A Note on Objects, Classes, and Prototypes

I’m beginning to feel the need for a feature that I recall Unreal licensees clamoring for in the mid-2000s. There is often in game development a need for something in between an Object and a Class. Fresh has now exposed this need.

The need is most clearly seen from the user’s perspective.

If you want to create an object, you can easily do so in the load file (Stage.xml in my case). This allows you to tweak the object’s individual properties and thus construct it.

This approach forces the object to be created immediately, however. It is not useful for constructing a value-tweaked object “after” load time—in response to a user event, for example.

For constructing objects “on the fly,” your best recourse is to use a class. And indeed, if you are also providing new behavior along with tweaked values, a class is the perfect solution.

But what if you want to create an object with tweaked values “on the fly”, yet don’t want it to have new behavior?

This is an extremely common use case. For example, in House of Shadows I show a footstep particle whenever the player taps the screen. The properties for this particle obviously need to be tweaked, which suggests using the XML file (stage.xml). Yet the particle must be created on the fly, not at load time, so the XML file is not helpful. Therefore I’m left with two other options: either create the emitter in the usual C++ way (*new*) and set its properties using member functions, or create a new class with the properly-set values and then *new* an instance of that class.

Neither of these solutions is pleasant. Using member functions is laborious and inflexible. Creating a new class is heavy-handed and likewise rigid. I would really like to create the object dynamically, using an existing class (ParticleEmitter), and yet tweak its properties using XML that is defined in a data file (perhaps stage.xml).

Thus, Objects are created too early. Classes are too rigid and heavy-handed. We need something in between.

I suppose the use case for this system might look something like the Asset system. Indeed, we could very nearly use the Asset system for this. The problem is that ParticleEmitters are not Assets; nor are many other classes for which this technique would be useful. So we could make the Asset system more general. Or we could create an underlying layer of loading and creation functionality that the Asset system inherits. Or we could create a distinct set of functionality, perhaps a “Prototype” or “Preset” system:

ParticleEmitter::Ptr pFootprints = Presets::instance().createPreset( “footprint” );

(…ignoring an important dynamic\_cast<> there).

Or we could have a “deferred” setting in manifests (such as stage.xml) that means, “Don’t create this object right way, but remember it for later (multiple) creation.” That amounts to much the same as the Presets system.

The need for this “in between” object creation approach is not acute yet, but I should introduce a solution eventually. It seems a very common need, and without it I’ll waste time working around the limitation.