IPO – TP-projet : des moutons dans le donjon

http://www.lri.fr/~blsk/IPO/

Un berger et son troupeau se sont abrités de la pluie dans un vieux château. Se rendant compte que le donjon abritait des loups, le berger cherche à faire ressortir ses moutons. Le voilà fort affairé : il doit tout à la fois guider les moutons vers la sortie et s'interposer entre eux et les loups.

Objectif du TP Ce TP-projet est à réaliser en binôme. Vous y reprenez l'ensemble des éléments réalisés lors du TP 9 (classes abstraites), et les intégrez dans un jeu à un joueur en temps réel. L'ensemble formé par ces deux TP sera à compléter pour le 4 décembre (23h59) et donnera lieu à une soutenance la semaine du 5 septembre. Le tout formera votre note de contrôle continu. L'ensemble de votre projet doit être versionné avec `Git`, et vous devez donner à votre enseignant l'accès à votre dépôt.

Vous trouverez sur la page du cours une archive `.jar` contenant une version de démonstration minimale de ce jeu. Pour lancer cette démonstration, vous devez également télécharger le fichier `laby2.txt` et le placer dans le même répertoire que le `.jar`. Puis double-clic sur `moutons.jar`, ou dans un terminal entrez la ligne `java -jar moutons.jar &`.

Description Le donjon est matérialisé par la classe `Terrain` et ses différentes sortes de `Cases`. Les moutons sont représentés par la classe `Personnage` et les loups par la classe `Monstre`. Tous ces éléments peuvent être repris intégralement, sans modification par rapport à ce qui était demandé au TP9.

Le berger est représenté par un nouveau sous-type d'`Entite`. Comme les autres entités, il peut se trouver sur n'importe quelle case traversable, et dispose d'une résistance. Comme les autres entités, il est un obstacle au déplacement des moutons et des loups :

— un mouton se trouvant face au berger changera de direction,

— un loup se trouvant face au berger l'attaquera, faisant décroître sa résistance.

Le jeu présente au joueur une vue de l'ensemble du donjon, mise à jour après chaque `tour` (10 fois par seconde). Le joueur y déplace le berger en temps réel à l'aide des flèches du clavier.

Travail à réaliser Vous trouverez sur la page du cours un fichier principal `Donjon.java` (complet), un squelette d'interface graphique `FenetreGraphique.java` (à compléter), et un nouvel exemple de description de terrain (`laby2.txt`), à combiner avec vos fichiers du TP9. Voici une liste des tâches à réaliser, dans l'ordre.

1. Créer un dépôt `Git` par binôme pour gérer votre projet, et placez-y tous les fichiers (instructions détaillées page suivante).
2. Compléter dans la classe `FenetreGraphique` les méthodes d'affichage, de sorte à pouvoir observer le comportement de votre jeu, sans interaction.
3. Créer une classe `Joueur` étendant `Entite`, pour représenter le berger. Dans le constructeur de la classe `Terrain`, étendre le décodage du fichier d'entrée pour qu'il interprète un `H` comme la position de départ du joueur (vous pouvez maintenant utiliser le fichier `laby2.txt`).
4. Ajouter à la définition de la classe `FenetreGraphique` l'annotation **`implements`** `KeyListener`, et étendre cette classe avec les méthodes nécessaires, de sorte que les flèches du clavier permettent de déplacer le joueur. *Note :* l'interface `KeyListener` a un fonctionnement similaire à `MouseListener` (mais c'est à vous de trouver le détail des méthodes à implémenter !)
5. Ajouter au berger la possibilité de sortir lui aussi, en appuyant sur la touche « espace » lorsqu'il se trouve sur une `Sortie`.
6. Ajouter à la classe `Jeu` une méthode testant si la partie est finie :
   — le jeu s'arrête lorsque le berger sort, avec un score égal au nombre de moutons sauvés,
   — le jeu s'arrête également lorsque le berger est mangé, avec un score divisé par deux.
7. Ajouter, à volonté, de nouveaux éléments pour enrichir le jeu.

Instructions pour l'initialisation de `Git`

Création du projet lui-même, à faire par l'un des membres du binôme.

1. Allez sur l'interface web du `gitlab` de l'université :
   `https://gitlab.dsi.universite-paris-saclay.fr/`
   et connectez-vous (avec vos identifiants habituels de l'université).
2. Cliquez en haut à droite : *new project*, puis sélectionnez *create blank project*.
3. Donnez un nom à votre projet. À la ligne *project URL*, assurez-vous que votre nom apparaît, ou allez le sélectionner en cliquant sur *pick a group or namespace*.
4. Assurez-vous que le niveau de visibilité *private* est sélectionné, puis confirmez la création.

Sélection des autres participants, à réaliser dans l'interface web du projet.

1. Dans le menu, sélectionnez *project information*, puis *members*.
2. Cliquez sur *invite member* et intégrez le deuxième membre du binôme, avec le rôle *maintainer*.
3. Recommencez et intégrez votre encadrant(e) de TP, avec un rôle *reporter* (ou supérieur), puis de même avec `thibaut.balabonski`.

Note : vous ne pouvez inviter que quelqu'un qui s'est déjà connecté à ce `gitlab` un jour. Si vous ne trouvez pas la personne dans la liste, demandez-lui de se connecter.

Instructions pour connecter `Git` et `IntelliJ IDEA`

Import du projet dans `IntelliJ`, à faire par chaque membre du groupe et sur chaque ordinateur utilisé.

1. Dans `IntelliJ`, allez sélectionner dans le menu *file*, *new*, puis *project from version control*.
2. Vérifiez que `Git` est sélectionné dans la ligne *version control*, puis indiquez l'URL du projet (l'adresse de la page principale de votre projet sous `gitlab`, que vous pouvez aller copier depuis la barre d'adresse). Vous pouvez aussi depuis ce menu sélectionner le dossier dans lequel se trouvera le projet.
3. Entrez vos identifiants pour autoriser `IntelliJ` à se connecter à votre compte `gitlab`, puis validez.

Initialisation des fichiers, à faire par l'un des membres du groupe.

1. Dans le projet `IntelliJ`, faites un clic droit *new*, *directory*, et nommez `src` le dossier créé.
2. Faites un clic droit sur `src`, puis sélectionnez *mark directory as*, et *sources root*.
3. Ajoutez dans le dossier `src` vos fichiers `.java`. Si l'interface vous propose des les ajouter à `Git`, acceptez.
4. Poussez les nouveaux fichiers sur le dépôt `Git` : dans le menu *git*, allez chercher les actions *commit* et *push* (l'action *commit* ouvrira dans l'interface une fenêtre où vous devez écrire un message décrivant le contenu de la modification que vous enregistrez).

Instructions pour travailler une fois `Git` en place

À réaliser par chaque personne travaillant sur le projet.

1. Au début de chaque session de travail, récupérez les fichiers du dépôt partagé pour mettre à jour vos fichiers locaux avec `pull`. Pour cela : cliquez dans le menu *git*, puis *pull*. Recommandation : c'est généralement une bonne idée, dans *modify options*, d'allez sélectionner `--rebase`.
2. Après chaque modification significative, enregistrez (localement) votre travail avec `commit`. Dans le menu *git*, sélectionner *commit*, et entrer un message décrivant la modification.
3. À la fin de chaque séance (voire plus souvent), publiez vos *commits* sur le dépôt partagé avec `push`. Dans le menu *git*, sélectionner *push*, et valider.

Pour éviter les interférences entre les modifications faites par plusieurs personnes travaillant en parallèle sur le même projet, il vaut mieux souvent publier ses modifications (`commit`/`push`) et récupérer les modifications des partenaires (`pull`).

Note : l'historique des *commit*s est daté, et permettra à votre encadrant d'apprécier la régularité de votre travail.