Projet : jeu de sokoban

m2103

01-05-2021

Table des matières

1 Le jeu de sokoban

Vous connaissez certainement ce jeu :

- Un terrain rectangulaire. Certaines cases sont occupées par des murs, qui bloquent le passage.
- Sur le terrain, il y a des caisses.
- il y a un personnage déplacé (4 directions) par le joueur qui peut
  - aller dans une case vide voisins
  - ou pousser une série de caisses, si il y a une case vide derrière
- **objectif** : amener toutes les caisses vers des destinations indiquées.

Exemple de situation

```
  0 1 2 3 4 5 6 7 8
0 . . # # # # . . .
1 # # # . . # # # #    # = mur
2 # . . . . . C . #    P = personnage
3 # . # . . # C . #    C = caisse
4 # . x . x # P . #    x = destination
5 # # # # # # # # #    . = case vide
```

Sans vouloir vous priver du plaisir de chercher, pour gagner il faut s'occuper d'abord d'amener la caisse du haut (2,6) dans la destination de gauche. Pour cela il faut contourner par la droite, pousser la caisse à gauche (4,2), manoeuvrer par en haut, etc. La suite de mouvements peut s'écrire `RUULLLULDDRDLL` (Right Up etc).

Objectif généraux du programme

- permettre à un joueur de mener des parties sur des plateaux différents.
- les plateaux seront obtenus à partir de différentes sources
  - des fichiers
  - une base de données

2 Modalités du projet

- Travail **en solo**.
- Utilisation des heures de TP et de TD pour avancer sur le projet.
- Projet GIT sur le serveur de l'IUT pour la sauvegarde régulière et le suivi par les enseignants.

- Travail à remettre fin Mai sur la plate-forme Moodle (projet netbeans + rapport)

Rappel : vos enseignants sont là pour vous aider[1]. Mais pour cela il faut avoir quelque chose à leur montrer, des questions précises à poser, et tenir compte de leurs indications pour avancer.

3 Objectif 1 : un jeu jouable

1. Le code source construit un exemple "en dur"

```
Board b = new Board("Hard-Coded Example", 5, 6);
b.addHorizontalWall(0,5,6); b.addHorizontalWall(4,5,6);
b.addVerticalWall(0,0,5);   b.addVerticalWall(0,5,5);
b.addBox(2,1);              b.addBox(2,3);
b.addTarget(3,1);           b.addTarget(3,2);
b.setPosition(3,4)
```

2. Il lance une boucle d'interaction avec le joueur

- affichage de la position
- saisie d'une ligne (suite de lettres L R U D, autres caractères ignorés)
- exécution des mouvements

jusqu'à ce que les caisses soient arrivées à destination, ou que le joueur abandonne.

La classe principale de votre projet (celle qui a le `main`) s'appellera `Player`.

4 Objectif 2 : lecture de jeu à partir d'un fichier

Un fichier peut contenir une description de plateau, par exemple à ce format simple

```
A Simple Board
##########
#x.x#....#
#...C..P.#
#....C...#
##########
```

Une approche classique et élégante pour faire ce genre de choses (charger des plateaux depuis différentes sources) est de définir une interface `BoardBuilder` :

```
interface BoardBuilder {
    Board build() throws BuilderException;
}
```

Le rôle d'un `Builder` est

1. d'*accumuler* des informations à propos d'un objet qu'il faudra construire
2. de *produire* effectivement cet objet quand on le lui demande (par appel à sa méthode `build`).

Une des implémentations sera la classe `FileBoardBuilder` qui

- lira un fichier texte
- **accumulera** les informations (le nom, les positions etc).
- et, quand on appellera la méthode `build`, fournira une instance de `Board`, si c'est possible.

Une autre implémentation serait un "`TextBoardBuilder`" à qui on donne des chaînes de caractères :

```
var builder = new TextBoardBuilder("A Simple Board");
builder.addRow("##########");
builder.addRow("#x.x#....#");
builder.addRow("#...CC.P.#");
builder.addRow("#........#");
builder.addRow("##########");
```

1. De préférence pendant les séances prévues.

```
var board = builder.build();
```

Il serait judicieux de

1. Commencer par implémenter le `TextBoardBuilder`.
2. S'en servir pour réaliser le `FileBoardBuilder`.

5 Objectif 3 : base de données

On veut maintenant mettre les plateaux dans une base SQLite. Ça doit pouvoir se faire avec une base à 2 tables.

```
BOARDS
| board_id    | name                 | nb_rows | nb_cols |
|-------------|----------------------|---------|---------|
| "simple"    | "A simple board"     | 5       | 10      |
| "difficult" | "A really tough one" | ...     | ...     |

ROWS
| board_id | row_num | description  |
|----------|---------|--------------|
| "simple" | 0       | "##########" |
| "simple" | 1       | "#x.x#....#" |
| "simple" | 2       | "#...CC.P.#" |
| "simple" | 3       | "#........#" |
| "simple" | 4       | "##########" |
```

1. Dans votre projet, définir une autre classe `Administrator` avec un `main` avec un menu du genre :

```
ADMINISTRATION INTERFACE - USE WITH CAUTION

1. Create new database
2. List boards
3. Show board
4. Add board from file
5. Remove board from database [DANGEROUS]
6. Quit.
```

Quelques idées

- définir un objet `Database` qui détiendra une connexion ouverte, et intercédera pour vous auprès de la base de données. Il aura par exemple les méthodes

```
void add(String id, Board board);
void remove(String id);
Board get(String id);
```

- la méthode `get` pourra employer une `TextBoardBuilder` pour construire une `Board` à partir des "rows" prises dans la base.
- pour l'option 4, faire appel à un `FileBoardBuilder` pour lire le fichier.
- Dans la classe `Player`, faire un menu pour sélectionner un des plateaux de la base.