Welcome to the Moai SDK open beta. This guide will walk you through getting started with Moai and give an overview of the state of the project.

Background

Moai SDK is a 2D game engine written in C++ and scriptable through Lua. It is an excerpt of a personal project that began life several years ago as a Maya scene graph exporter and 3D game engine. To make the Moai SDK as useful as possible and ensure no one using Moai hits a brick wall in their game development, we've decided to release Moai SDK under the free to use, open source CPAL license.

Moai SDK is for experienced game developers and designers who know their way around a game engine. Our goal is to make development fast, not to 'sugar coat' the development process. Finding the best way to organize your pipeline and make a cool product is still up to you.

You do not need Moai SDK to use Moai Cloud; Moai Cloud is our new (and very cool) cloud platform built around Lua that makes it easier to build the back-end for great mobile games. If you use Lua in your pipeline now and need any kind of server back-end, you should check out Moai Cloud regardless of whether or not you use Moai SDK. Since you can talk to Moai Cloud over HTTP, any game engine should be compatible. We'll be sharing more of our plans for Moai Cloud in the weeks ahead.

Moai SDK is ready for use in commercial projects. That said, please keep in mind that Moai SDK is still making the transition to production quality code. We'll be tightening the screws and optimizing Moai during the open beta, but since you have source access you can also tailor Moai to fit the needs of your project. Even though there's still a ways to go and there will always be frustrations, I think you'll find learning and working with an open source project to be a much better investment than basing your success on a proprietary solution.

By building an open source community around Moai we plan to quickly flesh out the feature set, add contributed tools, harden the code and surpass closed source alternatives.

Getting Started

This document is written for programmers, but we'll also talk about how to set up Moai for use by your designers.

To use Moai SDK, first get a copy of the binary SDK or the source tree. These contain four reference projects for building Moai: vs2008, vs2010, Android/NDK and Xcode. If

you download the binary SDK you will also have precompiled hosts for Windows and OSX as will as static libraries for all currently supported platforms and devices.

To run Moai on a given platform, you will need a 'host.' A Moai host is a application that creates an OpenGL view into which Moai will render and exposes input device events to the Moai runtime. Moai ships with reference hosts for each supported platform. Please note that the reference hosts are only examples; for a commercial game you should implement your own host with any device specific features you need.

The Moai host examples bind to Moai through an API called Aku. Aku wraps up access to the Moai runtime in a single C language header. If the existing Moai framework meets your project needs, you only need to work with Aku.

If you are working on a team with scripters or game designers, they will probably want to use the Moai binaries to do their work. If they don't mind building from Xcode or Eclipse, they can prototype directly on the device. Alternatively, they can use a desktop host.

The desktop host that ships with Moai is based on GLUT, a freely available library for writing OpenGL demos. The GLUT host runs as a 'native' app on the desktop, not as a phone simulator: we've only exposed the desktop input devices supported by GLUT – keyboard, mouse pointer and mouse buttons.

As of this writing we have not provided any 'simulator' style Moai host intended to mimic phone style inputs (multi-touch, accelerometer, GPS). Of course these features work and are supported by the iPhone and Android projects; there just isn't a simulation solution at this time. Game designers can work around this by branching their input handling code. We're also hoping members of the community step up to provide tools like device simulators and editors.

Learning Moai

Start by looking at AKU.h (moai-beta/src/aku). Aku is your one stop shop for binding Moai to a new host. It is a single header with no dependencies on Moai that exposes the Moai runtime through a C-language interface.

Now look at the basic samples (moai-beta/samples/basics). These samples aren't very exciting, but do provide a minimalist demonstration of the Moai Lua framework. Run each sample and also look at the Lua code. If any of the object initializations seem complex, remember that you can write Lua wrappers or adapt a tool to output the setup.

Once you've been through the samples, take a look at the included documentation of the Lua framework. You can download an offline copy of this documentation or build it yourself from the source tree (note that both doxygen and a command line build of Moai are needed as pre- and post- processing is performed using Moai/Lua as a parser).

As you get more comfortable with the framework, a good next step is to write some simple Lua wrappers to make setting up Moai objects easier.

Finally, read my 'Extending Moai' doc (look in the docs folder of the source tree) and try writing your own Lua-bound objects.

Project Status

Even though I've been working on the original codebase that spawned Moai for several years, Moai in its current, 2D-centric state is a fairly new animal. In this section I'd like to call your attention to the places where immediate work is still needed.

Sound

Moai's sound library is called Untz, and it was developed for Moai by San Francisco based mobile developer Retronyms. It is still a work in progress, but offers basic sound support on Windows, OSX and iPhone. Android support is coming in the next few weeks. On Windows, Untz uses DirectSound for backwards compatibility with Windows XP. On older releases of Windowsyou may need to install the latest DirectX SDK or get the DirectX redistributable.

There is also a Moai extension available for FMOD. FMOD is a well-designed, robust, industry standard library. If you want to use FMOD, you will need to build your own host and link in the FMOD framework objects and Aku interface. Be aware that we haven't used FMOD for about a year and make no guarantee that the binding provided is fully functional.

Our focus moving forward will be on continuing to develop Untz into a robust, cross platform alternative.

Particles

Moai's built-in particle engine is nearing stability in terms of feature set and use. It is very powerful, but still one of our more expensive features. You can use the bytecode scripting interface to prototype particle effects, but may need to re-implement your particle scripts as C language plug-ins for performance reasons. We're looking at some better solutions that will speed things up dramatically without changing the interface or requiring plug-ins. Stay tuned.

If you already license a C or C++ particle engine and tools, you'll find that integrating it with Moai is a snap. Look at the Box2D and Chipmunk integrations to see how we've handled 3rd party simulation packages. Also take a look at the Moai particle system. While we're going to be concentrating on the design of Moai's own particles, we'd be delighted to lend a hand to anyone who wants to write a binding for an existing system and offer it to the community; there are some excellent commercial particle tools out there.

Low Level Graphics API

We've been concentrating on sprite decks and animation objects to make it easier for designers and scripters to do their work quickly. These objects hide a lot of OpenGL, but are still fairly low level. For example, we don't do anything to hide texture limitations and UV coordinates from designers. If your designers aren't that technical, you may want Lua wrappers to help them. My own philosophy is that system resources are precious enough that designers should be aware of technical issues and learn to work with them.

If you look closely at the Moai framework, you'll see that we've started to sketch in low level 3D graphics support. For example, there's a dynamic vertex buffer object that works with a vertex format object to let you specify and draw arbitrary OpenGL vertex data. Static vertex buffer, index buffer and frame buffer objects are also coming soon.

You'll also see that we've included a shader node, even though we only support OpenGL ES 1.1 right now. The current implementation uses the fixed function pipeline, but by the end of the beta we should have support for OpenGL ES 2.0 and the programmable pipeline. At that stage, all drawing state will be gathered into and managed by shader nodes. Also, don't be surprised if our affine transform objects gain a row and a column.

High end features like occlusion culling and n-pass lighting won't be a priority for a while, but 3D remains very much in the DNA of Moai, so it will come in a future release - but not before support for OpenGL 2.0 and desktop OpenGL.

GUI Objects
Moai has pretty robust support for text layout and fonts, but the focus has been in-game usage, not productivity apps. My thought is that UI elements like lists, buttons, spinner controls, etc. can all be implemented in Lua or exposed to Lua as native controls.

We'd love to see someone spearhead the creation of a completely cross platform GUI toolkit written on Moai. We've added some stubs to support this (the layout object, the stretch patch object and the built-in support for frame-fitting in Moai's drawing objects), but I don't anticipate that the Moai team will be building a UI widget framework in the near term.

OpenSSL
Moai's HTTP interface uses libcurl. We have recently integrated OpenSSL to support HTTPS. We have also integrated luacrypto, luasocket and luacurl extensions.

Serialization
Moai's object serialization was removed during our last big refactor. You'll still see the stubs all over the place. The same is true for the printing functions used to convert Moai objects to formatted strings. Both the serialization and pretty printing will be making their way back into Moai soon.

In the meantime, you can still use Moai's built in serializer to read and write graphs of Lua tables. You can also use 3rd party Lua serializers (such as pickle and pluto), but these will not be aware or of support the Moai object model.

Extending Moai
A full discussion of ways to extend Moai is outside the scope of this document. You'll find a draft of the Moai extension guide in the docs folder of the source tree.

Future Directions

While we have our own ideas about what we'd like to see in Moai, the needs of the Moai community are more important. Personally, I'd really like Moai to be more than just an animation engine and graphics compatibility layer. While users will be able to accomplish a lot just by extending Moai in Lua, I'd like Moai to offer a much richer set of game engine components, even including APIs optimized for specific types of gameplay.

How You Can Help

The best way to help is to use Moai to make great games and to share your experiences with the community. The forums are up and ready for your input at www.getmoai.com/forums

Bug reports and feature requests are always welcome. Bug patches and platform specific optimizations are welcome as well.

We also welcome you to contribute framework extensions, bindings and tools and would love to feature them (and your studio) on our site. We appreciate open source contributions, but integrations with closed source, commercial products (such as FMOD) are welcome as well; we have no particular ideology to espouse other than the desire to offer more options for users of Moai.

Thanks for reading.

Patrick Meehan
patrick@ziplinegames.com