Code de base d'un jeu PyGame

Voici le code de base minimum nécessaire pour pouvoir développer un jeu avec PyGame.

Tous les jeux que tu vas développer avec pygame auront cette structure minimale de base.

Il n'est pas nécessaire de connaître la structure par cœur dans le cadre de ce cours, mais il est très intéressant de comprendre comment elle fonctionne.

Ton programme ci-dessous fonctionnera si ton écran est noir avec une petite boîte carrée affichée.

```python
import pygame
import sys

# Initialisation
pygame.init()

# Création de la fenêtre de jeu
screen = pygame.display.set_mode((800, 600))
clock = pygame.time.Clock()

# Boucle de jeu
running = True
while running:

    # Permet de quitter le programme proprement
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Mise à jour de l'écran
    pygame.display.flip()

    # Limiter la vitesse d'exécution (60 FPS)
    clock.tick(60)

pygame.quit()
sys.exit()
```

🚀 pygame.init()

Description

La fonction `pygame.init()` permet d'**initialiser tous les modules** de la bibliothèque PyGame nécessaires au bon fonctionnement du jeu (graphismes, son, joystick, etc.). C'est la **toute première chose à faire** avant d'utiliser PyGame.

📜 Syntaxe

```python
pygame.init()
```

🎯 À quoi ça sert ?

PyGame est composée de plusieurs modules : affichage, son, police de texte, gestion des entrées…
`pygame.init()` va essayer d'activer tous ces modules. Sans cette étape, beaucoup de fonctions PyGame ne fonctionneront pas correctement (voire pas du tout).

✅ Exemple

```
import pygame
pygame.init()
```

Une fois que c'est fait, tu peux créer une fenêtre, charger des images, jouer des sons, etc.

🧱 `screen = pygame.display.set_mode()`

La fonction `pygame.display.set_mode()` permet de **créer la fenêtre principale** du jeu, là où tous les éléments seront affichés. C'est l'une des premières fonctions à appeler après avoir initialisé PyGame avec `pygame.init()`.

📝 Syntaxe

```
surface = pygame.display.set_mode((largeur, hauteur))
```

📌 Paramètres

Paramètre	Type	Description
`(largeur, hauteur)`	`tuple`	La taille de la fenêtre de jeu, en pixels (largeur x hauteur)

✅ Exemple

```
screen = pygame.display.set_mode((800, 600))
```

- Crée une fenêtre de **800 pixels de large** et **600 pixels de haut**.
- La variable `screen` contient la surface sur laquelle on pourra dessiner (`blit`, `draw`, etc.).

💡 Bonnes pratiques

- Stocke toujours le résultat dans une variable (`screen`) : c'est **la surface principale** sur laquelle tu dessines.
- Utilise des **variables** pour la taille de l'écran (ex. `LARGEUR_ECRAN = 800`) pour pouvoir les réutiliser ailleurs.

🕒 `clock = pygame.time.Clock()`

Cette instruction crée un **objet "horloge"** dans PyGame.
Il est utilisé pour **contrôler la vitesse d'exécution du jeu**, c'est-à-dire le **nombre d'images par seconde** (FPS = frames per second).

🎯 À quoi ça sert ?

Dans un jeu, il est important que la boucle de jeu **ne tourne pas trop vite**.
Sinon, le jeu serait **trop rapide**, ou tournerait à une vitesse différente selon l'ordinateur.

Avec un objet `Clock`, tu peux **ralentir volontairement** la boucle de jeu pour avoir un rythme stable.

🔁 `while running:`

📝 Ce que ça fait

Ce code crée une **boucle infinie contrôlée par une variable**. C'est une **structure de base dans un jeu vidéo**, souvent appelée la **boucle de jeu** (*game loop*).

🧠 Explication

```
running = True
while running:
    # instructions exécutées en boucle
```

- `running` est une variable booléenne (vrai ou faux).
- Tant que `running` vaut `True`, le code **à l'intérieur de la boucle** continue à s'exécuter.
- Dès qu'on met `running = False`, la boucle s'arrête.

🎮 À quoi ça sert dans un jeu ?

C'est la **boucle principale du jeu**. Elle tourne en continu pour :

1. Lire les entrées clavier/souris
2. Mettre à jour les positions, les collisions, les scores…
3. Redessiner l'écran

🔄 `for event in pygame.event.get():`

Ce groupe d'instructions permet de vérifier tous les événements qui se passent dans pygame. Dans un premier temps, pour notre première série d'exercices, le plus important est simplement de se rappeler qu'**elle permet de quitter le jeu proprement**.

🕒 `clock.tick(60)`

`clock.tick(60)` limite la boucle à **60 tours par seconde**, donc le jeu s'exécutera à **60 FPS** maximum.

🔄 `pygame.display.flip()`

📝 Description

`pygame.display.flip()` met à **jour l'écran complet** avec tout ce qui a été dessiné depuis le début de la boucle. C'est l'étape **indispensable** pour que le joueur voie les changements à l'écran : sans elle, rien ne s'affiche.

🎯 Pourquoi c’est important ?

PyGame utilise un système de **double affichage** (*double buffering*) :

- Tu dessines **dans l’ombre** (sur une surface mémoire invisible).
- Ensuite, `flip()` **montre à l’écran** ce que tu viens de préparer.

Cela évite les scintillements et rend le jeu fluide.

✅ Exemple typique dans une boucle de jeu

```
while running:
    screen.fill((0, 0, 0))              # Effacer l'écran
    pygame.draw.rect(screen, rouge, ...) # Dessiner un élément
    pygame.display.flip()               # Mettre à jour l'affichage
```

💡 Bonnes pratiques

- Utilise **une seule fois** `flip()` à la fin de chaque boucle.
- Dessine **tout** avant de l’appeler.