Python Actors

Adding new nodes

This documentation is based on the Gazebosc github: https://github.com/hku-ect/gazebosc

Easiest method of adding a new node is using Python. You'll need to have a Gazebosc build with Python.

- In Gazebosc create a new Python actor: (right mouse click - Python)
- Click on the edit icon, a text editor will appear
- Paste the following text in the texteditor and click save

```
class MyActor(object):
    def handleSocket(self, addr, msg, type, name, uuid, *args, **kwargs):
        print("received OSC message {} {}".format(addr, msg))
        return ("/MyActorMsg", [ "hello", "world", 42] )
```

This is just the most basic actor which responds to incoming messages. A template you can use for a full feature actor is as follows:

```
class actor(object):
    def __init__(self, *args, **kwargs):
        self.timeout = 1000         # Use this timeout value for when you need recurring handleTimer events
                                    # Set to -1 to wait infinite (default)

    def handleApi(self, command, *args, **kwargs):
        print("The API command is {} and its arguments is {}".format(command, args))
        return None

    def handleSocket(self, address, data, *args, **kwargs):
        print("The osc address is {} and its data is {}".format(address, data))
        return ("/myreturnaddress", ["hello", 3, 2, 1])

    def handleTimer(self, *args, **kwargs):
        # This is a timed event, use it as you need
        print("My timed event with type: {}, name: {}, uuid: {}".format(args[0], args[1], args[2]))
        return ("/mytimedreturn", ["hello", 1, 2, 3])

    def handleCustomSocket(self, *args, **kwargs):
        # We'll explain this in the future
```

```
        return ("/myreturnaddress", ["hello", "world"])

    def handleStop(self, *args, **kwargs):
        # We are shutting down
        print("Bye bye from {}".format(args[1]))
```

Save this file as actor.py as the filename needs to equal the class name!

Node Lifetime

Once a node has been created, it goes through the following steps:

- **Construction**
  - if performed from loading a file, also passes and Deserializes data
- **CreateActor** (this is called after instantiation to preserve polymorphic response)
- **Threaded Actor events**
  - Init: actor has been created, and can be used to do threaded initializations (see OSCListener example)
  - Message: actor has received a message
  - Callback: actor has received a timeout (timed event, probably scheduled by calling the SetRate function)
  - Stop: actor has been stopped and threaded resources can be cleaned up (see OSCListener example)
- **Destruction**

Construction & Destructions

During these phases, you can prepare and clean up resources used by the class. Examples include UI char buffers for text or values (see PulseNode).

CreateActor

This GNode function can be overriden to perform main-thread operations once the actor has been created. Primary use-case at this time is calling the *SetRate* function (an API-call, which must be called from the main thread) to tell the node to send timeout events at a set rate (x times per second).

Threaded Actor events

Throughout the lifetime of the actor, the GNode class will receive events, and pass these along to virtual functions. Override these functions to perform custom behaviours (see above description for which events there are). Important to note is that this code runs on the thread, and you should not access or chance main-thread data (such as UI variables). For such cases, we are still designing *report functionality* (copied thread data that you can then use to update UI, for instance).

Destruction

When deleting nodes or clearing sketches, the node instance will be destroyed and its actor stopped.

Revision #5
Created 25 January 2024 16:00:17 by machiel
Updated 5 March 2024 14:54:38 by Astrid