Partie 1 - Construction d'un Modèle pour la gestion des Pokémons

Avant-propos

Ce projet va vous permettre de mettre en œuvre, en 3 étapes, les concepts vus en TD et lors des premières séances de TP.
L'objectif de ce projet est de construire une WebApp qui affichera des informations sur les *Pokémons*, et permettra de rechercher, trier, filtrer les Pokémons et leurs informations en fonction de divers critères qui vous seront détaillés ultérieurement. Cette WebApp permettra en outre de simuler des affrontements entre Pokémons dans une version simplifiée.
Une *WebApp* est une page web qui se comporte comme une application traditionnelle, à savoir un rafraîchissement dynamique du contenu affiché sans faire appel à des chargements de nouvelles pages HTML. Une telle application porte aussi le nom d'*Application Web Monopage* (*Single Page App*). Vous aurez à manipuler le DOM.
Les données qui serviront à remplir votre application proviendront d'un API Web Service.

Introduction

Dans cette partie, vous allez vous focaliser sur la lecture des données directement depuis des fichiers contenant des structures JSON. Nous verrons dans une partie future comment récupérer ces informations depuis un WebService.

1 Les fichiers de données

Récupérez les fichiers sur l'ENT :

- `pokemon.json` : les caractéristiques de bases des Pokémons.
- `pokemon_type.json` : les types de Pokémon.
- `pokemon_moves.json` : les attaques que pourra effectuer chaque Pokémon, en fonction de sa forme. Ces mouvements sont décomposés en attaques rapides et en attaques "chargées".
- `fast_moves.json` : le détail des attaques rapides, dont leurs types.
- `charged_moves.json` : le détail des attaques chargées, dont leurs types.
- `type_effectiveness.json` : le coefficient multiplicateur appliqué à la force d'une attaque en fonction de son type et du type du Pokémon qui subit l'attaque.
- `cp_multiplier.json` : le coefficient multiplicateur à appliquer aux caractéristiques de base d'un Pokémon en fonction de son niveau.
- `generation.json` : la génération du Pokémon. Jusque fin 2021, il y a eu 8 générations de Pokémons. Chaque génération apporte un ensemble de nouveaux Pokémons. Annoncée lors du *Pokémon Presents* du 27 février 2022, la neuvième génération vient de sortir, mais n'est pas décrite dans cet ensemble de données.

1.1 Les Pokémons

Cet ensemble de données contient la description de 898 Pokémons des 8 premières générations. Analysez la structure JSON d'un Pokémon décrite dans le fichier `pokemon.json`. On notera la présence des attributs suivants : `pokemon_id`, `pokemon_name`, `form`,`base_attack`, `base_defense` et `base_stamina`. Le fichier `generation.json` détermine la génération d'un Pokémon.

Ces données décrivent les caractéristiques de base d'un Pokémon. Certaines d'entre-elles pourront évoluer au fil du jeu. Cette évolution est décrite dans le fichier `cp_multiplier.json`. Ces caractéristiques de base seront utiles pour déterminer le déroulement d'un affrontement entre deux Pokémons.

Comme il est prévu à terme d'afficher les informations sur les Pokémons dans la WebApp, un dépôt contenant les images des Pokémons est disponible à l'adresse suivante :
`https://www.adl-web.fr/pokemon-base/`
complétée par un suffixe en fonction des types d'images que vous souhaitez récupérer :

- `images/<num_pokemon>.png` : les images complètes des Pokémons.
- `thumbnails/<num_pokemon>.png` : les images miniatures des Pokémons.

Par exemple : https://www.adl-web.fr/pokemon-base/images/001.png. Le numéro du Pokémon est donc codé sur 3 chiffres.

1.2 Les types de Pokémons

Le fichier `pokemon_type.json` décrit les différents types de Pokémons en fonction de leur forme. La forme permet de distinguer des variantes d'un même Pokémon (par exemple : "Normal", "Fall_2019", "Alola").

Un Pokémon d'une certaine forme possède 1 ou 2 types maximum. Si on analyse ce fichier, on peut dénombrer 18 types différents pour les Pokémons. Ces types s'appliquent également aux attaques.

Ce type va conditionner les attaques que chaque Pokémon pourra effectuer, mais aussi la résistance aux attaques qu'il subira.

1.3 Les attaques

Le fichier `pokemon_moves.json` décrit les différents types de mouvements (attaques) que les Pokémons peuvent effectuer en fonction de leur identifiant et de leur forme. Un Pokémon ne peut effectuer un mouvement que s'il correspond à son type. On retrouve les types de ces attaques dans les fichiers `fast_moves.json` et `charged_moves.json`.

Le principe d'un combat de Pokémons est que chacun des Pokémons attaque à son tour celui auquel il fait face. Son attaque, en fonction de son type et du type de son adversaire va avoir une certaine efficacité (elle pourrait même être totalement inefficace). Cette efficacité est décrite dans le fichier `type_effectiveness.json`. La clé de chacun des éléments de cette structure de données est le type de l'attaque lancée, et en fonction du type du Pokémon défenseur donne un coefficient multiplicateur.

Si le coefficient est supérieur à 1.0, alors on peut dire que l'attaque est efficace. S'il est inférieur à 1.0, on peut dire que le défenseur montre une certaine résistance à l'attaque.

A chaque tour, le pokemon défenseur va donc perdre un certain nombre de points de vie sur son total. Ce nombre de points de vie est donné par la formule décrite ici : Pokepedia - Calcul des dégâts. Nous nous occuperons des combats dans la suite, dans la mesure du possible.

2 Réalisation du Modèle

L'objectif est donc dans un premier temps d'implanter un modèle de données en JavaScript à partir de la description des données faites dans les différents fichiers que nous venons de citer. L'étape suivante sera d'utiliser les classes et leurs méthodes pour afficher les données à la demande dans la WebApp.

Nous vous conseillons de construire un diagramme de classes et de le compléter au fur et à mesures des étapes présentées ci-dessous.

2.1 La classe *Pokemon*

Q1. A partir de la description faite plus haut et de votre analyse des structures dans les fichiers JSON, proposez une classe `Pokemon`. Prévoyez une méthode `toString()` synthétique. Pour simplifier l'indexation, on ne désire modéliser que les Pokémons dont la forme (attribut `form`) a pour valeur "Normal".

Q2. Concevez une fonction `import_pokemon()` qui lit la source de données et crée des objets `Pokemon` que vous stockerez dans un tableau dont les clés sont les `id` des Pokémons. Ce tableau sera une variable de classe nommée `all_pokemons`.

2.2 La classe *Type*

Q1. A partir des fichiers `pokemon_type.json` et `type_effectiveness.json`, proposez une classe `Type`. Ce type permettra donc de connaître l'efficacité d'un type d'attaque contre un type de défenseur. N'oubliez pas la méthode `toString()`. De même que pour les Pokémons, on ne gardera que les types des formes dont la valeur est "Normal". Une variable de classes nommée `all_types` contiendra l'ensemble des types de Pokémons et sera indexé par le nom du type.

Q2. Complétez la fonction `import_pokemon()` qui lit la source de données et crée des objets `Type` au fur et à mesure que les Pokémons sont importés, si ce type n'est pas déjà dans `all_types`. Dans la classe `Pokemon`, vous devrez devrez faire le lien avec les types.

Q3. Complétez la classe `Pokemon` avec
— la méthode `getTypes()` qui retourne la liste des types (objets `Type`).

2.3 La classe *Attack*

Q1. A partir des fichiers `pokemon_moves.json`, `fast_moves.json` et `charged_moves.json`, proposez une classe `Attack` et complétez la classe `Pokemon`. N'oubliez pas la méthode `toString()`. Une variable de classe `all_attacks` contiendra toutes les attaques existantes, indexées par l'identifiant de l'attaque.

Q2. Complétez la fonction `import_pokemon()` qui lit la source de données et crée des objets `Attack` au fur et à mesure que les Pokémons sont importés, si cette attaque n'est pas déjà dans `all_attacks`.

Q3. Complétez la classe `Pokemon` avec une méthode `getAttacks()` qui donne la liste des attaques (objets `Attack`).

3 Tests

Ecrivez les fonctions qui permettent de répondre aux questions suivantes. Vous testerez (afficherez) dans la console d'un navigateur ou dans la page HTML, à votre convenance.

1. fonction `getPokemonsByType(typeName)` donnant la liste des Pokémons par type, celui-ci étant passé en argument.
2. fonction `getPokemonsByAttack(attackName)` donnant la liste des Pokémons par attaque, celle-ci étant passée en argument.
3. fonction `getAttacksByType(typeName)` donnant la liste des attaques par type, celui-ci étant passé en argument.
4. fonction `sortPokemonByName()` donnant la liste des Pokemons triés par nom dans l'ordre alphabétique.
5. fonction `sortPokemonByStamina()` donnant la liste des Pokémons triés dans l'ordre décroissant d'endurance (`stamina`).
6. fonction `getWeakestEnemies(attack)` qui retourne la liste des `Pokemon` qui ont le moins de résistance à une attaque donnée choisie parmi les attaques du Pokémon.
7. fonction `getStrongestEnemies(attack)` qui retourne la liste des `Pokemon` qui ont le plus de résistance à une attaque donnée choisie parmi les attaques du Pokémon.

D'autres fonctions vous seront demandées ultérieurement.

Ce document est mis à disposition selon les termes de la licence Creative Commons "Attribution – Pas d'utilisation commerciale – Partage dans les mêmes conditions 3.0 non transposé".