Nu Game Engine (pre-pre-preview)

Bryan Edds, 2014

# Table of Contents

Table of Contents 2

What’s It All About? 3

Basic 3

Basic 3

Purely-Functional(ish) 3

2d Game Engine 3

F# 3

Getting Started 4

Running the Nu Project (Nu.exe) 5

Basic Nu Start-up Code 5

What is NuEdit? 7

# What’s It All About?

The Nu Game Engine is a **Basic**, **Purely-Functional**(ish), **2d Game Engine** written in **F#**.

Let me explain each of those terms –

Basic – Nu is very young, so it has just about no frills. Is there a particle or special effect system? Not yet, I’m afraid. Is there a sprite animation system? Again, not yet. However, there is a tile map system that utilizes Tiled#, and there is a physics system that utilizes Farseer Physics. Rendering, audio, and other IO systems are handled in a cross-platform way with SDL2 / SDL2#. In addition to that, there is an asset management system to make sure your game can run on memory-constrained devices such as the iPhone. On top of all that, there is a built-in game editor called NuEdit! So while there are plenty of missing features, you can see they might be worth building for yourself!

Purely-Functional(ish) – Nu is built on immutable types, and unlike with other game engines, data transformations and state transitions are implemented with copying rather than mutation.

Don’t mistake Nu for being slow, however. Notice I said Purely-Functional-ish. The ‘ish’ means that there are some imperative operations going on in Nu, almost entirely behind the scenes. For example, the Farseer physics system is written in an imperative style in C#, and some parts of Nu are optimized with imperative code as well. Fortunately, nearly all of this will be transparent to you as the user. When writing Nu code, feel absolutely safe, if not empowered, to write in the pure-functional style.

2d Game Engine – Nu is not a code library. It is a game software framework, and thus sets up a specific way of approaching and thinking about the design of 2d games. Of course, Nu is intended to be a broadly generic toolkit for 2d game development, but there are some design choices that will equally constrain and help you. Figure out how to leverage Nu’s design for your game. If it’s a complete mismatch, it might be time to consider using something else.

F# - So why F#? First, and foremost, its cross-platformedness. Theoretically, Nu should run fine on Mono for systems such as Android, iOS, OSX, \*nixes. It definitely runs on .NET for Windows. Note my weasel-word “theoretically”; Nu is still in such an early stage that it has yet to be configured, deployed, or tested on Mono. Nonetheless, since Nu only takes dependencies on cross-platform libraries, there should be no reason why it can’t with a little bit of appropriate nudging.

But more on why F#. F# is probably the best mainstream language available for writing a cross-platform functional game engine. Unlike Clojure, the static type system makes the code easier to reason about and dare I say more efficient. Unlike JVM languages, F# allows us to code and debug with Visual Studio. Finally, I speculate that game developers have more familiarity with the .NET ecosystem than the JVM, so that leverage is right there.

# Getting Started

Nu currently does not have a binary distribution. Instead it has a github repository at <https://github.com/bryanedds/OmniBlade>. I’m going to assume you know (or can quickly figure out) how to pull down the repository on your own. Please take note of the license when pulling down the repository.

The first thing you might notice about the repository is that it contains more than just the Nu Game Engine. It also includes the source for the Aml programming language, the Prime F# code library, and some other loosely related stuff. Both Aml and Prime are required to build the Nu solution, and the rest of the stuff is safely ignored, so feel free to pull it all down.

To open the Nu solution, first make sure to have Visual Studio 2012 (or maybe 2013 – not tested!) installed. Then navigate to the ./Nu/Nu folder and open the Nu.sln file. Attempt to build the whole solution. If there is a problem with building it, try to figure it out, and failing that, ask me questions via [bryanedds@gmail.com](mailto:bryanedds@gmail.com).

Once you have built the solution, try running the game I’m currently making by setting the Nu project as the startup project and running.

# Running the Nu Project (Nu.exe)

When the app is run from Visual Studio, you notice a window popping up, filled with a nice white color. By default, Nu does nothing but clear the frame buffer with white. There is no interactivity in this program, as the engine is not yet being told to do anything.



Though this is not an interesting program, a look at the code behind it should be enlightening.

## Basic Nu Start-up Code

Here’s the main code presented with comments -

namespace Nu

open SDL2

open Nu

module Program =

let [<EntryPoint>] main \_ =

// this initializes all the .Net TypeConverters that Nu uses for serialization. This should

// always be the first line in your Nu program.

World.initTypeConverters ()

// this specifies the manner in which Nu is viewed. With this configuration, a new window

// is created with a title of "Nu Game Engine" and is placed at (32, 32) pixels from the

// top left of the screen.

let sdlViewConfig =

NewWindow

{ WindowTitle = "Nu Game Engine"

WindowX = 32

WindowY = 32

WindowFlags = SDL.SDL\_WindowFlags.SDL\_WINDOW\_SHOWN }

// this specifices the manner in which Nu's rendering takes place. With this configuration,

// rendering in Nu is hardware accelerated and synchronized with the system's vertical

// trace, making for smoother rendering.

let sdlRenderFlags =

enum<SDL.SDL\_RendererFlags>

(int SDL.SDL\_RendererFlags.SDL\_RENDERER\_ACCELERATED |||

int SDL.SDL\_RendererFlags.SDL\_RENDERER\_PRESENTVSYNC)

// this makes a configuration record with the specifications we set out above.

let sdlConfig =

Sdl.makeSdlConfig

sdlViewConfig

Voords.VirtualResolutionX

Voords.VirtualResolutionY

sdlRenderFlags

1024

// this is a callback that attempts to create 'the world' in a functional programming

// sense. In Nu, the world is represented as a complex record type name World.

let tryCreateWorld sdlDeps =

// Game dispatchers specify some unique, high-level behavior and data for your game.

// Since this particular program has no unique behavior, the vanilla base class

// GameDispatcher is used.

let gameDispatcher = GameDispatcher () :> obj

// here is an attempt to create Nu's world using SDL dependencies that will be created

// from the invoking function using the SDL configuration we defined above, the

// gameDispatcher immediately above, and a value that could have been used to

// user-defined data to the world had we needed it (we don't, so we pass unit).

World.tryCreateEmptyWorld sdlDeps gameDispatcher ()

// this is a callback that specifies your program's unique behavior when updating the world

// every tick. It's return type is a (bool \* World). The bool value is whether the program

// should continue (true), or exit (false). The World value is the state of the world

// after the callback has transformed it. It is here where we initially see Nu's purely-

// functional(ish) design. Nu's World type is almost entirely immutable, and the only way

// to update it is by making a new copy of an existing one (such as the one we receive in

// the parameter here). Since we need no special in this program, we simply return the

// world the we received.

let updateWorld world =

(true, world)

// after some configuration it is time to run Nu. We're off and running!

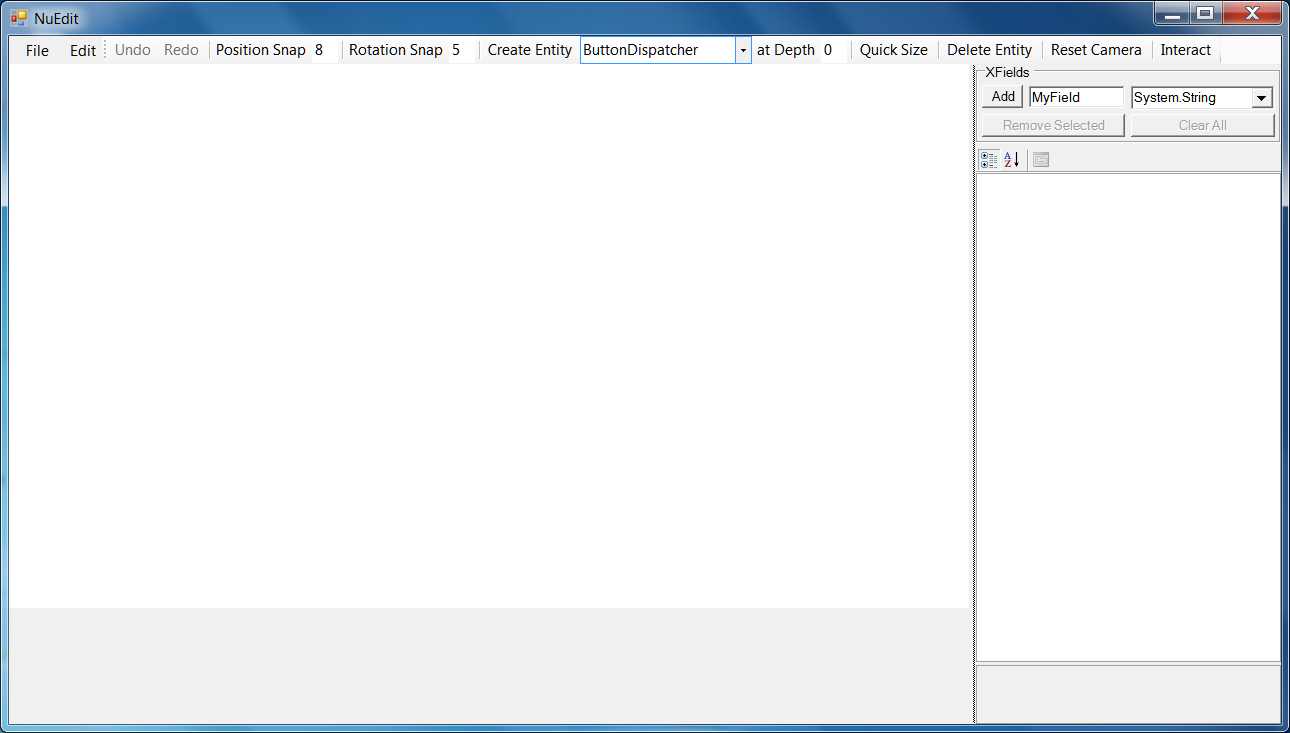
World.run tryCreateWorld updateWorld sdlConfig

Hopefully that was somewhat enlightening. You can find this code in Visual Studio with the Nu.sln solution loaded in the Program.fs file of the Nu project.

When creating a new Nu game project, you can copy and modify this file into your project to use as a template for your program!

# What is NuEdit?

NuEdit is Nu’s fairly usable game editor. Here is a screenshot –



***NOTE:*** *There may still be some stability issues with NuEdit, so save your documents early and often, and for goodness sakes use a source control system!*

Try running Nu’s game editor by setting the NuEdit project as the startup project in Visual Studio and running. Playing with NuEdit will be a reasonable way to further orient you.