Welcome to the Moai SDK 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 as a cross platform development framework 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 with Moai 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.

Of course you do not need Moai SDK to use Moai Cloud; Moai Cloud is a different and very cool technology. 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.

Moai is ready for use in commercial projects. Groundspeak, Inc. released a proof of concept app based on an earlier version of Moai in 2010 and we are working on several commercial titles now at Zipline Games. 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 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 closed source 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 the 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 code base. To make managing the project and accepting patches easier, we are using github to host the code. You can download the code using git or svn, but either way you'll need a github account:

1. Sign up for a free account with github.
2. Send your github username to us at support@ziplinegames.com. We will give you access to the moai-beta repository.
3. **Using git:** Follow the instructions on 'github' to clone the repository. You can use either ssh or http to get the repository. If you anticipate pulling versions of the repository frequently, we recommend ssh.
4. **Using svn:** Use your favorite svn client to check out the repository from http://svn.github.com/moai/moai-beta.git. You will be asked to enter your github credentials. You can check out using svn, but if you want to submit a patch you will need to use git.

In the code base you will find four reference projects for building Moai: vs2008, vs2010, Android/NDK and Xcode. Right now we support building to Windows, Android and iPhone. The reference projects are the easiest way to build Moai and the 3rd party libraries. (There is also a cmake project coming soon. Stay tuned).

There are some differences between the reference projects and what they build:

1. **vs2008, vs2010:** The Visual Studio projects build Moai for Win32. The x64 configuration in these projects is for future use and is not supported at this time. The Visual Studio projects build the Moai library (static and DLL) and (mostly static) binaries for Moai dependencies. They also build a sample Moai host app called moai.exe. This is an example written using GLUT; for commercial projects you will want to write your own Moai host.
2. **Xcode:** The Xcode project builds a static library for Moai. An example host application is in samples/iphone. The example implements an OpenGL view that encapsulates an instance of the Moai runtime. It also binds Moai to iPhone input devices such as the touch screen, accelerometer and GPS. You can use this sample as is or as a starting point for a custom Moai host.
3. **Eclipse:** The libmoai project within the Eclipse folder builds a shared library for Moai. An example host application called MoaiTest is also included. The example implements an OpenGL view that encapsulates an instance of the Moai runtime. It contains hooks to bind Moai to Android input devices such as the touch screen, accelerometer and GPS, but they still need to be implemented. You'll need to build the contents of the libmoai folder to create the Moai library before the example app will run. For convenience, an ANT script (build-libmoai.xml) is included in the sample project to accomplish this. You can use this sample as is or as a starting point for a custom Moai host. Note: The current Android emulator does not support OpenGL, so you'll need to test on an Android 2.2 device for now.

All of 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. As of this writing we have a Windows installer for the binaries and reference host app (moai.exe). The installer was written with NSIS. If you write your own Windows host, you can adapt this installer for your designers to use.

We haven't yet provided binaries or an installer for Mac. If your designers are working on Mac then they can use the Xcode environment and simulator to test their work or you can build a version of the GLUT sample app for them.

Note that the Moai GLUT binding we've provided is intended to run as a 'native' app on Mac or PC, not as a phone simulator. We've mapped in the native 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.

Our designers have been able to write their scripts to run on phone or PC by substituting the mouse pointer for a single screen touch. This has been OK for our test apps, but clearly isn't ideal. We'd like to offer an official phone simulator down the road, but in the meantime you'll need to break out your favorite GUI framework and tackle this yourself (if it's important to you).

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 all the Moai runtime and its controls 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 Moai's most commonly used objects. 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 Moai object setup.

Once you've been through the samples, take a look at the included documentation of the Lua framework. You can also 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 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 ships with sound support for FMOD. FMOD is a well-designed, robust, industry standard library with (we think) extremely reasonable licensing fees. If you want to use FMOD, just enable it in moaiconfig.h and extend the existing binding to meet your needs. Be aware that we haven't used FMOD for about a year and make no guarantee that the binding provided is fully functional.

The existing FMOD wrapper is minimal at best and we aren't planning to support it moving forward. We are working with San Francisco based iPhone developer Retronyms to write an open source alternative to FMOD. We'll be presenting that as a stand-alone library with tight integration into Moai's object model in the near future.

Particles

The existing particle system is also a work in progress. It is powerful, but incomplete and still difficult to use. High on our list of priorities is cleaning up its interface, providing a higher level object model for it and finding ways to integrate with popular particle designer tools.

If you have 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 (how many top ten iPhone games are 3D, again?), but 3D remains very much in our blood and the DNA of Moai, so it will come in a future release.

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 plan to integrate OpenSSL to support HTTPS as well as a full set of encryption algorithms. In the meantime, there are already some popular Lua extenstions for crypto, ssl and networking. You may find these meet your needs.

Serializaion
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 are new to github and still learning the ropes, but our understanding is that you can check in patches and send us pull requests that will invite us to diff your code and integrate your changes into our project.

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