Compulsory 1 Adam Aske

Github-link: https://github.com/adamaske/MyGE

I have created an engine with GLFW, GLAD and Lua. I started from a blank C++ project and have written the start of a game engine with Lua scripting available to change the game without recompiling. Lua, GLFW and GLAD are libraries located on my computer and linked with the visual studio solution.

MyGE is the class containing the main program loop. MyGE::Run starts by starting GLFW and creating a RenderWindow. It sets up the systems for the gameplay and connects the systems.

Until the RenderWindow is told to close the game is running. ProcessInput() is the first step of the gane loop.

```
//Main loop
while (!mWindow->ShouldCloseWindow()) {
    double time = glfwGetTime();
    //Process input
    ProcessInput();
    //Change Game

    mSceneManager->GetScene()->OnUpdate();

    //Render
    mWindow->Render();
}
```

GameObject

This is supposed to be superclass of objects which can be in the game world. This class is currently a container for the components, but I will soon change it to just be an ID and the components will refer to their owners ID. This will be a much more robust system than what currently is going on. Every GameObject can have a parent GameObject, therefore it can be initlized with and without a parent.

Adding a component to the GameObject consists of AddComponent, it takes in a component pointer and adds it to mComponents. This will be gone when changing the GameObject to an ID only. The future system will consist of components having a reference to their parent GameObject's ID, when the component is created it gets added to some sort of registry to keep track of then. When the components needs either other components from the Object, for

example a MeshRenderer component may want accsess the MaterialComponent, they then request a component of a type T with their Object's ID as a key.

```
void AddComponent(Component* comp) { mComponents.push_back(_Val: comp); };

template<typename T>
T* GetComponent(GameObject object) {
    for (int i = 0; i < mComponents.size(); i++)
    {

    }
    return nullptr;
}

TransformComponent* GetTransform() { return GetComponent<TransformComponent>(object: this); };
tected:
GameObject* mParent;

std::vector<Component*> mComponents;
```

MeshComponent has the functionality of reading obj files and creating vertices and indexing from it. It currently uses the cube.obj as a mesh, changing the obj file and recompiling changes the object shown in the scene.

Component

The component class is the most important part currently. A component is a piece of functionality we can attach to a GameObject. I will be spending a lot more time making this class robust. Every component shares a few traits, like Init and OnUpdate, these functions are overridden by the sub-components. The components will go from having functionality to just data,

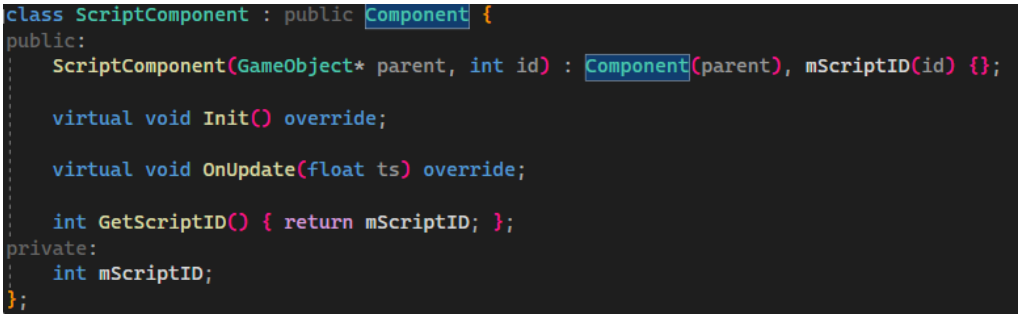
```
virtual void Init() override {
    mMatrix = glm::mat4(1.f);
    mPosition   = glm::mat4(1.f);
    mRotation   = glm::mat4(1.f);

    mScale= glm::mat4(1.f);
    // Legger til rette for simulering
    mVelocity = glm::vec3(1);
    //Kollisjoner
    mSize = glm::vec3(1);
    std::cout << "TransformComponent : Init" << std::endl;
};
```

Lua scripting

I am using the Lua dll. I have not managed to create a good example with Lua yet. The ScriptComponent is meant to let the user attach a script to GameObjects. With this a Object can be changed and iterated on without recompiling,Other usescases for the Lua scripting can be as memory and storing. The ScriptingManager will be incharge of the Scripts the engine has available. The Script-class should only be a container for an ID or Key to the Scripts.

I have not gotten far or advanved with Lua yet, but have created a test of it working. LuaExample.lua is located in src/Scripts. I imagine using Lua in the future for storing prefabs of objects, first I will make a AddComponent function to the Lua script and create the C++ for having it execute the intended behavior. After creating a good Lua implementation I plan on trying to implement PhysX.



```
class ScriptComponent : public Component {
public:
    ScriptComponent(GameObject* parent, int id) : Component(parent), mScriptID(id) {};

    virtual void Init() override;

    virtual void OnUpdate(float ts) override;

    int GetScriptID() { return mScriptID; };
private:
    int mScriptID;
};
```