

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.

Jeu	
pokemons :	(list) de Pokemon
nb_victoires :	int
nb_defaites :	int
nb_combats_joues :	int
__init__() : Constructeur de la classe getNbVictoires() : (int) Retourne nombre de victoires getNbDefaites() : (int) Retourne nombre de défaites getNbCombatsJoues() : (int) Retourne nombre de combats joués ajouterVictoire() : / Ajoute 1 victoire ajouterDefaite() : / Ajoute 1 défaite ajouterCombatJoue() : / Ajoute 1 combat joué pokemonAleatoire(pokemon_choisi) : (Pokemon) Retourne un objet Pokemon aléatoire de la liste <code>_pokemon</code> , qui n'est pas <code>pokemon_choisi</code> . choixPokemon() : Affiche une liste des pokémons vivants, invite l'utilisateur à choisir un pokémon, et le retourne. soignerPokemons() : Soigne tous les pokémons encore en vie (restaure les caractéristiques à leur état initial). deroulementTour() : / Gère le déroulement d'un tour.	
Autres fonctions du fichier <code>jeu.py</code> : getDonneesPokemon() : (list) Lit un fichier texte et retourne les données sur chaque pokémon. creerPokemons() : (list) Retourne une liste de pokémon créés à partir des données récupérées depuis un fichier texte avec la fonction précédente.	

Combat	
pokemon_joueur :	(Pokemon) Le pokémon du joueur
pokemon_ennemi :	(Pokemon) Le pokémon de l'adversaire
ntour :	int - Numéro du tour
__init__(pokemon_joueur, pokemon_ennemi) : Constructeur de la classe getNumeroTour() : (int) Retourne le numéro du tour actuel augmenterTour() : Augmente de 1 le nombre de tours afficherEtat() : Affiche les informations sur les pokémons (points de vie, état, etc.) activationEffets() : Exécute les effets liés à l'état des pokémon (poison, paralysie...) jouer() : (bool) Exécute le combat au tour par tour. attaquer(attaque, attaquant, cible) : / Lance une attaque sur un autre joueur messageVictoire() : Affiche un message de victoire messageDefaite() : Affiche un message de défaite.	

Pokemon	
nom :	(str) Nom du pokémon
types :	(list of str) Types du pokémon
pts_vies :	(int) Points de vie du pokémon
attaque :	(int) Attaque du pokémon
defense :	(int) Défense du pokémon
liste_attaques :	(list of dicts) Liste des attaques
etat :	(dict) Etat du pokémon.
__init__(nom, types, pots_vies, attaque, defense, liste_attaques) : Constructeur de la classe getNom, getVies, getAttaque, getDefense, getEtat : Récupérer les valeurs des attributs associés. setEtat() : Modifier la valeur de <code>_etat</code> baisserVies, baisserAttaque, baisserDefense : Baisser d'une certaine valeur les attributs concernés. augmenterVies, augmenterAttaque, augmenterDefense : Idem mais pour augmenter. estMort() : (bool) Retourne True si pokémon est mort, False sinon choixAttaqueAleatoire() : Retourne les infos d'une attaque choisie aléatoirement choixAttaque() : Affiche la liste des attaques puis invite le joueur à en choisir une. Un tableau contenant ses informations est retourné.	

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.