

Olympiades inter académique d'informatique 2025 - Classe de première -

Durée de l'épreuve : 3 heures

L'épreuve se déroule par équipe et sur ordinateur.

Le sujet est constitué de parties indépendantes qui peuvent être traitées dans l'ordre souhaité. Un dossier est mis à disposition des équipes avec les fichiers requis pour certaines questions.

Le dossier intitulé **SUJET** contenant les fichiers requis pour traiter les questions sera à mettre à disposition des équipes.

Les fichiers complétés par les candidats seront à joindre dans un unique dossier compressé et qui seront nommés de la façon suivante (sans espace ni accents) :

- NomEtablissement_VilleEtablissement_EquipeXX_Quadball.py
(pour les scripts complétés de la partie match de *Quadball*)
- NomEtablissement_VilleEtablissement_EquipeXX_Programme_blason.py
(pour les scripts complétés de la partie Blason de l'équipe)
- NomEtablissement_VilleEtablissement_EquipeXX_equipe.odt
où le numéro de l'équipe sera attribué par vos soins.

Le dernier fichier (equipe.odt) recense les noms des membres de l'équipe.

Les calculatrices sont autorisées selon la réglementation en vigueur.

Les démarches, même incomplètes seront prises en compte dans l'évaluation.

Un match de Quadball

Le *Quadball* est une adaptation réelle du sport fictif de Harry Potter. Il existe d'ailleurs une fédération française de *Quadball*, ainsi qu'une équipe représentant la France.



Document 1 : Match de Quadball (source : wikipedia, article Quadball)

Règles simplifiées du jeu

Chaque équipe est formée de sept joueurs chevauchant des balais volants fictifs. L'objectif est de marquer plus de points que l'équipe adverse.

L'équipement de jeu est constitué des objets suivants :

| | | |
|---|---|---|
|  |  |  |
| Un Souaffle | Un Vif d'or (porté par un joueur qui n'appartient à aucune des deux équipes) | Trois Cognards |

Pendant le jeu :

- lancer le *Souaffle* dans les anneaux adverses rapporte 10 points par but ;
- attraper le *Vif d'or* rapporte 60 points.

Les trois *Cognards* servent à diminuer la force et la précision d'un joueur lorsque celui-ci est touché.

Le match se termine lorsque le *Vif d'or* est attrapé.

Une équipe est composée de :

- 3 *poursuiveurs*, qui sont les seuls à pouvoir marquer des buts ;
- 2 *batteurs*, qui sont les seuls à gérer les *Cognards* ;
- 1 *gardien*, qui défend les anneaux ;
- 1 *attrapeur*, qui est le seul à pouvoir attraper le *Vif d'or*.

Première tâche à accomplir : créer un programme Python pour simuler un match entre deux équipes.

Deuxième tâche à accomplir : dessiner le blason d'une équipe de *Quadball*.

1^{re} tâche : Simulation d'un match

Partie 1 : Création des structures de données pour les joueurs et les équipes

Chaque joueur possède un ensemble unique d'attributs qui définissent son identité et ses capacités sur le terrain :

Identité :

- **Nom** : L'identifiant personnel du joueur
- **Poste** : La position spécifique occupée au sein de l'équipe

Attributs de performance :

- **Vitesse** : Un entier de 0 à 100 (0 représente l'immobilité totale et 100 indique la vélocité maximale)
- **Précision** : Un entier de 1 à 20, représentant la précision des tirs
- **Force** : Un entier de 0 à 10, mesurant la puissance physique

Pour exemple voici la composition de l'équipe ayant pour nom les *Firewall Fighters* :

| nom | poste | vitesse | précision | force |
|--------|-------------|---------|-----------|-------|
| Aria | Attrapeur | 95 | 18 | 6 |
| Hugo | Gardien | 75 | 16 | 8 |
| Brutus | Batteur | 70 | 14 | 10 |
| Fiona | Batteur | 72 | 15 | 9 |
| Zephyr | Poursuiveur | 88 | 17 | 7 |
| Luna | Poursuiveur | 85 | 19 | 6 |
| Orion | Poursuiveur | 87 | 16 | 8 |

1.1. Créer en langage Python une fonction nommée `creer_joueur` telle qu'à l'exécution, `creer_joueur(nom, poste, vitesse, precision, force)` renvoie un dictionnaire représentant **tous** les attributs d'un joueur.

1.2. Créer une fonction nommée `creer_equipe` qui initialise la structure de base d'une équipe de *Quadball* , dont les spécifications sont données page suivante.

Spécifications de la fonction `creer_equipe`

1. Paramètre d'entrée :

- `nom` : une chaîne de caractères représentant le nom de l'équipe

2. Valeur de retour

Un dictionnaire contenant les éléments suivants :

- `"nom"` : le nom de l'équipe (tel que fourni en paramètre)
- `"joueurs"` : une liste vide, prête à accueillir la liste des futurs joueurs de l'équipe
- `"Score"` : un compteur initialisé à 0, représentant le score du match en cours
- `"victoires"` : un compteur initialisé à 0, représentant le nombre de victoires de l'équipe
- `"vif"` : un booléen, initialisé à `False`, indiquant si le Vif d'or a été attrapé par l'équipe

1.3. Créer une fonction `ajouter_joueur` telle qu'à l'exécution, `ajouter_joueur(equipe, joueur)` ajoute un joueur à une équipe si celle-ci n'est pas complète (moins de 7 joueurs).

Partie 2 : Simulation des actions d'un joueur

On veut simuler l'action des joueurs par une fonction. Celle-ci doit renvoyer une chaîne de caractères décrivant l'action effectuée, selon les règles suivantes :

- Si le joueur est trop fatigué (probabilité basée sur son niveau de force), il trébuche et ne peut pas agir. La fonction renvoie la chaîne de caractères : "trébuche à cause du manque de force".
- Si le poste du joueur est *poursuiveur* :
 - Avec une probabilité basée sur sa précision, il peut marquer un but. La fonction incrémente le compteur `Score` et renvoie la chaîne de caractères : "marque un but !".
 - Sinon, il rate son tir et la fonction renvoie la chaîne de caractères : "rate son tir".
- Si le poste du joueur est *attrapeur* :
 - Avec une probabilité basée sur la moyenne de sa vitesse et sa précision, il peut attraper le *Vif d'or*. La fonction affecte au booléen `vifs` la valeur `True` et renvoie la chaîne de caractères : "attrape le Vif d'or !".
 - Sinon, il rate sa tentative et la fonction renvoie la chaîne de caractère :: "rate le Vif d'or".

- Pour tout autre poste, la fonction renvoie simplement : "défend".

Créer une fonction `action_joueur` telle qu'à l'exécution, `action_joueur(equipe, joueur)` renvoie la chaîne de caractère correspondant à l'action effectuée. Cette fonction pourra utiliser `random.random()` pour générer des nombres aléatoires compris entre 0 et 1.

Partie 3 : Simuler un match

Pour simuler un match de *Quadball*, il faut créer une fonction nommée `simuler_match`. Cette fonction prendra en entrée deux dictionnaires, `equipe1` et `equipe2`, représentant les équipes qui s'affrontent.

Chacun de ces deux dictionnaires d'équipe contiendra des informations clés telles que le nom de l'équipe, la liste de ses joueurs (eux-mêmes représentés par des dictionnaires avec leurs caractéristiques), le score de l'équipe (initialisé à 0 au début du match) et le nombre de victoires accumulées.

On simulera un match en 20 tours de boucle maximum. Avant de commencer, les scores seront remis à zéro et la fatigue des joueurs sera réinitialisée. À chaque tour, chaque joueur des deux équipes effectuera une action. Les scores seront mis à jour en fonction des actions des joueurs (10 points par but, 60 points et fin du match si un attrapeur capture le *Vif d'or*). La force de chaque joueur diminuera également à chaque tour, puisque leur fatigue augmente.

Le match se termine si le *Vif d'or* est attrapé, ou après 20 tours. Tout au long du match, les actions de chaque joueur et le score de chaque équipe doivent être affichés. Une fois le match terminé, l'équipe déclarée gagnante sera celle dont le score final sera le plus élevé. En cas d'égalité, le match sera proclamé nul.

Partie 4 : Gestion de la météorologie

La fonction `generer_meteo` permet de définir de manière aléatoire la météorologie lors de ce match de *Quadball*.

```
def generer_meteo():
    conditions = ["ensoleillé", "nuageux", "pluvieux", "venteux",
                 "brumeux"]
    return random.choice(conditions)
```

En s'inspirant de la structure de la fonction `simuler_match` existante, créer la nouvelle fonction `simuler_match_meteo`. Celle-ci doit intégrer la gestion dynamique de la météo tout en conservant la logique de base du déroulement

d'un match de *Quadball* . Les changements météorologiques doivent influencer de manière significative et réaliste le cours du jeu.

Partie 5 : Gestion des blessures

La fonction `generer_blessure` permet de définir de manière aléatoire des blessures à un joueur lors de ce match de *Quadball* .

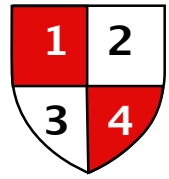
```
def generer_blessure():  
    types_blessures = ["légère", "moyenne", "grave"]  
    if random.random() < 0.05: # 5% de chance de blessure  
        return random.choice(types_blessures)  
    return None
```

En s'inspirant de la structure de la fonction `simuler_match` existante, créer la nouvelle fonction `simuler_match_blessure`. Celle-ci doit intégrer la gestion dynamique des blessures tout en conservant la logique de base du déroulement d'un match de *Quadball* . La gestion des blessures doit influencer de manière significative et réaliste le cours du jeu en changeant les caractéristiques du joueur.

2^e tâche : le blason de l'équipe

Chaque équipe de *Quadball* possède son propre blason.

L'une des équipes souhaite en dessiner un ayant la forme suivante :

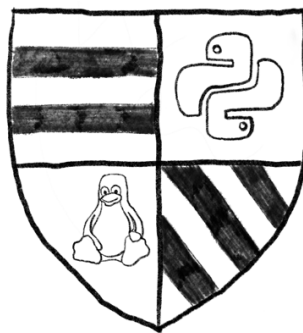


La zone numéro 2 doit contenir une représentation du logo python (fourni en annexe). Les deux serpents stylisés du logo devront être de deux couleurs différentes.

La zone numéro 3 doit contenir une représentation du logo Linux (fourni en annexe).

Les fonds des zones 2 et 3 doivent être unis, éventuellement de couleurs différentes.

Les fonds des zones 1 et 4 doivent être munis de rayures bicolores régulières, droites ou obliques.



*Document 2 - Croquis à main levée en noir et blanc d'un blason possible.
La disposition des rayures est indicative.*

Dans le tableau suivant se trouvent les noms et les composantes RVB des couleurs les plus couramment employées dans la création de blason, et qui sont à utiliser pour cet exercice :

| Couleurs | R | V | B | |
|----------|-----|-----|-----|--|
| or | 255 | 255 | 0 | |
| argent | 192 | 192 | 192 | |
| azur | 0 | 128 | 255 | |
| gueules | 226 | 13 | 13 | |
| sable | 0 | 0 | 0 | |
| orangé | 255 | 134 | 13 | |
| sinople | 20 | 148 | 20 | |
| pourpre | 120 | 3 | 115 | |

L'objectif de cette tâche est de compléter le fichier `Programme_blason.py` qui, à partir des images `blason.png`, `logo_python.png` et `logo_linux.png`, crée une image en couleur d'un blason répondant aux consignes données. Cette image sera nommée `Notre_blason.png`.

Rappels sur les pixels et la bibliothèque PIL

On rappelle que la couleur d'un pixel est représentée par trois entiers variant entre 0 et 255, qui constituent la quantité de rouge, de vert et de bleu. Une image est en niveaux de gris lorsque n'importe lequel de ses pixels a la même quantité de rouge que de vert et de bleu. On rappelle également que la couleur noire correspond aux valeurs (0, 0, 0).

Pour la réalisation de votre code, vous allez utiliser la bibliothèque PIL, dont voici un usage possible (le fichier Python correspondant devant se trouver dans le même dossier que l'image `image1.png`) :

```
from PIL import Image
img1 = Image.open("image1.png")

img2 = Image.new("RGB", img1.size)

largeur, hauteur = img1.size

img2.save("image_modifiee.png")
```

La variable `img2` représente une image, pour l'instant composée uniquement de pixels noirs, aux mêmes dimensions que l'image fournie.

Votre code utilisera `getpixel` et `putpixel` de la bibliothèque PIL.

Par exemple, le code :

```
(rouge, vert, bleu) = img1.getpixel((0,0))

img2.putpixel((0,0), (rouge, vert, bleu))
```

permet de récupérer les quantités de rouge, vert, bleu du pixel de coordonnées (0,0) de la première image et de colorer le pixel correspondant de la variable `img2`.

Enfin, l'instruction `img2.show()` vous permet à tout moment d'afficher l'image correspondante à la variable `img2`.