

# Introduzione a PyClutter

**Giuliani Vito, Ivan**

kratorius@lugbari.org

<http://zeta-puppis.com>

LinuxDay 2007



# Cos'è clutter

Clutter é:

- un *toolkit* per la creazione di interfacce grafiche animate ed accattivanti tramite l'uso delle librerie OpenGL
- scritto in C
- licenziato sotto LGPL<sup>1</sup>
- ancora in sviluppo

---

<sup>1</sup><http://www.gnu.org/licenses/lgpl.html>

# Cosa non é clutter

Clutter non é adatto a:

- creare giochi in OpenGL
- creare interfacce grafiche *comuni*



# PyClutter

Clutter dispone di *bindings* per:

- Perl
- Mono
- Vala
- Ruby
- **Python**

Tutto quello che serve per iniziare ad usare clutter in un programma scritto in Python é:

```
import clutter
```



## Gli oggetti disponibili

- Gli elementi visuali sono chiamati **actors** (attori)
- Non esiste un vero e proprio concetto di “finestra”, piuttosto esiste lo **stage**
- Non é possibile avere stage multipli (per il momento)
- Gli effetti grafici vengono applicati tramite i **behaviour**
- É possibile *combinare* gli actors uno con l'altro per creare ulteriori nuovi actors



## Cosa sono gli actors

Volendo fare un paragone, gli attori sono il corrispettivo dei widget delle GTK.

Sugli actors é possibile:

- impostare il livello di trasparenza
- effettuare operazioni di ridimensionamento
- applicare animazioni
- ...



# Le animazioni

Le animazioni vengono create utilizzando due “oggetti” che clutter mette a disposizione:

- la timeline
- i behaviour

In realtà esiste anche un altro “oggetto”, chiamato *alpha* che é necessario per poter creare le animazioni.



## Come é composta un'animazione?

Per poter creare un animazione bisogna compiere i seguenti passi:

- É necessario creare una **timeline**, ovvero una funzione che *conti* il tempo
- Bisogna associare un'**alpha function** alla timeline
- Bisogna impostare un **behaviour**, ovvero un certo tipo di animazione



## Come é composta un'animazione?

Per poter creare un animazione bisogna compiere i seguenti passi:

- É necessario creare una **timeline**, ovvero una funzione che *conti* il tempo
- Bisogna associare un'**alpha function** alla timeline
- Bisogna impostare un **behaviour**, ovvero un certo tipo di animazione



## Come é composta un'animazione?

Per poter creare un animazione bisogna compiere i seguenti passi:

- É necessario creare una **timeline**, ovvero una funzione che *conti* il tempo
- Bisogna associare un'**alpha function** alla timeline
- Bisogna impostare un **behaviour**, ovvero un certo tipo di animazione



# La timeline

- La **timeline** é un oggetto di clutter (`clutter.Timeline`)
- Serve a specificare il *tempo dell'animazione*
- I suoi parametri sono il numero di FPS<sup>2</sup> e il numero di frame totali
- Si può fare in modo che vada in *loop*

---

<sup>2</sup>Frames per secondo

# L'alpha function

- Senza un' *alpha function*, non sarebbe possibile creare animazioni solo con la timeline
- L' *alpha function* è una funzione che va legata alla timeline che, a ogni cambiamento di frame, restituisce un numero che è dipendente solo dal tempo



# I behaviour

- I behaviour sono oggetti predefiniti che controllano il comportamento di uno o più elementi dell'applicazione
- Vanno associati a un'alpha function e applicati a un actor
- É possibile creare behaviour da zero o utilizzare quelli preesistenti:
  - `clutter.BehaviourOpacity`
  - `clutter.BehaviourDepth`
  - `clutter.BehaviourPath`
  - `clutter.BehaviourRotate`
  - ...



## Il preambolo

Per usare clutter bisogna importare un solo modulo:

```
import clutter
```

Cosí facendo avremo a disposizione già tutti gli oggetti e funzioni di cui potremo aver bisogno.



## La classe principale

Nella creazione del programma, é stata utilizzata una classe chiamata `Main` che creerà la finestra e ne gestirà gli eventi/segnali.

Questa classe verrà istanziata non appena parte il programma:

```
def main():  
    try:  
        Main()  
    except KeyboardInterrupt:  
        print "Closed by CTRL+C"  
        return True  
    return False  
  
if __name__ == '__main__':  
    sys.exit(main())
```



## L'istanziamento della classe

- Tutte le classi in Python possono disporre di un metodo `__init__()` che viene chiamato quando la classe viene istanziata.
- Il nostro `__init__()` si occuperà di chiamare le varie funzioni (metodi della classe) che *disegneranno* la finestra.
- Una volta che la finestra sarà stata *disegnata*, verrà invocato il metodo `main` di `clutter`.



## L'inizializzazione della classe

Nel momento in cui la classe viene istanziata, viene chiamato il metodo `__init__`:

```
class Main:
    def __init__(self):
        self.setupStage()      # impostiamo lo stage
        self.setupLabel()     # impostiamo la label
        self.setupAnimation() # creiamo l'animazione

        self.stage.show_all() # mostriamo tutto...
        self.timeline.start()
        clutter.main()        # e avviamo l'applicazione
```



## L'inizializzazione della classe

Nel momento in cui la classe viene istanziata, viene chiamato il metodo `__init__`:

```
class Main:
    def __init__(self):
        self.setupStage()      # impostiamo lo stage
        self.setupLabel()     # impostiamo la label
        self.setupAnimation() # creiamo l'animazione

        self.stage.show_all() # mostriamo tutto...
        self.timeline.start()
        clutter.main()        # e avviamo l'applicazione
```



## L'inizializzazione della classe

Nel momento in cui la classe viene istanziata, viene chiamato il metodo `__init__`:

```
class Main:
    def __init__(self):
        self.setupStage()      # impostiamo lo stage
        self.setupLabel()     # impostiamo la label
        self.setupAnimation() # creiamo l'animazione

        self.stage.show_all() # mostriamo tutto...
        self.timeline.start()
        clutter.main()        # e avviamo l'applicazione
```



## L'inizializzazione della classe

Nel momento in cui la classe viene istanziata, viene chiamato il metodo `__init__`:

```
class Main:
    def __init__(self):
        self.setupStage()      # impostiamo lo stage
        self.setupLabel()     # impostiamo la label
        self.setupAnimation() # creiamo l'animazione

        self.stage.show_all() # mostriamo tutto...
        self.timeline.start()
        clutter.main()        # e avviamo l'applicazione
```





## Creiamo lo stage

```
def setupStage(self):  
    self.stage = clutter.stage_get_default()  
    self.stage.set_color(clutter.color_parse('#006666'))  
    self.stage.set_size(400, 100)  
    self.stage.connect('key-press-event', clutter.main_quit)
```

- Recuperiamo lo stage di default
- Impostiamo il colore dello sfondo a #006666
- Impostiamo la grandezza della finestra a 400x100 pixels
- Facciamo in modo che la pressione di un tasto chiuda la finestra



## Creiamo lo stage

```
def setupStage(self):  
    self.stage = clutter.stage_get_default()  
    self.stage.set_color(clutter.color_parse('#006666'))  
    self.stage.set_size(400, 100)  
    self.stage.connect('key-press-event', clutter.main_quit)
```

- Recuperiamo lo stage di default
- Impostiamo il colore dello sfondo a #006666
- Impostiamo la grandezza della finestra a 400x100 pixels
- Facciamo in modo che la pressione di un tasto chiuda la finestra



## Creiamo lo stage

```
def setupStage(self):  
    self.stage = clutter.stage_get_default()  
    self.stage.set_color(clutter.color_parse('#006666'))  
    self.stage.set_size(400, 100)  
    self.stage.connect('key-press-event', clutter.main_quit)
```

- Recuperiamo lo stage di default
- Impostiamo il colore dello sfondo a #006666
- Impostiamo la grandezza della finestra a 400x100 pixels
- Facciamo in modo che la pressione di un tasto chiuda la finestra



## Creiamo lo stage

```
def setupStage(self):  
    self.stage = clutter.stage_get_default()  
    self.stage.set_color(clutter.color_parse('#006666'))  
    self.stage.set_size(400, 100)  
    self.stage.connect('key-press-event', clutter.main_quit)
```

- Recuperiamo lo stage di default
- Impostiamo il colore dello sfondo a #006666
- Impostiamo la grandezza della finestra a 400x100 pixels
- Facciamo in modo che la pressione di un tasto chiuda la finestra



## Creiamo la label

```
def setupLabel(self):  
    self.label = clutter.Label()  
    self.label.set_font_name('Mono 32')  
    self.label.set_text('Hello world!!!')  
    self.label.set_color(clutter.color_parse('#FFFFFF'))  
    self.stage.add(self.label)
```

In questo modo:

- Creiamo l'oggetto etichetta
- Gli diciamo di usare il font "Mono", grandezza 32 pixel
- L'etichetta deve contenere il valore "Hello world!!!"
- E il testo dev'essere bianco
- Aggiungiamo l'etichetta all'interno dello stage



## Creiamo la label

```
def setupLabel(self):  
    self.label = clutter.Label()  
    self.label.set_font_name('Mono 32')  
    self.label.set_text('Hello world!!!')  
    self.label.set_color(clutter.color_parse('#FFFFFF'))  
    self.stage.add(self.label)
```

In questo modo:

- Creiamo l'oggetto etichetta
- Gli diciamo di usare il font "Mono", grandezza 32 pixel
- L'etichetta deve contenere il valore "Hello world!!!"
- E il testo dev'essere bianco
- Aggiungiamo l'etichetta all'interno dello stage



## Creiamo la label

```
def setupLabel(self):  
    self.label = clutter.Label()  
    self.label.set_font_name('Mono 32')  
    self.label.set_text('Hello world!!!')  
    self.label.set_color(clutter.color_parse('#FFFFFF'))  
    self.stage.add(self.label)
```

In questo modo:

- Creiamo l'oggetto etichetta
- Gli diciamo di usare il font "Mono", grandezza 32 pixel
- L'etichetta deve contenere il valore "Hello world!!!"
- E il testo dev'essere bianco
- Aggiungiamo l'etichetta all'interno dello stage



## Creiamo la label

```
def setupLabel(self):  
    self.label = clutter.Label()  
    self.label.set_font_name('Mono 32')  
    self.label.set_text('Hello world!!!')  
    self.label.set_color(clutter.color_parse('#FFFFFF'))  
    self.stage.add(self.label)
```

In questo modo:

- Creiamo l'oggetto etichetta
- Gli diciamo di usare il font "Mono", grandezza 32 pixel
- L'etichetta deve contenere il valore "Hello world!!!"
- E il testo dev'essere bianco
- Aggiungiamo l'etichetta all'interno dello stage



## Creiamo la label

```
def setupLabel(self):  
    self.label = clutter.Label()  
    self.label.set_font_name('Mono 32')  
    self.label.set_text('Hello world!!!')  
    self.label.set_color(clutter.color_parse('#FFFFFF'))  
    self.stage.add(self.label)
```

In questo modo:

- Creiamo l'oggetto etichetta
- Gli diciamo di usare il font "Mono", grandezza 32 pixel
- L'etichetta deve contenere il valore "Hello world!!!"
- E il testo dev'essere bianco
- Aggiungiamo l'etichetta all'interno dello stage



## Creiamo l'animazione

```
def setupAnimation(self):  
    self.timeline = clutter.Timeline(30, 25)  
    self.timeline.set_loop(True)  
    alpha = clutter.Alpha(self.timeline, clutter.sine_func)  
    self.behaviour = clutter.BehaviourOpacity(alpha, 0xdd, 0)  
    self.behaviour.apply(self.label)
```

Cosí facendo:

- Creiamo la timeline
- E la mettiamo in *loop*
- Creiamo la funzione alpha *collegandola* alla funzione seno
- Creiamo il behaviour che controllerá l'opacitá (e quindi la trasparenza dell'oggetto a cui verrà collegato)
- ...e lo applichiamo all'oggetto

## Creiamo l'animazione

```
def setupAnimation(self):  
    self.timeline = clutter.Timeline(30, 25)  
    self.timeline.set_loop(True)  
    alpha = clutter.Alpha(self.timeline, clutter.sine_func)  
    self.behaviour = clutter.BehaviourOpacity(alpha, 0xdd, 0)  
    self.behaviour.apply(self.label)
```

Cosí facendo:

- Creiamo la timeline
- E la mettiamo in *loop*
- Creiamo la funzione *alpha collegandola* alla funzione seno
- Creiamo il behaviour che controllerá l'opacitá (e quindi la trasparenza dell'oggetto a cui verrà collegato)
- ...e lo applichiamo all'oggetto

## Creiamo l'animazione

```
def setupAnimation(self):  
    self.timeline = clutter.Timeline(30, 25)  
    self.timeline.set_loop(True)  
    alpha = clutter.Alpha(self.timeline, clutter.sine_func)  
    self.behaviour = clutter.BehaviourOpacity(alpha, 0xdd, 0)  
    self.behaviour.apply(self.label)
```

Cosí facendo:

- Creiamo la timeline
- E la mettiamo in *loop*
- Creiamo la funzione alpha *collegandola* alla funzione seno
- Creiamo il behaviour che controllerá l'opacitá (e quindi la trasparenza dell'oggetto a cui verrà collegato)
- ...e lo applichiamo all'oggetto



## Creiamo l'animazione

```
def setupAnimation(self):  
    self.timeline = clutter.Timeline(30, 25)  
    self.timeline.set_loop(True)  
    alpha = clutter.Alpha(self.timeline, clutter.sine_func)  
    self.behaviour = clutter.BehaviourOpacity(alpha, 0xdd, 0)  
    self.behaviour.apply(self.label)
```

Cosí facendo:

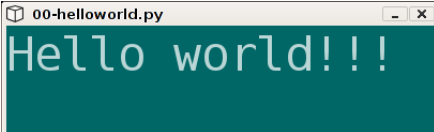
- Creiamo la timeline
- E la mettiamo in *loop*
- Creiamo la funzione alpha *collegandola* alla funzione seno
- Creiamo il behaviour che controllerá l'opacitá (e quindi la trasparenza dell'oggetto a cui verrà collegato)
- ...e lo applichiamo all'oggetto

## Mostriamo tutto

```
self.stage.show_all()  
self.timeline.start()  
clutter.main()
```

Cosí facendo mostriamo tutti gli actors e *avviamo l'applicazione...*

# L'hello world



```
00-helloworld.py
Hello world!!!
```

Figura: Il nostro “hello world”

# Riferimenti

- **Sito ufficiale di Clutter:**  
<http://www.clutter-project.org>
- **Sito ufficiale di Python:**  
<http://www.python.org>



# Licenza

Queste slide sono rilasciate con licenza GFDL<sup>3</sup>

---

<sup>3</sup><http://www.gnu.org/copyleft/fdl.html>