TP1: Système de combat RPG (IFT1025B - H24)

Ce projet est à faire seul ou en équipe de deux. Aucune équipe de 3 ne sera tolérée pour ce projet. Si lors de la remise il y a trois noms inscrits et que nous n'y étions pas au courant, vous serez pénalisés sur votre note finale.

Mise en situation

Vous développez un système de combat simple de jeu de rôle (RPG) textuel. Vous créez votre héros, qui peut avoir 3 classes différentes, et lui ferez traverser une quête qui sera décrite en une simple phrase. Il va combattre plusieurs ennemis à la suite, se reposer, se soigner et s'entraîner. À vous d'implémenter ce système pour donner vie au roleplay !

Les instructions

Avant tout, tâchez d'utiliser des `int` mais pas des `double` pour vos entiers. On fera en sorte que nos tests ne causent pas délibérément des *parseIntError* avec des chiffres trop grands.

1. Conception des personnages :

Vous devez commencer par concevoir et implémenter les classes des différents types de personnages. Vous pouvez les implémenter comme vous voulez tant que la spécification est respectée, mais on exige de voir :

- `Enemy` : La classe représentant les ennemis que le héros combat.
- `Hero` : La classe représentant le héros.

  Il y a 3 types de héros :

- Type attaque : lorsqu'il attaque, les dégâts qu'il inflige sont 2 fois ses points d'attaque normaux, mais il reçoit 2 fois les dégâts que l'ennemi inflige.
- Type defense : Il ne reçoit que la moitié des dégâts que l'ennemi inflige, mais il n'inflige que la moitié de ses points d'attaque quand il frappe l'ennemi.
- Type équilibre : Rien ne change.

(Indice : L'héritage participe à la beauté du code en évitant les répétitions, donc n'hésitez pas!)

Dans la phrase en input, le héros est de type Attaque quand son nom commence par A, il est de type Défense quand son nom commence par D, et il est de type Équilibre quand son nom commence par n'importe quoi d'autre.

2. Système d'expérience et de mise à niveau :

Pour mettre en œuvre l'expérience et le système de mise à niveau du héros, les exigences sont que :

- Le héros possède des attributs de santé et santé maximale (int **health**, **maxHealth**), de niveau (int **level**) et de points d'expérience (int **experience**).
- Le héros peut essayer de s'améliorer après avoir acquis de l'expérience, et les règles de mise à niveau doivent être calculées en fonction des points d'expérience. À mesure que vous montez de niveau, la santé maximale et les dégâts d'attaque de votre héros augmentent.

  Le héros gagne des points d'expérience à chaque fois qu'il bat un ennemi. Quand il en accumule assez, il augmente de niveau et améliore ses stats, puis ses points d'expérience reviennent à 0 jusqu'au prochain level up, où là :

  - Ses points de vie maximum augmentent de 12 (de plus, tous ses points de vie sont régénérés)
  - Ses points d'attaque augmentent de 6

Aussi, le héros peut atteindre un niveau maximum de 99. Donc tout point d'expérience gagné au-delà ne sert à rien.

La formule pour avoir le nombre de points d'expérience requis pour atteindre un niveau est :

exp_requis = 50 + niveau_voulu * 20 * (1.1^niveau_voulu)

Le héros commence au niveau 1 et a 0 points d'expérience. Ainsi, par exemple, atteindre le niveau 2 nécessite **50 + *2* * 20 * (1.1 ** *2*) = 98.4**, donc 99 points d'expérience (les points d'expérience sont un nombre entier donc on prend l'entier supérieur).

- Cependant, à chaque fois qu'il bat un ennemi, les ennemis suivants gagnent en force aussi :
  - Leurs points de vie initiaux augmentent de 10
  - Leurs points d'attaque augmentent de 5
  - Les points d'expérience qu'ils donnent augmentent de 8

  Ainsi, initialement, le tout premier ennemi que le héros combat a *100* points de vie, *25* points d'attaque, et donne *35* points d'expérience, tandis que le deuxième (si le héros survit au premier) aura *110* points de vie, *30* points d'attaque, et donnera *43* points d'expérience.

3. Système de combat :

Un combat se déroule de manière très simple : le héros inflige des dégâts à l'ennemi, puis l'ennemi inflige des dégâts au héros, et ainsi de suite jusqu'à ce que l'un d'entre eux meurt. Le héros frappe toujours en premier. Le héros ne récupère pas automatiquement ses points de vie après un combat, donc s'il combat trop d'ennemis à la suite, il peut mourir, donc échouer sa quête.

Aperçu

Pour lancer le programme, il faut d'abord aller au dossier source du fichier `Main`, puis faire :

```
javac Main.java
java Main "la phrase"
```

La phrase qui décrit la quête du héros sera ainsi mise en argument comme ci-dessus.

La phrase doit être obligatoirement sous la forme :
"[nom du perso
(String)],[points de vie **(nombre entier)**],[points d'attaque **(nombre entier)**],
[action],[action], [action], [action], ..."

Les 3 premiers mots de la phrase désignent le nom du héros, ses points de vie initiaux et ses points d'attaque initiaux.

Ensuite, chaque action est un String qui ne peut que prendre l'une des 4 formes suivantes :

- "**fought** *n* enemies", n étant un nombre entier donnant le nombre d'ennemis que le héros combat à la suite sans se soigner
- "**rested**" pour indiquer lorsque le héros se repose et soigne tous ses points de vie
- "**healed** *n* health points" n étant un nombre entier donnant le nombre de points de vie que le héros soigne (la vie du héros ne peut pas dépasser ses points de vie maximum)
- "**trained** to get *n* attack points" n étant un nombre entier donnant le nombre de points d'attaque que le héros gagne en s'étant entraîné

L'objectif du programme sera de faire exécuter la quête au héros, puis imprimer l'issue de la quête à la fin, dans laquelle on donne le nombre d'ennemis tués par le héros, le niveau qu'il a atteint et ses points de vie restants s'il a survécu. Le message sera imprimé différemment dépendamment de s'il est mort pendant la quête ou s'il a survécu à la fin.

Dans le dossier `Classes` fourni, vous avez la classe `ArgsProcessor` dans laquelle vous compléterez les *TODOs* pour faire fonctionner le programme, la classe `Hero` que vous allez compléter et qui est obligatoire pour faire marcher la fonction *doAction* dans ArgsProcessor, et la classe `Main` qui se contente d'exécuter la fonction *process* de ArgsProcessor avec les arguments donnés en ligne de commande.

Exemple:

Ainsi, voici 2 exemples d'exécution :

exemple 1

```
Input:

java Main "Kevin,150,20, fought 1 enemy, rested, trained to g
et 30 attack
points, fought 2 enemies"

Output:

In his quest, Kevin beat 3 enemies, attained level 2 and surv
ived with 162
HP!
```

exemple 2

```
Input:

java Main "Kevin,150,20, fought 65 enemies"

Output:
```

```
In his quest, Kevin died after beating 1 enemies and attainin
g level 1!
```

Pour tous les tests, assumez que la phrase mise en argument suit toujours parfaitement le format exigé. Un fichier texte contenant d'autres phrases exemples et l'output attendu sera ajouté sur StudiUM pour complémenter ces exemples.

Bon courage et bon codage !

Plagiat

Sachez que le plagiat entraine directement la note de **0** et mène à une **note dans votre dossier**. Tout partage de code ou copie de code qui n'est pas le vôtre, qu'il soit sous forme de texte, de photo ou tout autre format, est considéré comme du plagiat. Lors de la correction, nous allons utiliser plusieurs méthodes pour détecter le plagiat. Ne vous essayez pas, les conséquences seront sévères.

Vous pouvez consulter le site de l'université pour plus d'informations

Remise

Remettez tout votre code dans le même dossier sous format compressé .zip avec les noms des membres de l'équipe en camelCase avec une majuscule au début comme ceci : `TP1_PrenomNom1_PrenomNom2.zip` (où PrenomNom1 est votre nom ex: *MathisLaroche* et PrenomNom2 est celui de votre coéquipier.ère ex: *Enric Soldevila*). Remettez-le tout sur studium sous le devoir qui est affiché.

Pour compresser le fichier:

- MacOS: vous devriez avoir le bouton compresser lorsques vous sélectionnez les fichiers à compresser (https://support.apple.com/en-mz/guide/mac-help/mchlp2528/mac).
- Windows: je recommande WinRAR qui est un logiciel gratuit très répandu (https://www.win-rar.com/start.html?&L=10 oui leur site fait vieux, mais il s'agit du bon site).

Si vous avez des questions sur des instructions du devoir, n'hésitez pas à les poser sur le forum ! Il est possible qu'on fasse des modifications à l'énoncé pour clarifier les consignes, nous vous le ferons savoir si c'est le cas.

Correction

Les exercices sont à faire en équipe de **2**. Pour vous évaluer, nous allons nous fier à ces critères de correction :

- 55% **Fonctionnement du code** Votre code va être testé avec plusieurs valeurs d'entrée. Si certaines réponses ne sont pas valides, vous allez perdre des points. En revanche, si votre code fonctionne à merveille, vous allez avoir tous vos points ! Je vous conseille de bien tester votre programme.
- 10% **Compilation :** Votre programme arrive-t-il à passer la compilation ? Vous devez vous assurer que le code qu'on vous passe fonctionne avec le reste de votre programme tout restant inchangé. En effet, le squelette du programme qu'on vous fournit est sur quoi devra se baser le reste de votre code.
- 35% **Qualité du code :** Plus votre code est simple et facile à lire, plus vous allez avoir de points ! Voici des éléments qui contribuent à la beauté d'un code :
    - Est-ce que vos noms de variable sont descriptifs de leur utilisation ?
    - Est-ce que vous aérez vos expressions (par exemple, mettre des espaces avant et après un `=`) ?
    - Utilisez-vous des variables inutiles ? Évitez-vous la répétition de code ?
    - Est-ce que votre code est concis et bien commenté ?
    - Est-ce que votre code respecte bien les principes de la programmation orientée objet ?

Le travail doit être remis le 18 Février à minuit (11:59 pm) au plus tard!