

Alexis Moins

Sokoban

S2A'

24 mai 2021



Objectifs

1. **Classes et paquets** : Cette première partie présente les différentes composantes du programme, et comment elles s'assemblent pour rendre le jeu jouable et le programme cohérent.
2. **Choix et développement** : Cette seconde partie consiste à expliciter mes choix lors de la conception et du développement du programme

Classes et paquets

J'ai décidé de diviser le projet en 7 paquets différents. Avoir une idée précise des paquets, de leur utilité et des classes qu'ils contiennent m'a permis de travailler rapidement avec une idée claire de ce que je souhaitais développer. La plupart des paquets contiennent entre 2 et 3 classes et le plus gros paquet en contient 4.

1. App

Il s'agit du paquet principal du projet et sert de contact avec le joueur / l'utilisateur. Il contient l'unique classe du paquet, `Player`. Cette dernière dispose d'un menu principal reposant sur les trois méthodes `main()`, `displayMainMenu()` et `mainMenu()`. Elle dispose également de la méthode `selectBoard()`, laquelle permet, comme son nom l'indique, de sélectionner un des plateaux disponible dans la base de données pour y jouer.

2. Builders

Ce paquet regroupe les différents constructeurs de plateaux du jeu. Il permettent d'obtenir un plateau jouable à partir de deux sources différentes : un fichier et du texte.

- `BoardBuilder` : il s'agit d'une interface définissant la signature de la méthode `build` qui construit le plateau à l'aide du builder actuel et le renvoi. Toutes les autres classes du paquet builders implémentent cette interface et suivent donc son "contrat" en définissant la méthode `build()`.
- `FileBoardBuilder` : classe permettant de construire un plateau à partir d'un fichier. Pour ce faire, la classe utilise la méthode statique `deserialize()` pour ouvrir un fichier et stocker les lignes de ce dernier dans une liste. Elle dispose aussi de la méthode `convertToTextBuilder()`, utile pour passer d'un constructeur à un autre et ainsi ajouter plus facilement un plateau à la base de données. Enfin, `FileBoardBuilder` utilise bien évidemment la méthode `build()` pour renvoyer un plateau.
- `TextBoardBuilder` : cette classe construit un plateau à partir de chaînes de caractères. Pour ajouter une ligne au constructeur, il suffit de faire appel à la méthode `append()`. La méthode `deserialize()` se chargera ensuite de prendre chacune des lignes et d'en ajouter les éléments dans le plateau. Enfin, la méthode `build()` renverra le plateau construit grâce à la méthode `deserialize()`.

3. Database

Le paquet database regroupe les classes liées à la gestion de la base de données de sokoban. Elle contient 3 classes, Administrator, Database et SQLRequest.

- **Administrator** : la classe Administrator sert d'interface avec l'utilisateur pour la gestion de la base de données au travers des méthodes menu() et displayDatabaseMenu(). Une instance de la classe database est créée par un appel à la méthode manageDatabase() qui est une factory method et renvoie un nouvel Administrator. Depuis cette classe il est possible de lister les plateaux de la base de donnée (listBoards), de les dessiner à l'écran (showBoard), d'ajouter un plateau à la BD (addBoard), et d'en supprimer un (removeBoard).
- **Database** : cette classe opère la connexion avec le fichier contenant la base de données. Elle initialise cette dernière en renvoyant un objet Database avec la méthode newConnection() qui s'occupe également d'insérer les tables BOARS et ROWS dans la base de données si elles n'y sont pas déjà présentes. La classe Database dispose aussi de méthodes utilitaires pour : ajouter un plateau (add), récupérer la liste des ID valides (getListOfValidIDs), supprimer un plateau (remove), récupérer la liste des plateaux (getListOfBoards) et enfin, récupérer un plateau avec son ID (getBoardWithID).
- **SQLRequest** : cette classe est entièrement constituée d'attributs statiques. Ces derniers sont des requêtes SQL préparées sous la forme de chaîne de caractères. Elles seront utilisées dans la classe Database pour toute opération de récupération ou de modification de la base de données.

4. Elements

Ce paquet contient les deux types d'éléments du plateau, Entity et Tile, ainsi qu'un type générique, BoardElement.

- **BoardElement** : la classe BoardElement est abstraite et sert de super-type aux classes Entity et Tile. Elle dispose de méthodes getter de ses attributs protected, disponible pour ses enfants.
- **Entity** : cette classe, héritant de BoardElement, donne naissance aux différentes entités du plateau, à savoir le joueur et les caisses. Elle dispose de deux méthodes factory pour créer et retourner respectivement une nouvelle entité joueur (newPlayer) et une nouvelle entité caisse (newBox). Elle s'appuie également sur la méthode setPosition() pour déplacer une entité.
- **Tile** : la classe Tile, de manière similaire à la classe Entity, possède deux méthodes factory : une pour retourner un nouveau mur (newWall) et une pour retourner une nouvelle cible (newTarget).

5. Exceptions

Ce paquet contient les deux exceptions que j'ai jugé nécessaire à la réalisation du projet Sokoban : `InvalidDirectionException` et `PlayerLeavesException`.

- `InvalidDirectionException` : La classe `InvalidDirectionException` définit une exception lorsque le joueur choisit de se déplacer sans une direction qui n'est pas autorisée sur le plateau. Cette exception hérite de la classe `Exception` de Java.
- `PlayerLeavesException` : cette classe définit une exception lorsque le joueur décide de quitter le niveau. Cette exception hérite également de la classe `Exception` de Java.

6. Game


Le paquet `game` contient le cœur du jeu. Les classes `Board` et `Level` lient tous les composants entre eux pour proposer une application jouable.

- `Board` : la classe `Board` contient, comme son nom l'indique, le plateau du niveau. Elle dispose de méthodes `getter` pour les cibles, le joueur et les caisses. Elle s'appuie également sur les méthodes `addVerticalWall()` et `addHorizontalWall()` pour ajouter des murs sur le plateau, des méthodes `addBox()` et `addTarget()` pour ajouter respectivement une caisse et une cible au plateau. De plus, la classe est chargée d'afficher le plateau à l'écran avec les méthodes `draw()` et `displayColumnNumbers()`. Pour obtenir le caractère à afficher à l'écran, on pourra faire appel à la méthode `getCorrectCharacter()`, laquelle retourne le caractère de l'élément se trouvant aux coordonnées données. Pour se faire, la méthode fera appel à la méthode `findElement` faisant elle-même appel aux méthodes `findEntity()` et `findTile()` qui retournent l'entité (la tuile) au coordonnées données, ou `null` si aucun élément n'a été trouvé.
- `Level` : cette classe représente un niveau du jeu sokoban et contient donc un plateau. Elle dispose des méthodes `boardIsComplete()` pour déterminer si le plateau est terminé ou non (toutes les cibles sont recouvertes par des cibles).

7. Utils

Le paquet `utils` contient toutes les classes "utilitaires" du projet, celles qui proposent un service court implémenté dans d'autres classes d'autres paquets.

- `Color` : cette classe contient en attributs statiques les codes ASCII permettant d'afficher du texte coloré à l'écran. Le choix de créer une nouvelle classe dédiée est motivée par la perspective d'introduire de nouveaux éléments de `gameplay` et



notamment de nouvelles couleurs.

- **Coordinates** : la classe `Coordinates` correspond, comme son nom l'indique, aux différentes coordonnées du plateau. Elle dispose de la méthode `next()`, prenant en paramètre une direction, et retourne la coordonnées suivante, à partir de la coordonnées actuelle et dans la direction donnée. Elle s'appuie aussi sur des `getter y()` et `x()` pour récupérer les coordonnées précises, ainsi que sur les méthodes `hashCode()` et `equals()` pour vérifier l'égalité de deux objets de type `Coordinates`.
- **Direction** : l'énumération `Direction` fournit les quatre directions possibles sur le plateau (nord, sud, est, ouest). Elle dispose des `getter xTranslation()` et `yTranslation()` ainsi que de la méthode statique `correspondingTo()`, laquelle retourne la direction correspondant à un caractère passé en entrée.
- **Type** : l'énumération `Type` représente le type des différents éléments du plateau, soit `BOX`, `WALL`, `PLAYER` ou `TARGET`. Elle dispose des `getter character()` et `hasCollision()` relatifs à la nature du `Type`.
- **Utils** : cette classe propose des méthodes statiques permettant : de demander au joueur de saisir du texte (`getInput`), de demander au joueur de saisir du texte en affichant un message (`askPlayer`), de nettoyer l'écran (`clearScreen`) et d'afficher un message d'erreur (`errorMessage`)

Choix et développement

1. Builders

J'ai décidé, dans la classe `FileBoardBuilder`, de ne pas renvoyer d'erreur en cas de caractère non-reconnu dans le fichier importé (c'est à dire un caractère qui n'est pas lié à un type d'entité ni à un type de tuile). À la place, j'ai remplacé tout caractère invalide par une case vide. Notons qu'il serait sûrement plus judicieux de remplacer les caractères invalides par des murs au cas où le caractère invalide serait en bord de plateau. Dans ce seul cas-ci, remplacer le caractère par une case vide permettrait au joueur de quitter le plateau.

2. Database

Dans ce paquet, j'ai choisi d'implémenter une classe `Administrator`, disposant d'un objet `Database` en attribut d'instance. De ce fait, `Administrator` sert d'interface avec le joueur et dispose d'un menu. La classe fera ensuite appelle à son attribut `Database` pour exécuter les différentes opérations saisies par l'utilisateur.

3. Éléments

J'ai décidé de diviser les éléments du plateau en deux classes, les entités (que l'on peut bouger, qui sont la couche supérieure) et les tuiles (qui sont immobiles, qui sont la couche inférieure). Néanmoins, vu la redondance de certains des comportements de ces deux classes, j'ai décidé de les faire hériter d'une classe abstraite `BoardElement` qui possède les attributs `Type` et `Coordinates` dont bénéficient les sous-classes `Entity` et `Tile`. J'ai ensuite décidé de ne pas découper davantage les classes `Entity` et `Tile`, pour lesquelles il aurait été possible de proposer des classes `Box`, `Target` ou `Wall`. À l'inverse, j'ai choisi d'implémenter le principe de *factory methods* en proposant des méthodes statiques dans `Entity` et `Tile` renvoyant un nouveau type d'élément. Cela donne naissance à un trio de classes aérées, dont chacun des rôles est clairement découpé en s'appuyant sur les autres classes du paquet.

4. Exceptions

J'ai choisi de n'implémenter que deux classes d'exceptions, l'une pour spécifier le départ du joueur, l'autre pour notifier une direction invalide. Étant donné le peu d'exceptions, je n'ai pas trouvé judicieux de les faire hériter d'une classe d'exception générale du type `SokobanException`.

5. Game

Dans le paquet game, j'ai décidé de doter la classe Board de deux listes d'objets, une liste de tuiles appelée TILES et une liste d'entités appelée ENTITIES. Avec l'attribut entité PLAYER, nous disposons de trois couches sur le plateau. Si l'on considère les coordonnées (A, B), on vérifiera d'abord si le joueur est aux coordonnées (A,B) (auquel cas on affiche son caractère), sinon on vérifie si une entité s'y trouve (et on l'affiche) pour enfin vérifier si une tuile est présente. Si ce n'est pas le cas, cela signifie que la case est vide et nous représentons alors un point. En effet, en partant du principe que les plateaux sont toujours entourés par des murs, il est impossible d'avoir des coordonnées en dehors du plateau grâce aux collisions. C'est la logique implémentée par la méthode findElement() et getCorrectCharacter(). En suivant cette organisation, il n'est pas nécessaire d'avoir des collections mappant des coordonnées à des éléments. J'ai également décidé de regrouper les entités dans une même liste ENTITIES, (bien qu'elle ne contienne que des entités de type BOX pour le moment) dans l'optique où l'on souhaiterait ajouter d'autres entités comme des ennemis sur le plateau. Cette organisation des éléments sur le plateau me permet de récupérer tous les BoardElement d'un certain Type grâce aux méthodes stream() et filter(). Concernant la classe Level, j'ai choisi de tester s'il était possible de déplacer des caisses à l'aide d'une méthode récursive. La méthode pour déplacer les caisses dans la direction donnée après le test est également récursive (même si l'on aurait pu se contenter de déplacer la première caisse de la rangée à la suite de la dernière caisse de la rangée).

6. Utils

Dans le paquet Utils, j'ai fait le choix d'instancier avec des paramètres les objets des énumérations Direction et Type. De ce fait, plus besoin de méthodes statiques reposant sur des switch, seulement de getter pour des attributs d'instance. Ainsi, un Type bénéficiera d'un caractère et d'un booléen expliquant s'il aura des collisions avec les autres entités ou tuiles. De la même manière, une direction bénéficiera directement de la valeur correspondant à sa translation sur l'axe des abscisses et sur celui des ordonnées.

Le projet Sokoban m'a permis d'explorer les difficultés qu'un développeur peut rencontrer lorsqu'il est laissé libre de ses choix en matière de développement et de conception. En effet, il m'a fallu beaucoup de temps pour trouver une manière de représenter et de stocker les éléments du plateau qui me convienne. Néanmoins, j'ai le sentiment que ce projet a davantage renforcé mes compétences en développement informatique.