

Beste de savoir

Pygame pour les zesteurs

4 septembre 2018

Table des matières

I. Introduction	3
II. Pourquoi ce tutoriel?	5
III. Pourquoi Pygame ?	7
IV. Le site officiel de Pygame, une mine d'or!	9
V. A la découverte de Pygame !	11
1. Avant de commencer...	13
1.1. "A small game made with Pygame"	13
1.2. DonkeyPy, un DonkeyKong like	13
1.3. Un moteur de raycasting	14
2. Installer et importer Pygame	15
2.1. Installer Pygame	15
2.1.1. Windows	15
2.1.2. Linux (Debian et dérivés)	15
2.2. Import pygame	16
2.3. Vérifier qu'un module a bien été importé	17
Contenu masqué	17
3. Créer une simple fenêtre (personnalisable!)	19
3.1. Créons une fenêtre basique	19
3.2. Personnalisons notre fenêtre	21
Contenu masqué	25
4. Afficher des images	26
4.1. Lire une image	26
4.2. Convertir nos images	26
4.3. Afficher une image sur notre belle fenêtre!	27
5. La gestion des événements	32
5.1. Qu'est-ce qu'un événement ?	32
5.2. Comment les capturer ?	32

5.3. Les types d'événements disponibles	35
5.4. Les événements disponibles!	38
5.5. Entraînons-nous avec les événements!	39
Contenu masqué	42
6. TD?? Un petit jeu très simple	48
6.1. Le cahier des charges	48
6.2. Un menu et des constantes	49
6.2.1. Le menu	49
6.2.2. Les constantes	50
6.3. La gestion de la raquette	51
6.4. Gestion de la balle	51
6.5. Et enfin, le cœur de la maison, la cuisine!	52
Contenu masqué	55
7. Annexes	62
7.1. Les différents modules de Pygame	62
VI. Conclusion	64
VII. Remerciements	66
VIII.Le mot de la fin	68
% PYGAME POUR LES ZESTEURS % SuperFola % 21 janvier 2018	

Première partie

Introduction

I. Introduction

Les jeux-video ça vous plait ? Un peu, beaucoup ? Si vous connaissez le langage Python, alors ce tutoriel tombe à pic (enfin, pas comme un pigeon descendu par un chasseur) !

Deuxième partie

Pourquoi ce tutoriel ?

II. Pourquoi ce tutoriel ?

On a remarqué que beaucoup de bibliothèques avaient de très bons tutoriels, mais en anglais, et cela a toujours rebuté certains programmeurs, qu'ils soient débutants ou non ! De plus, les-dits tutoriels n'ont pas à proprement parler une bonne quantité de Travaux Pratiques (TP) / Travaux Dirigés (TD) pour aider le débutant dans sa première utilisation de la bibliothèque. Il doit donc lire toute une documentation, souvent complexe quand on vient à peine de débiter en programmation.

C'est pourquoi on a décidé de rédiger un tutoriel sur Pygame (qui sera plus complet que les quelques-uns que l'on peut voir sur d'autres sites), avec des TP et TD, et qui vous expliquera **tout**, seulement à partir d'un zeste de savoir à propos de Python (3) !

i

Prérequis

Comme dit précédemment, il faut que vous ayez des bases dans le langage de programmation Python, dans sa version 3. Si ce n'est pas le cas, on vous renvoie à ces différents tutoriels, très bien écrits :

- [Tutoriel de Gérard Swinnen](#) ↗
- [Tutoriel de martinqt et Dr@zielux](#) ↗
- [Tutoriel de Nohar, pas encore terminé](#) ↗

Objectifs

Ici vous allez apprendre à créer des fenêtres, charger des images et les utiliser, jouer une musique, et encore plein d'autres choses extrêmement intéressantes !

Troisième partie

Pourquoi Pygame ?

III. Pourquoi Pygame ?

Pygame [↗](#) est une bibliothèque bien connue chez les développeurs python, déjà parce que :

- c'est un *binding*¹ de la SDL 1.2 en C (et la SDL est très connue et utilisée!)
- c'est une bibliothèque qui permet de coder des jeux (entre autres), car :
 - elle permet d'afficher des images
 - de jouer des musiques
 - de faire du "pixel perfect" avec son module `mask`
 - d'écrire du texte dans vos jeux, par exemple un dialogue entre deux personnages
 - de créer des images de toutes pièces et de les enregistrer
 - de faire des dessins, et encore pleins d'autres choses !

Bon d'accord, c'est bien peu comme arguments.

En voici donc d'autres :

- elle est portable sur différents systèmes d'exploitation, donc vos programmes tourneront (normalement) aussi bien sur Windows, Linux et MacOS
- elle est simple à prendre en main
- son développement est très actif

Quatrième partie

Le site officiel de Pygame, une mine d'or!

IV. Le site officiel de Pygame, une mine d'or!

Pour ceux qui aimeraient avoir des exemples (qui sont classés par *tag*) de ce qui est faisable avec Pygame, voici un petit lien : [Tags - Pygame](#) ↗

Ici vous trouverez la liste de toutes les fonctions fournies par Pygame : [Index - Pygame Documentation](#) ↗

Et maintenant, allons-y!



La conception d'un binding peut être motivée par le fait de profiter des performances offertes par l'utilisation d'un langage bas niveau que l'on ne peut obtenir avec un langage de plus haut niveau. La réutilisation de code éprouvé peut également être une autre raison d'y recourir.

<https://fr.wikipedia.org/wiki/Binding> ↗

i

Une seconde partie traitant d'une utilisation plus poussée de Pygame est en cours de rédaction

1.

Nombre de bibliothèques sont écrites dans des langages proches de la machine comme le C ou le C++. Pour utiliser ces bibliothèques dans un langage de plus haut niveau, il est donc nécessaire de réaliser un binding.

Cinquième partie
A la découverte de Pygame !

V. *A la découverte de Pygame!*

Dans cette partie, nous allons voir comment importer Pygame, créer sa fenêtre, la customiser, y ajouter des images, et bien d'autres choses qui nous seront très utiles pour la suite de ce tutoriel!

1. Avant de commencer...

De manière à être sûr de pouvoir vous garder, chers lecteurs, nous allons d'abord voir quelques petits (ou non) jeux / projets / autre-chose-qui-utilise-Pygame-à-insérer-ici se basant sur cette bibliothèque.

1.1. "A small game made with Pygame"

Quoi de mieux qu'un jeu de plateformes pour commencer cette mise en bouche ? Et bien en voici un !

ÉLÉMENT EXTERNE (VIDEO) —

Consultez cet élément à l'adresse <https://www.youtube.com/embed/aUCyfdzP-i8>.

i

Ce projet a été réalisé par Bram Snijders

1.2. DonkeyPy, un DonkeyKong like

Ce projet a été réalisé par Pinkesh Badjatiya.

ÉLÉMENT EXTERNE (VIDEO) —

Consultez cet élément à l'adresse https://www.youtube.com/embed/apdKl_3m2Vk.

V. A la découverte de Pygame!

1.3. Un moteur de raycasting

Ce projet, bien plus avancé, a été réalisé par klafyvel, et utilise ... Pygame! Mais quelle surprise n'est-ce pas?

Pour faire simple, c'est une vue 3D (avec rotation sur les côtés bien sûr, à la Doom pour les nostalgiques) créée avec Pygame. Ce qui est intéressant ici est le fait que ce projet affiche un environnement 3D, avec une bibliothèque ne permettant de faire **que** de la 2D!

Le projet est disponible ici : <https://github.com/Klafyvel/RayCasting> ↗

Et en voici une video :

ÉLÉMENT EXTERNE (VIDEO) —

Consultez cet élément à l'adresse <https://www.youtube.com/embed/ZCd-GvIVBWM>.

Ahhh, ça met l'eau à la bouche, non?

La motivation est là? Non? Pas grave, on enchaîne quand même!

2. Installer et importer Pygame

Dans ce chapitre, nous allons installer Pygame, puis découvrir comment importer Pygame, et quelques modules (déjà livrés avec Pygame, n'ayez pas peur), ainsi que l'utilité des dits modules.

2.1. Installer Pygame

Allez hop, on met les mains dans le cambouis !

2.1.1. Windows

i

Si vous avez une vieille version de Pygame, voici quelques liens d'installations, mais nous recommandons fortement de passer à Python 3.6

☉ Afficher le contenu masqué

On va passer par `pip` pour installer Pygame. Pour cela, il faut ouvrir un terminal (`Win`+`R`) puis `cmd`, puis (`Entrée`), et on va taper ceci : `python -m pip install pygame`

Si pip vous dit ceci : `Requirement already satisfied (use --upgrade to upgrade): pygame in c:\python34\lib\site-packages`, c'est que Pygame est déjà installé sur votre machine. Sinon si vous obtenez

'python' n'est pas reconnu en tant que commande interne ou externe, un programme exécutable ou un fichier de commandes.

de la part de votre invite de commande, essayez `py -3.6 -m pip install pygame`, c'est que vous devez avoir plusieurs versions de Python installées (ou qu'il n'est pas dans votre `PATH`).

2.1.2. Linux (Debian et dérivés)

Pour les durs qui veulent tout installer à la main :

☉ Afficher le contenu masqué

V. A la découverte de Pygame!

Si comme nous vous êtes fainéant :

```
1 sudo apt-get install -y python3-pip
2 sudo apt-get install -y \
3     python3-numpy libav-tools libsdl-image1.2-dev \
4     libsdl-mixer1.2-dev libsdl-ttf2.0-dev libsmpeg-dev
5     libsdl1.2-dev \
6     libportmidi-dev libswscale-dev libavformat-dev libavcodec-dev \
7     libfreetype6-dev
8 sudo pip3 install -q pygame
```

Les manipulations pour installer Pygame sous Linux ont été testées sur un *L-o-W* (*Linux-on-Windows*), si jamais quelque chose ne fonctionne pas chez vous, n'hésitez pas à créer un sujet sur le forum !

2.2. Import pygame

Ça y est ! Nous y sommes. Le moment où l'on va pouvoir se rendre compte si vous avez bien installé Pygame ou ... si vous devez recommencer son installation

Ouvrez un interpréteur python, et tapez :

```
1 >>> import pygame
```

Si tout se passe bien, Python ne devrait même pas broncher. En revanche, s'il vous sort un magnifique :

```
1 >>> import pygame
2 Traceback (most recent call last):
3   File "<stdin>", line 1, in <module>
4 ImportError: No module named 'pygame'
```

C'est qu'il va vous falloir recommencer l'installation (pensez à regarder ce que pip/apt-get/votre-outil-pour-installer-Pygame a dit en installant Pygame, il y a souvent des traces à propos des erreurs lors de l'installation).

Pour ceux dont l'installation s'est bien passée, on peut passer à la suite !

2.3. Vérifier qu'un module a bien été importé

Parfois, il peut être utile de savoir si un module a été importé correctement.

Bon, on a dit *parfois*. Cela ne veut pas forcément dire qu'il **faut** le faire, juste que c'est une chose à savoir

Par exemple, si votre programme est destiné à tourner sous Linux, système où lors de l'installation de Pygame, certains modules ne sont pas forcément installés (car optionnels), vous pourriez donc avoir des surprises.

Passons donc aux méthodes :

- en appelant `pygame.init()`, celui-ci nous retourne un tuple, par exemple `(6, 0)`. Cela veut dire : 6 modules chargés, 0 échecs. C'est un début de piste pour vérifier si un module n'a pu être chargé, il suffit de tester `pygame.init()[1] == 0` (on ne le fait pas comme ça, directement surtout ! Il faut stocker le retour de `pygame.init()` dans une variable et faire les tests ensuite sur cette variable)



Pygame n'est pas très uniforme dans ses fonctions. En effet certains modules bénéficient de la méthode `get_init()` tandis que d'autres ont la méthode `was_init()`. A utiliser avec prudence dans un bloc `try & except`, donc.

- si l'on veut tester l'initialisation d'un module spécifique, on utilisera plutôt `pygame.module.was_init()` (qui retourne un `bool`)

Et pour initialiser un module à la main, on fera ceci : `pygame.module.init()`

Voilà, maintenant vous savez comment importer Pygame. C'était compliqué pas vrai ?

Allez, on peut passer à la suite !

Contenu masqué

Contenu 1

Pygame pour Python 3.4 : [téléchargement](#) ↗

Pygame pour Python 3.2 : [téléchargement](#) ↗

[Retourner au texte.](#)

Contenu 2

```
1 # On installe les dépendances
2 sudo apt-get install mercurial python3-dev python3-numpy
   libav-tools \
3     libsdl-image1.2-dev libsdl-mixer1.2-dev libsdl-ttf2.0-dev
   libsmpeg-dev \
4     libsdl1.2-dev libportmidi-dev libswscale-dev libavformat-dev
   libavcodec-dev
5
6 # On récupère Pygame depuis son dépôt
7 hg clone https://bitbucket.org/pygame/pygame
8
9 # On le construit et on l'installe
10 cd pygame
11 python3 setup.py build
12
13 sudo python3 setup.py install
14 # On supprime le dossier pygame inutile
15 cd
16 rm -rf pygame
```

[Retourner au texte.](#)

3. Créer une simple fenêtre (personnalisable!)

Dans ce chapitre, nous allons voir comment créer une fenêtre, et quelles possibilités nous sont offertes pour la personnaliser.

3.1. Créons une fenêtre basique

Attention les yeux, c'est très simple!

Voici le code d'une fenêtre basique :

```
1 import pygame
2
3 pygame.init()
4
5 ecran = pygame.display.set_mode((300, 200))
6
7 pygame.quit()
```

i

Si vous avez lancé ce script, ne partez pas! C'est normal si votre fenêtre s'est fermée en moins d'une fraction de seconde.

Détaillons ce code :

ligne 1 : on importe Pygame

ligne 3 : on initialise Pygame, très important! C'est cette ligne qui va charger tous les modules de Pygame (comme **font**, **draw**, **display** ...) et les initialiser

ligne 5 : on crée une fenêtre avec le module **display** de Pygame, en lui passant en paramètre un **tuple** contenant **300** et **200**. Cela veut dire que la fenêtre fera 300 pixels de large sur 200 pixels de haut

ligne 7 : il faut quitter proprement Pygame, cela va libérer toutes nos ressources, nous n'en avons plus besoin

Avec ce code, la fenêtre se ferme toute seule, car Pygame a considéré qu'il avait fini son travail. On ne lui a pas demandé de gérer autre chose, comme des événements (qui pourraient par exemple dire à Pygame : "attends que je clique sur la croix pour fermer ma fenêtre")!

V. A la découverte de Pygame!

Pour cela, il faut demander à Pygame de vérifier s'il y a des événements en cours (touche du clavier appuyée, clic souris ...) et il nous faut les **traiter** par la suite pour agir en conséquence :

```
1 import pygame
2 ##### par la même occasion cela importe pygame.locals dans l'espace
   de nom de Pygame
3
4 pygame.init()
5
6 ecran = pygame.display.set_mode((300, 200))
7
8 continuer = True
9
10 while continuer:
11     for event in pygame.event.get():
12         if event.type == pygame.KEYDOWN:
13             continuer = False
14
15 pygame.quit()
```

Maintenant, lancez votre code, et appuyez sur n'importe quelle touche du clavier. La fenêtre va se fermer !

Alors on reprend son calme, l'excitation de la première fenêtre va passer, ne vous inquiétez pas

ligne 2 : très intéressante ! On ne vous en avait par parlé dans la section précédente, en fait Pygame importe toutes un tas de **constantes**. De manière générale, les noms de tous les événements de Pygame sont importés dans votre programme. Sans ces constantes, nous aurions dû taper le code de chaque événement ! Et oui, car **KEYDOWN** n'est autre qu'un entier (ayant pour valeur 2) ! Tout comme les codes **K_a**, **K_g** ... qui représentent *respectivement* les touches

A et **G**

lignes 10 à 13 : le code le plus intéressant se trouve là. Une boucle for qui appelle une fonction obscure.

Oui oui, c'est bien ça. Dans l'ordre on fait ceci :

- on récupère l'événement que Pygame a capturé que l'on nomme **event** (il se peut qu'il n'y en ai pas, dans ce cas, le code n'est pas exécuté, et la boucle (qui contiendra tout le code d'affichage d'images, qui gèrera les événements ...) continue de boucler)
- dans le **if** maintenant :
 - on regarde si le "type" de l'événement est une entrée clavier
 - si oui, alors on met **continuer** à **False**, donc on quitte la boucle

i

On utilise un **bool** que l'on a appelé **continuer** pour savoir si la boucle doit continuer de tourner ou non.

Pour un début, cela peut paraître compliqué. Mais ne vous inquiétez surtout pas, c'est normal, et nous allons éclairer toutes ces nouvelles notions dans les prochains chapitres

3.2. Personnalisons notre fenêtre

Tout d'abord, nous allons voir quels arguments nous pouvons passer à `pygame.display.set_mode()`, autre que la taille de notre fenêtre.

La documentation de Pygame nous informe de ceci :

☉ Afficher le contenu masqué

On voit que l'on peut passer une résolution (la taille de notre fenêtre) et que si on ne met pas de taille ou que l'on met `(0, 0)` comme taille, la fenêtre fera la taille de votre écran.

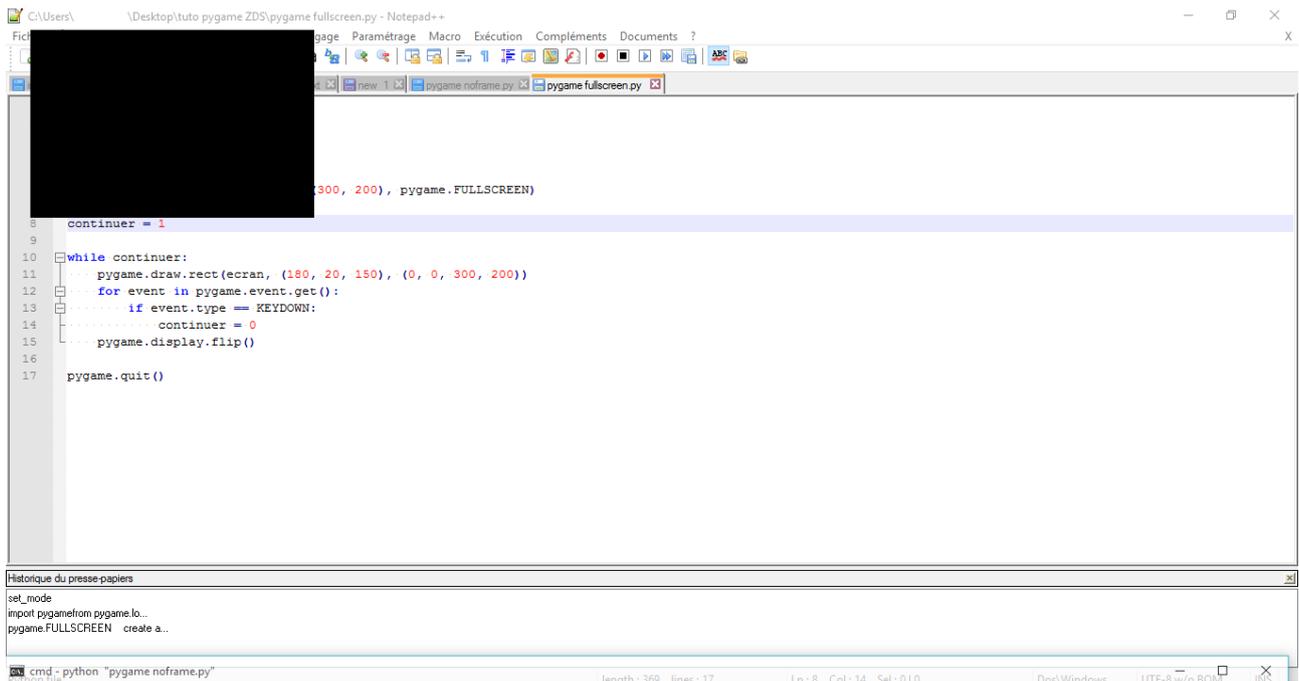
La documentation nous informe aussi qu'il est préférable de ne pas changer l'argument `depth`, car cela pourrait ralentir votre jeu, Pygame pouvant essayer de créer des couleurs n'existant pas.

Et enfin nous arrivons aux `flags`. Ce sont des arguments que nous *pouvons* passer à `pygame.display.set_mode()`, mais ils ne sont en aucun cas obligatoires.

3.2.0.1. Les flags

- `pygame.FULLSCREEN`
 - cela permet de créer une fenêtre en plein écran (donc plus de bordures, on ne pourra pas, à ce stade du tutoriel, fermer la fenêtre. Attendez que l'on ai vu les événements avant d'utiliser ce flag!)
- `pygame.DOUBLEBUF`
 - à utiliser pour un usage conjoint à OpenGL, nous ne nous s'en servons pas dans ce tutoriel
- `pygame.HWSURFACE`
 - seulement pour le mode plein écran, cela utilise une accélération matérielle
- `pygame.OPENGL`
 - crée une fenêtre de rendu compatible avec OpenGL
- `pygame.RESIZABLE`
 - la fenêtre sera redimensionnable
- `pygame.NOFRAME`
 - la fenêtre n'aura pas de bordures, il sera donc impossible de la déplacer

V. A la découverte de Pygame!



```
pygame.display.set_mode((300, 200), pygame.FULLSCREEN)
continuer = 1
while continuer:
    pygame.draw.rect(ecran, (180, 20, 150), (0, 0, 300, 200))
    for event in pygame.event.get():
        if event.type == KEYDOWN:
            continuer = 0
    pygame.display.flip()
pygame.quit()
```

FIGURE 3.1. – NOFRAME



FIGURE 3.2. – Une fenêtre avec le flag FULLSCREEN (oui, le code a été modifié pour que l'on puisse voir la fenêtre :P)

On vous doit sûrement des explications pour le mode `FULLSCREEN`, du moins si vous avez essayé (alors que l'on vous l'avait défendu). En mettant `pygame.display.set_mode((300, 200), pygame.FULLSCREEN)` vous aurez des surprises, car votre fenêtre prendra tout l'écran. Et oui, ici, Pygame va considérer que votre écran a une résolution de 300 sur 200.

V. A la découverte de Pygame!

Pour obtenir une fenêtre comme nous, donc ne pas avoir ce "petit" problème en mode `FULLSCREEN`, il faut utiliser l'astuce avec `pygame.display.set_mode((0, 0), pygame.FULLSCREEN)`, qui veut dire : "crée une fenêtre qui fait toute la taille de l'écran, et ce en `FULLSCREEN`". Sauf que vous ne voudrez peut-être pas utiliser tout l'écran, donc il faut créer une "sous surface" que l'on considérera comme notre fenêtre (oui, c'est tout à fait possible, étant donné que chacun de nos objets `ecran` sont eux même des images).

Voici donc le code que l'on a utilisé :

```
1 import pygame
2
3 pygame.init()
4
5 ekran = pygame.display.set_mode((0, 0), pygame.FULLSCREEN)
6
7 continuer = True
8
9 while continuer:
10     # ici on crée un rectangle de couleur rose, en x=0, y=0 et de
11     # taille 300 sur 200
12     # nous verrons plus tard comment faire plus en détail :)
13     pygame.draw.rect(ekran, (180, 20, 150), (0, 0, 300, 200))
14     for event in pygame.event.get():
15         if event.type == pygame.KEYDOWN:
16             continuer = False
17     # ici on actualise l'écran, car on a affiché un rectangle rose,
18     # et on veut qu'il soit
19     # visible. Si l'on avait pas mit cette instruction, on n'aurait
20     # jamais vu le rectangle !
21     pygame.display.flip()
22
23 pygame.quit()
```

Ici le rectangle rose sera votre "écran", mais vous pourrez dessiner à côté bien entendu (ici, en `FULLSCREEN`, ce n'est qu'un cadre si vous voulez).

3.2.0.2. Changer le titre et l'icône de notre fenêtre

Pour changer le titre de votre fenêtre, il faut jouer avec les méthodes offertes par le module `display`, comme ceci :

```
1 pygame.display.set_caption("Mon super titre de Ouf !")
```

Et pour changer l'icône de votre fenêtre, ce sera :

V. A la découverte de Pygame!

```
1 pygame.display.set_icon(image)
```

Ici, `image` est un objet Surface de Pygame. Autrement dit, il vous faut savoir charger une image pour changer l'icône de votre fenêtre

Là on peut se demander si notre icône doit être au format `ico` pour Windows, et `xbm` pour Linux. Sauf que ... sur ce coup là, Pygame réconcilie tout le monde, c'est juste une image!

En petite avant-première, voici comment en charger une :

```
1 image = pygame.image.load("monimage.png").convert()
```

Voilà! Maintenant, vous savez comment créer une fenêtre, lui donner un titre, changer son icône, et bien d'autres choses! Passons à la suite matelot!

Contenu masqué

Contenu 1

`pygame.display.set_mode()` Initialize a window or screen for display `set_mode(resolution=(0,0), flags=0, depth=0) -> Surface`

This function will create a display Surface. The arguments passed in are requests for a display type. The actual created display will be the best possible match supported by the system.

The resolution argument is a pair of numbers representing the width and height. The flags argument is a collection of additional options. The depth argument represents the number of bits to use for color.

The Surface that gets returned can be drawn to like a regular Surface but changes will eventually be seen on the monitor.

If no resolution is passed or is set to (0, 0) and pygame uses SDL version 1.2.10 or above, the created Surface will have the same size as the current screen resolution. If only the width or height are set to 0, the Surface will have the same width or height as the screen resolution. Using a SDL version prior to 1.2.10 will raise an exception.

It is usually best to not pass the depth argument. It will default to the best and fastest color depth for the system. If your game requires a specific color format you can control the depth with this argument. Pygame will emulate an unavailable color depth which can be slow.

When requesting fullscreen display modes, sometimes an exact match for the requested resolution cannot be made. In these situations pygame will select the closest compatible match. The returned surface will still always match the requested resolution.

The flags argument controls which type of display you want. There are several to choose from, and you can even combine multiple types using the bitwise or operator, (the pipe “|” character). If you pass 0 or no flags argument it will default to a software driven window. Here are the display flags you will want to choose from :

`pygame.FULLSCREEN` create a fullscreen display `pygame.DOUBLEBUF` recommended for `HWSURFACE` or `OPENGL` `pygame.HWSURFACE` hardware accelerated, only in `FULLSCREEN` `pygame.OPENGL` create an OpenGL renderable display `pygame.RESIZABLE` display window should be sizeable `pygame.NOFRAME` display window will have no border or controls

http://www.pygame.org/docs/ref/display.html#pygame.display.set_mode ↗

[Retourner au texte.](#)

4. Afficher des images

Et maintenant, passons au plus important dans un jeu video : les images !

4.1. Lire une image

Voici le tout petit code qui permet de lire une image (attention, l'image n'est **que** lue ! Il faudra ensuite la convertir dans un format lisible par Pygame !) :

```
1 image = pygame.image.load("image.png")
```

Pygame supporte les formats d'image suivant :

- JPG
- PNG
- GIF (non-animé)
- BMP
- PCX
- TGA (décompressé)
- TIF
- LBM (et PBM)
- PBM (et PGM, PPM)
- XPM

Source : <http://www.pygame.org/docs/ref/image.html> ↗

i

Nous verrons par la suite que l'on peut également charger une image en utilisant le module **Surface** de Pygame. Mais pour ne pas trop complexifier cette partie, ceci vous sera présenté dans le chapitre adéquat.

4.2. Convertir nos images

Une image c'est bien, mais si on essaye d'en afficher une, on aura de jolis *warning* à propos de la transparence / du format de l'image. Il nous faut donc la convertir dans un format facilement lisible et manipulable par Pygame.

V. A la découverte de Pygame!

Pour cela, on peut faire comme ceci (ce code ne conserve pas la transparence de l'image, il se peut donc que vous ayez des artefacts en l'affichant) :

```
1 image = image.convert() # une copie est renvoyée, ce n'est pas
   l'image originelle qui est modifiée, attention donc !
```

Il faut auparavant que votre image ait été chargée, comme montré auparavant.

Si votre image contient de la transparence, pour la garder, on fera ainsi :

```
1 image = image.convert_alpha() # même remarque qu'avec
   image.convert()
```

?

Mais dis donc ! C'est un peu lourd de devoir écrire `image = pygame.image.load(chemin)` puis `image = image.convert()` !

Effectivement oui.

C'est pour cela que l'on peut faire plus simple, et surtout plus rapide :

```
1 image = pygame.image.load(chemin).convert()
```

Nous, on met toujours `convert_alpha()`. Au moins, si on veut rajouter de la transparence plus tard dans l'image, il n'y aura pas besoin de retoucher au code

i

En plus, en utilisant `convert_alpha()`, le rendu de pixels *alphas*, autrement dit transparents, est optimisé par Pygame. Magique non ?

4.3. Afficher une image sur notre belle fenêtre !

Un petit rappel sur le code basique pour créer une fenêtre :

```
1 import pygame
2
3 pygame.init()
4
5 ecran = pygame.display.set_mode((600, 600))
6
```

V. A la découverte de Pygame!

```
7 continuer = True
8
9 while continuer:
10     for event in pygame.event.get():
11         if event.type == pygame.KEYDOWN:
12             continuer = False
13
14 pygame.quit()
```

Pour afficher une image, on va utiliser la méthode `blit` de `Surface`. Pour faire court, `Surface` est une classe de Pygame, mais est contenu dans le module `surface` de Pygame, qui est, fort heureusement, importé automatiquement avec notre bibliothèque favorite.

i

N'oubliez jamais que **toutes** nos images sont des objets `Surface`, y compris votre fenêtre

Donc on pourrait afficher une image sur une autre puis encore sur une autre ... etc.

Pour cela, rien de plus simple :

```
1 ecran.blit(image, position)
```

`image` est une image convertie dans un format lisible par Pygame, et `position` est un `tuple` (ou une `list`, ça marchera aussi) de 2 entiers représentant la position de l'image dans notre fenêtre.

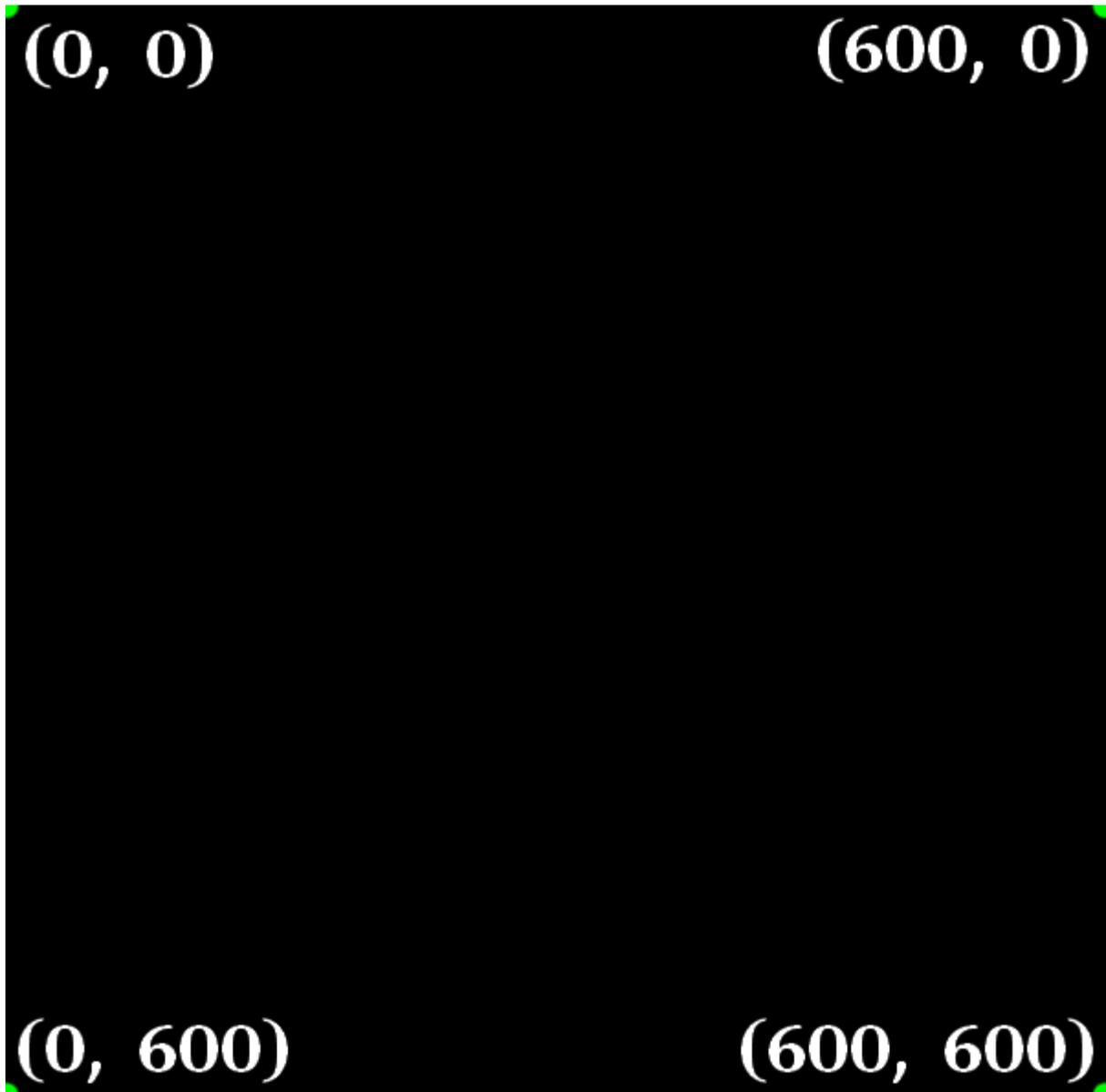
Cela peut être long de mettre `position = (0, 50)` puis `ecran.blit(image, position)`, on peut donc condenser et mettre directement le `tuple` en argument. Par contre dans un jeu où le personnage se déplace, il **faut** garder la position de l'image de notre personnage dans une variable!

?

Et quel est le point de départ ? Je veux dire, où se situe (0, 0) ?

Très bonne question ! Voici une petite image pour mieux comprendre :

V. A la découverte de Pygame!



Comme exercice, essayez de faire un code pour afficher cette image :

V. A la découverte de Pygame!



Si vous avez essayé d'afficher cette image, mais qu'elle ne s'est pas affichée, c'est bien ! Et oui, on n'avait pas précisé qu'il fallait ajouter `pygame.display.flip()` juste avant la fin de la boucle `while` pour actualiser votre fenêtre

L'exemple complet pour afficher une image (pour ceux qui sont au fond et qui n'entendent pas) :

```
1 import pygame
2
3 pygame.init()
4
5 ecran = pygame.display.set_mode((300, 200))
6 image = pygame.image.load("image.png").convert_alpha()
7
8 continuer = True
9
10 while continuer:
11     ecran.blit(image, (0, 50))
12     for event in pygame.event.get():
13         if event.type == pygame.KEYDOWN:
14             continuer = False
15     pygame.display.flip()
16
17 pygame.quit()
```

Ce qui nous donne bien ceci :

V. A la découverte de Pygame!

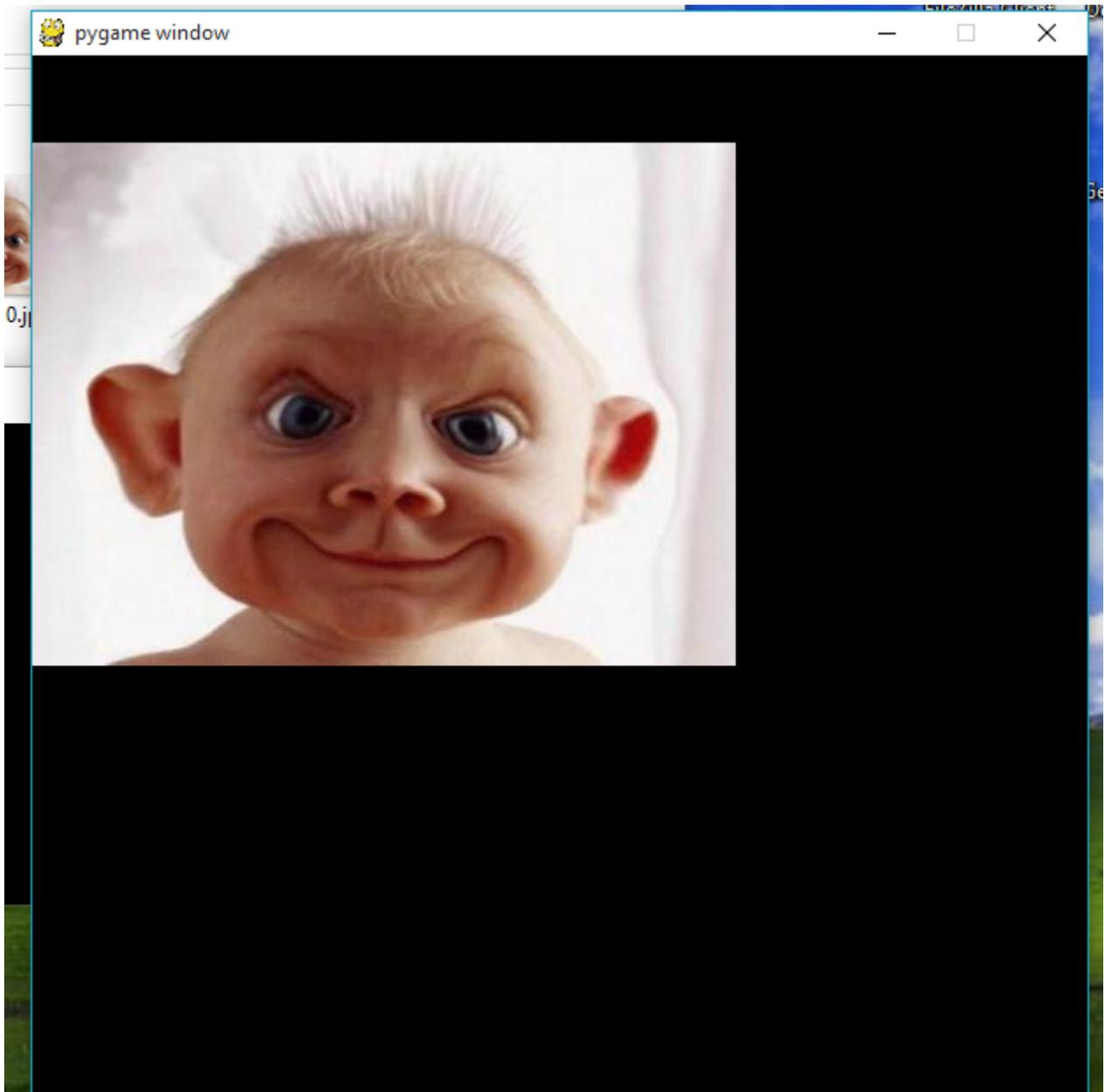


FIGURE 4.1. – Elle est belle cette image, hein ?

Voilà !

Maintenant vous savez tout sur les images (ou presque, en réalité on ne vous a pas tout dit, mais nous verrons cela dans la partie 2, chapitre sur le module `Surface`)!

5. La gestion des événements

Dans ce nouveau chapitre, on va pouvoir passer aux choses intéressantes : les interactions avec l'utilisateur par le biais des événements !

Allez hop, c'est parti !

5.1. Qu'est-ce qu'un événement ?

En informatique, un événement peut être une entrée clavier (soit l'appui soit le relâchement d'une touche), le déplacement de votre souris, un clic (encore une fois, appui ou relâchement, qui seront traités comme deux événements distincts). Un bouton de votre joystick peut aussi engendrer un événement, et même la fermeture de votre fenêtre est considéré comme un événement !

Pour Pygame, un événement est représenté par un `type` et divers autres attributs que nous allons détailler dans ce chapitre.

De plus, il faut savoir que chaque événement créé est envoyé sur une **file** (ou *queue*), en attendant d'être traité. Quand un événement entre dans cette *queue*, il est placé à la fin de celle-ci. Vous l'aurez donc compris, le premier événement transmis à Pygame sera traité en premier ! Cette notion est très importante, puisque nous allons nous en servir sous peu !

Ce type de *queue* est dit `FIFO` [↗](#) .

5.2. Comment les capturer ?

Comme on vous l'avait montré succinctement dans les chapitres précédents, on utilise le module `event` de Pygame.

Voici ce que nous dit la documentation à propos de ce module :

☞ Afficher le contenu masqué

Et comme on peut le voir, le module `event` ne permet pas **que** d'intercepter des événements. Il nous permet aussi de créer des événements. Et même d'en bloquer !

?

Hein ? Mais pourquoi voudrait-on bloquer un événement ?

V. A la découverte de Pygame!

Et bien, prenons l'exemple d'une boîte de dialogue. On va récupérer uniquement les chiffres et les lettres du clavier, et les afficher les unes à la suite des autres. Mais si notre utilisateur appuie sur `F8` ou l'on ne sait pas quelle autre touche de fonction ? Que va faire notre programme ? Il va essayer d'afficher la valeur de la touche tapée (donc `A` affichera 'A'). Mais quelle valeur est affectée à notre touche de fonction pour Pygame ? Aucune ! Notre programme planterait avec un joli `traceback`.

Voyons maintenant les différentes méthodes qui nous sont offertes pour capturer ces événements :

- `pygame.event.get()`
 - obtient **les** événements qui attendent sur la *queue*. S'il n'y en a pas, retourne un ensemble vide
- `pygame.event.poll()`
 - obtient **un seul** événement qui attend sur la *queue*. S'il n'y en a pas, retourne un ensemble vide
- `pygame.event.wait()`
 - **attend** qu'un événement arrive sur la *queue* et le retourne

S'il y a autant de méthodes différentes pour un seul but, c'est assez simple en fait (même si cela peut paraître tordu au premier abord). Regardons de plus près ce que font ces méthodes.

`pygame.event.get()` retourne une `Eventlist` d'après la documentation. Prenons le cas où l'on aimerait pouvoir gérer `CTRL + C` dans un programme. C'est là que `pygame.event.get()` nous sera très utile ! Car il va intercepter tous les événements, donc deux appuis de touches simultanés !

`pygame.event.poll()`, lui, retourne seulement une instance de `Eventtype`, donc un seul événement depuis la *queue*

`pygame.event.wait()` retourne **aussi** une instance de `Eventtype`, donc toujours un événement depuis la *queue*

?

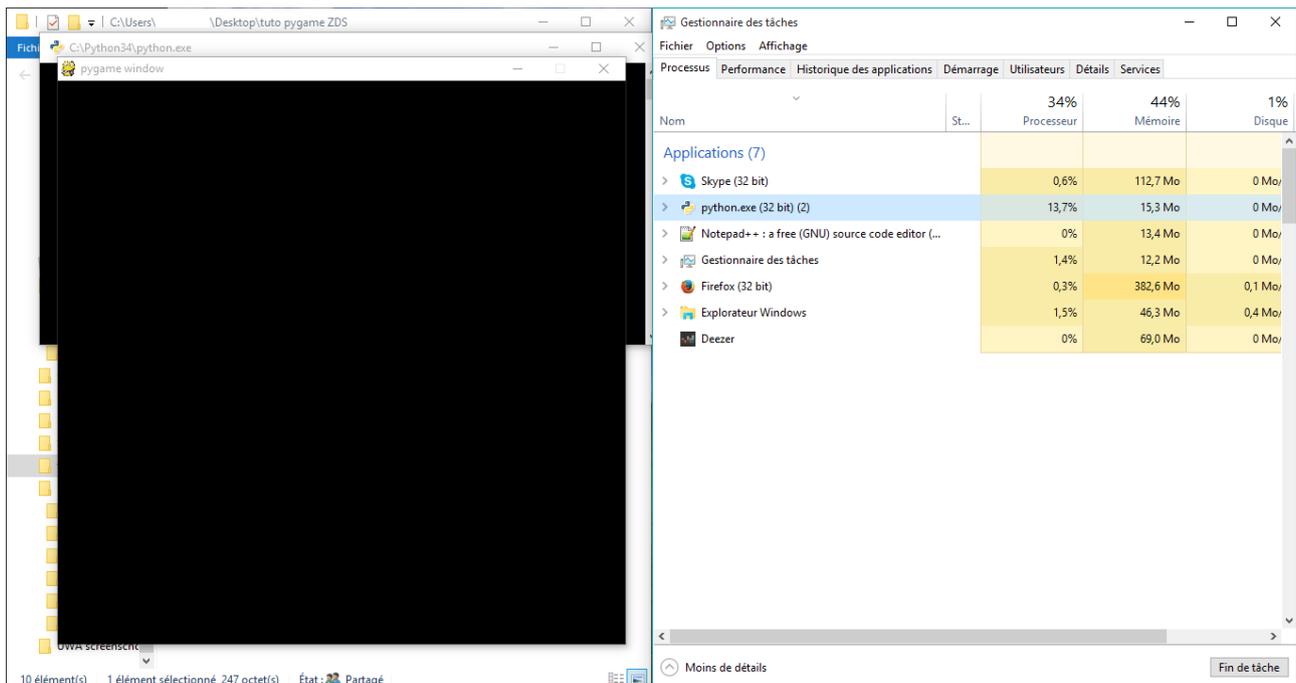
Pourquoi Pygame dispose de deux méthodes faisant exactement la même chose ?

Hé hé ! Ces méthodes ne font pas *totalemment* la même chose, elles ne font que retourner le même type d'objet

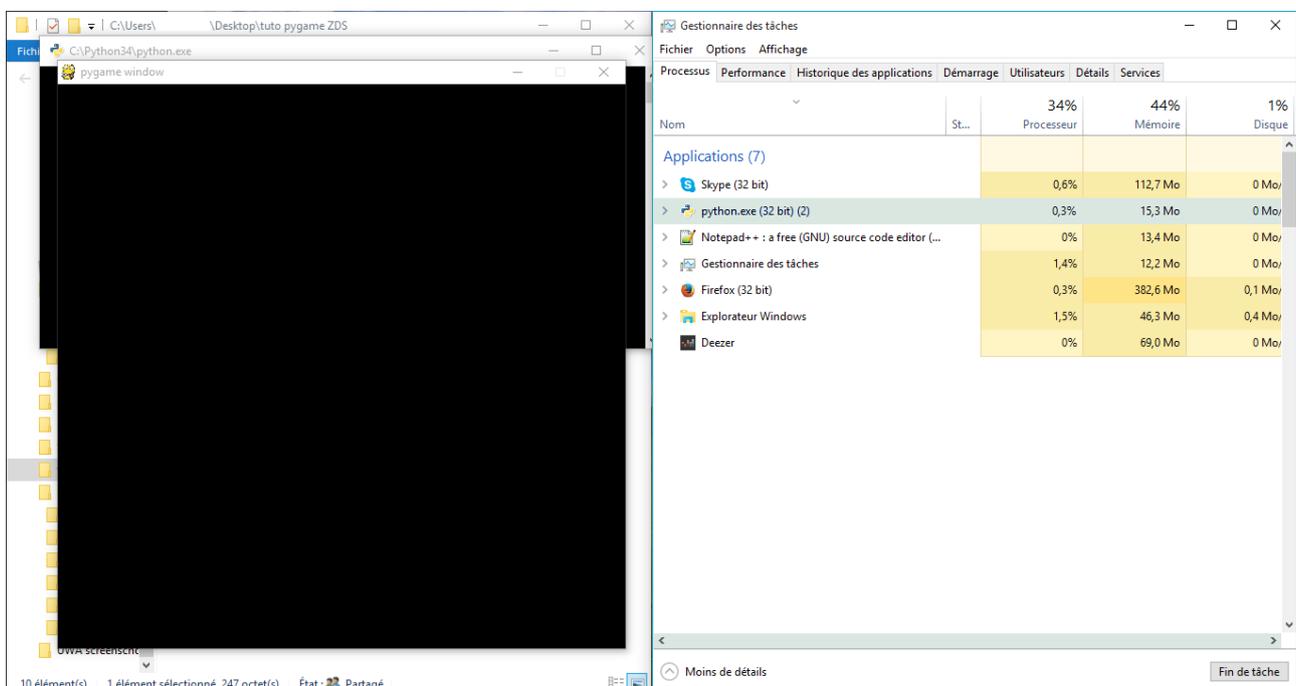
Si on se penche sur `pygame.event.poll()`, on verra que celle-ci va occuper notre processeur, même si aucuns événements ne se sont produits, en tentant d'en récupérer un (un événement) à chaque fois que l'on lui demande !

Regardons cette capture d'écran pour mieux comprendre :

V. A la découverte de Pygame!



Alors que `pygame.event.wait()`, lui va faire 'dormir' notre programme en attendant qu'un événement arrive sur la *queue*! En conséquence, s'il n'y a pas d'événement(s) sur la *queue*, et bien ... il va tout simplement bloquer le **thread** principal, autrement dit votre application va "s'arrêter" pendant une durée indéterminée ... Pas pratique si on veut mettre à jour la position de nos ennemis à chaque **frame**, par exemple.



Vous comprenez la différence maintenant ?

Si vous ne voulez pas utiliser tout le processeur de l'utilisateur, alors utilisez `pygame.event.wait()`, sinon si vous vous en fichez, prenez `pygame.event.poll()`! Et pour ceux qui souhaitent

V. A la découverte de Pygame!

intercepter plusieurs événements (ou non, on peut très bien avoir qu'un seul événement en attente sur la *queue*), il y a `pygame.event.get()`. C'est généralement celui-ci que l'on utilise, car il permet de tout couvrir.

Il faut aussi savoir que l'on ne va pas les utiliser de la même manière!

Utilisation de `pygame.event.get()` :

```
1 for event in pygame.event.get():
2     if event.type == KEYDOWN: ...
```

Utilisation de `pygame.event.poll()` (similaire à `pygame.event.wait()`) :

```
1 event = pygame.event.poll()
2 if event.type == KEYDOWN: ...
```

?

Attends une seconde ... `pygame.event.get()` retourne un itérable et pas `pygame.event.poll()` ainsi que `pygame.event.wait()` ?

Oui, c'est tout à fait ça!

5.3. Les types d'événements disponibles

Allez, on part faire du shopping!

Voyons ce que l'on peut mettre dans notre panier :

```
QUIT
ACTIVEEVENT
KEYDOWN
KEYUP
MOUSEMOTION
MOUSEBUTTONUP
MOUSEBUTTONDOWN
JOYAXISMOTION
JOYBALLMOTION
JOYHATMOTION
JOYBUTTONUP
JOYBUTTONDOWN
VIDEORESIZE
VIDEOEXPOSE
USEREVENT
```

V. A la découverte de Pygame!

<http://www.pygame.org/docs/ref/event.html> ↗

Ça en fait peu en fait, non ? En fait, ici ce ne sont que les **types** des différents événements que Pygame peut nous capturer.

On va quand même vous les détailler, car nous sommes des gentils, hein ? (Dîtes oui en hochant la tête)

Donc, reprenons.

QUIT, c'est lui qu'on utilise pour gérer l'événement de fermeture de notre fenêtre, donc quand l'utilisateur clique sur la croix. Il ne renvoie rien.

Concernant **ACTIVEEVENT**, Pygame nous indique ceci :

The window will receive pygame.ACTIVEEVENT events as the display gains and loses input focus.

<https://www.pygame.org/docs/ref/display.html> ↗

Donc le type d'événement **ACTIVEEVENT** arrive sur la *queue* uniquement quand votre fenêtre perd ou regagne le focus de votre souris. Il renvoie le gain (**gain**), c'est à dire si vous avez perdu ou gagné le focus, et son état (**state**), qui veut dire soit : la fenêtre est visible, la fenêtre est *icônifiée* dans la barre des tâches

KEYDOWN et **KEYUP**, on va les traiter ensemble, ils sont presque de la même famille.

Un événement de type **KEYDOWN** est placé sur la *queue* dès que vous appuyez sur une touche (donc au point où l'on en est dans l'écriture de ce tutoriel, on a placé plus d'une dizaine de milliers d'événements de ce type sur la *queue* de nos navigateurs). Il retourne le nom unicode (**unicode** ; donc 'b' par exemple), **key**, le code de la touche - par exemple le code de la touche **X** est 120, mais on tapera **K_x**, plus simple non ? Il retourne aussi **mod**, c'est un *modifier*, il indique quelles touches de contrôle sont actuellement actives (**CTRL**, **ALT**, **SHIFT** ... etc).

Alors qu'un événement de type **KEYUP** ne sera placé sur la *queue* que quand vous aurez relâché une touche ! Celui-là nous renverra uniquement le code de la touche relâchée, et le *modifier*.

?

Alors autant utiliser tout le temps **KEYUP** ! En plus c'est plus court que **KEYDOWN**

Et là je dis carton rouge pour le monsieur au premier rang !

Car un événement de type **KEYDOWN** est extrêmement utile ! Prenons l'exemple d'un jeu, puisque nous allons en créer un petit très bientôt. Votre personnage, vous **voulez** (dîtes pas le contraire, c'est certain) qu'il se déplace **tant que** vous appuyez sur **Z** / **Q** / **S** / **D**, pas vrai ? Alors c'est **là** que ce type d'événement nous sera utile.

Mais à contrario, le type d'événement **KEYUP**, lui, nous sera utile quand on appuie sur la touche pour faire sauter notre personnage ! On ne voudrait tout de même pas qu'il puisse sauter à l'infini, même en étant dans les airs, hein ? De cette manière, un seul appui à la fois, et on devra vérifier que notre personnage n'est pas en l'air avant de le faire sauter.

MOUSEMOTION est un événement produit lorsque vous déplacez votre souris (**uniquement** lors de son déplacement). Un événement de type **MOUSEMOTION** nous renverra la position

V. A la découverte de Pygame!

actuelle de la souris (**pos**), le mouvement relatif (**rel**) effectué depuis le dernier événement **MOUSEMOTION** (donc de combien de pixels, en x et y, notre souris s'est déplacée depuis son ancienne position), et les boutons actuellement enfoncés (**buttons**). **rel** et **pos** étant des **tuples** de deux éléments, une abscisse et une ordonnée.

MOUSEBUTTONUP et **MOUSEBUTTONDOWN** font la même chose sauf que ici encore, l'un est appelé dès qu'on appuie sur un bouton, et l'autre quand on relâche un bouton! Et tous deux renverront **pos** (un **tuple** de deux éléments également, l'abscisse et l'ordonnée), la position actuelle de la souris dans votre fenêtre, et **button**, les boutons enfoncés / relâchés. **button** prend la valeur 1 si on clic avec le bouton gauche, 2 pour le droit, 3 pour un clic molette, 4 pour un déplacement de la molette vers le haut, et 5 pour un déplacement de la molette vers le bas.

Tous les types d'événements commençant par **JOY** sont mis sur la *queue* quand vous générez un événement avec un joystick (encore faut il en connecter un)



Une manette

JOYAXISMOTION est appelé quand vous déplacez un des deux axes de votre manette, et renvoie **axis**, l'axe (0 : axe horizontal, 1 : axe vertical), **joy**, le joystick qui a déclenché l'événement, et **value**, -1 (haut ou gauche) ou +1 (bas ou droite)

JOYBALLMOTION est appelé quand vous bougez une des **ball** de votre manette, et renverra **joy**, le joystick ayant créé cet événement, **ball**, la **ball** qui a bougé, et **rel**, le mouvement relatif depuis le dernier mouvement de cette **ball**

JOYHATMOTION est appelé quand vous bougez un des **hat** de votre manette (si elle en a, hein) et renverra comme les précédents **joy**, **hat** et **value** (**hat** : le **hat** ayant bougé, **value** : -1 ou +1 pour "haut" ou "bas")

JOYBUTTONUP et **JOYBUTTONDOWN** renvoi la même chose, **joy**, et **button** (le bouton qui a déclenché l'événement), mais l'un est appelé à l'appuie, l'autre au relâchement du dit bouton

Voilà ce que nous dit Pygame à propos de **VIDEORESIZE** :

If the display is set with the `pygame.RESIZABLE` flag, `pygame.VIDEORESIZE` events will be sent when the user adjusts the window dimensions.

V. A la découverte de Pygame!

<https://www.pygame.org/docs/ref/display.html> ↗

Donc cet événement n'est mit sur la *queue* que si vous avez créé votre fenêtre avec ce fameux flag **RESIZABLE**

Et pour ce qu'il en est de **VIDEOEXPOSE**, il nous indique ceci, dans son obscure dialecte ... :

Hardware displays that draw direct to the screen will get pygame.VIDEOEXPOSE events when portions of the window must be redrawn.

<https://www.pygame.org/docs/ref/display.html> ↗

Celui-là ne sera donc pas très utile, car il est déclenché uniquement quand vous devez réactualiser une portion de votre écran

Quant à **USERVENT**, lui, est le type de tous les événements que vous, en tant que programmeur, pouvez créer.

5.4. Les événements disponibles!

Ouff, vous y êtes arrivé.

Si vous avez réussi à suivre tout ce que l'on a dit, bravo!

Maintenant que nous avons vu quels types d'événement sont disponibles, et comment les capturer, on va voir comment capturer ... ~~un Pokémon!~~ un événement *complet*, donc type et attributs inclus.

Vous vous rappelez comment capturer un événement? Super!

Pour les poissons rouges, voici le code :

```
1 for event in pygame.event.get():
2     if event.type == un_type:
3         une_fonction_a_executer()
```

Pour savoir sur quelle touche on a tapé, on y arrive. Mais avant! Il va nous falloir vous assommer avec ce qui va devenir votre livre de chevet pour fanatique de pygame (quel titre honorifique!): la documentation.

Et même plus précisément, la table des *keys* disponibles, que voici rien que pour vous en exclusivité! (pour une consultation à part : [lien](#) ↗)

👁️ Afficher le contenu masqué

Alors maintenant voyons comme on ferait pour capturer un appuie sur la touche **J** par exemple :

V. A la découverte de Pygame!

```
1 for event in pygame.event.get():
2     if event.type == pygame.KEYDOWN:
3         # ici on suppose que vous avez fait un simple import pygame
4         if event.key == pygame.K_j:
5             une_action()
```

Tout bête!

Quand on a un `event` du type que l'on veut traiter, on doit ensuite traiter ses attributs pour l'identifier, et le tour est joué!

5.5. Entrainons-nous avec les événements!

Et maintenant voici quelques exercices, que l'on va aussi coder bien entendu

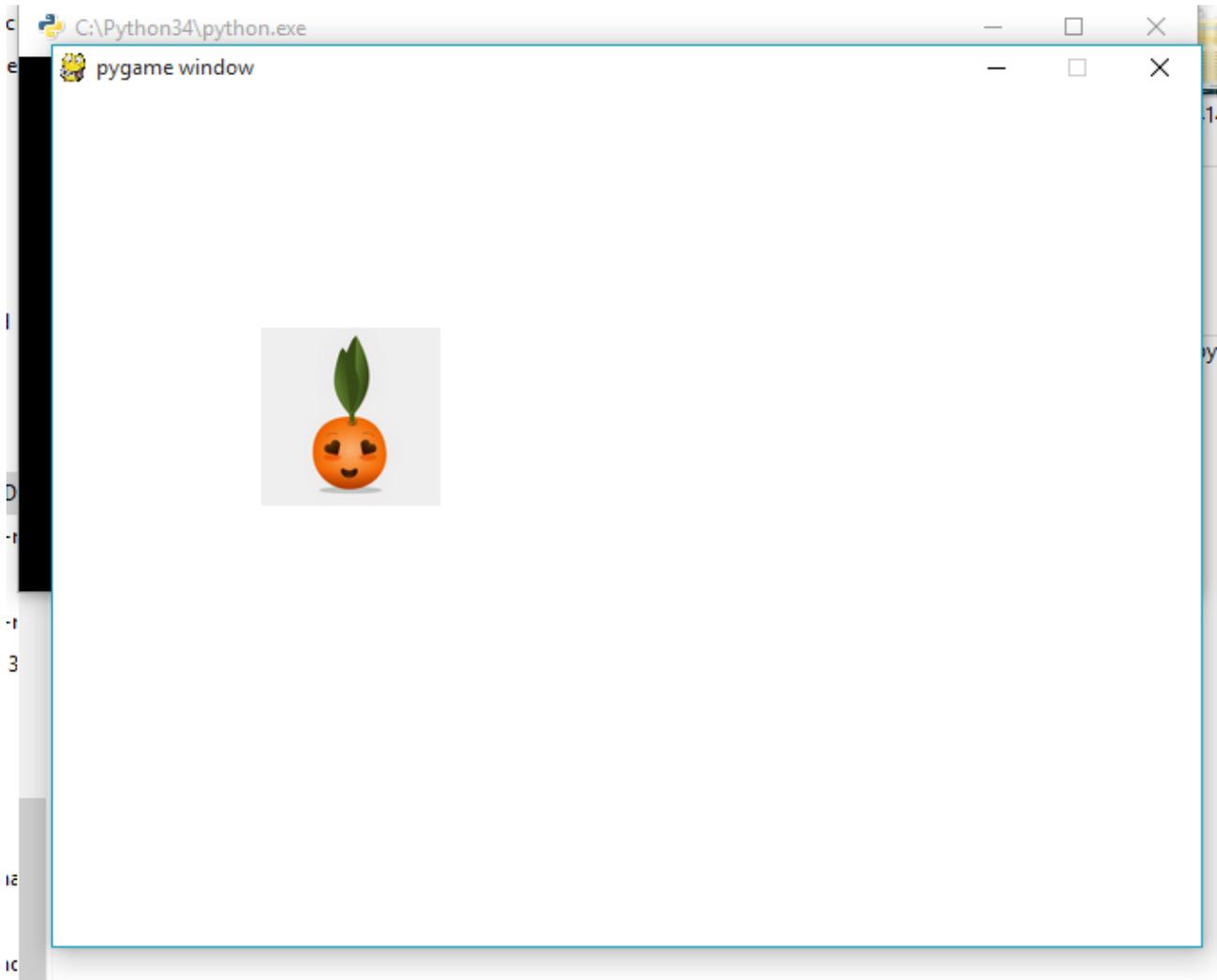
5.5.0.0.1. Premier exercice Coder un script pour créer une fenêtre, puis la détruire si l'on appuie sur `F`.

👁 Afficher le contenu masqué

5.5.0.0.2. Deuxième exercice Coder un script pour déplacer Clem à la souris (Clem doit toujours suivre la souris).

👁 Afficher le contenu masqué

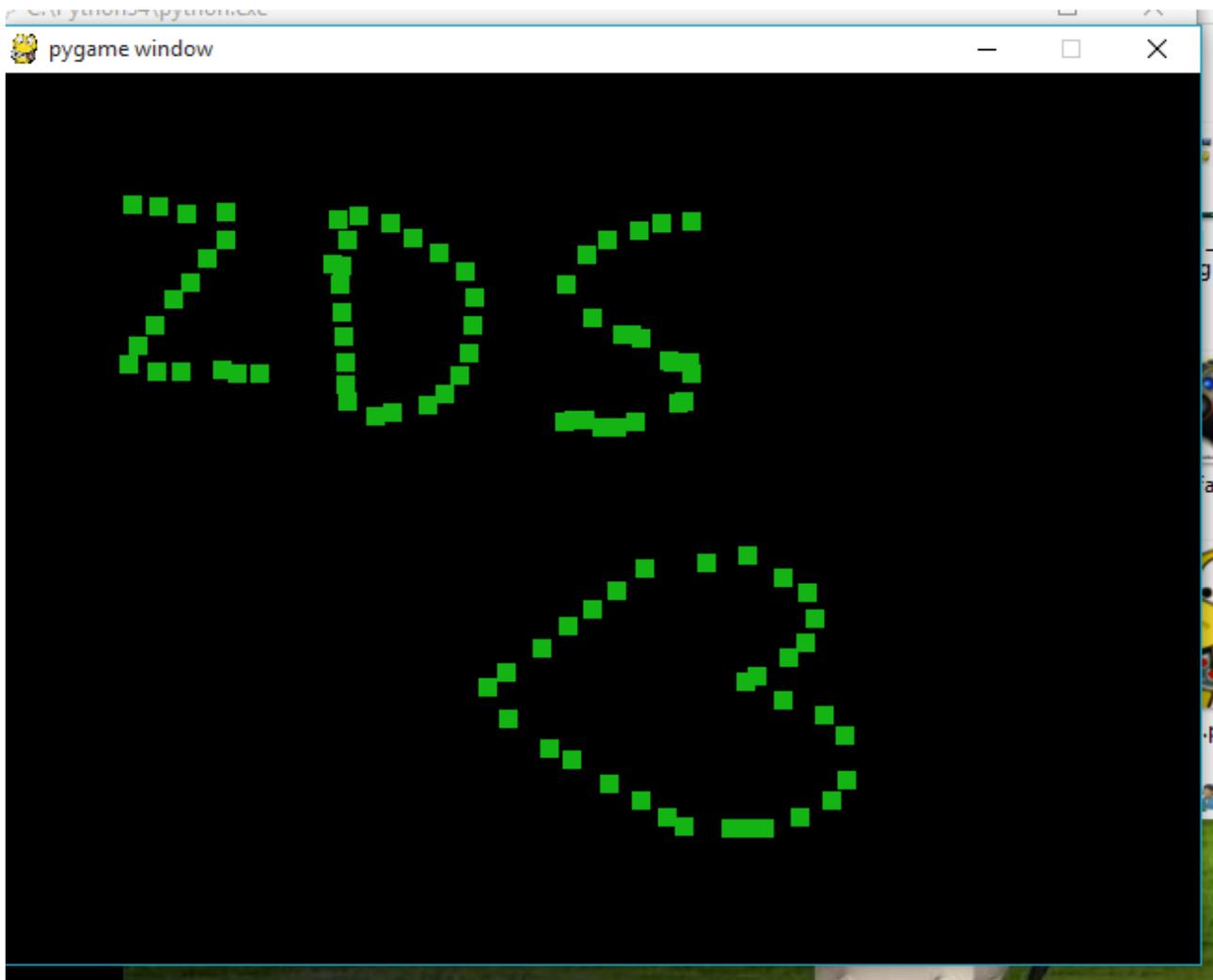
V. A la découverte de Pygame!



Et voilà! Maintenant, Clem suit votre souris!

5.5.0.0.3. Troisième exercice Coder un script pour dessiner sur l'écran au relâchement d'un bouton de la souris.

👁️ Afficher le contenu masqué



Regardez ce beau dessin!

5.5.0.1. Des exercices facultatifs

Ici on vous allez trouver une liste d'exercices, à faire ou pas, ça c'est vous qui choisissez.

Seule différence : vous êtes sans filet ! Si vous avez un problème avec votre code, il faudra ouvrir la documentation ou demander sur le forum si vraiment vous n'y arrivez pas

- une class `Event` pour simplifier la gestion des événements, et pouvoir par exemple taper ce code :

```
1 for event in my_event_listener():
2     if event == (MOUSEBUTTONDOWN, 1):
3         # si on a un clic souris enfoncé avec le bouton n°1 ...
4     if event == (KEYDOWN, K_t):
5         # si on a une key down avec la touche t ...
```

2. c'est un système universel pour la notation des couleurs. Pour plus d'informations, voir : [Wikipédia](#)

V. A la découverte de Pygame!

Après, à vous d'ajouter vos méthodes pour rendre cette gestion encore plus simple (ou pas)!

- une classe `Joystick` pour gérer plus facilement un joystick, vous avez carte blanche pour en faciliter l'utilisation!
- carrément une surcouche à la gestion des événements (tous, absolument tous) pour rendre l'usage plus orienté objet, à la [SFML](#) pour ceux qui connaissent

Et voilà! Maintenant vous savez comment gérer différents types d'événements, nous allons pouvoir passer au TD!

Contenu masqué

Contenu 1

```
pygame.event
pygame module for interacting with events and queues
pygame.event.pump — internally process pygame event handlers
pygame.event.get — get events from the queue
pygame.event.poll — get a single event from the queue
pygame.event.wait — wait for a single event from the queue
pygame.event.peek — test if event types are waiting on the queue
pygame.event.clear — remove all events from the queue
pygame.event.event_name — get the string name from an event id
pygame.event.set_blocked — control which events are allowed on the queue
pygame.event.set_allowed — control which events are allowed on the queue
pygame.event.get_blocked — test if a type of event is blocked from the queue
pygame.event.set_grab — control the sharing of input devices with other applications
pygame.event.get_grab — test if the program is sharing input devices
pygame.event.post — place a new event on the queue
pygame.event.Event — create a new event object
pygame.event.EventType — pygame object for representing SDL events
```

<http://www.pygame.org/docs/ref/event.html>

[Retourner au texte.](#)

V. A la découverte de Pygame!

Contenu 2

KeyASCII	ASCII	Common Name
K_BACKSPACE	\b	backspace
K_TAB	\t	tab
K_CLEAR		clear
K_RETURN	\r	return
K_PAUSE		pause
K_ESCAPE	^[]	escape
K_SPACE		space
K_EXCLAIM	!	exclaim
K_QUOTEDBL	”	quotedbl
K_HASH	#	hash
K_DOLLAR	\$	dollar
K_AMPERSAND	&	ampersand
K_QUOTE		quote
K_LEFTPAREN	(left parenthesis
K_RIGHTPAREN)	right parenthesis
K_ASTERISK	*	asterisk
K_PLUS	+	plus sign
K_COMMA	,	comma
K_MINUS	-	minus sign
K_PERIOD	.	period
K_SLASH	/	forward slash
K_0	0	0
K_1	1	1
K_2	2	2
K_3	3	3
K_4	4	4
K_5	5	5
K_6	6	6
K_7	7	7
K_8	8	8
K_9	9	9
K_COLON	:	colon

V. A la découverte de Pygame!

[Retourner au texte.](#)

Contenu 3

```
1 import pygame
2
3 pygame.init()
4
5 ecran = pygame.display.set_mode((640, 480))
6
7 continuer = True
8
9 while continuer:
10     for event in pygame.event.get():
11         if event.type == pygame.KEYDOWN:
12             if event.key == pygame.K_f:
13                 continuer = False
14
15 pygame.quit()
```

[Retourner au texte.](#)

Contenu 4

```
1 import pygame
2
3 pygame.init()
4
5 ecran = pygame.display.set_mode((640, 480))
6 clem = pygame.image.load("clem.png").convert_alpha()
7 pos_clem = (0, 0)
8
9 continuer = True
10
11 while continuer:
12     pygame.draw.rect(ecran, (255, 255, 255), (0, 0, 640, 480))
13     for event in pygame.event.get():
14         if event.type == pygame.MOUSEMOTION:
15             pos_clem = event.pos
16         if event.type == pygame.QUIT:
17             continuer = False
18     ecran.blit(clem, pos_clem)
19     pygame.display.flip()
20
21 pygame.quit()
```

Hum ... On vous des explications sur le `pygame.draw.rect(ecran, (255, 255, 255), (0, 0, 640, 480))`, non ?

En fait, on fait appel à la fonction `rect()` du module `draw` de Pygame. Cette fonction permet de dessiner un rectangle d'une couleur que l'on veut, où l'on veut, de la taille que l'on veut. Cela nous permet ici d'effacer l'écran à chaque `frame` pour ne pas avoir une trace pas très belle derrière Clem.

Voici son code pour mieux comprendre :

`pygame.draw.rect(surface, couleur, rectangle)`

- `surface` est une image (ou votre fenêtre)
- `couleur` est un tuple (de (0, 0, 0) à (255, 255, 255)) représentant notre couleur sous la forme RGB²
- `rectangle` est aussi un tuple, mais de quatre éléments :
 - le premier est le point de départ en abscisse
 - le second est le point de départ en ordonnée
 - le troisième est la largeur du rectangle
 - et le dernier la longueur du rectangle

[Retourner au texte.](#)

Contenu 5

```
1 import pygame
2
3 pygame.init()
4
5 ecran = pygame.display.set_mode((640, 480))
6
7 largeur = 10
8 hauteur = 10
9 couleur = (20, 180, 20)
10 continuer = True
11
12 while continuer:
13     for event in pygame.event.get():
14
15         if event.type == pygame.MOUSEBUTTONDOWN:
16             x, y = event.pos
17             pygame.draw.rect(ecran, couleur, (x, y, largeur,
18                                     hauteur))
19         if event.type == pygame.QUIT:
20             continuer = False
21     pygame.display.flip()
22
23 pygame.quit()
```

[Retourner au texte.](#)

6. TD?? Un petit jeu très simple

Et nous voici à notre premier TD !

Tout au long de cet exercice, on vous guidera pour réaliser ce petit jeu, puis à la fin, on vous proposera une liste d'améliorations possibles

6.1. Le cahier des charges

Notre cahier des charges va être simple :

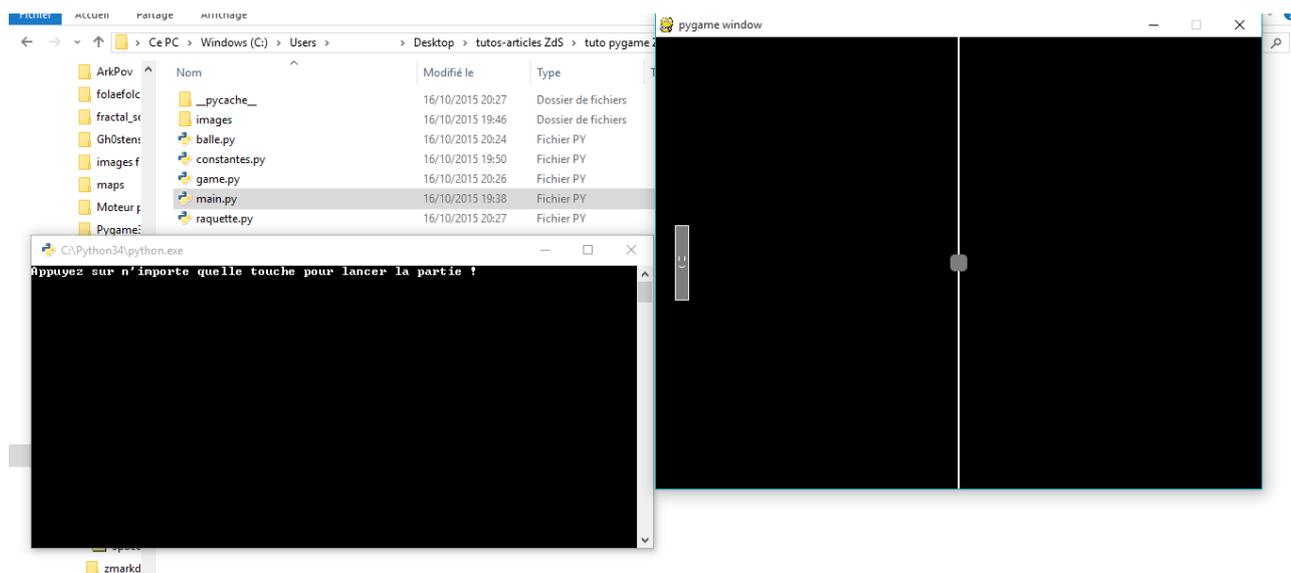
- avoir une raquette
- qu'il y ait une balle qui soit perpétuellement en mouvement dans le jeu, et qu'elle ne puisse pas sortir de l'écran

et enfin :

- faire un jeu qui marche

Vous l'aurez compris, nous allons coder le très célèbre Pong !

Et voici à quoi il ressemblera :



Voyons un peu comment nous allons agencer tout ça !



Étant fanatiques de la programmation orientée objet, on va en utiliser pas mal dans ce TD. Donc petit conseil, soyez à l'aise avec avant de continuer.

L'architecture que l'on a choisie est la suivante :

```
1 TD
2 |-->images
3 |---|---balle.png
4 |---|---raquette.png
5 |---|---menu.png
6 |---main.py
7 |---game.py
8 |---balle.py
9 |---raquette.py
10 |---constantes.py
```

6.2. Un menu et des constantes

6.2.1. Le menu

Le menu va devoir charger Pygame, lancer une boucle infinie qui affichera le fond du menu, et attendra que l'utilisateur veuille bien rentrer dans le jeu

En gros, on aura ceci :

```
1 menu
2 |---boucle
3 |---|---appuie sur une touche
4 |---|---|---lancement du jeu
5 |---|---|---|---appuie sur echap, on quitte la boucle du jeu
6 |---on se retrouve ici, dans la boucle du menu
```

Sans plus attendre, voici le code :

```
1 import pygame
2 from pygame import locals as const
3 from game import Game
4
5
6 def main():
```

```
7
8
9     print("Appuyez sur n'importe quelle touche pour lancer la partie !")
10
11 pygame.init()
12
13 ecran = pygame.display.set_mode((640, 480))
14 fond = pygame.image.load("images/menu.png").convert_alpha()
15
16 continuer = True
17 jeu = Game(ecran) # Game() est une class qui va se charger ...
18                 # du jeu :)
19
20 while continuer:
21     for event in pygame.event.get():
22         if event.type == const.QUIT or (event.type ==
23             const.KEYDOWN and event.key == const.K_ESCAPE):
24             # de manière à pouvoir quitter le menu avec echap
25             # ou la croix
26             continuer = 0
27         if event.type == const.KEYDOWN:
28             # start() sera une méthode de la class Game(), et
29             # s'occupera de lancer le jeu
30             jeu.start()
31
32     ecran.blit(fond, (0, 0))
33
34     pygame.display.flip()
35
36 pygame.quit()
37
38 if __name__ == '__main__':
39     main()
```

6.2.2. Les constantes

Dans la plupart des jeux, il y a des constantes. Il faut dire que ça simplifie **vraiment** la vie quand on souhaite modifier telle ou telle chose sans avoir à relire tout son code

Ici on va avoir besoin de peu de constantes, mais on créera quand même un fichier pour les stocker, histoire de bien séparer le *vrai* code de nos constantes.

On va donc avoir besoin des constantes suivantes :

- RIEN, pour indiquer que la raquette ne bouge pas
- HAUT, pour indiquer que la raquette monte
- BAS, pour indiquer ... que la raquette descend
- VITESSE_RAQUETTE, la vitesse de déplacement de notre raquette en pixels par seconde
- VITESSE_BALLE, la vitesse de déplacement de notre balle en pixels par seconde

V. A la découverte de Pygame!



RIEN, HAUT, et BAS auront des valeurs symboliques comme `-1`, `0`, et `1`.



Pour la vitesse de la raquette, on a choisi 20 pixels par secondes, et pour celle de la balle, 4 pixels par secondes.

6.3. La gestion de la raquette

Alors voilà! Nous y sommes.

Notre raquette devra pouvoir faire plein de choses, dont :

- s'afficher, bien entendu
- se déplacer de haut en bas tout en restant dans la fenêtre hein!
- fournir une méthode pour que la balle puisse regarder si elle entre en collision avec notre raquette
- ~faire des triples back flip de la mort ~

Allez hop, à vous de coder cette partie!



Une petite aide pour commencer ... ?

Mais bien sûr! (Pour ceux qui souhaitent tout coder sans indice, n'ouvrez pas le spoiler!)

👁 Afficher le contenu masqué



N'oubliez pas que notre raquette sera une classe!

Correction :

👁 Afficher le contenu masqué

6.4. Gestion de la balle

Voyons maintenant comment on va gérer notre balle.

Tout d'abord :

- il faut que la balle stocke un lien vers son image

V. A la découverte de Pygame!

- il faut qu'elle connaisse sa position
- il faut qu'elle ait en mémoire son vecteur directeur³ (et oui! on fait encore des maths)
- il lui faut une méthode pour qu'elle se déplace et qu'elle change de direction au contact d'un objet

Et voilà! Vous avez tout pour coder cette classe!



Hein ? Ça suffit pour coder la classe Ball ?

Oui. Le plus compliqué sera sûrement la méthode de déplacement, qui gèrera aussi les collisions.



Pour le changement de direction, vous embêtez pas, et utilisez le même vecteur directeur, en changeant simplement son sens et en ajoutant un peu d'aléatoire sur x (ou y) à chaque collision (sinon la balle suivra le même parcours).

Allez, hop hop hop! Au boulot!

Et voici notre version de cette classe :

👁 Afficher le contenu masqué

6.5. Et enfin, le cœur de la maison, la cuisine!

Non non, bien sûr, on parlait du cœur du jeu.

Pour ce qui est du jeu, c'est lui qui va créer la raquette, le terrain (graphiquement parlant hein), et la balle. On aura différentes méthodes, dont :

- `prepare`, c'est une méthode que SuperFola aime bien ajouter à chaque fois dans ses classe `Game`, car c'est elle qui va se charger de tout remettre à 0 s'il le faut, de re-régler la vitesse de répétition des touches etc ... Et oui! Imaginez un peu si on lance 2 fois le jeu (en supposant qu'après le 1er lancement, on avait quitté le jeu et que l'on soit revenu au menu pour le relancer)! Au 2ème lancement, le score sera le même qu'à la fin de la 1ère partie, et tout ce qui va avec!
- `update_screen`, qui sera chargée d'appeler toutes les méthodes d'affichage (de la raquette, de la balle) et de créer le terrain (graphiquement parlant).
- `process_event`, qui prendra en paramètre un événement unique (envoyé depuis une boucle `for` dans la méthode `start`) et qui se chargera d'effectuer les actions liées au-dit événement.
- `start`, qui va lancer le jeu (donc boucle `while`) et appeler toutes les autres méthodes, dans cet ordre :
 - `prepare`

3. Un vecteur directeur est un déplacement en x et en y. Donc c'est juste un tuple/une liste qui indique de combien doit on déplacer tel objet, en abscisse et ordonnée.

V. A la découverte de Pygame!

- boucle `while` :
- boucle `for` des événements :
- appel à `process_event(...)`
- `update_screen`
- déplacement de la balle
- `pygame.display.flip`

Pour ceux qui veulent s'entraîner, pas de problèmes, le code est sous spoiler!

Correction :

👁 Afficher le contenu masqué

Et voilà! Ce premier TD est fini.

C'était assez court, mais ça vous a donné du fil à retordre non?

Vous avez dû remarquer que notre balle n'entre pas en collision correctement avec la raquette ou les murs non? Et si on allait voir la méthode de déplacement de notre balle?

La voici :

```
1  def move(self, raquette):
2      tmp = self.pos[0] + self.vect_dir[0] * VITESSE_BALLE,
3          self.pos[1] + self.vect_dir[1] * VITESSE_BALLE
4      collision = raquette.collide_with_me(tmp, (self.b_large,
5          self.b_haut))
6      if collision or tmp[0] <= 0 or tmp[0] + self.b_large >=
7          self.ecran.get_width():
8          self.vect_dir[0] = - self.vect_dir[0] + randint(100,
9              225) / 1000
10         self.vect_dir[1] += randint(100, 225) / 1000
11     if tmp[1] <= 0 or tmp[1] + self.b_haut >=
12         self.ecran.get_height():
13         self.vect_dir[0] += randint(100, 225) / 1000
14         self.vect_dir[1] = - self.vect_dir[1] + randint(100,
15             225) / 1000
16
17     # dans tous les cas, on déplace la balle !
18     self.pos[0] += self.vect_dir[0]
19     self.pos[1] += self.vect_dir[1]
```

La ligne à incriminer est celle-ci, c'est certain!

```
1  tmp = self.pos[0] + self.vect_dir[0] * VITESSE_BALLE, self.pos[1] +
    self.vect_dir[1] * VITESSE_BALLE
```

V. A la découverte de Pygame!

En effet, en regardant bien, on voit que l'on n'applique pas la position calculée dans `tmp` si on est en collision avec un mur ou la raquette à cette nouvelle position. Or on se déplace de `VITESSE_BALLE`, qui ne vaut pas 1! Donc si la position actuel + 4 est dans la raquette, notre position actuelle ne sera pas forcément dans la raquette elle même, mais on applique quand même notre changement de direction.

Maintenant que l'on sait cela, corrigeons le!

Il faudrait donc avancer la balle même si elle entre en collision et changer en même temps sa direction (elle se déplace assez vite pour qu'on ne puisse pas voir que la balle *entre* dans la raquette ou dans un mur).

Voici donc une possible correction :

```
1     def move(self, raquette):
2         self.pos[0] += self.vect_dir[0] * VITESSE_BALLE
3         self.pos[1] += self.vect_dir[1] * VITESSE_BALLE
4
5         collision = raquette.collide_with_me(self.pos,
6         (self.b_large, self.b_haut))
7         if collision or self.pos[0] <= 0 or self.pos[0] +
8         self.b_large >= self.ecran.get_width():
9             self.vect_dir[0] = - self.vect_dir[0] + randint(100,
10            225) / 1000
11            self.vect_dir[1] = randint(100, 225) / 100
12         if self.pos[1] <= 0 or self.pos[1] + self.b_haut >=
13            self.ecran.get_height():
14            self.vect_dir[0] = randint(100, 225) / 100
15            self.vect_dir[1] = - self.vect_dir[1] + randint(100,
16            225) / 1000
```

Liste d'améliorations possibles :

- avoir des bonus au fur et à mesure que le temps passe
- que la balle accélère plus le temps de jeu augmente
- que des briques apparaissent sur le terrain, et fassent changer la balle de trajectoire en la percutant, un peu à la manière du Casse briques

C'est une petite liste d'améliorations possibles, mais cela va vous occuper un petit de bout de temps je pense

Contenu masqué

Contenu 1

pygame.event
pygame module for interacting with events and queues
pygame.event.pump — internally process pygame event handlers
pygame.event.get — get events from the queue
pygame.event.poll — get a single event from the queue
pygame.event.wait — wait for a single event from the queue
pygame.event.peek — test if event types are waiting on the queue
pygame.event.clear — remove all events from the queue
pygame.event.event_name — get the string name from an event id
pygame.event.set_blocked — control which events are allowed on the queue
pygame.event.set_allowed — control which events are allowed on the queue
pygame.event.get_blocked — test if a type of event is blocked from the queue
pygame.event.set_grab — control the sharing of input devices with other applications
pygame.event.get_grab — test if the program is sharing input devices
pygame.event.post — place a new event on the queue
pygame.event.Event — create a new event object
pygame.event.EventType — pygame object for representing SDL events

<http://www.pygame.org/docs/ref/event.html> ↗

Retourner au texte.

Contenu 2

On avait dit quoi ?

👁 Afficher le contenu masqué

Retourner au texte.

V. A la découverte de Pygame!

Contenu 3

```
1 import pygame
2 from constantes import *
3
4
5 class Raquette:
6     def __init__(self, ecran: pygame.Surface):
7         self.ecran = ecran
8         self.ecran_large = self.ecran.get_width() # ca vous nous
          servir pour centrer la raquette et regarder
9
          # si la raquette
          sort de
          l'écran ou
          non
10        self.ecran_haut = self.ecran.get_height() # idem
11        self.image =
          pygame.image.load("images/raquette.png").convert_alpha()
12        self.pos = [20, (self.ecran_haut - self.image.get_height())
          // 2] # on centre notre raquette à droite
13
14    def render(self):
15        self.ecran.blit(self.image, self.pos)
16
17    def move(self, dir: int=RIEN):
18        # on test toutes les collisions possibles
19        # et voici l'utilité de nos constantes !
20        if dir == HAUT:
21            if self.pos[1] - VITESSE_RAQUETTE >= 0:
22                self.pos[1] -= VITESSE_RAQUETTE
23            else:
24                self.pos[1] = 0
25        elif dir == BAS:
26            if self.pos[1] + VITESSE_RAQUETTE <= self.ecran_haut -
                self.image.get_height():
27
                self.pos[1] += VITESSE_RAQUETTE
28            else:
29                self.pos[1] = self.ecran_haut -
                self.image.get_height()
30
31    def collide_with_me(self, pos_objet: tuple, taille_objet:
        tuple):
32        # utile pour savoir si la balle collide avec la raquette
33        if self.pos[0] <= pos_objet[0] <= self.pos[0] +
            self.image.get_width() and \
34            self.pos[1] <= pos_objet[1] <= self.pos[1] +
            self.image.get_height():
35
            # le coté gauche de la balle est dans la raquette
36            return True
37        elif self.pos[0] <= pos objet[0] + taille objet[0] <=
```

Retourner au texte.

Contenu 4

```
1 import pygame
2 from constantes import *
3 from random import randint
4
5
6 class Balle:
7     def __init__(self, ecran: pygame.Surface):
8         self.ecran = ecran
9         self.vect_dir = [-VITESSE_BALLE, VITESSE_BALLE]
10        self.image =
11            pygame.image.load("images/balle.png").convert_alpha()
12        self.b_large = self.image.get_width() # la taille de notre
13            balle en x
14        self.b_haut = self.image.get_height() # et la taille en y
15        self.pos = [(self.ecran.get_width() - self.b_large) // 2,
16                    (self.ecran.get_height() - self.b_haut) // 2]
17
18    def move(self, raquette):
19        tmp = self.pos[0] + self.vect_dir[0] * VITESSE_BALLE,
20            self.pos[1] + self.vect_dir[1] * VITESSE_BALLE
21        collision = raquette.collide_with_me(tmp, (self.b_large,
22            self.b_haut))
23        if collision or tmp[0] <= 0 or tmp[0] + self.b_large >=
24            self.ecran.get_width():
25            self.vect_dir[0] = - self.vect_dir[0] + randint(100,
26                225) / 1000
27            self.vect_dir[1] += randint(100, 225) / 1000
28
29        if tmp[1] <= 0 or tmp[1] + self.b_haut >=
30            self.ecran.get_height():
31            self.vect_dir[0] += randint(100, 225) / 1000
32            self.vect_dir[1] = - self.vect_dir[1] + randint(100,
33                225) / 1000
34
35        # dans tous les cas, on déplace la balle !
36        self.pos[0] += self.vect_dir[0]
37        self.pos[1] += self.vect_dir[1]
38
39    def render(self):
40        self.ecran.blit(self.image, self.pos)
```

Retourner au texte.

V. A la découverte de Pygame!

Contenu 5

```
1 import pygame
2 from pygame import locals as const
3 from constantes import *
4 from raquette import Raquette
5 from balle import Balle
6
7
8 class Game:
9     def __init__(self, ecran: pygame.Surface):
10         self.ecran = ecran
11         self.raquette = Raquette(self.ecran)
12         self.balle = Balle(self.ecran)
13         self.continuer = True
14         self.controles = {
15             HAUT: const.K_UP,
16             BAS: const.K_DOWN
17         }
18
19     def prepare(self):
20         pygame.key.set_repeat(200, 50)
21         self.continuer = True
22         self.raquette = Raquette(self.ecran)
23         self.balle = Balle(self.ecran)
24
25     def update_screen(self):
26         pygame.draw.rect(self.ecran, (0, 0, 0), (0, 0) +
27             self.ecran.get_size()) # on dessine le fond
28         pygame.draw.rect(self.ecran, (255, 255, 255),
29             (self.ecran.get_width() // 2 - 1, 0, 2,
30             self.ecran.get_height())) # la séparation au milieu du
31             terrain
32         self.raquette.render() # on dessine notre raquette
33         self.balle.render() # on dessine notre balle
34
35     def process_event(self, event: pygame.event):
36         if event.type == const.KEYDOWN:
37             # et revoici l'utilité de nos constantes, utilisées
38             # comme clé de dictionnaire :)
39
40             # comme ça on peut plus facilement changer les
41             # controles ;)
42             if event.key == self.controles[HAUT]:
43                 self.raquette.move(HAUT)
44             if event.key == self.controles[BAS]:
45                 self.raquette.move(BAS)
46         if event.type == const.QUIT:
47             self.continuer = False
48
49     def start(self):
50         self.prepare()
51
52         while self.continuer:
53             for event in pygame.event.get():
```

V. A la découverte de Pygame!

Retourner au texte.

7. Annexes

Ce chapitre est une annexe, vous n'êtes en aucuns cas obligés de le lire!

7.1. Les différents modules de Pygame

Ici on ne va pas vous présenter **tous** les modules de Pygame, juste les principaux. Après, pour les curieux, vous avez le site officiel de [Pygame](#) ↗

En important Pygame, il y a une chose que n'avez pas vu : c'est qu'en fait il est composé de plein de petits modules très utiles, comme ceux-ci :

7.1.0.1. `display`

C'est lui qui va nous permettre de créer nos fenêtres, de rafraichir notre écran, de réduire notre fenêtre ...

7.1.0.2. `mixer`

Lui, il nous donne la possibilité de lire de la musique, nous verrons cela dans la prochaine partie . Rien que pour vous mettre l'eau à bouche, il permet aussi de mettre en pause la musique, de savoir si une musique est jouée, de réserver des *channels*, d'en créer ...

7.1.0.3. `draw`

C'est un module très utile, qui permet de créer des formes en spécifiant la taille, la couleur et la position (voir parfois d'autres paramètres)

7.1.0.4. `event`

Celui là, c'est notre meilleur ami. Et ça sera bientôt le vôtre! Il permet de récupérer des évènements, comme un appui sur une touche, le déplacement de votre souris, la touche X de votre manette de Xbox ... Il permet aussi de bloquer des événements (si on les bloque, Pygame ne les "écouterà" plus), d'en créer ... bref c'est **le** module le plus important (à nos yeux) dans Pygame.

V. A la découverte de Pygame!

7.1.0.5. **image**

Lui il permet de charger des images, et de les sauvegarder (utile pour faire des *screenshots* en jeu)

7.1.0.6. **mouse**

Celui-ci ressemble un peu à **event**, car il permet aussi d'obtenir l'état de la souris, par exemple : votre souris est en $x=0$, $y=12$, le bouton gauche est enfoncé ... Mais il nous permet aussi de déplacer le pointeur de la souris (uniquement dans notre fenêtre hein), de savoir si la souris est dans l'interface de jeu ...

7.1.0.7. **Surface**

Ce module est carrément une bibliothèque à lui tout seul, c'est pourquoi on ne décrira que quelques fonctionnalités de celui-ci. Il nous permet de créer des images, de les rendre transparentes à un certain degré, de convertir nos images dans un format lisible par Pygame, d'afficher des images sur d'autres images (il faut savoir que notre fenêtre est une image pour Pygame!), de modifier pixel par pixel une image ... bref c'est une machine à tout faire (sauf le café hélas)!

Et voilà, on en a terminé (du moins pour cette partie)

Maintenant, partons à l'attaque des fonctionnalités plus "avancées" de Pygame!

Voilà, maintenant vous connaissez la marche à suivre pour créer une fenêtre basique et y ajouter ce qui est le plus important dans un programme : les **images**!

Maintenant, partons à l'abordage de la seconde partie de ce tutoriel!

Sixième partie

Conclusion

VI. Conclusion

i

SuperFola a ouvert un dépôt Github de projets d'exemples utilisant Pygame (voir les différentes branches), juste [ici](#) . N'hésitez surtout pas à y contribuer !

Septième partie

Remerciements

VII. Remerciements

Merci à Karnaj, Looping, Wizix, the_new_sky, Emeric, Kje, Nohar, Smokiev, klafyvel, Arius, (et bien d'autres que l'on oublie sûrement!), pour leur contribution (soutien, relecture, proposition d'exercice, ... etc)!

Huitième partie

Le mot de la fin

VIII. Le mot de la fin

Voilà, ce tutoriel est maintenant terminé

On espère qu'il vous a plu et que vous le garderez dans vos favoris (n'oubliez pas le petit `CTRL+D`, ça sert beaucoup :D)!

Nous attendons avec impatience vos retours sur ce tutoriel, et nous avons hâte de voir ce que vous allez pouvoir réaliser avec Pygame!