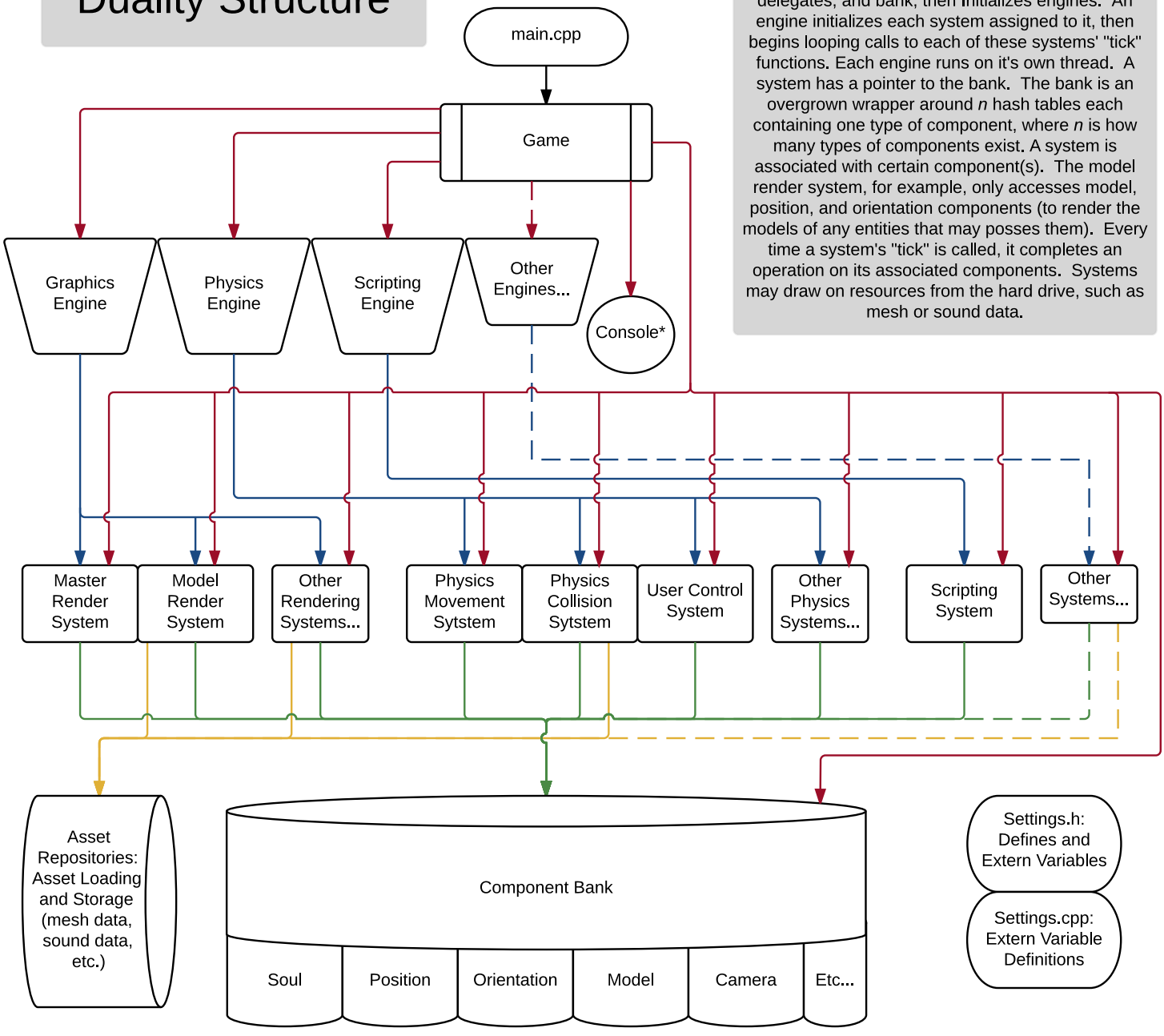
Duality Structure



Game instantiates engines, systems, console, delegates, and bank, then Initializes engines. An engine initializes each system assigned to it, then begins looping calls to each of these systems' "tick" functions. Each engine runs on it's own thread. A system has a pointer to the bank. The bank is an overgrown wrapper around *n* hash tables each containing one type of component, where *n* is how many types of components exist. A system is associated with certain component(s). The model render system, for example, only accesses model, position, and orientation components (to render the models of any entities that may posses them). Every time a system's "tick" is called, it completes an operation on its associated components. Systems may draw on resources from the hard drive, such as mesh or sound data.

Game contains all engines, threads, systems, and the bank. Inter-System communication, while preferably avoided, may sometimes be necessary or desirable. This is accomplished through some home-brewed c++ delegates that are stored in Game. Pointers to these delegates may be given to systems. For example, take the user control system, where all keyboard input is received. If the user is typing in the in-game console, all commands must be submitted to the scripting system. Game holds a delegate to the scripting system's "submit command" routine, and the user control system is allowed access to this delegate, among others.

For now, settings such as screen resolution and other "access-once" variables are extern (global) variables. They are declared in Settings.h, which is #included by any code that needs these settings. As long as the settings aren't used except for initialization of systems, this seems safe to me. This allows for them to be read in from a config file at startup.

* Console is where a record of all textual input and output to and from Duality is stored, along with a few handy interfaces that allow submissions to this log and retrievals of part or all of the log for saving to the disk or rendering to the screen. For this last purpose, a pointer to Console is given to the Console Rendering System.