Documentation pour s'initier à Pygame

Objectifs :

➢ Débuter avec Pygame

Tous les fichiers ci-dessous (sources, images, sons) sont donnés.

Les fichiers images et sons doivent se trouver dans le même dossier que les fichiers sources qui les utilisent.

1 Qu'est-ce que Pygame ?

Pygame est une bibliothèque Python (écrite principalement en C) qui permet de créer des jeux simples. Elle permet de faire facilement des animations (déplacements d'images à l'écran , ...), de gérer du son ou de la vidéo.
Pygame n'est pas fourni par défaut lorsqu'on installe Python. Pour l'installer, on peut choisir une distribution comme Edupython qui installe, en même temps que Python, un certain nombre de modules.
On peut également l'installer après avoir installé Python à condition de bien choisir le fichier correspondant au système d'exploitation et à la version de Python déjà installée (Téléchargement de pygame). Attention, pygame ne fonctionne qu'avec une version 32 bits de Python (si on a une version 64 bits de Python, il faut la désinstaller et installer une version 32 bits).
Le principe consiste à créer une boucle *while* principale : à chaque passage, une nouvelle image, où chaque objet est légèrement déplacé, est générée dans un buffer (mémoire tampon qui évite de dessiner directement à l'écran pour éviter de voir les éléments se dessiner au fur et à mesure). L'écran est ensuite effacé, puis la nouvelle image (une fois entièrement mise à jour dans le buffer) est affichée à l'écran.
La succession rapide de toutes ces images donne l'impression de mouvement dans la fenêtre graphique.
Les explications ci-dessous sont sous forme d'exemples typiques commentés : il faudra ensuite les adapter à chaque cas particulier.
Exemple typique d'utilisation de Pygame (les explications sont en commentaire) :

```
"""
Balle qui rebondit sur les bords de la fenêtre graphique.
"""
import pygame

taille = largeur, hauteur = 600, 400 #Taille de la fenêtre
vitesse = [3 , 3] #On déplace de 3 pixels en abscisse ou 3 pixels en
    ordonnée à chaque fois. Plus le nombre de pixels est élevé, plus ça
     semble rapide.
noir = 0, 0, 0 #Couleur pour effacer la fenêtre

pygame.init() #Initialisation de tous les modules de Pygame
fenetre = pygame.display.set_mode(taille) #Création de la fenêtre avec
    les bonnes dimensions. fenetre est une Surface
balle = pygame.image.load("ball.png") #Chargement de l'image qui
    représente la balle. balle est une Surface
ballerect = balle.get_rect() #Attention: on ne travaille pas sur la
    Surface balle directement mais sur le rectangle où elle se trouve:
    c'est celui-ci qu'on déplace pour obtenir la nouvelle position
continuer = True #variable booléenne qui permet de sortir de la boucle
    si elle est False. On quitte pygame si elle est False
while continuer:
    for event in pygame.event.get(): #On parcourt tous les événements
        générés (clavier, souris, ...)
```

```
        if event.type == pygame.QUIT: #Si c'est un événement qui
            correspond à la sortie de pygame (clic sur la croix ...),
            continuer devient False
            continuer = False
    ballerect = ballerect.move(vitesse) #On déplace le rectangle: le
        nouveau rectangle obtenu correspond à la nouvelle position de l
        'image. Attention, on travaille dans un buffer (tampon) et rien
        ne se passe à l'écran ici
    if ballerect.left < 0 or ballerect.right > largeur: # On inverse le
        déplacement si on heurte un bord de la fenêtre
        vitesse[0] = -vitesse[0]
    if ballerect.top < 0 or ballerect.bottom > hauteur:
        vitesse[1] = -vitesse[1]
    fenetre.fill(noir) #On efface la totalité de la fenêtre (toujours
        dans le buffer) en la remplissant de noir
    fenetre.blit(balle, ballerect) #Permet de dessiner (dans le buffer)
        la Surface balle à l'endroit où se trouve le rectangle
        ballerect
    pygame.display.flip()#Met à jour l'affichage de la fenêtre: affiche
        le contenu du buffer à l'écran.
    pygame.time.Clock().tick(100)#Permet de ralentir l'exécution sur un
        ordinateur très rapide: permet d'exécuter la boucle 100 fois
        par seconde maximum
pygame.quit()#Ne pas oublier de quitter proprement pygame
```

Remarques :

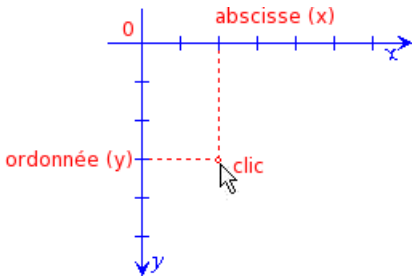
- On importe pygame (*import pygame*) et on initialise tous les modules de pygame (*pygame.init()*).
- On génère une fenêtre graphique (appelée *fenetre* dans l'exemple : *fenetre = pygame.display.set_mode(taille)*).
- On charge l'image du fichier *ball.png* qui doit se trouver dans le même dossier que le module. Les images sont représentées par des objets de type Surface. La fenêtre graphique (ici *fenetre*) est une Surface.
- On crée une boucle principale (*while continuer :*) avec la gestion d'un événement qui permet de sortir (*pygame.QUIT*).
- Pour déplacer une image (de type Surface), on utilise le rectangle (de type Rect) où elle se trouve (*ballerect = balle.get_rect()*), et pas l'image elle-même. On décale alors ce rectangle (*ballerect = balle-rect.move(vitesse)*), ce qui donne la nouvelle position.
- On efface l'écran (*fenetre.fill(noir)*)
- On réaffiche l'image dans le nouveau rectangle obtenu (*fenetre.blit(balle,ballerect)* ; cette instruction dessine l'image à la position où se trouve le nouveau rectangle *ballerect*). Tout ceci se passe dans le buffer et pas à l'écran.
- on met à jour l'affichage avec l'instruction *pygame.display.flip()* qui affiche à l'écran le contenu du buffer.
- L'instruction *pygame.time.Clock().tick(100)* permet de ralentir l'exécution sur un ordinateur très rapide : elle permet d'exécuter la boucle 100 fois par seconde maximum. Elle calcule le temps écoulé depuis le dernier appel à *pygame.time.Clock().tick* et met le programme en attente si ce temps est inférieur à un centième de seconde.

2 La fonction blit

Cette fonction (appelée aussi "méthode" en programmation orientée objets) permet de dessiner une image (de type Surface) sur une autre.
La syntaxe *surface1.blit(surface2, position)* permet de dessiner *surface2* sur *surface1* à l'emplacement *position*. *position* peut être un couple de coordonnées correspondant au coin haut gauche de *surface2* dans le système de coordonnées de *surface1*.
position peut également être un rectangle (de type Rect) et le coin haut gauche de ce rectangle donne la position du coin haut gauche de *surface2* dans le système de coordonnées de *surface1*. Dans la pratique, on pourra utiliser un rectangle de mêmes dimensions que la Surface *surface2* pour positionner *surface2* dans *surface1* car ce rectangle permet de gérer le déplacement (méthode *move* de la classe Rect), de savoir si la souris survole l'image (méthode *collidepoint* de la classe Rect), de gérer les collisions entre images (méthode *colliderect* de la classe Rect).
Un rectangle de mêmes dimensions que *surface2* peut être obtenu avec la syntaxe : *rectangle = surface2.get_rect()*
Le dessin de *surface2* sur *surface1* se passe dans un premier temps dans une mémoire tampon (buffer) et il faut rafraîchir l'affichage (avec la méthode *pygame.display.flip()*) pour que le buffer soit affiché à l'écran.
Pour déplacer une image à l'écran, voir les explications dans le paragraphe précédent.

3 Dessiner des formes : lignes, rectangles, cercles, texte...

On rappelle que les axes de coordonnées sont différents de ceux utilisés en mathématiques : l'origine du repère est en haut à gauche de la fenêtre graphique. L'axe des ordonnées est orienté vers le bas :



Pour dessiner des lignes, rectangles, cercles..., on utilise le module *draw* de pygame.
Tout est d'abord dessiné dans le buffer et il faut ensuite mettre à jour l'affichage avec *pygame.display.flip()*.
Pour les couleurs, on peut utiliser des tuples à trois valeurs de la forme *(rouge, vert, bleu)* où *rouge*, *vert*, *bleu* sont des entiers entre 0 et 255.

- Pour tracer une ligne, la syntaxe est *pygame.draw.line(Surface, color, start_pos, end_pos, width=1)*. Le premier argument est la Surface sur laquelle la ligne est dessinée. *start_pos* et *end_pos* sont des couples de coordonnées. *width* est l'épaisseur.
- Pour tracer un rectangle, la syntaxe est *pygame.draw.rect(Surface, color, Rect, width=0)*. Le premier argument est la Surface sur laquelle le rectangle est dessiné. L'argument de type *Rect* peut être un rectangle ou un tuple de 4 valeurs : les deux premières sont les coordonnées haut gauche du rectangle à dessiner et les deux dernières sont la largeur et la hauteur. Si *width* vaut 0, le rectangle est rempli.
- Pour tracer un cercle, la syntaxe est *pygame.draw.circle(Surface, color, pos, radius, width=0)*. Le premier argument est la Surface sur laquelle le cercle est dessiné. L'argument *pos* est un tuple de 2 valeurs représentant les coordonnées du centre. Si *width* vaut 0, le cercle est rempli.
- Pour tracer du texte à l'écran, on utilise le module *font* de pygame (font signifie police de caractères en anglais).
  On crée un objet de type Font qui représente la police (nom de la police, taille des caractères, gras,...) ; *police = pygame.font.SysFont('Arial', 20, bold=True)*
  On crée un objet de type Surface (c'est une image qui représente le texte) à partir de cette police : *texte = police.render("Cliquer ici", True, (0,128,0))*. Le premier argument est le texte à afficher, le deuxième indique si on applique l'antialias (anti crénelage : True signifie que les lettres seront lissées pour éviter le crénelage). Le troisième argument est la couleur.
  On dessine la Surface *texte* obtenue sur une autre Surface avec la méthode *blit* (voir le paragraphe sur *blit*).

Exemple typique de tracés de dessins et de texte (les explications sont en commentaire) :

```
"""
    Dessins de lignes, rectangles, cercles, texte
"""

import pygame

taille = largeur, hauteur = 600, 400 #Taille de la fenêtre


pygame.init() #Initialisation de tous les modules de Pygame
fenetre = pygame.display.set_mode(taille) #Création de la fenêtre avec
    les bonnes dimensions. fenetre est une Surface
fenetre.fill((255,255,255))
pygame.draw.line(fenetre, (255,0,0), (10,50), (500, 300), 5)#dessin d'
    une ligne dans fenetre
pygame.draw.rect(fenetre, (128,128,128), (100,150,70,30), 0) #dessin d'
    un rectangle. 100,150 sont les coordonnées haut gauche du rectangle
    . 70,30 sont la largeur et la hauteur.
pygame.draw.circle(fenetre, (0,255,0), (300,300), 50, 5)#desin d'un
    cercle. (300,300) sont les coordonnées du centre, 50 est le rayon.
```

```
police = pygame.font.SysFont('Arial', 20, bold=True) #On crée la policequ'
    on va utiliser pour le texte (20 est la taille)
texte = police.render("Bonjour tout le monde", True, (0,128,0)) #On
    crée une Surface qui contient le texte à afficher (True signifie qu
    'on lisse le texte pour éviter le crénelage)
fenetre.blit(texte, (200,100)) #On affiche la Surface texte dans la
    Surface fenetre avec blit
pygame.display.flip()#Met à jour l'affichage de la fenêtre: affiche le
    contenu du buffer à l'écran.

continuer = True #variable booléenne qui permet de sortir de la boucle
    si elle est False. On quitte pygame si elle est False
while continuer:
    for event in pygame.event.get(): #On parcourt tous les événements
        générés (clavier, souris, ...)
        if event.type == pygame.QUIT: #Si c'est un événement qui
            correspond à la sortie de pygame (clic sur la croix ...),
            continuer devient False
            continuer = False
pygame.quit()#Ne pas oublier de quitter proprement pygame
```

4 Gestion des événements souris

A chaque passage dans la boucle principale, on parcourt tous les événements récupérés par pygame (*for event in pygame.event.get() :*) et on teste si le type de chaque événement correspond à un événement souris : MOUSEBUTTONUP (bouton souris relâché), MOUSEBUTTONDOWN (bouton souris enfoncé), MOUSEMOTION (souris déplacée).
Chaque événement *event* qui provient de la souris possède des informations qu'on peut utiliser : par exemple, *event.pos* donne les coordonnées de la souris et *event.button* donne le bouton cliqué (en cas de clic) :1 pour le gauche, 2 pour le milieu ou gauche+droite, 3 pour le droit, 4 pour la molette vers le haut...
event.pos[0] représente l'abscisse de la souris et *event.pos[1]* représente l'ordonnée.
Exemple typique d'utilisation des événements souris (les explications sont en commentaire) :

```
"""
    Dessine un disque de rayon et de couleur aléatoires à l'endroit où
        on clique avec la souris.
"""
import pygame
from pygame.locals import * #Permet de charger des constantes comme
    KEYDOWN ou KEYUP... dans l'espace de nom de ce module. Ceci permet
    d'écrire KEYDOWN au lieu de pygame.KEYDOWN
from random import randint

taille = largeur, hauteur = 800, 600 #Taille de la fenêtre graphique
couleurs = [(255,0,0), (0,255,0), (0,0,255), (0,255,255), (255,0,255),
    (255,255,0), (255,255,255)]
pygame.init() #Initialisation de tous les modules pygame.
fenetre = pygame.display.set_mode(taille) #Création de la fenêtre
    graphique.

continuer = True
while continuer:
    for event in pygame.event.get(): #On parcourt tous les événements
        récupérés par pygame. On peut récupérer des informations sur
        chaque événement en fonction de son type (touche clavier
        appuyée, coordonnées de la souris...)
        if event.type == QUIT: #Si c'est un événement de fermeture de
            fenêtre (clic sur la croix...), on sort de la boucle.
            continuer = False
        if event.type == MOUSEBUTTONDOWN and event.button==1: #Si c'est
            un événement correspondant au bouton de souris gauche
            enfoncé
            rayon = randint(5,50) #rayon aléatoire
```

```
            couleur = couleurs[randint(0, 6)] #couleur aléatoire
            pygame.draw.circle(fenetre, couleur, event.pos , rayon, 0)
                #Les coordonnées de la souris sont dans event.pos Ces
                données sont récupérées automatiquement par pygame.
    pygame.display.flip() #Mise à jour de l'affichage en affichant le
        contenu du buffer à l'écran. Ici, on n'efface pas l'écran à
        chaque fois pour conserver tous les disques affichés.
pygame.quit()
```

5 Gestion des événements clavier

A chaque passage dans la boucle principale, on parcourt tous les événements récupérés par pygame (*for event in pygame.event.get() :*) et on teste si le type de chaque événement correspond à un événement clavier : KEYUP (touche relâchée), KEYDOWN (touche enfoncée).
Chaque événement *event* qui provient du clavier possède des informations qu'on peut utiliser : par exemple, *event.key* donne la touche qui a été enfoncée ou relâchée (K_a pour a, K_B pour B, K_UP pour la flèche du haut...)
Exemple de gestion des touches du clavier (les explications sont en commentaire) :

```
"""
    Déplace une balle grâce aux flèches du clavier.
"""

import pygame
from pygame.locals import * #Permet de charger des constantes comme
    KEYDOWN ou KEYUP... dans l'espace de nom de ce module. Ceci permet
    d'écrire KEYDOWN au lieu de pygame.KEYDOWN

pygame.init() #Initialisation de tous les modules pygame.

taille = largeur, hauteur = 800, 600
vitesse = 3
noir = 0, 0, 0

fenetre = pygame.display.set_mode(taille) #Création de la fenêtre
    graphique

balle = pygame.image.load("ball.png") #Chargement de l'image
ballerect = balle.get_rect() #On récupère le rectangle où se trouve la
    Surface balle
continuer = True
deplacement = 0,0
while continuer:
    for event in pygame.event.get(): #On parcourt tous les événements
        générés
        if event.type == QUIT: #Si c'est de type Quitter, on sort de la
            boucle
            continuer = False
        if event.type == KEYDOWN: #Si l'événement est de type Touche
            enfoncée, on gère les 4 cas
            if event.key == K_DOWN:#Flèche vers le bas: on augmente l'
                ordonnée et l'abscisse ne change pas, etc...
                deplacement = 0,vitesse
            if event.key == K_UP :
                deplacement = 0,-vitesse
            if event.key == K_RIGHT :
                deplacement = vitesse,0
            if event.key == K_LEFT :
                deplacement = -vitesse,0

    ballerect = ballerect.move(deplacement) #On déplace le rectangle
    fenetre.fill(noir) #On efface l'écran
    fenetre.blit(balle, ballerect) #On dessine (dans le buffer) la
```

```
        Surface balle là où se trouve le rectangle
    pygame.display.flip() #On met à jour l'affichage: le contenu du
        buffer est envoyé à l'écran
    pygame.time.Clock().tick(100) #On ralentit l'exécution pour les
        ordinateurs très rapides: la boucle est exécutée 100 fois par
        seconde au maximum
pygame.quit() #On quitte proprement pygame (nettoyage de la mémoire)
```

6 Création d'un bouton

Dans pygame, les boutons sont créés par le programmeur : c'est lui qui gère l'événement souris qui permet de savoir si on clique sur le bouton.
Le mieux est de créer une Surface qui représente le bouton et de dessiner dedans un rectangle et du texte, ou bien de charger dedans une image. Pour savoir si le curseur de la souris se trouve au-dessus du bouton, on utilise l'instruction *bouton_rectangle_position.collidepoint(event.pos)* qui vaut *True* si le curseur est au-dessus. *bouton_rectangle_position* représente le rectangle où se trouve la surface *bouton* obtenu avec *bouton_rectangle_position = bouton.get_rect()*.
event.pos représente la position de la souris.
Exemple de création d'un bouton avec dessin d'un rectangle (les explications sont en commentaire) :

```
"""
    Dessine un bouton rectangulaire avec du texte à l'écran sur lequel
        on peut cliquer. Voir le résultat dans la console.
"""
import pygame
from pygame.locals import * #Permet de charger des constantes comme
    KEYDOWN ou KEYUP... dans l'espace de nom de ce module. Ceci permet
    d'écrire KEYDOWN au lieu de pygame.KEYDOWN

taille = largeur, hauteur = 800, 600 #Taille de la fenêtre graphique
pygame.init() #Initialisation de tous les modules pygame.

fenetre = pygame.display.set_mode(taille) #Création de la fenêtre
    graphique.
largeur_bouton, hauteur_bouton = 150, 50 #Taille du bouton
#Une méthode pratique est de créer une Surface pour représenter le
    bouton. Ca évitera ensuite de faire des tests pénibles pour savoir
    si le curseur souris est au-dessus du bouton.
bouton = pygame.Surface((largeur_bouton, hauteur_bouton)) #On crée la
    Surface avec les bonnes dimensions
bouton.fill((180, 180, 180)) #On remplit le outon de gris
pygame.draw.rect(bouton, (255, 0, 0), bouton.get_rect(), 1)
police = pygame.font.SysFont('Arial', 20, bold=True) #On crée la policequ'
    on va utiliser pour le texte
texte = police.render("Cliquer ici", True, (0,128,0)) #On crée une
    Surface qui contient le texte à afficher
bouton.blit(texte, ((largeur_bouton-texte.get_width())/2,(
    hauteur_bouton-texte.get_height())/2))#On met le texte sur le
    bouton en le centrant

bouton_rectangle_position = bouton.get_rect()#On récupère le rectangle
    où se trouve le bouton
bouton_rectangle_position.center = (largeur/2, hauteur/2) #On place le
    rectangle au milieu de l'écran

continuer = True
while continuer:
    for event in pygame.event.get(): #On parcourt tous les événements
        récupérés par pygame.
        if event.type == QUIT: #Si c'est un événement de fermeture de
            fenêtre (clic sur la croix...), on sort de la boucle.
            continuer = False
        if event.type == MOUSEBUTTONDOWN and event.button==1 and
```

```
            bouton_rectangle_position.collidepoint(event.pos): #Si c'
                est un événement correspondant au bouton de souris gauche
                enfoncé et si la souris se trouve dans le rectange
                print("Vous avez cliqué")#On affiche un message dans la
                    console
                #On peut ajouter ici une fonction à exécuter
    fenetre . fill ((255 ,255 ,255) ) # On efface l' écran
    fenetre.blit(bouton, bouton_rectangle_position) #On dessine le
        bouton à l'endroit où se trouve le rectangle
    pygame.display.flip() #On met à jour l'affichage: tout ce qui a été
        créé dans le buffer est envoyé à l'écran
pygame.quit()
```

Exemple de création d'un bouton avec une image (les explications sont en commentaire) :

```
"""
    Dessine un bouton à l'écran sur lequel on peut cliquer.
"""
import pygame
from pygame.locals import * #Permet de charger des constantes comme
    KEYDOWN ou KEYUP... dans l'espace de nom de ce module. Ceci permet
    d'écrire KEYDOWN au lieu de pygame.KEYDOWN

taille = largeur, hauteur = 800, 600 #Taille de la fenêtre graphique
pygame.init() #Initialisation de tous les modules pygame.
fenetre = pygame.display.set_mode(taille) #Création de la fenêtre
    graphique.
bouton = pygame.image.load("bouton.png") #Chargement de l'image et
    création de l'objet bouton de type Surface
bouton_rectangle_position = bouton.get_rect()#On récupère le rectangle
    où se trouve la Surface bouton
bouton_rectangle_position.center = (largeur/2, hauteur/2) #On place le
    rectangle au milieu de l'écran

continuer = True
bouton_enfonce = False #Variable booléenne qui permet de savoir si le
    bouton a été enfoncé afin de le redécaler vers le haut lorsqu'on
    relâche le bouton de la souris
while continuer:
    for event in pygame.event.get(): #On parcourt tous les événements
        récupérés par pygame.
        if event.type == QUIT: #Si c'est un événement de fermeture de
            fenêtre (clic sur la croix...), on sort de la boucle.
            continuer = False
        if event . type == MOUSEBUTTONDOWN and event . button ==1 and
            bouton_rectangle_position.collidepoint(event.pos): #Si c'
            est un événement correspondant au bouton de souris gauche
            enfoncé et si la souris se trouve dans le rectange
            bouton_enfonce = True
            bouton_rectangle_position = bouton_rectangle_position . move
                (2,2) #On décale le bouton vers le bas
            #On peut ajouter ici une fonction à exécuter
        if event . type == MOUSEBUTTONUP and bouton_enfonce : #Si on
            relâche le bouton de la souris et que le bouton a été
            enfoncé
            bouton_rectangle_position = bouton_rectangle_position . move
                (-2,-2) #On décale le bouton vers le haut
            bouton_enfonce = False
    fenetre.fill((255,255,255)) #On efface l'écran
    fenetre.blit(bouton, bouton_rectangle_position) #On dessine le
        bouton à l'endroit où se trouve le rectangle
    pygame.display.flip() #On met à jour l'affichage: tout ce qui a été
```

```
        créé dans le buffer est envoyé à l'écran
pygame.quit()
```

7 Le son

Il existe un module, nommé *mixer*, qui est plutôt adapté aux sons brefs (streaming impossible), comme les bruitages. On utilisera les formats .wav ou .ogg et on fera une conversion au préalable pour les autres formats, avec le logiciel Audacity ou iTunes par exemple.
Exemple d'utilisation d'un son avec le module mixer (les explications sont en commentaire) :

```
"""
    Exemple d'utilisation du module mixer.
    Appuyer sur espace et laisser la touche enfoncée pour jouer.
    Relâcher la touche espace pour passer en pause.
    Appuyer sur la touche entrée pour recommencer à 0.
"""

import pygame
from pygame.locals import *

#Initialisation
pygame.init()
fenetre = pygame.display.set_mode((300,300))
son = pygame.mixer.Sound("son.wav")#On charge le son dans un objet de
    type Sound

continuer = True
en_pause = False #

while continuer:
   for event in pygame.event.get():
      if event.type == QUIT:
         continuer = False
      #Lancer le son si on appuie sur la touche espace et que ce n'est
          pas en pause
      if event.type == KEYDOWN and event.key == K_SPACE and en_pause ==
          False:
         son.play()
         en_pause = True
      #Sortir de pause si on appuie sur la touche espace et que c'est
          en pause
      if event.type == KEYDOWN and event.key == K_SPACE and en_pause ==
          True:
         pygame.mixer.unpause()
      #Mettre en pause si on relâche la touche espace
      if event.type == KEYUP and event.key == K_SPACE:
         pygame.mixer.pause()
      #Stopper le son si on appuie sur entrée, pour recommencer à 0
      if event.type == KEYDOWN and event.key == K_RETURN:
         son.stop()
         en_pause = False
pygame.quit()
```

Il existe un autre module, nommé *music* qui est plus adapté aux gros fichiers son car on peut utiliser le streaming (lecture de gros fichiers sans attendre la fin de leur chargement).
Il permet également de créer des listes de lecture. On utilise la syntaxe suivante :
pygame.mixer.music.load("musique.wav") pour charger le fichier son
pygame.mixer.music.queue("instruments.wav") pour ajouter un autre fichier son à la fin (liste de lecture)
pygame.mixer.music.play() pour lancer la lecture de la liste
pygame.mixer.music.stop() pour arrêter la lecture et revenir au début de la musique coupée (pas forcément la première)
pygame.mixer.music.pause() pour mettre en pause
pygame.mixer.music.unpause() pour sortir de la pause

8 Liens utiles

- Documentation officielle de pygame (choisir un module ou thème en haut. Pour chaque fonction, des exemples sont fournis en cliquant sur le bouton "Search examples for...")
- Cours pygame sur Openclassrooms
- Tutoriels Pygame sur developpez.com