title: "SAé 2.01-02-05 : RPG SIMULATOR" author: "Lucas DA SILVA FERREIRA" and "Baptiste FOURNIE" date: "08/06/2023" #"dd-mm-yyyy" format

SAé 2.01-02-05 : RPG SIMULATOR

Sommaire

Introduction

La SAé 2.01-02-05 est un projet de réalisation d'une application visant à simuler le comportement d'un joueur dans un scénario de RPG. Lors de ce projet nous avons tout d'abord réaliser toute la partie Gestion et Organisation de ce dernier. On entend bien ici que ce projet est un projet transversale d'une part qu'elle est l'ensemble de 3 SAés et regroupant plusieurs ressources d'enseignements de ce Semestre 2.

Le "RPG SIMULATOR" est une application de simulation de RPG qui permet de simuler dans un ou plusieurs contexte de scénario, un ou plusieurs parcours avec différent comportements selon la sélection de l'utilisateur.

Les scénarios fournies vont du "scénario_0.txt" au "scenario_10.txt", l'application possède différents paramètrages dont celle de choisir entre des parcours efficaces ou exhaustives, des parcours en terme de durée, de nombre de quêtes ou bien de déplacements. On peux choisir également entre les pires ou les meilleures solutions selon les choix précédents.

Qualité de Développement

Cycle de vie du projet :

Nous avons décider de réaliser ce projet dans cycle en V récursif, dans l'intêret d'avoir le plus agilité sur notre projet. Car la conception était pas sûre et le cycle en V récursif nous permettait de tester et réaliser le code méthode par méthode et si besoin, réaliser une modification du même nom mais en "VERSION 2.0".

Repartition des tâches :

Le projet à été réaliser par les deux membre du groupes, la base du programme telle ce qui défini les Quetes "Quete.java", les Scénarios "Scenario.java", la lecture des Scénarios "lectureFichierTexte.java", le Classement "Classement.java" et aussi la partie Interface Homme-Machine (IHM) ont été réalisés par les deux membres. Puis les Algorithmes ont été réalisés par Baptiste FOURNIE. Les tests en générale ont été réalisés par Lucas DA SILVA FERREIRA. La rédaction du dossier de test à été rédiger pour la quasi totalité par Lucas DA SILVA FERREIRA.

Les commits sont visibles comme étant à 100% fait par Lucas DA SILVA FERREIRA, mais en réalités due a l'utilisation de la fonction "Code with me" de IntelliJ, les commits sont fait par Lucas DA SILVA FERREIRA (hôte de la session CODE WITH ME) mais pas forcément écris par ce dernier.

Outils utilisés :

Le projet à necessité plusieurs outils et fonctionnalités, telles que :

- JUnit 5 : Pour l'implementation des tests.
- IntelliJ : Pour sa fonctionnalité "Code With Me" très utile pour développer sur le même projet en temps réel.
- StarUML : Pour la réalisation des Diagrammes de classes.
- Visual Studio Code : Pour la réalisation des Markdowns et permettre que les changements soit visibles puis implémentation de chaque changement par copier/collés sur IntelliJ.
- GitHub : Pour le versionnage du projet.

Méthode de développement :

Nous avons préféré pour une meilleure agilité, de conduire notre projet sur une méthode de développement agiles et qu'elle soit piloté par les tests réalisés AVANT la progammation du code devant répondre au besoin. Cela permet d'adopter une agilité pratique dans l'eventualité de proceder à des changements en cas de problèmes et de pas perdre en tête la spécification à répondre au cours du projet.

Définitions liés à la "Méthode de développement" :

Développement piloté par les tests (Test-Driven Development, TDD) : Le TDD est une approche de développement qui consiste à écrire des tests automatisés avant d'écrire le code. Cette méthode met l'accent sur la qualité du code en veillant à ce qu'il réponde aux exigences spécifiées par les tests. Le TDD favorise une meilleure couverture de test et une réduction des erreurs introduites lors de l'écriture du code.

Méthodes agiles : Les méthodes agiles, telles que Scrum ou Kanban, mettent l'accent sur la collaboration, la flexibilité et l'itération rapide. Elles encouragent la communication régulière entre les membres de l'équipe de développement, la livraison fréquente de fonctionnalités fonctionnelles et l'adaptation aux changements. Les méthodes agiles favorisent une approche itérative de développement qui permet de détecter et de résoudre les problèmes de qualité plus tôt dans le processus.

Conception Générale

Diagramme de Haut Niveau :

Le diagramme de haut niveau permet de voir toutes les classes mais sans leur attributs, ni leurs méthodes.

UML DE MODELE/VUE/CONTROLEUR :

on peut voir que UML Modele/vue/controleur respecte bien l'architecture du modéle Mcv. Aucue classe de modéle ne connait une classe de package vue ou controleur(les classes modéles ne possédent aucun attribut de cotroleur ou de vue). Par contre la vue Connait le modéle et le controleur. La VboxRoot est la racine de la vue donc elle posséde en attribut static toutes les classes de vue (à part Formulaire Exception et l'interface). A part menuBarres toutes les autres classes de vue possédent en attributs des classes de modéles. Presques toutes les classes héritent d'une classe de base de JavaFx(VBoxRoot héritent de VBOx, Formulaire de gridPane etc...)

Diagramme de Bas Niveau :

Le diagramme de bas niveau permet de voir toutes les classes avec leur attributs et leurs méthodes.

LES CLASSES DU MODELE :

On peut voir que le diagramme de classe du package "modéle" posséde 9 classes et une interface. On peut rajouter que Classement Exception est throws dans Classement (meme chose pour launcher exception avec la classe launcher=. On remarque Parcours et Quete posséde l'interface comparble car les deux seront mis dans des treeSet ou TreeMap.

LES CLASSES DE LA VUE :

On peut également voir que le diagramme de classe du package "vue", On voit que VboxRoot est la classe majeure de rue . Elle posséde des attributs de toutes les autres classes de vue. RPG Application peremet de lancer l'application.

LES CLASSES DU CONTROLEUR :

Pour finir on voit que le diagramme de classe du package "controlleur", Elle hérite de EventHander et ne posséde qu'une méthode handle qui permet à l'utilisateur d'intéragir avec utilisateur. Elle ne posséde pas d'attribut.

Structure de données et stratégies algorithmiques :

A travers plusieurs extraits d'algorithmes nous pouvons vous montrer que nos méthodes algorithmiques sont des solutions brutes. En effet on parle de solution brute car elle sont concu pour un type de contexte (Efficace ou Exhaustif) et un type d'objectif (durée, nombre de quêtes, déplacements)

- Voici quelques extrait de la solution Gloutonne version Efficace et version Exhaustive en exemple :

LA CLASSE ALGORITHMIQUE "recursiviteGloutonneEfficace()" :

```
 java public static void recursiviteGloutonneEfficace(Scenario parScenario,ArrayList
listparcours){ Parcours parparcours = listparcours.get(listparcours.size() -1); //regarde si la
quete 0 est possible if ((parparcours.queteFinPossibleEfficace())){ // rajoute la quête et sa
durée parparcours.ajouteDuree(parparcours.getQueteFin());
parparcours.ajouteQueteFaite(parparcours.getQueteFin()); // rajoute le parcours finis au
classement Classement.ajout(parparcours); } else{ Quete QueteActuelle =
parparcours.getQueteActuelle(); //regarde les quetes possibles parparcours.quetesPossibles();
//prend les quetes les plus proches de queteActuelle HashSet ensQueteProche=
QueteActuelle.queteProche(parparcours.getQuetePossible()); for (Quete q: ensQueteProche){
parparcours.ajouteDuree(q); parparcours.ajoutexp(q.getExperience());
parparcours.ajouteQueteFaite(q);
Algorithme.recursiviteGloutonneEfficace(parScenario,listparcours); parparcours =
listparcours.get(listparcours.size() - 1); // crée un nouveau parcours sur la base du parcours
actuelle listparcours.add(new Parcours(parScenario, parparcours.getChexp(),
parparcours.getduree(),0, "duree", parparcours.getQuetesFaite(),
parparcours.getQuetesNonFaite(), new HashSet<>())); // enléve dans liste parcours le parcours
inutile pour la récursivité listparcours.remove(listparcours.size()-2); parparcours =
listparcours.get(listparcours.size() - 1); // regarde si le parcours a fait la quete 0 pour
l'enlever if(parparcours.getQuetesFaite().containsKey(0)) {
parparcours.enleverQueteFaite(parparcours.getQueteFin());
parparcours.enleverDuree(parparcours.getQueteFin()); } // enleve la quete q avec sa durée et
son expérience parparcours.enleverQueteFaite(q); parparcours.enleverDuree(q);
parparcours.ajoutexp(-q.getExperience()); } } }
```

LA CLASSE ALGORITHMIQUE "recursiviteGlutonneExhaustive()" :

```
 java /**
```

- permet de déterminer le chemin le plus court en faisant toutes les quêtes
- @param parScenario (Scenario): scenario qu'utilise les parcours
- @param listparcours (ArrayList Parcours): liste chronologique des parcours ajoutés */

```
  public static void recursiviteGlutonneExhaustive(Scenario parScenario,ArrayList
  listparcours){ Parcours parparcours = listparcours.get(listparcours.size() -1); //regarde
  si la quete 0 est possible if ((parparcours.queteFinPossibleExhaustive())){ // rajoute la
  quête et sa durée parparcours.ajouteDuree(parparcours.getQueteFin());
  parparcours.ajouteQueteFaite(parparcours.getQueteFin()); // rajoute le parcours finis au
  classement Classement.ajout(parparcours); } else{ Quete QueteActuelle =
  parparcours.getQueteActuelle(); //regarde les quetes possibles
  parparcours.quetesPossibles(); //prend les quetes les plus proches de queteActuelle HashSet
  ensQueteProche= QueteActuelle.queteProche(parparcours.getQuetePossible()); for (Quete q:
```

```
ensQueteProche){ parparcours.ajouteDuree(q); parparcours.ajoutexp(q.getExperience());
parparcours.ajouteQueteFaite(q);
Algorithme.recursiviteGlutonneExhaustive(parScenario,listparcours); parparcours =
listparcours.get(listparcours.size() - 1); // crée un nouveau parcours sur la base du
parcours actuelle listparcours.add(new Parcours(parScenario, parparcours.getChexp(),
parparcours.getduree(),0, "duree", parparcours.getQuetesFaite(),
parparcours.getQuetesNonFaite(), new HashSet<>())); // enléve dans liste parcours le
parcours inutile pour la récursivité listparcours.remove(listparcours.size()-2);
parparcours = listparcours.get(listparcours.size() - 1); // regarde si le parcours a fait
la quete 0 pour l'enlever if(parparcours.getQuetesFaite().containsKey(0)) {
parparcours.enleverQueteFaite(parparcours.getQueteFin());
parparcours.enleverDuree(parparcours.getQueteFin()); } // enleve la quete q avec sa durée
et son expérience parparcours.enleverQueteFaite(q); parparcours.enleverDuree(q);
parparcours.ajoutexp(-q.getExperience()); } } }
```

On voit ici qu'il y a deux contextes, l'efficace et l'exhaustive, et que la solution gloutonne est le contexte dans lequel le joueur fait tout les quetes les plus proches. Il y a une méthode pour une paire (contexte/objectif). ici respectivement (efficace/glouton) et (exhaustif/glouton).

Cette méthodologie d'algorithmie (efficace/exhaustif) est faite pour tout les objectifs combiné par un choix entre le pire ou les meilleurs solutions.

Conclusion Générale

Le projet aurait nécessité éventuellement l'ajout d'un css pour la présentation de l'application, ont à néanmoins espacés les éléments mais ce projet mériterait d'être plus approfondit sur ca présentation.

L'algorithme de djakista en itératif aurait aussi pu être un point à appronfondir également, il aurait été un moyen d'avoir une solution du scénario 10 en un temps record.

Annexes

Vous trouverez **ici**, le Git du projet "SAE_2.01-02-05_RPG-SIMULATOR".

Vous trouverez **ici**, le dossier de test "DOSSIER_TEST.md". Pour des raisons de légèreté du rapport et que le dossier de TEST possède l'équivalent de 35 pages PDF, on a préfére disposer ce lien.

Merci d'avoir lu ce rapport de projet et de voir ce projet finalisé!

Ce projet à été réaliser par Lucas DA SILVA FERREIRA et Baptiste FOURNIE. Deux étudiants de l'Institut Universitaire de Technologie de Velizy. (2022-2023), le 08/06/2023 à l'IUT de Vélizy.