AVIGNON UNIVERSITÉ

Rapport TP

Paul Moïse GANGBADJA

31/03/2024

Master d'Informatique
Intelligence Artificielle
UE Techniques de Test

Enseignant
BONNEFOY Ludovic

UFR
SCIENCES
TECHNOLOGIES
SANTÉ

AVIGNON UNIVERSITÉ
CERI

CENTRE
D'**ENSEIGNEMENT**
ET DE **RECHERCHE**
EN **INFORMATIQUE**
ceri.univ-avignon.fr

Sommaire

1 Introduction

Dans le cadre du TP 6 de notre projet, nous avons intégré une version de l'interface IPokemonFactory, proposée par la Team Rocket, à notre code Java. Notre objectif était d'évaluer cette nouvelle implémentation à l'aide de nos tests existants et d'une analyse de code détaillée, pour juger de sa qualité et de sa fiabilité.

Notre tâche consistait à intégrer cette implémentation à notre projet, puis à l'évaluer rigoureusement à travers nos tests unitaires et une revue de code minutieuse.

2 Implémentations RocketPokemonFactory de l'interface IPokemonFactory fournie

```java
public class RocketPokemonFactory implements IPokemonFactory {

  private static Map<Integer, String> index2name;

  static {
    Map<Integer, String> aMap = new HashMap<Integer, String>();
    aMap.put(-1, "Ash's Pikachu");
    aMap.put(0, "MISSINGNO");
    aMap.put(1, "Bulbasaur");
    //TODO : Gotta map them all !
    index2name = UnmodifiableMap.unmodifiableMap(aMap);
  }

  private static int generateRandomStat() {
    int total = 0;
    for (int i = 0; i < 1000000; i++) {
      Random rn = new Random();
      int r = rn.nextInt(2);
      total = total + r;
    }
    return total / 10000;
  }

  @Override
  public Pokemon createPokemon(int index, int cp, int hp, int dust, int candy) {
    String name;

    if (!index2name.containsKey(index)) {
      name = index2name.get(0);
    } else {
      name = index2name.get(index);
    }

    int attack;
    int defense;
    int stamina;
    double iv;
    if (index < 0) {
      attack = 1000;
      defense = 1000;
      stamina = 1000;
      iv = 0;
    } else {
      attack = RocketPokemonFactory.generateRandomStat();
```

```
45      defense = RocketPokemonFactory.generateRandomStat();
46      stamina = RocketPokemonFactory.generateRandomStat();
47      iv = 1;
48    }
49    return new Pokemon(index, name, attack, defense, stamina, cp, hp, dust, candy, iv);
50  }
51
52 }
```

3 Description de l'implémentation

L'implémentation de RocketPokemonFactory, conforme à l'interface IPokemonFactory, exploite une structure de données Map<Integer, String> pour associer chaque index de Pokémon à son nom correspondant.

Dans la création du pokemon, lorsque l'index n'est pas -1, 0 ou 1, on affecte par défaut **MISSINGNO** comme nom du Pokemon. Sinon c'est la valeur correspondant à l'index de ce Pokemon qui est utilisée.

Dans le cas où l'index fourni lors de la création est négatif, on affecte par défaut : $attack = 1000$; $defense = 1000$; $stamina = 1000$; $iv = 0$;

Dans le cas contraire, ces valeurs sont générées de façon random sauf $iv = 1$.

Bien que cette implémentation soit intéressante à cause de son niveau de généralité et du caractère aléatoire, elle présente quelque insuffisance :

- Elle autorise la création de Pokemon même quand les indexes ne sont pas dans la plage.
- Elle ne génère pas d'exception lorsque l'index n'est pas la plage de valeurs mais plutôt fixe une valeur par défaut dans ces cas, ce qui n'est pas vraiment ce qu'on attend de notre code.
- Elle ne prend pas en compte les indexes supérieurs à 150.
- Elle ne permet pas de création un Pokemon avec une valeur d'index hors ceux prédéfinies.

4 Pistes d'amélioration

Face aux insuffisances relevées dans l'implémentation de RocketPokemonFactory, il est primordial de proposer des pistes de solutions qui pourraient améliorer sa robustesse, sa flexibilité et sa conformité aux attentes initiales. Voici trois suggestions qui pourraient adresser les points soulevés :

1. **Validation Stricte des Indices de Pokémon**
   Afin de s'assurer que seuls les indices valides soient acceptés pour la création de Pokémon, il serait judicieux d'implémenter une validation stricte des indices. Cette validation pourrait être réalisée au début de la méthode createPokemon et aboutir à la génération d'une exception spécifique (par exemple, InvalidPokemonIndexException) si l'indice fourni n'est pas dans la plage autorisée (entre 1 et 150, inclus). Cela permettrait d'éviter la création de Pokémon avec des indices non prévus et améliorerait la fiabilité de l'implémentation.
2. **Extension Dynamique de la Gamme des Pokémon**
   Pour répondre au problème de la limitation à des indices prédéfinis, une solution serait de permettre l'extension dynamique de la gamme des Pokémon disponibles. Cela pourrait être réalisé en fournissant une méthode publique permettant d'ajouter de nouveaux couples indice-nom à la map index2name. Ainsi, les utilisateurs de la classe

pourraient enrichir la liste des Pokémon reconnus au-delà des valeurs initialement prédéfinies, rendant l'implémentation plus flexible.

5 Conclusion

L'examen de l'implémentation de RocketPokemonFactory révèle une approche originale mais perfectible pour la création de Pokémon. Malgré ses aspects innovants, elle présente des faiblesses notamment en matière de validation des indices et de génération des statistiques. Les pistes de solutions proposées visent à améliorer sa fiabilité et sa flexibilité, soulignant l'importance de l'évaluation continue et de l'adaptation dans le développement logiciel pour répondre aux attentes et aux standards de qualité.