IFT1025 PROGRAMMATION 2 – AUTOMNE 2024 – TRAVAIL PRATIQUE 2

Dans ce TP vous allez travailler sur une version évoluée du jeux vidéo DIROgue du TP1. Nous vous donnons une version du DIROgue qui utilise l'architecture client-serveur, ainsi qu'une application graphique qui permet de faire un *replay* d'un donjon généré.

- **DIROgueClient** se connecte avec un serveur, et lit les entrées de l'utilisateur, en supportant 3 commandes :
  - Load : lire un fichier qui contient les instructions pour recréer une aventure, selon les consignes de la partie D du TP1.
  - Save : sauvegarder un fichier avec le rapport, selon les consignes de la partie D du TP1.
  - Exit : sortir.
- **DIROgueServeur** lance un serveur qui reçoit les instructions pour recréer une aventure et les gère avec des EventHandlers appropriés pour chacune.
- **Application de replay** visualise un donjon généré en utilisant le patron de conception MVC. Via la vue Main nous pouvons lire un fichier avec un rapport et l'afficher. Via la vue Replay nous pouvons suivre un donjon généré vers l'avant et vers l'arrière

Nous vous donnons un code-squelette qui comporte 5 packages et plusieurs fichiers java.

- **dirogue.exemple :** package principal où se trouvent les classes avec des méthodes main(). Nous vous donnons également une implémentation du Labyrinthe et de l'Aventure qui utilise les Collections de java.
- **dirogue.exemple.code_squelette** : package avec le code squelette du TP1 et la classe Server que nous avons vue dans la classe
- **dirogue.exemple.rencontre** : package avec les différents types de rencontre selon la partie E du TP1. La classe Rencontre contient la méthode qui fait randomise la création des objets selon la partie F du TP1.
- **dirogue.exemple.view** et **dirogue.exemple.controllers** : ces packages contiennent les vues et les contrôleurs pour l'application graphique, selon le patron de conception MVC.

NB : il y a plein de code qui vous est donné dans ce TP, **ce qui pourrait sembler intimidant au début, surtout si vous commencez en mode panique, 1-2 jours avant la date limite**. Cependant, vous devriez être déjà habitué.e.s avec la structure de la grande majorité du code. Il s'agit du code avec la structure du TP1, code avec la structure client-serveur que nous avons vu en classe et code JavaFX qui suit le patron de conception MVC. Alors, oui, il y a beaucoup de code à comprendre, mais je suis sûr que vous êtes plus que capable de surmonter ce défi !

Ce défi est donc de :

1. Importer le code dans votre IDE, en utilisant Maven (voir annexe 1, à la fin d'énoncé)
2. **Lire** et **comprendre** le code et sa fonctionnalité
3. Faire les 10 tâches suivantes (balises //TODO dans le code) :

#	Description	Resource
1	TODO: Lire le fichier et envoyer les commandes au serveur ligne par ligne.	DIROgueClient.java
2	TODO: Se connecter au serveur.	DIROgueClient.java
3	TODO: Implémenter le handler et ajouter un corridor.	DIROgueServer.java
4	TODO: Sauvegarder le fichier de rapport de l'aventure.	DIROgueServer.java
5	TODO: Charger le fichier de rapport avec un FileChooser et afficher le texte	MainController.java

6	TODO: Ajouter un corridor entre deux pièces avec les identifiants fournis	MonLabyrinthe2.java
7	TODO: Trouver les pièces connectées	MonLabyrinthe2.java
8	TODO: Aller à l'étape précédente du rapport. En atteignant la première étape, il devrait rester là et ne pas générer d'erreur.	ReplayController.java
9	TODO: Aller à l'étape suivante du rapport. En atteignant la dernière étape, il devrait rester là et ne pas générer d'erreur.	ReplayController.java
10	TODO: Créer l'interface utilisateur de Replay	ReplayView.java

L'ordre des tâches dans le tableau ci-dessus n'est pas significative.

Pour les tâches 5, 8, 9, 10, voir l'annexe 2, à la fin de l'enoncé.

Cet énoncé comporte plusieurs pages. **Assurez-vous de toutes les lire, avant de commencer :**

A) CLIENT (20%)

Faire les tâches 1 et 2. Votre Client doit établir une connexion avec le serveur sur localhost, port 1370. Puis, il demande de l'utilisateur de choisir la bonne fonctionnalité et envoie les commandes et instructions pertinentes au serveur via un socket.

B) SERVEUR (20%)

Faire les tâches 3 et 4. Vous devez implémenter un handler pour l'instruction «corridor» en utilisant les méthodes de la classe MonLabyrinthe2. Puis vous devez implémenter un handler pour la commande «save» en utilisant les méthodes de la classe MonAventure.

C) NOUVELLE IMPLEMENTATION DU LABYRINTHE (20%)

Faire les tâches 6 et 7. La classe MonLabyrinthe2 fournit une implémentation de l'interface Labyrinthe qui valorise les interfaces List, Map (et les classes ArrayList, HashMap) pour stocker les listes d'adjacence. Vous devez donner l'implémentation de deux de ses méthodes selon la nouvelle structure.

D) MVC : MAIN CONTROLLER (10%)

Faire la tâche 5 : utiliser une instance de la classe javafx.stage.FileChooser pour lire le contenu d'un fichier avec des instructions et mettre à jour la textArea de la classe MainView avec son contenu.

E) MVC : REPLAY VIEW (20%)

Faire la tâche 10 : implémentez l'interface utilisateur pour la vue de replay en utilisant JavaFX.

F) MVC: REPLAY CONTROLLER (10%)

Faire les taches 8 et 9 : compléter les méthodes du contrôleur de la vue Replay.

PRÉCISIONS GLOBALES

Travaillez en équipes de 2. Le travail tout seul ne sera pas permis sauf aux rares cas et avec approbation antérieure du professeur. Le TP est dû le **dimanche 8 décembre à 23h59** via StudiUM. Aucun retard ne sera accepté.

Un membre de l'équipe doit soumettre un ZIP avec :

a) Un fichier **README.TXT** avec les noms des 2 membres de l'équipe et toute information qui nous aidera à compiler et exécuter votre code. (Soyez concis – Ce fichier n'a pas besoin d'être très long !)
b) Les 6 classes nécessaires pour effectuer les tâches 1-10 uniquement : DIROgueClient.java, DIROgueServer.java, MainController.java, MonLabyrinthe2.java, ReplayController.java, ReplayView.java
c) Un simple rapport texte expliquant le rôle de chaque membre dans le projet, détaillant clairement les tâches de chacun, le temps que chaque membre a consacré à ses tâches, etc.

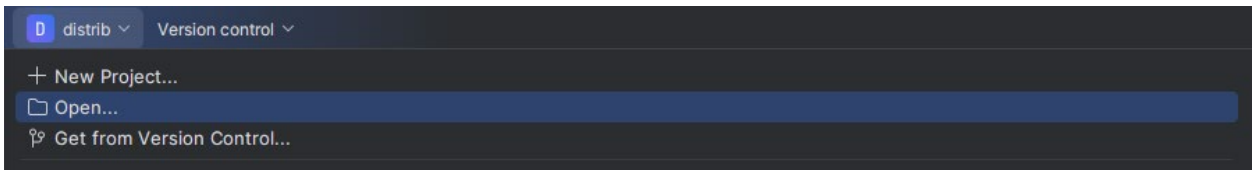
L'autre membre doit soumettre juste une copie du fichier README.TXT (les deux fichiers doivent être identiques).

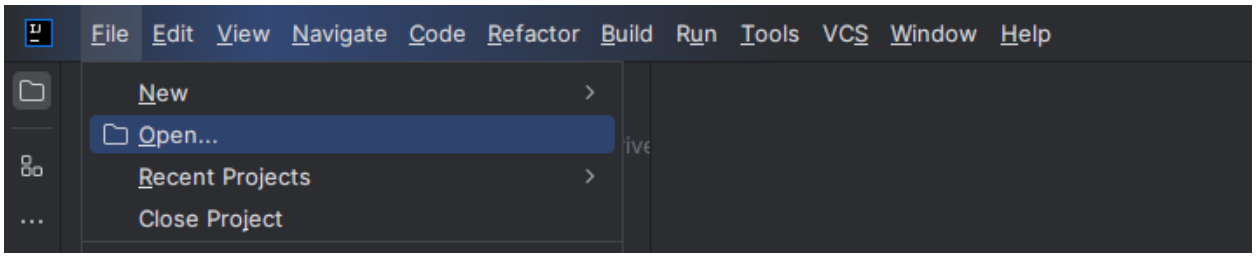
Nous devons être en mesure de placer vos fichiers dans notre propre version du code squelette, et le compiler et exécuter. **Code qui ne compile ou n'exécute pas sera accordé strictement un 0, même s'il s'agit de quelque chose d'aussi simple comme le changement d'une déclaration de package. Vous perdrez également des points si vous n'incluez pas le rapport et si vous soumettez seul.**

ANNEXE 1 : COMMENT IMPORTER LE PROJET DANS VOTRE IDE

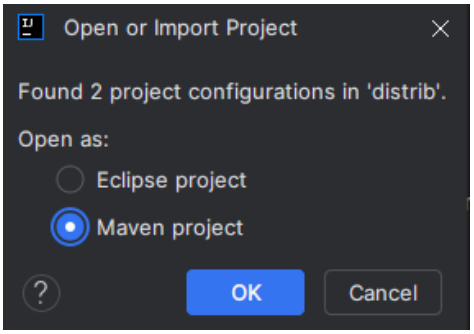
Nous décrivons le processus pour IntelliJ, version « Ultimate 2024.2.1 ». Si vous choisissez d'utiliser un autre IDE pour ce TP, soyez averti que nous ne pourrons pas vous aider à résoudre les problèmes que vous rencontrerez.

1. Téléchargez le fichier ZIP contenant le code et décompressez-le dans le dossier de votre choix.
2. Sur IntelliJ, sélectionnez :
   File > Open > Choisir le dossier où vous avez décompressé le fichier





Ensuite, Open as "Maven project"



3. Attendez pour que Maven télécharge les dépendances nécessaires pour JavaFX et prépare le projet.
4. Vous êtes prêts à travailler.

ANNEXE 2 : COMPORTEMENT ATTENDU DE L'APPLICATION GRAPHIQUE DE REPLAY

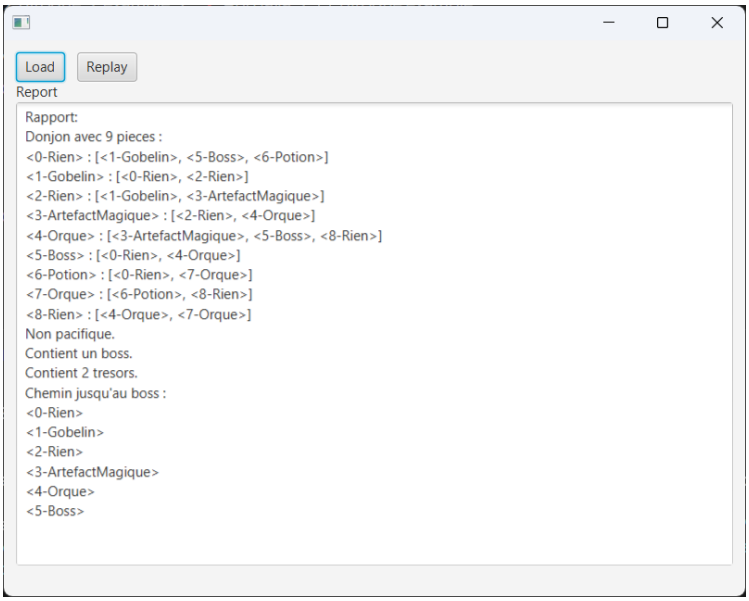
Lors du démarrage de l'application, nous nous trouvons sur la *MainView*.

Ici, nous pouvons visualiser le contenu du fichier rapport.txt après avoir cliqué le bouton *Load* et avoir choisi le fichier via le *FileChooser*.

(Vous devez implémenter ce comportement à la Tâche 5.)

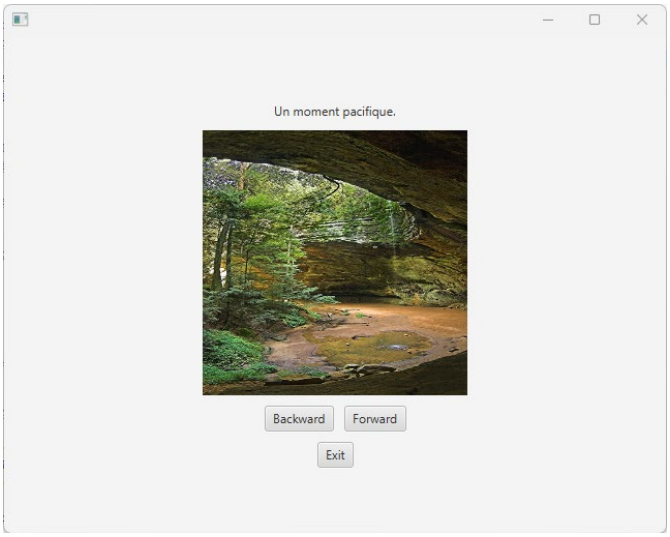
Puis, nous pouvons cliquer sur le bouton *Replay* pour réanimer le donjon.

Cela change la vue vers *ReplayView*.

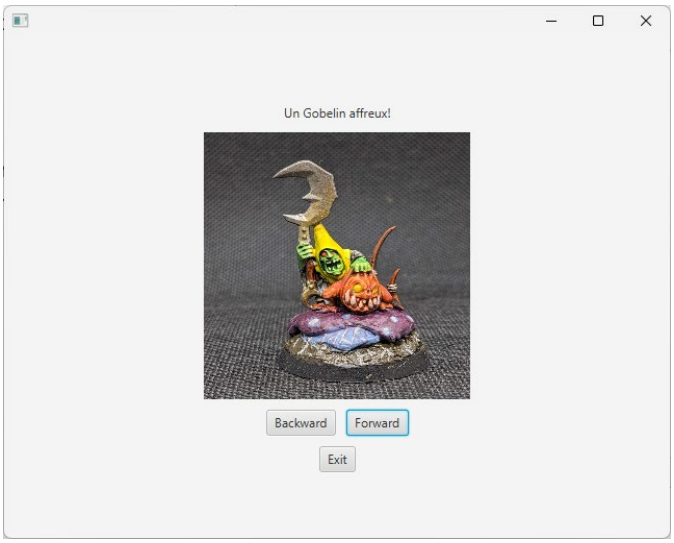


Vous devez créer cette vue (Tâche 10).

La première chose que nous devrions voir c'est l'Exterieur, car c'est toujours le premier nœud dans le donjon.



Nous pouvons cliquer sur le bouton *Forward* pour avancer et explorer le donjon. (Tâche 9, à implémenter).



Nous voyons l'équivalent du «Piece 1 monstre» où un gobelin a été généré.

Nous pouvons faire avancer plus avec le bouton *Forward,* le faire reculer vers l'avant avec le bouton *Backward* (Tâche 8, à implémenter), ou le faire sortir complétement avec le bouton *Exit.*

Les images ci-dessus ne vous sont pas fournies. Si vous voulez, vous pouvez mettre l'image que vous aimez pour chaque rencontre en changeant l'image utilisé dans sa méthode *getSprite()*.