Sujet BAC 4 : Programmation Orientée Objet

1. Exercice n°1 : Métropole J1 : Ex.5 - 2022

Exercice 1 P.O.O - Laser Game

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):
        " Appelle le constructeur et initialise "
        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 gâchette "
        if self.est_actif:
            self.nb_de_tirs_emis += 1

    def est_determine(self):
        " Le joueur réalise-t-il un grand nombre de tirs ? "
        return self.nb_de_tirs_emis > 500  # Un booléen est renvoyé.

    def subit_un_tir(self, id_recu):
        " Méthode déclenchée par les capteurs de la veste "
        if self.est_actif:
            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 `joueur_1`, 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** : `joueur_1 = ["Sniper", 319, "A"]`
- **Instruction 2** : `joueur_1 = new Joueur["Sniper", 319, "A"]`
- **Instruction 3** : `joueur_1 = Joueur("Sniper", 319, "A")`
- **Instruction 4** : `joueur_1 = 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.

2.a. Écrire la méthode `redevenir_actif` qui rend à nouveau le joueur actif uniquement s'il était précédemment désactivé.

2.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 détermine si l'identifiant passé en paramètre est un identifiant d'un joueur de l'équipe, en renvoyant un booléen,
  - `decremente_score` qui diminue 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.

🐍 **Script Python**

```
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.decremente_score(20)
            else:
                self.decremente_score(10)
```

3.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.

3.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` :

🐍 **Script Python**

```
...         # si le participant réalise un grand nombre de tirs
    ...     # le score de la Base augmente de 40
```

2. Exercice 2 : D'après 2022, Centres étrangers, J2, Ex. 4

Exercice 2 P.O.O - La Bataille

Simon souhaite créer en Python le jeu de cartes « la bataille » pour deux joueurs. Les questions qui suivent demandent de reprogrammer quelques fonctions du jeu.

Règles du jeu de la bataille

Préparation

- Distribuer toutes les cartes aux deux joueurs.
- Les joueurs ne prennent pas connaissance de leurs cartes et les laissent en tas face cachée devant eux.

Déroulement

- À chaque tour, chaque joueur dévoile la carte du haut de son tas.
- Le joueur qui présente la carte ayant la plus haute valeur emporte les deux cartes qu'il place sous son tas.
- Les **valeurs des cartes** sont : dans l-ordre de la plus forte à la plus faible : As, Roi, Dame, Valet, 10, 9, 8, 7, 6, 5, 4, 3 et 2 (la plus faible)

Si deux cartes sont de même valeur, il y a « bataille ».

- Chaque joueur pose alors une carte face cachée, suivie d'une carte face visible sur la carte dévoilée précédemment.
- On recommence l'opération s'il y a de nouveau une bataille sinon, le joueur ayant la valeur la plus forte emporte tout le tas.

Lorsque l'un des joueurs possède toutes les cartes du jeu, la partie s'arrête et ce dernier gagne.

Pour cela Simon crée une classe Python `Carte`. Chaque instance de la classe a deux attributs : un pour sa `valeur` et un pour sa `couleur`. Il donne au valet la valeur 11, à la dame la valeur 12, au roi la

valeur \(13\) et à l'as la valeur \(14\). La couleur est une chaine de caractères : `"trefle"`, `"carreau"`, `"coeur"` ou `"pique"`.

1. Simon a écrit la classe Python `Carte` suivante, ayant deux attributs `valeur` et `couleur`, et dont le constructeur prend deux arguments : `val` et `coul`.

1.a. Recopier et compléter les `...` des lignes 3 et 4 ci-dessous.

```
1   class Carte:
2       def __init__(self, val, coul):
3           ... .valeur = ...
4           ... = coul
```

1.b. Parmi les propositions ci-dessous quelle instruction permet de créer l'objet « 7 de cœur » sous le nom `c7` ?

- `c7.__init__(self, 7, "coeur")`
- `c7 = Carte(self, 7, "coeur")`
- `c7 = Carte(7, "coeur")`
- `from Carte import 7, "coeur"`

2. On souhaite créer le jeu de cartes. Pour cela, on écrit une fonction `initialiser` :

- sans paramètre
- qui renvoie une liste de 52 objets de la classe `Carte` représentant les 52 cartes du jeu.

Voici une proposition de code. Recopier et compléter les lignes suivantes pour que la fonction réponde à la demande :

🐍 **Script Python**

```
def initialiser() :
    jeu = []
    for coul in ["coeur", "carreau", "trefle", "pique"]:
    for val in range(...):
        carte_cree = ...
        jeu.append(carte_cree)
    return jeu
```

3. On rappelle que dans une partie de bataille, les deux joueurs tirent chacun une carte du dessus de leur tas, et celui qui tire la carte la plus forte remporte les deux cartes et les place en dessous de son tas.

Parmi les structures linéaires de données suivantes : Tableau, File, Pile, quelle est celle qui modélise le mieux un tas de cartes dans ce jeu de la bataille ? Justifier votre choix.

4. Écrire une fonction `comparer` qui prend en paramètres deux objets de la classe Carte : `carte_1`, `carte_2`. Cette fonction renvoie :

- \(0\) si la valeur des deux cartes est identique ;

- \(1\) si la carte `carte_1` a une valeur strictement plus forte que celle de `carte_2` ;
- \(-1\) si la carte `carte_2` a une valeur strictement plus forte que celle de `carte_1` .