SNAP Environment for Interactive Experimental Control

EEGLAB Workshop 2013, Track B

Christian A Kothe
SCCN, UCSD
Purpose

• Allow to *rapidly prototype* neuroscience experiments and game-like human-computer interactions with significant complexity

• *Generalize and advance* basic neuroscience experiments *towards practically relevant applications*

• Full source code, *no license restrictions on academic or commercial use and deployment* ([https://github.com/sccn/SNAP](https://github.com/sccn/SNAP))
Approach

- Relies on **Python** as the scripting language and leverages its packages
- Uses the **Panda3d** game engine for graphics, audio, input, physics, GUI and low-level real-time subsystems
- Adds a thin **layer for experiment scripting**
- Adds some **extra low-level subsystems** (LSL, RPC, Pathfinding, ...)

Swartz Center for Computational Neuroscience
The Panda3d Engine

• **Con:**
  – Relatively clean but dated core (’97) but still actively developed, messy features at the outer fringes
  – Limited support for in-engine editing
  – no modern lighting/rendering model

• **Pro:**
  – Complete game engine, formerly commercial (Disney), now open source (MIT license) and maintained by CMU
  – Written in C++ (fast) and scriptable via Python (convenient)
  – Very comprehensive feature set for game/simulation purposes (750k LOC)
  –Remarkably good documentation (panda3d.org)
SNAP Architecture

User-Created Experiment Modules
- DAS
- MBF
- LSE
- Flanker
- Speech

SNAP Components
- Stimulus Presentation
- Event Markers
- UI Tools
- Task Prefabs
- Misc Tools

Panda3d
- Core
- Graphics
- Audio
- Physics
- GUI
- Network

Python and Packages
- Python
- RPyC
- Win32
Basic Scripting

```python
from framework.latentmodule import LatentModule
import random

class Main(LatentModule):
    def __init__(self):
        LatentModule.__init__(self)

        # set defaults for some configurable parameters:
        self.num_trials = 50  # number of trials in first part
        self.text_probability = 0.5  # probability that a text is displayed instead of a picture

    def run(self):
        self.marker(10)  # emit an event marker to indicate the beginning of the experiment
        self.write('This is a sample experiment. You will be lead through a few trials in the first part. Press the space bar when you are ready. ', 'space')

        for k in range(self.num_trials):
            # show a 3-second cross-hair
            self.crosshair(3)
            # display either a text or a picture
            if random.random() < self.text_probability:
                self.marker(1)
                self.write('A text. ', 'text', scale=0.5)
            else:
                self.marker(2)
                self.picture('monkey.jpg', 2, scale=0.3)
            # wait for 2 seconds
            self.sleep(2)

        self.sound('nice_bell.wav', volume=0.5)
        self.write('You successfully completed the experiment.')
```
Complex Scripting

- Example: earlier experiment prototype (MBF)
Relationship to LSL and BCILAB

• Natively sends event marker streams to LSL
• Can be remotely controlled by the LSL LabRecorder experiment control features (e.g., load/config/start/stop)
• BCILAB can remotely control the value of any set of module parameters online (for example, the “task load level”)
• Can also read/write any LSL stream manually through the Python API
Benefits and Caveats

• Pro:
  – Basic scripting is as simple as it can get
  – Scales gracefully from basic to very complex experiments, both in terms of features and performance
  – Integrates painlessly with LSL and BCILAB

• Con:
  – Design workflow not as convenient and simple as rich visual editors of commercial products
  – Limited user support (but solid documentation)
Demo
Questions?