But

Le but est d'utiliser la POO afin d'implémenter le jeu Pong.

La problématique

Une balle évolue dans une fenêtre graphique dont elle ne doit jamais sortir sur les côtés latéraux (sinon le jeu s'arrête). La balle rebondit sur les bords inférieurs et supérieurs. Chaque joueur contrôle une raquette qu'il peut faire monter ou descendre. Si la balle touche la raquette, celle-là rebondit sur cette dernière, sinon la balle passe la raquette et le joueur contrôlant la raquette passée perd.

Dans un premier temps, vous implémenterez le jeu avec un joueur (celui à droite), et dans ce cas la balle rebondira également sur le bord gauche de la fenêtre. La partie sera perdue si la balle dépasse la raquette de droite.

Dans un second temps, vous ajouterez le deuxième joueur.

Les touches bas/haut serviront pour diriger la raquette de droite, les touches s/z celle de gauche s'il y a deux joueurs.

Exemple / aide

◇ On pourra importer le module `projet_pong_pygame.py` via la ligne `import projet_pong_pygame as ppg`.
Ce module contient des couleurs codées en RGB, les dimensions de la fenêtre dans laquelle évoluera la balle et les raquettes (`DIMX` et `DIMY`), différentes fonctions d'affichage (lire le module fourni et/ou taper `help(projet_pong_pygame)`).

◇ En `pygame`, l'axe des abscisses est bien orientée mais l'axe des ordonnées est orientée vers le bas.

◇ On peut générer des entiers aléatoirement entre 5 et 15 avec

```
from random import randint
nb_alea = randint(5,15)
```

◇ Lorsque que la balle touche le mur du haut, sa vitesse (vx,vy) devient $(vx,-vy)$.

◇ La boucle principale de la fonction `main() -> None` pourra être composée comme suit :

```
ppg.init()
# initialisation de la balle et de(s) raquette(s)
bjouer = True
while bjouer :
    # effacement de la balle
    # mise à jour des coordonnées / vitesses de la balle

    ppg.miseAJour_actions()
    mv = ppg.procEvent( raq1 ) # mouvement verticale de la raquette
    # si mv!=0, effacement de la raquette et mise à jour de ses coordonnées
    # affichage de la balle et de(s) raquette(s)

    # vérification du contact balle/raquette

    miseAJour_affichage()
    if ppg.sortie(): bjouer = False
    ppg.wait( 100 - 10*LEVEL )
# message de fin + temporisation
ppg.fin()
```

Étapes

Par la suite, on ne commentera pas les méthodes (seulement la classe englobante) mais on définira les types de leur entrées et sorties. Ceci dit, on continuera à commenter les fonctions.

Rq : on continuera à mettre les tests des fonctions et méthodes dans un commentaire triple quotes (`"""`).

Rq2 : excepté pour les constructeurs, le premier argument des méthodes n'est pas spécifié.

a) Dans la section T^le NSI du site [forhan.maths.free.fr], télécharger le fichier `projet_pong_pygame.py`.

b) Compléter la classe `Balle` pour laquelle vous définirez outre le constructeur, la méthode `__str__() -> str`.
Par défaut, la vitesse de la balle sera initialisée à (0,0) et son rayon vaudra 5. Sans le dernier paramètre de couleur, la balle sera jaune.

rq : la méthode `__str__() -> str` sera définie pour que `print( Balle((2,3)) )` affiche `"Balle : Coordonnées (2,3) / Vitesse (0,0)"`.

Balle
coordonnees (tuple)
vitesse (tuple)
color (tuple)
rayon (int)

```
class Balle:
    """ commentaires de la classe """
    def __init__(self, x: int, y: int, c: tuple=(255,255,0)) -> 'Balle':
        ...
```

c) Définir les méthodes `get_color() -> tuple` et `get_rayon() -> int`, qui permettent l'accès de l'attribut codant la couleur de balle et du rayon.

d) Définir également les méthodes `get_x() -> int`, `get_y() -> int`, `set_x(x: int) -> None` et `set_y(y: int) -> None`, qui permettent l'accès et la modification des coordonnées.

e) Définir également les méthodes `get_vx() -> int`, `get_vy() -> int`, `set_vx(vx: int) -> None` et `set_vy(vy: int) -> None`, qui permettent l'accès et la modification des vitesses.
À ce stade, une fois `ppg.init()` exécuté, vous pouvez afficher la balle avec `ppg.miseAJour_balle(Balle(2,3))` suivi de `ppg.miseAJour_affichage()`.

On souhaite maintenant définir la classe `Raquette`.

f) Compléter la classe `Raquette` pour laquelle vous définirez outre le constructeur, la méthode `__repr__() -> str`.
Par défaut, la vitesse (verticale) de la raquette sera initialisée à 0, sa longueur à 50 et sa largeur à 4. Sans le dernier paramètre de couleur, la raquette sera blanche.

rq : la méthode `__repr__() -> str` sera définie pour que `Raquette(2,3)` renvoie dans le shell `"Raquette(2,3)"`.

Raquette
coordonnees (tuple)
vitesse (int)
color (tuple)
longueur(int)
largeur(int)

```
class Raquette:
    """ commentaires de la classe """
    def __init__(self, x: int, y: int, c: tuple=(255,255,255)) -> 'Raquette':
        ...
```

g) Définir les méthodes `get_color() -> tuple`, `get_vitesse() -> int`, `get_longueur() -> int` et `get_largeur() -> int`, qui permettent l'accès des attributs codant la couleur, la vitesse, la longueur et la largeur de la raquette.

h) Définir également les méthodes `get_x() -> int`, `get_y() -> int`, `set_x(x: int) -> None` et `set_y(y: int) -> None`, qui permettent l'accès et la modification des coordonnées.
À ce stade, une fois `ppg.init()` exécuté, vous pouvez afficher la raquette avec `ppg.miseAJour_raquette( Raquette(10,ppg.DIMY//2) )` suivi de `ppg.miseAJour_affichage()`.

i) Définir également la méthode `touche_y(y: int) -> bool`, qui permet de savoir si l'ordonnée `y` est à hauteur de la raquette.

j) Définir également la méthode `touche_x(x: int, vx: int) -> bool`, qui permet de savoir si un objet à une abscisse `x` avec une vitesse `vx` atteindra la raquette une fois le déplacement `vx` effectué.

⚠ selon le sens du déplacement, le test conditionnel est légèrement différent.

k) Proposer une procédure `main() -> None`, qui affiche et fasse bouger la balle (sans la raquette) à l'aide des fonctions du module `projet_pong_pygame`.

l) Proposer une procédure `main() -> None`, qui affiche et fasse bouger la raquette (sans la balle) à l'aide des fonctions du module `projet_pong_pygame`.

m) Finaliser votre procédure `main() -> None`, afin de pouvoir jouer à un joueur.

n) Modifier votre procédure `main() -> None`, afin de joueur à deux joueurs (on devra utiliser la fonction `ppg.procEvent2()`).

o) Approfondissement :

- ◇ Chronométrer le temps resté à jouer et en faire un score.
- ◇ Proposer l'ajout d'une deuxième balle ou une accélération de la balle au bout d'un certain temps (vous pouvez changer la couleur de la balle en fonction de celle-ci).
- ◇ Proposer une raquette avec plusieurs zones d'effets (*e.g.* zones extérieures qui augmentent l'angle, zone intérieure qui augmente la vitesse et diminue l'angle, ...)
- ◇ Proposer de sauvegarder les cinq meilleurs scores dans un fichier `projet_pong_scores.csv`
- ◇ Proposer un menu() dans lequel vous pouvez choisir le nombre de joueurs, de balles, la taille de la raquette, etc.

nom,score
Python,360
Ping,288
Pong,100

Rendu

À la fin du temps imparti, chaque groupe présentera en 3 minutes à l'oral les résultats qu'il a obtenu et déposera/enverra via l'ENT le fichier `TNSI-pong-NOM1-NOM2.py` (+ le module pygame s'il a été modifié) contenant le code `python`.