Tarea6

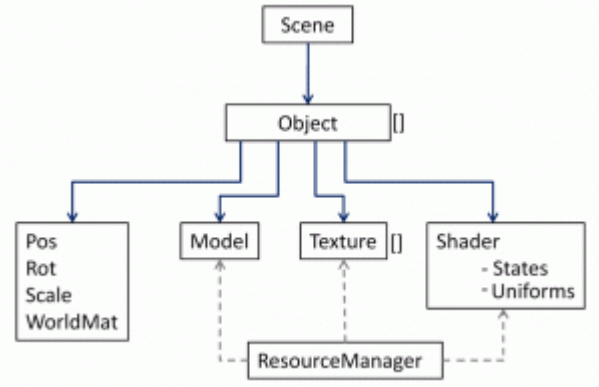
3D Engine

Instrucciones:

- Leer y comprender:

  https://docs.gameloft.org/3d-training/#Engine_structure

- En este Engine intervienen dos actores:
  - Un Resource Manager, el cual se encargara con lo relacionado a recursos: cargarlos / mantenerlos / destruirlos / proveer accesibilidad a ellos.
  - Un Scene Manager, el cual se encargara de mantener los objetos, y aquí se trabajara todo lo referente a objetos.



- Un usuario deberá ser capaz de trabajar con el ejecutable de tu aplicación y los archivos RM.txt donde se describen los recursos y el archivo SM.txt donde se describe la escena.
- Un usuario que no es programador deberá ser capaz de configurar y usar tu aplicación. Todo debe ser configurable atreves de los archivos RM.txt y SM.txt.
- No usar STL (Estándar Template Library); ejemplo: Vector, list, stack, etc. Ya que no es propio utilizarla en video juegos porque afecta el FPS.

Guía:

```
STL se puede sustituir por arrays de elementos, para cada tipo de elemento;
primero deberas leer la cantidad de elementos desde el archivo RM.txt y/o
SM.txt, y despues hacer un array con la cantidad de espacios necesarios usar
el operador new[] (ejemplo: leer y despues  guardar el numero de modelos en
```

nModels, luego reservar el espacio de memoria para un array con esas dimenciones *Model* pModels = new Model[nModels];*, luego usando un ciclo for para cargar los modelos *for(int i =0; i<nModels; i++)*)

Sera necesario agregar algunas clases a tu aplicación:

a) Clase "ResourceManager" (debe ser singleton, en la siguiente dirección puedes ver como escribir una clase singleton https://docs.gameloft.org/wp-content/uploads/2012/04/SingletonSample1.txt)

- Numero de modelos y puntero a la clase "Models".
- Numero de texturas 2D y puntero a la clase "Texture".
- Numero de "cube textures" y un puntero a la clase "Texture" (Solo hay una clase "Texture", pero habra dos funciones Init() una para texturas 2D y otra para texturas cube). Por el momento, no se agregaran las "cube textures" ni sus funciones para trabajar con ellas (en proximos task se hara).
- Numero de "programs" (shaders) y puntero a la clase "Shader".
- Ua función para cargar todo lo del archivo RM.txt y para realizar otras inicializaciones.
- Metodos de acceso como: Model *GetModelById(int ID), donde ID es la Id del recurso en RM.txt. Lo mismo para las texturas y programas(shaders).

b) Clase "SceneManager": (Singleton)

- Numero de objetos y puntero a la clase "Object".
- Numero de luces y puntero a la clase "Lights" (se implementara en un futuro).
- Tu camara (en forma de puntero/objeto).
- Una función de cargado del archivo SM.txt y para hacer otras inicializaciones (ejemplo: llamar el Init() para los objectos, llamar el Init() para la camara).
- Una función de Draw() (Esta funcion se llamara desde NewTrainingFramework.cpp-> Draw()). Su funcion es llamar el Draw() de todos los objetos en escena.
- Se llamara a actualizar desde NewTrainingFramework -> Update, usando algo como SM::Update() y desde SM::Update() se llamara a Camera::Update(deltaTime).
-

c) Clase “Object”:

- Numero de texturas 2D para los objetos y un puntero “Texture**” para accesar a sus handles.
- Numero de texturas cube para los objetos y un puntero “Texture**” para accesar a sus handles.(Solo para futura implementación)
- Puntero a las clases “Model” y “Shader”.
- Matrices de Posicion, rotación, escala y worldMatrix para ese objecto.
- Calcular la WorldMatrix con una funcion y usar bandera que indique cuando es necesario recalcularla.
- Funcion Update(), llamada desde SM::Update()
- Funcion Draw(), llamada desde SM::Draw()
- Funcion Init(), llamada desde SM::Load()
- Numero de “parametros especificos”, que seran necesarios para cuando se agreguen luces mas adelante.
- Usar variables para cada posible parametro, se debe de mantener el codigo facil de mantener, con mucha simpleza, sin complicaciones.

d) En tu clase “Camera”, deberas tener los atributos near, far, fov, speeds, los cuales seran leidos desde el archivo SM.txt, y se inicializara desde Camera::Init() que se manda llamar desde SM::Load().

Ejemplo:

Resource Manager

+Model *ptr[]
+Texture *2Dptr[]
+Texture *Cptr[]
+Shader *ptr[]
+loadRM();
+getModelById(int id)
+getTextureById(int id)
+getShaderById(int id)

Object

+Matrix WorldMatrix
+Model *ptr
+Texture **2Dptr
+Texture **Cptr
+Shader *ptr
+int ID
+int model_id
+int shader_id
+int text2DId[]
+int textCubeId[]
+Vector3 pos
+Vector3 rotation
+Vector3 scale
+int numText2D
+int numTextCube
+Init()
+update()
+draw()
+calc_WM()

Scene Manager

+Object *ptr[]
+Camera *ptr
+loadSM()
+initObjects();
+void update(deltaTime, keypressed);
+void draw();

Ejemplo de inicialización de tus objetos:

```
void Object::init()
 {
        if(object_info.num_textures>0)
                TextureObject = new Texture*[object_info.num_textures];
        for(int i= 0; i< object_info.num_textures; i++)
        {
                TextureObject[i] = ResourceManager::GetInstance()-
>GetTexturebyID(object_info.textures_ids[i]);
        }
…
```