



# Sokoban

Rapport de projet

**Candau Sébastien**  
**S2A''**

## Sommaire:

### I- Présentation

- 1.1 Le jeu
- 1.2 L'interface administrateur

### II- La programmation du jeu

- 2.1 Déroulé d'une partie
- 2.2 Les fichiers
- 2.3 La base de données

### III- Le code

- 4.1 Les packages
- 4.2 La taille du code

## 1. Présentation

### 1.1 Le jeu

Le joueur doit ranger des caisses sur des cases cibles. Il peut se déplacer dans les quatre directions, et pousser (mais pas tirer) une seule caisse à la fois. Une fois toutes les caisses rangées, le joueur valide le niveau.

### 1.2 L'interface administrateur

Notre Sokoban possède une interface administrateur depuis laquelle le joueur peut gérer la mise en place des différents plateaux de jeu.

```
ADMINISTRATION INTERFACE - SEBCANDAU
0. Create new database
1. List boards
2. Show board
3. Add board from file
4. Remove board from database [DANGEROUS]
5. Quit.
Choisissez un numéro :
```

L'administrateur peut alors choisir de créer une nouvelle base de données si ce n'est pas déjà fait, d'y ajouter des plateaux de jeu à partir de fichiers stockés en leur entrant un nom et une description de la difficulté, de les lister et de les supprimer. Nous reviendrons plus en détails sur la base de données dans la partie du compte rendu qui lui est dédiée.

Si des plateaux ont été ajoutés, le joueur n'a plus qu'à lancer une partie et de choisir le plateau sur lequel il veut jouer.

## 2. La programmation du jeu

### 2.1 Déroulé d'une partie

#### 2.1.1 Le menu principal

Premièrement, le joueur fait face à un premier choix. Il doit choisir entre lancer une nouvelle partie ou bien accéder à l'interface d'administration.

```
MENU INTERFACE - SEBCANDAU
0. Play Game
1. Admin Menu
```

Lorsque l'utilisateur choisit l'option 1, une suite d'événements se passent. Premièrement, une connexion à la base de données est établie s'il en existe une. Ensuite, une requête est faite afin d'afficher les différents plateaux de jeu disponibles. Il sera alors demandé à l'utilisateur d'entrer le nom du plateau sur lequel il souhaite jouer. Lorsque le nom du plateau est validé, une partie se lance...

Tout ceci est géré dans la méthode main de la classe Player

```
if (unChoix.equals("0")) {
    base.select_Names();
    String nom = choixNum("entrez le nom du plateau : ");
    String contentBoard = base.select_board(nom);
    PrintWriter writer = new PrintWriter(nom + ".txt", "UTF-8");
    writer.write(contentBoard);
    writer.close();
    FileBoardBuilder file = new FileBoardBuilder(nom + ".txt");
    file.read();
    Board board = file.build();
    Game game = new Game();
    game.start(board);
    File removeFile = new File(nom + ".txt");
    removeFile.delete();
}
```

### 2.1.2 Déroulé d'une partie

Une fois une partie lancée, le plateau de jeu choisi est affiché, il est demandé à l'utilisateur d'entrer une suite de déplacements. Les directions sont alors gérées, si les coordonnées des caisses se trouvent sur les coordonnées des cases gagnantes, alors la partie est gagnée.

```
public void start(Board newBoard) {  
    Board board = newBoard;  
    board.showBoard();  
    while (true) {  
        Scanner sc = new Scanner(System.in);  
        System.out.println("Veuillez saisir une suite de déplacement :");  
        String order = sc.nextLine();  
        char[] actions = new char[order.length()];  
        for (int i = 0; i < actions.length; i++) {  
            int rowPlayer = board.getLinePlayerPosition();  
            int colPlayer = board.getColPlayerPosition();  
            int rowBox = board.getLineCaissePosition();  
            int colBox = board.getColCaissePosition();  
            actions[i] = order.charAt(i);  
            if (actions[i] == 'R') {  
                Direction.right(board, rowPlayer, colPlayer);  
            }  
            if (actions[i] == 'D') {  
                Direction.down(board, rowPlayer, colPlayer);  
            }  
            if (actions[i] == 'U') {  
                Direction.up(board, rowPlayer, colPlayer);  
            }  
            if (actions[i] == 'L') {  
                Direction.left(board, rowPlayer, colPlayer);  
            }  
            board.showBoard();  
            System.out.println("");  
        }  
        board.haveWinPos();  
        if (win(board)) {  
            System.out.println("Niveau accompli !");  
            break;  
        } else {  
            System.err.println("Niveau non accompli !");  
            break;  
        }  
    }  
}
```

Création d'un nouveau plateau et affichage de celui-ci

On assigne une suite de déplacements dans un tableau de String

En fonction de chaque direction, une action est effectuée

On vérifie si il y a un gagnant

### 2.1.3 Les directions

Les directions sont gérés par la classe Direction. Le programme passe la chaîne de caractères entrée caractère par caractère, pour chaque caractères correspondant à une direction, une méthode de la classe direction est appelée.

Exemple pour la direction "Left" :

```

/**
 *
 * @param board
 * @param rowPlayer
 * @param colPlayer
 */
public static void left(Board board, int rowPlayer, int colPlayer) {
    if (board.getCasé(rowPlayer, colPlayer - 1).toString().equals("C")) {
        int rowBox = rowPlayer;
        int colBox = colPlayer - 1;
        if (!board.getCasé(rowBox, colBox - 1).toString().equals("#")) {
            board.setCaissePosition(rowBox, colBox - 1);
            board.setCasé(rowPlayer, colPlayer, "empty");
            board.setPlayerPosition(rowPlayer, colPlayer - 1);
        }
    }
    if (board.getCasé(rowPlayer, colPlayer - 1).toString().equals(".")) {
        board.setCasé(rowPlayer, colPlayer, "empty");
        board.setPlayerPosition(rowPlayer, colPlayer - 1);
    }
}

```

La fonction de déplacement Left prend en paramètres les coordonnées du joueur. En fonction de ces coordonnées, à l'aide de différents Get et Set de la classe Board (permettant de récupérer les positions des caisses par exemple), la case du joueur est transformée en case vide mais la case à sa gauche est remplacée par le joueur.

### 2.1.4 Le plateau de jeu

Un plateau de jeu est défini par la classe Board. Il possède comme attributs un tableau à deux dimensions de cases, un nombre de lignes et un nombre de colonnes variables. Il est affiché sous forme de chaîne de caractères grâce à la méthode showBoard().

```

/**
 * affiche le plateau
 *
 */
public void showBoard() {
    for (int j = 0; j < nbRow; j++) {
        for (int i = 0; i < nbCol; i++) {
            System.out.print(" " + board[j][i]);
        }
        System.out.println("");
    }
}

```

### 2.1.5 Les cases

Enfin, les cases de notre plateau sont définies par la classe Case. Une case possède comme attributs des coordonnées ainsi qu'un type sous forme de chaîne de caractères.

En fonction de son type (joueur, vide, caisse ou case gagnante), une case est représentée par des caractères différents.

```


/**
 * retourne la case en chaine de caractère. Elle diffère selon le type
 * @return
 */
@Override
public String toString() {
    String c = ".";
    if ("wall".equals(this.type)) {
        c = "#";
    }
    if ("caisse".equals(this.type)) {
        c = "C";
    }
    if ("player".equals(this.type)) {
        c = "P";
    }
    if ("winPos".equals(this.type)) {
        c = "X";
    }
    if ("empty".equals(this.type)) {
        c = ".";
    }
    return c;
}

```

## 2.2 Les fichiers

Comme demandé dans le cahier des charges, notre Sokoban charge ses plateaux de jeu à partir de fichiers textes contenant une représentation sous forme de caractères du plateau

Tous les plateaux sont stockés dans le répertoire ressources/ et leur contenu doivent être construits de cette façon pour être reconnus :

 fichier\_1 - Bloc-notes

Fichier Edition Format

```

#####
#x.x#...#
#...C..P.#
#...C...#
#.....#
#####|

```

La construction du plateau à partir des fichiers est gérée dans la classe FileBoardBuilder. Premièrement, la classe charge le fichier et implémente de cette façon ses différents attributs. Ensuite, pour lire le fichier il faut appeler la méthode Read() de la classe. Elle lit ligne par ligne notre fichier et assigne ainsi le contenu à un attribut de type String la représentation du plateau.

La méthode build doit ensuite être appelée, elle doit être capable de reconnaître chaque caractère de notre String et de construire le plateau en fonction.

```

/**
 * construit le plateau de jeu
 *
 * @return
 * @throws IOException
 */
public Board build() throws SQLException, IOException {
    Board board = new Board(lines.size(), lines.get(1).length());
    for (int i = 0; i < lines.size(); i++) {
        for (int j = 0; j <= lines.get(1).length() - 1; j++) {
            if (lines.get(i).charAt(j) == '#') {
                board.setCase(i, j, "wall");
            } else if (lines.get(i).charAt(j) == 'P') {
                board.setCase(i, j, "player");
            } else if (lines.get(i).charAt(j) == 'C') {
                board.setCase(i, j, "caisse");
            } else if (lines.get(i).charAt(j) == 'x') {
                board.setCase(i, j, "winPos");
            } else if (lines.get(i).charAt(j) == '.') {
                board.setCase(i, j, "empty");
            } else {
                System.err.println("Le fichier choisi est invalide.");
                Player.aGame();
            }
        }
    }

    return board;
}

```

## 2.3 La base de données

La base de données est représentée dans la classe DataBase. Chaque méthode est appelée en fonction du besoin de l'utilisateur. Chaque méthode représente une requête utile pour l'utilisateur.

### 2.3.1 Création de la table Board

Notre base de données possède une table Board qui est créée en même temps que la base. La table BOARD possède trois occurrences "NAME" le nom de plateau, "BOARD" la représentation du plateau, "difficulty" une petite description du niveau de difficulté de la table.

```

/**
 * Créer la table BOARDS avec ses trois occurrences NAME, BOARD, difficulty
 */
public void createTable() {
    this.connect();
    if (this.connection != null) {
        try {
            String SQL = "CREATE TABLE BOARDS (NAME TEXT NOT NULL"
                + ",BOARD TEXT NOT NULL, difficulty TEXT NOT NULL);";
            Statement stt = this.connection.createStatement();
            stt.executeUpdate(SQL);
        } catch (SQLException ex) {
            Logger.getLogger(DataBase.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}

```

### 2.3.2 Ajout de valeurs à notre table

L'utilisateur ajoute des plateaux de jeu à partir de fichiers à notre base de données. Pour cela, une simple requête permet d'afficher l'ensemble des fichiers contenus dans le répertoire ressources/.

```
/**
 * Méthode permettant de choisir un fichier contenu dans le répertoire
 * ressource
 *
 * @return
 */
public String chooseFile() throws IOException, SQLException {
    File repertoire = new File("./ressources");
    String liste[] = repertoire.list();

    if (liste != null) {
        for (int i = 0; i < liste.length; i++) {
            System.out.println(i + " " + liste[i]);
        }
    } else {
        System.err.println("Nom de repertoire invalide");
    }
    String choix = Player.choixNum("Choisissez un numéro : ");
    int choixNum = 0;
    try {
        choixNum = Integer.parseInt(choix);
    } catch (NumberFormatException e) {
        System.err.println("ERREUR - retour au menu.");
        Player.aGame();
    }
    return "ressources/" + liste[choixNum];
}
```

L'utilisateur entre le numéro du fichier, il doit entrer un nom pour le plateau ainsi qu'une description de la difficulté puis toutes ces informations sont alors envoyées à notre base de données. Une requête est alors effectuée et notre plateau est ajouté.

```
private void addBoardFromFile() throws FileNotFoundException, IOException, SQLException {
    FileBoardBuilder file = new FileBoardBuilder(chooseFile());
    file.read();
    Board board = file.build();
    String baseName = Player.choixNum("choisissez un nom pour votre plateau: ");
    String difficulty = Player.choixNum("entrez une brève description de la difficulté: ");

    System.out.println(board.toString());
    base.connect();
    base.insert(baseName, board.toString(), difficulty);
}
```

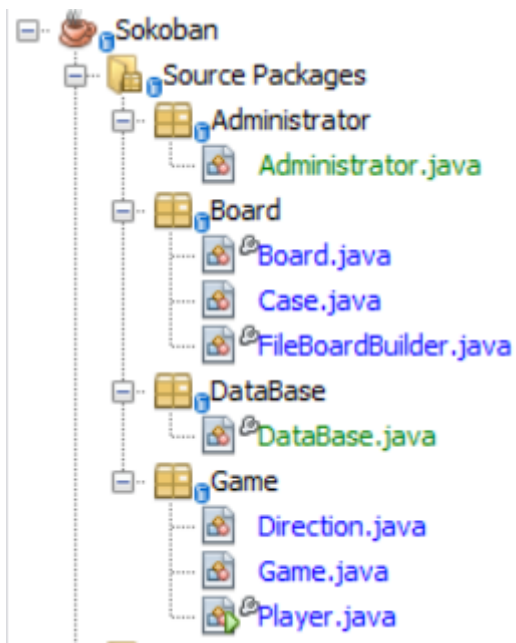
### 2.3.3 Exceptions SQL

En cas d'erreur SQL diverses (par exemple pas de base de donnée créée, etc..) c'est la méthode exception() qui est appelée. Elle permet tout simplement d'indiquer au joueur qu'il y a une erreur et appelle la méthode Player.Game(), ce qui fait revenir au menu d'accueil.

```
private void exception() throws IOException, SQLException {
    System.err.println("ERREUR - retour au menu principal");
    Player.aGame();
}
```

## 4. Le code

### 4.1 Les packages



### 4.1 La taille du code

Nombre de classes : **8**

Nombre de fonctions : **49**

Total de lignes: **1044**