Projet C++: Pokémon Auto-Battle

L3 MIASHS info

Objectifs :

Le but du projet est de concevoir un programme en C++ permettant de simuler un combat automatique entre deux joueurs et leurs équipes de Pokémon. Les joueurs disposent d'une liste de Pokémon (6 maximum). Pour un combat entre deux joueurs, chacun choisit au plus 3 Pokémon. Les Pokémon combattent dans l'ordre choisi. Un Pokémon reste en jeu tant qu'il n'est pas vaincu. À la fin du combat, le joueur à qui il reste au moins un Pokémon gagne. Après le combat, tous les Pokémon retrouvent leurs points de vie initiaux.
Chaque Pokémon est un objet dérivé d'une classe de base commune. Le combat entre Pokémon suit une séquence de tours jusqu'à la victoire de l'un des protagonistes, selon les règles suivantes :

- Le Pokémon le plus rapide a plus de chances de commencer en premier.
- Les Pokémon attaquent chacun leur tour.
- Dès que les points de vie (PV) d'un Pokémon atteignent 0, le combat est terminé et ce Pokémon est considéré comme vaincu.

Gestion du hasard :

Pour tout ce qui concerne le hasard, utilisez la fonction rand() qui renvoie un entier aléatoire. Il faut initialiser la graine avec srand(). Exemple :

```
#include <cstdlib>
#include <ctime>
#include <iostream>
using namespace std;
int main() {
    srand(time(0)); // À faire une seule fois au début du programme
    cout << rand() << endl;
    return 0;
}
```

Description fonctionnelle minimale

Remarque : Les classes présentées ci-dessous sont des propositions de base. Vous pouvez (et devez) les compléter. Votre implémentation peut être différente tant qu'elle met en œuvre de l'héritage.

Classe de base : Pokemon

Attributs génériques : nom, points de vie (PV), attaque, défense, vitesse.

Méthodes :

- `afficherStat()` : affiche les statistiques du Pokémon.
- `getType()` **:** méthode virtuelle pure renvoyant le type du Pokémon sous forme d'entier.
- `Attaque(Pokemon (& ou *) cible)` : gère l'algorithme d'attaque entre deux Pokémon.
- `PerdrePV(int montant)` : réduit les PV du Pokémon ciblé.

Classes dérivées pour chaque type

Exemples : FirePokemon, WaterPokemon, GrassPokemon.

- Redéfinir la méthode `getType()`**.**
- Possibilité, dans un second temps, d'ajouter une attaque spéciale selon le type ou un effet (ex. brûlure).

Création aléatoire de Pokémon
Créer une fonction (non membre) qui génère un Pokémon aléatoirement. Pour simplifier, on peut considérer que tous les attributs ont une valeur entre 0 et 100.

Gestion des affrontements

Chaque combat se déroule en tours :

- Le Pokémon A attaque le Pokémon B,
- Puis le Pokémon B attaque le Pokémon A,

…
Cette alternance se poursuit jusqu'à ce qu'un Pokémon soit KO (PV ≤ 0).

Calcul des dégâts :
Le calcul des dégâts peut intégrer :

- Les statistiques d'attaque et de défense des Pokémon.
- Un bonus/malus de type (ex. Feu > Plante, Eau > Feu, etc.).
- Un facteur aléatoire faible pour introduire de la variabilité.

Exemple de formule de dégâts :

```
Dégâts = (AttaqueAttaquant * PuissanceAttaque / DéfenseCible)
         * MultiplicateurType
         * FacteurAléatoire
```

Multiplicateur de type :

- x2 pour une attaque très efficace (eau sur feu, par exemple)
- x1 pour une attaque neutre
- x0.5 pour une attaque peu efficace (feu sur eau, par exemple)

Fonction de combat entre deux Pokémon
Créer une fonction qui, étant donnés deux Pokémon, détermine le vainqueur du combat.

Classe Joueur

Chaque joueur possède une liste de Pokémon (vous pouvez utilisez la STL). Un joueur a au moins un attribut nom.

Méthodes :

- `AjouterPokemon(Pokemon* p)` : ajoute un Pokémon à la liste du joueur.
- `SelectionnerEquipe()` **:** permet de sélectionner une équipe de 3 Pokémons (ou plus) parmi ceux du joueur.

Créer une fonction pour générer un joueur aléatoirement.
Créer une fonction de combat entre deux équipes de joueurs, détermine le vainqueur.

Bonus possibles (liste non exhaustive)

- Ajouter davantage de types ou de sous-types.
- Proposer différents types d'attaques.
- Gérer des états (endormi, paralysé, etc.).
- Organiser un tournoi.

Travail à rendre

Le travail se fait en binôme. Le projet doit couvrir tout le sujet (hors bonus). La note sera basée sur :

- La qualité du code (structure, commentaires, lisibilité). Rendu sur l'ENT.
- La soutenance : celle-ci portera sur des questions relatives au code. Il n'y a rien de particulier à préparer, mais une démonstration du programme sera demandée. La soutenance se fera avec votre enseignant de TP lors de la dernière séance.