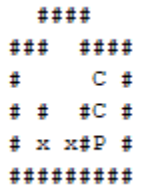
Rapport Sokoban

M2103

Riboulet Célian **05/2021**

Table des matières

1/ introduction



Le projet sokoban réalisé concerne un jeu dans un terminal qui consiste à déplacer un joueur dans le but de positionner les caisses d'un niveau à des endroits spécifiques.

Ce rapport décrit la conception et la réalisation de ce projet.

2/ Analyse

1) Spécification des besoins

Bases :

- Créer des niveaux depuis un fichier texte et pouvoir les importer dans le jeu.
- Pouvoir charger un niveau et jouer dessus.
- Déplacer le joueur dans les directions possibles en récupérant les "inputs" de l'utilisateur depuis le terminal.
- Le joueur doit pouvoir pousser une caisse quand il est possible de le faire.
- Une caisse doit pouvoir pousser une quantité infini d'autres caisses.
- Le jeu doit se terminer lorsque toutes les cellules de destination du niveau ont une caisse à leur position.

Compléments :

- Charger les niveau depuis une base de données.
- Pouvoir interagir en tant qu'administrateur avec la base de données pour pouvoir modifier son contenu..
- Pouvoir afficher un niveau d'une façon plus élégante que un caractère par cellule du plateau.

2) Choix de conception

Déroulement de la partie :

Pour gérer le déroulement d'une partie avec le menu qui le précède, une simple classe Sokoban suffit, c'est cette même classe qui comprend le point d'entrée du programme.

Gestion des données du niveau

Les données du niveau sont stockées dans la classe **Map**, on y trouve aussi des méthodes permettant d'agir sur des éléments du niveaux, comme déplacer un objet.
Ces méthodes sont indirectement commandées par l'utilisateur.

Si l'utilisateur écrit 'R' alors le joueur doit se déplacer à droite, ce déplacement est géré dans la classe Map.

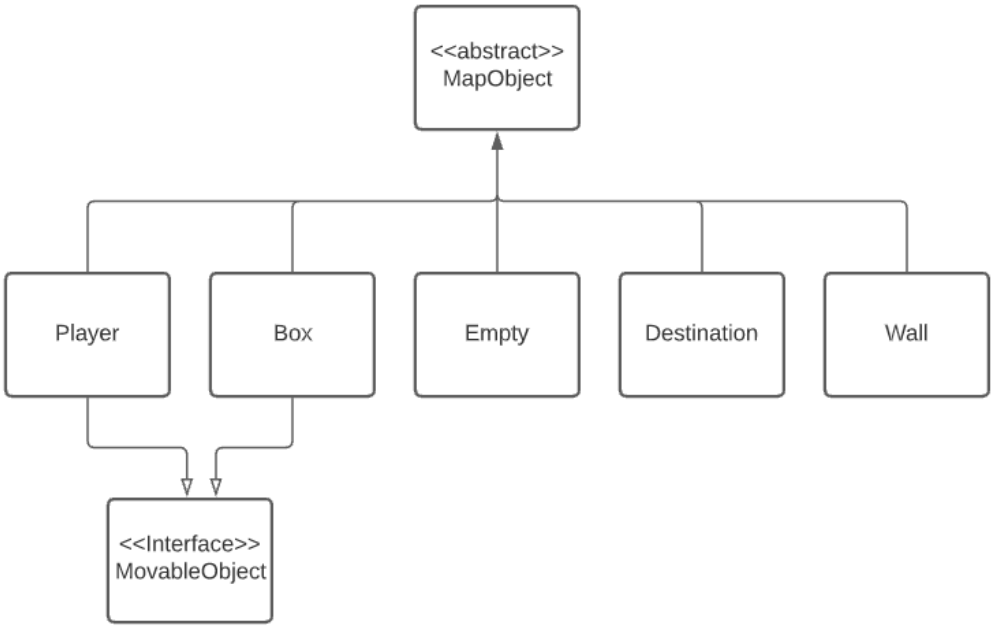
Le niveau est séparé en cellule, chacune de ces cellules représente une case sur laquelle il est possible ou non de se déplacer en tant que joueur (**Player**) ou en tant que boîte (**Box**).

D'un point de vue technique, une cellule = une case d'un tableau 2D : `private MapObject[][] map;`

Une case est une instance de la classe **MapObject**, cela peut sembler excessif d'utiliser une instance par cellule du niveau mais nous offres l'avantage

- de pouvoir stocker une infinité de données par cellule
- d'appliquer à certains type de case des particularité grâce à des interface (Par exemple l'interface MovableObject s'applique à une classe Box et Player qui toutes deux héritent de la classe MapObject)

La class MapObject et son interface MovableObject :



Chacune des cellules d'un niveau sont donc comme dis précédemment des instances de la classe MapObject.

La classe MapObject étant abstraite, il existe pour chacun des types de cellule possible une classe héritant de la classe MapObject
(Soit : Player/Box/Empty/Destination/Wall).

L’interface MovableObject ajoute à certaines classes la possibilité d’être déplacée sur la map.

Important : quand un MapObject est créé, une position lui est assignée, cette position correspond aux coordonnées de la cellule du plateau sur laquelle il se trouve est n’est PAS modifiable directement.

Ainsi pour déplacer un MapObject (Implémentant MovableObject), il faut lui demander une copie de lui même qui elle contient une position différente.
Ce mécanisme permet d'éviter certaines erreurs de positionnement et oblige le code à explicitement mettre à jour une position.

3/ Aperçu final du jeu

1) Menu

- Au lancement du jeu, le joueur doit sélectionner un niveau sur lequel jouer

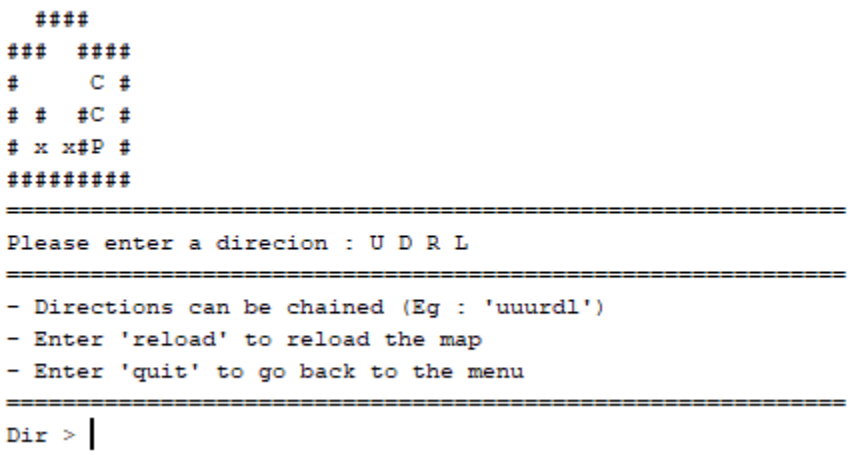
```
================================================================
Choose a map :
1 -  Tutorial [easy]
2 -  Push [easy]
3 -  Too close [medium]
4 -  Stucked [hard]
Num >
```

- La qualité d’affichage du niveau doit ensuite être sélectionnée

```
================================================================
Choose your graphics quality :
1 - High details and big drawings (Recommended)
2 - Compact
Num >
```

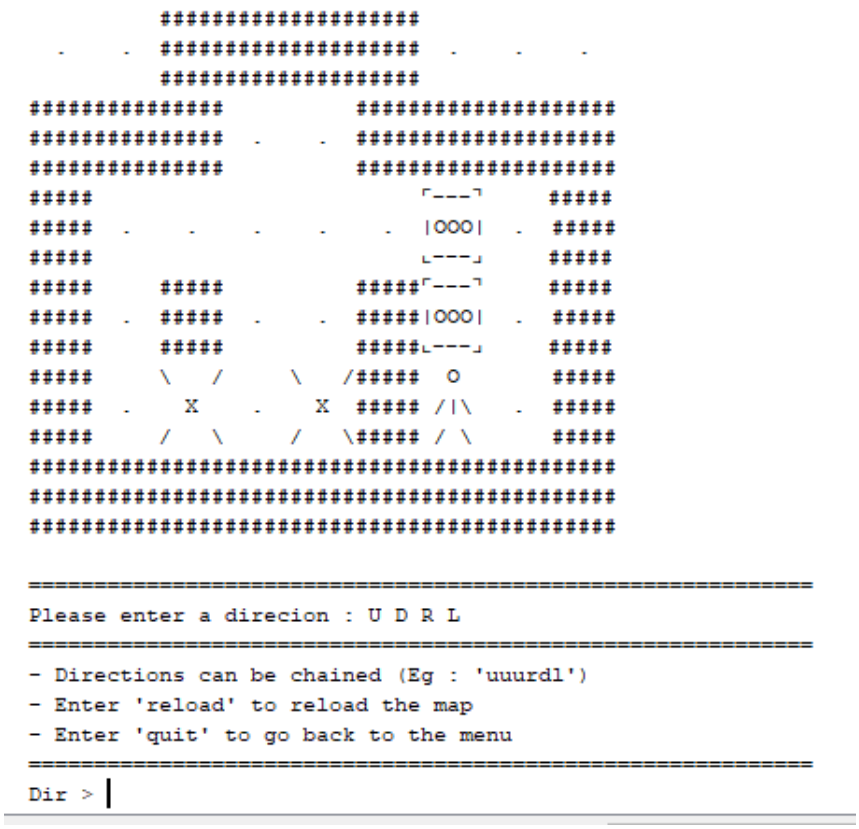
2) Affichage des niveaux

- Affichage d'un niveau en mode compact



Au premier tour, des indications sont écrites en plus du niveau, par la suite seulement le "Dir" sera affiché pour alléger le tout.

- Affichage du même niveau en mode détaillé :



3) Administration de la base de donnée

Menu de sélection et affichage du contenu de la base de donnée

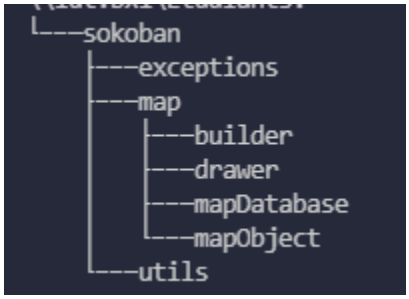
```
DB Administration :
0 : Quit
1 : Initialize DB
2 : See DB
3 : Add map from file
4 : Delete map
5 : drop tables [DANGEROUS]
2

Maps :
|--id-|-----------------name-|------difficulty-|
|   0 |              Tutorial |             easy |
|   1 |                  Push |             easy |
|   2 |             Too close |           medium |
|   3 |               Stucked |             hard |
-------------------------------------------------


Rows :
|--mapId-|--rowId-|------------------------content-|
|      0 |      0 |                    ############ |
|      0 |      1 |                    #..P.......# |
|      0 |      2 |                    #.....C...X# |
|      0 |      3 |                    ############ |
|      1 |      0 |                    ############ |
|      1 |      1 |                    ###...#....# |
|      1 |      2 |                    #...CP.....# |
|      1 |      3 |                    #..C..#.Cx## |
|      1 |      4 |                    #..##.###..# |
|      1 |      5 |                    #...#x#x...# |
|      1 |      6 |                    ############ |
|      2 |      0 |                       ..####... |
|      2 |      1 |                      ###..#### |
|      2 |      2 |                      #.....C.# |
|      2 |      3 |                      #.#..#C.# |
|      2 |      4 |                      #.x.x#P.# |
|      2 |      5 |                      ######### |
|      3 |      0 |                   ############ |
|      3 |      1 |                   #x.P.x.C.C.x# |
|      3 |      2 |                   #.#.##.#.##.# |
|      3 |      3 |                   ###.C.CCCC.x# |
|      3 |      4 |                   #####....#### |
|      3 |      5 |                   ############ |
-------------------------------------------------------
```

4/ Réalisation

1) Organisation en package



Sokoban : tous les éléments du jeu (Menu/Main).
Exceptions : l'ensemble des erreurs possibles des différents systèmes.
Map : ensemble du code liée au plateau.
Builder : construction d'un plateau à partir d'un fichier ou base de données.
Drawer : affichage du plateau.
MapDatabase : base de données des maps.
MapObject : objets de la map.
Utils : classes utilitaire (Lecture terminal / Vecteurs)

2) Lecture des inputs (Scanner)

La lecture du terminal et donc d'une entrée utilisateur étant redondante dans le projet, j'ai décidé de simplifier l'utilisation de la classe Scanner via une classe utils.ScannerUtils

contenant les méthodes :

```
awaitString(): String
```

Attend que l'utilisateur rentre du texte et renvoie directement la réponse.

```
awaitInt(): int
```

Idem mais pour un entier

```
awaitIntInRange(int, int): int
```

Pour un entier compris entre deux bornes

Ces méthodes sont utilisables très simplement et évitent d'avoir à gérer les cas spécifiques et les erreurs liées aux entrées utilisateur partout dans le projet.

3) Déplacement des objets

Rappel : les objets pouvant se déplacer implémentent l'interface MovableObject
Cette interface contient la méthode Move.

Un déplacement se fait en 2 étapes :

- Vérifier si un déplacement est possible
- Déplacer l'objet, mais si ce déplacement doit d'abord déplacer un autre objet, revenir au début de cette phrase avec ce nouvel objet en tête (Récursivité).

Les 2 étapes présentent une récursivité, le joueur doit pouvoir pousser une caisse qui elle-même doit pouvoir pousser une autre caisse.

Chaque étape contient sa propre méthode (Dans la classe **Map** car cette dernière contient les informations utiles sur son contenu).

Etape 1 :

```
private boolean handleMovePosibility(MovableObject object, Vector2 direction) {
      ...
     if (objectAtPredictedPosition instanceof MovableObject)
         return handle ComPatibility((MovableObject) objectAtPredictedPosition, direction);
      ...
}
```

Etape 2 :

```
 public void moveObject(MovableObject object, Vector2 direction) {
      handleMovePosibility(object, direction);
      ...
      if (nextPositionObject instanceof MovableObject)
            moveObject((MovableObject) nextPositionObject, direction);
      ...
}
```

4) Base de données

L'ensemble du code lié à la base de donnée se trouve dans le package map.map Database
On y retrouve deux classes :

- **MapDatabase** : classe principale qui gère la connection et les requêtes à la base de données
- **MapDatabaseAdministration** : pour gérer la base de données (Ajouter/Supprimer map, etc..)

D'un point de vue extérieur au package, il suffit de créer une instance de **MapDatabase**, et l'instance se charge de se connecter et garder la connexion. Il suffit donc ensuite d'utiliser l'API de MapDatabase (ex : get Maps(), getRows(), etc...)

A la manière de EntityFramework en C#, j'ai décidé de lier le contenu de mes tables à des classes java afin de pouvoir après une requête manipuler des objets.

On retrouve donc les classes "nested" MapDatabase.Map et MapDatabase.Row qui ont des attributs identiques aux éléments des tables MAPS et ROWS de la base de données.

5) Affichage haut détails

L'affichage demandé pour le projet étant assez simpliste, j'ai décidé de faire une version d'affichage plus haute en détails.

Une cellule étant originalement un caractère du plateau, elle prend avec ce mode d'affichage 3x5 caractère dans le terminal.

Exemple d'une seule case :

```
┌───┐
|OOO|
└───┘
```

Stockée comme ceci dans un HashMap (Clé : type de l'objet, Valeur : String d'affichage)

```
put(MapObject.ObjectType.BOX, "┌ --- ┐\n|OOO|\n└ --- ┘");
```

Pour chaque ligne du plateau, 3 lignes sont affichées dans le terminal (Chaque ligne utilisant une partie de la String d'affichage de part et d'autre d'un "\n").

```
ngs.get(obj.TYPE).split("\n")[i];
```

6) Tests unitaires

Des tests unitaires sont présents dans le but de tester toutes les méthodes publiques des classes des différents packages.

5/ Remarques

- L'ensemble du projet rendu correspond à ce qui est demandé.
  Seule la création d'un plateau à partir de méthodes simples n'a pas été faite, la création d'un plateau peut donc se faire uniquement à partir d'un fichier ou à partir de la base de données.

- L'utilisation d'instance de MapObject pour chacune des cellules du plateau peut être excessive mais offre une façon simple d'ajouter des éléments de jeu.
  Cela peut donc à l'avenir permettre d'implémenter d'autres règles de jeu ou autre facilement.

- Figer une instance de MapObject à une position et en créer une copie quand on veut la déplacer peut éviter certaines erreurs mais au final c'est moins performant et je pense que ça ne doit pas être une bonne idée de faire ça dans un langage sans garbage collector.