Présentation

Gabriel Pettier
Développeur python
Contributeur puis Core-Développeur Kivy depuis 2010

http://tshirtman.fr
http://github.com/tshirtman
http://twitter.com/tshirtman

http://tangibledisplay.com
Développement d'interactions tangibles

Kivy

Bibliothèque de Widgets orientés multitouch

- Développé majoritairement en pur Python, le reste en Cython pour les performances.
- Multiplatforme: Windows, OSX, Linux, Android et IOS supportés.
- API graphique directement basée sur OpenGL ES 2.0.
- Projet Libre et collaboratif, licence MIT.

Concepts

- Instruction

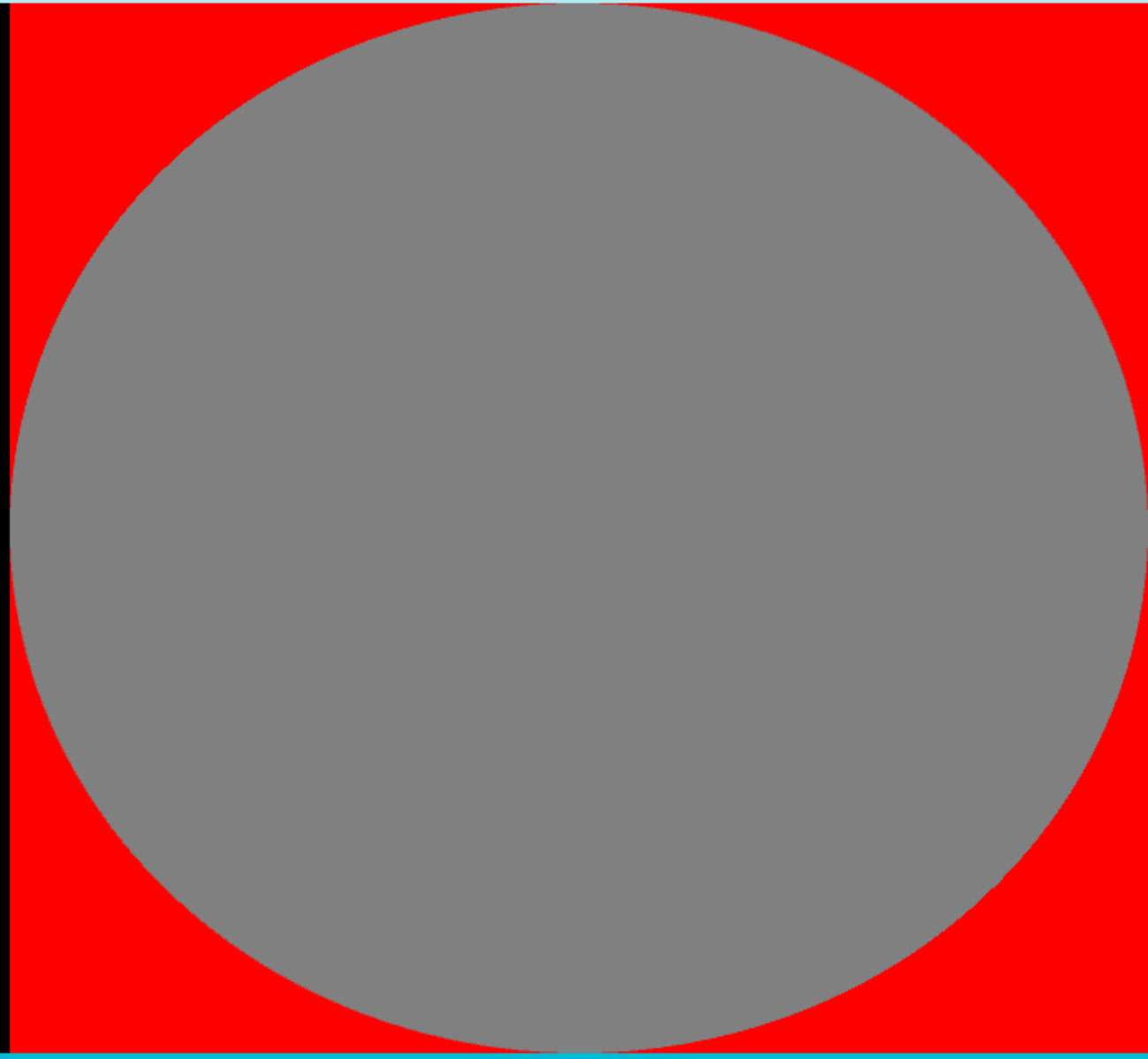
→ Permet d'indiquer un ordre au GPU

- Évènement

→ Propage une information à tous les acteurs concernés

Instruction exemple

```
with widget.canvas:
    Color(1, 0, 0, 1, mode='rgba')
    Rectangle(pos=widget.pos, size=widget.size)
    Color(.5, .5, .5, 1, mode='rgba')
    Ellipse(pos=widget.pos, size=widget.size)
```



Évènement

```
def function(*args):
    print(args)

widget.bind(on_touch_down=function)
```

touch me

<kivy.factory.ContentLabel object at 0x7fd9a794fc18>,<MouseMotionEvent spos=(0.9992679355783309, 0.0026041666666666297) pos=(1365.0, -36.00000000000003)>
<kivy.factory.ContentLabel object at 0x7fd9a794fc18>,<MouseMotionEvent spos=(0.9992679355783309, 0.0026041666666666297) pos=(1365.0, -36.00000000000003)>
<kivy.factory.ContentLabel object at 0x7fd9a794fc18>,<MouseMotionEvent spos=(0.9992679355783309, 0.0013020833333333703) pos=(1365.0, -36.99999999999997)>
<kivy.factory.ContentLabel object at 0x7fd9a794fc18>,<MouseMotionEvent spos=(0.9992679355783309, 0.0013020833333333703) pos=(1365.0, -36.99999999999997)>

Properties

Permet de lier les attributs des objets à des évènements.

- StringProperty
- NumericProperty
- ObjectProperty
- AliasProperty
- etc...

Doit être déclaré sur des sous-classes d'EventDispatcher, qui implémentent le pattern "Observer". Ainsi, il est possible à l'objet de réaliser des actions quand ses propriétés changent de valeur.

```
class ClassWithProperties(EventDispatcher):
    name = StringProperty('')
    count = NumericProperty(0)

def on_name(self, value):
    print "my name changed to %s" % value
```

Widgets

- Hérite d'EventDispatcher
- Contient un Canvas (groupe d'instructions).

Propose un certain nombre d'évènements par défaut, notamment:

- on_touch_down
- on_touch_move
- on_touch_up

```
class MyWidget(Widget):
   active = BooleanProperty(False)

    def __init__(self, **kwargs):
        super(MyWidget, self).__init__(**kwargs)
        self.bind(on_pos=self.update, on_size=self.update)
        with self.canvas:
            self.color = Color(1, 1, 1, 1, mode='rgba')
            self.rectangle = Rectangle(pos=self.pos, size=self.size)

    def on_active(self, active):
        if self.active:
            self.color.rgba = (.5, .5, 0, 1)
        else:
            self.coler.rgba = (1, 1, 1, 1)

    def update(self, *args):
        self.rect.pos = self.pos
        self.rect.size = self.size

    def on_touch_down(self, touch):
        if self.collide_point(*touch.pos):
            self.active = not self.active
```

Layouts

Les Widgets sont par défaut complètement libres de leur placement et de leur tailles, qui sont absolues (origine bas-gauche de l'écran).

Certains widgets sont spécialisés dans le placement et le dimensionnement de leurs sous widgets (enfants).

- FloatLayout
- BoxLayout
- AnchorLayout
- GridLayout
- StackLayout
- etc...

Layouts - 2 - Hints

La taille/position des widgets enfants d'un Layout est généralement contrôlée via les propriétés size_hint et pos_hint, qui sont relative au Layout.

```
f = FloatLayout()
f.add_widget(
    Button(
        size_hint=(.2, .1),
        pos_hint={'center': (.5, .5)})
```

La taille peut être définie arbitrairement en désactivant size_hint dans les directions souhaités:

```
f = FloatLayout()
f.add_widget(
    Button(
        size_hint_x=None,
        pos_hint={'center': (.5, .5)},
        size_hint_y=.1,
        width=250,
        text='another button'))
```

Le language KV

Python:

- super pour la logique
- moins pour la déclaration d'arbres de widgets.

Kv:

- Syntaxe déclarative
- Détection des dépendances et créations de branchements automatiques

```
w = Widget()
with w.canvas:
    Color(1, 0, 1, 1, mode='rgba')
    rect = Rectangle(pos=w.pos, size=w.size)

def update(self, *args):
    rect.pos = w.pos
    rect.size = w.size

w.bind(pos=update, size=update)
```

vs

```
from kivy.lang import Builder

KV = """
Widget:
    canvas:
        Color:
            rgba: 1, 0, 1, 1
        Rectangle:
            pos: self.pos
            size: self.size
"""

Builder.load_string(KV)
```

Le language KV - 2 - Bindings

1. Callback:

```
Widget:
    on_pos:
        print('my pos is %s' % self.pos)
Button:
    on_press:
        print("I've been pressed")
```

2. Dépendance:

```
Widget:
    canvas.before:
        Rectangle:
            pos: self.pos
            size: self.size
```

Le language KV - 3 - règles

- root rule: Ce qui sera retourné par le loader
  une seule autorisée par chaine/fichier.

```
MyWidget:
    Label:
        text: 'example'
```

- règle de classe: permet de configurer le style et le
  contenus de toutes les instances d'une classe.

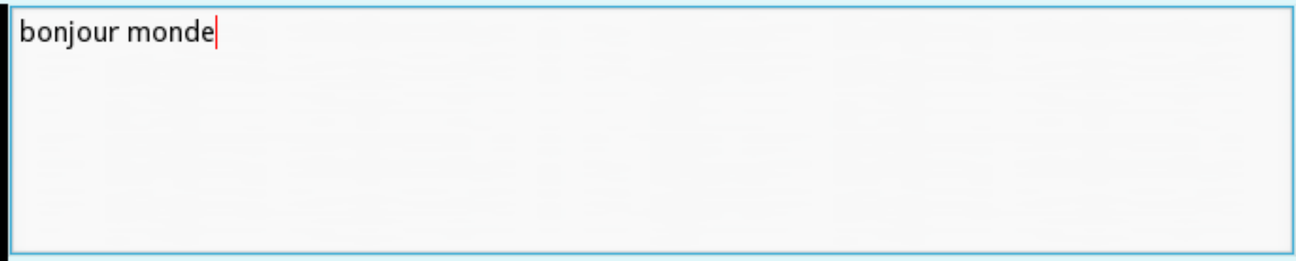
```
<Widget>:
    Label:
        text: 'example'
```

- Classes dynamiques: permet de créer une classe en KV,
  en déclarant l'héritage avec '@'.

```
<MyWidget@Widget>:
    Label:
        text: 'example'
```

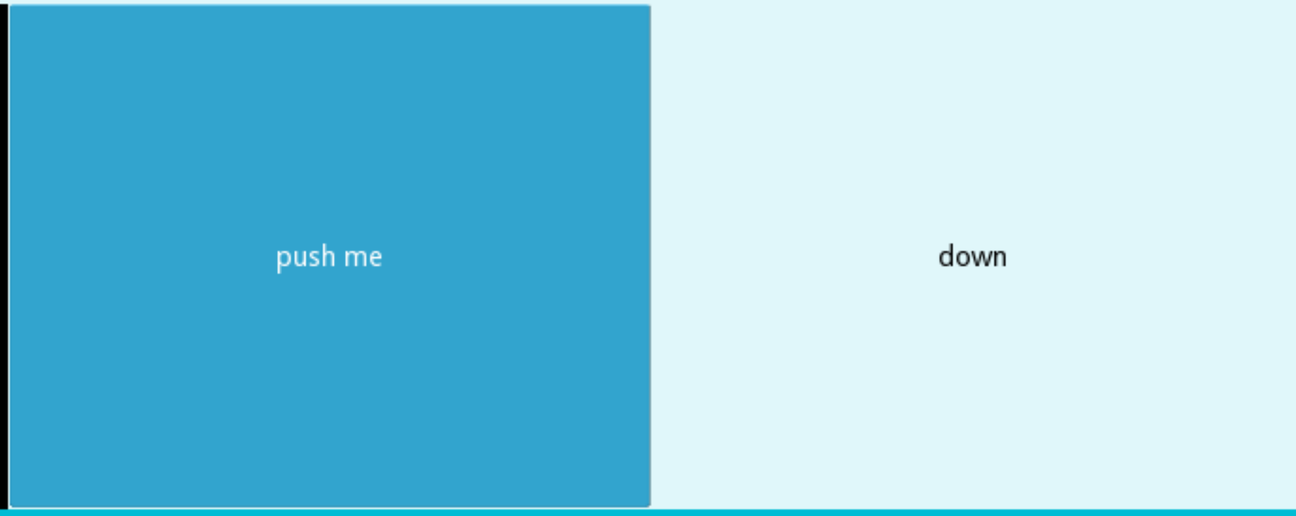
Utilisation d'id et ids pour référencer d'autres widgets

```
BoxLayout:
    orientation: 'vertical'
    TextInput:
        id: ti
    Label:
        text: ti.text
```



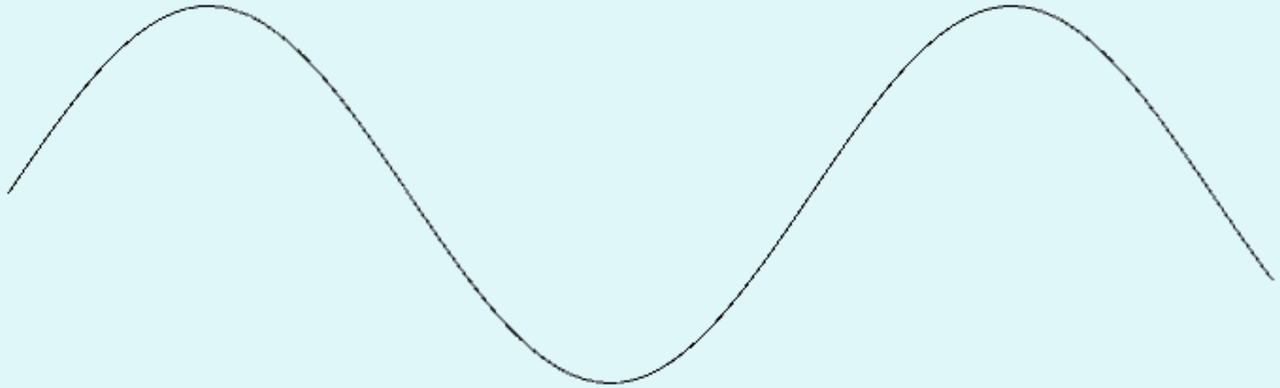

```
<MyWidget@BoxLayout>:
    Button:
        id: btn
        text: 'push me'

BoxLayout:
    Label:
        text: box.ids.btn.state
    MyWidget:
        id: box
```



Le language KV - 5 - directives

```
#:import sin math.sin
#:import chain itertools.chain
Widget:
    canvas:
        Color:
            rgba: 0, 0, 0, 1
        Line:
            points:
                list(chain(*zip(
                [self.x + x * self.width / 100.
                for x in range(100)],
                [self.center_y + sin(y/10.) * 100
                for y in range(100)])))
```



```
#:set font_size 35
#:set default_color (0, 0, 0, 1)

Label:
    font_size: font_size
    text: "hello world"
    color: default_color
```

hello world

```
#:include somefile.kv

SomeWidgetDefinedInSomeFile:
    text: "Hey"
```

Hey

Application

Deux manière de lancer une application:

- runTouchApp

- App().run()

```
from kivy.base import runTouchApp
from kivy.uix.label import Label

runTouchApp(
    Label(
        text='hello world'))
```

```
from kivy.app import App
from kivy.uix.label import Label

class HelloWorld(App):
    def build(self):
        return Label(
            text='hello world')

HelloWorld().run()
```

Applications - 2 - avantages

- App offre de nombreuses fonctions utilitaires (on_pause, on_stop, on_resume, build_config, build_settings...)
- App charge le fichier kv associé à la classe (convention de nommage 'HelloWorldApp → helloworld.kv) dans la methode build par défaut, et le fichier de configuration (helloworld.ini) si existant.
- App est un EventDispatcher → peut servir de contrôleur principal (accessible depuis kv via le mot clé "app").

Utilitaires

- Clock
  Permet de planifier des taches répétitives
  (schedule_interval) ou non (schedule_once)
- Animation
  Transitionne la valeur d'une NumericProperty ou
  ListProperty (contenant un nombre fixe de valeurs
  Numériques) d'une valeur à une autre, dans un temps donné,
  via une fonction de transition configurable.
- UrlRequest
  urllib wrapper pour simplifier le travail en arrière
  plan (event lors du succès/échec)

etc

Outils

Kivy vient avec un certains nombre d'outils pour faciliter le développement, les connaître peut vous faire gagner beaucoup de temps.

▸ Modules:

Permettent de modifier le comportement d'une application kivy:

→ Inspector
→ Monitor
→ Recorder
→ Monitor
→ Screen

▸ Garden

Modules communautaires à importer directement dans votre projet

→ garden.graph
→ garden.pi
→ garden.ddd
→ etc...

Questions ?

Bonus: Nouveautés 1.9

- SDL2 provider (good bye pygame!)
- ffpyplayer video provider (ffmpeg)
- EffectWidget (shaders)
- Window's KeyboardHeight property (android)
- support SVG!
- Rebind in kv
- Tesselator

+ nombreuses corrections et ajouts mineurs.