Principes et pratique de la programmation orientée objet

- Final -

Consignes :

- Les documents sont interdits ;
- Attention, l'énoncé est recto/verso ;
- Relisez-vous et faites attention à la grammaire et l'orthographe ;
- Attention au soin : votre copie n'est pas un brouillon. Aucun effort de lecture ne sera fait lors de la correction. Une réponse jugée illisible ne sera pas prise en compte.

I. Modélisation UML (environ 10 points)

Vous êtes en charge de la conception d'un outil d'aide à la décision pour coach sportif. Cet outil a pour objectif de permettre à un entraineur de club de football de simuler des matchs de football sous différentes conditions (choix de joueurs, place, stratégie adverse, …) pour maximiser les résultats de l'équipe qu'il supervise. Cet outil est utilisé par l'entraineur qui définit des stratégies de jeu pour un match et par le directeur sportif qui gère les joueurs et le budget de l'équipe.

Les grandes fonctionnalités de cet outil sont donc regroupées autour de :

- La gestion des joueurs (ajouter/retirer un joueur dans l'équipe, associer un coût pour le salaire de chaque joueur) ;
- La définition de configurations à simuler pour les prochains matchs qui consiste à (1) sélectionner un ensemble de joueurs de l'équipe, (2) leur attribuer un poste durant le match (attaquant, milieu de terrain, défenseur), (3) définir des stratégies de jeu (c'est à dire une stratégie d'attaque, et une stratégie de défense qui peuvent varier au cours du match en fonction du score) et (4) gérer la fatigue des joueurs en définissant les instants de remplacement des joueurs sur le terrain avec ceux sur le banc.
- La simulation d'un match permet de confronter la configuration définie à différents modèles d'équipes virtuelles, qui suivent une configuration définie par le logiciel.
- L'analyse des résultats permet d'obtenir toutes les statistiques de matchs de la configuration définie face aux configurations virtuelles, à savoir : les scores, les joueurs qui ont marqués, les joueurs qui ont reçu une pénalité, la fatigue des joueurs, le coût salarial associé à la sélection de joueurs. Sur la base de ces informations, l'entraineur et le directeur sportif établissent une sélection finale.

Chaque joueur est caractérisé par son nom, son prénom son âge, sa taille et son poids et un ensemble de caractéristiques, représentées sous forme d'un réel compris entre 0 et 1, qui vont permettre de modéliser la performance de ses actions. Celles-ci sont : sa précision, son agilité, son agressivité et son état de fatigue. Les actions qu'il peut faire sont : se déplacer, faire une passe, dribler un joueur adverse, contrer un joueur adverse et tirer au but. Chaque action menée peut être réussie ou non en fonction des caractéristiques du joueur et du joueur adverse selon le cas. Le gardien est un joueur spécial qui peut, en plus des actions précédentes, attraper le ballon à la main. Enfin, l'équipe compte un capitaine qui est un joueur décoré de cette fonction qui voit toutes ses actions être moins sujettes à l'agressivité (car le capitaine doit toujours garder son calme). En outre, c'est le seul joueur qui est autorisé à discuter avec les arbitres en cas de litige et il est doté de cette action supplémentaire.

Une équipe regroupe un ensemble de joueurs. Par contre une sélection pour un match est un sous-ensemble de 22 joueurs sélectionnés à qui on associe un numéro et un poste. Un match possède un algorithme qui produit un ensemble d'évènements horodatés (but, pénalité, blessure, changement de joueur) fonctions des actions des joueurs et des stratégies mises en place par l'entraineur. Chaque événement est associé au(x) joueur(s) qui l'ont produit.

I.1. Proposer un diagramme des cas d'utilisation de ce simulateur.

I.2. Proposer une solution de modélisation pour l'intégration dans le diagramme de classes des stratégies de défense et d'attaque. Justifier ce choix.

I.3. Proposer une solution de modélisation pour l'intégration dans le diagramme de classes de la notification à l'entraineur de l'ensemble des évènements qui se produisent durant le match afin qu'il puisse modifier sa configuration de jeu durant son déroulement. Justifier ce choix.

I.4. Proposer une solution de modélisation pour l'intégration dans le diagramme de classes du rôle de capitaine attribué à un joueur. Justifier ce choix.

I.5. Proposer un diagramme de classes pour représenter chacun des composants du simulateur. Préciser tous les attributs et méthodes nécessaires ainsi que les relations entre les classes et les cardinalités associées. On intégrera dans ce diagramme les solutions proposées dans les questions I.2, I.3 et I.4.

II. Production de code Java (environ 6 points)

Cette seconde partie est indépendante de la partie précédente. On s'intéresse ici à l'implémentation d'une partie de l'outil d'aide à la décision pour coach sportif précédemment modélisé. Plus spécifiquement, dans le cadre de l'algorithme de simulation de déroulement de match, on s'intéresse au code capable de simuler les échanges du ballon entre les joueurs. Cette partie du simulateur s'articule ainsi autour d'une classe Joueur et d'une classe Ballon, cœur de la partie simulée, dont la déclaration partielle est donnée ci après.

```
public class Ballon {

    private Joueur possesseur; // le joueur qui a le ballon en sa possession
    private HashSet<Joueur> receveurs; // l'ensemble des joueurs prêts à recevoir une
passe
    // le joueur j qui souhaite faire une passe à un des receveurs
    public void passer(Joueur j) { ... }

    // un joueur j souhaite récupérer le ballon en tentant de dribbler son possesseur
    public void dribbler (Joueur j) { ... }

    // indique que j est prêt à recevoir une passe du possesseur
    public void pretARecevoir(Joueur j) { ... }

    // getter qui indique si j est le posesseur actuel du ballon
    public boolean possede(Joueur j) {
        return this.possesseur.equals(j);
    }

    // getter qui renvoie le possesseur actuel
    public Joueur possesseur(); {
      return this.possesseur;
    }
}
```

II.1 On modélise dans cette partie un joueur de façon simplifiée par son nom, son équipe, son agilité et sa fatigue. Chaque joueur possède une référence sur le ballon qui lui permet d'appeler ses méthodes lors du déroulement du jeu. Une méthode déclarée *boolean estDansEquipe(Joueur j)* permet de savoir si deux joueurs appartiennent à la même équipe. Une méthode déclarée *double dribble()* permet de déterminer une performance de dribble en calculant *agilite – fatigue++ / agilite*. Enfin, une méthode *String toString()* permet de retourner le nom, l'équipe et la performance de dribble du joueur. Donner le code de la classe Joueur qui permet d'implémenter ces spécifications.

II.2. On s'intéresse maintenant au cœur de la classe Joueur. Un joueur est un thread qui exécute en boucle (tant que le joueur n'est pas sur le banc), l'algorithme suivant : si le joueur possède le ballon, il fait la passe à un autre joueur. Sinon, si le ballon est possédé par un joueur de son équipe, il s'indique prêt à recevoir le ballon. Enfin, si le ballon est possédé par un joueur de l'autre équipe, il le dribble pour tenter de récupérer le

ballon. Etendre la classe Joueur (il est inutile de recopier le code de la question précédente) pour prendre en compte ces fonctionnalités.

II.3. On s'intéresse maintenant à la classe Ballon. Etant donné qu'un match se joue avec un unique ballon, donner le code qui permet d'assurer la présence d'une unique instance de cette classe dans le simulateur.

II.4. Justifier l'emploi d'une collection de type *HashSet* pour référencer l'ensemble des receveurs potentiels d'une passe.

Etant donné le code de la classe Joueur donné dans la question II.2, on comprend que le ballon s'apparente à un objet dont l'accès aux méthodes se fait de façon concurrente de la part des joueurs qui vont tous tenter au sein du flot d'instructions de leur thread de faire une passe (pour son possesseur), dribbler le possesseur ou indiquer être prêt à recevoir une passe. En utilisant les outils d'exclusion mutuelle et de synchronisation conditionnelle, il vous est proposé de donner le code des méthodes *passer*, *dribbler* et *pretARecevoir*.

II.5. La méthode *passer* consiste pour un joueur à attendre qu'au moins un joueur de son équipe soit référencé dans la collection de *receveurs*. Si à l'issue de cette attente, il est toujours en possession du ballon (sans avoir été dribblé), alors il passe le ballon au premier joueur présent dans la collection de receveurs. Donner le code de la méthode *passer*.

II.6. La méthode *pretARecevoir* indique que le joueur qui l'appelle est prêt à recevoir une passe. Il s'enregistre alors dans la collection de receveurs et réveille le possesseur du ballon en attente de receveurs. Donner le code de la méthode *pretARecevoir*.

II.7. La méthode *dribbler* permet à un joueur de l'équipe adverse de s'emparer du ballon auprès de son possesseur pendant qu'il attend qu'un joueur soit prêt à recevoir une passe. Pour ce faire, cette méthode compare la performance de la méthode *dribble* de chacun des joueurs qui s'affrontent et attribue à celui de valeur la plus élevée la possession du ballon. Elle réveille aussi le possesseur du ballon en attente de receveurs car il a pu en perdre la possession. Donner le code de la méthode *dribbler*.

Vous trouverez ci-après un exemple de trace d'exécution de ce cœur de simulateur qui comporte deux joueurs de l'équipe de France (Zidane et Platini) faces à deux joueurs de l'équipe du Brésil (Pelé et Ronaldo). A chaque début de ligne, on donne le nom du joueur (par ex. Zidane), son équipe (France) et sa performance au dribble (D:9.0) qui diminue au fil du match.

```
Zidane (France; D:9.0) a le ballon.
Zidane (France; D:9.0) veut faire la passe mais il attend car il n'y a pas de receveur.
Platini (France; D:7.0) est prêt à recevoir la passe.
Zidane (France; D:9.0) fait la passe à Platini (France; D:7.0)
Pelé (Brésil; D:8.0) veut dribbler Platini (France; D:7.0) : Ok!
Pelé (Brésil; D:7.9) a le ballon.
Ronaldo (Brésil; D:6.0) est prêt à recevoir la passe.
Platini (France; D:6.9) veut dribbler Pelé (Brésil; D:7.9) : Echoué!
Pelé (Brésil; D:7.8) a le ballon.
Zidane (France; D:8.9) veut dribbler Pelé (Brésil; D:7.8) : Ok!
Zidane (France; D:8.8) a le ballon.
Pelé (Brésil; D:7.6) veut dribbler Zidane (France; D:8.8) : Echoué!
Zidane (France; D:8.7) a le ballon.
Ronaldo (Brésil; D:6.0) veut dribbler Zidane (France; D:8.7) : Echoué!
Zidane (France; D:8.6) a le ballon.
Zidane (France; D:8.6) veut faire la passe mais il attend car il n'y a pas de receveur.
Platini (France; D:6.7) est prêt à recevoir la passe.
Pelé (Brésil; D:7.5) veut dribbler Zidane (France; D:8.6) : Echoué!
Zidane (France; D:8.4) a le ballon.
Zidane (France; D:8.4) fait la passe à Platini (France; D:6.7)
Ronaldo (Brésil; D:5.8) veut dribbler Platini (France; D:6.7) : Echoué!
Platini (France; D:6.6) a le ballon.
Platini (France; D:6.6) veut faire la passe mais il attend car il n'y a pas de
receveur.
Pelé (Brésil; D:7.4) veut dribbler Platini (France; D:6.6) : Ok!
Pelé (Brésil; D:7.3) a le ballon.
...
```

III. Questions de cours (environ 4 points)

III.1. Les quatre interfaces de collections sont Set, List, Map et Queue. Pour chacune d'entre elles, donner la propriété fonctionnelle qu'elle offre.

III.2. Les exceptions Java sont régies par le principe de « *Catch or Specify* ». Expliquer à quoi correspond chacun de ces deux termes et montrer la réalité de chacun d'eux dans le langage Java à travers leur traduction syntaxique.

III.3. Dans le cadre des interfaces graphique et en particulier du patron de conception MVC, donner le nom et expliquer le rôle de chacun des composants de ce modèle. Indiquer les relations qui existent entre chacun des composants et, lorsqu'il existe, donner le nom du patron de conception qui formalise cette relation.

III.4. Le patron de conception *Decorator* suit le modèle architectural abstrait rappelé ci-après. Dans le cadre des entrées/sorties en Java, et en particulier celles correspondant à la lecture de données en mode octet, indiquer le rôle de chacune des quatre classes de ce patron et illustrer chacune d'entre elles par un exemple de classe de la plateforme Java SE.