Quick Multitouch Apps using kivy and Python

About Me!

Python and Kivy

+

Setting up Kivy...

1) in Linux

2) in Windows

3) Mac OSX

Hello World in Kivy :)

```
import kivy
kivy.require('1.0.6') # replace with your current kivy version

from kivy.app import App
from kivy.uix.button import Button

class MyApp(App):
    def build(self):
        return Button(text='Hello World')

if __name__ in ('__android__', '__main__'):
    MyApp().run()
```

Controlling the Environment

Many environment variables are available to control the initialization and behavior of Kivy.

$ KIVY_TEXT=cairo python main.py

Or set the variables before importing kivy:

```
import os
os.environ['KIVY_TEXT'] = 'cairo'
import kivy
```

Controlling the Environment

→ To force default config, use KIVY_USE_DEFAULTCONFIG

→ KIVY_WINDOW: pygame

→ KIVY_TEXT: pil, cairo, pygame

→ KIVY_VIDEO: gstreamer, pyglet, ffmpeg

→ KIVY_AUDIO: gstreamer, pygame

→ KIVY_IMAGE: pil, pygame

Controlling the Environment

→ KIVY_CAMERA: gstreamer, opencv, videocapture

→ KIVY_SPELLING: enchant, osxappkit

→ KIVY_CLIPBOARD: pygame, dummy

Configuring Kivy

The location of the configuration file is in:

<HOME_DIRECTORY>/.kivy/config.ini

If your user is named “karan”, the file will be located at:

Windows: C:\Users\karan\.kivy\config.ini
MacOSX: /Users/karan/.kivy/config.ini
Linux: /home/karan/.kivy/config.ini

Understanding the config file: read docs on kivy.config

Architecture of Kivy

1) Core Providers and Input Providers
2) Graphics
3) Core
4) UIX (Widgets & Layouts)
5) Modules
6) Input Events (Touches)
7) Widgets and Event Dispatching

Core Providers and Input Providers

We try to abstract from basic tasks such as opening a window, displaying images and text, playing audio, getting images from a camera, spelling correction and so on.

We call these *core* tasks.

Core providers allow us to accomplish these core tasks!

Core Providers and Input Providers

→ An input provider is a piece of code that adds support for a specific input device, such as Apple’s trackpads, TUIO or a mouse emulator.

→ If you need to add support for a new input device, you can simply provide a new class that reads your input data from your device and transforms them into Kivy basic events.

Graphics

→ Kivy’s graphics API is our abstraction of OpenGL. On the lowest level, Kivy issues hardware-accelerated drawing commands using OpenGL.

→ All of kivy's widgets themselves use this graphics API, which is implemented on the C level for performance reasons.

Graphics

→ The graphics API can automatically optimise the graphics commands issued by your kivy app.

→ You can, of course, still use raw OpenGL commands if you prefer that. :)

→ Targeted version is OpenGL 2.0 ES($GLES_2$)

→ use this for cross-platform compatibility

Core

Clock

You can use the clock to schedule timer events. Both one-shot timers and periodic timers are supported.

Cache

If you need to cache something that you use often, you can use our class for that instead of writing your own.
interface description.

Core

Gesture Detection

We ship a simple gesture recognizer that you can use to detect various kinds of strokes, such as circles or rectangles. You can train it to detect your own strokes.

Core

Kivy Language

The kivy language is used to easily and efficiently describe user interfaces.

Properties

These are not the normal properties that you may know from python. It is our own properties class that links your widget code with the user

UIX

Widgets

→ Widgets are user interface elements that you add to your program to provide some kind of functionality.

→ They may or may not be visible. Examples would be a file browser, buttons, sliders, lists and so on.

→ Widgets receive MotionEvents.

UIX

Layouts

→ You use layouts to arrange widgets.
It is of course possible to calculate your widgets' positions yourself, but often it is more convenient to use one of our ready made layouts.

→ Examples would be Grid Layouts or Box Layouts.

→ You can also nest layouts.

Modules

→ Similar to addons for broswers

→ we can add new functionality in old kivy apps

→ ex. FPS counter, exit button overlay

Input Events (Touches)

Down

→ A touch is down only once, at the very moment where it first appears.

Move

→ A touch can be in this state for a potentially unlimited time.

→ A ‘Move’ happens whenever the 2D position of a touch changes.

Input Events (Touches)

Up

→ A touch goes up at most once, or never.

→ obviously you wouldn't want to keep on pressing forever?

Widgets and Event Dispatching

Tree Based Widget Strucutre

→ Widgets from a hierarchy just like a tree

→ There is a root widget and children widgets

→ similar to other popular GUI hierarchy approaches...ex. Qt

Widgets and Event Dispatching

→ Is a widget hungry?

→ Digest and Pass of Events == just like us?

→ Flow of events along the Tree from Root to children

Widgets and Event Dispatching

```
# This is analogous for move/up:
def on_touch_down(self, touch):
    for child in self.children[:]:
        if child.dispatch('on_touch_down', touch):
            return True
```

Your First Kivy Widget :D

Lets have some more FUN! :D

Manipulating the Widget Tree

The tree can be manipulated with 3 methods:

add_widget(): add a widget as a child
remove_widget(): remove a widget from the children list
clear_widgets(): remove all children from a widget

```
layout = BoxLayout(padding=10)
button = Button(text='My first button')
layout.add_widget(button)
```

Events

You have 2 types of events living in Kivy:

→ Clock events: if you want to call a function X times per seconds, or if you want to call a function later. ex. Similar to timers in Qt

→ Widget events: if you want to call a function where something change in the widget, or attach a function to a widget specific event. e.x. Similar to signals in Qt

Clock Events

clock.schedule_interval(fn_name, time to call the fn)

clock.unschedule(fn_name) == to unschedule a previously scheduled event

Or return False in fn_name to automatically unschedule

clock.schedule_once(fn_name, time to call the fn)

Clock Events for Trigering

triggering can be achieved with:

```
Clock.unschedule(my_callback)
Clock.schedule_once(my_callback, 0)
```

But this is expensive, since no matter whether an event has been scheduled or not we are unscheduling it still...

Clock Events for Trigering

triggering can be achieved with:

```
Clock.unschedule(my_callback)
Clock.schedule_once(my_callback, 0)
```

But this is expensive, since no matter whether an event has been scheduled or not we are unscheduling it still...

Clock Events for Trigering

```
# better way to implemnent triggers =
Clock.create_trigger(my_callback)

# later on

trigger()
```

Widget Events

A widget have 2 types of events:

Property event: if your widget change of pos or size, you'll have an event fired, And we can also define our own custom properties too

Widget defined event: a Button will have even fired when it's pressed or released.ex. Button pressed or released

Input management

→ Input Event Architecture

→ Motion Profiles

→ Touch Events as specialized motion events

→ Touch Shapes

→ Double Tap

→ Grabbing Touches and Limitations

Kivy Language

→ Used for interface specification of kivy apps

→ can use .kv files to generate User Interfaces...

→ example?

Kivy on Android! Yay! :)

Kivy on Android!

→ how to run on Android?

→ how to package for Android?

→ Debugging on Android?

→ Supported Devices?

Kivy Packaging?

→ Kivy Packaging on Other Platforms?

→ Windows

→ Linux

→ MacOSX

Puzzled?

ANY DOUBTS?

Apart from me, go BUG these people!

Mailing Lists, kivy-users and kivy-dev on google groups

And

#kivy on irc.freenode.net

Finally!

The End :)