

**Technical Design Document**

Team Mocha

GAM300A

Fall 2014

Development Team

Jonathan Butt Producer and Audio Programmer

Nicholas Fuller Technical Director and Game Designer

Kai-Chuan Hsiao Physics Developer and Gameplay Programmer

Kenneth Tan Graphics Developer

Art Team

JiYing Zhao Art Director

Michael Kan Concept

Audio Team

Geoffrey Walker Sound Designer

Contents

[Overview 3](#_Toc405531337)

[Architecture 3](#_Toc405531338)

[Intercommunication 4](#_Toc405531339)

[Barista Threading 4](#_Toc405531340)

[Graphics 5](#_Toc405531341)

[Game-Specific Tech 5](#_Toc405531342)

[Physics 6](#_Toc405531343)

[Audio 6](#_Toc405531344)

[Scripting 6](#_Toc405531345)

[Coding Methods 7](#_Toc405531346)

[Repository 7](#_Toc405531347)

[Debugging 8](#_Toc405531348)

[Tools 9](#_Toc405531349)

[Appendix A 10](#_Toc405531350)

[Mocha Studio 10](#_Toc405531351)

# Overview

## Architecture

*Subsystem-oriented component based architecture with an emphasis on a clear distinction between front and backend.*

**From Bottom-To-Top:**

**Mocha Core**

**Managed-To-Native Layer**

**Barista**

**Subsystems**

**Components**

**Engine-To-Managed Layer SharpMocha**

**MochaDotNET Mocha Script**

**In Detail:**

|  |  |
| --- | --- |
| **Mocha Core** | Handles entry point to the engine. Routes calls to the Barista based on Editor Modes and start up preferences. |
| **Managed-To-Native Layer** | PInvoke wrapper to the C++ Engine. |
| **Barista** | Core Task Management for the entire engine. Runs on a dependency system and drives all systems. |
| **Subsystems** | Logic-oriented systems, e.g. Physics, GFX, etc. |
| **Components** | Data-oriented components. Interface with Subsystems in order to change internal state. |
| **Engine-To-Managed Layer** | Marshalling Layer for C++->C# invokes. |
| **MochaDotNET** | Auto-generated PInvoke interface layer. |
| **SharpMocha** | Collection of interface (.i) files to generate MochaDotNET |
| **Mocha Script** | C# based scripting language using interface built from MochaDotNET |

## Intercommunication

Communication Guidelines are as follows:

|  |  |
| --- | --- |
| Component -> Component | Not Allowed – Dependency Required |
| Component -> Subsystem | Allowed |
| Subsystem -> Component | Loose Coupling Allowed |
| Subsystem -> Subsystem | Not Allowed |

## Barista Threading

At the core of the C++ engine is the Barista, a clever name for a robust scheduler. We prefer to refer to this as a scheduler instead of the more common task manager. It’s functionality mimics that of an OS scheduler.

Barista organizes tasks/systems based on their dependencies and then delegates them to threads appropriately. The scheduling can be very smart about this, delegating such that the burden of synchronization is lifted from the user.

Barista also reports high-level profiling statistics for debugging use.

Barista handles both permanent operations, such as GFX, as well as transient ones, such as loading assets. There is little difference in the client side code other than to report when the operation has finished and what operation it would conflict with.

For instance, our main loop is registered mostly in serial with a few exceptions

//generate main loop in the barrista

barrista().RegisterToPrimary(GSM::getInstance());

barrista().RegisterToPrimary\_AfterLast(Renderer::getInstance());

barrista().RegisterToPrimary\_AfterLast(mFrameController);

barrista().RegisterToPrimary\_AfterLast(AudioManager::getInstance());

barrista().RegisterDebugOutputToParallel(AudioManager::getInstance());

barrista().RegisterToPrimary\_AfterLast(InputManager::getInstance());

barrista().Register\_Parallel(mPhysicsEngine);

When registering to the Barrista, operation must extend from either ISubsytem or Task:

class PhysicEngine : public ISubsystem

struct Task\_AvailableAsset\_Refresh : public Task

And overload the appropriate functions.

## Graphics

The Graphics Subsystem uses Deferred DirectX 11 with a slew of visual effects to give us an advanced system for rendering. We support physically based rendering in a partial form, with full support to come. Each object may have a physically accurate material roughness and shininess. Reflections are currently pre-generated, but we are close to real time reflections. In addition, the graphics systems support Frustum culling, multiple render windows, and camera render textures. We render to the Oculus Rift simultaneously with our primary and secondary render target renders. A variety of post process effects are applied to the targets depending on the application. Effects include: Motion Blur, Gaussian Blur, Bloom, Pixelated Censoring, and Film Grain.

Assets are loaded through our Asset Manager which handles all the runtime reloading logic and hot-loading.

## Game-Specific Tech

Subject REDACTED is a technically complex game. We spent most of the first semester preparing for the challenges we will face. There are certain features we need that will be listed in this section.

* Ability to render to multiple viewports at the same time.
* Ability to render to texture many different cameras and swap them out in realtime.
* Ability to quickly produce content with Mocha Studio.
* Ability to handle multiple audio listeners.
* Ability to handle multiple windows.
* Overall requirement to maintain a playable FPS despite extreme rendering overhead.
* Ability to perform multiple post process passes based on which camera is being rendered.

## Physics

Our physics engine currently supports Static and Rigidbody objects. It is using GJK/EPA for collision detection and Constrain Base for Resolution.

Broadphase culling, a type of AABB tree, provides most of our optimization.

In the future, we hope to implement OctTrees to further segregate our large levels.

As raycasting is used extensively in gameplay code, we spent the majority of development time producing a robust system for doing so. You may filter groups of objects to raycast to based on tags and proximity to origin point.

## Audio

The Audio system provides a wrapper around the FMOD Studio API. The Roaster generates “vente” files for each FMOD event provided. These files encapsulate the event’s GUID for quicker lookup. On load the Audio Manager loads all the bank files in the asset folder. The CSound component acts as an interface to the Audio Manager. Upon request to create an audio file from CSound, the Audio Manager then loads the event by getting the GUID of the event (located in the .vente file) and passing it into FMOD. The Audio Manager keeps a record of the audio assets created for quicker access if requested for later.  
  
The CSound component can store multiple handles to audio files, all of which can be played at the same time. The component can request to play, stop, pause, resume, set loop, set volume or create an audio file.

## Scripting

The entire engine is wrapped in an interface layer for C#. The wrapper will be immediately familiar to developers who have used Unity in the past because many concepts are borrowed from it. Possible functions to call are documented in Engine/interface/\*.i files and may be used as a reference for what is possible with the scripting.

Cross-script invoking is possible by providing a function name and game object to invoke to. A ‘hint’ to the script class type may be given as a template argument to increase performance. Future improvements will include direct variable access and a fast calling convention for inter-script activity.

## Coding Methods

Variable Names Camel

Function/Class Pascal

Components must be prefixed with ‘C’.

Systems must contain word ‘System’ in them.

Inter-communications require wrapper classes.

Client-side code must be:

* Input sanitized
* Commented
* Exception-safe

## Repository

Mercurial HG

Because of the relative modularity of our code and size of team, we have not found the need to layout any heavy guidelines of repository usage.

## Debugging

Mocha Studio has multiple debugging means. Aside from a simple console, there is also an interact-able in-editor console. Here you can find runtime script errors, compile issues, and more.

Debug Drawing is provided in both scripts and the engine with capabilities to draw Spheres, Lines, Cubes, and Meshes. See *DebugDrawer* for details.

Assertions are not used in the Engine above extremely low-level code. Due to the nature of the integration with the editor, recoverable errors are prioritized.

In the editor, you can find debug statistics reported by Barista in a handy graph.

All debug controls are run by Barista, merely unregistering or never adding the operation will cease all output.

## Tools

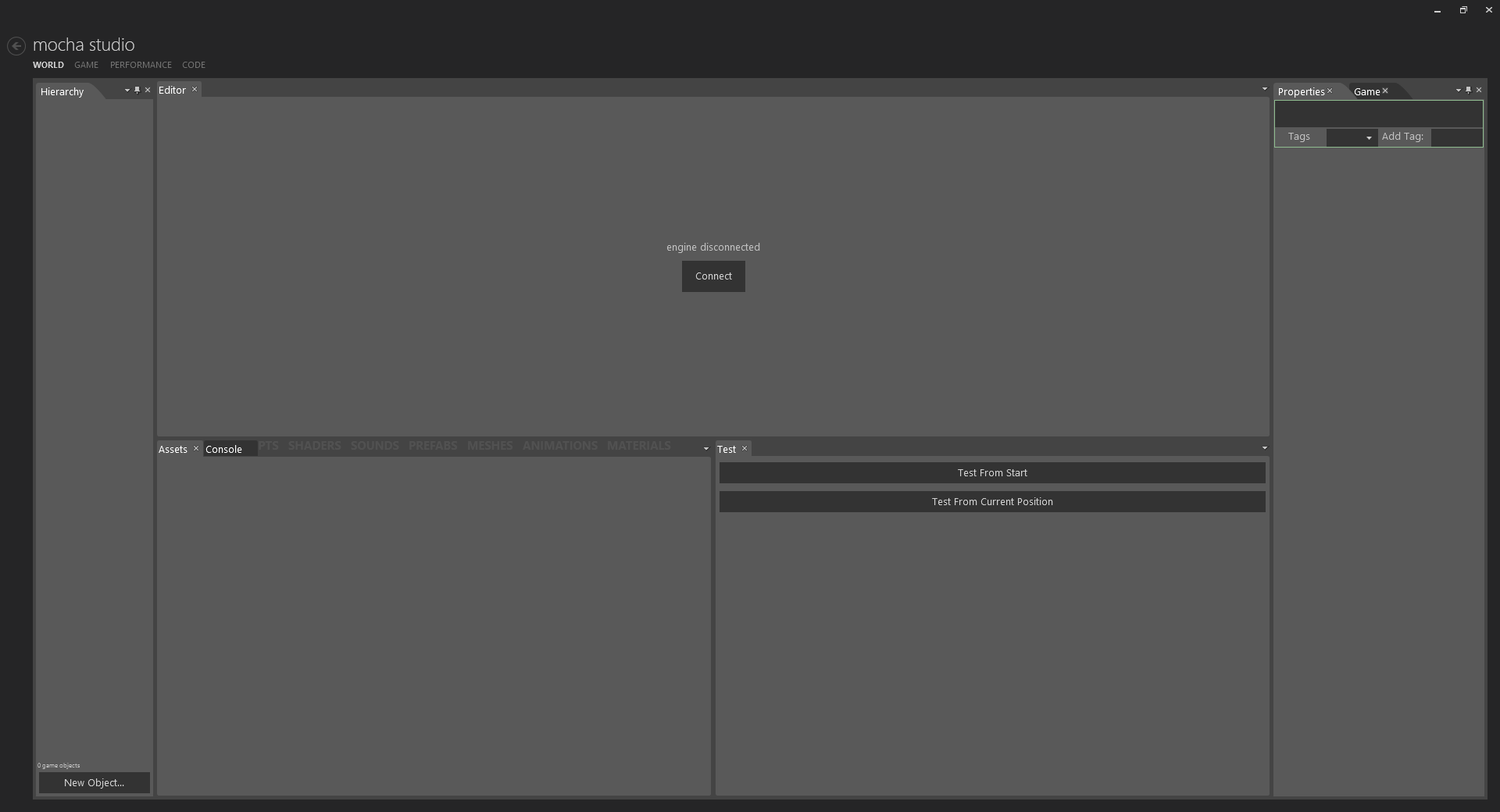
Mocha Studio comes with a full-fledged editor capable of making content without ever leaving the interface.

Technically, Mocha Studio is a WPF app with a near-codeless backend running a variety of custom controls. It features a material editor, code editor, performance viewer, game only run mode, asset manager, script reflection, scene editing, map save/load/merge and more.

See *Appendix A* for details on use of Mocha Studio.

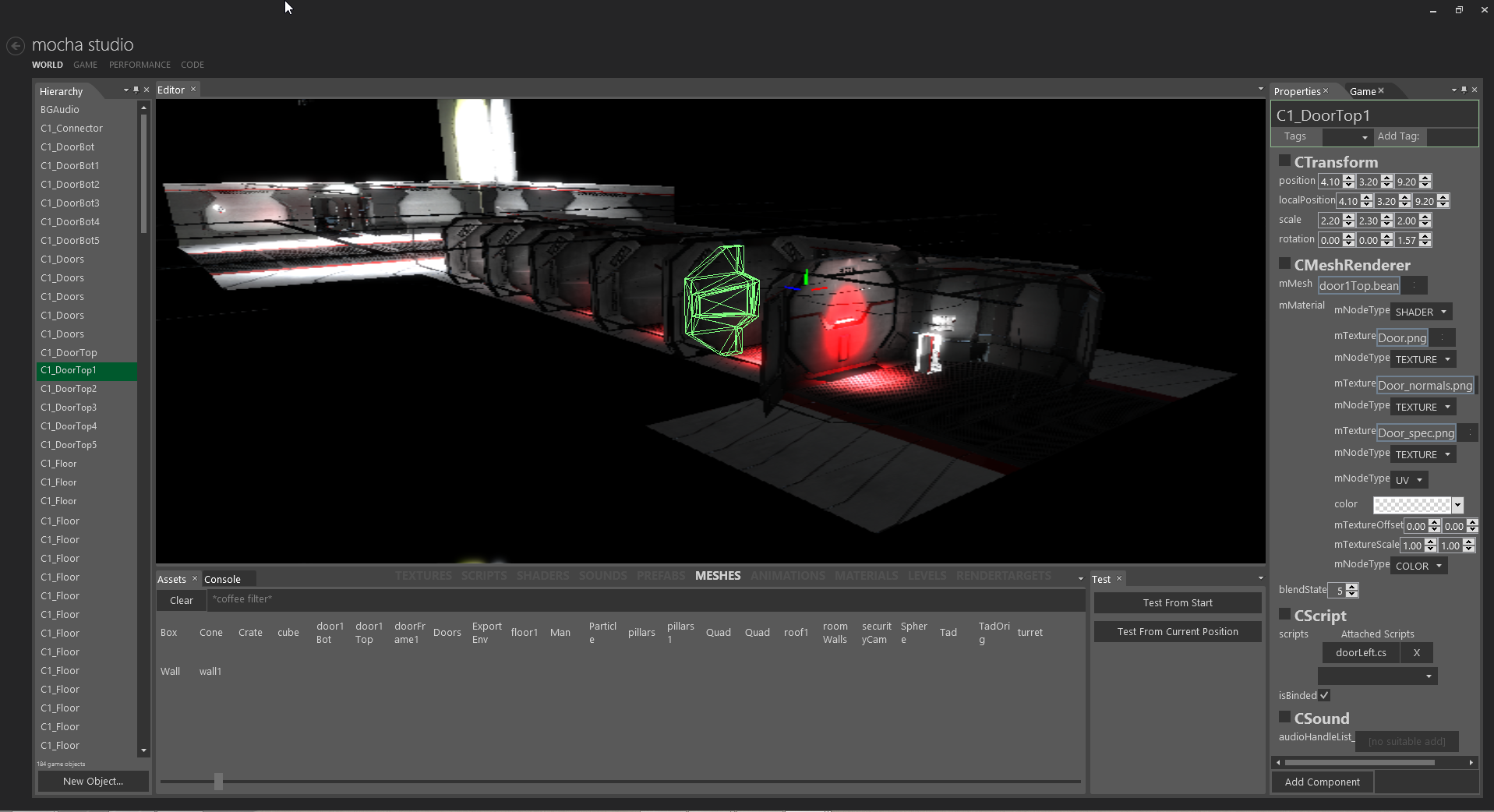
# Appendix A

## Mocha Studio

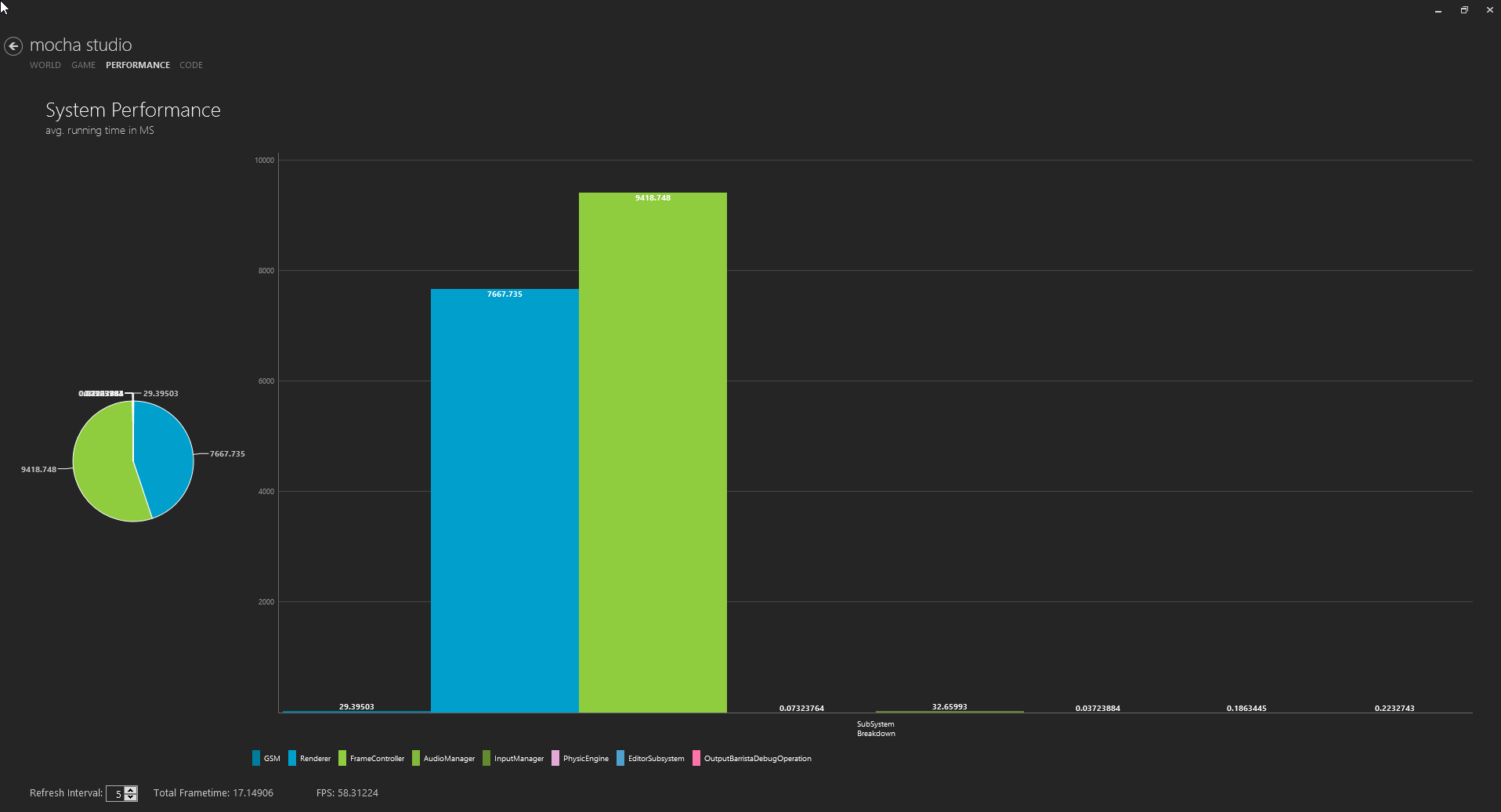


To begin using Mocha Studio:

1. Run Mocha Editor.exe
2. Press Connect



Once the engine has connected to the editor, you may begin content creation.

* The Hierarchy Panel on the left shows all the game objects in the scene
* The Properties Panel on the right shows the properties of the currently selected object
* Ctrl or shift-click to select multiple objects.
* Selected objects’ meshes will be drawn to show selection.
* Widgets will appear around the base of the last selected mesh
* W – Translate, E – Rotate, R – Scale (to switch modes)
* Right clicking on a game object will give a list of options such as delete and duplicate.
* At the bottom of the screen is the AssetManager, drag items from here on AssetHandle boxes in the Properties Panel to switch meshes, textures, etc.
* Click Test from Start to begin testing. (Test from Current Position not currently functional)
* The second tab, Console, shows scripting output from the current testing session as well as errors and warnings.
* Hotkey: F2 duplicates an object.
* Performance tab will show you a breakdown of the current running systems. 
* Code will give you access to the code editor, where you may drag scripts from the Asset display onto the editor.
* Ctrl-S saves and reloads a script.