Pudding (a widget system for Soya)

Release 0.1-0

Dunk Fordyce, dunk@dunkfordyce.co.uk

May 28, 2005

Abstract

This document describes how to use Pudding with Soya. Pudding is a replacement widget system for Soya's current widget system.

Contents

1 What is Pudding?

Pudding is a widget system primarily for Soya, it could however with some tweaking be used for orther libraries such as pyopengl.

1.1 Why Pudding?

Pudding was started as a replacement to Soya's current widget module. The current module, while usefull, is hard to extend.

There are several other opengl UI libraries available but all have theyre problems or would be complicated to use with soya.

Pudding has been designed to allow components to be created from a core set of base classes. This allows the developer to create any sort of widget, either, virtually from scratch or from a higher level.

1.2 Some cake to have and eat

Here is a minimal example of using pudding for the impatient:

```
import soya
import pudding

soya.path.append('data')

soya.init()
pudding.init()

scene = soya.World()

sword_model = soya.Shape.get("sword")
sword = soya.Volume(scene, sword_model)
sword.x = 1
sword.rotate_lateral(90.)

light = soya.Light(scene)
light.set_xyz( .5, 0., 2.)

camera = soya.Camera(scene)
camera.z = 3.

soya.set_root_widget(pudding.core.RootWidget())
soya.root_widget.add_child(camera)

button_bar = pudding.container.HorizontalContainer(soya.root_widget,
                               left = 10, width= 164, height = 64)
button_bar.set_pos_bottom_right(bottom = 10)
button_bar.anchors = pudding.ANCHOR_BOTTOM

button1 = button_bar.add_child( pudding.control.Button(label = 'Button1'),
                                pudding.EXPAND_BOTH)
button2 = button_bar.add_child( pudding.control.Button(label = 'Button2'),
                                pudding.EXPAND_BOTH)

logo = pudding.control.Logo(soya.root_widget, 'mylogo.png')

pudding.idler.Idler(scene).idle()
```

2 Software Requirements

You need to have the following software installed:

- Python 2.3 http://python.org
- Soya (and all relevant dependancies) http://oomadness.tuxfamily.org/en/soya/index.html

Optional software includes:

- ElementTree http://effbot.org/zone/element-index.htm
- pycairo http://cairographics.org

3 Pudding Basics

This section will introduce the basics of `pudding`.

3.1 Initializing `pudding`

Using pudding is as simple as adding two extra statements to your Soya application.

```
import soya
import pudding

soya.init()
pudding.init()
```

You are now ready to create a `pudding` root widget to add components to.

3.2 The `RootWidget` class

To use pudding you *must* use `pudding.core.RootWidget`.

```
# ... initialize soya and pudding

soya.set_root_widget(pudding.core.RootWidget())
```

To add your camera to the root widget use:

```
# ... initialize soya and pudding

scene = soya.World()
camera = soya.Camera(scene)

soya.set_root_widget(pudding.core.RootWidget())

soya.root_widget.add_child(camera)
```

3.3 Hello World!

The infamous hello world script with pudding:

```
import soya
import pudding

soya.init()
pudding.init()

scene = soya.World()

camera = soya.Camera(scene)

soya.set_root_widget(pudding.core.RootWidget())
soya.root_widget.add_child(camera)

text = pudding.control.SimpleLabel(soya.root_widget, label = "Hello World!")

pudding.idler.Idler(scene).idle()
```

4 Module: pudding – Main pudding module

4.1 Functions

init(*style=None*)
Intialise {pudding}. {style} should be a subclass of {pudding.style.Style}

process_event()
This gets the event list from soya and filters it for any events handled by widgets. It returns an array with the events that have not been used. If you use the {pudding.idler.Idler} then this function is called in {idler.begin_round} and the events unprocessed put in {idler.events.}

4.2 Classes

exception ConstantError
Inherits: PuddingError Exception

Error using a {pudding} constant

exception PuddingError
Inherits: Exception

A {pudding} exception

4.3 Constants

ALIGN_BOTTOM

ALIGN_LEFT

ALIGN_RIGHT

ALIGN_TOP

ANCHOR_ALL

ANCHOR_BOTTOM

ANCHOR_BOTTOM_LEFT

ANCHOR_BOTTOM_RIGHT

ANCHOR_LEFT

ANCHOR_RIGHT

ANCHOR_TOP

ANCHOR_TOP_LEFT

ANCHOR_TOP_RIGHT

BOTTOM_LEFT

BOTTOM_RIGHT

CENTER_BOTH

CENTER_HORIZ

CENTER_VERT

CLIP_BOTTOM

CLIP_LEFT

CLIP_NONE

CLIP_RIGHT

CLIP_TOP

CORNERS

EXPAND_BOTH

EXPAND_HORIZ

EXPAND_NONE

EXPAND_VERT

STYLE

TOP_LEFT

TOP_RIGHT

5 Module: `pudding.core` – Core objects for `pudding`

5.1 Classes

class `Base`

Inherits:

The base class for all widgets. Note a Base control doesnt render anything to the screen or it does it in a fashion where position and size are not relevant. For graphical controls subclass {pudding.Control} instead

`child`
child object

`parent`
parent object

`advance_time`(*self, proportion*)
soya advance_time event

begin_round(*self*)
soya begin_round event

end_round(*self*)
soya.end_round event

on_init(*self*)
event occurs at the end of initialisation for user processing

on_set_child(*self, child*)
event triggered when the child attribute is set

process_event(*self, event*)
process one event. returning False means that the event has not been handled and should be passed on to other widgets. returning True means that the event has been handled and the event should no longer be propogated

class Control
Inherits: Base

The main graphical base class for all widgets.

anchors

bottom
distance from the bottom edge of the screen to the bottom edge of the control

height
height of the control

left
distance from the left edge of the screen to the left edge of the control

right
distance from the right edge of the screen to the right edge of the control

screen_bottom

screen_left

screen_right

screen_top

top
distrance from the top edge of the screen to the top edge of the control

visible
is the object visible

width
width of the control

do_anchoring(*self*)
move the control based on anchor flags

end_render(*self*)
shuts down opengl state

on_hide(*self*)
event when the control is made invisible

on_resize(*self*)
event when the control is resized

on_show(*self*)
event when the control is made visible

process_event(*self, event*)
process one event. returning False means that the event has not been handled and should be passed on to other widgets. returning True means that the event has been handled and the event should no longer be propogated.

render(*self*)
render the whole object. setup and take down opengl, render self and render all children

render_self(*self*)
renders the current object. ie dont render the children, render self

resize(*self, left, top, width, height*)
set the position and size of the control

set_pos_bottom_right(*self, right=None, bottom=None*)
whereas using .right and .bottom effect the width and height of the control this will effect the left and the top

start_render(*self*)
sets up opengl state

class InputControl
This class should be used with multiple inheritance to create some standard events. call InputControl.process_event(self,event) from your widgets process_event call.

Note the methods on_mouse*, on_key_*, on_focus and on_loose_focus

focus

on_focus(*self*)
event triggered when the control gets focus

on_key_down(*self, key, mods*)
event triggered when a key is pressed

on_key_up(*self, key, mods*)
event triggered when a key is released

on_loose_focus(*self*)
event triggered when the control looses focus

on_mouse_down(*self, x, y, button*)
event triggered when a mouse button is pressed

on_mouse_over(*self, x, y, buttons*)
event triggered when the mouse moves over the control

on_mouse_up(*self, x, y, button*)
event triggered when a mouse button is released

process_event(*self, event*)
process an individial event and then pass it on the correct event handler. if that handler returns True the event is assumed to of been dealt with

process_mouse_event(*self, event*)
process a mouse event. focus is set if the mouse is over the widget. the event handlers on mouse_* are called from here

class RootWidget
Inherits: `Container` `Control` `Base`

The root widget to be used with {pudding}.

If your display looks incorrect try resizing the window. If that corrects the display then you need to call root_widget.on_resize() at some point before the user gets control.

add_child(*self, child*)
Add a child to the root widget. {RootWidget} also accepts cameras as children altho these are stored in .cameras

on_init(*self*)
Declares self.cameras

on_resize(*self*)
Resize all cameras and children

start_render(*self*)
Load the identity matrix for the root widget

widget_advance_time(*self, proportion*)
Called once or more per round

widget_begin_round(*self*)
Called at the beginning of every round

widget_end_round(*self*)
Called at the end of every round

6 Module: pudding.control – most basic widget for pudding

6.1 Classes

class Button
Inherits: Box Control Base InputControl

A simple button widget. The label is a child SimpleLabel widget. Note the on_click method provided

label
label on the button

on_click(*self*)
event triggered when the button is ”clicked” either by the mouse or the keyboard

on_mouse_up(*self, x, y, button*)
use the mouse up event handler to implement the on_click handler

on_resize(*self*)
use the resize event to move and resize the buttons child label

render_self(*self*)
render the box with current settings

class Console
Inherits: VerticalContainer Container Control Base

A simple console style widget

on_focus(*self*)
automatically give focus to the input when the console gets focus

on_key_press(*self, key, mods*)
allow scrolling thru the buffer

on_loose_focus(*self*)
automatically give focus to the input when the console gets focus

on_resize(*self*)
update child controls

on_return(*self*)
send all input to the output and clear the input ready for more

class Image
Inherits: Control Base

A simple image control

material

rotation

shade

render_self(*self*)
render the image to screen

class Input
Inherits: Box Control Base InputControl

Simple input box using a child SimpleLabel widget. Note the on_value_changed method

cursor
text used as a cursor. defaults to '_'

prompt
static text used as a prompt

value
the text in the input box

clear(*self*)
clear the value of the input

on_focus(*self*)
append the cursor sybol when focus is gained

on_key_down(*self, key, mods*)
process and key strokes and add them to the current value

on_loose_focus(*self*)
remove the cursor symbol when the focus is lost

on_resize(*self*)
set the position and size of our child label

on_return(*self*)
event triggered when the return key is pressed

on_value_changed(*self*)
event triggered when the value is changed by the user

set_height_to_font(*self*)
set the height of the input control to the height of the font

class Label
Inherits: SimpleLabel Control Base InputControl

Label with events. Created using SimpleLabel and InputControl with multiple inheritance

MAXLEN

process_event(*self, event*)
let InputControl class deal with events

class Logo
Inherits: Image Control Base

Class to display an image in the bottom right corner, usefull for logo's

class Panel
Inherits: Box Control Base InputControl

A simple window/panel control with a title. modify the style class to change the way this is draw

label

process_event(*self, event*)
default event handling

render_self(*self*)
render the box with current settings

class PrePostLabel
Inherits: SimpleLabel Control Base

A label with static pre/post -fix

MAXLEN

post

pre

set_display_text(*self, text*)
set the display text with the pre and post strings

class SimpleLabel
Inherits: Control Base

A simple, unresponsive label widget

MAXLEN

autosize
should the label automatically adjust its size to accomodate all the text

clip
if there is too much text do we clip the left or right. must be pudding.core.[pudding.CLIP_LEFT — pudding.CLIP_RIGHT]

color
color of the text

font
font used for rendering

label
text displayed

wrap

on_resize(*self*)
update position with anchoring and apply wrapping/clipping

on_set_label(*self*)
event triggered when the label is changed

render_self(*self*)
draw the text with the current settings

set_display_text(*self, text*)
get the text we should actually display. usefull if you want to add a constant string or perform some processing before the string gets wrapped or clipped or whatever

update(*self*)
refresh settings based on clip and autoresize etc

7 Module: pudding.container – containers for pudding

7.1 Classes

class HorizontalContainer
Inherits: Container Control Base

class to resize all children in a row

on_resize(*self*)
resize all children into a row

class VerticalContainer
Inherits: Container Control Base

class to resize all children in a column

on_resize(*self*)
resize all children into a column

8 Module: pudding.idler – a simple replacement idler for soya

8.1 Classes

class Idler
Inherits: Idler

Simple replacement for the soya.Idler that calls pudding.process_event in begin_round and places all unhandled events into idler.events.

begin_round(*self*)
call pudding.process event and put all events in self.events so the ”game” can handle other events

idle(*self*)
resize all widgets and start the idler

9 pudding.sysfont – sysfont, used in the font module to find system fonts