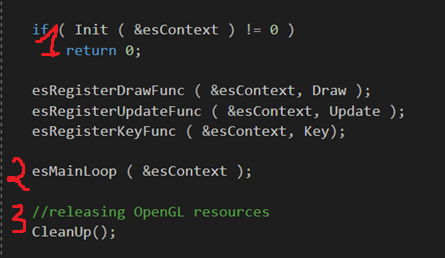
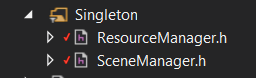
Overview



1. Init() : Initialize the game, loading scene and necessary files
2. esMainLoop() : Game Loop
3. CleanUp() : Clean Game trashes

esMainLoop:

* Key(): Only get called when a key is pressed or released
* Update(): Is called every frames
* Draw(): Is called every frames as well but should be implied as a container for rendering sprites



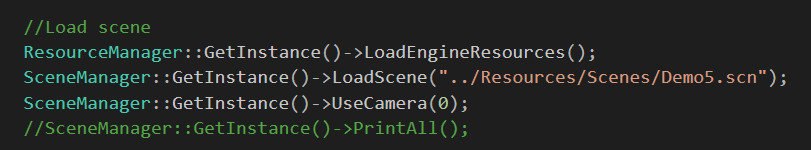
SceneManager: Holds all GameObjects in the current scene.

* Call Update() and Draw() for all GameObjects.

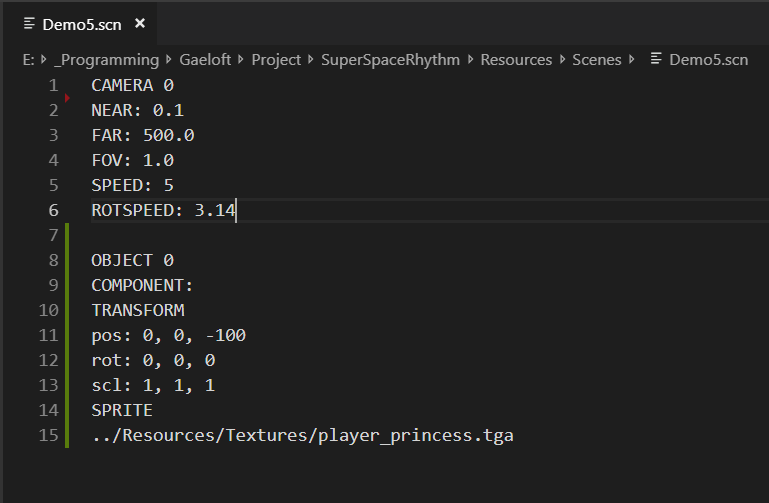
ResourceManager: Holds all loaded resources informations.

Any changes to the game in these singletons are not recommended, logics and renders should have their own GameObject Components.

Loading a scene



When starting the game, Init() will load a default scene for feature testing with SceneManager::GetInstance()->LoadScene(char \* scenefilename);



Inside the scene file, gameObjects and cameras are formatted like above.

The layout of an object is like this:

OBJECT #

COMPONENT:

COMPONENT\_NAME

COMPONENT\_DETAILS

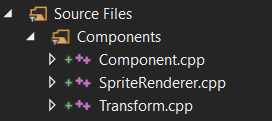
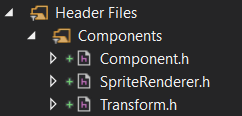
[COMPONENT\_NAME]

[COMPONENT\_DETAILS]

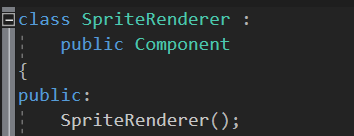
[OBJECT #+1] //Note that currently # has no usage

How to add a component?

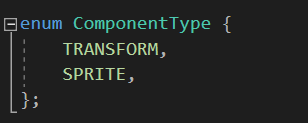
Adding another object is easy, you just write another one in the scene file and it will be registered into the SceneManager. But in order to make a new component, you must make a new class for it in the Component folder.



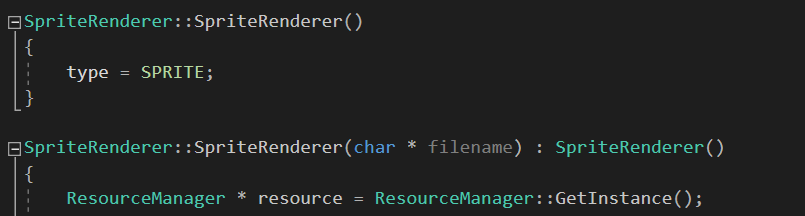
First thing first, make sure your new component inherit the Component class



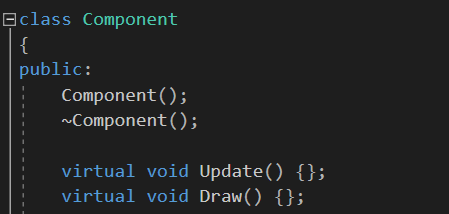
Add the enum for your component in ComponentType enum inside Component.h



Then assign the type for your component in its’ default constructor



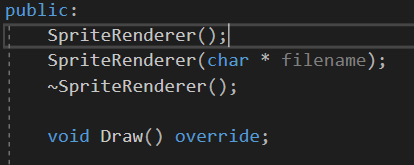
Every components has 2 virtual functions that can be overrided: Update() and Draw()



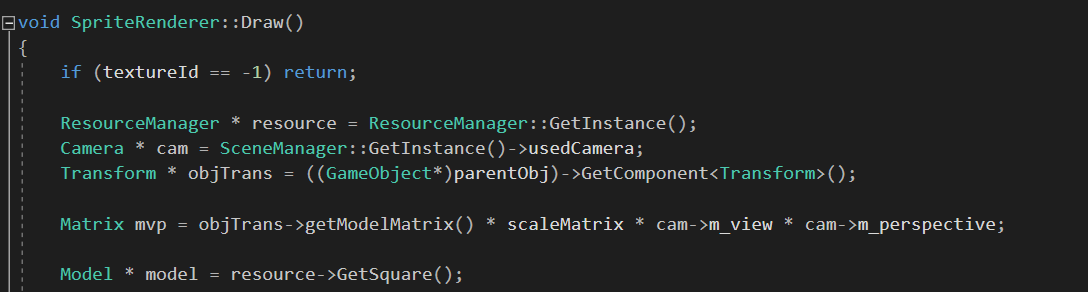
Update() is called every frames by SceneManager::GetInstance()->Update() inside Opengl’s Update()

Draw() is called every frames by SceneManager::GetInstance()->DrawAll() inside Opengl’s Draw()

To override them, simply put an override declaration in your component’s header file



Then write its’ definition in the .cpp file

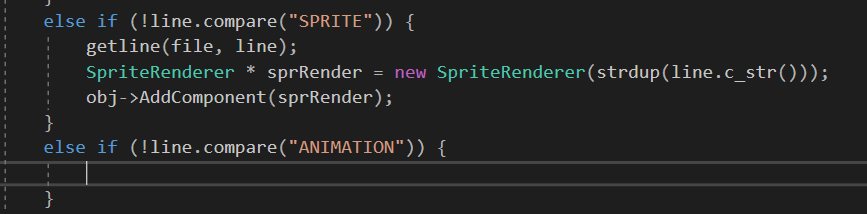


At this point, your component can be added during runtime and should operate like how you expect it to.

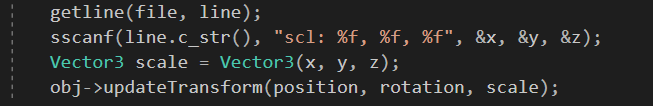
In order for your component to be recognized by the SceneManager::GetInstance()->LoadScene() function, you must append some code to it. Go to the last component in the read block then add an:

else if (!line.compare([YOUR COMPONENT NAME])) {

}

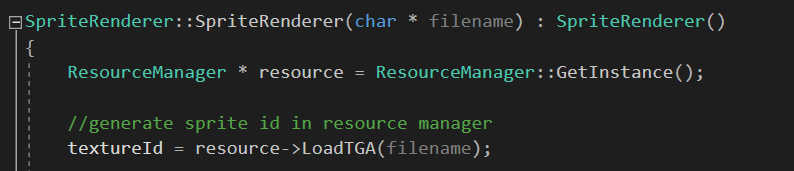


Inside the curly brackets, add codes on how the component informations should be read from file (getline(file, line) for next line, sscanf(line.c\_str(), “[format]”, &variable) to get variables). Take a look at how the currently implemented component are written to get a clearer idea.



Make sure you have generated an object of your component then obj->AddComponent(Component \*); to add it to the gameObject. Transform is an exception since all gameObjects are created with a Transform component.

If you wish to use a custom constructor for your component, be sure to inherit it from the default constructor so that the type for that constructor is set.



You should be able to recognize your component from the scene file now

