

150 Smoots Lighting Installation Design Document

Russell Cohen

November 23, 2010

1 Intro

The system, which we will describe henceforth as SmootLight is a modular system designed with the following goals in mind:

- The system must abstract away from all components while remaining useful (**Renderer**, **Input**, **Behavior**)
- The system must be modular and be easy to write new code for.
- More goals as I think of them

We accomplish this in the following manner:

- The system is configured by an XML file which specifies its components.
- All classes are initialized with a dictionary as an argument containing anything a class may need. All objects are passed between members as python dictionaries because their easy serialization.

2 Operations Class Patterns

2.1 SmootCoreObject

- **Inherits from:** None
- **Inherited by:** All 2nd level classes (PixelAssembler, Renderer, Input, Behavior)
- **Brief Description:** SmootCoreObject is essentially a super-object that makes things easy for us. It does the following actions:
 - Defines a constructor that sets argDict
 - Defines a `__getitem__`, which lets us access items in argDict as if the class was a dictionary. (`self['itemName']`)
 - Defines `validateArgs` and `validateArgDict` which validate the incoming arguments against a dictionary containing argument names as keys and an error message to display if they are missing as values. This file should be named `classname.params` and look like a python dict (`{'key':value, 'key2':value2}`)

Note that at this point, the only class using this functionality is the PixelEvent class.

- **Argument Requirements:** No required parameters in argDict

2.2 PixelAssembler

- **Inherits from:** SmootCoreObject
- **Inherited by:** LineLayout, ZigzagLayout
- **Brief Description:** PixelAssembler is a class that defines the positions of lights. It provides a method `getLightLocations` which give a list of all light locations for a given strip. Inheriting classes must define `layoutFunc` which returns the next location given the previous location. (They may simply override `getLightLocations` instead, if they wish, but be careful when doing so). In heriting classes may defint `initLayout` which is called at initialization.
- **Argument Requirements:**
 - `lightToLightSpacing`: this is the length of wire between 2 adjacent LEDs. Common values are 4 or 12.
 - `numLights`: Number of lights in a strip.
 - `originLocation`: Location of the first light.

2.3 Renderer

- **Inherits from:** SmootCoreObject
- **Inherited by:** PygameRenderer, IndoorRenderer
- **Brief Description:** Renderer is a class that serves as an abstract class for renderers interacting with the system. Inheriting classes must define `render`, which is passed a `lightSystem` object. Inheriting classes may define `initRenderer` which is called on initiation.
- **Argument Requirements:** No required arguments

2.4 Input

- **Inherits from:** SmootCoreObject, `threading.Thread`
- **Inherited by:** PygameInput, TCPInput, UDPInput
- **Brief Description:** Input is a abstract class which facilitates Inputs. It does this by providing a method that is polled at a periodic interval within which the inheriting class can raise an event. Inheriting classes must define `sensingLoop` which is called at the interval specified in the config. Inheriting classes should call `respond` with an dictionary as an argument to raise an event. Classes using (not inheriting) input must pass a scope into the `argDict` which offers a `processInput` method. Inputs are marked as Daemon threads, and are therefore killed when their parent is killed.
- **Argument Requirements:**
 - `InputId`: The string id of a given input. Must be unique.
 - `Optional:RefreshInterval`: The interval in seconds (will soon be changed to milliseconds) between sucessive calls to the `sensingLoop` method. TODO: make timeout.

2.5 Behavior

- **Inherits from:** SmootCoreObject
- **Inherited by:** EchoBehavior, DebugBehavior
- **Brief Description:** Abstract class for a behavior. On every time step, the behavior is passed the inputs from all sensors it is bound to as well as any recursive inputs that it spawned during the last time step. Inheriting classes **MUST** define `processBehavior`. `processBehavior` should return a list of dictionaries which define the properties of the light response. The must return a location `PixelEvent` class. Soon be be deprecated: *They must give a location and color. They may define a function pointer which defines a custom mapping. [More on this later. Bug Russell if you want to do it].* Call `recursiveResponse` to queue a input on the next iteration with a dictionary argument. This will be passed in via recursive inputs.
- **Argument Requirements:**
 - **Inputs:** A list of input Ids specifying input to the behavior. In the future, this may also contain behavior ids.

2.6 PixelEvent

- **Inherits from:** SmootCoreObject
- **Inherited by:** StepResponse
- **Brief Description:** Abstract class defining the behavior of a light after it has been turned on. Inheriting classes should definit `lightState` which is passed a `timeDelay` in ms as an argument. `lightState` should return a color or `None` if the response is complete.
- **Argument Requirements:**
 - **Color:** The color of response. *This is may be removed in the future*

3 The Screen Class and Its Relatives

3.1 Screen

- **Inherits from:** None
- **Inherited by:** None
- **Brief Description:** The `Screen` class is a representation of an entire system of pixels, distributed over space. The `Screen` class and its relatives process responses (mapped to pixels via a `LayoutEngine`), and add `PixelEvents` to the individual pixels. The `Screen` provides an instance that renderers can call to determine the color of all the individual pixels. It contains a list of `PixelStrips`, which each address the individual pixels. `Screen` offers a `respond` method which takes a dictionary containing information about a response. TODO: detail the contents of this dictionary (maybe somewhere else). Note: `Screen` will not process its responses until `timeStep` is called which processes all responses that have been queued since the last time that `timeStep` was called. `Screen` also offers an iterator over *all* lights, accesible by using an expression like: `for light in screen:.` For addressing of specific `PixelStrips`, `self.pixelStrips` is exposed.
- **Argument Requirements:** No required parameters

3.2 PixelStrip

- **Inherits from:** None
- **Inherited by:** None
- **Brief Description:** The `PixelStrip` class is a representation of a string of Pixels that are connected in physical space (eg. a strip of lights). A `PixelStrip` takes a `LayoutBuilder` (*Name up for debate, currently known as layout engine*) as an argument. The `argDict` of the `LayoutBuilder` is passed becomes the `argDict` of the `PixelStrip`. `PixelStrip` generally shouldn't be addressed directly unless you need Strip-Identification for rendering purposes. You should never, for example, call **respond** on a `PixelStrip` directly, unless you really know what you're doing. Well, actually you should never need to do that. never. Don't do it.
- **Argument Requirements:** Takes a `LayoutBuilder` as an argument.

4 Best Practices

4.1 Variable and function naming

I'm pretty bad about being consistent. However, in all future code, please adhere to the following:

- Classes: `FirstLetterCaps`
- Functions: `camelCase`
- Property Names: `FirstLetterCaps`
- Constants: `ALL_CAPS_WITH_UNDERSCORES`

4.2 Time

For time, use the `Util.time()` method to return the current time in ms.

4.3 Accessing Pixels

The ideal method for accessing Pixels in a screen is to use its iterator. Iterating over the individual `PixelStrips` is also an acceptable method.

4.4 Determining the state of a Pixel

The best practice for determining the color of a `Pixel` is to call `lightState` (may be renamed to `State TODO`). This ensures all current active responses running on the `Pixel` contribute correctly.

4.5 Color

For color, use a tuple of (R,G,B) 0-255 for each. Colors can be easily manipulated with members of the `Util` class.