La Programmation Orientée Objet

Structures de données

Contenu : Vocabulaire de la programmation objet : classes, attributs, méthodes, objets.

Capacités attendues : Écrire la définition d'une classe.
Accéder aux attributs et méthodes d'une classe.

EXERCICE 1

Extrait sujet Métropole 2022 Jour 1

Cet exercice porte sur la Programmation Orientée Objet.
Les participants à un jeu de LaserGame sont répartis en équipes et s'affrontent dans ce jeu de tir, revêtus d'une veste à capteurs et munis d'une arme factice émettant des infrarouges.
Les ordinateurs embarqués dans ces vestes utilisent la programmation orientée objet pour modéliser les joueurs. La classe ***Joueur*** est définie comme suit :

```
class Joueur:
    def __init__(self, pseudo, identifiant, equipe) :
        ''' constructeur '''
        self.pseudo = pseudo
        self.equipe = equipe
        self.id = identifiant
        self.nb_de_tirs_emis = 0
        self.liste_id_tirs_recus = []
        self.est_actif = True
    def tire(self) :
        '''méthode déclenchée par l'appui sur la gachette'''
        if self.est_actif == True:
            self.nb_de_tirs_emis = self.nb_de_tirs_emis + 1
    def est_determine(self):
        '''methode qui renvoie True si le joueur réalise un
        grand nombre de tirs'''
        return self.nb_de_tirs_emis > 500
    def subit_un_tir(self, id_recu) :
        '''méthode déclenchée par les capteurs de la
        veste'''
        if self.est_actif == True :
            self.est_actif = False
            self.liste_id_tirs_recus.append(id_recu)
```

1. Parmi les instructions suivantes, recopier celle qui permet de déclarer un objet *joueur1*, instance de la classe ***Joueur***, correspondant à un joueur dont le pseudo est "*Sniper*", dont l'identifiant est *319* et qui est intégré à l'équipe "*A*" :
   Instruction 1 : joueur1 = ["Sniper", 319, "A"]
   Instruction 2 : joueur1 = new Joueur["Sniper", 319, "A"]
   Instruction 3 : joueur1 = Joueur("Sniper", 319, "A")
   Instruction 4 : joueur1 = Joueur{"pseudo":"Sniper", "id":319, "equipe":"A "}
2. La méthode **subit_un_tir** réalise les actions suivantes :
   Lorsqu'un joueur actif subit un tir capté par sa veste, l'identifiant du tireur est ajouté à l'attribut *liste_id_tirs_recus* et l'attribut *est_actif* prend la valeur *False* (le joueur est désactivé). Il doit alors revenir à son camp de base pour être de nouveau actif.
   a. Écrire la méthode **redevenir_actif** qui rend à nouveau le joueur actif uniquement s'il était précédemment désactivé.
   b. Écrire la méthode **nb_de_tirs_recus** qui renvoie le nombre de tirs reçus par un joueur en utilisant son attribut liste_id_tirs_recus.
3. Lorsque la partie est terminée, les participants rejoignent leur camp de base respectif où un ordinateur, qui utilise la classe ***Base***, récupère les données.
   La classe ***Base*** est définie par :
   - ses attributs :
     - *equipe* : nom de l'équipe (str). Par exemple, "A"
     - *liste_des_id_de_l_equipe* qui correspond à la liste (list) des identifiants connus des joueurs de l'équipe
     - *score* : score (int) de l'équipe, dont la valeur initiale est 1000
   - ses méthodes :
     - **est_un_id_allie** qui renvoie **True** si l'identifiant passé en paramètre est un identifiant d'un joueur de l'équipe, **False** sinon
     - **incremente_score** qui fait varier l'attribut *score* du nombre passé en paramètre
     - **collecte_information** qui récupère les statistiques d'un participant passé en paramètre (instance de la classe Joueur) pour calculer le score de l'équipe.

```
def collecte_information(self,participant):
    if participant.equipe == self.equipe : # test 1
        for id in participant.liste_id_tirs_recus:
            if self.est_un_id_allie(id): # test 2
                self.incremente_score(-20)
            else:
                self.incremente_score(-10)
```

a. Indiquer le numéro du test (**test 1** ou **test 2**) qui permet de vérifier qu'en fin de partie un participant égaré n'a pas rejoint par erreur la base adverse.

b. Décrire comment varie quantitativement le score de la base lorsqu'un joueur de cette équipe a été touché par le tir d'un coéquipier.

On souhaite accorder à la base un bonus de 40 points pour chaque joueur particulièrement déterminé (qui réalise un grand nombre de tirs).

4. Recopier et compléter, en utilisant les méthodes des classes **Joueur** et **Base**, les 2 lignes de codes suivantes qu'il faut ajouter à la fin de la méthode **collecte_information** :

..... *#si le participant réalise un grand nombre de tirs*

..... *#le score de la Base augmente de 40*

EXERCICE 2

Extrait sujet Mayotte 2022 Jour 1

Cet exercice porte sur les structures de données (programmation objet)

Dans un jeu de plateforme, des bulles de couleurs et de diamètres différents se déplacent de manière aléatoire. À chaque fois qu'une bulle touche une bulle plus grande, la petite cède son contenu à la plus grande, et donc celle-ci augmente de surface. Par exemple, si une bulle de 1 cm² rencontre une bulle de 4 cm², la petite bulle disparait et la plus grande a désormais une surface de 5 cm². À chaque collision, la vitesse de la grande bulle est réduite de moitié.

Le développeur a choisi de coder en Python, chaque bulle est un objet disposant entre autres des attributs suivants :

- *xc*, *yc* sont deux entiers, les coordonnées du pixel placé au centre de la bulle,
- *rayon* est un entier, le rayon de la bulle en pixels,
- *couleur* est un entier, la couleur de la bulle,
- *dirx*, *diry* sont deux décimaux (float) qui déterminent les déplacements à l'horizontale et à la verticale à chaque fois que la bulle se déplace. Ces deux valeurs déterminent donc la direction et la vitesse de la bulle. Par exemple si *dirx* vaut 0.5 et *diry* vaut 0.0, la bulle se déplace vers la droite uniquement alors que si *dirx* vaut -1.0 et *diry* vaut 0.0, la bulle se déplace vers la gauche et deux fois plus vite que précédemment.

On suppose que toutes les fonctions de la bibliothèque math ont déjà été importées par l'instruction **from math import ***.

La fonction **randint** de la bibliothèque **random** prend en paramètre deux entiers et renvoie un entier aléatoire dans la plage définie par les deux paramètres.

Exemple : *randint (-1, 5)* peut renvoyer une des valeurs suivantes : -1, 0, 1, 2, 3, 4, 5.

1. Pour simplifier, on se limitera à un jeu de six bulles. Au départ, on crée une liste appelée *Mousse* de longueur six contenant six emplacements vides :

Mousse = [None, None, None, None, None, None]

Le code ci-dessous montre le début du programme et notamment la structure définition de la classe nommée **Cbulle** ainsi que le code permettant le déplacement d'une bulle.

```
from random import randint
from math import *
class Cbulle:
    def __init__(self):
        self.xc = randint(0, 100)
        self.yc = randint(0, 100)
        self.rayon = randint(0, 10)
        self.dirx = float(randint(-1, 1)) # dirx et diry valent
        self.diry = float(randint(-1, 1)) # -1.0 ou 0.0. ou 1.0
        self.couleur = randint(1,65535)
    def bouge(self):
        # déplace la bulle
        self.xc = self.xc + self.dirx
        self.yc = self.yc + self.diry
```

On crée les six bulles une à une et ces objets sont stockés dans les emplacements vides de la liste Mousse.

Mousse = [bulle1, bulle2, bulle3, bulle4, bulle5, bulle6]

Lors d'une collision, la bulle la plus petite disparait et est remplacée dans la liste par la valeur *None* tandis que la plus grosse a sa surface qui augmente.

Au cours d'une partie, si une ou plusieurs bulles ont disparue, le programme peut en introduire de nouvelles dans le jeu : dans ce cas, lorsqu'une nouvelle bulle apparaît, elle remplace le premier None de la liste Mousse.

a. Recopier les quatre dernières lignes et compléter les ..... du code python ci-dessous.

```
def donnePremierIndiceLibre(Mousse):
    """
    Mousse est une liste.
    La fonction doit renvoyer l'indice du premier
    emplacement libre (contenant None) dans la liste Mousse
    ou renvoyer 6 en l'absence d'un emplacement libre dans
    Mousse.
    """
    i = 0
    while ..... and Mousse[i] != None :
        .....
    return i
```

b. Lorsque le jeu crée une bulle (instance de la classe Cbulle), il doit ensuite la placer dans la liste Mousse à la place d'un None. Écrire la fonction **placeBulle(B)** qui reçoit en paramètre un objet de type ***Cbulle*** et qui place cet objet dans la liste Mousse. Cette fonction ne renvoie rien, mais la liste Mousse est modifiée. Si aucun emplacement n'est disponible, la fonction ne modifie rien.

2. Pour le bon déroulement du jeu, on a besoin aussi d'une fonction **bullesEnContact(B1, B2)** qui renvoie **True** si la bulle B2 touche la bulle B1 et **False** dans le cas contraire.
On peut remarquer que deux bulles sont en contact si la distance qui sépare leur centre est inférieure ou égale à la somme de leurs rayons.
On dispose de la fonction **distanceEntreBulles(B1, B2)** qui calcule et renvoie la distance entre les centres de bulles B1 et B2.
Écrire la fonction **bullesEnContact(B1, B2)**.

3. Quand une petite bulle touche une plus grosse bulle, on appelle la fonction collision, ci-dessous, où *indPetite* est l'indice de la petite bulle et *indGrosse* l'indice de la grosse bulle dans **Mousse**.
Recopier et compléter les ..... de la fonction collision.

```
def collision(indPetite, indGrosse, mousse) :
    """
    Absorption de la plus petite bulle d'indice indPetite
    par la plus grosse bulle d'indice indGrosse. Aucun test
    n'est réalisé sur les positions.
    """
    # calcul du nouveau rayon de la grosse bulle
    surfPetite = pi*Mousse [indPetite].rayon**2
    surfGrosse = pi*Mousse [indGrosse].rayon**2
    surfGrosseApresCollision = .....
    rayonGrosseApresCollision = sqrt(surfGrosseApresCollision/pi)
    #réduction de 50% de la vitesse de la grosse bulle
    Mousse[indGrosse].dirx = .....
    Mousse [indGrosse].diry = .....
    #suppression de la petite bulle dans Mousse
    ……
```