Compte rendu du projet SQL

Dénomination: "Gestionnaire Pokémon"

Description: Le programme utilise une API pour récupérer les données des Pokémon, crée une base de données SQLite pour stocker ces données, et offre des fonctionnalités telles que la recherche de Pokémon et de dresseurs, l'ajout de nouveaux dresseurs, et la simulation de combats entre dresseurs et Pokémon.

Détails de l'implémentation:

1. **Création de la base de données**:
   - Deux tables sont créées dans la base de données SQLite: une pour les Pokémon (table `Pokemon`) et une pour les dresseurs (table `Trainer`).
   - Les colonnes des tables incluent des attributs tels que le nom, l'attaque, la défense, les PV, le type, etc.
2. **Insertion des données des Pokémon**:
   - Les données des Pokémon provenant de l'API sont insérées dans la table `Pokemon`.
3. **Insertion des données des dresseurs**:
   - Les données des dresseurs, comprenant le nom et les Pokémon qu'ils possèdent, sont créées manuellement et insérées dans la table `Trainer`.
4. **Fonctions de recherche**:
   - Les fonctions `search_pokemon` et `search_trainer` permettent de rechercher un Pokémon ou un dresseur dans la base de données en fonction du nom.
   - Les résultats sont affichés avec des détails tels que les statistiques, le type, et l'évolution.
5. **Fonction de création d'un nouveau dresseur**:
   - La fonction `new_trainer` permet à l'utilisateur d'entrer son nom et attribue aléatoirement six Pokémon au nouveau dresseur, qui sont ensuite ajoutés à la base de données.
6. **Fonction de simulation de combat**:
   - La fonction `fight` permet à l'utilisateur de sélectionner deux dresseurs et leurs Pokémon pour simuler un combat. Les dégâts sont calculés en fonction des statistiques des Pokémon.
7. **Menu principal**:
   - Un menu principal permet à l'utilisateur de choisir entre la recherche, la création d'un nouveau dresseur, la simulation de combat, ou la sortie du programme.
8. **Boucle principale**:
   - Le programme est conçu pour s'exécuter en boucle tant que l'utilisateur ne choisit pas de sortir.

Répartition des tâches:

- **Anaël:** J'ai travaillé sur le côté SQL et implémentation de combat de ce projet, ce qui inclut la création de la table Pokémon et Trainer, l'insertion des données dans celle-ci et la fonction de combat.

Problèmes rencontrés: La mise en œuvre de la fonction de combat a présenté des défis, en particulier pour calculer les dégâts en fonction des statistiques des Pokémon. Dans la fonction `get_pokemon_by_id` , le `return cursor.fetchone()` renvoyait un tuple, ce qui a causé des problèmes, j'ai dû donc modifier cette ligne pour qu'elle renvoie une liste avec `return list(cursor.fetchone())` . Mais l'implémentation de ce code a causé d'autres problèmes: la fonction `search_pokemon` ne marchait plus. J'ai donc remis l'ancien code, mais j'ai du ajouter 2 variables à la fonction `fight` : pokemon1pv et pokemon2pv.

- **Marouan:** Je me suis occupé de la fonction de recherche, avec l'implémentation de la recherche de dresseur et de Pokémon ainsi que le menu principal.

  Problèmes rencontrés: La mise en place de `search_pokemon` a été assez compliquée, vu que je devais prévoir toutes les réactions possibles de l'utilisateur, notamment le fait qu'il entre un nom incomplet. Puis j'ai découvert la possibilité de mettre le caractère "%" au niveau de la condition de la requête SQL pour palier à ce problème. De plus, j'ai rencontré un problème d'affichage dans le résultat de la recherche notamment avec l'affichage de "evolution" et "pre_evolution" qui me renvoyé l'identifiant des Pokémon dû aux caractéristiques de la Table Pokemon. Néanmoins, la mise en place de la fonction `get_pokemon_by_id` qui a été implémentée et utilisée par Anaël, j'ai pu renvoyé les noms des Pokémons, de la "Pre-Evolution" et aussi de leur "Evolution".