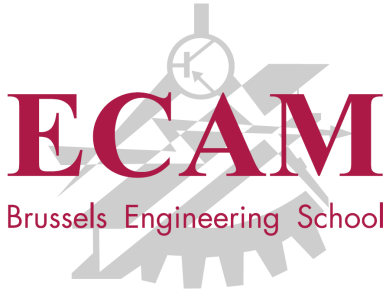


PI2T Développement informatique

Séance 8

Programmation graphique

Sébastien Combéfis, Quentin Lurkin

17 mars 2016

Objectifs

- Programmation graphique et évènementielle
  - Utilisation de la librairie Kivy
  - Application graphique et widgets
- Programmation de jeu
  - Dessin avec les canvas
  - Gestion des collisions


Framework Kivy

Librairie Kivy

- Framework open-source pour créer des interfaces utilisateur

  Application desktop ou mobile, jeux...

- Plusieurs avantages offerts par la librairie
  - Multi-plateforme (Linux, Windows, OS X, Android, iOS)
  - Framework stable, API documentée...
  - Moteur graphique basé sur OpenGL ES 2 (utilisation du GPU)

- Kivy est disponible sur GitHub

  https://github.com/kivy/kivy

Hello World (1)

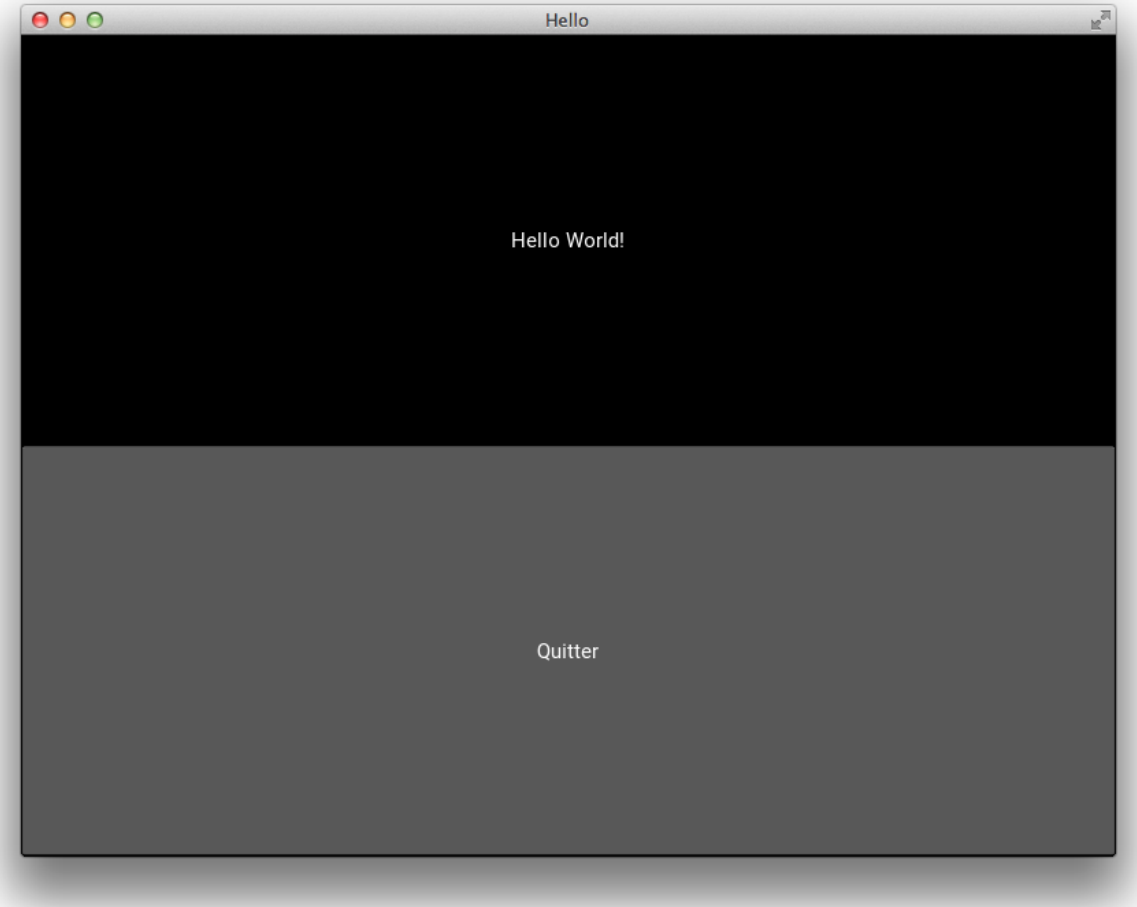
- Interface représentée par une classe de type `App`
  - Définition des composants dans la méthode `build`
  - Liaison d'un gestionnaire d'évènement avec `bind`

```
1  from kivy.app import App
2  from kivy.uix.button import Button, Label
3  from kivy.uix.boxlayout import BoxLayout
4
5  class HelloApp(App):
6      def build(self):
7          box = BoxLayout(orientation='vertical')
8          box.add_widget(Label(text='Hello World!'))
9          quitbtn = Button(text='Quitter')
10         quitbtn.bind(on_press=self._quit)
11         box.add_widget(quitbtn)
12         return box
13
14     def _quit(self, instance):
15         App.get_running_app().stop()
16
17 if __name__ == '__main__':
18     HelloApp().run()
```

Hello World (2)

- Interface graphique découpée en deux parties verticalement

  Label en haut et bouton en bas



Application minimale

- Créer le programme dans un fichier `main.py`

  Convention utile pour builder les versions mobiles

- Objet de type `App` initialise une série de choses avec `run`
  - interaction avec le hardware de l'écran
  - discuter avec des dispositifs d'entrée comme écran multitouch, clavier, accéléromètre...
  - planification de plusieurs tâches

```
1 from kivy.app import App
2
3 App().run()
```

Application graphique

- Code d'une application graphique placé dans une classe

  La classe doit être de type `App`

- Lancement de l'application par la méthode `run`

- Possibilité de décrire l'interface avec le langage KV

  Langage balisé de description d'interfaces graphiques

```
1  from kivy.app import App
2
3  class MetroApp(App):
4      pass
5
6  if __name__ == '__main__':
7      MetroApp().run()
```

Langage KV

- Fichier `.kv` qui porte le même nom que l'application

  Donc `metro.kv` dans notre exemple

- Liste des composants avec leurs propriétés

  Label, champ texte, layout, onglets...

```
1  #:kivy 1.0
2  # metro.kv
3  # author: Sébastien Combéfis
4  # version: March 16, 2016
5
6  Label:
7      text: "Hello World"
```

Widget

- Boite avec un comportement et pouvant contenir des boites

  La classe `Widget` représente une boite vide

- Exemples de widgets
  - Un `Label` permet d'afficher un texte
  - Un `Button` permet de répondre à un toucher ou click
  - Un `TextInput` réagit aux évènements clavier
- Un widget complexe peut être construit à partir d'autres

  `TabbedPanel`, `FileChooser`...

Hiérarchie de widgets

- Déclaration d'un widget racine par fichier KV

  Il s'agit simplement de celui déclaré le plus à l'extérieur

- Chaque déclaration possède un bloc *specifier*
  - Définition de propriétés du widget
  - Définition de widgets fils
- Le widget racine est directement attaché à la fenêtre

Application Métro (1)

- Fenêtre avec deux zones : contrôle en haut et affichage en bas

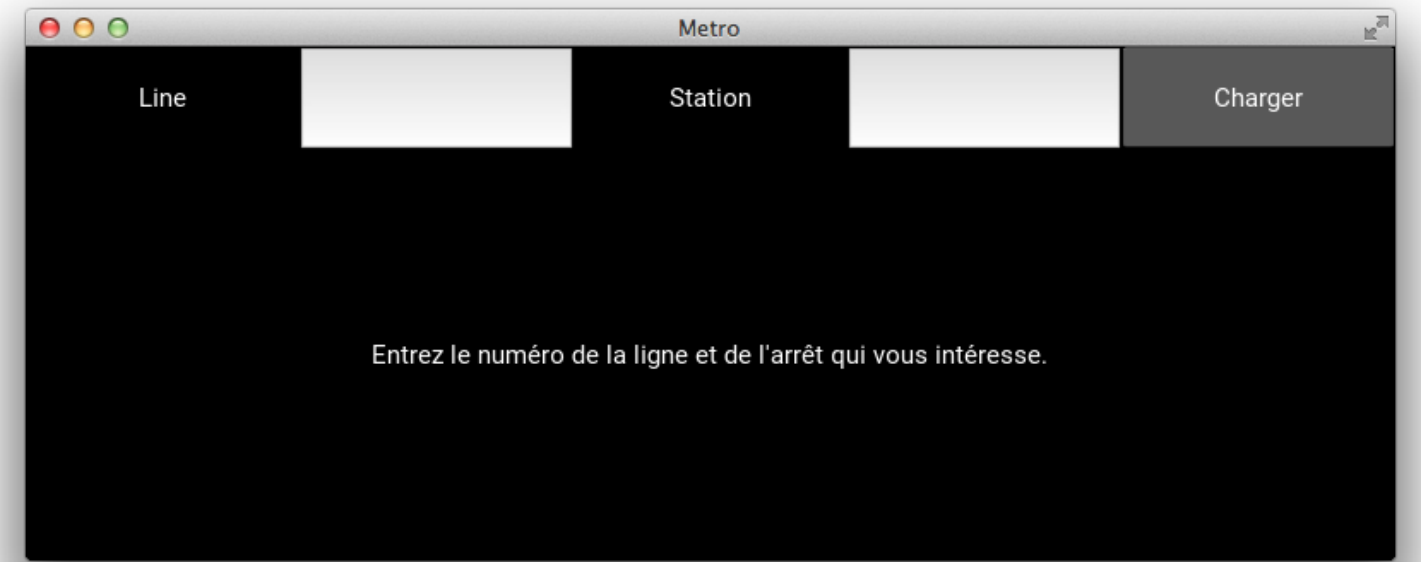
  Les tabulations définissent la hiérarchie des widgets

```
1  BoxLayout:
2      orientation: 'vertical'
3      BoxLayout:
4          orientation: 'horizontal'
5          size_hint: (1,.2)
6          Label:
7              text: 'Line'
8          TextInput:
9              multiline: False
10         Label:
11             text: 'Station'
12         TextInput:
13             multiline: False
14         Button:
15             text: 'Charger'
16     Label:
17         text: "Entrez le numéro de la ligne et de l'arrêt qui vous
           intéresse."
18         size_hint: (1,.8)
```

Application Métro (2)

- Taille de départ de la fenêtre déterminée par configuration

```
1 from kivy.config import Config
2 Config.set('graphics', 'width',  800)
3 Config.set('graphics', 'height', 300)
```



Évènement

- Un évènement est quelque chose qui se produit

  Kivy produit pleins d'évènements divers et variés, en permanence

- Boucle d'évènements lancée par la méthode `run`
  - Parcours permanent des sources d'évènements
  - Relais des évènements vers le code de l'application

- Plusieurs types d'évènements

  Clic, déplacement de souris, clavier, timer, accéléromètre...

Gestionnaire d'évènements

- Un gestionnaire d'évènement répond à leurs occurrences

  Représenté par une fonction ou méthode

- Le gestionnaire reçoit des informations sur l'évènement
  - Source de l'évènement
  - Informations diverses selon le type d'évènement

```
1 class MetroApp(App):
2     def loadschedule(self):
3         print('coucou')
```

Binding

- Attache d'un gestionnaire pour un évènement sur widget cible

  Utilisation d'une propriété dans le fichier KV

- Propriété spécifique par type d'évènement

  Par exemple, `on_press` pour un clic

```
1  #:import App kivy.app.App
2
3  BoxLayout:
4      orientation: 'vertical'
5      BoxLayout:
6          # ...
7          Button:
8              text: 'Charger'
9              on_press: App.get_running_app().loadschedule()
10     Label:
11         text: "Entrez le numéro de la ligne et de l'arrêt qui vous
           intéresse."
12         size_hint: (1,.8)
```

Créer son propre widget

- Création d'un nouveau widget en créant une nouvelle classe

  Le type de la classe est typiquement un widget layout

- Comportement déplacé et modification du fichier KV

```python
class MetroForm(BoxLayout):
    def loadschedule(self):
        print('coucou')

class MetroApp(App):
    pass
```

```
MetroForm:
    orientation: 'vertical'
    BoxLayout:
        # ...
        Button:
            text: 'Charger'
            on_press: root.loadschedule()
    # ...
```

Propriété (1)

- Lien entre le fichier KV et le code Python avec des propriétés

  Objet `ObjectProperty` représente les propriétés dans le widget

- Permet un accès direct au widget et à ses méthodes

  Accessibles comme variables d'instance avec `self`

```
1  class MetroForm(BoxLayout):
2      line_input = ObjectProperty()
3      station_input = ObjectProperty()
4
5      def loadschedule(self):
6          print(self.line_input.text)
7          print(self.station_input.text)
```

Propriété (2)

- Ajout d'un identifiant unique pour les widgets avec `id`

  Uniquement accessible à l'intérieur du fichier KV

- Lien entre les propriétés et les identifiants

```
1  MetroForm:
2      orientation: 'vertical'
3      line_input: line
4      station_input: station
5      BoxLayout:
6          # ...
7          TextInput:
8              id: line
9              multiline: False
10         # ...
11         TextInput:
12             id: station
13             multiline: False
14         # ...
15     # ...
```

Application Métro (3)

- Récupération des horaires des prochains métro avec API STIB

http://m.stib.be/api/getwaitingtimes.php?line=XXX&halt=YYY

```
1  class MetroForm(BoxLayout):
2      line_input = ObjectProperty()
3      station_input = ObjectProperty()
4      result_output = ObjectProperty()
5
6      def loadschedule(self):
7          url = 'http://m.stib.be/api/getwaitingtimes.php?line={}&halt={}'.
           format(self.line_input.text, self.station_input.text)
8          with urllib.request.urlopen(url) as response:
9              self.result_output.text = response.read().decode()
```

Application Métro (4)

- Réponse fournie par l'API au format XML

```
<waitingtimes>
    <stopname>ALMA</stopname>
    <position>
        <latitude>50.85</latitude>
        <longitude>4.453</longitude>
    </position>
    <message mode="1">
        DU 17/3 A 12H AU 18/3 A 14H, SOMMET EUROPEEN. STATION SCHUMAN FERMEE.
        CORRESPONDANCE BUS A MAELBEEK.
    </message>
    <waitingtime>
        <line>1</line>
        <mode>M</mode>
        <minutes>1</minutes>
        <destination>GARE DE L'OUEST</destination>
        <message/>
    </waitingtime>
    <waitingtime>
        <line>1</line>
        <mode>M</mode>
        <minutes>6</minutes>
        <destination>GARE DE L'OUEST</destination>
    <message/>
    </waitingtime>
</waitingtimes>
```

Application Métro (5)

- Extraction des données du XML par expression régulière

Récupération de la direction du métro et du temps d'attente

```
1  with urllib.request.urlopen(url) as response:
2      xml = response.read().decode()
3      pattern = r"<waitingtime>.*?<minutes>([0-9]+)</minutes>.*?<
       destination>([A-Z ']+)</destination>.*?</waitingtime>"
4      p = re.compile(pattern)
5      schedule = ''
6      for m in p.finditer(xml):
7          schedule += 'Vers {} : prochain dans {} minutes\n'.format(m
           .group(2), m.group(1))
8      self.result_output.text = schedule
```

Bonne pratique

- Séparation recommandée entre présentation et logique

  Avoir un fichier KV et un fichier Python

- Layout de la fenêtre et insertion des composants en KV

  En définissant des `id` et des propriétés

- Aspects logiques et gestionnaire d'évènements dans le Python

  Lien avec les composants grâce aux propriétés

Programmation de jeu

Dessin

- Capacité de dessin sophistiquée (statique ou animé)

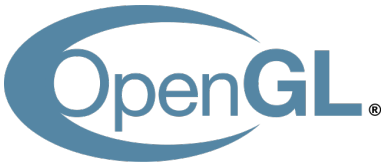
  Exploitation des capacités de OpenGL et SDL

- OpenGL est une API de calcul d'images 2D ou 3D

  Géométrie d'objets et calcul de projection à l'écran

- SDL est une bibliothèque utilisée pour créer des jeux 2D

  Affichage vidéo, audio numérique, périphériques (clavier, souris)...



Canvas (1)

- Utilisation du canvas d'un widget comme zone de dessin

  Zone de dessin vierge obtenue avec des widgets de type layout

- Séquence d'opérations graphiques à réaliser dans le canvas

  Dessin de ligne, ellipse, rectangle ; choix de la couleur...

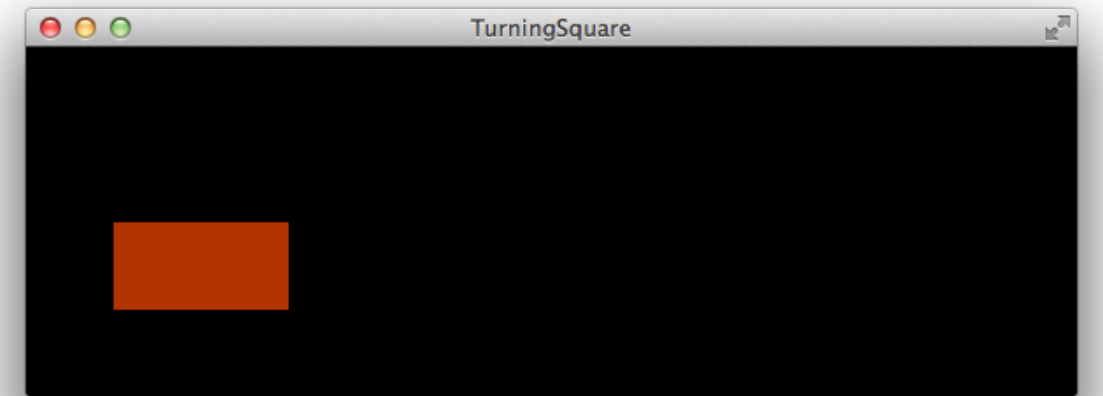
```
1  TurningSquareForm:
2      canvas:
3          Color:
4              rgb: [0.7, 0.2, 0]
5          Rectangle:
6              pos: (50, 50)
7              size: (100, 50)
```

Canvas (2)

- Création de l'application graphique par défaut

  Deux classes vides (application et widget personnalisé)

```
1  class TurningSquareForm(BoxLayout):
2      pass
3
4  class TurningSquareApp(App):
5      pass
6
7  if __name__ == '__main__':
8      TurningSquareApp().run()
```



Transformation (1)

- Trois transformations possibles sur les objets dessinés

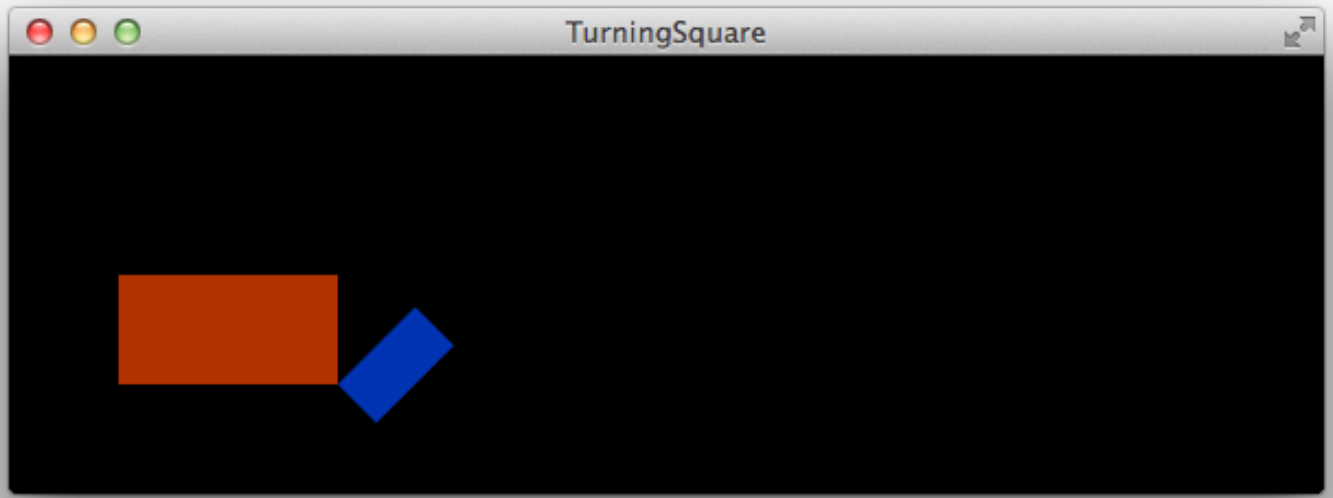
  Rotation, translation et mise à l'échelle

```
TurningSquareForm:
    canvas:
        Color:
            rgb: [0.7, 0.2, 0]
        Rectangle:
            pos: (50, 50)
            size: (100, 50)
        Color:
            rgb: [0, 0.2, 0.7]
        Rotate:
            origin: (150, 50)
            angle: -135
            axis: (0, 0, 1)
        Scale:
            origin: (150, 50)
            x: 0.5
            y: 0.5
        Rectangle:
            pos: (50, 50)
            size: (100, 50)
```

Transformation (2)

- L'ordre d'application des transformations est important

  Tout comme certains paramètres comme l'origine ou l'axe



Composant déplaçable

- Définition d'un nouveau composant qui est déplaçable
  - Le composant `DraggableWidget` est générique
  - Le composant `DraggableRectangle` est un cas particulier

```
1  MovableRectanglesForm:
2      DraggableRectangle:
3      DraggableRectangle:
4          size: (50, 50)
5          color: [0, 0.2, 0.7]
6
7  <DraggableWidget>:
8      size_hint: (None, None)
9
10 <DraggableRectangle>:
11     size: (100, 100)
12     color: [0.7, 0.2, 0]
13     canvas:
14         Color:
15             rgb: self.color
16         Rectangle:
17             pos: (10, 10)
18             size: (self.size[0] - 20, self.size[1] - 20)
```

Application MovableRectangles

- Définition d'un widget générique déplaçable

  Gestionnaires pour les évènements touch (down, move et up)

```
1  class DraggableWidget(RelativeLayout):
2      def __init__(self, **kwargs):
3          self.__selected = None
4          super(DraggableWidget, self).__init__(**kwargs)
5      # ...
6
7  class DraggableRectangle(DraggableWidget):
8      pass
9
10
11 class MovableRectanglesForm(BoxLayout):
12     pass
13
14 class MovableRectanglesApp(App):
15     pass
16
17 if __name__ == '__main__':
18     MovableRectanglesApp().run()
```

Touch down

- Test de collision entre coordonnée de touch et rectangle

  `collide_points` test si un point est dans soi-même

- Deux situations possibles
  - Si collision, marquer le rectangle comme sélectionné
  - Sinon, comportement par défaut

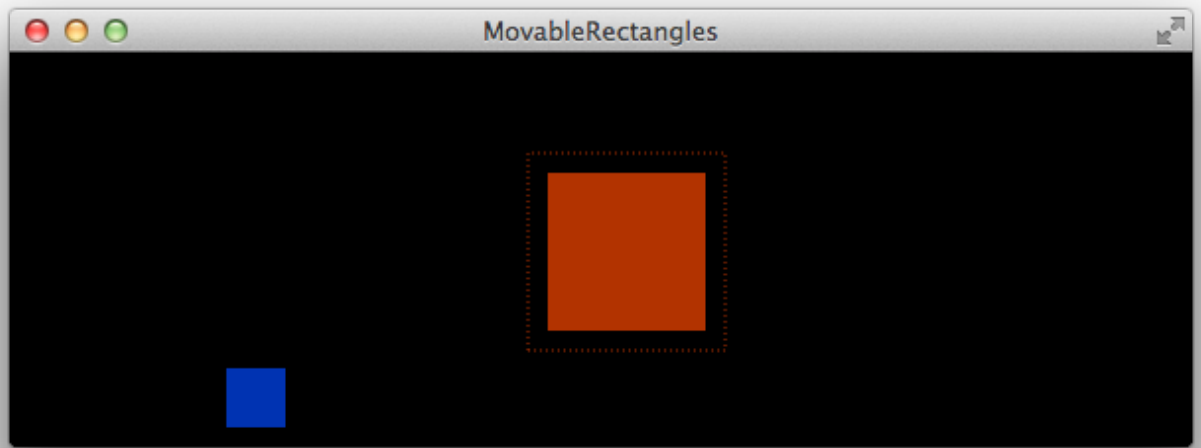
```
1  def on_touch_down(self, touch):
2      if self.collide_point(touch.x, touch.y):
3          self.select()
4          return True
5      return super(DraggableWidget, self).on_touch_down(touch)
```

Touch up

- Déselection du rectangle actuellement sélectionné si existant

  Et suppression du dessin de sélection de l'écran

```
1  def on_touch_up(self, touch):
2      if self.__selected:
3          self.unselect()
4          return True
5      return super(DraggableWidget, self).on_touch_up(touch)
```



Touch move

- Translation du rectangle vers la nouvelle position

  Si cette dernière se trouve dans les limites du parent

```
1  def on_touch_move(self, touch):
2      (x, y) = self.parent.to_parent(touch.x, touch.y)
3      if self.__selected and self.parent.collide_point(x - self.width
       / 2, y - self.height / 2):
4          self.translate(touch.x - self.__ix, touch.y - self.__iy)
5          return True
6      return super(DraggableWidget, self).on_touch_move(touch)
```

Sélection/Désélection

- Sélection d'un objet lors d'un touch down

  Dessin d'un rectangle pointillé autour de l'objet

- Désélection d'un objet lors d'un touch up

  Suppression du rectangle pointillé

```
1  def select(self):
2      if not self.__selected:
3          self.__ix = self.center_x
4          self.__iy = self.center_y
5          with self.canvas:
6              self.__selected = Line(rectangle=(0, 0, self.width,
               self.height), dash_offset=2)
7
8  def unselect(self):
9      if self.__selected:
10         self.canvas.remove(self.__selected)
11         self.__selected = None
```

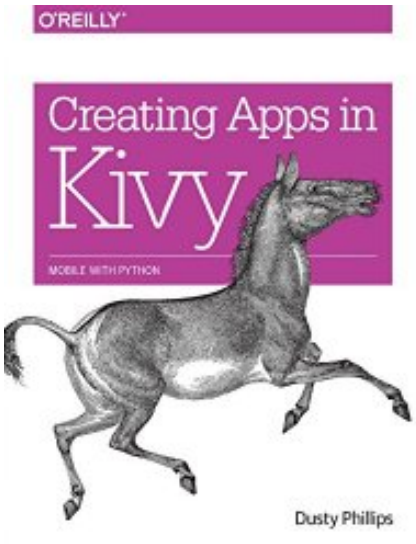
Translation

- Translation de l'objet sélectionné

  Calcul de la nouvelle position avec le shift de touch move

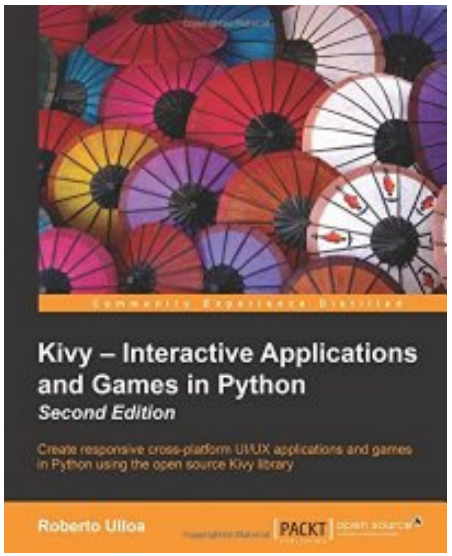
```
1  def translate(self, x, y):
2      self.center_x = self.__ix = self.__ix + x
3      self.center_y = self.__iy = self.__iy + y
```

Livres de référence



ISBN

978-1-491-94667-1



ISBN

978-1-785-28692-6

Crédits

- Photos des livres depuis Amazon
- https://www.flickr.com/photos/mrseb/5367646778
- https://www.flickr.com/photos/genericbrandproductions/4529592666