Division des enseignements en informatique

TP Pokemons

Au cours de ce TP, nous allons chercher à réaliser un jeu inspiré des jeux Pokemons : le joueur incarnera un dresseur possédant des pokemons qui pourront affronter d'autres pokemons.

Dans cette première version du jeu, nous n'implémenterons pas de carte à l'intérieur de laquelle le joueur pourrait se déplacer : le jeu se déroulera au tour par tour et, à chaque tour de jeu, un événement tiré aléatoirement se produira. Les événements que nous gèrerons seront les suivants :

— un pokemon sauvage apparait : le dresseur peut l'affronter, tenter de le capturer ou s'enfuir ;
— un dresseur apprait et vous affronte ;
— il ne se passe rien, vous pouvez vous reposer.

Pour ce travail, nous nous efforcerons de respecter quelques principes dans l'organisation du code : la partie métier (les fonctions pour jouer) sera séparée de la partie l'interface graphique (affichage et interactions avec le joueur), elle-même distincte des interrogations de la base de données.

1. Dans votre IDE préféré, créez un nouveau projet `Pokemon`.
2. Ajoutez trois packages à votre projet (si votre IDE préféré est le bloc note, nous vous rappelons qu'un package est simplement un répertoire avec un fichier `__init__.py` à sa racine) : `metier`, `ihm` et `bdd`.
3. Récupérez le fichier `affichage.py` et placez le dans le package `ihm`. Il contient trois méthodes :
   — `afficher(message)` utilisée pour afficher des messages dans la console et les copier dans un fichier de log ;
   — `logger(message)` pour écrire un message dans un fichier de log ;
   — `choisir(liste_choix, message)` pour inviter l'utilisateur à choisir un élement parmis une liste de choix.

1 Le modèle de l'application

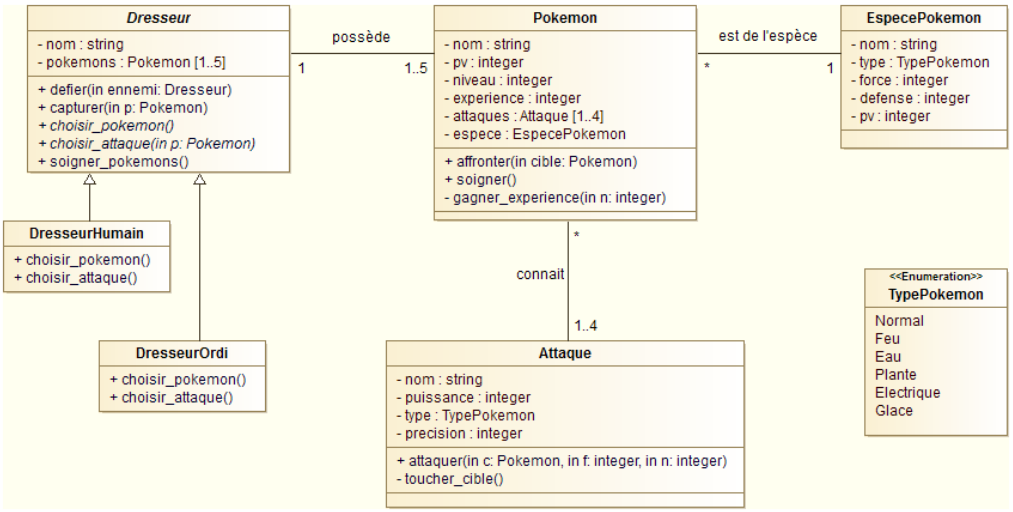

FIGURE 1 – Diagramme de classes de notre jeu de pokemon

Un dresseur peut posséder de 1 à 5 pokemons. Il est soit de type humain (`DresseurHumain`), et dans ce cas les méthodes `choisir_pokemon()` et `choisir_attaque()` requièrent une interaction de l'utilisateur, soit de contrôlé par l'ordinateur, et les méthodes de choix retournent alors un élément tiré au hasard parmis les possibilités. Un pokemon sauvage est considéré comme appartenant à un dresseur.

Une classe `PokemonEspece` permet de stocker les informations communes à tous les pokemons d'une espèce donnée : type de pokemon, force, défense, vitesse, etc. Il s'agit des valeurs de base pour tous les pokemons de cette espèce. Pour un pokemon donné, ces caractéristiques sont pondérées par son niveau.

Un pokemon connaît plusieurs attaques. Chacune d'elle possède une puissance et une précision qui déterminera si elle est esquivée par le pokémon attaqué ou touche sa cible.

Enfin, pokemons et attaques sont d'un type donné (normal, feu, eau, électrique, etc.). La correspondance des types a une influence sur les dégats infligés lors d'un combat : par exemple une attaque de type *eau* est deux fois plus efficaces sur un pokemon de type *feu* que sur un pokemon *normal*.

	Normal	Feu	Eau	Plante	Electrique
Normal	1	1	1	1	1
Feu	1	0.5	0.5	2	1
Eau	1	2	0.5	0.5	1
Plante	1	0.5	2	0.5	1
Electrique	1	1	2	0.5	0.5

La formule pour calculer les dégats est la suivante :

```
degats = (((niveau * 0.4 + 2)* force * puissance)/(defense * 50)+ 2)* coeff
```

où :

— niveau est le niveau du pokemon attaquant ;
— force est la force du pokemon attaquant ;

— puissance est la puissance de l'attaque ;
— defense est la défense du pokemon attaqué ;
— coeff est le coefficient d'efficacité en fonction des types de l'attaque et du pokémon attaqué.

Les pokemons victorieux d'un combat gagnent de l'expérience, ce qui leur permet ensuite de monter de niveau. Nous utilisons les règles suivantes pour les montées de niveau :

— expérience gagnée = niveau du pokemon battu / 2
— si expérience > niveau :
  — niveau = niveau + 1
  — expérience = 0

Notre modèle ne permet pas de soigner un pokemon pour lui remonter son niveau de vie. Pour compenser, nous remettrons les points de vie des vainqueurs à leur maximum après chaque combat.

Le déroulement d'un combat est synthétisé à l'aide d'un diagramme de séquence :

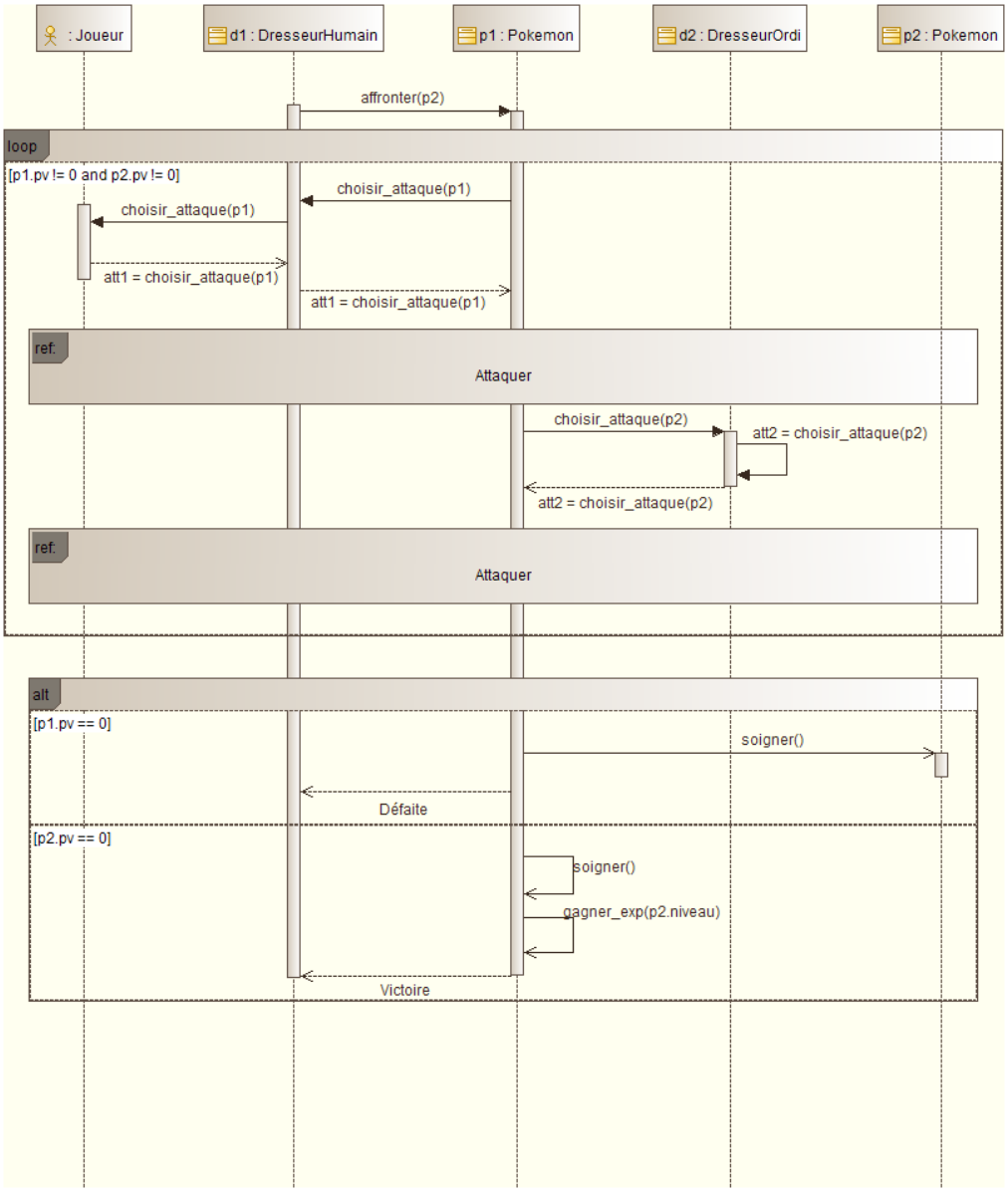

FIGURE 2 – Diagramme de séquence du déroulement d'un combat

Le cadre de référence *Attaquer* renvoie au diagramme suivant :

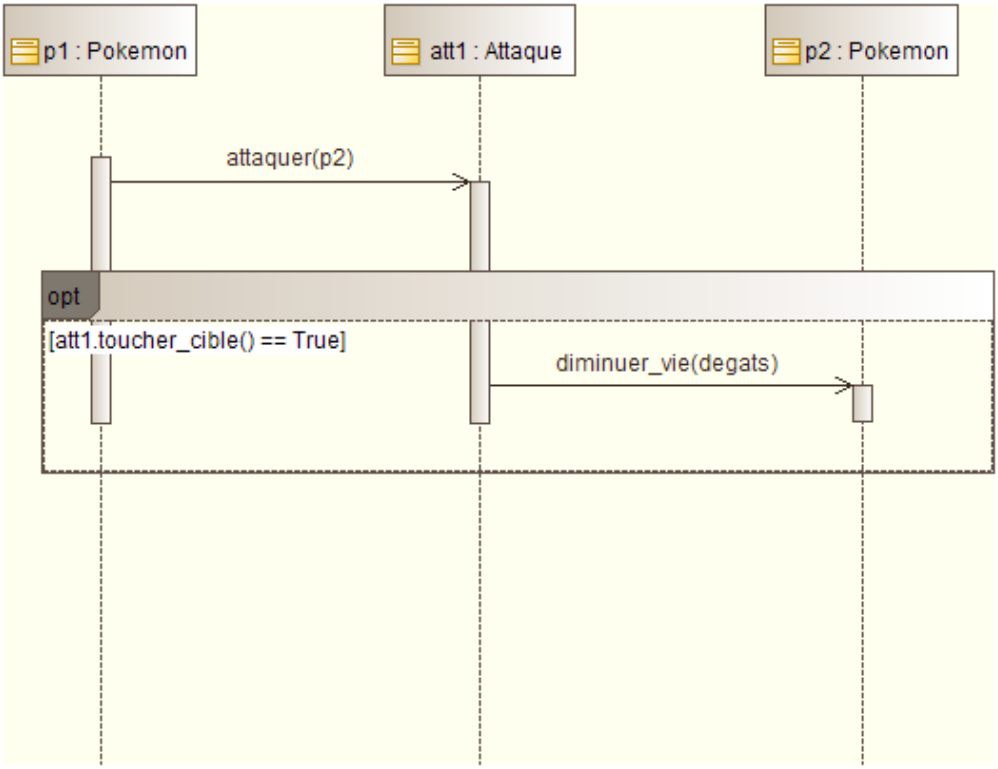

FIGURE 3 – Diagramme de séquence d'une attaque'

2 Les structures de base du jeu

Dans cette partie nous allons créer trois classes au coeur de notre programme `Pokemon`, `PokemonEspece` et `Attaque` respectivement dans les modules `pokemon.py`, `pokemon_espece.py` et `attaque.py` du package `metier`.

4. Ajoutez ces trois fichiers à votre projet et définissez y les trois classes.

Le constructeur de la classe `PokemonEspece` attend cinq paramètres :

— le nom de l'espèce (`Pikachu` par exemple) ;
— le type de l'espèce (par exemple `Feu`) ;
— les niveaux de force, défense et points de vie des pokemons de l'espèce.

Pour cette classe et pour les suivantes, les attributs doivent être marqués comme privés. Vous écrirez des méthodes de lecture de ces attributs lorsqu'ils auront besoin d'être lu en dehors de la classe.

5. Ajoutez le contructeur de la classe `PokemonEspece`.
6. Ajoutez également une méthode gérant l'affichage d'une instance de `PokemonEspece` sous la forme `Pikachu (pokemon ELECTRIQUE)`.

Le constructeur de la classe `Pokemon`attend trois paramètres obligatoires :

— le nom du pokemon (par exemple `Alfred`) ;
— l'espèce de ce pokemon (une instance de la classe `PokemonEspece`) ;

— la liste non vide des attaques connues par le pokemon (des instances de la classe `Attaque` que nous n'avons pas encore crée).

Le niveau du pokemon est un paramètre optionnel. S'il n'est pas renseigné, sa valeur sera de 1. L'expérience d'un pokemon sera toujours initialisée à 0. Enfin ses points de vie sont calculés à partir des points de vie des pokemons de son espèce et de son niveau :

```
pv = int(pv_espece * (niveau + 5)/ 4)
```

7. Ajoutez le contructeur de la classe `Pokemon`.
8. Prévoyez également une méthode pour que l'affichage d'une instance de pokemon contienne que son nom.

Pour créer une nouvelle attaque, nous devons préciser son nom, sa puissance et son type. Une précision peut également être précisée, sinon elle est prise égale à 90.

L'affichage d'une attaque fournit une réprésentation de la forme `Attaque éclair (ELECTRIQUE - 18)` où 18 serait la puissance de l'attaque.

9. Complétez la classe `Attaque`.

3 Gestion d'un combat

Nous allons maintenant implémenter les méthodes utilisées lors d'un combat entre 2 pokemons.

La méthode infligeant des dégats et la méthode `attaquer()` de la classe `Attaque`. Sa signature est la suivante :

```
attaquer(cible Pokemon, force integer, niveau integer)
```

où `force` et `niveau` sont respectivement la force de l'espèce du pokemon attaquant et le niveau du pokemon.

10. Ecrivez dans la classe Pokemon l'accesseur de l'attribut `pv`. Vous vous assurerez que la valeur reste dans les bornes permises par le modèle (entre 0 et `pv_espece * (niveau + 5)`).
11. Implémentez la méthode `attaquer()`. Nous vous rappelons que la formule pour calculer les dégats d'une attaque est donnée en introduction et qu'un test de précision doit être lancé pour déterminer si l'attaque touche sa cible ou pas.
12. Dans la classe `Pokemon`, écrivez les méthodes `soigner()` et `gagner_experience(val)` qui sont appelées lorsqu'un pokemon gagne un combat.
13. En vous référant au diagramme de séquence donné en introduction pour le déroulement d'un combat, écrivez une méthode `affronter(cible)`. Un pokemon attaquera pour l'instant toujours avec la première attaque de la liste des attaques connues. La méthode retournera un booléen : `True` en cas de victoire et `False` sinon.
14. Si vous ne n'avez pas déjà fait, prévoyez des messages informant le joueur de ce qu'il se passe lors d'un combat (qui attaque qui, est-ce l'attaque a touché, combien de dégat elle a fait, etc.).

4 Ajout des dresseurs

A ce stade le jeu n'est pas très intéressant car tout est automatique. Pour permettre au joueur de choisir ses pokemons / attaques, nous allons maintenant ajouter la classe `Dresseur`

15. Dans un nouveau script `dresseur.py` ajoutez une classe abstraite `Dresseur` et deux classes qui en hérite `DresseurHumain` et `Dresseur Ordi`.
16. Ajoutez la méthode `choisir_attaque(pokemon)` dans ces trois classes (elle est abstraite dans la classe abstraite `Dresseur`). Pour la classe `DresseurHumain`, la méthode nécessitera un choix du joueur.

Note : Nous vous rappelons que toutes les interractions avec le joueur (affichage de message dans la console, choix d'une action) doivent être implémentées dans le package 'ihm'.

17. Modifiez la fonction `affronter(cible)` précédente pour que le choix des attaques utilise les fonctions codées question 15.
18. Sur le même principe, ajouter des méthodes `choisir_pokemon()` dans le classe `Dresseur` et ses dérivées.
19. Complétez le programme avec une méthode `defier(ennemi)` permettant à un dresseur de défier un dresseur ennemi. Les deux dresseurs s'affrontent jusqu'à ce que l'un d'eux n'ai plus de pokemon.

5 Utilisation d'une base de données

Pour rendre notre jeu pleinement opérationnel, il nous reste encore à permettre de sauvegarder une partie pour y rejouer plus tard. Nous utiliserons pour cela une base de données SQLite. Nous en profiterons pour initialiser les pokemons et dresseurs du jeu avec de enregistrements dans cette base.

20. Récupérez les fichiers `__main__.py`, `utilitaire.py` que vous placerez respectivement à la racine du projet et dans l package `metier`. Placez également les deux fichiers `config_bdd.py` et `acces_bdd.py` dans le package `bdd`.

— Le fichier `__main__.py` sera la fichier à exécuter pour lancer le jeu. Il contient uniquement des appels de fonctions situées dans les deux autres fichiers.
— `acces_bdd.py` contient quatre fonction permettant d'utiliser une base de données SQLite. Ces fonctions ne sont pas encore implémentées.
— `config_bdd.py` contient une unique fonction `creer_bdd(db_name)` avec les instructions SQL permettant de créer une base de données pour notre jeu. Ce fichier utilise les quatre fonctions d'`acces_bdd.py`.
— `utilitaire.py` contient diverses fonctions utiles dans le programme. Certaines sont déjà implémentées. D'autres le seront dans la suite de ce TP.

Dans tous les cas, la documentation vous permet de comprendre ce que doivent faire les différentes fonctions.

21. Complétez les quatre fonctions du script `acces_bdd.py` :
22. Implémentez la fonction `initialiser_partie(db_name)` qui est appelée lorsqu'une partie a été chargée mais qu'elle ne contient pas de dresser humain (il s'agit donc d'une nouvelle partie). La fonction doit demander au joueur le nous qu'il souhaite donner à son dresseur, la pokemon avec lequel il désire commencer (bulbizare, salamèche ou bulbizare) comment il veut le nommer.

23. Il ne reste plus qu'à permettre de sauvegarder la partie pour revenir y jouer après avoir fermé le jeu. Implémentez la méthode `sauvegarder_partie()`.