Projet - *Aventuriers du Rail (Europe)*

IUT Montpellier-Sète – Département Informatique

Ce projet a lieu dans le cadre des Situations d'Apprentissage et Évaluation du BUT Informatique (SAE S2.01 et S2.02). Il fera intervenir plusieurs compétences acquises durant le Semestre 2 : le développement orienté objets, les notions de qualité de développement (tests unitaires, gestion de version, etc), les algorithmes de graphes et les interfaces homme-machine.

Dans la **Phase 3** du projet, vous allez programmer une interface graphique en *JavaFX* pour la mécanique du jeu *Aventuriers du Rail (Europe)* que vous avez programmé durant la **Phase 1**. Notez que la **Phase 3** se déroulera en parallèle de la **Phase 2** du projet (voir le cours de *Graphes* pour plus de détails).

Organisation du travail

- **Phase 3 : *implémentation d'une interface graphique sous JavaFX***
  - **Cours concerné** : *Développement d'applications avec IHM*
  - **Support de cours**
- **Enseignants:**
  - Sophie Nabitz, Cyrille Nadal, Nathalie Palleja, Xavier Palleja, Petru Valicov
  - Nihal Ouherrou – responsable de la partie *Ergonomie* du cours
- **Période de réalisation :** 22 avril - 5 juin 2022
- Des séances de TP (ergonomie et programmation *JavaFX*) seront dédiées intégralement au projet après les vacances de Pâques.

Pour des questions : * Le forum Piazza - à privilégier lorsque vous avez des questions sur le projet. * Email pour une question d'ordre privée concernant le projet.

Consignes générales

Un squelette du code vous est fourni et vous devrez : * Écrire les classes qui vous sont décrites, en ajoutant les fonctions et attributs qui vous semblent utiles. * Vérifier que votre interface graphique réagit comme attendu. Des tests unitaires de cette interface ne sont pas demandés. * Sauf indication explicite de la part des enseignants, ne pas modifier les méthodes et classes de la logique du jeu qui vous sont données.

Des consignes détaillées du projet vous sont données dans le fichier Consignes.md. ***Le non-respect de ces consignes impliquera une pénalité de plusieurs points sur la note finale du projet.***

Présentation

Vous connaissez déjà les règles, mais si besoin, vous pouvez les retrouver dans le repertoire `documents`.

Vous partirez d'une version du moteur déjà implémentée (la logique de l'application), qu'il ne sera pas nécessaire de modifier. Cette partie, est en fait une adaptation de la correction de la Phase 1 du projet. Vous considèrerez donc cette partie comme totalement encapsulée, non invocable directement, et vous la ferez exécuter en utilisant des interfaces *Java* jouant un rôle de façades permettant de communiquer avec la logique interne. Les méthodes de ces interfaces sont sommairement commentées, vous devez vous y référer (en complément du présent document) afin d'identifier laquelle utiliser dans quelle situation.

Vous allez implémenter plusieurs classes (certaines méthodes vous sont suggérées), dans lesquelles vous allez ajouter tout ce qui sera utile à votre IHM.

Architecture générale du code

Les classes sont exposées dans un diagramme de classes simplifié de l'application :

Pensez à le consulter en détails pour mieux vous situer.

Ce modèle ne détaille pas les classes et relations de la partie logique, ni ne liste les méthodes des classes que vous utiliserez. Pour ne pas surcharger le diagramme, les attributs et les fonctions ne sont pas présentés. Vous y voyez surtout les classes qui sont nécessaires à votre travail.

Le package `rails`, en gris sur le diagramme, représente la logique interne du jeu. Vous y accédez en utilisant les interfaces en bleu, dont le nom (qui commence par un `I` majuscule) correspond aux classes d'implémentation. En lisant ces interfaces "façades", vous verrez que de nombreuses propriétés sont déclarées, vous devez les utiliser, dans vos vues (package `vues`), pour définir vos écouteurs d'événements et de changements qui mettront à jour votre IHM.

La partie qui vous intéresse se trouve dans le package `vues`. Ce package contient les classes d'IHM à implémenter (en vert sur le diagramme), que vous pouvez renommer si vous le souhaitez. Dans la suite de ce document, ces classes seront désignées par `VueXXX`.

Organisation interne de l'application

La classe `RailsIHM` (en rose sur le diagramme de classes) vous est fournie. Elle représente l'`Application` JavaFX et effectue essentiellement plusieurs tâches : - Elle crée un thread (`ServiceDuJeu`) sur lequel tournera la logique interne du jeu et qui s'exécutera en parallèle de celui de l'interface graphique (`JavaFx Application Thread`). - Elle crée la scène en la construisant avec la première

vue (objet de la classe `VueDuJeu`). - Elle peut, si souhaité, démarrer l'interface graphique sur une fenêtre permettant de saisir le nombre et les noms des joueurs de la partie (voir le paragraphe correspondant plus loin).

Organisation interne des interfaces d'accès à la logique du jeu

Dans les interfaces qui permettent d'accéder à la logique interne du jeu, vous constaterez que sont d'abord présentées les propriétés du jeu que vous pouvez écouter. Ensuite certains getters et enfin les méthodes que vous pouvez invoquer pour activer la logique interne. A priori, les propriétés essentielles pour la mise en place de l'interface graphique sont exportées.

Description sommaire des différentes vues

- La classe `VueDuJeu`, première vue de l'application, présente la vue générale de l'interface graphique. Celle qui vous est donnée ne fait qu'afficher le plateau du jeu, vous allez donc la transformer pour intégrer d'autres éléments, correspondant aux classes décrites ci-dessous.
- La classe `VueDuJoueurCourant` présente les éléments appartenant au joueur courant, actualisés à chaque changement de joueur.
- La classe `VueAutresJoueurs` présente les éléments des joueurs autres que le joueur courant, en cachant ceux que le joueur courant n'a pas à connaitre.
- La classe `VuePlateau` présente les routes et les villes sur le plateau, avec lesquelles le joueur pourra interagir afin de jouer son action.
- La classe `VueCarteWagon` représente la vue d'une unique carte Wagon et la classe `VueDestination` représente la vue d'une unique carte Destination.
- La classe `VueChoixJoueurs`, qui peut éventuellement être affichée en début de partie, permet de choisir le nombre et les noms des joueurs de la partie. En consultant les fichiers de ces classes, vous prendrez connaissance ces éléments essentiels à implémenter, auxquels vous avez toute liberté d'ajouter ce qui vous convient.

Quelques informations techniques

Accès au jeu

Dans votre code, pour déclencher les traitements du jeu à la suite des actions de l'utilisateur sur l'interface graphique, vous devez passer par l'interface `IJeu`. Par exemple, lorsque l'utilisateur choisit de passer, en cliquant sur un élément graphique qui correspond à "*Passer*", il faut appeler la méthode `passerAEteChoisi` de l'interface `IJeu`. La vue principale du jeu (`VueDuJeu`) est construite avec une référence sur le jeu. Par conséquent, si l'on souhaite accéder à cette référence `IJeu` depuis d'autres vues, alors on peut le faire à travers la scène de la vue principale (méthode `getScene()` définie pour tous les `Node`), et

le cas échéant, à la racine de cette scène (`getRoot()` de la classe `Scene`). Ainsi, par exemple le code suivant `((VueDuJeu) getScene().getRoot()).getJeu()` permet d'accéder au jeu si la vue du jeu est la racine de la scène. Oui, il y a un cast, mais là il est nécessaire ! :smirk:

Ajout de propriétés

Si vous le jugez nécessaire, vous pouvez ajouter d'autres propriétés que celles proposées dans les interfaces, afin de les écouter pour actualiser votre interface graphique. Pour cela, vous pouvez transformer en propriétés certains attributs basiques des classes de la logique du jeu. Dans ce cas, vous n'oublierez pas de définir les 3 méthodes setter, getter et accès à la propriété. Le code interne peut alors être amené à changer : toute affectation de la propriété devra alors passer par la méthode `setValue(..)` et l'accès à la valeur de propriété par la méthode `getValue()`.

Exécution des écouteurs de changement sur le thread de l'interface graphique

La mise à jour des propriétés se fait dans la partie interne du jeu : par exemple, le changement du joueur courant se passe en interne dans le jeu. En conséquence d'un tel changement, les composants graphiques vont s'actualiser : dans le même exemple, les cartes du nouveau joueur courant vont être présentées. Cette mise à jour graphique doit **obligatoirement** se faire sur le thread de l'interface graphique. Comme le thread de l'interface graphique tourne en parallèle de l'exécution de la partie interne, il traite à son rythme les tâches qui lui sont affectées. Il faut donc ajouter les tâches nouvelles dans sa file d'attente. Par conséquent, à chaque début du code d'un gestionnaire de changement d'une propriété, vous ajouterez le code suivant :

```
Platform.runLater( () -> {
    // ici le code de mise à jour de l'interface graphique
}
```

L'instruction ci-dessus permet d'exécuter le code sur le thread `JavaFx Application Thread`.

Utilisation de la classe `Pane`

Les vues qui sont proposées héritent actuellement de la classe `Pane`. Vous allez probablement changer cette classe de base afin d'utiliser un composant qui vous parait plus approprié.

Si toutefois vous préférez continuer à utiliser cette classe (le fait de conserver `Pane` en classe de base ne limite pas vos choix de conception de l'IHM), vous devez créer les composants graphiques et les ajouter à la liste des enfants avec `getChildren()` de `Pane`. Et inversement, si vous utilisez un autre conteneur comme classe de base, il faudra ne plus utiliser cette méthode `getChildren()`.

Ajout d'une fenêtre initiale pour le choix des joueurs

Dans un second temps, lorsque votre vue du jeu sera finalisée, vous pourrez travailler sur une autre fenêtre de départ, qui permet de saisir le nombre et les noms des joueurs.

L'application est actuellement organisée pour lancer la vue principale du jeu. Lorsque vous souhaitez lancer, au départ, cette autre fenêtre, il faudra changer la valeur du booleén `avecVueChoixJoueurs` déclaré comme attribut dans `RailsIHM`. Ceci aura pour effet de provoquer l'instanciation d'un objet de la classe `VueChoixJoueurs`. Lorsque la saisie des noms des joueurs est terminée, vous ferez en sorte que la mise à jour de la liste observable `nomsJoueurs` (qui est faite à la fin de la méthode `setListeDesNomsDeJoueurs()`) déclenche le listener `quandLesNomsJoueursSontDefinis` de `RailIHM`, qui démarre la partie.

La vue du plateau

Cette vue vous est en grande partie fournie. Elle charge les composants du plateau à partir du fichier `plateau.fxml` du répertoire `resources/fxml`. La classe `VuePlateau` correspond au contrôleur du *fxml*. Pour chaque élément du plateau (cercle ou rectangle), un gestionnaire d'événement de click de souris invoque la fonction `choixRouteOuVille()`. Cette fonction devra utiliser la fonction `uneVilleOuUneRouteAEteChoisie(String)` du jeu pour appeler le bon traitement dans la logique du jeu.

Les instructions pour l'utilisateur

La propriété `instruction` qui est proposée dans l'interface `IJeu` représente l'information à transmettre à l'utilisateur pour qu'il puisse choisir sa prochaine action. Elle est automatiquement mise à jour dans la logique interne du jeu, et vous pouvez vous en servir dans votre interface pour afficher le message approprié.

Rendu attendu

L'intégralité du code source du projet doit résider dans le dépôt GitLab IHM associé à votre équipe de projet (ce dépôt sera créé dans votre compte GitLab).

Évaluation

L'évaluation de votre code sera faite par l'équipe enseignante et une soutenance du projet aura lieu dans la semaine du 7 juin (date à préciser). Deux aspects de vos projets seront évalués : * l'implémentation en JavaFX des différents composants et des traitements associés (gestion des événements, propriétés/bindings, l'utilisation des contrôles et des conteneurs JavaFX) * l'ergonomie de votre interface graphique

Bon travail à tous !