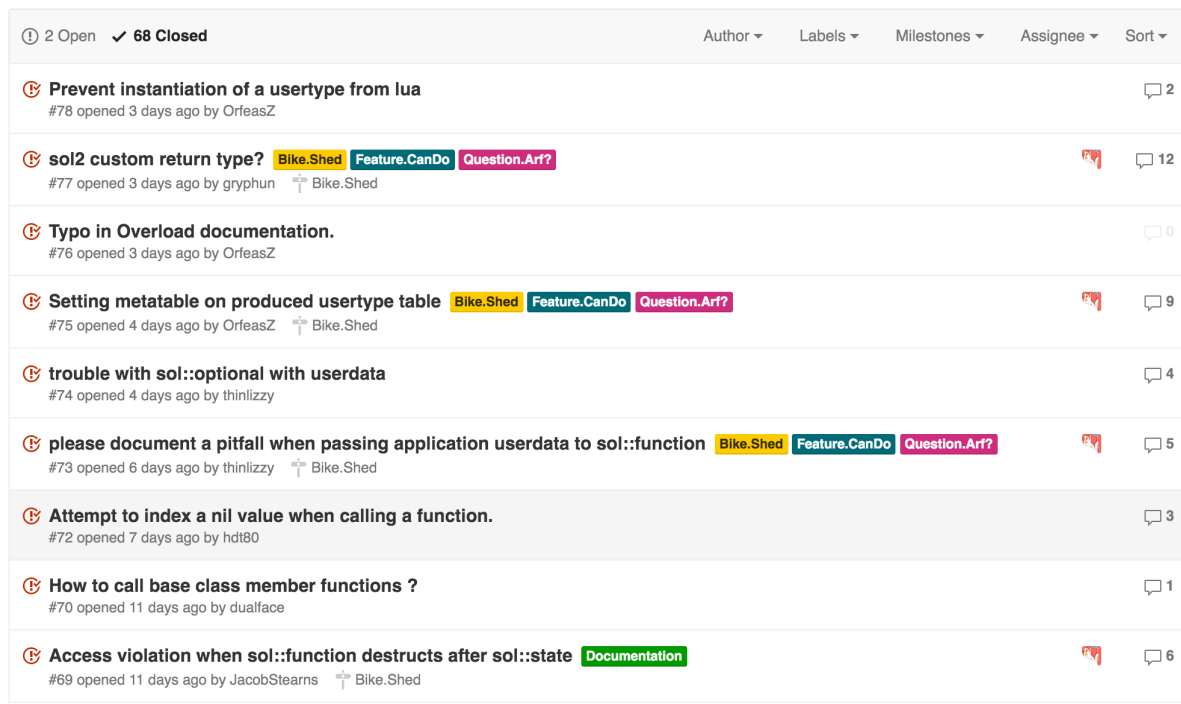


Figure: Recently closed issues



"What problems will you encounter will happen if a new version of Lua is released?" Lua has a (somewhat bad) habit of breaking old APIs simply for what can only be OCD reasons and not providing useful back-compatibility for them enabled by default in vanilla builds of lua. We have restricted ourselves to a specific subset of lua functions and we also use keplerproject/lua-compat (<http://sol2.readthedocs.io/en/latest/api/compatibility.html>) as a built-in, hand-modified dependency to create an abstraction layer that will insulate us between lua version changes. It has so far worked when Lua was upgraded from Lua 5.2 to 5.3.

IV. Project Plan

Goals

JeanHeyd planned to improve documentation, benchmark testing, and continue to add new features and improve the popularity of Sol 2. By all means he wildly succeeded and will continue to do so, although there is plenty of work to still do.

Iris planned on testing out Sol 2 as a way to bring together C++'s type safety and object oriented goodness with Lua's high performance, easy syntax, and neat scripting abilities. Unfortunately, she hit a wall with trying to make the project compile and build (10 hours give or take), and in wrangling with C++ and SFML itself. She became somewhat frustrated trying to do very simple things in SFML, such as create a sprite and move it across the screen and destroy it when it moves off. As a result, she started digging around for Youtube tutorials and resources online when she came across this forum post by one of the developers of SFML:

Nexus
SFML Team
Hero Member

Posts: 6093
Thor Developer



Re: General SFML help?
« Reply #3 on: July 14, 2013, 08:56:55 pm »

Do not watch Youtube videos. The ones from CodingMadeEasy contain sometimes really questionable and ancient C++ style; if you are a newcomer to the language, they will do more harm than good. The problem is, most Youtube tutorials concerning programming are written by people with a lot of time, but not necessarily with a lot of experience. Many think they do something good for the community (and their intent is very noble!), but as hard as it sounds, in the end people get used to a lot of bad habits that are difficult to get rid of.

In C++, the only option is really to take a **good** book (a lot of literature is not recommended, either) and to invest the time to read it thoroughly. As you have recognized, C++ is very complex -- even when you're an expert in other languages, it will be difficult to learn it.

But if you are not interested in C++, it is probably not worth the effort to learn it. It will take months until you know the basics, and then you still have to learn all the idioms, best practices and different paradigms to be able to program efficiently. Only do it if you really feel motivated, otherwise you can use an SFML binding for a different language.

« Last Edit: July 14, 2013, 08:59:21 pm by Nexus »

 Logged

In particular, notice he says it takes “months” just to learn the basics of C++, and only then once one becomes comfortable with the language should they use SFML for C++. Otherwise he recommends using a different SFML binding. In essence, most of this headache could have been avoided with some forethought and research, but for the purposes of this project Iris is glad to have learned a bit about programming games in C++ and SFML.

Roles

- Iris: Demo application designer, Tutorial documentation, Presentation, Final report
- JeanHeyd: Library designer, Library builder, Documentation, Presentation

Timeline

- March 29: Project proposal submitted
- April 8: Iris forks Sol2 project from Github and begins following JeanHeyd's tutorial to get started with compilation
- April 12: Meet and figure out compilation issues on Xcode 7.2/Mac OS X
- April 16: Fix all compilations and have minimal working prototype for SFML game
- April 10-now: JeanHeyd updates documentation to reflect all the issues they ran into : <http://sol2.readthedocs.org/en/latest/tutorial/getting-started.html>
- April 10-now: JeanHead continues building out test cases and benchmark testing for Sol2: <http://sol2.readthedocs.org/en/latest/benchmarks.html>
- April 21: Minimum viable product for the Demo game works with SFML and compiles with Sol
- April 22: Make the Main player code into its own C++ class
- April 23: Add ability to shoot ammo
- April 24: Ammo now moving across screen
- April 25: Adding enemy class, collision detection, finish up SFML game, incorporate Sol
- April 25: Work on presentation and Sol 2 logo
- April 26: Presentation

Lessons Learned

Iris: Don't try to learn C++ for games unless you really know what you're doing! You will need to know low-level memory management and OpenGL first. Getting anything to compile requires knowing where libraries get installed on your particular machine, then linking them properly to the binary. Sometimes this isn't completely obvious so it's always better to rely on manual compile before writing a Makefile or relying on an IDE. Sol is a cool library but it probably fits a very niche use at the moment.

JeanHeyd: The most enlightening part of this was understanding the difference between users. For example, user Bulat-Zenshin wrote on a github issue "that sol is a header-only library (that don't need compilation). that's enough - we know how to run compiler and use git". Conversely, many others -- including Iris -- needed some tips for how to pull the entire repo (including submodules), how to set up their include paths, and get the compiler going. In the end, for the documentation -- despite Bulat-Zenshin's advice -- we ended up taking the lowest common denominator out of everyone's abilities. The goal of the documentation is to make sure that everyone can access the library: assumptions as to skill level and competency with C++, git or various tools isn't really beneficial, especially in a "Getting Started"-style of tutorial. Following from that, it was clear that documentation was sovereign when it came to helping out users. Even the API documentation, while not the best starting point, turned out to be integral for documenting corner cases in the library and have proper notification. Better tutorials are probably still a thing that we need as well, since despite having a fairly decent "Quick 'n' Dirty" we still have users opening issues with basic questions about how to use things!

IV. Appendix

1. Xcode build settings needed for running demo:

▼ Architectures

Setting

SolTest2

► Additional SDKs	
Architectures	Native Architecture of Build Machine (x86_64) - \$(NATIVE_ARCH_ACTUAL) ⚙
Base SDK	No SDK (Latest OS X) ⚙
Build Active Architecture Only	No ⚙
▼ Supported Platforms	OS X ⚙
Debug	OS X ⚙
Release	OS X ⚙
Valid Architectures	i386 x86_64

> | SolTest2

Info

Build Settings

ECT

SolTest2

ETS

⌘ SolTest2

Basic

All

Combined


Levels

+

Q~

Dead Code Stripping	No ⚙
Display Mangled Names	No ⚙
Don't Dead-Strip Inits and Terms	No ⚙
Dynamic Library Install Name	
Dynamic Library Install Name Base	
Exported Symbols File	
Generate Position-Dependent Executable	No ⚙
Initialization Routine	
Link With Standard Libraries	Yes ⚙
Mach-O Type	⚙
Order File	
Other Librarian Flags	
▼ Other Linker Flags	-l sfml-system -l sfml-window -l sfml-graphics -l sfml-audio -l sfml-network
Debug	-l sfml-system -l sfml-window -l sfml-graphics -l sfml-audio -l sfml-network
Release	-l sfml-system -l sfml-window -l sfml-graphics -l sfml-audio -l sfml-network
Other Text-Based API Flags	
▼ Path to Link Map File	<Multiple values>
Debug	build/SolTest2.build/Debug/build/-LinkMap--.txt
Release	build/SolTest2.build/Release/build/-LinkMap--.txt
Perform Single-Object Prelink	No ⚙
Prelink libraries	
Preserve Private External Symbols	No ⚙
Quote Linker Arguments	Yes ⚙
Re-Exported Framework Names	
Re-Exported Library Names	
Re-Exported Library Paths	
Runpath Search Paths	
Separately Edit Symbols	No ⚙

Filter

	Basic	All	Combined	Levels	+
st2	Strings File Output Encoding				UTF-16 ▾
	Wrapper Extension				
st2					
	▼ Search Paths				
	Setting				 SolTest2
	Always Search User Paths				Yes ▾
	Framework Search Paths				/Library/Frameworks/
	▼ Header Search Paths				<Multiple values>
	Debug				/usr/local/include/
	Any Architecture Any SDK ▾				./lua-5.3.2/**
	Release				/usr/local/include/
	Any Architecture Any SDK ▾				./lua-5.3.2/**
	▼ Library Search Paths				<Multiple values>
	Debug				/usr/local/lib/
	Any Architecture Any SDK ▾				./lua-5.3.2/**
	Release				/usr/local/lib/
	Any Architecture Any SDK ▾				./lua-5.3.2/lib/
	Rez Search Paths				

2. External Software used for testing Sol:

- Travis-CI: Continuous integration (<https://github.com/travis-ci/travis-ci>)
- Github - philsquared/Catch: Testing library for C++ (<https://github.com/philsquared/Catch>)
- Github - rmartinho/nonius: Benchmarking framework (<https://github.com/rmartinho/nonius>)
- Github - satoren/lua_binding_benchmark: another library author's benchmark (https://github.com/satoren/lua_binding_benchmark)

3. External references in writing demo game:

- SFML Documentation (<http://www.sfm1-dev.org/documentation/2.3.2/>)

4. Thanks:

- Rapptz who originated it all
- All of Sol's open source contributors
- Professor Kaiser and the TA's