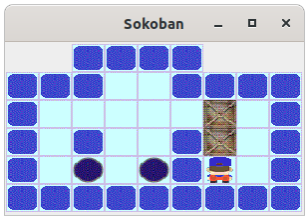
POO – Programmation Orientée Objet en Java

Fiche de TP numéro 7 - Projet Sokoban

Sokoban est un jeu créé et écrit par Hiroyuki Imabayashi dans les années 70-80 (`https://fr.wikipedia.org/wiki/Sokoban`).

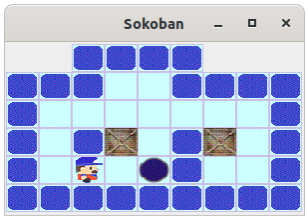
C'est un jeu de réflexion : un robot doit pousser des caisses sur des emplacements prévus. Il peut seulement pousser une caisse (pas la tirer), et une seule caisse à la fois. Vous allez devoir programmer ce jeu. Quelques cartes correspondant à différents niveaux vous sont proposées comme base de travail.

Vous allez devoir proposer une version du jeu en mode graphique et une en mode console (mode texte). Dans les images ci-dessous, la même situation est vue en mode graphique et en mode texte.



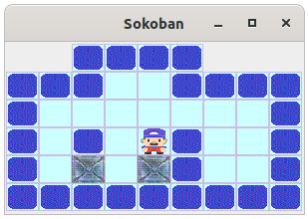

```
//#####///
###   ####
#      $ #
# #   #$ #
# .  .#@ #
#########
```

FIGURE 1 – Niveau 1 - début




```
//#####///
###   ####
#        #
# #$ #$ #
# + .#   #
#########
```

FIGURE 2 – Niveau 1 - en cours de résolution




```
//#####///
###   ####
#        #
# # @#   #
# * *#   #
#########
```

FIGURE 3 – Niveau 1 - C'est gagné !

Dans le mode texte, les `#` sont les murs, les `/` du vide, `@` est le robot, `$` les caisses, `.` les destinations, `*` une caisse sur une destination, `+` le robot sur une destination.

Un peu d'organisation

Comme vous allez proposer deux versions du jeu, il n'est pas question de programmer deux fois l'application, seulement de proposer deux affichages différents. Cela nécessite un peu d'organisation dans le code.

Étape 1 : *Un peu d'organisation*

Vous devez retrouver la séparation du modèle et de la vue que vous avez déjà pratiquée pour le jeu des LiLines et pour le Memory. Vous allez le formaliser par la création de deux packages. Dans le répertoire `src`, créez un répertoire`modele`, un répertoire `vuetexte` et un répertoire `vueGraphique`.

Un peu d'analyse

Dans un premier temps, on va s'attacher à proposer un modèle pour une carte de niveau. Tout se passe dans le répertoire `modele`, et tous les fichiers de ce répertoire comportent comme première ligne :

```
package modele;
```

Étape 2 : *Observation*

Regardez les cartes proposées. Combien d'éléments différents font-elles apparaître ?

Étape 3 : *Représentation des éléments*

La première question à se poser, c'est de savoir comment on peut représenter les éléments de base d'un plateau de jeu : des cases qui sont soit un mur, soit du vide, soit le sol, soit une destination... Les caisses, comme le robot, se posent sur un sol ou sur une destination. Proposez une hiérarchie de classes qui permette de représenter tous les éléments d'une carte qui ne se déplacent pas. Chaque élément doit pouvoir répondre aux questions : y-a-t-il un joueur sur moi ? y-a-t-il une caisse sur moi ? Associez un caractère avec chacun de ces éléments.

Étape 4 : *Représentation de la carte*

On peut maintenant réfléchir à la carte (que nous appellerons la classe `Carte` dans la suite du sujet), qui contient essentiellement tous les éléments de base. Définissez la classe, son ou ses attributs. Faites au plus simple, le modèle pourra être enrichi au fur et à mesure. Pour construire une carte, il suffira d'avoir une liste de chaînes de caractères.

Étape 5 : *Dans le répertoire `vueTexte`*

Dans la méthode `main` d'une classe `SokobanTexte`, écrivez (à la main), une telle liste représentant la carte `map1.txt`. Construisez une instance de la carte à partir de cette liste. N'oubliez pas de déclarer que la classe `SokobanTexte` est dans le package `vueTexte`, et n'oubliez pas l'importation nécessaire du `modele`.

Étape 6 : *Affichage en mode texte*

Complétez votre classe représentant une carte par une méthode `toString()`.

Étape 7 : *Programme principal*

Vérifiez avec son affichage que tout est correct.

Lire dans un fichier

Étape 8 : *Lire une carte*

Les cartes de niveaux sont données dans des fichiers texte. Écrivez une classe `Lecture` (dans le package `modele`) qui prend en paramètre de son constructeur un nom de fichier. Son constructeur lui permet d'initialiser trois attributs : la liste des lignes du fichiers sous la forme d'une liste de chaînes de caractères, le nombre de lignes et la taille de ses lignes. Vous pouvez prendre comme hypothèse que toutes les lignes sont de la même taille. Inspirez-vous des exemples du cours !

Étape 9 : *Charger une carte*

Modifiez la classe `Carte` pour qu'elle fasse appel dans son constructeur à la classe `Lecture`. Cela va vous permettre de varier les cartes lues. Le nom du fichier sera choisi (pour le moment) dans la méthode `main`, au moment de l'instanciation de la carte.

Étape 10 : *Tests*

Prenez en compte les modifications dans la méthode `main`. Vérifiez que tout est correct : l'affichage de votre carte doit correspondre au contenu du fichier.

Les déplacements

Tout se passe dans le package `modele`.

Étape 11 : *Les directions*

Le joueur peut juste diriger le robot dans l'une des quatre directions : haut, bas, gauche, droite. Proposez une solution pour représenter les directions. Il sera intéressant de mémoriser les incréments en abscisse et en ordonnée associés à chaque direction.

Étape 12 : *Le robot*

Proposez une modélisation pour le robot. Quels sont ces principaux attributs ? Ses principales méthodes ? Qui a toutes les informations pour contrôler les déplacements du robot ?

Étape 13 : *La carte*

Modifiez la classe pour une carte de niveau afin de prendre en compte le robot.

Étape 14 : *Le robot se déplace*

Quelle est l'action principale du joueur sur le root ? Quelle est l'information que la `Carte` a besoin d'avoir en paramètre pour le déplacement du robot ? Implémentez cette méthode.

Étape 15 :

La fin de la partie ? Pour le moment, nous n'avons pas toutes les informations pour pouvoir bien gérer la fin de partie. Écrivez une méthode `finDePartie` qui retourne toujours `false`.

Une version en mode texte

Nous allons faire une première version en mode texte (et onc dans le package `vueTexte`).

Étape 16 : *Le mode texte*

Pour l'application en mode texte, la classe `ModeTexte` va

— gérer les interactions avec le joueur

— contrôler le déroulement d'une partie

Pour pouvoir gérer les interactions avec le joueur, il faut choisir les lettres avec lesquelles le joueur va donner la direction au robot. Par exemple, on peut décider que (pour un clavier azerty), la lettre `q` désigne la gauche, `z` le haut, `d` la droite, `s` le bas. Vous pouvez faire un autre choix.

— Comme toute vue, la classe `ModeTexte` doit avoir le modèle en attribut.
— Comment associer à un caractère une direction (vous avez déjà défini les directions) ? Définissez la structure de données que vous aurez choisie comme attribut de la classe `ModeTexte`.
— Écrivez le constructeur de `ModeTexte`.
— À l'aide de la méthode `lireCar` définie dans la classe `Outil` qui vous est fournie, écrivez une méthode qui dialogue avec l'utilisateur jusqu'à ce qu'il saisisse l'un des caractères attendus.
— Écrivez une méthode qui lance une partie.

Étape 17 : *Lancer une partie en mode texte*

Mettez à jour la classe `SokobanTexte` pour lancer une partie en mode texte.

Détecter la fin de partie

Étape 18 : *Fin de partie*

La partie est finie lorsqu'il y a une caisse sur chaque destination. Pour tester cela, le plus simple est de mémoriser les coordonnées des destinations lorsque la carte est chargée. Modifiez le constructeur de la classe `Carte` pour mettre à jour ce nouvel attribut. Réécrivez la méthode `finDePartie`.

Une version graphique

Tout se passe dans le package `vueGraphique`.

Étape 19 : *Afficher la carte*

Proposez une première version de la classe `VueSokoban` qui affiche la carte en fonction des informations fournies par le modèle. Comme le mode texte, le modèle doit être un attribut de cette classe. Pensez à associer à chaque caractère d'une carte (`@`, `#`, ...) une image (elles sont disponibles sur Moodle). Dans un premier temps, le robot sera toujours affiché avec la même image. Créez une nouvelle classe `Sokoban` dans le package `vueGraphique` qui contient seulement une méthode `main`. Inspirez-vous de la classe `SokobanTexte`.

Étape 20 : *Gérer les événements*

Vous voulez récupérer les événements *flèche gauche* - `KeyEvent.VK_LEFT`, *flèche droite* - `KeyEvent.VK_RIGHT`, *flèche haut* - `KeyEvent.VK_UP` et *flèche bas* - `KeyEvent.VK_DOWN`. Pour cela il faut avoir un `KeyListener`. La méthode `getKeyCode()` de `KeyEvent` retourne le code de la touche enfoncée. Associez à chaque touche une direction. Envoyez un message au modèle pour le déplacement. S'il y a eu un déplacement, ne mettez pas à jour les éléments ed la carte qui ne changent jamais.

Étape 21 : *Gérer les différentes images du robot*

Quelle information faut-il ajouter dans la classe `Robot` pour que la vue sache quelle image du robot afficher ? Modifiez la classe `Carte` pour qu'elle permette à la vue d'accéder à cette information. Modifiez la `Vue` pour afficher la bonne image du robot en fonction de la direction choisie.

Améliorations

Vous pouvez maintenant proposer des améliorations :

— ajouter un compteur de mouvements ;

— le joueur peut annuler le dernier mouvement ;
— on peut redémarrer la partie ;
— lorsqu'on a fini une carte, on peut passer au niveau suivant (`map1.txt`,`map2.txt`, `map3.txt`, ...)
— toute autre idée...

Rendre le projet

Vous devez rendre :
— deux fichiers `sokobanToto.jar` et `sokobanTexteToto.jar` où `Toto` sera remplacé par votre nom. Ces fichiers pourront être interprétés par la commande `java -jar sokobanToto.jar` (ou `java -jar sokobanTexteToto.jar`) ;
— un fichier `src-sokobanToto.zip` (où `Toto` sera également remplacé par votre nom) qui contiendra le répertoire de votre projet, avec les fichiers sources (les *.java*), la documentation, les images utilisées et les cartes en mode texte.