**§ 8 – Scripting System**

Making a game is a lot of work, and so is compiling the engine. Tightly-woven C++ code, as efficient as it may be, is far too difficult to program and the compilation times makes quick prototyping basically impossible.

SFEngine contains an embedded scripting engine, [Chaiscript](http://chaiscript.com/). Chaiscript is designed to be embedded, and the efficiency allows thousands of script calls per frame.

There are some basic methods that can be invoked inside your scripts. Some are object methods and some are just global methods.

**§ 8.1.1 – Global Methods – Render Settings**

**SetAALevel(int level)**

Change the Anti-Aliasing level currently being used to render objects. *This change can be rejected if your device’s hardware does not support the level requested*

**SetBrightness(int level)**

Change the brightness level currently being used when rendering objects.

**SetContrast(int level)**

Change the contrast level currently being used when rendering objects.

**SetFramerateLimit(int limit)**

Change the framerate limit (max FPS)

**SetGamma(int level)**

Change the gamme setting currently being used when rendering objects

**§ 8.1.2 – Global Methods – Level Methods (Pre-Play)**

**LoadLevel(string configFilePath)**

Signal the engine to being loading a level. This will not cause a transition to happen, but will cause loading to begin in the background. *This can cause performance degradation, especially if very large assets are being loaded*.

**SwitchToLevel(string levelName)**

Signal the engine to switch to a level. This should only be called after calling LoadLevel. This will cause the old level to be unloaded and game state moved to the new level.

**§ 8.1.3 – Global Methods – Level Methods (In-Play)**

**SpawnActor(string ID, string class, vec2f Position)**

Signal to the level to spawn an actor at the position provided. An ID must be provided to uniquely identify the actor object and the class that you wish to spawn must be provided.

**SpawnProjectile(string class, vec2f Position, vec2f Velocity, vec2f Acceleration)**

Signal to the level to spawn a projectile at the position provided. A velocity and acceleration must be provided in order to spawn correctly. The class name must be passed as the first parameter so that the level can determine what type of projectile to spawn.

**SpawnStaticObject(string ID, string class, vec2f Position)**

Signal to the level to spawn a static object at the position provided. This will add the object to the QuadTree used in the level.

**SpawnLight(string ID, float attenuation, float intensity, vec2f position, bool CastShadows)**

Signal to the level to spawn a light at the location provided. Passing “true” for **CastShadows** will cause the light to cast shadows, but this will incur runtime cost.

**§ 8.1.4** **– Object-Agnostic Methods**

Some script methods can be called without needing to be associate with an object (ie no "this" needed in the script".

**§ 8.1.5 – Object-Specific Methods**

Some script methods can only be called when associated with an object, or a game object must be given as a parameter.

**Kill(shared\_ptr<GenericActor> ptrToActor)**

"Kills" the actor. This will invoke the object's "**Killed"** script method, which will inform it that you have killed it. It cannot reject this, but it can cause a respawn.

**Respawn(shared\_ptr<GenericActor> ptrToActor)**

"Respawns" the actor. This will invoke the object's "restore" method, restoring it to its state when it first spawned into the level. This can be rejected by an object if respawning is not possible (or a respawn method is not written).

**AddComponent**

There are multiple methods that can be used to add components to objects. Generally, this is done to attach physics components to actors.