Pygame Zero Documentation

Version 1.2

Daniel Pope

avr. 21, 2019

Table des matières

Pygame Zero permet de créer des jeux de façon simple et intuitive.

Il est destiné à être utilisé dans l'éducation, afin que les professeurs puissent enseigner les bases de la programmation sans avoir besoin d'expliquer l'"API (Application Programming Interface) de Pygame ni d'écrire une boucle d'événements.

CHAPITRE 1

Les cours

1.1 Introduction à Pygame Zero

1.1.1 Créer une fenêtre

D'abord, crée un fichier vide appelé `intro.py`.

Vérifie que cela démarre et ouvre une fenêtre vide en utilisant la commande :

```
pgzrun intro.py
```

Tout dans Pygame Zero est optionnel : un fichier vide est un programme Pygame Zero valide !

Tu peux quitter le jeu en cliquant sur le bouton de fermeture de la fenêtre ou en pressant les touches `Ctrl-Q` (`-Q` sur Mac). Si le jeu se bloque pour une raison indéterminée, tu peux le terminer en pressant `Ctrl-C` dans la fenêtre de terminal.

1.1.2 Dessiner un arrière-plan

Ensuite, ajoutons une fonction *`draw()`* et définissons les dimensions de la fenêtre. Pygame Zero va appeler cette fonction à chaque fois qu'il a besoin de dessiner l'écran.

Dans `intro.py`, ajoute ce qui suit :

```
1 WIDTH = 300
2 HEIGHT = 300
3
4 def draw():
5     screen.fill((128, 0, 0))
```

Relance `pgzrun intro.py` et l'écran devrait maintenant afficher un carré rouge !

Que fait ce programme ? ?

`WIDTH` et `HEIGHT` contrôlent la largeur et la hauteur de la fenêtre. Le programme définie la taille de la fenêtre à 300 pixels dans chaque direction.

`screen` est une variable pré-définie qui représente la fenêtre d'affichage. Il y a *collection de fonctions pour dessiner des images et des formes*. L'appel à la méthode `screen.fill()` remplit l'écran avec une couleur unie, définie comme un tuple de couleur `(rouge, vert, bleu)`. `(128, 0, 0)` sera un rouge moyennement foncé. Essaye de changer ces valeurs avec des nombres compris entre 0 et 255 et regarde les couleurs que tu peux créer.

Ajoutons maintenant un *sprite* que nous pouvons animer.

1.1.3 Dessiner un *sprite*

Avant de pouvoir dessiner quoique ce soit, nous allons avoir besoin de l'image d'un extraterrestre. Tu peux cliquer avec le bouton de droite sur celle ci-dessous et l'enregistrer (« Enregistrer l'image sous… » ou quelque chose du genre).

(Cette image a un paramètre de transparence, ou « alpha », qui est super pour les jeux ! Mais elle est conçue pour un arrière-plan sombre, il se peut donc que tu ne vois pas le casque de l'extraterrestre avant qu'il soit affiché dans le jeu).

Astuce : Tu peux trouver plein d'images gratuites, incluant celle-ci, sur kenney.nl. Celle-ci vient de Platformer Art Deluxe pack.

Tu dois enregistrer le fichier au bon endroit pour que Pygame Zero puisse le trouver. Crée un répertoire appelé `images` et enregistre l'image à l'intérieur en l'appelant `alien.png`. Les noms de fichiers doivent être en minuscule, sinon Pygame Zero va se plaindre en t'alertant d'un risque potentiel d'incompatibilité sur certaines plateformes.

Si tu as fait tout cela, ton projet devrait ressembler à ceci :

```
.
├── images/
│   └── alien.png
└── intro.py
```

`images/` est un répertoire standard dans lequel Pygame Zero va chercher tes images.

Il y a une classe pré-définie appelée `Actor` que tu peux utiliser pour représenter une image (*sprite*) qui doit être dessinée à l'écran.

Définissons-en une maintenant. Change le fichier `intro.py` afin qu'il contienne :

```
1 alien = Actor('alien')
2 alien.pos = 100, 56
3
4 WIDTH = 500
5 HEIGHT = alien.height + 20
6
7 def draw():
8     screen.clear()
9     alien.draw()
```

Ton extraterrestre devrait maintenant apparaître à l'écran ! En donnant la chaîne `'alien'` à la classe `Actor`, il charge automatiquement l'image correspondante. L'objet obtenu a des attributs comme la position et la taille. Ceci nous permet de définir la hauteur de la fenêtre (`HEIGHT`) basée sur la hauteur de l'extraterrestre.

La méthode `alien.draw()` dessine le *sprite* sur l'écran à sa position courante.

1.1.4 Déplacer l'extraterrestre

Positionnons l'extraterrestre en dehors de l'écran, change la ligne `alien.pos` comme suit :

```
alien.topright = 0, 10
```

Note comment tu peux définir `topright` pour déplacer l'extraterrestre par son coin haut-droit. Si le bord droit de l'extraterrestre est positionné à `0`, alors il sera en dehors de l'écran juste à gauche. Maintenant faisons le bouger. Ajoute les lignes suivantes à la fin du fichier :

```
def update():
    alien.left += 2
    if alien.left > WIDTH:
        alien.right = 0
```

Pygame Zero va appeler la fonction *`update()`* une fois par *frame*. En déplaçant l'extraterrestre par un petit nombre de pixels à chaque *frame*, cela va le faire glisser au travers de l'écran. Une fois qu'il a atteint le bord droit de l'écran, nous le repositionnons à gauche.

Tes fonctions `draw()` et `update()` marchent de façon similaire mais ont été conçues pour des buts différents. La fonction `draw()` dessine l'extraterrestre à sa position courante tandis que la fonction `update()` est utilisée pour le déplacer à l'écran.

1.1.5 Gérer les clics

Faisons en sorte que le jeu fasse quelque chose quand tu cliques sur l'extraterrestre. Pour faire cela, nous avons besoin de définir une fonction appelée *`on_mouse_down()`*. Ajoute ce code au fichier source :

```
1 def on_mouse_down(pos):
2     if alien.collidepoint(pos):
3         print("Eek !")
4     else:
5         print("Tu m'as manqué !")
```

Tu peux démarrer le jeu et essayer de cliquer sur l'extraterrestre et à côté.

Pygame Zero est intelligent dans la façon d'appeler tes fonctions. Si tu ne définis pas ta fonction avec un paramètre `pos`, Pygame Zero va l'appeler sans position. Il y a aussi un paramètre `button` pour `on_mouse_down`. Nous aurions pu écrire :

```
def on_mouse_down():
    print("Tu cliques !")
```

ou :

```
def on_mouse_down(pos, button):
    if button == mouse.LEFT and alien.collidepoint(pos):
        print("Tu m'as touché !")
```

1.1.6 Sons et images

Maintenant faisons apparaître l'extraterrestre blessé. Enregistre ces fichiers :

— alien_hurt.png - enregistre le en tant que `alien_hurt.png` dans le répertoire `images`.

— eep.wav - crée un répertoire appelé `sounds` et enregistre le fichier en tant que `eep.wav` dans ce répertoire.

Ton projet doit maintenant ressembler à ceci :

```
.
├── images/
│   └── alien.png
│   └── alien_hurt.png
├── sounds/
│   └── eep.wav
└── intro.py
```

`sounds/` est un répertoire standard dans lequel Pygame Zero va chercher tes fichiers audio.

Maintenant changeons la fonction `on_mouse_down` pour utiliser ces nouvelles ressources :

```
def on_mouse_down(pos):
    if alien.collidepoint(pos):
        alien.image = 'alien_hurt'
        sounds.eep.play()
```

Maintenant, quand tu cliques sur l'extraterrestre, tu devrais entendre un son et le *sprite* devrait se changer en un extraterrestre pas content.

Il y a cependant un bogue dans le jeu, l'extraterrestre ne redevient jamais joyeux (mais le son se fait entendre à chaque clic). Réparons ça tout de suite.

1.1.7 L'horloge

Si tu es habitué à Python en dehors de la programmation de jeux, tu peux connaître la méthode `time.sleep()` qui attend un certain délai. Tu peux être tenté d'écrire ton programme comme suit :

```
1 def on_mouse_down(pos):
2     if alien.collidepoint(pos):
3         alien.image = 'alien_hurt'
4         sounds.eep.play()
5         time.sleep(1)
6         alien.image = 'alien'
```

Malheureusement, cela n'est pas utilisable dans un jeu. `time.sleep()` bloque toute activité, nous voulons que le jeu continue de fonctionner et de s'animer. En fait nous devons sortir de la fonction `on_mouse_down` et laisser le jeu décider quand réinitialiser l'extraterrestre au cours de son activité normale, tout en continuant d'appeler vos fonctions `draw()` et `update()`.

Ce n'est pas difficile avec Pygame Zero, car il y a la classe pré-définie *`Clock`* qui peut ordonnancer l'appel à des fonctions dans le futur.

D'abord, réorganisons notre programme. Nous pouvons créer des fonctions pour définir l'apparence de l'extraterrestre blessé et aussi le remettre dans son état normal :

```
1 def on_mouse_down(pos):
2     if alien.collidepoint(pos):
3         set_alien_hurt()
4
```

(suite sur la page suivante)

(suite de la page précédente)

```
5
6  def set_alien_hurt():
7      alien.image = 'alien_hurt'
8      sounds.eep.play()
9
10
11 def set_alien_normal():
12     alien.image = 'alien'
```

Cela ne va rien changer pour l'instant. `set_alien_normal()` ne sera pas appelée. Mais changeons `set_alien_hurt()` en utilisant l'horloge afin que `set_alien_normal()` soit appelée un moment plus tard. :

```
def set_alien_hurt():
    alien.image = 'alien_hurt'
    sounds.eep.play()
    clock.schedule_unique(set_alien_normal, 0.5)
```

`clock.schedule_unique()` va faire en sorte que `set_alien_normal()` soit appelée après `0.5` seconde. `schedule_unique()` empêche aussi que l'appel à la même fonction soit ordonnancé plus d'une fois, comme quand par exemple tu cliques très rapidement.

Essaye et tu verras l'extraterrestre revenir à l'état normal après 0.5 seconde. Essaye de cliquer rapidement et vérifie que l'extraterrestre ne revienne dans l'état normal que 0.5 seconde après le dernier clic.

`clock.schedule_unique()` accepte comme intervalle de temps à la fois des entiers et des décimaux. Dans ce tutoriel nous utilisons un nombre décimal pour l'illustrer mais essaye d'utiliser les deux pour voir l'effet que produit chaque valeur.

1.1.8 Résumé

Nous avons vu comment charger et dessiner des *sprites*, jouer des sons, gérer les actions de l'utilisateur et utiliser une horloge pré-définie.

Tu voudrais sûrement améliorer le jeu pour compter les points ou faire bouger l'extraterrestre de façon plus erratique.

Il y a encore plus de fonctionnalités qui font que Pygame Zero est facile à utiliser. Regarde la documentation *des objets pré-définis* pour apprendre le reste de l'API.

1.2 Migrer depuis Scratch

Ce tutoriel compare une réalisation de Flappy Bird écrite en Scratch avec une écrite avec Pygame Zero. Les programmes Scratch et Pygame Zero sont en une certaine mesure très similaires.

La version Pygame Zero peut être trouvée dans le dépôt de Pygame Zero.

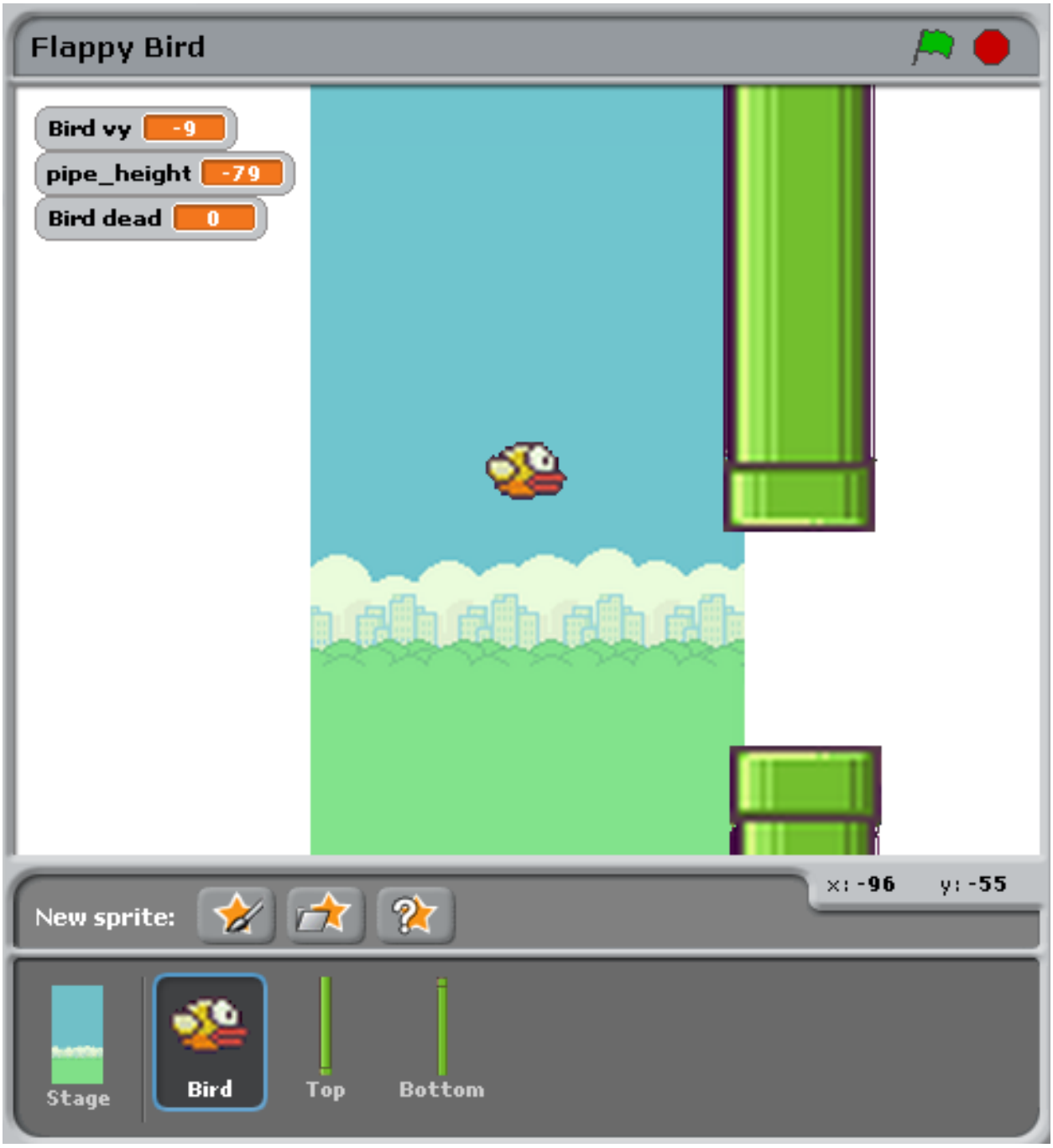
Tu peux aussi télécharger la version Scratch depuis le même dépôt.

La version Pygame Zero inclut la gestion du score, qui est omise dans les exemples de code de cette page afin de simplifier la comparaison.

Le code Python montré ci-dessous est réorganisé pour plus de clarté au travers des exemples.

1.2.1 La scène

Voici comment la scène est organisée dans le programme Scratch :



Il y a juste trois objets à part l'arrière plan : l'oiseau et les tuyaux du haut et du bas.

Cela correspond au programme Pygame Zero, en définissant ces objets à l'aide d'acteurs (`Actors`) :

```
bird = Actor('bird1', (75, 200))
pipe_top = Actor('top', anchor=('left', 'bottom'))
pipe_bottom = Actor('bottom', anchor=('left', 'top'))
```

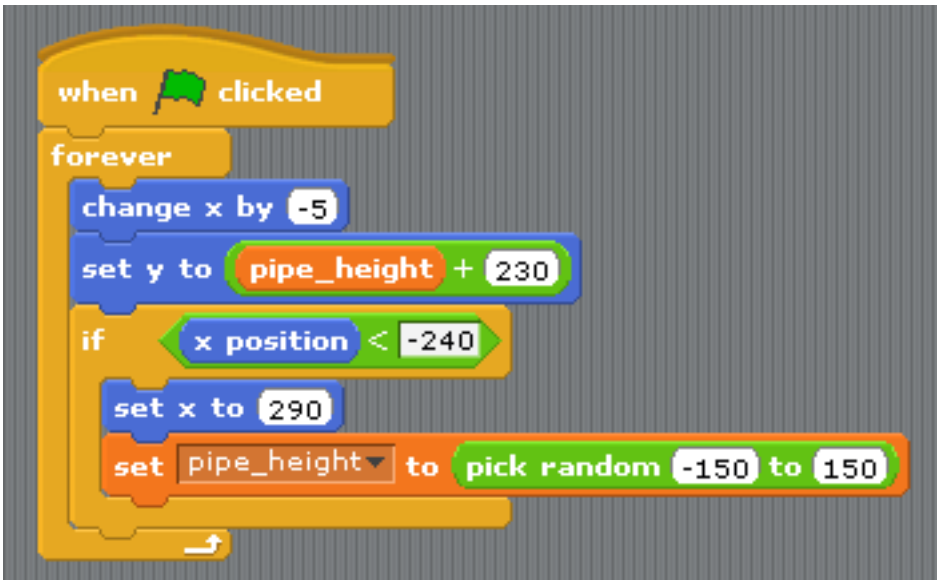
Avec Pygame Zero, nous devons nous assurer de dessiner ces objets. En général cela donne un peu plus de flexibilité sur la façon de dessiner la scène :

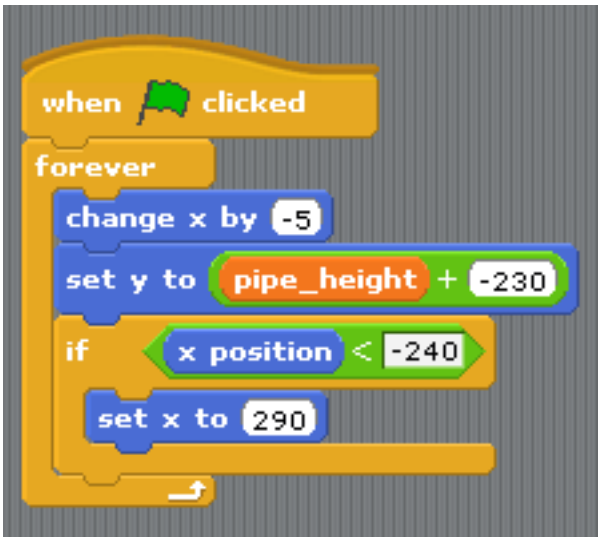
```
def draw():
    screen.blit('background', (0, 0))
    pipe_top.draw()
    pipe_bottom.draw()
    bird.draw()
```

1.2.2 Le mouvement des tuyaux

Les tuyaux se déplacent à une vitesse constante indépendante de l'oiseau. Quand ils disparaissent du bord gauche de l'écran, ils réapparaissent à droite et leur position verticale change aléatoirement.

En Scratch, cela peut être obtenu en créant deux scripts différents pour les tuyaux du haut et du bas.





Pour résumer ce qui se passe ici :

— La condition `x position < -240` est vraie quand un tuyau disparaît à gauche de l'écran et c'est ce qui déclenche la remise à zéro des tuyaux.
— La variable `pipe_height` est utilisée pour coordonner les deux tuyaux. Parce que l'espace entre eux doit rester le même, nous ne pouvons pas choisir les deux hauteurs aléatoirement. Donc un script contient la logique nécessaire et pas l'autre.
— La ligne `set y position to pipe height +/- 230` place un tuyaux au-dessus de `pipe_height` et l'autre en dessous `pipe_height`.

Ce code devient bien plus simple avec Pygame Zero. Nous pouvons écrire une seule fonction qui met à jour les deux tuyaux. En fait, je l'ai découpée d'une façon différente pour montrer clairement que les traitements se font pour les deux tuyaux en même temps :

```
import random

WIDTH = 400
HEIGHT = 708
GAP = 130
SPEED = 3

def reset_pipes():
    pipe_gap_y = random.randint(200, HEIGHT - 200)
    pipe_top.pos = (WIDTH, pipe_gap_y - GAP // 2)
    pipe_bottom.pos = (WIDTH, pipe_gap_y + GAP // 2)

def update_pipes():
    pipe_top.left -= SPEED
    pipe_bottom.left -= SPEED
    if pipe_top.right < 0:
        reset_pipes()
```

Une petite différence ici est que j'extrais les valeurs que je compte réutiliser sous la forme de « constantes », écrite en MAJUSCULE. Cela me permet de les changer à un seul endroit lorsque je veux ajuster le jeu. Par exemple dans le code au dessus, je pourrais élargir ou rétrécir l'espacement entre les deux tuyaux seulement en changeant `GAP` (écart).

La plus grosse différence est qu'il n'y a pas de `répéter indéfiniment` dans le code Python. C'est la grosse différence entre Scratch et la plupart des languages de programmation textuel : vous devez mettre à jour le jeu d'un pas d'animation et ensuite retourner. Retourner donne à Pygame Zero la chance de faire des chose comme traiter les entrées ou redessiner l'écran. Boucler indéfiniment et le jeu resterait juste bloqué, donc toute boucle doit se finir rapidement.

Pygame Zero appelle la fonction `update()` quand il veux que tu mettes à jour l'animation d'un pas, donc nous avons juste besoin d'appeler `update_pipes()` :

```
def update():
   update_pipes()
```

1.2.3 L'oiseau

La méthode décrite ci-dessus sur comment traduire Scratch en programme Python s'applique aussi pour le comportement de l'oiseau. Regardons d'abord le code Python cette fois.

Le code de mise à jour de l'oiseau est organisé en une fonction appelée `update_bird()`. La première chose que cette fonction contient est le code de gestion des déplacement de l'oiseau selon la gravité :

```
GRAVITY = 0.3

# Initial state of the bird
```

(suite sur la page suivante)

(suite de la page précédente)

```
bird.dead = False
bird.vy = 0

def update_bird():
    uy = bird.vy
    bird.vy += GRAVITY
    bird.y += bird.vy
    bird.x = 75
```

Voici une formule simple de gravité :

— Gravité signifie **accélération constante vers le bas**.
— Accélération est une variation de **vitesse**.
— Vitesse est une variation de **position**.

Pour représenter cela, nous avons besoin d'une variable `bird.vy` qui est la vitesse de l'oiseau dans la direction `y`. C'est une nouvelle variable que nous définissons, pas quelque chose que Pygame Zero nous fournit.

— Gravité signifie accélération constante vers le bas : `GRAVITY` est supérieure à 0.
— L'accélération est une variation de vitesse : `GRAVITY` est ajoutée à `bird.vy`
— La vitesse est une variation de position : `bird.vy` est ajoutée à `bird.y`

Note que l'oiseau ne bouge pas horizontalement ! Sa position `x` reste à 75 tout au long du jeu. Nous simulons le mouvement horizontal en déplaçant les tuyaux vers lui. Ça donne l'impression d'une caméra se déplaçant en suivant l'oiseau. Donc il n'y a pas besoin d'une variable `vx` dans le jeu.

La section suivante fait battre les ailes de l'oiseau :

```
if not bird.dead:
    if bird.vy < -3:
        bird.image = 'bird2'
    else:
        bird.image = 'bird1'
```

Cela vérifie que l'oiseau se déplace vers le haut ou le bas. Nous affichons l'image `bird2` s'il se déplace vers le haut et sinon l'image `bird1`. (-3 a été choisi après plusieurs essais pour que cela ait l'air convainquant).

La section suivante vérifie si l'oiseau a percuté un tuyau :

```
if bird.colliderect(pipe_top) or bird.colliderect(pipe_bottom):
    bird.dead = True
    bird.image = 'birddead'
```

Si c'est le cas, nous positionnons `bird.dead` à `True`. C'est une **valeur booléenne** signifiant qu'elle vaut soit vrai (`True`) soit faux (`False`). Nous pouvons utiliser cela pour facilement vérifier si l'oiseau est en vie. Si ce n'est pas le cas, il ne répondra plus aux commandes du joueur.

Et la dernière section vérifie si l'oiseau est arrivé en dehors, en bas ou en haut, de l'écran du jeu. Si c'est le cas, il réinitialise l'oiseau :

```
if not 0 < bird.y < 720:
    bird.y = 200
    bird.dead = False
    bird.vy = 0
    reset_pipes()
```

Que fait `reset_pipes()` ici ? Comme j'ai organisé le code des tuyaux en une fonction séparée, je peux juste l'appeler à tout moment lorsque je veux réinitialiser les tuyaux. Dans ce cas, cela fait un jeu plus intéressant car il laisse une chance au joueur de réagir quand l'oiseau est réinitialisé à sa position d'origine.

Encore une fois, cela doit être appelé à chaque *frame*, donc nous l'ajoutons à `update()` :

```
def update():
    update_walls()
    update_bird()
```

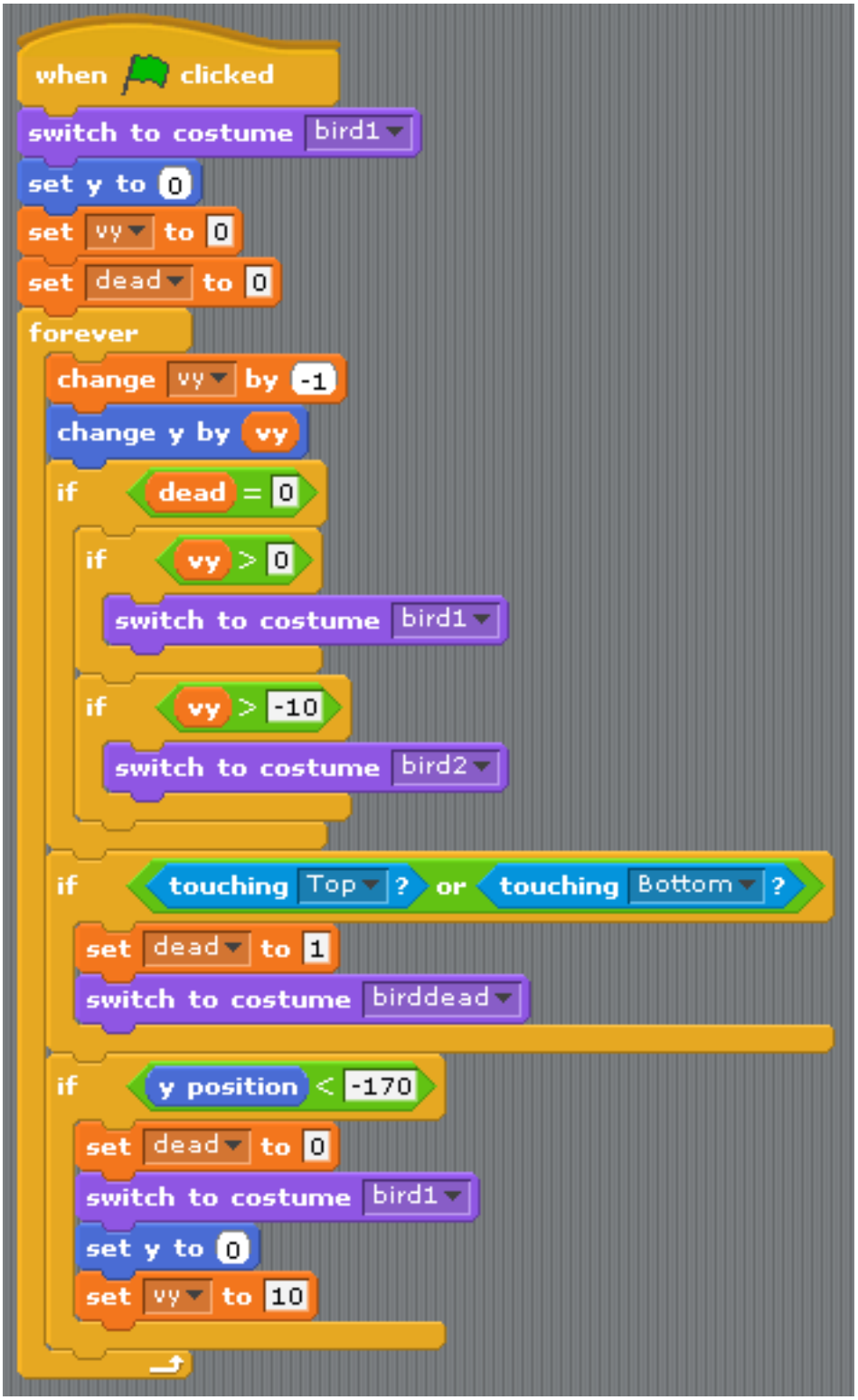
La dernière partie du code de l'oiseau est celle qui permet de répondre aux commandes du joueur. Quand nous pressons une touche, l'oiseau bat des ailes vers le haut. Pygame Zero va appeler la fonction `on_key_down()` - si vous l'avez définie - dès qu'une touche est pressée :

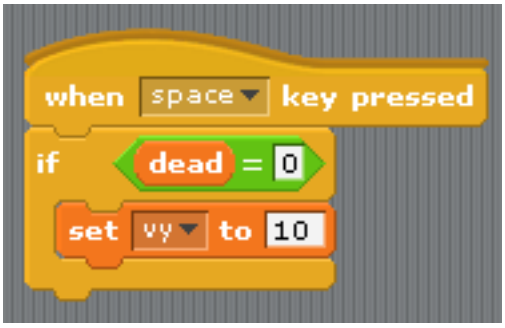
```
FLAP_VELOCITY = -6.5

def on_key_down():
    if not bird.dead:
        bird.vy = FLAP_VELOCITY
```

Ici, si l'oiseau n'est pas mort, nous définissons `vy` avec un nombre négatif : dans Pygame Zero cela signifie qu'il commence à se déplacer vers le haut.

Tu devrais être capable de trouver plein de similitudes entre le code Python et ce code Scratch :


when clicked
switch to costume bird1
set y to 0
set vy to 0
set dead to 0
forever
change vy by -1
change y by vy
if dead = 0
if vy > 0
switch to costume bird1
if vy > -10
switch to costume bird2
if touching Top ? or touching Bottom ?
set dead to 1
switch to costume birddead
if y position < -170
set dead to 0
switch to costume bird1
set y to 0
set vy to 10



Les grosses différence entre Scratch et Pygame Zero sont les suivantes :

— Tu ne peux pas boucler indéfiniment avec Pygame Zero, juste mettre à jour pour un pas et retourner.
— Les coordonnées sont différentes. Dans Pygame Zero, le coin en haut à gauche de l'écran est `x = 0, y = 0`. La direction `x` va de gauche à droite comme avant, mais `y` va vers le bas de l'écran ! C'est pour cela que `GRAVITY` est un nombre positif et `FLAP_VELOCITY` est un nombre négatif dans le code Python.
— `bird.dead` est un booléen, donc nous pouvons écrire du code comme `if not bird.dead` au lieu de `dead = 0` en Scratch.

1.2.4 Résumé

Beaucoup des concepts disponibles dans Scratch se traduisent directement dans Pygame Zero.

Voici quelques points de comparaison :

Dans Scratch	Dans Pygame Zero
`ajouter 1 à y` (up)	`bird.y -= 1`
`ajouter -1 à y` (down)	`bird.y += 1`
`basculer sur le costume [nom]`	`bird.image = 'name'`
`si dead = 0`	`if not bird.dead:`
`mettre dead to 0`	`bird.dead = False`
`si touche le Top?`	`if bird.colliderect(pipe_top)`
`Quand le drapeau est cliqué...` `répéter indéfiniment`	Met le code dans la fonction `update()`.
`Quand la touche [X] est pressée`	`def on_key_down():`
`nombre aléatoire entre a et b`	`import random` pour charger le module `random`, puis `random.randint(a, b)`
(0, 0) est le centre de la scène	(0, 0) est le coin en haut à gauche de la fenêtre

Dans certains cas, le code est plus simple en Python car il peut être organisé de façon à être plus lisible.

La puissance des acteurs de Pygame Zero fait aussi que la manipulation des coordonnées est plus simple. Nous pouvons utiliser les positions ancre (`anchor`) pour positionner les tuyaux et nous avons été capables de voir si un tuyau était en dehors de l'écran en vérifiant `pipe_top.right < 0` plutôt que `if x position < -240`.

CHAPITRE 2

Référence

2.1 Les *hooks* d'évènements

Pygame Zero va automatiquement détecter et appeler les *hooks* (littéralement « crochet » ou « hameçon ») d'évènements que tu définis. Cette approche vous épargne d'écrire vous même la mécanique de boucle d'évènements.

2.1.1 Les *hooks* d'une boucle de jeu

Une boucle de jeu typique ressemble à peu près à ça :

```
while game_has_not_ended():
    process_input()
    update()
    draw()
```

Le traitement des entrées est un peu plus complexe, mais Pygame Zero te permet facilement d'écrire les fonctions `update()` et `draw()` dans ton programme de jeu.

draw()

Elle est appelée par Pygame Zero quand il a besoin de redessiner l'écran de jeu.

`draw()` ne prend aucun argument.

Pygame Zero tente de décider quand l'écran de jeu a besoin d'être redessiné afin d'éviter de le faire si rien n'a changé. À chaque pas de la boucle de jeu, il va dessiner l'écran dans les situations suivantes :

— Si tu as défini une fonction `update()` (voir ci-dessous).
— Si un événement d'horloge est émis.
— Si un événement d'entrée a été déclenché.

Une façon de se faire piéger est lorsque tu tentes de modifier ou animer quelques chose dans la fonction `draw`. Par exemple, ce code est faux : l'extraterrestre n'est pas garanti de bouger tout le long de l'écran :

```
def draw():
    alien.left += 1
    alien.draw()
```

Le code correct utilise la fonction `update()` pour modifier ou animer les objets et `draw` simplement pour dessiner à l'écran :

```
def draw():
    alien.draw()

def update():
    alien.left += 1
```

update(*) ou update(dt*)

Elle est appelée par Pygame Zero pour faire progresser ta logique de jeu. Elle va être appelée continuellement 60 fois par seconde.

Il y a deux façons d'écrire une fonction `update`.

Dans les jeux simples, tu peux supposer qu'un court laps de temps (une fraction de seconde) est passé entre chaque appel à `update()`. Peut être que la durée de ce laps n'a pas d'importance : tu peux juste bouger les objets d'un nombre constant de pixels par images (ou les accélérer par une constante, etc.)

Une approche plus sophistiquée est de baser le mouvement et les calculs physiques sur la quantité réelle de temps écoulée entre deux appels. Cela produit des animations plus fluides, mais les calculs induits peuvent être plus compliqués et tu dois être attentif à ne pas créer de comportements non-prédictibles lorsque le laps de temps écoulé devient plus grand.

Pour utiliser l'approche basée sur le temps, tu peux changer la fonction `update` en ajoutant un paramètre unique. Si ta fonction `update` prend un argument, Pygame Zero va lui passer le temps écoulé en secondes. tu peux l'utiliser pour adapter le calcul des mouvements.

2.1.2 Les *hooks* de gestion d'événements

Comme pour les *hooks* de la boucle de jeu, ton programme Pygame Zero peut répondre aux événements d'entrée en définissant des fonctions avec des noms spécifiques.

Comme pour la fonction `update()`, Pygame Zero va inspecter ta fonction de gestion d'événements pour déterminer comment l'appeler. Tu n'as donc pas besoin d'avoir d'arguments à ta fonction. Par exemple, Pygame Zero va être capable d'appeler toutes ces variantes de la fonction `on_mouse_down` :

```
def on_mouse_down():
    print("Mouse button clicked")

def on_mouse_down(pos):
    print("Mouse button clicked at", pos)

def on_mouse_down(button):
    print("Mouse button", button, "clicked")

def on_mouse_down(pos, button):
    print("Mouse button", button, "clicked at", pos)
```

Il le fait en regardant le nom des paramètres, ils doivent donc être épelés exactement comme ci-dessus. Chaque *hook* d'événements a un ensemble différent de paramètres que tu peux utiliser comme décrit ci-dessous.

on_mouse_down(*[pos][, button]*)

Appelée lorsqu'un bouton de la souris est pressé.

Paramètres

— **pos** – un tuple (x, y) donnant la position du pointeur de souris lorsque le bouton a été pressé.

— **button** – une valeur de l'énumération *mouse* indiquant quel bouton a été pressé.

on_mouse_up(*[pos][, button]*)

Appelée lorsqu'un un bouton de la souris est relâché.

Paramètres

- **pos** – un tuple (x, y) donnant la position du pointeur de souris lorsque le bouton a été relâché.
- **button** – une valeur de l'énumération *mouse* indiquant quel bouton a été relâché.

on_mouse_move([*pos*][, *rel*][, *buttons*])

Appelée lorsque la souris est bougée.

Paramètres

- **pos** – un tuple (x, y) donnant la position du pointeur de souris à laquelle la souris a bougé.
- **rel** – un tuple (delta_x, delta_y) représentant le déplacement relatif du pointeur de souris.
- **buttons** – un ensemble de valeurs de l'énumération *mouse* indiquant quels boutons étaient pressés durant le déplacement.

Pour gérer le glissement de la souris (mouvement avec bouton pressé), utilise un code tel que le suivant :

```
def on_mouse_move(rel, buttons):
    if mouse.LEFT in buttons:
        # la souris a été glissée, faire quelque chose avec `rel`
        ...
```

on_key_down([*key*][, *mod*][, *unicode*])

Appelée lorsqu'une touche est pressée.

Paramètres

- **key** – un entier indiquant quelle touche a été pressée (voir *ci-dessous*).
- **unicode** – lorsque cela a du sens, le caractère qui a été entré. Toutes les touches ne produisent pas un caractère affichable - beaucoup sont des caractères de contrôle. Dans le cas où la touche ne correspond pas à un caractère Unicode, cela sera la chaîne vide.
- **mod** – un masque de bits représentant les touches mortes également pressées.

on_key_up([*key*][, *mod*])

Appelée lorsqu'une touche est relâchée.

Paramètres

- **key** – un entier indiquant quelle touche a été relâchée (voir *ci-dessous*).
- **mod** – un masque de bits représentant les touches mortes pressées.

on_music_end()

Appelée lorsque une *piste de musique* se termine.

Note qu'elle ne sera pas appelée si la piste audio est configurée pour boucler.

Boutons et touches

Les objets pré-définis `mouse` et `keys` peuvent être utilisés pour déterminer quels boutons ou touches étaient pressés dans les événements ci-dessus.

Note que les événements de la molette de la souris apparaissent comme des boutons pressés avec les constantes `WHEEL_UP`/`WHEEL_DOWN`.

class mouse

Une énumération pré-défini de boutons qui peut être reçue par les *hooks* `on_mouse_*`.

LEFT

MIDDLE

RIGHT

WHEEL_UP

WHEEL_DOWN

class keys

Une énumération pré-défini de touches qui peut être reçue par les *hooks* `on_key_*`.

BACKSPACE
TAB
CLEAR
RETURN
PAUSE
ESCAPE
SPACE
EXCLAIM
QUOTEDBL
HASH
DOLLAR
AMPERSAND
QUOTE
LEFTPAREN
RIGHTPAREN
ASTERISK
PLUS
COMMA
MINUS
PERIOD
SLASH
K_0
K_1
K_2
K_3
K_4
K_5
K_6
K_7
K_8
K_9
COLON
SEMICOLON
LESS
EQUALS
GREATER
QUESTION
AT
LEFTBRACKET
BACKSLASH
RIGHTBRACKET
CARET
UNDERSCORE
BACKQUOTE
A
B
C

D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
DELETE
KP0
KP1
KP2
KP3
KP4
KP5
KP6
KP7
KP8
KP9
KP_PERIOD
KP_DIVIDE
KP_MULTIPLY
KP_MINUS
KP_PLUS
KP_ENTER
KP_EQUALS
UP
DOWN
RIGHT
LEFT
INSERT
HOME

END
PAGEUP
PAGEDOWN
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12
F13
F14
F15
NUMLOCK
CAPSLOCK
SCROLLOCK
RSHIFT
LSHIFT
RCTRL
LCTRL
RALT
LALT
RMETA
LMETA
LSUPER
RSUPER
MODE
HELP
PRINT
SYSREQ
BREAK
MENU
POWER
EURO
LAST

De plus, tu peux accéder a un ensemble de constantes représentant les touches mortes :

class keymods
Constantes représentant les touches mortes qui ont été pressées durant les événements `on_key_up`/`on_key_down`.
LSHIFT
RSHIFT
SHIFT

LCTRL
RCTRL
CTRL
LALT
RALT
ALT
LMETA
RMETA
META
NUM
CAPS
MODE

2.2 Objets pré-définis

Pygame Zero fournit des objets pré-définis utiles pour t'aider à construire des jeux facilement.

2.2.1 Screen

Text Formatting

The *Screen*'s `draw.text()` method has a very rich set of methods for position and formatting of text. Some examples :

```
screen.draw.text("Text color", (50, 30), color="orange")
screen.draw.text("Font name and size", (20, 100), fontname="Boogaloo", fontsize=60)
screen.draw.text("Positioned text", topright=(840, 20))
screen.draw.text("Allow me to demonstrate wrapped text.", (90, 210), width=180,␣
↪lineheight=1.5)
screen.draw.text("Outlined text", (400, 70), owidth=1.5, ocolor=(255,255,0), color=(0,
↪0,0))
screen.draw.text("Drop shadow", (640, 110), shadow=(2,2), scolor="#202020")
screen.draw.text("Color gradient", (540, 170), color="red", gcolor="purple")
screen.draw.text("Transparency", (700, 240), alpha=0.1)
screen.draw.text("Vertical text", midleft=(40, 440), angle=90)
screen.draw.text("All together now:\nCombining the above options",
    midbottom=(427,460), width=360, fontname="Boogaloo", fontsize=48,
    color="#AAFF00", gcolor="#66AA00", owidth=1.5, ocolor="black", alpha=0.8)
```

In its simplest usage, `screen.draw.text` requires the string you want to draw, and the position. You can either do this by passing coordinates as the second argument (which is the top left of where the text will appear), or use the positioning keyword arguments (described later) :

```
screen.draw.text("hello world", (20, 100))
```

`screen.draw.text` takes many optional keyword arguments, described below.

Font name and size

Fonts are loaded from a directory named `fonts`, in a similar way to the handling of images and sounds. Fonts must be in `.ttf` format. For example :

```
screen.draw.text("hello world", (100, 100), fontname="Viga", fontsize=32)
```

Keyword arguments :

— `fontname` : filename of the font to draw. By default, use the system font.
— `fontsize` : size of the font to use, in pixels. Defaults to `24`.
— `antialias` : whether to render with antialiasing. Defaults to `True`.

Color and background color

```
screen.draw.text("hello world", (100, 100), color=(200, 200, 200), background="gray")
```

Keyword arguments :

— `color` : foreground color to use. Defaults to `white`.
— `background` : background color to use. Defaults to `None`.

`color` (as well as `background`, `ocolor`, `scolor`, and `gcolor`) can be an (r, g, b) sequence such as `(255,127, 0)`, a `pygame.Color` object, a color name such as `"orange"`, an HTML hex color string such as `"#FF7F00"`, or a string representing a hex color number such as `"0xFF7F00"`.

`background` can also be `None`, in which case the background is transparent. Unlike `pygame.font.Font.render`, it's generally not more efficient to set a background color when calling `screen.draw.text`. So only specify a background color if you actually want one.

Colors with alpha transparency are not supported (except for the special case of invisible text with outlines or drop shadows - see below). See the `alpha` keyword argument for transparency.

Positioning

```
screen.draw.text("hello world", centery=50, right=300)
screen.draw.text("hello world", midtop=(400, 0))
```

Keyword arguments :

```
top left bottom right
topleft bottomleft topright bottomright
midtop midleft midbottom midright
center centerx centery
```

Positioning keyword arguments behave like the corresponding properties of `pygame.Rect`. Either specify two arguments, corresponding to the horizontal and vertical positions of the box, or a single argument that specifies both.

If the position is overspecified (e.g. both `left` and `right` are given), then extra specifications will be (arbitrarily but deterministically) discarded. For constrained text, see the section on `screen.draw.textbox` below.

Word wrap

```
screen.draw.text("splitting\nlines", (100, 100))
screen.draw.text("splitting lines", (100, 100), width=60)
```

Keyword arguments :

— `width` : maximum width of the text to draw, in pixels. Defaults to `None`.
— `widthem` : maximum width of the text to draw, in font-based em units. Defaults to `None`.
— `lineheight` : vertical spacing between lines, in units of the font's default line height. Defaults to `1.0`.

`screen.draw.text` will always wrap lines at newline (`\n`) characters. If `width` or `widthem` is set, it will also try to wrap lines in order to keep each line shorter than the given width. The text is not guaranteed to be within the given width, because wrapping only occurs at space characters, so if a single word is too long to fit on a line, it will not be broken up. Outline and drop shadow are also not accounted for, so they may extend beyond the given width.

You can prevent wrapping on a particular space with non-breaking space characters (`\u00A0`).

Text alignment

```
screen.draw.text("hello\nworld", bottomright=(500, 400), align="left")
```

Keyword argument :

— `align` : horizontal positioning of lines with respect to each other. Defaults to `None`.

`align` determines how lines are positioned horizontally with respect to each other, when more than one line is drawn. Valid values for `align` are the strings `"left"`, `"center"`, or `"right"`, a numerical value between `0.0` (for left alignment) and `1.0` (for right alignment), or `None`.

If `align` is `None`, the alignment is determined based on other arguments, in a way that should be what you want most of the time. It depends on any positioning arguments (`topleft`, `centerx`, etc.), `anchor`, and finally defaults to `"left"`. I suggest you generally trust the default alignment, and only specify `align` if something doesn't look right.

Outline

```
screen.draw.text("hello world", (100, 100), owidth=1, ocolor="blue")
```

Keyword arguments :

— `owidth` : outline thickness, in outline units. Defaults to `None`.

— `ocolor` : outline color. Defaults to `"black"`.

The text will be outlined if `owidth` is specified. The outlining is a crude manual method, and will probably look bad at large sizes. The units of `owidth` are chosen so that `1.0` is a good typical value for outlines. Specifically, they're the font size divided by 24.

As a special case, setting `color` to a transparent value (e.g. `(0,0,0,0)`) while using outilnes will cause the text to be invisible, giving a hollow outline. (This feature is not compatible with `gcolor`.)

Valid values for `ocolor` are the same as for `color`.

Drop shadow

```
screen.draw.text("hello world", (100, 100), shadow=(1.0,1.0), scolor="blue")
```

Keyword arguments :

— `shadow` : (x,y) values representing the drop shadow offset, in shadow units. Defaults to `None`.

— `scolor` : drop shadow color. Defaults to `"black"`.

The text will have a drop shadow if `shadow` is specified. It must be set to a 2-element sequence representing the x and y offsets of the drop shadow, which can be positive, negative, or 0. For example, `shadow=(1.0,1.0)` corresponds to a shadow down and to the right of the text. `shadow=(0,-1.2)` corresponds to a shadow higher than the text.

The units of `shadow` are chosen so that `1.0` is a good typical value for the offset. Specifically, they're the font size divided by 18.

As a special case, setting `color` to a transparent value (e.g. `(0,0,0,0)`) while using drop shadow will cause the text to be invisible, giving a hollow shadow. (This feature is not compatible with `gcolor`.)

Valid values for `scolor` are the same as for `color`.

Gradient color

```
screen.draw.text("hello world", (100, 100), color="black", gcolor="green")
```

Keyword argument :
— `gcolor` : Lower gradient stop color. Defaults to `None`.
Specify `gcolor` to color the text with a vertical color gradient. The text's color will be `color` at the top and `gcolor` at the bottom. Positioning of the gradient stops and orientation of the gradient are hard coded and cannot be specified.

Requries `pygame.surfarray` module, which uses numpy or Numeric library.

Alpha transparency

```
screen.draw.text("hello world", (100, 100), alpha=0.5)
```

Keyword argument :
— `alpha` : alpha transparency value, between 0 and 1. Defaults to `1.0`.
In order to maximize reuse of cached transparent surfaces, the value of `alpha` is rounded.

Requires `pygame.surfarray` module, which uses numpy or Numeric library.

Anchored positioning

```
screen.draw.text("hello world", (100, 100), anchor=(0.3,0.7))
```

Keyword argument :
— `anchor` : a length-2 sequence of horizontal and vertical anchor fractions. Defaults to `(0.0, 0.0)`.
`anchor` specifies how the text is anchored to the given position, when no positioning keyword arguments are passed. The two values in `anchor` can take arbitrary values between `0.0` and `1.0`. An `anchor` value of `(0,0)`, the default, means that the given position is the top left of the text. A value of `(1,1)` means the given position is the bottom right of the text.

Rotation

```
screen.draw.text("hello world", (100, 100), angle=10)
```

Keyword argument :
— `angle` : counterclockwise rotation angle in degrees. Defaults to `0`.
Positioning of rotated surfaces is tricky. When drawing rotated text, the anchor point, the position you actually specify, remains fixed, and the text rotates around it. For instance, if you specify the top left of the text to be at `(100, 100)` with an angle of `90`, then the Surface will actually be drawn so that its bottom left is at `(100, 100)`.

If you find that confusing, try specifying the center. If you anchor the text at the center, then the center will remain fixed, no matter how you rotate it.

In order to maximize reuse of cached rotated surfaces, the value of `angle` is rounded to the nearest multiple of 3 degrees.

Constrained text

```
screen.draw.textbox("hello world", (100, 100, 200, 50))
```

`screen.draw.textbox` requires two arguments : the text to be drawn, and a `pygame.Rect` or a `Rect`-like object to stay within. The font size will be chosen to be as large as possible while staying within the box. Other than `fontsize` and positional arguments, you can pass all the same keyword arguments to `screen.draw.textbox` as to `screen.draw.text`.

L'objet `screen` représente l'écran de ton jeu.

C'est une fine couche autour d'une surface Pygame qui te permet facilement de dessiner des images à l'écran.

class Screen

surface
La surface Pygame brute qui représente l'espace mémoire de l'écran. Tu peux l'utiliser pour les opérations graphiques avancées.

clear()
Réinitialise l'écran en noir.

fill(*(rouge, vert, bleu)*)
Rempli l'écran d'une couleur pleine.

blit(*image*, *(gauche, haut)*)
Dessine l'image sur l'écran à la position donnée.

`blit()` accepte soit une surface ou une chaîne comme paramètre `image`. Si `image` est une chaîne alors l'image au nom correspondant sera chargée depuis le répertoire `images/`.

`draw.`**line**(*début*, *fin*, *(r, v, b)*)
Dessine une ligne de `début` à `fin`.

`draw.`**circle**(*pos*, *rayon*, *(r, v, b)*)
Dessine un cercle.

`draw.`**filled_circle**(*pos*, *rayon*, *(r, v, b)*)
Dessine un disque.

`draw.`**rect**(*rect*, *(r, v, b)*)
Dessine le contour un rectangle.

Prend un *Rect* en paramètre.

`draw.`**filled_rect**(*rect*, *(r, v, b)*)
Dessine un rectangle plein.

`draw.`**text**(*texte*[, *pos*], ***kwargs*)
Dessine du texte.

Il existe une API extrêmement riche pour positionner et formater du texte, regarder *Text Formatting* pour plus de détails.

`draw.`**textbox**(*texte*, *rect*, ***kwargs*)
Dessine du texte dimensionné pour remplir le *Rect* donné.

Il existe une API extrêmement riche pour positionner et formater du texte, regarder *Text Formatting* pour plus de détails.

2.2.2 Rect

La classe Pygame Rect est disponible en tant que classe prédéfinie. Elle peut être utilisée de nombreuses façons, de la détection de clics dans une région au dessin d'une boîte sur l'écran :

Par exemple, tu peux dessiner une boîte avec :

```
ROUGE = 200, 0, 0
BOITE = Rect((20, 20), (100, 100))

def draw():
    screen.draw.rect(BOITE, ROUGE)
```

2.2.3 Chargement de resources

Les objets `images` et `sounds` peuvent être utilisés pour charger des images et des clips audio depuis des fichiers stockés dans les répertoires `images` et `sounds` respectivement. Pygame Zero va se charger de lire ces resources à la demande et va les mémoriser afin d'éviter de les relire.

Tu doit généralement t'assurer que tes images sont nommées avec des lettres minuscule, des chiffres et le caractère souligné _ seulement. Ils doivent aussi commencer par une lettre.

Les noms de fichiers suivants vont fonctionner correctement lors du chargement de resources :

```
alien.png
alien_hurt.png
alien_run_7.png
```

Ceux-ci ne fonctionneront pas :

```
3.png
3degrees.png
my-cat.png
sam's dog.png
```

Images

Pygame Zero peut charger des images aux formats `.png`, `.gif` et `.jpg`. le format PNG est recommandé : il autorise des images de haute qualité et gère la transparence.

Nous devons nous assurer que le répertoire `images` existe. Si ton projet contient les fichiers suivants :

```
space_game.py
images/alien.png
```

Alors `space_game.py` peux dessiner l'image "alien" à l'écran avec ce code :

```
def draw():
    screen.clear()
    screen.blit('alien', (10, 10))
```

Le nom passé à `blit()` est le nom du fichier image dans le répertoire images, sans l'extension de fichier.

Ou en utilisant l'API *Acteurs* :

```
alien = Actor('alien')

def draw():
    alien.draw()
```

Dans les deux cas, il y a quelques restrictions sur les noms de fichiers : ils ne peuvent seulement contenir des lettres latines minuscules, des chiffres et le caractère souligné _. Ceci afin d'éviter des problèmes de compatibilité quand ton jeu est exécuté sur différents systèmes d'exploitation qui ont différentes sensibilités à la casse.

Surfaces d'images

Tu peux également charger des images depuis le répertoire `images` en utilisant l'objet `images`. Ceci t'autorise à travailler avec les données de l'image elle-même, obtenir ces dimensions, etc. :

```
forest = []
for i in range(5):
    forest.append(
        Actor('tree', topleft=(images.tree.width * i, 0))
    )
```

Chaque image chargée est une `Surface` Pygame. Tu va généralement utiliser `screen.blit(...)` pour les dessiner à l'écran. Elle fournit également des méthodes pratiques pour obtenir la taille de l'image en pixels :

class Surface

get_width()
Retourne la largeur de l'image en pixels.

get_height()
Retourne la hauteur de l'image en pixels.

get_size()
Retourne un tuple (largeur, hauteur) indiquant la taille en pixels de la surface.

get_rect()
Retourne un objet `Rect` qui est prérempli avec les limites de l'images comme si l'image était située à l'origine.

Dans les faits, ceci est équivalent à :

```
Rect((0, 0), image.get_size())
```

Sons

Pygame Zero peut charger des sons aux formats `.wav` et `.ogg`. le format WAV est adapté aux petits effets sonores, tandis que OGG est un format compressé qui convient mieux à la musique. Tu peux trouver des fichiers .ogg et .wav libres de droits en ligne que tu peux utiliser dans ton jeu.

Nous devons nous assurer que le répertoire `sounds` existe. si ton projet contient les fichiers suivants :

```
drum_kit.py
sounds/drum.wav
```

Alors `drum_kit.py` peux jouer le son de batterie dès que la le bouton de la souris est pressé avec le code suivant :

```
def on_mouse_down():
    sounds.drum.play()
```

Chaque son chargé est un object Pygame `Sound` et a diverses méthodes pour jouer et stopper le son, ainsi qu'obtenir sa longueur en secondes :

class Sound

play()
Joue le son.

play(*n*)
Joue le son en boucle `n` fois.

Paramètres **n** – Le nombre de fois à répéter. Si tu donnes `-1` comme nombre de répétition, le son va se répéter indéfiniment (ou jusqu'à ce que tu appelles *`Sound.stop()`*)

stop()
Arrête de jouer le son.

get_length()
Retourne la durée du son en secondes.

Tu devrait éviter d'utiliser l'objet `sounds` pour jouer de longs morceaux de musique, parce que Pygame va complètement charger la musique en mémoire avant de la jouer, ceci peut utiliser beaucoup de mémoire, ainsi qu'introduire un délai au chargement de la musique.

2.2.4 Musique

Nouveau dans la version 1.1.

Avertissement : L'API musique est expérimentale et peux causer des problèmes sur certaines plateformes.

En particulier :
— le format MP3 peut ne pas être disponible dans toutes les distributions Linux.
— certains fichier OGG Vorbis semble bloquer Pygame en consommant 100% du CPU.
Dans le dernier cas, le problème peut être réglé en re-encodant (si possible avec un encodeur différent).

L'objet prédéfini `music` fournit le moyen de jouer de la musique depuis le répertoire `music/` (à côté de tes répertoires `images/` et `sounds/`, si tu en as). Le système musical va charger la piste de musique par petits morceaux pendant que la musique est jouée, évitant le problème de l'utilisation de `sounds` pour jouer de long morceaux de musique.

Une autre différence avec les effets sonores est que une seule piste de musique peut être jouée à la fois. Si tu joues une piste différente, la piste en cours d'exécution et stoppée.

`music.`**play**(*nom*)
Joue une piste de musique depuis le fichier donné. La piste va jouer indéfiniment.

Cela va remplacer la piste en cours et annuler toutes les pistes mises en attente avec `queue()`.

Tu n'as pas besoin d'inclure l'extension dans le nom de la piste, par exemple, pour jouer le fichier `handel.mp3` en boucle :

```
music.play('handel')
```

`music.`**play_once**(*nom*)
Similaire à `play()`, mais joue la musique une fois seulement.

`music.`**queue**(*nom*)
Similaire à `play_once()`, mais au lieu de stopper la musique en cours, la piste va être mise sur liste d'attente pour être jouée une fois que la musique en cours sera finie (ou après les autres musiques mises en attente précédemment).

`music.`**stop**()
Arrête la musique.

`music.`**pause**()
Met la musique en pause temporairement. Elle peut être redémarrée en appelant `unpause()`.

`music.`**unpause**()
Redémarre la musique.

`music.`**is_playing**()
Retourne vrai si la musique en en train d'être jouée (et n'est pas en pause), faux dans les autres cas.

`music.`**`fadeout`**(*durée*)
Diminue progressivement le volume et stoppe la musique en cours d'exécution.

Paramètres **`duration`** – La durée en secondes pendant laquelle le volume va diminuer. Par exemple, pour diminuer le son pendant une demi seconde, appelle `music.fadeout(0.5)`.

`music.`**`set_volume`**(*volume*)
Défini le volume de la musique.

Cela prend un nombre décimal entre 0 (pas de son) and 1 (volume maximum).

`music.`**`get_volume`**()
Retourne le volume courant de la musique.

Si tu as démarré la lecture de la piste de musique en utilisant *`music.play_once()`*, tu peux utiliser le *`hook on_music_end()`* pour faire quelque chose quand le morceau se termine, par exemple, pour choisir aléatoirement le prochain morceau.

2.2.5 Horloge

Souvent pendant l'écriture d'un jeu, tu veux planifier des événements pour qu'ils se produisent un moment plus tard. Par exemple, nous voudrions qu'un super chef extraterrestre apparaisse après 60 secondes. Ou peut-être qu'un bonus apparaisse toutes les 20 secondes.

Plus subtiles sont les situations où tu veux retarder une action pour une durée plus courte. Par exemple, tu pourrais avoir une arme laser qui prend 1 secondes à se recharger.

Nous pouvons utiliser l'objet `clock` pour programmer l'appel d'une fonction à se faire dans le future.

Commençons par définir une fonction `fire_laser` que nous voulons exécuter dans le future :

```
def fire_laser():
    lasers.append(player.pos)
```

Ensuite lorsque le bouton de tir est pressé, nous allons demander à l'objet `clock` de l'appeler pour nous, après exactement 1 seconde :

```
def on_mouse_down():
    clock.schedule(fire_laser, 1.0)
```

Note que `fire_laser` est elle-même une fonction, sans parenthèse, car elle n'est pas appelée ici ! L'horloge se chargera de le faire pour nous.

(C'est une bonne habitude d'écrire les durées en secondes avec une décimale, comme `1.0`. C'est ainsi plus évident en se relisant que cela représente une durée et pas un compte de quelques chose.)

L'objet `clock` fournit les méthodes utiles suivantes :

`class Clock`

`schedule`(*callback*, *delai*)
Programme *callback* à être appelée après le délai donné.

Un nombre répété d'appels à `schedule` va programmer l'appel à la fonction autant de fois.

Paramètres
— **`callback`** – Une fonction qui ne prend aucun paramètre.
— **`delai`** – Le délai en secondes à attendre avant que la fonction ne soit appelée.

`schedule_unique`(*callback*, *delai*)
Programme *callback* à être appelée une fois après le délai donné.

Si *callback* avait déjà été programmée, cela l'annule et la reprogramme. Cela s'applique aussi si elle était programmée plusieurs fois : après l'appel à `schedule_unique`, la fonction sera programmée exactement une fois.

Paramètres
— **`callback`** – Une fonction qui ne prend aucun argument.
— **`delai`** – Le délai en secondes à attendre avant que la function ne soit appelée.

`schedule_interval`(*callback*, *intervalle*)
Programme *callback* pour être appelée de façon répétée.

Paramètres
— **`callback`** – Une fonction qui ne prend aucun argument.
— **`intervalle`** – L'intervalle en secondes entre chaque appel à *callback*.

`unschedule`(*callback*)
Déprogramme l'appel à *callback* s'il a été précédemment programmé (soit parce qu'il a été programmé avec `schedule()` et n'a pas encore été exécuté ou parce qu'il a été programmé pour se répéter avec `schedule_interval()`.

Note que l'horloge de Pygame Zero ne tient que des références faibles aux fonctions que tu lui donnes. Il ne déclenchera pas d'événements programmés si les objets et méthodes ne sont pas référencés ailleurs. Cela empêche l'horloge de garder des objets vivants et de continuer à déclencher des événements inattendus après qu'ils sont normalement morts.

La contre-partie aux références faibles et que tu ne pourras pas programmer des lambdas ou tout autre objet créé uniquement pour être programmé. Tu devras garder une référence à l'objet.

2.2.6 Acteurs

Une fois que tu as pleins d'images se déplacant dans le jeux, il est intéressant d'avoir quelque chose qui regroupe à un seul endroit l'image et où elle se trouve à l'écran. Nous allons appeler chaque image se déplaçant à l'écran un acteur (`Actor`). Tu peux créer un acteur en fournissant au moins le nom d'une image (depuis le répertoire d'images décrit au-dessus). Pour dessiner un extraterrestre :

```
alien = Actor('alien', (50, 50))

def draw():
    screen.clear()
    alien.draw()
```

tu peux déplacer l'acteur en changeant son attribut `pos` dans la fonction `update` :

```
def update():
    if keyboard.left:
        alien.x -= 1
    elif keyboard.right:
        alien.x += 1
```

et tu peux changer l'image utilisée pour déssiner l'acteur en changeant son attribut `image` pour un nouveau nom d'image :

```
alien.image = 'alien_hurt'
```

Les acteurs ont tous les mêmes attributs et méthodes que *Rect*, en incluant les méthodes telles que .colliderect() qui peut être utilisée pour tester si deux acteurs se percutent.

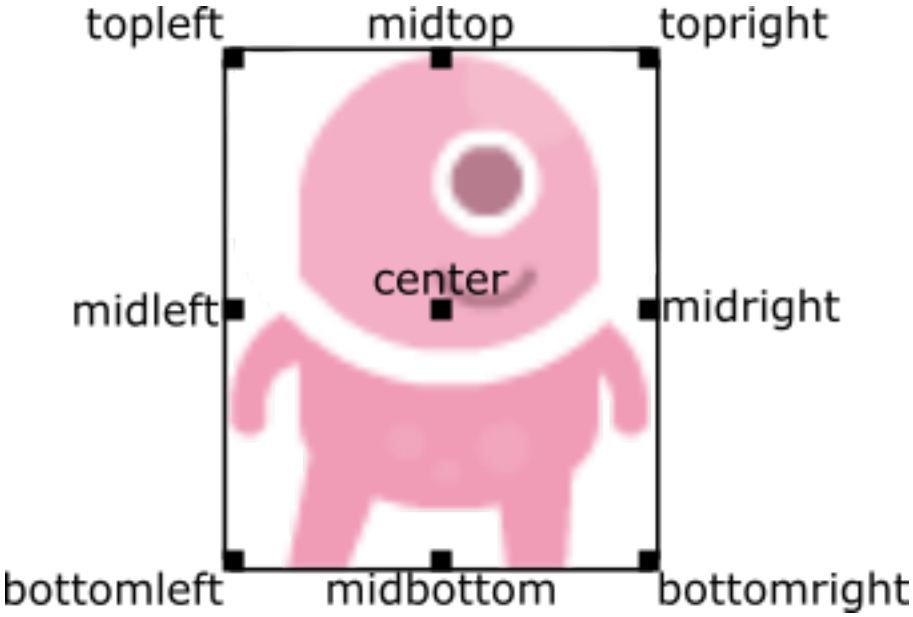
Positionner les acteurs

Si tu donnes une nouvelle valeur à un des attributs de position, alors l'acteur va se déplacer. Par exemple :

```
alien.right = WIDTH
```

va positionner l'extraterrestre de façon à ce que sa droite est la valeur `WIDTH`.

De façon similaire, tu peux aussi donner la position initiale de l'acteur dans le constructeur, en donnant l'un des mots-clés suivant en argument : `pos`, `topleft`, `topright`, `bottomleft`, `bottomright`, `midtop`, `midleft`, `midright`, `midbottom` ou `center` :

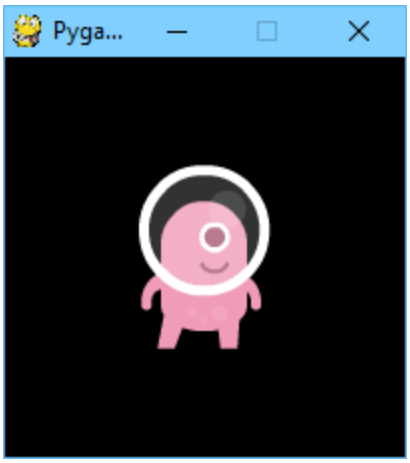


Cela peut être fait pendant la création ou en affectant une paire de coordonnées (x, y). Par exemple :

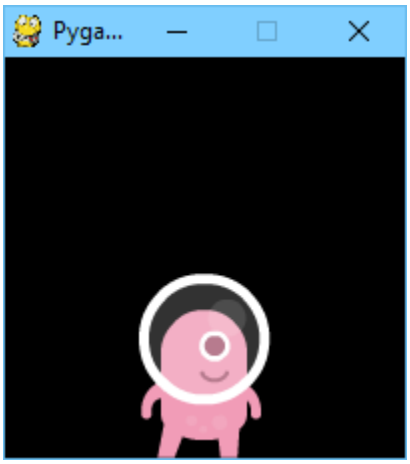
```
WIDTH = 200
HEIGHT = 200

alien = Actor('alien', center=(100,100))

def draw():
    screen.clear()
    alien.draw()
```



En changeant `center=(100, 100)` en `midbottom=(100, 200)` cela te donnes :



Si tu ne spécifies pas de position initiale, l'acteur sera positionné par défaut dans le coin en haut à gauche (équivalent à `topleft=(0, 0)`).

Point d'ancrage

Les acteurs ont une « position d'ancrage », qui est une façon pratique de positionner les acteurs dans la scène. Par défaut, le point d'ancrage est le centre, si bien que l'attribut `.pos` représente le centre de l'acteur (de même pour les coordonnées `x` et `y`). Il est commun de définir le point d'ancrage à une autre partie du sprite (peut être les pieds afin de pouvoir facilement faire tenir l'acteur sur quelque chose) :

```
alien = Actor('alien', anchor=('center', 'bottom'))
spaceship = Actor('spaceship', anchor=(10, 50))
```

`anchor` est défini comme le tuple `(xanchor, yanchor)`, où les valeurs peuvent être des décimaux ou les chaînes `left`, `center`/`middle`, `right`, `top` ou `bottom`.

Rotation

Nouveau dans la version 1.2.

L'attribut `.angle` de l'acteur controle la rotation du sprite, en degrés, dans le sens trigonométrique (sens inverse des aiguilles d'une montre).

Le cente de rotation est le *point d'ancrage* de l'acteur.

Note que cela va changer la hauteur (`width`) et la largeur (`height`) de l'acteur.

Par exemple, pour faire tourner lentement un astéroïde dans l'espace :

```
asteroid = Actor('asteroid', center=(300, 300))

def update():
    asteroid.angle += 1
```

Pour le faire tourner dans l'aute sens, nous écririons `update()` ainsi :

```
def update():
    asteroid.angle -= 1
```

Un autre exemple : nous pourrions faire qu'un acteur `ship` soit toujours tourné vers la souris. Comme *`angle_to()`* retourne 0 pour la droite, l'acteur *ship* devrait se tourner vers la droite :

```
ship = Actor('ship')

def on_mouse_move(pos):
    ship.angle = ship.angle_to(pos)
```

Souviens toi que les angles bouclent : 0 degrés == 360 degrés == 720 degrés. De même -180 degrés == 180 degrés.

Distance et angle

Les acteurs ont des méthodes très pratiques pour calculer leur distance ou angle avec d'autres acteurs ou des paires de coordonnées `(x, y)`.

`Actor.`**`distance_to`**`(`*cible*`)`
: Retourne la distance depuis la position de l'acteur à la *cible* en pixels.

`Actor.`**`angle_to`**`(`*cible*`)`
: Retourne l'angle depuis la position de l'acteur à la *cible* en degrés.

  Cela va retouner un nombre entre -180 et 180 degrés. La droite est 0 degrés et les angles augmentent en allant dans le sens horaire inverse (sens trigonométrique).

  Donc :
  — La gauche est 180 degrés.
  — Le haut est 90 degrés.
  — Le bas est -90 degrés.

2.2.7 Le clavier

Tu as sans doute remarqué que nous avons utilisé l'objet `keyboard` dans le code au-dessus. Si tu veux savoir quelles touches du clavier sont pressées, tu peux lire les attributs de l'objet pré-défini `keyboard`. Si, par exemple, la flêche gauche est maintenue enfoncée, alors `keyboard.left` sera vrai (`True`), sinon il sera faux (`False`).

Il y a un attribut pour chaque touche, quelques exemples :

```
keyboard.a  # La touche 'A'
keyboard.left  # La touche de navigation gauche
keyboard.rshift  # La touche majuscule droite
```

(suite sur la page suivante)

(suite de la page précédente)

```
keyboard.kp0  # La touche '0' sur le pavé numérique
keyboard.k_0  # La touche prinicipale '0'
```

L'ensemble des constantes de touche est donné dans la documentation Buttons and Keys, mais les attributs sont en minuscule, parce que ceux sont des variables et non des constantes.

Obsolète depuis la version 1.1 : Le noms en majuscule ou préfixés (ie. `keyboard.LEFT` ou `keyboard.K_a`) sont maintenant dépréconisés : utilise plutôt des noms d'attributs en minuscule.

Nouveau dans la version 1.1 : Tu peux aussi interroger l'état des touches en utilisant les constantes des touches elles-même :

```
keyboard[keys.A]  # Vrai si la touche 'A' est pressée
keyboard[keys.SPACE]  # Vrai si la barre d'espace est pressée
```

2.2.8 Animations

Tu peux animer pratiquement tous dans pygame en utilisant la fonction pré-définie `animate()`. Par exemple, pour bouger un *acteur* depuis sa position courante à l'écran vers la position `(100, 100)` :

```
animate(alien, pos=(100, 100))
```

animate(*objet*, *tween='linear'*, *duration=1*, *on_finished=None*, ***cibles*)

Anime les attributs de *objet* depuis leurs valeurs courantes vers celles spécifiées par mots-clés dans *cibles*.

Paramètres

- **tween** – Le type de *tweening* à utiliser.
- **duration** – La durée de l'animation en secondes.
- **on_finished** – Fonction appelée à la fin de l'animation.
- **targets** – Le valeurs cibles des attributs à animer.

L'argument *tween* peut être l'un des suivants :

"linear"	Anime avec une vitesse constante du début à la fin
"accelerate"	Démarre doucement et accélère à la fin
"decelerate"	Démarre rapidement et ralentit à la fin
"accel_decel"	Accélère jusqu'à la moitié puis ralentit à la fin
"end_elastic"	Oscille légèrement à la fin
"start_elastic"	Oscille légèrement au démarrage
"both_elastic"	Oscille aux deux extrémités
"bounce_end"	Accélère jusqu'à la fin et rebondit
"bounce_start"	Rebondit au démarrage
"bounce_start_end"	Rebondit aux deux extrémités

La fonction `animate()` retourne une instance de la classe `Animation` :

class Animation

stop(*complete=False*)

Arrête l'animation et la complète optionnellement jusqu'à sa valeur finale.

Paramètres **complete** – Donne à l'attribut animé sa valeur cible.

running

Vaut vrai si l'animation est en cours d'exécution. Il vaudra faux si la durée de l'animation est dépassée ou si la méthode `stop()` a été appelée avant.

on_finished
Tu peux définir cet attribut avec une fonction qui sera appelée à la fin normale de l'animation. Le paramètre `on_finished` de la fonction `animate()` défini aussi cet attribut. Elle ne sera pas appelée si `stop()` est appelée. Cette fonction ne prend pas de paramètres.

2.2.9 Générateur de sons

Nouveau dans la version 1.2.

Pygame Zero peut jouer des sons en utilisant un synthétiseur intégré.

`tone.`**`play`**(*hauteur*, *duration*)
Joue une note à la hauteur donnée pour la durée donnée.

La durée est en secondes.

La *hauteur* peut être donnée comme un nombre, dans ce cas cela sera la fréquence de la note en hertz.

Sinon, la hauteur peut être définie comme une chaîne représentant le nom d'une note et un octave. Par exemple :
— `'E4'` sera un mi au 4ème octave.
— `'A#5'` sera un la dièse au 5ème octave.
— `'Bb3'` sera un si bémol au 3ème octave.

Créer des notes, en particulier de longues notes, prend du temps (jusqu'à plusieurs millisecondes). Tu peux créer des notes à l'avance afin de ne pas ralentir ton jeu pendant qu'il s'exécute :

`tone.`**`create`**(*hauteur*, *duration*)
Crée et renvoie un objet *Sound*.

Les paramètres sont les mêmes que ceux de *play()*, décrit au-dessus.

Ceci peut être utilisé dans un programme Pygame Zero comme ceci :

```
beep = tone.create('A3', 0.5)

def on_mouse_down():
    beep.play()
```

CHAPITRE 3

Guide Utilisateur

3.1 Installer Pygame Zero

3.1.1 Inclus avec Mu

L'éditeur Mu, qui est destiné aux débutants, inclus une version de Pygame Zero.

Vous devez basculer vers le mode Pygame Zero pour pouvoir l'utiliser. Écrivez alors un programme et utiliser le bouton Play pour le lancer avec Pygame Zero.

Note : La version de Pygame Zero inclus avec Mu peut ne pas être la plus récente ! Vous pouvez trouver quelle version est installée en exécutant le code suivant :

```
import pgzero
print(pgzero.__version__)
```

3.1.2 Installation indépendante

Avant tout, vous avez besoin que **Python 3** soit installé ! Il est généralement déjà installé si vous utilisez **Linux** ou un **Raspberry Pi**. Vous pouvez le télécharger depuis python.org sur d'autres systèmes.

Windows

Pour installer Pygame Zero, utiliser **pip**. Depuis une invite de ligne de commande, entrez

```
pip install pgzero
```

Mac

Dans une fenêtre de terminal, entrez

```
pip install pgzero
```

Linux

Dans une fenêtre de terminal, entrez

```
sudo pip install pgzero
```

Certains systèmes Linux l'appelle `pip3`, si la commande d'au-dessus affiche un message du genre `sudo: pip: command not found` essayez alors :

```
sudo pip3 install pgzero
```

Parfois pip n'est pas installé. Si c'est le cas, essayer la commande suivante avant de ré-exécuter la commande précédente :

```
sudo python3 -m ensurepip
```

3.1.3 Installer le REPL

le REPL de Pygame Zero est une fonctionnalité optionnelle. Elle peut être activée lors de l'installation avec `pip` en ajouteant `pgzero[repl]` à la commande pip line :

```
pip install pgzero[repl]
```

Si vous n'êtes pas sûre d'avoir le REPL déjà installé, vous pouvez toujours exécuter cette commande (cela ne cassera rien s'il est déjà installé !).

3.2 Using the REPL (Read-Evaluate-Print Loop)

The REPL allows you to interact with a running Pygame Zero game using Python commands. As you type it will offer suggestions based on variables that exist in your program. This can be useful for debugging your game or tuning how difficult it is.

REPL is short for a Read-Evaluate-Print Loop ; it means :

1. **Read** a line of code typed by you
2. **Evaluate** the code
3. **Print** the result
4. **Loop** back to step 1 !

This is an *optional feature* that may need to *be installed* if it was not originally installed with Pygame Zero. If you try using the REPL, Pygame Zero will let you know if it is not installed.

3.2.1 Running a Pygame Zero program with the REPL

If you normally run your Pygame Zero program using the terminal, add `--repl` to the command line when running `pgzrun`. For example, if your game is in a file called `mygame.py`, run :

```
pgzrun --repl mygame.py
```

3.2.2 Using the REPL

Python code that you type at the REPL is evaluated as if you had typed it into your game file.

For example, if your game file contains the code

```
alien = Actor('alien', pos=(54, 60))

def draw():
    screen.clear()
    alien.draw()
```

Then at the REPL you could type `alien` to see the alien object :

```
>>> alien
<Actor 'alien' pos=(54, 60)>
```

You can set attributes on the `alien` object and see it move :

```
>>> alien.x = 90
```

3.3 Running Pygame Zero in IDLE and other IDEs

Nouveau dans la version 1.2.

Pygame Zero is usually run using a command such as :

```
pgzrun my_program.py
```

Certain programs, such as integrated development environments like IDLE and Edublocks, will only run `python`, not `pgzrun`.

Pygame Zero includes a way of writing a full Python program that can be run using `python`. To do it, put

```
import pgzrun
```

as the very first line of the Pygame Zero program, and put

```
pgzrun.go()
```

as the very last line.

3.3.1 Example

Here is a Pygame Zero program that draws a circle. You can run this by pasting it into IDLE :

```
import pgzrun


WIDTH = 800
HEIGHT = 600

def draw():
    screen.clear()
    screen.draw.circle((400, 300), 30, 'white')


pgzrun.go()
```

3.4 Autres bibliothèques comme Pygame Zero

Pygame Zero a démarré une tendance pour les bibliothèques « zéro » Python. Nos amis ont créé ces superbes bibliothèques. Certaines d'entre elles peuvent être combinées avec Pygame Zero !

3.4.1 Network Zero

Network Zero simplifie la découverte et la communication en réseau de plusieurs machines ou processus sur une même machine.

Prudence : Si vous voulez utiliser Network Zero avec Pygame Zero, soyez sûr qu'il ne **bloque** pas (arrête tout en attendant des messages). Cela va interrompre Pygame Zero qui va arrêter les animations à l'écran ou même répondre aux entrées. Donnez toujours aux options `wait_for_s` ou `wait_for_reply_s` la valeur de `0` secondes.

3.4.2 GUI Zero

GUI Zero est une bibliothèque pour créer des interfaces graphiques avec fenêtres, boutons, curseurs, champs texte, etc..

Parce que GUI Zero et Pygame Zero utilisent des approches différentes pour dessiner à l'écran, ils ne sont pas utilisables ensemble.

3.4.3 GPIO Zero

GPIO Zero est une bibliothèque pour contrôler les appareils connectés aux broches d'entrées/sorties générales (GPIO) sur un Raspberry Pi.

GPIO Zero marche généralement dans son propre *thread*, ce qui signifie qu'il va généralement bien fonctionner avec Pygame Zero.

Prudence : En copiant les exemples de GPIO Zero, ne copiez pas les appels de fonction `time.sleep()` ou les boucles `while True:`, car ils vont empêcher Pygame Zero d'animer l'écran ou répondre aux entrées. Utiliser plutôt les fonctions *Horloge* pour appeler périodiquement des fonctions, ou la fonction *`update()`* pour vérifier une valeur à chaque *frame*.

3.4.4 Adventurelib

Adventurelib est une bibliothèque qui simplifie l'écriture de jeux orientés texte (mais qui ne fait pas tout à votre place !).

Écrire des jeux orientés texte demande des aptitudes différentes que d'écrire des jeux graphiques. Adventurelib est destiné à des développeurs Python d'un niveau légèrement plus avancé que celui de Pygame Zero.

Adventurelib ne peut actuellement pas être combiné avec Pygame Zero.

3.4.5 Blue Dot

Blue Dot vous permet de contrôler vos projets Raspberry Pi sans fils en utilisant un appareil Android comme une télécommande Bluetooth.

Blue Dot tourne généralement dans son propre *thread*, ce qui signifie qu'il va généralement bien fonctionner avec Pygame Zero.

Prudence : Évitez les appels à la fonction `time.sleep()`, les boucles `while True:` et les méthodes bloquantes de Blue Dot `wait_for_press` et `wait_for_release`, car celles-ci vont empêcher Pygame Zero d'animer l'écran ou de répondre aux entrée. Utiliser plutôt les fonctions *Horloge* pour appeler périodiquement des fonctions, ou la fonction *`update()`* pour vérifier une valeur à chaque *frame*.

Astuce : Vous connaissez d'autres bibliothèques qui ont leur place ici ?

Ouvrer un problème sur le *tracker* de problèmes pour nous le faire savoir !

3.5 Changelog

3.5.1 1.2 - 2018-02-24

— New : *Actors can be rotated* by assigning to `actor.angle`
— New : Actors now have *`angle_to()`* and *`distance_to()`* methods.
— New : Actors are no longer subclasses of Rect, though they provide the same methods/properties. However they are now provided with floating point precision.
— New : `tone.play()` function to allow playing musical notes.
— New : `pgzrun.go()` to allow running Pygame Zero from an IDE (see *Running Pygame Zero in IDLE and other IDEs*).
— New : show joypad icon by default
— Examples : add Asteroids example game (thanks to Ian Salmons)
— Examples : add Flappy Bird example game
— Examples : add Tetra example game (thanks to David Bern)
— Docs : Add a logo, fonts and colours to the documentation.
— Docs : Documentation for the *anchor point system for Actors*
— Docs : Add *Migrer depuis Scratch* documentation
— Fix : `on_mouse_move()` did not correctly handle the `buttons` parameter.
— Fix : Error message when resource not found incorrectly named last extension searched.
— Fix : Drawing wrapped text would cause crashes.
— Fix : *`animate()`* now replaces animations of the same property, rather than creating two animations which fight.
— Updated ptext to a revision as of 2016-11-17.
— Removed : removed undocumented British English `centrex`, `centrey`, `centre` attribute aliases on ZRect (because they are not Rect-compatible).

3.5.2 1.1 - 2015-08-03

— Added a spell checker that will point out hook or parameter names that have been misspelled when the program starts.
— New ZRect built-in class, API compatible with Rect, but which accepts coordinates with floating point precision.
— Refactor of built-in `keyboard` object to fix attribute case consistency. This also allows querying key state by `keys` constants, eg. `keyboard[keys.LEFT]`.
— Provide much better information when sound files are in an unsupported format.
— `screen.blit()` now accepts an image name string as well as a Surface object, for consistency with Actor.
— Fixed a bug with non-focusable windows and other event bugs when running in a virtualenv on Mac OS X.
— Actor can now be positioned by any of its border points (eg. `topleft`, `midright`) directly in the constructor.

— Added additional example games in the `examples/` directory.

3.5.3 1.0.2 - 2015-06-04

— Fix : ensure compatibility with Python 3.2

3.5.4 1.0.1 - 2015-05-31

This is a bugfix release.

— Fix : Actor is now positioned to the top left of the window if `pos` is unspecified, rather than appearing partially off-screen.

— Fix : repeating clock events can now unschedule/reschedule themselves
Previously a callback that tried to unschedule itself would have had no effect, because after the callback returns it was rescheduled by the clock.
This applies also to `schedule_unique`.

— Fix : runner now correctly displays tracebacks from user code

— New : Eliminate redraws when nothing has changed
Redraws will now happen only if :

- The screen has not yet been drawn
- You have defined an update() function
- An input event has been fired
- The clock has dispatched an event

3.5.5 1.0 - 2015-05-29

— New : Added `anchor` parameter to Actor, offering control over where its `pos` attribute refers to. By default it now refers to the center.

— New : Added Ctrl-Q/-Q as a hard-coded keyboard shortcut to exit a game.

— New : `on_mouse_*` and `on_key_*` receive `IntEnum` values as `button` and `key` parameters, respectively. This simplifies debugging and enables usage like :

```
if button is button.LEFT:
```

3.5.6 1.0beta1 - 2015-05-19

Initial public (preview) release.

CHAPITRE 4

Autocollants par

Sticker Mule a gracieusement offert de fournir des autocollants de portable Pygame Zero aux étudiants.

4.1 Laptop Stickers

Sticker Mule have graciously offered to provide a number of stickers for Pygame Zero users for free.

Laptop stickers are a great way to encourage students to continue tinkering and learning, as well as spreading the word about Pygame Zero.

The stickers should look a little like this (not to scale) :

4.1.1 For learners

Due to the costs of distribution, and because Pygame Zero is a free community library, **we don't have a way of getting stickers directly to you yet.**

It may be possible to obtain stickers at conferences and meet-ups. Please check back soon.

4.1.2 For educators/meet-ups

We would like to distribute stickers primarily via educators and educational meet-ups. At this time it is not known how many stickers we will be able to distribute in this way (and it may be prohibitively expensive to ship them outside the UK).

Please fill out our Google Form to express your interest.

4.1.3 For developers

Free stickers are primarily intended for learners. However, if a pull request you make to Pygame Zero or a translation is accepted, we would be very happy to give you a free laptop sticker if the opportunity arises.

Please request a sticker in your Pull Request comments (or make yourself known at a conference/meet-up).

If you attend educational events or Python events regularly, and you would be willing to distribute stickers, this could also be useful. Please let us know.

CHAPITRE 5

Améliorer Pygame Zero

5.1 Roadmap

Pygame Zero is an open source project, and as with any such project, the development roadmap is subject to change.

This document just lays out some goals for future releases, but there is **no guarantee** that these targets will be hit.

5.1.1 Translations

Pygame Zero is aimed at young users, whose English skills might not be good enough to read the documentation if it isn't in their own language.

Adding translations of the documentation would help to bring Pygame Zero to new users. This is something that needs contributors to help with. My own language skills aren't good enough !

Please see *the translating guide* if you think you can help.

5.1.2 Gamepad Support

Github Issue : #70

SNES-style gamepads are now extremely cheap. For example, they are sold for a few pounds from the Pi Hut, in packs of 2 at Amazon, and even in some Raspberry Pi bundles.

Gamepad support should not be limited to these specific models ; rather, we should treat this as a lowest-common-denominator across modern gamepads, as nearly all more modern gamepads have at least as many buttons and axes.

This feature needs to be added in a way that will not **require** a gamepad to play any Pygame Zero game, in order to follow the principle of *Rendre accessible*.

5.1.3 Surface juggling

Github Issue : #71

Pygame experts make lots of use of off-screen surfaces to create interesting effects.

Pygame Zero chose to consider only the screen surface, which we wrap with a richer `Screen` API for drawing, etc.

The problem is that there is no easy path to using additional surfaces - Pygame Zero immediately becomes dead weight as you start to look past that curtain.

We should look to smooth out this path to make Pygame Zero Actors and Screen work better with custom surfaces.

5.1.4 Storage

Github Issue : #33

It would be useful for users to be able to save and load data.

The obvious application is save games, but saving and loading whole games can be pretty hard to get right. The simpler application would just be saving settings, customisations, high scores, or the highest level reached.

Python of course has APIs for reading and writing files, but this has additional complexity that teachers might not want to teach immediately.

5.2 Les principes de Pygame Zero

Veuillez lire attentivement ce qui suit avant de contribuer.

Étant donné que Pygame Zero s'adresse essentiellement aux débutants, nous devons faire très attention à ne pas introduire de difficultés pour les programmeurs qui n'ont pas encore appris à les surmonter.

5.2.1 Rendre accessible

L'objectif principal de Pygame Zero est d'être accessible aux programmeurs débutants. La conception de l'API en est évidemment influencée.

Ceci s'applique également à des éléments comme les exigences matérielles : Pygame Zero a choisi à l'origine de ne prendre en charge que les entrées du clavier et de la souris, afin d'être accessible à tous les utilisateurs.

5.2.2 Être prudent

À l'amorce du développement de Pygame Zero, Richard et moi (Daniel) avons effectué plusieurs allers-retours sur diverses fonctionnalités. Nous les avons ajoutées, les avons essayées et les avons retirées.

Les fonctionnalités doivent être rejetées si elles sont trop expérimentales ou si elles peuvent prêter à confusion.

Cela s'applique également aux autres éléments comme la prise en charge de l'OS : nous interdisons les noms de fichiers qui ne sont pas susceptibles d'être compatibles entre les systèmes d'exploitation.

5.2.3 Fonctionnel

Pygame Zero couvre pratiquement l'intégralité de Pygame - mais nous n'exposons pas toutes les fonctionnalités. Nous n'exposons que les fonctions qui fonctionnent vraiment bien sans peine et masquons une partie des autres éléments qui fonctionnent moins bien ou qui nécessitent des étapes supplémentaires.

5.2.4 Minimiser les temps d'exécution

En fin de compte, Pygame Zero est un environnement de développement de jeux et l'adéquation des performances est un élément clé.

Il n'est pas vraiment judicieux de faire une vérification coûteuse à chaque image pour détecter un écueil potentiel. Au lieu de cela, nous pouvons effectuer un contrôle au départ, ou vérifier seulement quand une exception est levée pour la diagnostiquer et rapporter plus d'informations.

5.2.5 Des erreurs limpides

Lorsque des exceptions sont levées par Pygame Zero, elles doivent délivrer des messages d'erreur clairs qui expliquent le problème.

5.2.6 Bien documenter

Comme tous les projets, Pygame Zero nécessite une bonne documentation. Les demandes d'intégration (pull requests) sont plus susceptibles d'être acceptées si elles sont accompagnées des documentations adéquates.

Essayez d'éviter les phrases compliquées et les termes techniques dans la documentation, afin qu'elle soit plus facilement lisible par des programmeurs inexpérimentés.

5.2.7 Minimiser les changements radicaux

Dans les structures de l'éducation, les utilisateurs n'ont pas toujours le contrôle sur la version d'une bibliothèque qu'ils utilisent. Ils ne savent pas comment installer ou mettre à jour afin d'obtenir la dernière version.

Il est ici plus important d'obtenir des fonctionnalités correctes du premier coup que dans beaucoup d'autres projets.

5.3 Contribuer à Pygame Zero

Le projet Pygame Zero est hébergé sur GitHub :

https://github.com/lordmauve/pgzero

5.3.1 Rapporter un bogue ou une demande

Vous pouvez rapporter un bogue ou une demande de fonctionnalité que vous pensez être nécessaire à Pygame Zero en utilisant le Github issue tracker.

Voici quelques conseils à garder à l'esprit avant de procéder :

— Vous pourriez ne pas être le premier ! Vous devriez vérifier si quelqu'un n'a pas déjà rapporté le problème en cherchant parmi la liste des problèmes existants, à la fois ouverts ou fermés.
— Les développeurs ont besoin de savoir quelle version de Pygame Zero vous utilisez et sur quel système d'exploitation (Windows, Mac, Linux, etc.) et sa version (Windows 10, Ubuntu 16.04, etc.) vous le faites tourner.

5.3.2 Comment créer une *pull request*

Vous pouvez contribuer des changements à Pygame Zero en créant une *pull request*.

C'est une bonne idée de d'abord *rapporter un problème* afin de pouvoir discuter si votre changement a du sens.

Github dispose d'aide sur comment créer une pull request, mais voici une version rapide :

1. Soyez sûr d'être authentifié sur Github.
2. Allez à page Github de Pygame Zero.
3. Cliquez sur « Fork » pour créer votre copie personnelle du dépôt.
4. Dupliquez ce dépôt sur votre ordinateur :

   ```
   git clone git@github.com:monlogin/pgzero.git
   ```

   Rappelez vous de remplacer `monlogin` par votre login Github.
5. Créez une branche dans laquelle vous allez apporter les changements. Choisissez un nom de branche qui décrit le changement que vous voulez apporter.

   ```
   git checkout -b ma-nouvelle-branche master
   ```
6. Écrivez le changement que vous souhaitez apporter.
7. Ajouter les fichiers que vous souhaitez valider :

   ```
   git add pgzero
   ```
8. Valider la révision des fichiers avec un message explicite :

   ```
   git commit -m "Correction du problème #42 en renommant les paramètres"
   ```

   Vous pouvez répéter les étapes 6 à 8 autant de fois que nécessaires jusqu'à obtenir la fonctionnalité désirée.
9. Poussez la nouvelle révision sur votre dépôt distant.

   ```
   git push --set-upstream origin ma-nouvelle-branche
   ```
10. Aller sur la page Github de votre dépôt and cliquez sur le bouton « Créer une pull request ».

5.3.3 Installer une version en développement

Il est possible d'installer une version en cours de développement en utilisant *pip*. Depuis le répertoire racine de votre copie locale des sources, lancez :

```
pip3 install --editable .
```

La version installée va maintenant contenir tous les changements que vous apporterez au code source.

Autrement, si vous ne voulez pas l'installer du tout, vous pouvez le lancer ainsi :

```
python3 -m pgzero <name of pgzero script>
```

Par exemple :

```
python3 -m pgzero examples/basic/demo1.py
```

5.3.4 Comment lancer les tests

Les tests peuvent être exécutés avec :

```
python3 setup.py test
```

5.3.5 Aider à traduire la documentation

L'API de Pygame Zero sera toujours en anglais, mais vous pouvez diffuser Pygame Zero à plus d'utilisateurs dans le monde si la documentation est disponible dans leur langue.

Si vous parlez couramment une autre langue, vous pouvez contribuer en traduisant tout ou partie de la documentation.

La documentation est écrite en reStructuredText, qui est un language textuel à balises destiné à la documentation technique. Essayez autant que possible de préserver le formatage actuel. reStructuredText n'est pas très difficile une fois que vous êtes habitué.

Créer une traduction consiste à créer un dépôt séparé sur Github avec une copie de la documentation, réécrite (au moins en partie) dans la langue que vous souhaitez contribuer. L'avantage est que vous pouvez travailler sur la traduction à votre propre rythme, sans soumettre des pull requests au projet `pgzero` lui même. Consultez le guide de traduction sur *Read The Docs* pour plus de détails.

Si cela vous semble abordable, voici comment vous pouvez procéder :

1. D'abords, ouvrez un problème sur le pgzero issue tracker. Vous devriez chercher s'il n'existe pas un autre problème concernant la traduction que vous voulez entreprendre, avant d'en ouvrir un nouveau. Cela garantira que vous ne travaillerez pas sur une traduction que quelqu'un d'autre aurait déjà faite (vous pourriez peut-être plutôt collaborer).
2. Créer a nouveau dépôt Github, appeler le pgzero-*langue*, par exemple `pgzero-spanish` si vous aller traduire en espagnol.
3. Dupliquez ce dépôt sur votre ordinateur.
4. Téléchargez le répertoire `doc/` de Pygame Zero et valider une révision dans votre projet. Vous pouvez le faire en l'extrayant du fichier ZIP du dépôt. Vous avez uniquement besoin du répertoire `doc/`, vous pouvez effacer les autres fichiers.
5. Maintenant, vous pouvez travailler sur les fichiers .rst dans le répertoire `doc` en les traduisant avec votre éditeur préféré. Vous devriez valider des révision régulièrement et les pousser sur Github.
6. Postez un commentaire avec un lien vers votre dépôt dans le problème Github créer à l'étape 1. Vous pouvez le faire dès que vous avez de la matière à montrer, cela va aider d'autres personnes à contribuer à vos traductions s'ils sont intéressés.
7. Configurer la documentation pour la construire sur Read The Docs. Encore une fois, postez un commentaire sur le problème Github quand cela fonctionne.
8. Nous pouvons alors relier la nouvelle documentation traduite avec la documentation officielle de Pygame Zero.

Notez que Pygame Zero va être mis à jour et la documentation va être changée en conséquence. En utilisant Git il est possible de voir ce qui a changé dans la documentation originale, afin que vous puissiez apporter les changements nécessaires dans la documentation traduite.

Index

Q

R

S

T

U

V

W

X

Y

Z