

A dark blue vertical bar on the left side of the page. A blue arrow points to the right from the bar, containing the date.

29/01/2023

Rapport Sokoban

Several thin, curved lines in dark blue and light grey originate from the bottom left corner and sweep upwards and to the right.

Alexandre Cancel et Ganesh Jhugroo

Table des matières

Introduction	2
I. Fenêtre et image de dialogue	3
JFrame, JDialog	3
JPanel	3
Fonction dessiner	4
Classe Grille	5
Base de données	5
Les problèmes	6
Explications	6

introduction

Lors de notre projet, nous étions en binôme. Ainsi, nous avons dû collaborer ensemble pour effectuer notre travail. Ce projet s'appuie sur le projet que nous avons dû faire l'année dernière en créant un jeu SOKOBAN sur Qt Creator dans le langage de programmation C++ sauf que cette année, nous devons le réaliser à l'aide du langage de programmation JAVA.

Dans ce rapport, nous allons expliquer les choses que nous avons dû faire de manière différente pour passer du C++ au JAVA tout en expliquant les difficultés que nous avons rencontrées au cours de notre projet.

I. Fenêtre et image de dialogue

JFrame, JDialog

Lors de la création de nos fenêtres, nous devons décider quel type de fenêtre nous voulions créer. La première fenêtre que nous avons décidé de créer a été notre fenêtre principale, il s'agit pour elle d'un **JFrame** tandis que toutes nos autres fenêtres sont des fenêtres de dialogue, nous avons donc utilisé **JDialog**. **JDialog** et **JFrame** viennent tous les deux de la bibliothèque **javax.swing**

```
package Sokoban;  
  
import javax.swing.JFrame;  
  
public class FenetrePrincipale extends JFrame {
```

Cette partie-là était plutôt simple à réaliser et à comprendre, car tout était expliqué dans le cours.

JPanel

Tout comme **JDialog** et **JFrame**, **JPanel** vient de la bibliothèque **javax.swing**.

Contrairement au langage de programmation C++, lors de la création d'une fenêtre, il faut aussi créer un panel propre à chaque fenêtre pour pouvoir afficher les éléments que nous voulons faire apparaître grâce aux différentes fonctions dessinées présentes dans notre code.

Toutes nos classes **JPanel** ont été programmées de la même façon.

```
package Sokoban;  
  
import javax.swing.JPanel;  
import java.awt.Graphics;  
  
@SuppressWarnings("serial")  
public class PanelPrincipal extends JPanel {  
  
    // Référence sur la fenêtre principale.  
    private FenetrePrincipale maFenetrePrincipale;  
  
    /**  
     * Create the panel.  
     */  
    public PanelPrincipal(FenetrePrincipale FenetrePrincipale) {  
        maFenetrePrincipale = FenetrePrincipale;  
    }  
  
    /**  
     * Gestionnaire d'événement associe a l'événement "paint" du panel.  
     */  
    @Override  
    public void paintComponent(Graphics g) {  
        super.paintComponent(g);  
        // On appel la méthode dessiner de la fenêtre principale...  
        maFenetrePrincipale.dessiner(g);  
    }  
}
```

Par exemple pour le panel de la fenêtre principale, premièrement, nous avons initialisé le paramètre **maFenetrePrincipale** venant de la classe **FenetrePrincipale**. Ensuite, nous avons

créé la fonction **PanelPrincipal()** qui prend en paramètre **FenetrePrincipale** de classe **FenetrePrincipale**, cette fonction permet que **maFenetrePrincipale** prenne la "valeur" de **FenetrePrincipale**. Puis nous avons dû ajouter la fonction **paintComponent()**. Cette fonction permet d'afficher les éléments présents dans notre fonction dessiner de la classe **FenetrePrincipale**, dans cette fonction, nous utilisons **Graphics** de la bibliothèque **java.awt** et nous implémentons la fonction comme montrée au-dessus.

Fonction dessiner

Lors de notre projet, nous avons implémenté un grand nombre de fonctions **dessiner()**. Dans un premier temps pour dessiner le fond sur nos différentes fenêtres puis dans un second temps pour dessiner les différents éléments de notre jeu sur notre grille comme par exemple les caisses ou encore notre personnage. En fonction de si nous implémentons la fonction pour une fenêtre ou un élément de notre grille, notre fonction **dessiner()** est différente. Mais certaines choses restent les mêmes, dans les deux cas, nous utilisons **ImageIcon** de la bibliothèque **javax.swing** pour importer notre image depuis le disque de notre ordinateur.

Comme par exemple dans l'image ci-dessous, qui correspond à notre classe caisse :

```
package Sokoban;

import java.awt.Graphics;
import javax.swing.ImageIcon;

public class caisse extends mobile{

    //attribut
    protected boolean coloree = false;
    protected ImageIcon c;
    protected ImageIcon cm;

    //méthodes
    public caisse() {
        super();
        coloree = false;
    }

    public caisse(int x, int y, int hauteur, int largeur, boolean coloree) {
        super(x, y, hauteur, largeur);
        this.coloree = coloree;
        c = new ImageIcon("../src/Sokoban/caisse1.png");
        cm = new ImageIcon("../src/Sokoban/caisse.png");
    }

    public boolean getColoree() {
        return coloree;
    }

    public void setColoree(boolean coloree) {
        this.coloree = coloree;
    }

    public int getType() {
        if (!coloree) {
            return 1;
        }
        else {
            return 5;
        }
    }

    public void dessiner(Graphics g) {
        if (coloree) {
            g.drawImage(cm.getImage(), p.getX()*30,p.getY()*30,largeur, hauteur, null);
        }
        else {
            g.drawImage(c.getImage(), p.getX()*30,p.getY()*30,largeur, hauteur, null);
        }
    }
}
```

Dans la fonction **dessiner()**, nous remarquons que nous utilisons aussi **Graphics** comme dans les **JPanel**, mais cette fois, nous utilisons **drawImage()** pour afficher notre image en fonction de différents paramètres.

Tandis que pour nos fenêtres, notre fonction dessiner est implémentée comme présentée juste en dessous pour la classe **FenetrePrincipale**.

```
public void dessiner(Graphics gr) {  
    Graphics bufferImage;  
    Image offscreen;  
    offscreen = createImage(this.getContentPane().getWidth(), this.getContentPane().getHeight());  
    bufferImage = offscreen.getGraphics();  
    bufferImage.drawImage(fond.getImage(), 0, 0, 610, 610, this);  
    gr.drawImage(offscreen, 0, 0, null);  
}
```

Dans ce cas-là, nous remarquons quelques différences avec notamment un **bufferImage** et un **offscreen**, mais nous utilisons aussi **drawImage()** comme pour les éléments de notre grille.

Dans cette partie-là, la chose que nous avons trouvée la plus compliquée à réaliser a été de trouver la bonne façon de noter un chemin pour que nous puissions lancer notre code aussi bien sur l'ordinateur de Ganesh que celui d'Alexandre. La solution était de mettre ".." avant "/src/Sokoban/xxx.xxx". Nous avons rencontré ce problème tout au long de notre projet, à chaque fois que nous avons un chemin provenant de l'un de nos ordinateurs, nous avons dû faire cette manipulation.

Classe Grille

La classe sur laquelle nous avons eu le plus de difficulté a été sans aucun doute la classe **grille**, car cette classe permet de lire nos fichier .txt qui correspondent à nos différents niveaux. Nous avons cherché longtemps un moyen de lire notre fichier de la manière la plus facile possible pour ensuite créer notre grille. Dans un premier temps, nous lisions notre fichier ligne par ligne que nous mettions dans un tableau pour ensuite parcourir le tableau pour créer notre grille. Malheureusement, nous nous sommes vite rendu compte que cette méthode n'était pas du tout optimale et que nous avons plusieurs problèmes que nous n'arrivions pas à résoudre. Nous avons donc décidé de lire notre fichier .txt caractère par caractère. Pour cela, nous avons utilisé **Scanner** de la bibliothèque **java.util** pour ouvrir notre fichier et nous avons mis tous les éléments de notre fichier dans une **ArrayList<>** ensuite nous avons utilisé les **.get()** pour récupérer les éléments et les ajouter dans les **ArrayList<>**, **grille** et **grilleM** qui correspondent respectivement à notre grille d'élément immobile et à notre grille d'élément mobile.

```
public grille(String nomfile) throws FileNotFoundException{  
    entiers = new ArrayList<Integer>();  
    try (Scanner scan = new Scanner(new File(nomfile))) {  
        while (scan.hasNextInt()) {  
            entiers.add(scan.nextInt());  
        }  
        hauteur = entiers.get(0);  
        largeur = entiers.get(1);  
        dim = entiers.get(2);  
        grille = new ArrayList<immobile>();  
        grilleM = new ArrayList<mobile>();  
    }  
}
```

Base de données

Nous avons décidé de nous occuper de cette partie une fois tout notre code fini et que nous étions sûrs qu'il marche correctement. Malheureusement, pour cette partie-là, nous avons

rencontré énormément de problèmes que nous n'avons pu résoudre, c'est pourquoi, nous n'avons pu créer de bases de données et tester notre code.

Problèmes

Nous avons principalement travaillé sur l'ordinateur personnel d'Alexandre qui possède un Macbook Air de 2017. Or avec un tel type d'ordinateur, il nous était impossible d'installer un outil de création de base de données. Nous avons donc décidé d'essayer sur l'ordinateur personnel de Ganesh malheureusement, alors qu'il nous était avant possible de lancer notre code sur son ordinateur, une fois que nous avons voulu nous occuper de la base de données, notre code ne marchait pas sur cet ordinateur alors qu'il fonctionnait très bien sur l'ordinateur d'Alexandre ou d'autres camarades. N'arrivant pas à résoudre ce problème dont nous ne comprenions pas la raison, nous nous sommes résignés à demander de l'aide à d'autres camarades pour nous expliquer comment ils ont réalisé la partie sur les bases de données afin de comprendre comment nous aurions pu faire.

Explications

Tout d'abord, il faut choisir un outil capable d'intégrer une base de données au programme Sokoban, en prenant bien en compte la comptabilité à l'ordinateur., il en existe plusieurs. Il faut ensuite installer cet outil sur l'ordinateur et lancer l'application. Une fois l'application lancée, on vérifie que les serveurs fonctionnent correctement, puis on peut enfin créer une base de données. Pour finir, il faut ajouter plusieurs lignes de code pour faire le lien entre le programme Sokoban et la base de données à partir de la bibliothèque **java.sql**

```
public void appelBDD(String nom, int niv, int compteur) {  
    String BDD = "Sokoban";  
    String url = "jdbc:mysql://localhost:3306/" + BDD;  
    String user = "root";  
    String passwd = "";  
    try {  
        Class.forName("com.mysql.jdbc.Driver");  
        Connection conn = DriverManager.getConnection(url, user, passwd);  
        Statement stm=conn.createStatement();  
        System.out.println("Connecter");  
        String query = "INSERT INTO resultat (`Pseudo`, `Niveau`, `NbCoups`) " + "VALUES ('"+nom+"',"+String.valueOf(niv)+", "+String.valueOf(compteur)+")";  
        stm.executeUpdate(query);  
    } catch (Exception e){  
        e.printStackTrace();  
        System.out.println("Erreur");  
        System.exit(0);  
    }  
}
```

Avant d'insérer le résultat d'un joueur dans la base de données, nous devons d'abord nous connecter à la base de données à partir de son url avec le nom d'utilisateur et le mot de passe. Une fois cela fait, nous pouvons inscrire les différentes informations dans la base de données pour chaque utilisateur.