Programmation en Java — Projet

L3 Informatique – 1er semestre – automne 2021 – Version 1, en date du 12 octobre 2021

Sujet : Réaliser le jeu Splendor en Java

Le but de ce projet est de réaliser une version PC offline d'un jeu de société : **Splendor**. Il s'agit d'un jeu de stratégie où plusieurs joueurs s'affrontent pour asseoir leur contrôle sur le commerce des pierres précieuses.

Le jeu

La règle détaillée du jeu Splendor est disponible **en ligne**. Nous vous invitons à la lire avec attention, puisqu'il s'agit du jeu que vous allez devoir programmer.

Conseils

- ▷ Lisez l'ensemble de l'énoncé avant même de commencer quoi que ce soit !
- ▷ Le but de ce projet est de nous montrer que vous savez programmer "objet". Vous serez donc pénalisés si vous n'exploitez pas suffisamment les notions vues en cours.
- ▷ Lisez bien les règles du jeu et prenez bien le temps de réfléchir à la meilleure façon de mettre en place tel ou tel mécanisme du jeu avant de commencer à coder. Si vous commencez à coder avant d'avoir suffisamment réfléchi, vous serez, en pratique, obligés de revenir en arrière, ce qui vous demandera beaucoup plus de temps et d'efforts.
- ▷ Le sujet est évolutif : respectez bien les phases de réalisation, mais gardez à l'esprit ce que vous devrez faire dans les phases suivantes lorsque vous faîtes des choix d'implantation !

Le programme à réaliser

Vous allez réaliser votre jeu en 4 phases. Chaque phase devra être terminée et fonctionnelle avant de passer à la phase suivante.

Phase 1 : La base

Dans un premier temps, vous devez réaliser une version simplifiée du jeu, avec uniquement un mode de jeu permettant à deux joueurs humains de jouer l'un contre l'autre, avec un affichage **en ligne de commande**. Dans cette version simplifiée,

- ▷ on a supprimé les nobles : les cartes sont donc le seul moyen d'acquérir des points de prestige ;
- ▷ on effectue l'affichage en ligne de commande ;
- ▷ on a supprimé la possibilité de réserver une carte, et donc les jetons joker or ;
- ▷ il y a un seul niveau de cartes : chaque carte coûte trois jetons de sa couleur et rapporte un point de prestige, et on a huit cartes de chaque couleur.

Phase 2 : Le jeu complet

Une fois la phase 1 terminée, vous devez réaliser le jeu complet, permettant de faire jouer de deux à quatre joueurs humains, et toujours un affichage **en ligne de commande** :

- ▷ la liste complète des $40 + 30 + 20$ cartes du jeu est disponible **ici** ;
- ▷ la liste complète des 10 nobles présents dans le jeu est disponible **ici**.

Vous devrez donc trouver une manière adaptée de laisser à l'utilisateur le choix entre la version de base du jeu et la version complète.

Phase 3 : Affichage graphique

Une fois la phase 2 terminée, il vous est demandé de mettre en place une interface graphique simple. Vous devrez trouver une manière adaptée de laisser à l'utilisateur le choix entre l'interface graphique et l'interface en ligne de commande.
Afin de réaliser cette interface graphique, vous devrez utiliser la bibliothèque d'interface graphique `zen` fournie avec ce sujet (fichier `zen5.jar`).
Pour ajouter un jar à un projet sous Eclipse, il faut :

- ▷ Rajouter un dossier `lib` dans le répertoire du projet et y placer le fichier `.jar`.
- ▷ Dans Eclipse, faire un clic droit sur le fichier `.jar` et choisir `Build Path > Add to Build Path`.

Phase 4 : Améliorations

Une fois la phase 3 terminée, deux améliorations vous seront demandées :

- ▷ ajouter la présence de joueurs simulés par l'ordinateur : nous laissons à chaque groupe le soin de mettre au point des stratégies pour les joueurs simulés ;
- ▷ proposer une extension des règles du jeu ; vous devrez trouver une manière adaptée de laisser à l'utilisateur le choix entre la version basique du jeu (issue de la phase 1), le jeu complet (issu des phases 2 et 3) et l'option étendue.

Toute autre amélioration sera considérée négativement tant que les améliorations ci-dessus n'ont pas toutes été mises en place.

Consignes de rendu

- ▷ Ce projet est à faire en **binôme** (c'est-à-dire exactement **deux** personnes). Vous aurez jusqu'au dimanche **14 novembre** à **23:59** pour indiquer, sur e-learning, le nom de votre partenaire de projet. Si vous avez des difficultés à trouver un partenaire de projet, approchez-vous sans attendre de votre chargé de TP, pour qu'il vous aide à ce propos.

  Si vous n'avez pas indiqué votre choix de partenaire à la date demandée

  - ▷ et si vous n'avez pas non plus participé au TP noté du lundi 8 novembre, vous aurez d'office 0 au projet ;
  - ▷ et si vous avez participé au TP noté du lundi 8 novembre, un partenaire de projet vous sera attribué d'office, de manière arbitraire.

- ▷ Une soutenance β sera organisée durant l'après-midi du mardi **14 décembre**. Pendant cette soutenance, vous ferez une démonstration sur une machine des salles de TP et serez interrogés sur le projet. Le but de cette soutenance est, entre autres, de permettre à votre enseignant de vous encourager si vous êtes sur la bonne voie, et de vous remettre dans le droit chemin sinon.

- ▷ Un premier travail intermédiaire, qui sera présenté lors de la soutenance β, doit être déposé sur e-learning avant le dimanche **12 décembre** à **23:59**. Il est attendu, à cette étape du projet, que les phases 1 et 2 soient terminées.
- ▷ La date limite de rendu final est le dimanche **23 janvier** à **23:59**. Passé ce délai, la zone de rendu sera fermée, et nous ne prendrons pas en compte un éventuel rendu qui nous serait envoyé autrement (par exemple par mail). En revanche, il est possible de déposer plusieurs fois votre rendu dans le temps imparti ; seule la dernière version sera alors prise en compte.

Rendu intermédiaire

Votre rendu intermédiaire devra consister en une archive au format `zip` : tout `rar`, `tar.gz`, `7z` ou autre ne sera pas ouvert. Cette archive contiendra :

- ▷ un répertoire `src` contenant les sources du projet ;
- ▷ un répertoire `docs` contenant un manuel de l'utilisateur (`user.pdf`) et un manuel expliquant l'architecture que vous avez choisie (`dev.pdf`) ; le manuel `user.pdf` doit être lisible par Rémi Forax Junior, 9 ans, qui connaît les règles de Splendor et essaiera d'utiliser votre programme sans avoir lu d'autre document, ni même l'énoncé du projet ;
- ▷ un répertoire `docs/doc` contenant la javadoc, écrite **en anglais** ;
- ▷ un répertoire `lib` contenant les librairies dont dépend l'application ;
- ▷ un `jar` exécutable `Splendor.jar`, qui fonctionne avec la commande `java -jar Splendor.jar`.

Cette archive aura pour nom `Nom1_Nom2_Splendor.zip`, où les noms sont ceux des membres du binôme par ordre alphabétique. L'extraction de cette archive devra créer un répertoire intitulé `Nom1_Nom2_Splendor` et contenant tous les éléments demandés ci-dessus.

Rendu final

Votre rendu final devra consister en une archive au format `zip` : tout `rar`, `tar.gz`, `7z` ou autre ne sera pas ouvert. Cette archive contiendra :

- ▷ un répertoire `src` contenant les sources du projet ;
- ▷ un répertoire `docs` contenant un manuel de l'utilisateur (`user.pdf`) et un manuel expliquant l'architecture que vous avez choisie (`dev.pdf`) ; le manuel `user.pdf` doit être lisible par Rémi Forax Junior, 9 ans, qui connaît les règles de Splendor et essaiera d'utiliser votre programme sans avoir lu d'autre document, ni même l'énoncé du projet ; le manuel `dev.pdf` doit notamment inclure une section dédiée aux améliorations et corrections apportées depuis la soutenance β ;
- ▷ un répertoire `classes`, vide dans l'archive, et qui contiendra les classes une fois compilées ;
- ▷ un répertoire `lib` contenant les librairies dont dépend l'application ;
- ▷ un `jar` exécutable `Splendor.jar`, qui fonctionne avec la commande `java -jar Splendor.jar`, et qui possède donc un fichier `manifest` adéquat ;
- ▷ un fichier `build.xml`, écrit à la main, qui permet de
  - ▷ compiler des sources (`target compile`) ;
  - ▷ créer le `jar` exécutable (`target jar`) ; il devra s'agir de la `target` par défaut ;
  - ▷ générer la javadoc, écrite **en anglais**, dans le répertoire `docs/doc` (`target javadoc`) ;
  - ▷ nettoyer le projet pour qu'il ne rete plus que les éléments demandés (`target clean`).

Cette archive aura pour nom `Nom1_Nom2_Splendor.zip`, où les noms sont ceux des membres du binôme par ordre alphabétique. L'extraction de cette archive devra créer un répertoire intitulé `Nom1_Nom2_Splendor` et contenant tous les éléments demandés ci-dessus.

Critères de notation

- ▷ Rémi Forax Junior, 9 ans, doit pouvoir jouer à votre jeu ;
- ▷ la propreté et la lisibilité du code auront un poids très important dans la note ;
- ▷ l'architecture que vous aurez définie (interfaces, classes, etc) devra être donnée dans les documents PDF et aura également un poids très important dans la note ; ainsi, votre code devra être modulable, de manière à ce qu'ajouter d'autres extensions (par exemple davantage de cartes, un niveau de plus, ...) soit aussi facile que possible ;
- ▷ votre code ne devra pas contenir de méthodes de plus de 20 lignes ;
- ▷ pas de duplication de code, et respect des principes de programmation objet ;
- ▷ pas de variable globale;
- ▷ pas de code inutile ;
- ▷ présence des différents rapports et, par conséquent, orthographe correcte !
- ▷ prise en considération des remarques faites lors de la soutenance β pour le rendu final.

Règles à respecter impérativement – Mort subite

Voici une liste de règles qu'il vous faudra respecter impérativement. Si vous ne respectez pas ne serait-ce qu'une seule de ces règles, la notre de votre projet sera 0, et celui-ci ne sera pas évalué.

- ▷ Vous **devez** participer à la soutenance β le **14 décembre** ; si un seul des deux membres d'un binôme participe à la soutenance β, le membre absent aura 0.
- ▷ Vous **devez** déposer un premier travail **sur e-learning** avant le **12 décembre** à **23:59**. Un projet envoyé par mail et/ou après la date butoir recevra la note de 0.
- ▷ Vous **devez** déposer votre version finale **sur e-learning** avant le **23 janvier** à **23:59**. Un projet envoyé par mail et/ou après la date butoir recevra la note de 0.
- ▷ Dans les deux cas (version intermédiaire et version finale), votre code **doit** compiler.
- ▷ Vous **devez** inclure une javadoc écrite **en anglais** et des fichiers `user.pdf` et `dev.pdf` avec votre version finale. Un projet qui omettrait ne serait-ce qu'un seul de ces trois éléments recevra la note de 0.
- ▷ Votre archive **devra** avoir le bon nom, être une archive `.zip`, et produire un répertoire qui a le bon nom.
- ▷ Le projet ne **devra pas** utiliser ou inclure de libraire externe autre que celles indiquées dans le sujet.
- ▷ Le projet ne **devra pas** contenir de code copié-collé du net. La présence d'un tel code sera interprétée comme une tentative de tricherie.
- ▷ Le projet ne **devra pas** utiliser de classes du package `java.io` autres que les classes `InputStream`/`OutputStream`, `BufferedReader`/`BufferedWriter` et l'exception `IOException`. En particulier, il ne **devra surtout pas** utiliser `java.io.File`.
- ▷ Le projet ne **devra pas** contenir de champ avec une visibilité autre que `private`, et toute méthode de visibilité `public` devra commencer par vérifier que ses arguments sont raisonnables. Par exemple, si une fonction lance une `Exceptionnaliste`, celle-ci doit être due à un appel à `Objects.requireNonNull` qui aura détecté que l'argument proposé était nul.

Références

1. **Ant Manual** pour la construction du fichier `build.xml`
2. **How to create an executable jar?**
3. **JavaDoc**
4. **Les entrées/sorties sur fichier**
5. La bibliothèque graphique **Zen 5** et sa **documentation**
6. Un **exemple de code** utilisant le modèle de développement Modèle-Vue-Contrôleur, mais qui ne respecte pas plusieurs des règles de mort subite (et vaut donc 0) : javadoc manquante pour quelques classes ou méthodes publiques, absence de contrôle des arguments dans certaines méthodes publiques, ...