SAÉ 2.1&2

Premier livrable

Développement du modèle

1 Introduction

L'objectif de cette double SAÉ va être de développer un **Livre dont vous êtes le héros (LDVEH)** en version simplifiée. Le programme sera développé en Java et vous devrez respecter les principes de qualité et de développement objet, ainsi que les conventions syntaxiques de Java.

Cette SAÉ va être découpée en trois livrables, chacun noté indépendamment. Le rendu de chaque livrable se fera via un dépôt GIT sur la forge. L'ordre sera le suivant :

1. Développement de la partie modèle.
2. Exploration de graphs au travers de différents algorithmes.
3. Réalisation d'une interface graphique.

Au terme de cette journée, votre programme devra proposer les fonctionnalités suivantes d'un LDVEH :

— Avoir un message indiquant où se situe le joueur
— Pouvoir choisir une destination
— Pouvoir gérer un combat
— Indiquer si le joueur gagne ou perd

Votre projet doit être un projet MAVEN.

2 L'UML

Vous allez devoir reproduire l'UML de la Figure 1 afin de répondre aux besoins du projet.

Il y aura deux objectifs : créer un monde où l'on peut se déplacer, ainsi que gérer des combats entre un joueur et un monstre.

Plus de précisions dans les différentes sous-parties suivantes.

Attention, Les constructeurs, accesseurs ainsi que mutateurs ne sont pas représentés mais vous devez quand même les créer. Vous êtes aussi libre d'ajouter toute fonction ou classe vous semblant utile, l'UML ne comportant que le minimum nécessaire.

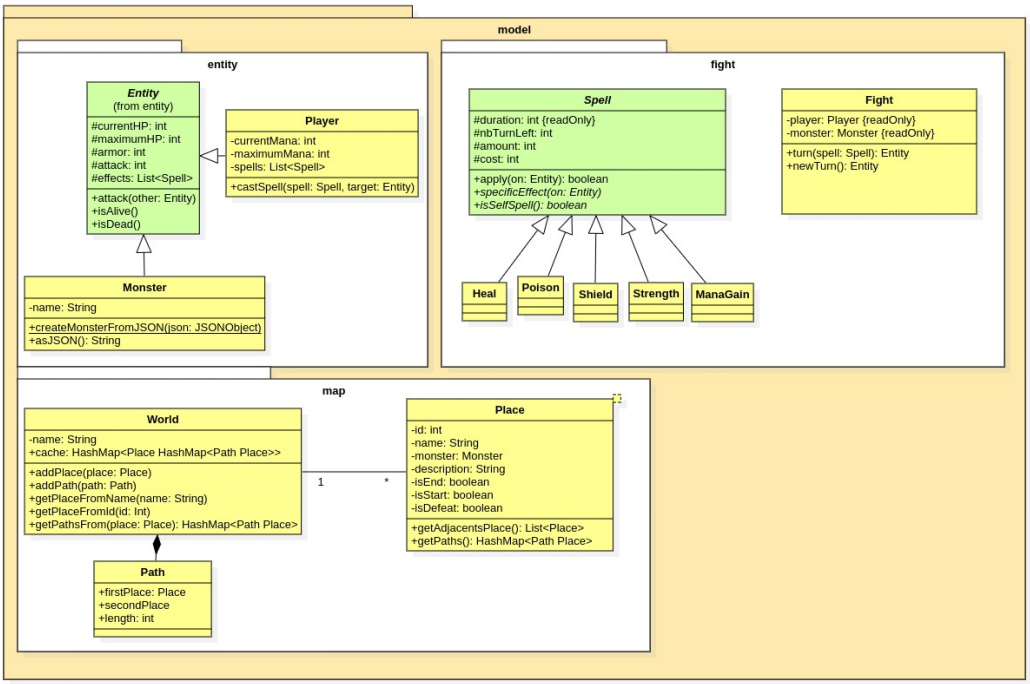

FIGURE 1 – L'UML

2.1 Le monde

Si on tente d'ajouter un `Path` dont l'un des deux lieux qu'il relit n'est pas dans le monde, une `UnknownPlaceException` doit être levée.

Chaque lieu a un nom, un texte à afficher quand le joueur arrive sur place et peut-être marqué comme lieu de départ de l'aventure, lieu de victoire de l'aventure ou lieu de défaite (par exemple, le joueur tombe dans un ravin). De plus, chaque lieu ne contient pas forcément un monstre.

Chaque lieu est relié à un ou plusieurs autres lieux par des chemins. Chaque chemin connecte uniquement deux lieux sur une certaine distance.

On doit pouvoir sauvegarder et charger facilement un monde dans un fichier JSON. (Voir section 3)

2.2 Les entités & les combats.

2.2.1 Les entités

Les monstres et le joueur sont caractérisés par un nombre de points de vie, un nombre de points de vie maximum, ainsi que des statistiques d'armure et d'attaque.

Le joueur possède en plus une statistique de mana lui permettant de lancer des sorts.

Ses sorts peuvent être soit des boosts de caractéristiques, soit des sorts appliquant un effet sur une durée limitée. Les sorts sont les suivants :

— Un soin sur le joueur
— Un poison sur le monstre
— Un sort redonnant du mana au joueur

— Un boost d'armure
— Un boost d'attaque

Vous êtes libre d'imaginer tout sort supplémentaire.

En complément, les monstres et le joueur ont une attaque de base dont les dégâts sont ATTAQUE - ARMURE. Les dêgats minimums doivent être de un.

2.2.2 Les sorts

Dans la classe `Spell`, la fonction `apply()` s'occupe de tout ce qui est commun à chaque sorts : Vérifier que la cible n'est pas null, actualiser nombre de tours restants, etc.

La fonction `applyEffect` est celle qui applique les dégâts / boosts spécifiques au sort.

2.2.3 Les combats

Un combat se déroule au tour par tour jusqu'à la mort du joueur ou du monstre.

Lors d'un combat :

— Les effets s'appliquent pour le joueur et le monstre avant le début d'un tour ;
— Le joueur joue avant le monstre à chaque tour ;
— Le monstre ne joue pas si le joueur le tue avant son tour

La fonction `newTurn` applique les effets de début de tour.

La fonction `turn` lance le sort choisit par le joueur, si **`null`** est donné, le joueur attaque avec l'attaque de base.

Ces deux fonctions renvoient le gagnant s'il y a eu un mort, **`null`** autrement.

3 Chargement et enregistrement en JSON

3.1 Sauvegarde et chargement d'un monde

Il doit être possible de lire ou d'écrire un monde depuis/vers un fichier du disque. Toute erreur à la lecture ou à l'écriture d'un fichier doit lever l'exception adéquate.

Les fichiers sont stockés au format JSON.

Le format JSON est très utile pour représenter des structures de données complexes. Il repose sur l'idée d'associer chaque élément à un couple **Clé - Valeur** que vous pouvez manipuler grâce aux `HashMap` en Java.

3.1.1 *JSONObject*

Vous allez devoir créer vos propres outils de gestion de JSON, pour cela, créez et complétez les classes *JSONObject* et *JSONArray* dont l'UML est ci-dessous. Les deux classes ont quasiment le même comportement, la seule différence est la façon dont elles stockent leurs données (*List* contre *Map*).

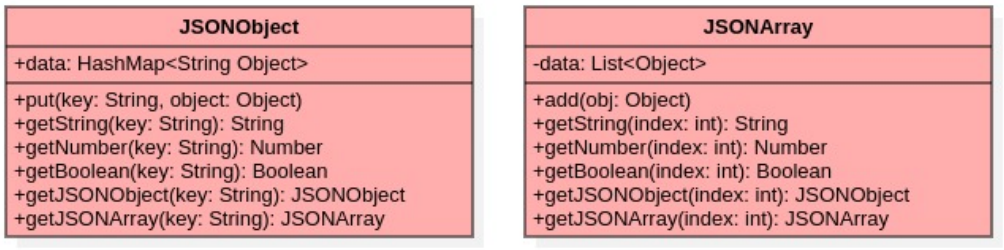

FIGURE 2 – JSONObject UML

Les accesseurs (`getNumber`, `getString`, etc.) doivent lever une `ClassCastException` si jamais l'objet associé à la clé ou à l'index donné n'est pas de la bonne classe.

3.1.2 *JSONParser*

Créez une classe `JSONParser` avec un constructeur `JSONParser(String toParse)` et une méthode `JSONObject parse()` qui permet de lire un fichier JSON.

Préambule :

Pour faciliter le parsing du fichier, il serait pratique d'éliminer tous les espaces blancs (espace, retour à la ligne, etc.) de ce dernier. Seulement, il peut y en avoir dans les chaînes de caractères et ceux-ci ne doivent pas être effacés. Pour repérer ceux inutiles, nous allons utiliser une Regex (Regular Expression). Une regex permet de repérer des motifs dans des chaînes de caractères.

Rajoutez cette ligne dans le constructeur :

```
this.toParse = toParse.replaceAll("\\s+", "");
```

Cette dernière va repérer TOUS les espaces blanc du fichier grâce à l'instruction \\s+ et les remplacer par rien, donc les supprimer. Cela revient au problème initial qui est la suppression non voulue de ceux présents dans les chaînes de caractères.

Vous allez donc devoir étoffer cette Regex pour qu'elle ne repère que les espaces blancs non présent dans une chaîne de caractère, soyez imaginatif. Le site regex101 est très utile pour tester et expliquer des Regex. Testez y la Regex fournie avec le texte du **Monde1.json** pour mettre en surbrillance les espaces repérés.

Note : La vraie Regex (très compliquée) pour repérer des espaces blancs non compris entre des guillemets est ci-dessous. Vous pouvez vous en servir mais votre regex doit quand même figurer dans le code (commentaire ou non). C'est cette dernière qui sera notée.

```
\s+(?=(?:[^"]*"[^"]*")*[^"]*)
```

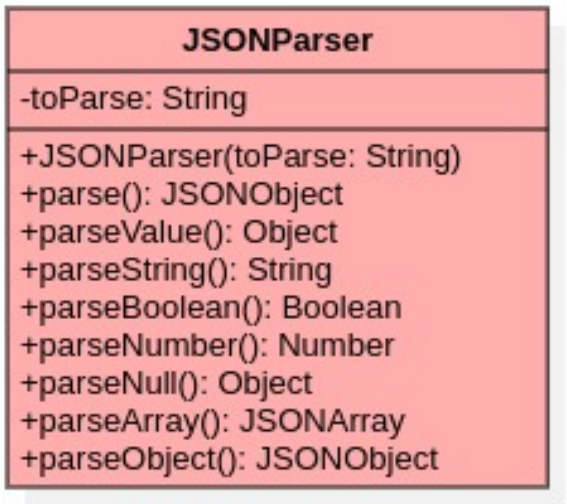

FIGURE 3 – JSONParser UML

Le parsing :

Pour parser le fichier vous allez devoir : parcourir les caractères un à un, garder en mémoire la position où vous êtes et faire un traitement particulier selon le caractère rencontré. Quelques indications :

- La méthode principale `parse` appelle la fonction `parseObject()` qui renvoie un `JSONObject`, ici l'objet du fichier JSON.
- La méthode `parseObject` avance caractère par caractère, si elle rencontre un :
  - **}** elle renvoit le `JSONObject` généré
  - **,** elle continue
  - Sinon, elle récupère la clef de l'objet avec `parseString`, ignore le caractère **:** et récupère la valeur de l'objet avec `parseValue`.
- La méthode `parseValue` s'occupe de déléguer l'appel au bon `parseTYPE` selon le caractère rencontré, si elle voit un **"** elle appelle `parseString`, un **{** elle appelle `parseObject`, un **t** ou un **f** elle appelle `parseBoolean`, etc.
- `ParseNumber` renvoie une instance de `Number` qui peut être soit un entier, soit un nombre flottant.

3.2 WorldIO

Cette classe fournit deux fonctions statiques : `saveWorld(World w, File f)` et `loadWorld(InputStream input)`.

Le schéma JSON du monde est le suivant :

World : { "world" : **string**, "places" : **array[Place]**, "paths" : **array[Path]** }

Place : { "id" : **int**, "name" : **string**, "description" : **string**, "monster" : **Monster|null**, "end" : **boolean**, "start" : **boolean**, "defeat" : **boolean** }

Monster : { "name" : **string**, "hp" : **int**, "armor" : **int**, "attack" : **int** }

Path : { "firstPlace" : **int**, "secondPlace" : **int**, "distance" : **int** }

4 L'heure de jouer !

Créez une classe `Game` dans un package `controller`. Cette classe doit fournir une fonction `play()` lancant le jeu, gérant les déplacements, les combats, etc.

Vous trouverez sur e-campus des exemples de trames de jeu possible ainsi qu'un exemple de monde possible.

5 Rendu

A l'issue de la journée, votre travail devra être déposé sur une branche `main`. Vous prendrez soin d'y avoir inclus toutes les classes demandées ainsi que toutes celles que vous auriez pu créer. Les tests unitaires doivent aussi y être déposés.

6 Notation

Le barème à titre indicatif :

— Reproduction de l'UML	
Code et documentation	5 point
tests	2 point
— La classe `WorldIO`	
Code, gestion des exceptions et documentation	1 point
tests	1 point
— JSONParser	
Code et documentation	2.5 point
tests	0.5 point
— JSONObject & JSONArray	
Code et documentation	1 points
— Jouabilité et affichage	
Jouabilité et gestion des erreurs clavier	2 points
Affichage	1 point
— **Passage de nos test unitaires**	2 points
— Travail en équipe	
utilisation de branches	1 point
commits réguliers des membres du groupe	1 point