Clément Gresh
M1 IMPAIRS
2021-2022

Projet Interfaces Graphiques
Plombier

Table des Matières

Table des Figures

I. INTRODUCTION

Le but de ce projet est de programmer en *java* un jeu de type « Plombier » en utilisant les composant *SWING*. Après avoir choisi un niveau, une grille s'affiche avec des sources de différentes couleurs ainsi qu'une réserve contenant des tuyaux. Il faut alors :

- relier chaque source à au moins une autre de même couleur.
- ne relier aucune source à une autre de couleur différente.
- qu'aucun tuyau connecté à une source ne débouche sur rien.

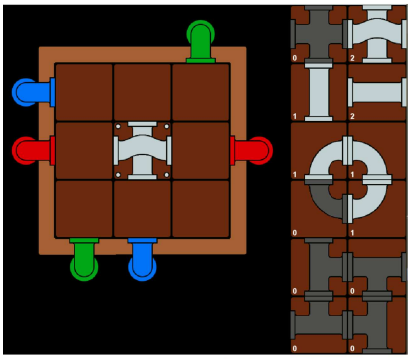

Figure 1: plateau à l'initialisation

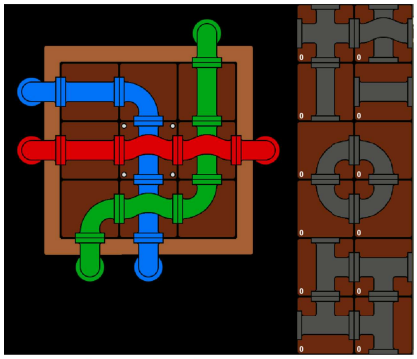

Figure 2: plateau gagnant

Ce projet de Programmation Orientée Objet suit une structure Modèle-Vue-Contrôleur (MVC). Ce rapport détaillent le fonctionnement de chacune de ces parties ainsi que les choix d'implémentation qui ont été faits.

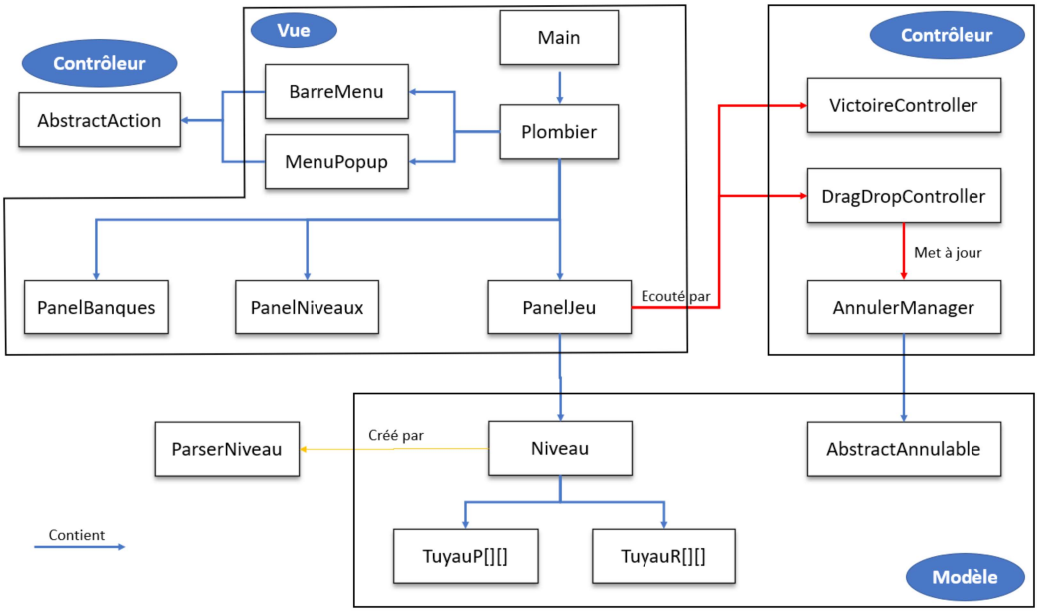

Figure 3: arborescence du projet

II. MODÈLE

1. Tuyaux et énumérations

Un niveau est principalement composé d'un tableau représentant le plateau de jeu et d'un autre représentant la réserve. Ils contiennent tous deux des objets héritant de la classe Tuyau, respectivement :

- des objets *TuyauP* qui possèdent les attributs « fixe », indiquant si un tuyau est inamovible, et « visite », servant lors de la mise à jour des couleurs. Ces attributs sont en fait des tableaux à une composante, excepté pour le tuyaux de type *Over* pour lesquels il y en a deux.
- des objets *TuyauR* qui ont un attribut « nombre » permettant d'indiquer combien de tuyaux de chaque catégorie sont disponibles dans la réserve.

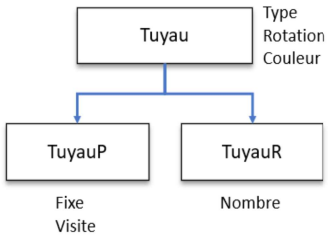

Figure 4: classes des tuyaux

Les attributs communs aux différents types de tuyaux sont décrits à l'aide de trois classes d'énumération :

- *Type* qui correspond aux différents modèles de tuyaux (*line*, *fork*, etc.).
- *Dir* qui correspond à leur rotation (e.g. ouest pour 3 quarts de tour).
- *Couleur*.

La classe *Type* définit également un tableau statique d'objets de classe *Dir* contenant les ouvertures de chaque modèle de tuyau. Pour savoir si un tuyau avec rotation a une ouverture dans une direction donnée, on demande donc au modèle correspondant après avoir soustrait la rotation. Par exemple, pour savoir si un tuyau *fork1* a une ouverture à l'est, on consulte le tableau pour voir si le modèle de *fork* a une ouverture au nord.

2. Niveau et parseur

La création d'un niveau se fait à l'aide d'une classe abstraite *ParserNiveau*. Ce traitement lui a été délégué car il est spécifique à la syntaxe utilisée dans les fichiers *level.p*. Celui-ci crée tout d'abord la réserve sous la forme d'un tableau 6x2 contenant les différents types de tuyaux (tous ayant alors un effectif de 0). Il lit ensuite la hauteur et la largeur du plateau et crée un tableau le représentant en utilisant ces dimensions. Puis, il parcourt le « plateau » contenu dans le fichier. A chaque tuyau trouvé :

- s'il est fixe, celui-ci est ajouté au plateau.
- sinon, l'effectif du tuyau correspondant est augmenté dans la réserve.

Une méthode statique *type* détermine le type du tuyau rencontré (e.g. 'O' correspond au type *OVER*) tandis que la méthode statique *couleur* détermine la couleur d'une source lors de son ajout au plateau (e.g. 'R' correspond à *ROUGE*).

Une fois les variables largeur, hauteur, réserve et plateau enregistrées, il devient possible de créer un niveau.

```
public class Niveau {
    private final int hauteur;
    private final int largeur;
    private final TuyauP[][] plateau;
    private final TuyauR[][] reserve;
    private boolean victoire = false;
```

Figure 5: attributs d'un Niveau

3. Mise à jour des couleurs et victoire

Lorsque cela est demandé par un contrôleur, le niveau met à jour les couleurs des tuyaux du plateau. Il détermine par la même si la configuration courante correspond à une victoire. Pour cela, on parcourt les tuyaux en partant des sources. La couleur courante est alors celle de la source.

Pour chaque tuyau, on met l'attribut « *visite* » à vrai puis on examine ses orientations de sortie (i.e. toutes ses orientations sauf celle d'entrée) et on vérifie qu'elles donnent bien sur l'ouverture d'un autre tuyau. Les méthodes *Dir.ligne* et *Dir.colonne* permettent de déterminer les coordonnées sur le plateau de la case suivante. Par exemple, si une ouverture est au sud, on passe à la ligne suivante en restant sur la même colonne.

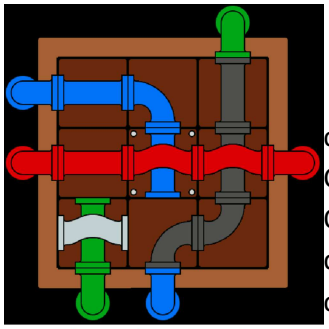

Figure 6: coloration

Si le nouveau tuyau n'a pas déjà été visité ou bien a été visité mais est d'une couleur différente (autre que le noir), on le repeint avec la couleur courante. Cette dernière passe à noir lorsqu'on arrive à une source de couleur différente. On refait alors le parcours dans le sens inverse. Ces conditions permettent d'éviter une boucle infinie tout en garantissant que la couleur noire l'emporte en cas de conflit.

Les tuyaux de type *OVER* reçoivent un traitement particulier. Il y a en effet des attributs « visite » et « couleur » pour chacune de leurs deux composantes et seuls ceux la composante effectivement traversée sont mis à jour. De même, seule l'ouverture de sortie de la composante visitée est examinée pour la recherche du tuyau suivant.

Il n'y a pas victoire (i.e. l'attribut *victoire* est mis à faux) si :

- une source est reliée à une source de couleur différente.
- un tuyau a une ouverture ne débouchant sur rien (y compris pour une source).

Ces conditions garantissent que toutes les sources doivent être reliées à au moins une autre de même couleur. Il est à remarquer que, puisqu'on parcourt les tuyaux en partant des sources, il peut y avoir victoire même si un tuyau présent sur le plateau n'est connecté à rien. Il apparaît alors en blanc. Mais il semble peu utile d'exiger de le retirer avant de déclarer la victoire. De même, il n'est pas nécessaire d'avoir utilisé tous les tuyaux de la réserve pour gagner.

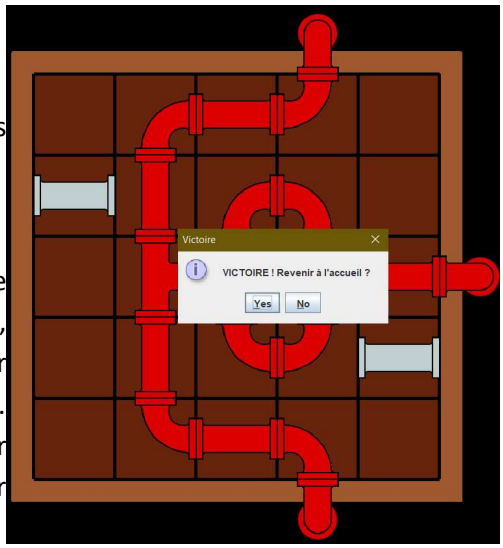

Figure 7: victoire et tuyaux déconnectés

Comme c'est le cas dans certains jeux, notamment de stratégie, il est ici laissé la possibilité de rester dans la partie après avoir gagné. Il suffit alors d'un clic gauche n'apportant aucune modification ou bien de revenir à une configuration gagnante pour se voir à nouveau proposer de passer au niveau suivant ou de revenir à l'accueil.

III. VUE

1. Plombier et accueils

Les chemins vers les niveaux et les images, qui sont des paramètres externes, sont transmis aux composants en ayant besoin via la classe Main. Comme les chemins peuvent différer en fonction du compilateur ou de l'IDE, le Main fournit en fait le premier qui fonctionne parmi une liste donnée et lance une exception sinon. De même, le nombre de banques de niveaux disponibles est indiqué dans une variable statique NBR_BANQUES.

La classe Plombier est le panel principal de l'application. Il crée les menus, le *UndoManager* et permet d'afficher et d'intervertir les *JPanel* ci-dessous :

- *PanelBanques* qui contient un bouton par banque de niveaux disponible.
- *PanelNiveaux* qui contient un bouton par niveau disponible (i.e. par fichier présent dans le dossier de la banque choisie).
- *PanelJeu* qui contient la grille correspondant au niveau choisi.

PanelBanques et *PanelNiveaux* étant similaires, ils dérivent tous deux d'une classe *AbstractAccueil*. Elle définit, outre les paramètres utiles à l'affichage (*layout*, taille, etc.), une méthode creerBouton. C'est ensuite une boucle dans le constructeur de chacune des deux classes qui est responsable d'appeler cette méthode. Cela permet que la modification du nombre de banques ou de niveaux (et donc de boutons à afficher) ne nécessite pas de modification du code.

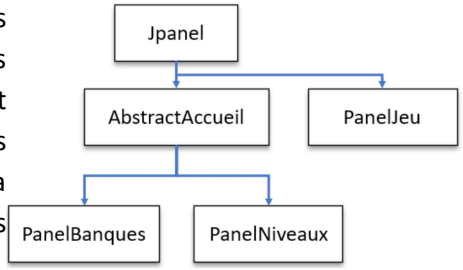

Figure 8: les 3 panels d'affichage

La classe Plombier permet également de déterminer s'il y a un niveau suivant dans la banque après le niveau en cours ou encore d'afficher une fenêtre de confirmation (e.g. lors d'une victoire). Tant que cette fenêtre est ouverte, l'application est maintenue au premier plan. Une méthode *pressAlt* permet alors de sélectionner YES ou NO à l'aide des touches Y et N sans avoir à appuyer sur Alt en même temps.

```
// Affiche le panel selectionne et met a jour les boutons du menu
private void afficher(JPanel panel, boolean retour, boolean recommencer)

// Determine s'il y a un niveau suivant dans la banque de niveau
public boolean isThereNextLevel() {...4 lines }

// Demande confirmation avant la cloture de la frame
public void confirmClose() {...5 lines }

// Ouvre une fenetre de confirmation de type YES/NO
static public int confirmation(JFrame frame, String titre, String texte)

// "Presse" le bouton Alt
public static void pressAlt() {...8 lines }
```

Figure 9: quelques méthodes de la classe Plombier

2. Fenêtre de jeu

La classe *PanelJeu*, héritant de *JPanel*, instancie un niveau à l'aide de *ParserNiveau*. Elle détermine ensuite sa propre largeur ainsi que le taille d'une case en fonction de sa hauteur (constante) et du nombre de cases du plateau. Cela lui permet de créer, à partir du fichier *pipes.gif*, une image d'arrière-plan (avec un plateau et une réserve vides) qui pourra ensuite être utilisée tout au long de la partie. Des méthodes utilitaires *typeCase* et *rotations* permettent de déterminer les caractéristiques des cases, bordures et coins du plateau.

Par la suite, à chaque appel de la méthode *paintComponent*, le tableau *plateau* est parcouru et, pour chaque tuyau rencontré, l'image correspondante est dessinée sur l'arrière plan. De même pour la réserve où les tuyaux sont dessinés en noir lorsque leur effectif est nul et en blanc sinon.

Les classes d'énumération *Type* et *Couleur* aident à définir le modèle mais sont aussi liées à la vue. En effet, leurs valeurs sont énumérées dans le même ordre qu'elles apparaissent sur l'image pipes.gif : BLANC correspond aux tuyaux de la première ligne, SOURCE à ceux de la première colonne, etc. Connaître le *Type* et la *Couleur* d'un tuyau permet donc de récupérer facilement l'image qui le représente.

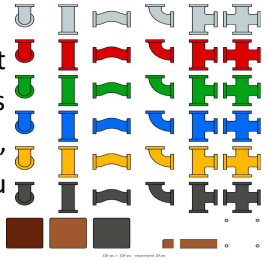

Figure 10: pipes.gif

Il ne reste ensuite qu'à appliquer une rotation pour les tuyaux le nécessitant ou à combiner les deux images dans le cas d'un *Over*. La classe abstraite *ImageUtils* regroupe les méthodes statiques permettant d'effectuer ces opérations.

3. Menus

Les classes d'actions, dérivant de AbstractAction, permettent ici de factoriser les propriétés (nom, icône, action à effectuer, etc.) des boutons des différents menus. Les actions de retour à l'accueil, recommencer le niveau et quitter le jeu sont créés dans le panel Plombier. Il les active et désactive en fonction du panel affiché (accueils ou jeu).

Les actions « annuler » et « rétablir » sont quant-à-elles créées par le manager *AnnulerManager*. Il gère leur activation en fonction du panel affiché et des modifications du plateau de jeu. La liste des actions annulables est remise à zéro à chaque changement de panel.

Toutes ces actions sont envoyées aux menus (barre de menu et menu popup) qui les intègrent.

IV. CONTRÔLEUR

1. Drag & Drop

Le contrôleur correspondant à la fonctionnalité de Drag & Drop est ajouté en tant que *MouseListener* et *MouseMotionListener* au panel de jeu. Il étend la classe *MouseAdapter* afin de n'avoir à redéfinir que les méthodes qui lui sont utiles : *mousePressed*, *mouseDragged* et *mouseReleased*.

Dans *mousePressed*, on détermine tout d'abord où le clic a eu lieu. S'il s'agit :

- d'une case de la réserve, on vérifie qu'un tuyau est disponible (effectif supérieur à 0).
- d'une case du plateau, on vérifie la présence d'un tuyau non fixe.

Si c'est le cas, on crée un nouveau *Tuyau* par copie. On crée alors l'image correspondante (en blanc) qu'on envoie à la vue en tant *qu'ImageIcon* avec les coordonnées auxquelles elle doit être affichée. La méthode *mousePressed* se contente ensuite de mettre à jour les coordonnées de cette image et de demander le rafraîchissement de la vue.

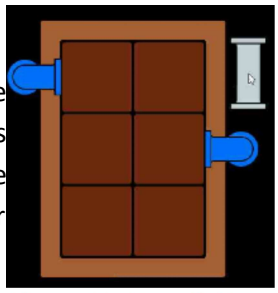

Figure 11: Drag & Drop

Dans *mouseReleased*, si un tuyau est en train d'être déplacé, on détermine où le clic a été relâché. S'il s'agit d'une case vide du plateau, on ajoute le tuyau déplacé à celui-ci – il s'agit en fait d'une copie puis d'une suppression. Sinon, on le renvoie dans la réserve. Dans les deux cas, on détermine d'abord sur quelle case le tuyau doit être envoyé. Puis on déplace l'image de Drag&Drop en ligne droite jusqu'à celle-ci en mettant en pause le *thread* à chaque itération pour que le mouvement soit visible. C'est le *thread* principal qui est mis en pause afin d'éviter les erreurs que pourraient générer des actions ayant lieu avant la fin du mouvement et donc avant la mise à jour des données.

2. Undo/redo

La méthode *mouseReleased* du contrôleur de *Drag* & Drop est également en charge d'ajouter des actions annulables au *AnnulerManager* (qui hérite de *UndoManager*). Une telle action est créée lors des déplacements d'un tuyau :

- de la réserve au plateau.
- du plateau à la réserve.
- du plateau à une case différente de celui-ci.

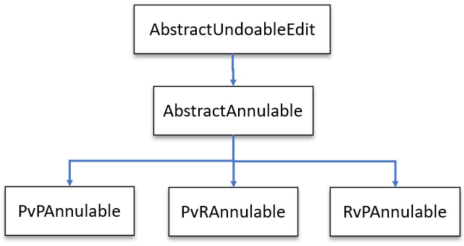

Figure 12: classes d'actions annulables

Il y a donc trois classes d'actions annulables. Une classe mère *AbstractAnnulable*, héritant de *AbstractUndoableEdit*, permet de factoriser ce qu'elles ont en commun. Elle définit notamment des méthodes *ajouterTuyau* et *enleverTuyau* qui peuvent aussi bien agir sur le plateau que sur la réserve. Enfin, une méthode *maj* permet de mettre à jour la vue et de déclarer la victoire s'il y a lieu.

```
protected void ajouterTuyau(int l, int c, int zone){
    if(zone == PLATEAU) niveau.getPlateau()[l][c] = new TuyauP(tuyau);
    else if(zone == RESERVE) niveau.getReserve()[l][c].augmenter();
}
```

Figure 13: méthode ajouterTuyau de la classe AbstractAnnulable

V. CONCLUSION

Ce projet a été formateur, aussi bien concernant java que la librairie SWING. C'était mon premier projet de programmation de cette envergure. Il m'a permis d'approfondir de nombreux sujets, du simple affichage via une interface graphique à la réalisation de fonctionnalités spécifiques telles que *undo/redo*.

Je tiens à remercier les professeurs du module d'interfaces graphiques, MM. Carton et Padovani, pour leur aide tout au long du semestre, ainsi que mes camarades de classe sans lesquel.le.s il est probable que je n'aurais pu mener à bien ce projet.

De nombreuses fonctionnalités pourraient encore être implémentées : un éditeur de niveau, une aide à l'utilisateurice, un générateur de niveaux aléatoires, un solveur permettant de trouver automatiquement une solution à un niveau (ce qui comprendrait une composante algorithmique), etc.