Application Flow

Below is the current flow of the application in function calls. The order is important for certain calls such as OpenGL functions, any object which relies on a certain system.

The main function is inside entryPoint.h which will create a new Application which is an extern function defined in the sandbox’s Game Application class.

* **entryPoint.h::main**
* **Application Constructor (Engine)**
  + Initialises the System Manager with a value for the maximum number of systems
  + Individual Systems are then added. Once added, their start function is called. The systems are added in the following order:
    - Logger
    - Randomiser
    - TimerSystem
    - WindowAPISystem
      * GLFW is initialised
    - WindowManager
      * Main window is created with values stored in the engine’s config file
    - Physics
    - EventManager
    - ResourceManager
      * Certain resource configuration values are loaded from the config json file
      * All resources are then loaded into the manager
    - FontManager
      * Freetype initialised
      * First & Last Glyph set
      * Default fonts stored in the engine’s config file are loaded into the manager
    - SceneManager
    - RenderSystem
      * TextureUnitManager created
      * 2D and 3D renderers initialised and sent the texture unit manager
    - Inspector
      * Inspector is initialised
  + FPS Timer and TotalTime timers are created

At this point, all the systems have been started and their start functions have all been completed. A main window has been created with all its properties, all resources have been loaded into memory, fonts loaded, and the renderers have been initialised.

* **GameApplication Constructor (Sandbox)**
  + SceneLoader loads new scenes
    - New scene is created by the Scene Manager, if it is the first scene, it will be set as the active scene
    - The scene’s core data will be loaded
      * Layers
      * Settings (i.e Physics)
      * Render Passes
    - Entities are then loaded into the scene
      * Entity is added to the scene
      * Entity’s base data is set
      * Components are attached using the json data

At this point, all scenes that are needed on launch have been created and the entities loaded from JSON with all components attached. The main camera for the scene will have been set when that camera was loaded.

* **Application::run (Engine)**
  + The scene to be processed this iteration of the game loop is retrieved before any processing is done (Current active scene)
  + The timestep and the total time of the application is retrieved and stored locally
  + Event Manager Update
    - Update the event data times
    - Entities Preupdate call
    - InputPoller checks for all keys currently pressed and calls the appropriate input functions
    - InputPoller checks for all mouse buttons currently pressed and calls the appropriate input functions
    - The active scene is updated
      * Any entity scheduled to be deleted, are deleted and remove from entity list, issuing an entity list change
      * All entity components receive their update
      * LayerManager is updated
      * Physics World Updated
    - Update all registered windows
      * Swap back buffer
      * Poll Events (Window event call backs occur here)
    - Entities Postupdate call

All entities are updated, the active scene is updated, all windows are updated.

* + RenderSystem::onRender
    - Entity list is retrieved if the entity list is updated
    - Get all render passes of the scene
    - Go through each render pass and render
  + Inspector Inspects
    - If the scene has the inspector enabled
  + Deregister scheduled scenes
    - Destroys all scenes which are set to be destroyed
  + Deregister Scheduled Windows
    - Destroys all windows which are set to be destroyed
  + Check the exit conditions of the application
* **GameApplication Destructor (Sandbox)**
  + Currently empty
* **Application Destructor (Engine)**
  + Destroy the system manager
    - Which destroys all systems in reverse order and their resources