ETGG3802	Lab8: Input Manager	Points: 100

There are more points than normal on this lab. So just do some combination of tasks that will get you to 100 (or beyond).

1. Get a working copy of Lab7
2. (**20 points**) Show a "debug stats" panel in the main window that displays some…debug stats (frame-rate, # triangles, etc.). The main ogre window has functions to get these. You might be able to generalize the stuff we did in the log manager (or not). Be able to toggle this (and maybe the log manager visual log on / off with a back-tick (tilde) key-press
3. (**10 points**) Make a new InputManager class.
   a. A few universal designs I want you to add:
      i. Make it a Singleton (and all that this entails)
   b. You have a implementation choice to make here:
      i. Use OgreBites::InputListener
         1. Our Application currently does this (that's why keyPressed and keyReleased are called by Ogre)
         2. InputListener doesn't do gamepad stuff if you're going for the bonus, but it probably does enough to do the basic lab
         3. You might want to make the InputManager and InputListener [and possibly *not* make the Application]. Don't foget to ues addInputListener.
      ii. Don't: just use raw SDL
         1. Incorporate the SDL dependencies (from the web)
         2. Remove the call to OgreBites::ApplicationContext::framestarted at the start of Application's frameStarted (this is what does input processing, including consuming events)
4. (**10 points**) Once Input handling is moved to the InputManager, add a few simple tests. I did this by looking for a few special keys:
   a. Escape: call a quit method in Application (that just calls mRoot->queueEndRendering)
   b. ~: toggle visibility of the visual log and stats (part 1).
5. (**15 points**) Have the InputManager parse a simple xml file (bindings.xml, similar to mine on blackboard). At least pay attention to the key (action and axis) bindings.
6. (**20 points**) Have our GameObjectManager, as it parses a scene file, look for a special property tag that indicates a script name. Using Lab7 functionality, make this new game object "script-aware" and associate that class with it. In addition, make a top-level function called "load_scene" that will call GameObjectManager's load_scene functionality.
   a. Hint: you could probably use your create_game_object function <u>in C++</u> to make the C++ game object, its twin, and everything!

Items 7 and 8 were written with the intention that you complete item#6. If you can't get 6 working, you could come up with some alternative test to make sure they work (put a note in your submission so I know where to look)

7. (**20 points**) Device-polling support: write a C++ (and python) binding to get the state of input:
   a. Get_axis(name) -> a float [-1.0…+1.0]
   b. Get_action(name) -> bool [true if that action is pressed right now]
   c. Is_valid_axis(name) -> bool [true if that axis exists]
   d. Is_valid_action(name) -> bool [true if that action exists]
8. (**25 points**) Callback support (event-handling support)
   a. Have some kind of implementation of the Observer pattern to register select GameObjects with the InputManager. When an action or axis event happens, call a (python) method called "handle_input" that notifies the listener of that change.1
   b. Have python and C++ functions to register / deregister a listener.

   9. (**20 points**) Gamepad support (in addition to keyboard)