Cours Python : La bibliothèque Pygame

I – Présentation & Généralités

Avant toutes choses, il est très important d'avoir déjà quelques notions de Python. Comme les différentes boucles (*if*, *for*, *while*, ...) et différentes structures (*def*, *class*, ...). C'est pour cela que vous trouverez en annexes un résumé des différentes structures dont nous aurons probablement besoin au cours de ce chapitre.

I. 1 – Pygame, c'est quoi ?

Pygame est une bibliothèque Python permettant la réalisation simple de jeux interactif. Cette bibliothèque est principalement basée sur la SDL (*Simple Directmedia Library*) qui est une bibliothèque libre multiplateforme permettant ma gestion du multimédia dans la programmation.

Remarque : La SDL est écrite en C, mais est utilisable avec un grand nombre de langages, comme le ... C/C++, Pascal, Perl ou encore ... Python !

En résumé, cette bibliothèque nous servira principalement pour :

- L'affichage vidéo 2D
- La gestion de l'audio
- La gestion de périphériques de commandes (clavier, souris...)

Pygame est donc l'adaptation de la SDL au service de Python, mais est aussi constitué de quelques ajouts et modifications de son auteur.
Cette bibliothèque est assez intuitive et constitue un très bon moyen de se lancer dans la programmation graphique avec Python.

I. 2 – Installation

Pour ce qui est de l'installation, il vous suffit de vous rendre sur la page :
http://www.pygame.org/download.shtml

Et de sélectionner la version correspondante à votre version de Python.
La version 2.7 est fortement recommandée car elle est bien plus stable avec Pygame.

I. 3 – Comment réaliser un jeu ?

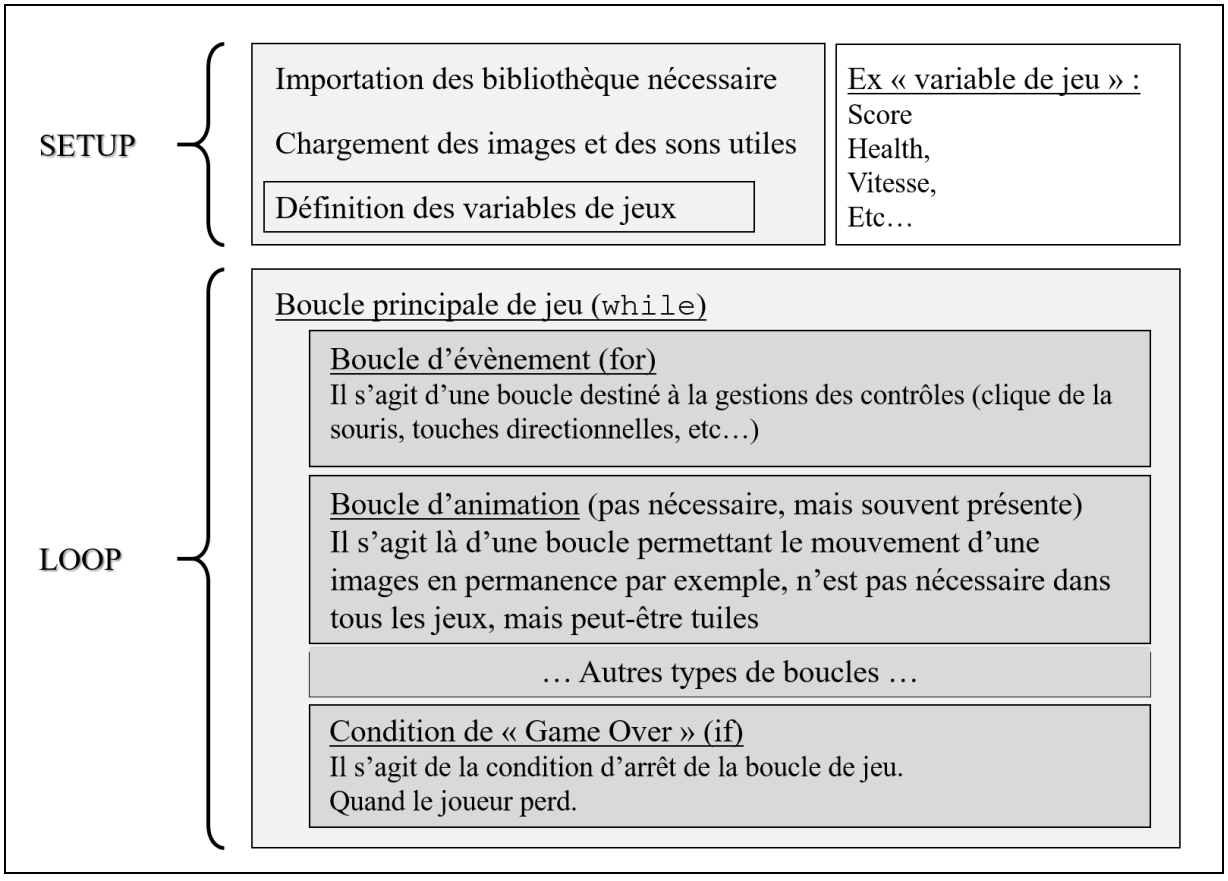
Avant de se lancer dans la programmation « la tête la première », il faut déjà savoir quel type de jeu on souhaite créer ? de quelle façon ? quel langage nous allons utiliser pour le coder ?.

Il faut donc tout d'abord se poser de nombreuses questions « clefs ». Il faut réaliser un Cahier des Charges c'est-à-dire quels seront vos objectifs à atteindre. Voici un exemple de raisonnement possible avant de créer un petit jeu :

- Quels types de jeu ?
  ⇨ Combat, Plateforme, RPG, Arcade, Simulation, Réflexion…

- Quels langages, et quelle bibliothèque je vais utiliser pour faire mon jeu ?
  ⇨ Ici, avec Pygame, une grande variété de jeu est faisable, en voici une liste non exhaustive : *Donkey Kong, Pacman, Pong, Street Fighter, Space Invader, Mario Bros, Mario Kart, Flappy bird, Mini Zelda/Pokémon like, Sudoku, Snake, 2048, Démineur, Game of life, etc…*

- Comment va fonctionner mon jeu, comment vais-je structuré mon code afin qu'il soit compréhensible ? Comment le joueur peut-il perdre ? ou gagné ?
  ⇨ Ici, c'est assez simple, car tous les jeux Pygame fonctionnent sur quasiment la même structure :



II – Premier pas

II. 1 – Importation de la bibliothèque

Après avoir installé Pygame, et avant de se lancer dans la rédaction de centaines de ligne de code, il va falloir importer cette librairie. Pour l'importer c'est assez simple, il suffit de faire comme pour n'importe quelle autre libraire, c'est-à-dire, ajouter en début de notre programme :

```
import pygame
from pygame.locals import *
```

Remarque : La seconde ligne permet d'importer les constantes de Pygame directement dans l'espace de votre script. Pour y accéder, il suffira de taper *CONSTANTE*, plutôt que *pygame.CONSTANTE*. Cela nous permettra une meilleure lisibilité dans l'utilisation de ces constantes, nous verrons plus tard à quoi elles serviront.

II. 2 – Ouverture d'une fenêtre

Avant toute chose, cette bibliothèque nécessite d'être initialisée. Pour cela, on utilise l'instruction suivante :

```
Pygame.init()
```

Une fois Pygame initialisé, nous allons demander la création d'une fenêtre. Pour faire ceci, il suffit d'entrer :

```
fenetre = pygame.display.set_mode((640, 480))
```

Maintenant, analysons cette ligne :

- Premièrement, nous déclarons une variable *fenêtre*
- Nous appelons ensuite la fonction *set_mode()* contenue dans le module "display" de Pygame
- Cette fonction prend en paramètre un tuple contenant la largeur et la hauteur de la fenêtre voulue. Attention, c'est un tuple qui est passé en argument, et non pas deux arguments différents, il est donc nécessaire de mettre les parenthèses.

Remarque : Arrivé à ce stade, lorsque nous exécutons le code, une fenêtre noire avec la dimension souhaitée apparait, mais il nous est impossible de la fermer pour le moment. Nous verrons comment la fermer dans la partie III (Gestion des évènements).

II. 3 – Afficher une image

Maintenant, essayons de rendre cette fenêtre noire un peu plus esthétique. Essayons d'afficher des images, mettre une image de fond et un petit personnage.

Pour commencer, il faut pré-charger les images au début du code, pour cela, la ligne de code à utiliser est :

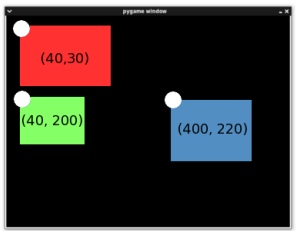
```
fond = pygame.image.load("background.jpg").convert()
```

Ici, nous avons chargé l'image dans la variable « `fond` ».

Le principe d'affichage de la SDL est à connaître pour bien afficher ses images : `fenetre` est une surface vide, sur laquelle on va "coller", ou "empiler" les autres images. Le fond doit donc être empilé sur la surface vide de la fenêtre. Cela s'apparente à un système de calque, donc attention à l'ordre d'appel, une image pourrait être cachée derrière une autre !

Pour mettre une image, ou « coller » une image sur la surface, on utilise la fonction `blit()` de la bibliothèque Pygame. Comme ceci :

```
fenetre.blit(fond, (0,0))
```



La fonction `blit()` prend comme première argument l'image pré-chargée que l'on souhaite coller, et en second argument, il va prendre les positions de l'endroit on l'on souhaite afficher notre image sous forme de tuple. Les coordonnées sont celles de l'angle en haut à gauche de l'image.

Jusqu'ici, si vous avez exécuté votre code, votre image n'est pas apparue, et c'est normal, car il faut rafraichir, actualisé la surface pour faire apparaitre notre « collage ». Pour ce faire il suffit de taper :

```
pygame.display.flip()
```

Voilà, maintenant vous savez afficher une image dans une fenêtre ! Mais une simple image ne constitue pas un jeu, il manque des animations, des contrôles et des interactions.

Passons maintenant à la gestion des évènements, afin de pouvoir créer des contrôles.

III – Gestion des évènements

III. 1 – Les touches du clavier

Pygame est une librairie qui fonctionne avec des évènements. Mais qu'est-ce qu'un évènement, un événement est simplement une action que l'utilisateur effectuera, par exemple cela peut être « Appuyez sur la touche E » ou encore « Bouger la souris ». Pour cela, Pygame enregistre dans une liste tous les nouveaux évènements lorsqu'il y en a un et de manière asynchrone. Ces évènements sont alors enregistrés dans la « variable » : `pygame.event.get()`

Tous les programmes utilisant pygame fonctionnenet de la même manière, ils utilisent une boucle infinie du type : `while game_over`. C'est-à-dire que le jeu va exécuter notre boucle à l'infini jusqu'à ce qu'une variable « `continuer` » ou « `game_over` » passe à 0. Ce sera notre condition d'arrêt de notre jeu.

Voici un morceau de code de « base » :

```
continuer = 1

while continuer:
        for event in pygame.event.get():
                if event.type == QUIT :
                        continuer = 0
```

Analysons ce bout de code :

- Tout d'abord on crée la boucle infinie.
- Ensuite, on utilise une boucle *for*, pour parcourir tous les événements reçus grâce à la fonction *get()* du module "event" de Pygame. Cette fonction retourne une liste d'objets Event, pour lesquels on peut connaître le type, la touche enfoncée si c'est au clavier, la position du curseur si c'est un clic, etc...
- La condition teste si l'événement est de type QUIT (c'est à dire un Alt+F4 ou un clic sur le bouton de fermeture)
- Si la condition est satisfaite, on demande à la boucle de s'arrêter.

Remarque : Ici, on note `QUIT` au lieu de `pygame.QUIT` car nous avons fait au début l'import de : `from pygame.locals import *`. Il s'agit d'une CONSTANTE pour Pygame.

Pygame gère de nombreux éléments, en voici une liste non exhaustive : (Touche du clavier)

Lettres: K_a ... K_z Nombres: K_0 ... K_9 Controles: K_TAB K_RETURN K_ESCAPE K_SCROLLOCK K_SYSREQ K_BREAK K_DELETE K_BACKSPACE K_CAPSLOCK K_CLEAR K_NUMLOCK Parenthèses: K_RIGHTBRACKET, K_LEFTBRACKET K_RIGHTPAREN, K_LEFTPAREN	Ponctuation: K_SPACE K_PERIOD K_COMMA K_QUESTION K_AMPERSAND K_ASTERISK K_AT K_CARET K_BACKQUOTE K_DOLLAR K_EQUALS K_EURO K_EXCLAIM K_SLASH, K_BACKSLASH K_COLON, K_SEMICOLON K_QUOTE, K_QUOTEDBL K_MINUS, K_PLUS K_GREATER, K_LESS Touches F: K_F1 ... K_F15 Touches d'édition: K_HELP K_HOME K_END K_INSERT K_PRINT K_PAGEUP, K_PAGEDOWN K_FIRST, K_LAST	SHF,CTL,ALT etc: K_LALT, K_RALT K_LCTRL, K_RCTRL K_LSUPER, K_RSUPER K_LSHIFT, K_RSHIFT K_RMETA, K_LMETA Flèches: K_LEFT K_UP K_RIGHT K_DOWN Autres: K_MENU K_MODE K_PAUSE K_POWER K_UNDERSCORE K_HASH Clavier numérique: K_KP0 ... K_KP9 K_KP_DIVIDE K_KP_ENTER K_KP_EQUALS K_KP_MINUS K_KP_MULTIPLY K_KP_PERIOD K_KP_PLUS

Comment utiliser ces constantes du clavier pour détecter l'appuie sur une touche ?
Premièrement, il est plus simple de détecter l'action « APPUYER » et ensuite la « TOUCHE » correspondante. Pour ce faire nous rajoutons dans notre boucle `while` puis `for` les bonnes conditions.

Comme ceci :

```
continuer = 1

while continuer:
        for event in pygame.event.get():
                if event.type == QUIT :
                        continuer = 0

                if event.type == KEYDOWN:
                        if event.key == K_SPACE:
                                print("Espace")
                        if event.key == K_RETURN:
                                print("Entrée")
```

Ici, nous affichons sur la console « Entrée » lorsque l'utilisateur appuie sur la touche entrée, et « Espace » lorsque l'utilisateur appuie sur la touche espace.

Remarque : Il peut être parfois intéressant de pouvoir effectuer l'action correspondante à la touche plusieurs fois en laissant la touche enfoncée, pour cela il faut rajouter, avant la boucle principale, la fonction `pygame.key.set_repeat(delay, time)`

III. 2 – Les évènement liés à la souris

Maintenant, passons aux évènements liés à la souris.

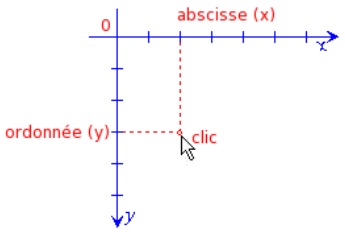
Le type d'événement crée lors d'un clic est MOUSEBUTTONDOWN, (ou MOUSEBUTTONUP au relâchement du bouton). Un événement de souris possède deux attributs : le bouton (`event.button`) et la position du clic dans la fenêtre (`event.pos`).

`event.button` peut prendre les valeurs suivantes :

1 Bouton gauche
2 Bouton milieu (ou droite + gauche)
3 Bouton droit
4 Molette haut
5 Molette bas

`event.pos`, lui, renvoie un tuple contenant l'abscisse et l'ordonnée à partir de l'angle haut-gauche, c'est-à-dire le bout de la pointe de la flèche.

On peut donc imaginer de faire un script permettant d'afficher un message sur la console lorsque l'on clique droit sur une partie de l'écran. Ici nous prendrons la partie en haut à droite.




```
if event.type == MOUSEBUTTONDOWN and event.button == 3 and event.pos[1] < 150 and
event.pos[0] > 100:
    print("Zone dangereuse")
```

Le code précèdent permet donc d'afficher « Zone dangereuse », lorsque l'on clique (`event.type == MOUSEBUTTONDOWN`) droit (`event.button == 3`) avec la souris, en haut à droite, entre y<150 (`event.pos[1] < 150`) et x>100 (`event.pos[0] > 100`) ;

IV – Mise en mouvement d'image

IV. 1 – Les « Rects »

Maintenant que nous connaissons comment fonctionne les évènements, nous allons essayer de mettre en mouvement des images.

Pour cela nous utiliserons un nouvel objet, l'objet `Rect`, qui permet de manipuler des surfaces rectangulaires. Comme toutes nos images sont rectangles, cela facilitera leurs déplacements en fonction du temps.

`Rect` stocke en fait les positions d'une surface. Pour créer un Rect, nous utilisons la méthode de Surface `get_rect()`.
Pour obtenir le Rect *position_perso* à partir de *perso*, la variable de chargement de notre image :

```
perso = pygame.image.load("personage.png").convert()

position_perso = perso.get_rect()

fenetre.blit(perso, position_perso)
```

Analysons ce morceau de code, tout d'abord, nous chargeons l'image de notre `perso` comme nous l'avons vu dans la première partie. Ensuite, nous stockons la position de cette image dans la variable `position_perso` à l'aide de la fonction de Pygame `.get_rect()`, cette fonction n'est applicable que sur les variables images.

Maintenant, imaginons que l'on veuille faire bouger cette image, pour cela on utilise :

```
nom_du_rect.move(déplacement_x, déplacement_y)
```

Il suffit dès à présent d'intégrer ce morceau de code dans une boucle, et après une condition. Par exemple, nous pouvons faire bouger l'image après chaque appui sur une touche à l'aide des évènements.

IV. 2 – Les collisions

Il peut être parfois utile de savoir s'il y a « collision » de deux images, par exemple pour éviter que le joueur rentre dans un mur, ou encore pour faire perdre le joueur s'il ne faut pas qu'il touche tels ou tels objet. Pour cela on peut utiliser le test suivant :

```
if position_perso.colliderect(rect_objet):
```

V – Sources

Comme vous pouvez vous l'imaginez, je n'ai pas tous deviner tout seul, alors ou est-ce que j'ai pu trouver des éléments pour rédiger ce cours ? J'ai utilisé de nombreux sites, en voici la liste.

Vous y trouverez surement des informations intéressantes pour la réalisation de votre jeu, car en effet je n'ai pas pu énoncer toutes les possibilités de Pygame, et il y en a des tas d'autres, une infinité !

- http://www.pygame.org
- https://openclassrooms.com/courses/interface-graphique-pygame-pour-python
- https://zestedesavoir.com/tutoriels/846/pygame-pour-les-zesteurs/

Pour toutes questions, n'hésitez pas à me contacter à l'adresse : aymeric.deliencourt@isen.yncrea.fr