TP 3 : Les interactions entre les joueurs et les pokemons !

Aujourd'hui notre but sera de continuer de construire les interactions entre les pokemons et leurs joueurs. Les joueurs pourront collectionner de la nourriture, qu'ils pourront donner à leurs pokemons. De plus, aujourd'hui nous allons finalement permettre aux usagers d'interagir avec le programme.

Lien utile : https://pokemondb.net/pokedex/all .

1. Nous allons commencer par modifier la classe Pokemon. Chaque pokemon aura un score qui indiquera s'il a faim, s'il est content et s'il est loyal à son maître actuel. Plus tard, ces scores vont impacter la capacité du pokemon de lutter et de progresser.

    - Nous allons donc premièrement définir ces attributs, qui seront très simples : des scores entre 0 et 100 (ce sont donc quel type d'attributs ?). Les attributs s'appelleront : appetit, satisfaction, loyaute.
    - Réharmonisez vos constructeurs pour prendre en compte le fait qu'un pokemon est toujours créé avec : appetit = 50, satisfaction = 50, loyaute = 0.
    - Pour manipuler ces attributs vous aurez besoin de méthodes qui vous donnent les valeurs stockées par ces attributs (des méthodes getAppetit(), getLoyaute(), getSatisfaction()), ainsi que des méthodes qui mettent à jour la valeur d'un attribut : (par exemple baisseAppetit(int difference), monteAppetit(int difference), baisseLoyaute(int difference), monteLoyaute(int difference), baisseSatisfaction(int difference), monteSatisfaction(int difference)). Attention : il faut prendre en compte que les valeurs de l'appétit, de la satisfaction et de la loyauté d'un pokemon ne peuvent jamais sortir de l'interval 0-100 !
    - Mettez à jour votre méthode toString() qui devra désormais également afficher les trois nouveaux attributs (au-delà des valeurs qu'elle affiche déjà).

    - Règles de base :
        - *Capturer* un pokemon réduit sa loyauté à 0, sa satisfaction à 10 (si elle était plus large) et baisse son appetit à 10 (il refuse de manger).
        - *Libérer* un pokemon : s'il a une loyauté supérieure à 10 lorsqu'il est libéré, il perd un nombre de points de satisfaction égale à la différence entre sa loyauté actuelle et 10 (ou on la réduit à 0, selon besoin). (Par exemple : un pokemon qui a une loyauté de 36 et une satisfaction de 40 e libéré -- il perd 36-10 = 26 points de satisfaction. Si le pokemon n'a que 20 points de satisfaction lorsqu'il est libéré, alors sa satisfaction baisse à 0.). La loyauté du pokemon baisse à 0.

- La première fois qu'un nouveau maître *donne un nom* à un pokemon, sa loyaute monte par 10 points (ou elle est mise à 100 si le pokemon avait une loyaute supérieure à 90).
- Si le même maître *choisit après un autre nom* pour un pokemon qui a déjà un nom, sa loyauté baisse par 10 points (ou elle est mise à 0 si le pokemon avait une loyaute inférieure à 10).

2. Nos pokemons devront manger. Créez une classe de base, qui s'appellera Nourriture. Lors du prochain TP, nous allons utiliser l'héritage pour construire un nombre d'Ingrédients qui hériteront de la classe Nourriture. Mais, pour le propos de ce TP, nous allons utiliser le(s) constructeur(s) de la classe Nourriture pour définir des divers types d'ingrédients, qui pourront nourrir les pokemons.
   - Définissez une nouvelle classe Nourriture avec les attributs suivants :
     - Une variable appelée apport, qui stockera à quel point la nourriture soulage l'appétit d'un pokemon (quel type de variable sera celle-là, alors ?)
     - Une variable de type String qui stockera le nom de l'ingrédient.
     - Une variable appelée compatibilites, qui sera de type String[] : elle stockera les types de pokemon (plante, eau, etc.) qui aiment bien manger cette nourriture. Quelle taille aura ce String[] ?
     - Une variable entière appelée frequence, qui indiquera la rareté de ce type de nourriture (pour ce TP, par contre, la fréquence de chaque ingrédient sera uniforme à 30).

   - Dans ce TP, nous utiliserons un constructeur réduit pour cette classe, puisque la fréquence restera la même pour n'importe quel ingrédient. Ce constructeur aura la signature :

     Nourriture(int, String, String[])

   - Écrivez une méthode toString() qui affiche les attributs de chaque objet de la classe Nourriture.

   - Il va falloir qu'on puisse savoir si un certain type de pokemon (plante, eau) est parmi les compatibilités d'une certaine nourriture. Nous écrirons une méthode qui s'appellera estCompatible. Quel sera le type de la sortie de cette méthode ? Quel(les) seront les entrées ?

3. Testez le fonctionnement de vos nouvelles méthodes, classe, et attributs dans votre méthode main :
   - Créez un type de nourriture qui s'appelle "tartiflette", avec un apport de 35. Ce type de nourriture sera compatible avec les prochains types de pokemons : Dragon, Feu, Combat,

Normal, Eau, Electrique. Créez des pokemons qui sont compatibles avec la tartiflette et des pokemons qui ne le sont pas (par exemple un pokemon de type Eau et un de type Plante). Qu'est-ce qui se passe si on teste la compatibilité de la tartiflette avec les deux pokemons ?

- Un deuxième type de nourriture s'appelle "ratatouille", avec un apport de 10. Il est compatible avec les prochains types de pokemons : Plante, Eau, Vol, Feu, Normal, Electrique, Combat. Testez encore une fois la compatibilité de votre type de nourriture avec deux pokemons que vous avez crées.

4. Revenons à notre classe Nourriture. Nous allons créer un générateur aléatoire de nourriture, en utilisant l'attribut frequence. Une fréquence de 30 veut dire que 30 fois sur 100, un objet du type nourriture sera généré. Pour cela, nous pouvons utiliser la méthode random() de la classe Math. Regardez la documentation de cette méthode sur : https://docs.oracle.com/javase/7/docs/api/java/lang/Math.html .

   Pour utiliser la classe Math, alors on va devoir mettre en préambule la ligne de code :

   import java.lang.Math;

   Math.random() donne en sortie une valeur du type double aléatoirement choisie entre 0.0 et 1.0. Ce que nous voudrions bien, c'est d'avoir une valeur entre 0 et 100. Puis, si la valeur est entre 0 et 30, alors, on génère un nouvel objet du type nourriture. Sinon, on génère un objet vide. Premièrement nous allons essayer avoir une sortie de Math.random() entre 0 et 100. Comment allons-nous faire ça ? Puis, nous allons vouloir arrondir la valeur aléatoirement choisie entre 0 et 100, en utilisant la méthode Math.round(). Quel type de variable sera la valeur arrondie ?

   Dans la classe Nourriture, écrivez une méthode avec la signature :

   Nourriture genAlea()

   Cette méthode va générer de façon aléatoire, soit un objet du type Nourriture (avec les paramètres this.apport, this.nom, this.compatibilites) – si la valeur aléatoire arrondie est inférieure à la valeur de frequence (=30), soit null.

5. Testez la méthode genAlea() dans votre méthode principale : faites exécuter cet algorithme 10 fois pour vos deux types de nourriture et regardez le résultat. Est-ce que vous avez réussi générer au moins un type de nourriture ?