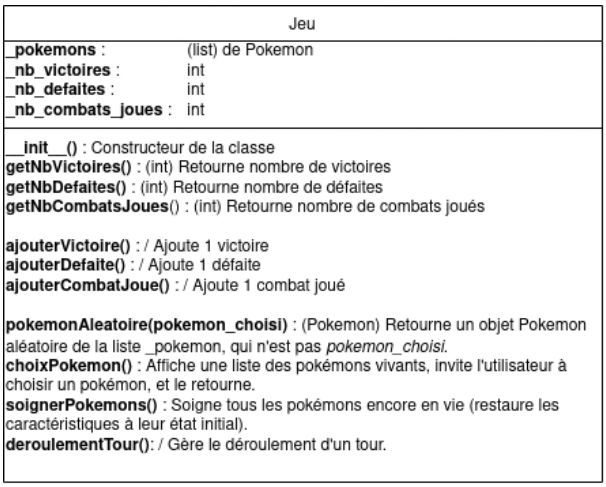
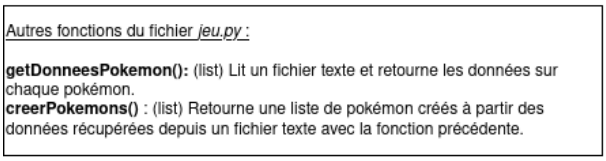
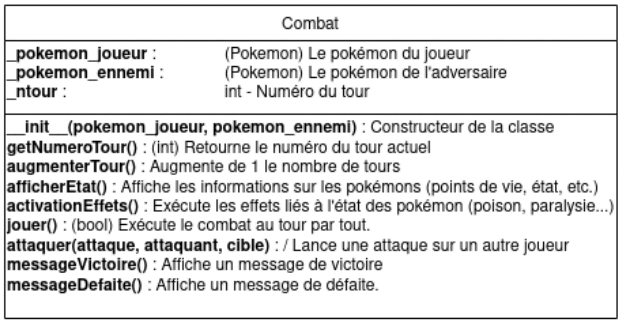
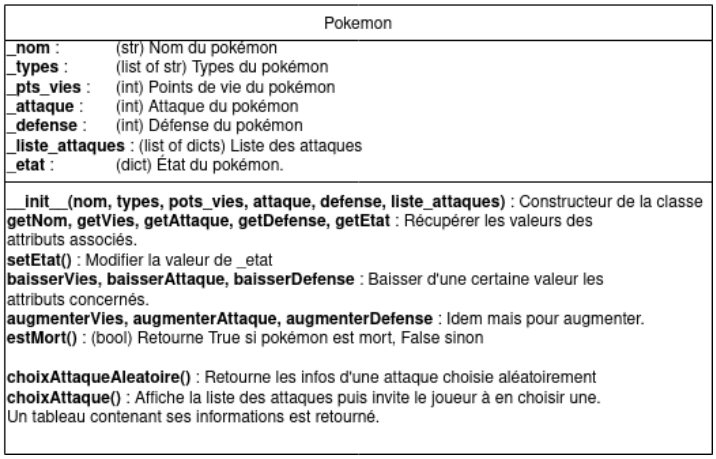
Sujet : Implémentation d'un mini-jeu Pokémon en Python à partir d'un diagramme UML

L'objectif de ce projet est d'implémenter un mini-jeu de combat Pokémon en utilisant le langage Python, en respectant les classes et méthodes décrites dans le diagramme UML fourni.









Description générale

Le jeu consiste à gérer des combats entre des Pokémon disposant de points de vie, d'attaque, de défense et d'un ensemble d'attaques. Chaque tour de combat permet à un Pokémon d'attaquer un autre, jusqu'à ce que l'un des deux soit vaincu.

Le système est divisé en trois grandes classes :

1. **Jeu** : Gère la progression globale du jeu, le suivi des victoires/défaites et les combats joués.
2. **Combat** : Gère le déroulement d'un affrontement entre deux Pokémon.
3. **Pokemon** : Représente un Pokémon avec ses caractéristiques (nom, types, points de vie, attaque, défense, attaques disponibles).

Deux fonctions supplémentaires permettent de charger les données des Pokémon à partir d'un fichier texte.

Travail demandé

1. **Implémenter la classe** `Pokemon` avec tous ses attributs (`nom`, `types`, `vies`, `attaque`, `defense`, `liste_attaques`, `etat`) et ses méthodes (gestion des états, attaques, augmentation/diminution des caractéristiques, etc.).

2. **Implémenter la classe** `Combat` qui gère le tour par tour, les attaques lancées, et qui affiche les résultats (victoire/défaite).

3. **Implémenter la classe** `Jeu` qui supervise l'ensemble des combats, garde trace du nombre de victoires, défaites et combats joués, et permet de choisir un Pokémon aléatoire ou parmi ceux disponibles.

4. **Coder les fonctions utilitaires** :

    - `getDonneesPokemon()` : lit un fichier texte et renvoie les données de chaque Pokémon.
    - `creerPokemons()` : crée les objets Pokémon à partir des données lues.

5. **Écrire un programme principal** qui :

    - Initialise la liste de Pokémon,
    - Lance des combats,
    - Permet de suivre l'évolution du joueur (victoires, défaites, combats).

Contraintes

- Respecter le diagramme UML fourni (noms de méthodes, signatures, responsabilités).
- Commenter le code afin d'expliquer vos choix d'implémentation.
- Assurer la modularité (chaque classe dans un fichier séparé est recommandé).

Bonus (optionnel)

- Ajouter des effets spéciaux aux attaques (paralysie, poison, sommeil, etc.).
- Sauvegarder les scores dans un fichier pour garder une trace entre les parties.