IPO – TP-projet : feu furieux

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

Vous devez vous échapper d'un manoir en flammes avant que la chaleur n'ait raison de votre résistance. Attention, certaines portes sont fermées et il vous faudra en trouver les clés pour progresser vers la sortie.

Objectif du TP. Ce TP-projet est à réaliser en binôme. Vous allez réaliser un nouveau jeu pour lequel vous pouvez vous inspirer de ce qui a été réalisé au TP8. Votre travail sera à compléter pour le 10 décembre (23h59) et donnera lieu à une soutenance la semaine du 11 décembre. 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.

Description. Le manoir (classe `Terrain`) est représenté par un tableau à deux dimensions, de cases qui peuvent être :

- des murs (intraversables, en noir),
- des halls (traversables, en dégradé de blanc à rouge selon l'intensité de la chaleur),
- la sortie (en bleu),
- des portes, qui peuvent être ouvertes (même couleur qu'un couloir) ou fermées (en vert).

Déplacement et dégâts. Le joueur peut se déplacer librement dans les pièces du manoir, sur la sortie, ou par une porte ouverte. Chacune de ces cases a un attribut `chaleur` dont la valeur peut aller de 0 à 10. À chaque arrivée sur une case, la résistance du joueur décroît en fonction de la chaleur de la case sur laquelle il arrive. La résistance décroît de même à chaque tour de jeu (10 pas seconde) en fonction de la case sur laquelle il se trouve.

Propagation des flammes. À chaque tour de jeu, les flammes se propagent de proche en proche dans les cases traversables, avec la règle suivante : on additionne les chaleurs d'une case et de toutes ses huit voisines, on tire un nombre au hasard entre 0 et 199, et : si ce nombre est inférieur à la chaleur totale, on incrémente de 1 (maximum 10), si ce nombre est supérieur à 190 on décrément de 1 (minimum 0), et sinon on ne change rien.

Clés et portes. Une porte est ouverte ou fermée. Une porte ouverte se comporte comme une case `Hall` ordinaire. Passer une porte fermée consomme une clé, et n'est possible que si le joueur possède effectivement au moins une clé. La porte est alors définitivement ouverte. Les clés se trouvent sur des cases `Hall` et sont représentées par de petits carrés gris. Le joueur les ramasse automatiquement en passant dessus. Les clés sont interchangeables, mais sont à usage unique (il suffit donc de tenir le compte du nombre de clés que possède le joueur, et de l'incrémenter ou le décrementer en fonction des événements).

Visibilité. Le joueur ne connaît pas le plan du manoir. Sa visilité est limitée à un une zone de 37 cases, correspondant à un cercle approximatif d'un rayon de trois cases. Si le joueur est sur la case de coordonnées (x, y), une case de coordonnées (x', y') est dans son rayon de visibilité lorsque $(x - x')^2 + (y - y')^2 \leq 10$.

Fin de partie. La partie est perdue si la résistance du joueur atteint zéro, et gagnée si le joueur atteint la sortie. Le score est la résistance restant à la fin de la partie.

Travail à réaliser. Vous trouverez sur la page du cours les squelettes d'un fichier principal `Furfeux.java` (moteur principal du jeu, et démarrage), d'une interface graphique `FenetreJeu.java` (affichage et traitement du clavier), et de classes `Terrain`, `Case` et `Joueur` (éléments du jeu), ainsi que le descriptif d'un terrain exemple (`manoir.txt`). Vous devez d'abord créer un dépôt `Git` par binôme pour gérer votre projet, et y placer tous les fichiers (instructions détaillées à la fin du sujet). Vous devez ensuite réaliser les tâches suivantes :

- Compléter la hiérarchie des cases, pour avoir en particulier les quatre classes concrètes `Mur`, `Hall`, `Porte` et `Sortie`. Observez la méthode de génération d'un terrain à partir d'un fichier de description (constructeur de la classe `Terrain`) pour voir de quels constructeurs chacune à besoin. Attention, une case peut avoir plusieurs constructeurs.
- Compléter dans la classe `FenetreGraphique` les méthodes d'affichage, de sorte à pouvoir observer la portion de terrain qui se trouve autour du joueur.
- Compléter dans la classe `Joueur` les méthodes permettant au joueur de se déplacer.
- 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.
- Faire en sorte qu'à chaque entrée sur une case et à chaque nouveau tour, le joueur subisse des dégâts égaux au niveau de chaleur de cette case.
- Compléter dans la classe `Furfeux` une méthode testant si la partie est finie, c'est-à-dire si le joueur est sorti ou si sa résistance est tombée à zéro.
- Ajouter au joueur la possibilité de ramasser les clés que contiennent certaines cases `Hall`, et de les utiliser pour ouvrir les portes.
- Ajouter à chaque nouveau tour une phase de propagation du feu pour chaque case traversable.
- 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 manu 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.