ETGG3801	Lab5: **Game Object Manager (GOM)**	Points: **89**
Assigned: **10/24/2017**	Due: **11/28/2017 @ 8:59am SHARP!!!**	

Tasks:

1. Changes to GameObject class
   a. (**4 points**) Create an **update** method in Component that takes a float delta-time value. Give it an empty body. Also make a similar method in GameObject that calls the update method of each component[1].
   b. (**4 points**) Create a **setVisible** (or maybe setEnabled?) method in Component. Override this in MeshComponent (and set the entity to not-visible). Create a similar method in GameObject that calls the Component methods for all attached Components.
   c. (**8 points**) Add a new **setParent** method in GameObject (pass a pointer to a GameObject).
      i. Make sure to properly detach the scene node before doing this.
      ii. Give the user the option to preserve the world-space "look" of this object (position, orientation, scale).
   d. (**4 points**) Add some "relative" transform methods: rotateWorld, rotateLocal, translateWorld, translateLocal, and scale (this one is always relative to the local axes).
      i. I'd like the rotate methods to take a degree (float) and axis (Ogre::Vector3) – build the quaternion internally.
      ii. You might want to overload these methods to take Ogre objects as well – sometimes that's more convenient.
   e. (**4 points**) Add some getters too: getOrientation, getPosition, getScale (make one version world-relative and another parent-relative)
      i. SceneNode::_getDerivedXYZ is for world.
2. Create a **GameObjectManager** class
   a. (**5 points**) Make it a Singleton. Don't forget the convenience macro and template-specialization of the static member.
   b. (**3 points**) Create and destroy the Game object manager in the application class.
   c. (**6 points**) Ensure only the GOM can create instances of GO's (a factory)
      i. Make the GO constructor and destructor protected
      ii. Make GOM a friend class.
      iii. Add a **createGameObject** method.
   d. (**10 points**) Have a fairly efficient way for the user to group GO's into named groups
      i. The GO's in each group should be destroyable / hideable as a group (have a method to do this). You might also have to modify the GameObject (and Component / MeshComponent?) to support this.
      ii. (**maybe bonus points**) Allow more than one level in this hierarchy.
   e. (there might be bonus points for **"fancy" features or elegant design**)
3. Create a simple "game" in our application class[2]
   a. (**1 point**) Misc: Enable shadows (insert this in your startup routine – it'll help show depth):

```
mSceneManager->setShadowTechnique(Ogre::SHADOWTYPE_STENCIL_MODULATIVE);
mSceneManager->setAmbientLight(Ogre::ColourValue);
```

[1] Why? In later labs, we'll add more functionality to our MeshComponent (e.g. animation). Other components must be updated every frame. But…if a component has no need for this, they simply don't re-define the update method.

[2] We're going to make a super-simple "game" in our application class. Later in the course (and 3802), we'll replace this C++-style of game-creation with scripting support.

b. (**20 points**) Scene setup (see the demo: get it as close as humanly possible)
c. (**20 points**) Run-time changes:
    i. 'P' to change the parent of the spinning ogre head, preserving world-space "look"
    ii. 'H' to hide the bullets (use our GOM method)
    iii. fire a bullet every so often. Give each a random time to live and speed.
d. **IMPORTANT: NO TRIG!!!** (everything should be done through the scene hierarchy)

4. Here's a demo of my running lab5: **https://youtu.be/e3iAjhthGrE**
5. (**? points**) We'll discuss this in class – maybe give the option of making some more components (Camera and Light?) Or…do we want to do that in a dedicated lab?
6. Just turn in your source and include files on blackboard (**-5 for forgetting files…**)