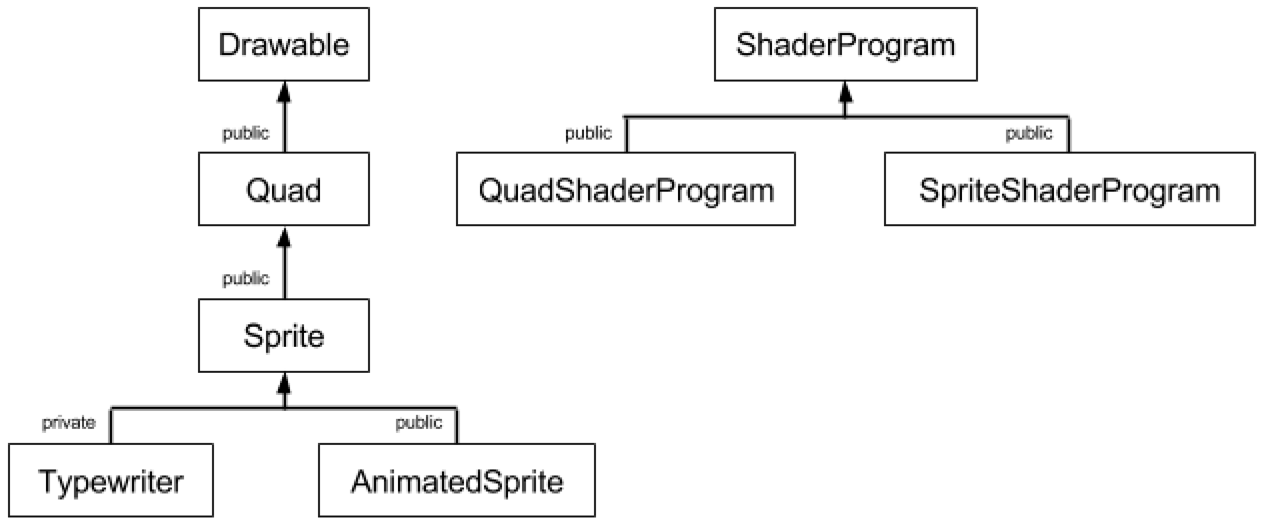
1. Classes

Most classes in this library are in the *MyFirstEngine* namespace. Those that aren't are in anonymous, cpp-local namespaces due to being hidden helper classes not exposed to library users. There is one other namespace, *HTML*, that contains ANSI extended ASCII character constants for HTML special characters, such as *nbsp* (with a value of '\xA0', the character for a non-breaking space) and a function, *Map()*, that returns a const reference to a *CharacterMap* object mapping strings like "nbsp" to the corresponding extended ASCII characters.

Most classes in this library either don't derive from anything or only derive from utility bases like *Singleton* or *NotCopyable*. There are a few that do inherit from other notable classes:



There are 19 classes in the *MyFirstEngine* namespace available for use (or derivation from) by users:

- *GameWindow:* A class storing the dimensions and title of a window, capable of opening or closing an actual window (and associated OpenGL context) with said dimensions and title.
- *Camera:* A singleton class that controls where other objects are seen from and whether to use parallel or perspective projection (and, if using perspective projection, what focal length to use) when rendering them to the screen, along with the size of the screen in world coordinates (for example, using the pixel dimensions of the window in order to make world coordinates and pixel coordinates match).
- *Mouse:* A singleton that tracks when mouse buttons were last pressed or released and provides static functions for retrieving the location of the cursor relative to the center of a given *GameWindow*, which *GameWindow* (if any) the cursor is currently in, whether

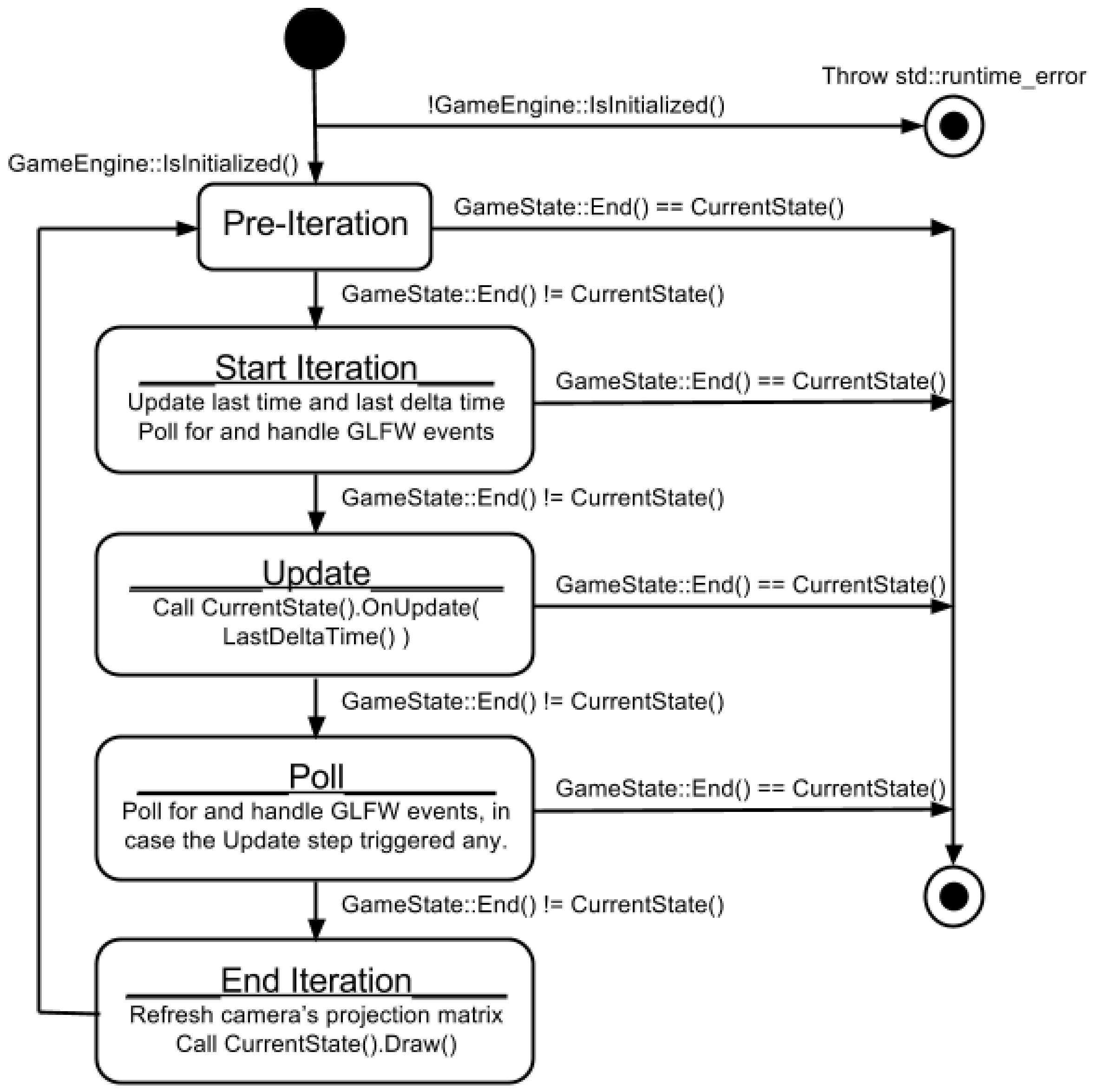
or not a given mouse button is currently pressed, and how long a given mouse button has been in its current state.

- *Keyboard:* A singleton that tracks when keyboard keys were last pressed or released and provides static functions for checking whether a single key, either of the two shift/control/alt/super keys, or any key at all is currently pressed, and how long said key or keys has/have been in that state.
- *GameState:* A base class for game states, with virtual functions *Draw()*, *OnUpdate(double a_dDeltaTime)*, *OnCloseWindow(GameWIndow& a_roWindow)*, *OnEnter()*, *OnSuspend()*, *OnResume()*, and *OnExit()* for redefining in derived classes to (when serving as the current state) draw objects to the screen or carry out actions on each frame update, whenever a window is closed, after being added to the top of the stack of states, before another state is pushed on top of it on the stack, after the state above it is popped off the stack, or before being popped off the stack itself, respectively. There are also functions for making the state replace or be pushed on top of the current state in the stack managed by *GameEngine*. A special state returned by the static function *End()* indicates that the game should stop running.
- *GameEngine:* A singleton with functions for managing stacks of modelview matrices, projection matrices, and game states, for initializing or terminating the engine as a whole and all the other objects that need to be initialized or terminated with it, and for running the engine - that is, for looping through updating the last time and elapsed time, polling for events, calling the current state's *OnUpdate* function, polling for events again, refreshing the *Camera* projection matrix, and drawing the current state to the screen, as long as the current state is something other than *GameState::End()*.
- *Texture:* A class containing the name of a texture file and functions for loading, binding, or unloading an OpenGL texture from that file.
- *Shader:* A class for loading and compiling OpenGL shader objects from source files. If a source file has already been loaded and compiled, the object ID is reused instead of compiling a duplicate.
- *ShaderProgram:* A class for linking shaders together into an OpenGL shader program object. Virtual functions *SetupData()*, *UseData()*, and *DestroyData()* are redefined by child classes in order to load, bind, or destroy vertex data buffers or to save the locations of uniform variables.
- *QuadShaderProgram:* Publically derived from *ShaderProgram*. A singleton class that loads a shader program from files resources/shaders/QuadVertex.glsl and resources/shaders/QuadFragment.glsl that draws a 1x1 rectangle, centered at the origin, of a solid color in model space (that is, a rectangle that gets scaled up, rotated, and moved according to the model matrix at the top of the stack in *GameEngine*).
- *SpriteShaderProgram:* Publically derived from *ShaderProgram*, recycles buffers used by *QuadShaderProgram*. A singleton class that loads a shader program from files resources/shaders/SpriteVertex.glsl and resources/shaders/SpriteFragment.glsl that draws a 1x1 rectangle, centered at the origin, with a slice of a given texture with a given starting UV coordinate and UV coordinate slice size tinted by a given color, all in model space.

- *Drawable:* Base class for all things that can be drawn to the screen. Contains variables (with associated Get/Set functions) for scale, position, rotation, color, and whether or not the object is visible. Stores a model matrix that gets recalculated before drawing the object if any of its properties has changed. The const function *Draw()* checks if the object is visible and, if it is, pushes the object's model matrix onto the stack maintained by *GameEngine*, calls the virtual *DrawComponents()* function that gets redefined in derived classes in order to actually draw whatever the object is, then pops the model matrix back off the stack.
- *Quad:* Publicly derived from *Drawable*, draws a solid rectangle of a single color using the *QuadShaderProgram*.
- *Frame:* A struct containing pixel coordinates and sizes for describing the size and center of a drawable object, a slice of texture, and where to draw that slice of texture relative to the boundaries of the drawable object.
- *Sprite:* Publically derived from *Quad*, draws a textured rectangle, with the texture colorized by the object color attribute (default is white, which leaves the texture as is) using the *SpriteShaderProgram*. The texture can include transparency. Stores a pointer to a *Texture* object, a pointer to a list of *Frame* structs, and an index indicating which *Frame* object in said list to use in determining what parts of the texture to render (if the list pointer is null or points to an empty list, the entire texture is used).
- *AnimatedSprite:* Publically derived from *Sprite*, with additional attributes for framerate (zero meaning advance a frame every update, no matter how much time has passed between frames) and number of loops (zero meaning loop forever), along with methods for starting or pausing animation or moving to a specific point in the animation (either by frame or, if framerate is non-zero, time). The *Update(double a_dDeltaTime)* function must be called in order to automatically advance frames when appropriate, probably in a *GameState* object's *OnUpdate* function.
- *CharacterMap:* Maps strings naming symbols (such as "nbsp" or "yen" or whatever custom symbol names the user cares to define) to characters 0 - 255.
- *Font:* Class containing a pointer to a *Texture*, a list of *Frame* objects indicating where (if anywhere) in the texture each character 0-255 is shown, the size of a single em and of the distance between single-spaced lines in pixels, a character to substitute for characters or symbols not depicted in the font, and a *CharacterMap* for defining symbols not mapped by *HTML::Map()* (or for remapping symbols in that map to 0 so their characters can be used for something else).
- *Typewriter:* Privately derived from *Sprite*, a singleton class used for drawing text and symbols to the screen (as long as a font is set). Contains static functions for changing the font, font size (the display size of a single em), font color, line spacing (as a multiple of single lines), character spacing (as a number of ems in addition to the normal, frame-based kerning), tab width, and start position, along with static functions for drawing strings ('\n' and '\t' result in moving the position of the next character to either one line down and aligned with the start position or to the next multiple of tab width away from the horizontal start position, respectively), drawing special symbols, drawing tabs or starting new lines, or returning to the start position.

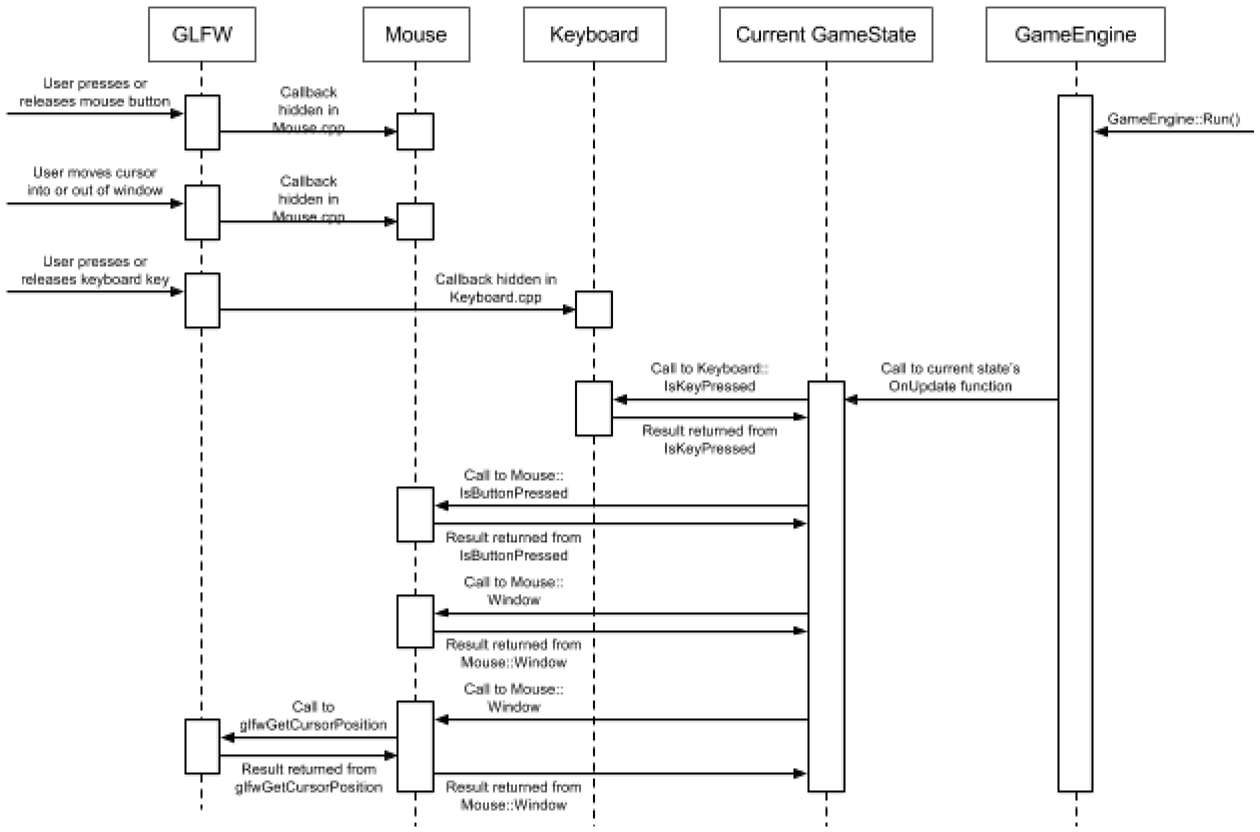
2. State loop

The *GameEngine::Run()* loop is fairly straightforward:


Throw std::runtime_error
!GameEngine::IsInitialized()
GameEngine::IsInitialized()
Pre-Iteration
GameState::End() == CurrentState()
GameState::End() != CurrentState()
Start Iteration
Update last time and last delta time
Poll for and handle GLFW events
GameState::End() == CurrentState()
GameState::End() != CurrentState()
Update
Call CurrentState().OnUpdate(LastDeltaTime())
GameState::End() == CurrentState()
GameState::End() != CurrentState()
Poll
Poll for and handle GLFW events, in case the Update step triggered any.
GameState::End() == CurrentState()
GameState::End() != CurrentState()
End Iteration
Refresh camera's projection matrix
Call CurrentState().Draw()

3. Input sequences

User input is handled by the *Mouse* and *Keyboard* singletons. Whenever a *GameWindow* is opened, *Mouse* and *Keyboard* methods are called to register cpp-only event handlers for pressing mouse or keyboard buttons or for the cursor entering or leaving the window. These event handlers update data in the *Mouse* and *Keyboard* translation units that user-facing *Mouse* and *Keyboard* methods can then read. For the cursor position, *Mouse* just calls GLFW.


GLFW
Mouse
Keyboard
Current GameState
GameEngine
User presses or releases mouse button
Callback hidden in Mouse.cpp
User moves cursor into or out of window
Callback hidden in Mouse.cpp
User presses or releases keyboard key
Callback hidden in Keyboard.cpp
GameEngine::Run()
Call to Keyboard:: IsKeyPressed
Result returned from IsKeyPressed
Call to current state's OnUpdate function
Call to Mouse:: IsButtonPressed
Result returned from IsButtonPressed
Call to Mouse:: Window
Result returned from Mouse::Window
Call to Mouse:: Window
Call to glfwGetCursorPosition
Result returned from glfwGetCursorPosition
Result returned from Mouse::Window

4. Testing

The EngineDemo project demonstrates most of the important library functionality. It opens a game window containing several moving and rotating quads of different sizes and colors. An animated sprite follows the cursor, animating except when the right mouse button is pressed, dissappearing if the cursor leaves the game window, and becoming semi-transparent if the space key is pressed. Text, including both regular characters and special symbols, is printed on the screen in multiple colors. Finally, the application can be closed either by pressing the escape key or by closing the game window.

5. External libraries

Aside from OpenGL, the MyFirstEngine library relies on SOIL to load image data and GLFW to handle windows and user input.