Étude de cas informatique

Préliminaires

Cette épreuve a pour objet l'étude du cas des compétitions d'ultimate. Chaque partie comprend la définition d'un certain nombre de problématiques à résoudre et présente des objectifs concrets, ainsi que la réflexion sur les moyens de les atteindre. Il est conseillé de bien lire l'ensemble du sujet avant de commencer à répondre à l'une des parties.

Attendus. Il est attendu des candidates et des candidats des réponses construites. Ils seront aussi évalués sur la précision, le soin et la clarté de la rédaction. Notamment, tout code doit être lisible, intelligible et documenté.

Dépendances. Ce sujet contient cinq parties. Les différentes parties et un grand nombre de leurs questions sont largement indépendantes. Il est possible d'aborder les différentes parties dans l'ordre qui vous conviendra le mieux mais en indiquant clairement à quelle question il est répondu.

Description du système étudié

L'ultimate. L'ultimate (ou *flying disc* en anglais) est un sport collectif inscrit au mouvement olympique. L'origine de ce sport est attribuée à des étudiants de l'Université de Yale, en 1940. Le jeu consiste à se transmettre un disque volant (aussi connu sous le nom de *frisbee*) [1] entre les joueurs d'une même équipe pour progresser sur le terrain de jeu. Lorsqu'un joueur réceptionne le disque, il ne peut plus se déplacer, mais peut établir un pied pivot (comme au handball, par exemple). Il a alors 10 secondes pour effectuer une passe à un(e) membre de son équipe.

Le but du jeu est donc d'amener le disque dans la zone de but adverse par une succession de passes entre les joueurs d'une même équipe. Si une passe est échouée, atterrit hors des limites du terrain ou est interceptée par l'équipe adverse, alors cette dernière récupère la possession du disque (l'attaque) et doit à son tour progresser par passes successives pour rejoindre son extrémité de terrain. Un point est marqué quand un joueur réceptionne le disque transmis par un membre de son équipe dans la zone d'en-but. Après chaque point marqué, les équipes se tiennent sur leur ligne d'en-but. L'équipe qui a marqué le dernier point lance le disque. L'autre équipe prend possession du disque là où il atterrit et devient alors l'équipe attaquante.

Les matchs. Un match d'ultimate se remporte, soit en 15 points, soit au temps (100 min). Dans le second cas, à la fin du temps réglementaire, le score à atteindre est alors le score le

1. « Frisbee » est une marque déposée de l'entreprise Wham-O-Holding basée à Hong Kong.

Tournez la page S.V.P.

plus haut +1 point—ce qu'on appelle le « cap à 1 », dans la limite des 15 points. Par exemple, si le score est de 13 – 10 à la fin du temps réglementaire, alors la première équipe à atteindre 14 points (13+1) gagne le match. L'ultimate se pratique dans sa version la plus courante sur terrain en herbe à l'extérieur (7 contre 7), mais peut aussi se pratiquer en intérieur (5 contre 5), ou sur la plage (5 contre 5).

L'ultimate se démarque des autres sports collectifs par le fait qu'il s'agit d'un sport auto-arbitré, quel que soit le niveau auquel il est pratiqué. Chaque joueuse/-eur ou équipe peut donc appeler une faute, ce qui suspend le jeu (mais pas le décompte du temps) et engage une phase d'échange entre les joueurs pour décider si le jeu doit se poursuivre, ou si les joueurs doivent revenir à la situation de jeu précédant la faute. Par exemple, les contacts étant interdits, un joueur peut appeler une faute s'il s'estime gêné dans ses déplacements ou empêché d'effectuer une passe.

Les compétitions. En compétition, on retrouve plusieurs catégories : *open*, femmes, mixte (4 hommes/3 femmes ou 4 femmes/3 hommes). Il n'existe donc pas de catégorie réservée exclusivement aux hommes, la catégorie *open* étant aussi ouverte aux femmes. Les compétitions d'ultimate peuvent prendre plusieurs formes.

La formule championnat regroupe un ensemble d'équipes qui doivent toutes se rencontrer. Pour chaque journée du championnat, une équipe rencontre une et une seule autre équipe adverse. Un championnat peut être organisé en 2 phases : une phase aller et une retour. Le nombre de phases est décidé en amont d'un championnat et détermine le nombre de journées à jouer en fonction du nombre d'équipes inscrites.

La formule tournoi consiste en un mini-championnat qui est organisé sur un jour—ou un week-end—entre un nombre restreint d'équipes qui se rencontrent en une seule phase. En fonction du nombre d'équipes inscrites à un tournoi, il est possible de répartir ces dernières de manière équilibrée (plus ou moins une équipe selon le nombre d'équipes inscrites au tournoi) dans un ensemble de poules afin de limiter le nombre total de matchs joués.

La formule division consiste quant à elle en une succession de tournois. Les équipes concurrentes peuvent alors être réparties sur un ensemble de divisions. Par exemple, 30 équipes peuvent être réparties en 6 divisions de 5 équipes. À chaque journée, le premier de chaque division est promu dans la division supérieure (s'il en existe une), tandis que le dernier du groupe descend dans la division inférieure (si elle existe). Le nombre de journées est déterminé à l'avance.

Les classements. À la fin de toute compétition, il est possible d'établir un classement qui correspond à un ordre total des équipes engagées. Les règles de classement d'une compétition sont déterminées avant que celle-ci ne débute. Ces règles déterminent notamment le nombre de points attribués par victoire, ainsi que la gestion des situations d'égalité entre plusieurs équipes (e.g., en favorisant l'équipe qui a marqué le plus de points).

Toutes les formules peuvent éventuellement se conclure par une session de *playoff*. Durant cette session, les 2^N meilleures équipes de la phase préliminaire sont alors convoquées pour se

rencontrer dans une série de matchs à éliminations directes (e.g., quart de finale, demi-finale, finale pour une session de *playoff* à 8 équipes). La sélection des couples d'équipes pour les $N/2$ premiers matchs d'une session *playoff* peuvent être déterminés aléatoirement entre les équipes sélectionnées, ou en tenant compte du classement (e.g., la 1^e équipe contre la 8^e, la 2^e contre la 7^e, etc.). L'ordre dans lequel les couples sont établis fixe alors la suite des matchs jusqu'à la finale. L'équipe qui gagne alors la finale est alors proclamée gagnante de la compétition.

En dehors des compétitions officielles, il existe également des tournois ouverts, appelés *hat*. Les joueuses/-eurs s'y inscrivent de manière individuelle et la composition des équipes est tirée au sort, en respectant des conditions de parités et en essayant d'équilibrer les équipes. Outre, le genre, les joueuses/-eurs indiquent leur niveau de jeu et leur poste de prédilection (passeur, receveur, sans préférence) afin de constituer des équipes d'un niveau équivalent. Une fois les équipes constituées, ces tournois ouverts suivent le même règlement qu'une compétition de formule tournoi.

Dans la suite de ce sujet, nous nous intéressons à la modélisation d'un système information en ligne, commme une application web par exemple, qui permettrait de gérer des compétitions organisées par les fédérations d'ultimate, depuis l'inscription des joueurs à une compétition, jusqu'à la gestion des résultats.

Partie I. Application mobile

Dans un premier temps, nous souhaitons concevoir une application web permettant *i)* aux spectateurs de suivre l'évolution du score d'un match depuis leur téléphone, et *ii)* aux joueurs de mettre à jour ce score en direct. Nous proposons notamment de nous intéresser à l'interface graphique permettant de consulter le score des matchs d'une compétition.

Question 1. Le domaine sur lequel est déployé l'application est `https://app.ultimate.fr/`. Expliquez à quoi fait référence le caractère `s` du protocole `https://`. Quel est le port réseau standard associé à ce protocole ?

Question 2. Quel est l'intérêt d'avoir recours au protocole HTTPS pour une application web ?

Question 3. Sur quelles couches du modèle OSI (*Open Systems Interconnection*) repose le protocole HTTPS ?

Question 4. Écrire une requête HTTP minimale permettant d'afficher la page principale de l'application web étudiée, et expliquer le rôle de chacun de ses champs.

Question 5. Chaque compétition doit pouvoir disposer de sa propre adresse pour consulter facilement les scores et autres informations relatives à son organisation (composition des équipes, planning des matchs, classement temporaire, etc.). Proposez une structuration d'URL pour l'application web qui permette de consulter le score d'un match `m` organisé dans le cadre de la compétition `c`.

Tournez la page S.V.P.

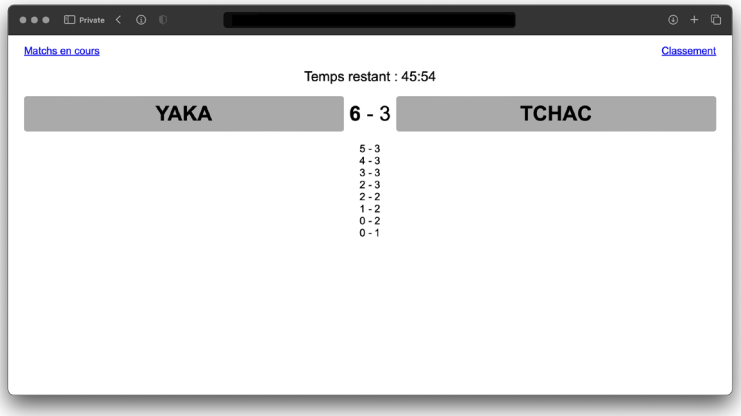

FIGURE 1 – IHM de la page de suivi d'un match accessible pour un spectateur. Le contenu de la page est centré et les polices sont volontairement agrandies pour faciliter la lecture des informations importantes. De haut en bas, les liens hypertextes permettent d'accéder à la liste des matchs en cours et le(s) classement(s) de la compétition. Le temps restant indique le nombre de minutes et secondes à jouer avant la fin du temps officiel. En cas d'application du "cap à 1", le temps restant compte à partir de zéro jusqu'à ce que le vainqueur du match soit désigné. L'historique des scores, indiqué dans la partie inférieure de l'écran, liste la séquence des précédents scores du match.

Question 6. Est-il possible de faire en sorte que la même application web côté serveur puisse servir des requêtes en provenance de clients de types «navigateur web» ou «application mobile dédiée» ? Si non, pourquoi ? Si oui, comment peut-elle le faire ?

Question 7. Le code compilé d'une telle application mobile de suivi des matchs peut-il s'exécuter sur n'importe quel équipement ? Pourquoi ? Est-ce également le cas du code de l'application web ? Pourquoi ?

Question 8. Du point de vue du réseau, discutez les avantages et inconvénients d'utiliser une application mobile dédiée ou une application web pour fournir un service de suivi de scores.

Partie II. Gestion d'un match d'ultimate

Nous proposons ensuite de nous intéresser plus spécifiquement à l'interface humain machine (IHM) permettant de suivre en direct le score des matchs d'une compétition. Dans cette partie, nous considérons plus précisément le cas d'une interface HTML qui est accessible depuis n'importe quel navigateur web comme, par exemple, celle illustrée dans la figure 1.

Question 9. Donnez le code HTML/CSS statique qui correspond à l'esquisse d'IHM décrite dans la figure 1. Les informatiques relatives au nom des équipes, des scores et de l'historique peuvent être indiqués dans le texte de la réponse.

Question 10. Enrichissez le code HTML/CSS précédent pour proposer une solution supportant la mise-à-jour dynamique, i.e. sans rechargement de la page, de l'affichage du score d'un match en cours, ainsi que l'historique de l'évolution du score pour ce match.

Question 11. Donnez le code JavaScript qui permet à l'application cliente de recevoir les informations relatives à l'évolution du score du match consulté. Le code proposé doit permettre d'être informé qu'un point vient d'être marqué par l'une ou l'autre des deux équipes sur le terrain. Le nombre de points de l'équipe qui vient de marquer doit s'afficher en gras.

Dans un second temps, nous proposons d'enrichir cette page HTML pour ajouter des boutons de contrôle du score qui ne soient accessibles qu'aux joueurs. Ces boutons correspondent aux parties grisées dans la figure 1. En cliquant sur le nom d'une équipe des deux équipes, un joueur autorisé incrémente alors immédiatement le score de cette équipe.

Question 12. Pourquoi l'utilisation de HTTPS ne suffit pas à limiter l'accès aux boutons de contrôle du score ? Décrivez précisément une solution, sans en fournir le code, pour restreindre l'accès à cette page à un ensemble de joueurs autorisés.

Question 13. Donnez le code JavaScript de la fonction `marquerPoint(equipe)`, invoquée en cliquant sur le bouton associé à l'équipe `equipe` pour incrémenter le score d'une unité en sa faveur.

Question 14. Expliquez précisément comment il est possible pour l'application web de s'assurer qu'un point qui vient d'être marqué n'est pas comptabilisé plusieurs fois par plusieurs joueurs qui seraient connectés simultanément sur cette page.

Question 15. Proposez une solution côté client, en JavaScript, pour que des accès concurrents des joueurs ne conduisent pas à incrémenter plusieurs fois le score d'une même équipe pour enregistrer un point qu'elle viendrait de marquer.

Question 16. Est-il également nécessaire de modifier le code de l'application côté serveur pour éviter cette situation ? Pourquoi ?

On considère la définition Python d'un match d'ultimate entre 2 équipes sous la forme de classes Python permettant de consulter et manipuler le score comme suit :

```
class Equipe:
    def __init__(self,name):
        self._name = name

    def __str__(self):
        return self._name

    def __repr__(self):
        return f'Equipe({self._name})'
```


Tournez la page S.V.P.

```
class Match:
    def __init__(self, locaux, visiteurs):
        if locaux == visiteurs:
            raise Exception("Les équipes doivent être distinctes.")
        self._score = {locaux:0, visiteurs:0}
        self._dernier = None

    def equipes(self):
        return self._score.keys()

    def score(self):
        return self._score.copy()

    def gagnant(self):
        return max(self._score, key=self._score.get)

    def perdant(self):
        return min(self._score, key=self._score.get)

    def marquerPoint(self, equipe):
        self._score[equipe] += 1
        self._dernier = equipe

    def annulerPoint(self):
        if self._dernier is None:
            raise Exception("Aucun point à annuler.")
        self._score[self._dernier] -= 1
        self._dernier = None

    def __str__(self):
        return self._score
```

À chaque match est donc associé 2 équipes distinctes (les locaux et les visiteurs) et un score initial fixé à 0 - 0.

Question 17. Dans la suite du sujet, nous souhaitons utiliser la classe `Equipe` dans différentes structures de données, comme des dictionnaires, en nous assurant que le nom de l'équipe est bien unique, i.e. on s'attend à ce que l'expression `Equipe("TCHAC") == Equipe("Tchac")` soit évaluée à vrai. Complétez le code de la classe Python `Equipe` décrite ci-dessus afin de pouvoir l'utiliser comme clé d'identification d'une équipe au sein d'une compétition.

Question 18. Donnez le code Python d'une classe `GestionMatch` qui permet de gérer un match d'ultimate. La solution proposée doit notamment permettre d'implémenter la règle du "cap à 1" en tenant compte de tous les états dans lesquels un match peut être (*planifié*, *en cours*, *cap*, *terminé*). Le code attendu doit expliciter ces états et l'ensemble des méthodes et attributs requis pour suivre l'évolution du score d'un match.

Pour information, les scores sont mémorisés dans une base de données SQL via une table `HistoriqueScores` dont la définition est donnée comme suit :

```
CREATE TABLE Equipes (
    EquipeID INT PRIMARY KEY AUTO_INCREMENT,
    Nom VARCHAR(255) NOT NULL UNIQUE
);

CREATE TABLE Matchs (
    MatchID INT PRIMARY KEY AUTO_INCREMENT,
    EquipeAID INT,
    EquipeBID INT,
    FOREIGN KEY (EquipeAID) REFERENCES Equipes(EquipeID)
    FOREIGN KEY (EquipeBID) REFERENCES Equipes(EquipeID)
);

CREATE TABLE HistoriqueScores (
    HistoriqueID INT PRIMARY KEY AUTO_INCREMENT,
    MatchID INT,
    ScoreEquipeA INT,
    ScoreEquipeB INT,
    Estampille TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (MatchID) REFERENCES Matchs(MatchID)
);
```

Question 19. Donnez la requête SQL permettant de récupérer le score courant du match qui oppose les équipes YAKA et TCHAC.

Question 20. Proposez une solution pour annuler le dernier point marqué si une équipe appelle une faute qui remet en cause ce point. On considère ici qu'il n'est pas utile de conserver une trace de la faute qui a conduit à l'annulation du point. Si le code client nécessite d'être modifié, décrivez les changements à apporter au code HTML, CSS, et JavaScript pour mettre en place cette fonctionnalité du côté client. Si le code serveur nécessite d'être modifié, décrivez les changements à intégrer au code Python et à la base de données.

Partie III. Gestion de tournois d'ultimate

Nous proposons désormais d'étudier l'organisation de tournois d'ultimate qui se dérouleraient en deux temps : une étape préliminaire de poules, suivie d'une étape finale de *playoff* permettant de déterminer l'équipe gagnante d'un tournoi. Le nombre de poules d'un tournoi est déterminé en fonction du nombre de créneaux horaires disponibles, du nombre de terrains et du nombre de phases (par défaut, une seule phase est organisée). Lors de l'étape préliminaire, par souci d'équité, il est souhaitable qu'une équipe ne joue pas deux matchs dans deux plages consécutives d'une même journée, leurs laissant ainsi un temps suffisant de récupération.

Question 21. En partant du principe qu'un tournoi dispose de `t` terrains et `c` créneaux horaires, donnez le code Python de l'algorithme `liste_combinaisons(t,c)` qui liste les différentes combinaisons de nombres d'équipes (`n`) et de poules (`p`) acceptables en cherchant à maximiser le nombre d'équipes qu'il est possible d'inscrire à un tournoi en fonction du nombre de poules retenues par les organisateurs. En sachant qu'il est inutile qu'une équipe ne s'inscrive pour jouer moins de 3 matchs, ignorez toute combinaison qui ne respecte pas cette contrainte. Par exemple,

Tournez la page S.V.P.

l'appel à la fonction `liste_combinaisons(4, 10)` pour calculer les combinaisons possibles avec 4 terrains et 10 créneaux devra retourner `{3: 12, 2: 10, 1: 6}`—i.e., un maximum de 12 équipes peuvent s'inscrire si les organisateurs privilégient 3 poules, 10 équipes pour 2 poules ou 6 équipes pour une poule unique.

Question 22. Donnez le code Python d'une fonction `liste_matchs(n,p)` qui génère la liste des matchs à jouer en fonction de la liste des `n` équipes inscrites et le nombre de poules souhaitées `p`. Les équipes sont réparties aléatoirement et de manière équilibrée dans les poules (plus ou moins une équipe selon le nombre total d'équipes). Le code produit doit obligatoirement faire apparaître la notion de `Poule` d'un tournoi sous la forme d'une classe dont le constructeur prend en paramètre la liste des équipes inscrites dans une poule donnée. La liste des matchs est générée à la construction de la poule. La classe `Poule` doit également fournir le code d'une méthode `classement()` permettant de retourner le classement courant d'une poule sous la forme d'une liste ordonnée d'équipes. Les équipes sont classées en priorité par nombre de matchs gagnés, puis par différence entre le nombre total de points marqués et le nombre total de points encaissés.

On s'intéresse maintenant à la planification d'un tournoi, c'est-à-dire l'association d'un match à un créneau horaire et un terrain, exactement. Pour ce faire, on souhaite disposer d'une classe `Planning` qui est initialisée avec une liste de matchs, un nombre de terrains et un nombre de créneaux disponibles pour le tournoi. La liste de matchs est triée par terrain, puis par créneau—*i.e.*, les n premiers éléments de la liste correspondent aux matchs du 1^{er} créneau sur les n terrains disponibles, et ainsi de suite. La classe `Planning` permet ensuite de consulter un même planning par terrain ou par créneau. La méthode `par_terrain()` retourne notamment un ensemble de terrains qui contient une liste de matchs ordonnés par le créneau (un créneau non-occupé par un match est associé à la valeur `None`). De manière similaire, la méthode `par_creneau()` retourne une liste de créneaux qui contient un ensemble de terrains associés à un match (un terrain non-occupé par un match est associé à la valeur `None`).

Question 23. Donnez le code Python de la classe `Planning` qui permet d'initialiser la structure de données servant à gérer un planning de tournoi à partir des paramètres d'initialisation. La classe doit fournir le code des fonctions `par_terrain()` et `par_creneau()` afin de consulter le planning sous différentes vues.

Question 24. Donnez le code Python de 4 tests unitaires pour la classe `Planning` permettant de s'assurer qu'un planning de tournoi généré respecte les contraintes d'organisation décrites dans le sujet.

Question 25. Donnez le code Python d'une fonction `creer_planning(matchs,t,c)` qui construit le planning d'un tournoi en tenant compte du nombre `t` de terrains disponibles et le nombre `c` de plages horaires. Un planning alloue à chaque match un créneau horaire et d'un terrain. N'hésitez pas à préciser quelle(s) optimisation(s) vous avez mise(s) en œuvre dans le code de la fonction `creer_planning(matchs,t,c)` pour améliorer le temps de calcul d'un planning correct.

Question 26. Donnez le code Python d'une fonction `creer_playoff(nb_tours, poules)` qui sélectionne les premières équipes parmi la liste de `poules` d'un tournoi pour générer une liste initiale de matchs de *playoff*, à élimination directe, qui se dérouleront en `nb_tours` tours jusqu'à la finale.

Question 27. Donnez le code Python d'une fonction `tour_suivant(matchs)` qui analyse la liste des `matchs` joués pour préparer le tour suivant de la phase de `playoff` sous la forme d'une nouvelle liste de matchs en faisant se rencontrer les gagnants du tour courant uniquement.

Question 28. Donnez le code Python d'une fonction `joue_playoff(tours, poules)` qui utilise les fonctions des 2 questions précédentes pour orchestrer le déroulement de l'étape de *playoff* et retourner le podium du tournoi sous la forme d'une liste ordonnée de 3 équipes, en sachant que la troisième place est attribuée au terme d'une petite finale qui oppose les équipes qui ont perdu leur demi-finale.

Partie IV. Gestion d'une compétition en divisions

On s'intéresse ici à la formule division qui offre une forme de compétition hybride, entre championnats et tournois. À la différence des poules d'un tournoi, la composition des divisions n'a rien d'aléatoire puisqu'elle s'appuie sur le classement des équipes à l'issue de la précédente saison (qui n'était pas nécessairement organisée sous une formule division) pour déterminer comment les équipes doivent être regroupées. À noter que toute nouvelle équipe doit intégrer la dernière division.

Question 29. Donnez le code Python d'une classe `CompetitionDivision` dont le constructeur prend pour paramètres *i)* un `classement` d'équipes (de la meilleure à la moins bonne), *ii)* la `taille` maximale des divisions en terme de nombre d'équipes et *iii)* le nombre de `journées` prévues pour cette compétition. Vous proposerez la structure de données adéquate pour gérer une division, telle que décrite dans l'énoncé. Outre la définition du constructeur, la classe doit également fournir une fonction `classement()` qui retourne le classement courant d'une compétition comme une liste ordonnée d'équipes.

Question 30. Donnez le code Python d'une méthode `prochaine_journee()` au sein de la classe `CompetitionDivision` qui, tant que le nombre de journées planifiées n'a pas été atteint, récupère le classement de chaque division, pour créer les divisions de la prochaine journée de compétition en respectant la règle décrite dans l'énoncé. La solution proposée doit conserver l'historique des précédentes journées et la fonction `classement_journee(journee)`, dont vous fournirez le code Python, doit notamment permettre de récupérer le classement de la journée `journee`. Il est attendu que cette méthode et cette fonction déclenchent une exception `ValueError` si elles sont invoquées dans des conditions incorrectes (que vous expliciterez).

Question 31. Donnez le code Python d'une fonction `jouer_division(anciennes,nouvelles,taille,j)` qui crée une compétition de type division, à partir de la `taille` maximale des divisions en terme de nombre d'équipes, des équipes ordonnées `anciennes` qui étaient classées la saison précédentes et des équipes ordonnées `nouvelles` qui sont inscrites à partir de cette saison, puis organise `j` journées de compétition. La fonction retourne le podium de la compétition sous la forme d'une liste de 3 équipes ordonnées.

Tournez la page S.V.P.

Question 32. Les notions de divisions et de poules sont-elles différentes ? Est-il possible de factoriser le code des classes associées ?

Partie V. Gestion d'un tournoi *hat*

Cette partie couvre enfin le cas particulier des tournois *hat* et de la gestion des joueuses/-eurs qui y participent en s'inscrivant individuellement. Pour s'inscrire à un tournoi *hat*, les joueuses/-eurs saisissent notamment leurs noms/prénoms, adresse, ville, niveau de jeu (évalué entre 1 et 10, 10 correspond à un niveau «excellent») et postes de prédilection (passeur, receveur, sans préférence) dans un formulaire en ligne, avant de s'acquitter des frais d'inscription. On considère que ces informations sont accessibles via une classe Python `Joueur`.

Question 33. Proposez une définition de la classe Python `Joueur` et une extension de la classe Python `Equipe`, nommée `EquipeHat`, pour faciliter le calcul d'indicateurs comme le niveau de jeu, le ratio des genres et la répartition des postes via des fonctions dont vous expliciterez la signature et le type de retour.

Question 34. Donnez le code de 4 tests unitaires qui permettent de s'assurer qu'une fonction Python `creer_equipes(joueurs,n)` constitue bien les `n` équipes d'un tournoi en équilibrant le niveau de jeu cumulé des joueurs, la répartition des postes de prédilection et les genres au sein des équipes.

Question 35. Donnez le code Python de la fonction `creer_equipes(joueurs,n)` qui retourne une liste de `n` instances de la classe `EquipeHat` qui se répartissent la liste des `joueurs` de manière équitable.

En amont de leur participation à des tournois *hat*, les participant(e)s peuvent communiquer pour organiser leurs trajets respectifs vers le lieu du tournoi. En particulier, il est généralement possible d'avoir recours à du co-voiturage pour transporter un maximum de joueuses/-eurs dans un minimum de véhicules. Pour ce faire, tout(e) joueuse/-eur inscrit(e) peut se déclarer *i)* en recherche d'un véhicule ou *ii)* disposant d'un véhicule, auquel cas elle/il indique le nombre de sièges disponibles. On considère qu'on dispose d'une matrice des plus courtes distances entre les `N+1` villes en lien avec le tournoi—i.e., la ville organisatrice et la liste des villes des joueuses/-eurs inscrits pour un co-voiturage. Les villes de départs sont celles où un(e) joueuse/-eur se déclare comme disposant d'un véhicule, tandis que la ville d'arrivée est celle où est organisée le tournoi.

On s'intéresse donc à établir un arbre couvrant de poids minimal dont la racine est la ville organisatrice et dont les feuilles sont les villes dans lesquelles des véhicules sont disponibles. On souhaite donc, à partir de la matrice de distance entre toutes les villes, du nombre de joueuses/-eurs à récupérer par ville, et de la liste des villes de départ, construire une liste ordonnée de villes à visiter pour chaque ville de départ.

Question 36. Quelle est la pré-condition à respecter pour pouvoir calculer un tel arbre couvrant de poids minimal ?

Par exemple, on considère un tournoi *hat*, organisé à Rennes, pour lequel les joueuses/-eurs inscrit(e)s habitent Paris, Lille, Lyon, Nantes, Bordeaux, Auxerre, Amiens.

Question 37. En partant du principe que les villes de départ sont Lille, Lyon, Nantes—avec des capacités respectives de 4, 3 et 6 places—et que les villes de Paris, Nantes, Bordeaux, Auxerre et Amiens recensent 2, 1, 3, 3 et 2 personnes à véhiculer, donnez la liste des co-voiturages à envisager pour permettre à tous les joueuses/-eurs de rejoindre le tournoi organisé à Rennes. Pour illustrer votre proposition, vous pouvez préciser le contenu de la matrice des distances avec des valeurs fictives qui vous serviront à justifier que votre solution minimise la distance totale parcourue.

Question 38. Expliquez pourquoi plusieurs co-voiturages sont possibles.

Pour construire la liste des co-voiturages possibles, on introduit la notion de véhicule dont la définition Python est donnée sous la forme d'une classe `Vehicule` :

```
class Vehicule:
    def __init__(self,conducteur,capacite,depart,distances):
        self._conducteur = conducteur
        self._capacite = capacite
        self._position = depart
        self._distances = distances
        self._distance = 0
        self.passagers = []

    def places_disponibles(self):
        return self._capacite - len(self.passagers)

    def distance_parcourue(self):
        return self._distance

    def ajouter_passager(self,passager):
        if self.places_disponibles() == 0:
            raise ValueError("Le véhicule est complet")
        self.passagers.append(passager)

    def visiter_ville(self,ville):
        if ville in self._distances:
            self._distance += self._distances[self._position][ville]
            self._position = ville
```

Question 39. Donnez le code Python de la fonction `lister_covoiturages(destination,distances,passagers,vehicules)` qui calcule une liste des co-voiturages à prévoir pour chaque conducteur déclaré. Le paramètre `vehicules` prend la forme d'un ensemble de tuples `(ville,listeVehicules)` qui indique, pour chaque ville de départ, la liste des véhicules disponibles. Le paramètre `passagers` prend la forme d'un ensemble de tuples `(ville,listeJoueurs)` qui indique, pour chaque ville à visiter, la liste des joueurs à récupérer. La fonction retourne une liste de `Vehicule`, mis à jour avec les joueurs qui doivent être récupérés, auxquels sont associés la liste ordonnée des villes à traverser. La fonction doit s'assurer que la capacité de chaque véhicule est respectée et que la distance parcourue par l'ensemble des véhicules est minimale.

Tournez la page S.V.P.

Question 40. Donnez le diagramme de classes, avec la liste des attributs et la signature des méthodes (sans le code), qui permet de gérer les différentes formules de compétitions couvertes par le sujet (championnat, tournoi, division, *playoff* et *hat*), tout en faisant apparaître les autres concepts abordés dans le sujets. Vous prendrez soin d'indiquer les classes et méthodes abstraites, ainsi que les différentes relations entre les concepts représentés.

Question 41. Après avoir rappelé ce que sont les principes SOLID, vous illustrerez quel(s) principe(s) peuvent s'appliquer à la modélisation des compétitions que vous avez proposé.

Question 42. Dans quelle mesure la modélisation de ce système logiciel est dédié à la pratique de l'ultimate ? Précisez quelles sont les classes qui sont spécifiques à la gestion des règles de ce sport et quel mécanisme de conception pourrait être appliqué pour permettre de généraliser cette modélisation à la gestion de compétitions pour d'autres sports collectifs.