Projet Java

Plomberie

Par TekneX

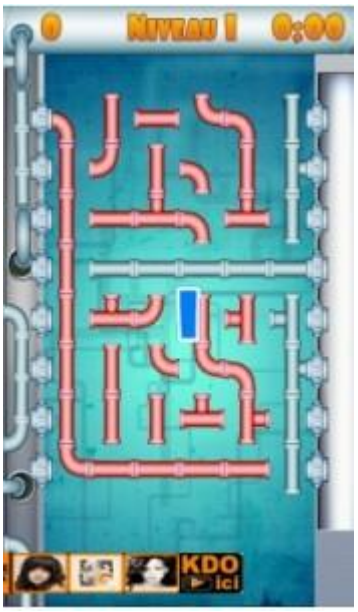
Sommaire :

Introduction

Le projet de Java de l'année 2021 était centré sur le langage Java, un langage orienté objet. Un des avantages de Java est sa portabilité. En effet, contrairement au C par exemple, il est très facile d'exporter un programme en Java sur différentes machines, et même différents systèmes d'exploitation. Un deuxième grand avantage est sa nature même, puisqu'il est orienté objet, ce qui permet la création d' « objets » permettant de créer des types de variables (tel que les int ou les char), mais également de pouvoir intégrer dans ces objets des fonctions (ou méthodes), ou encore des attributs. Ce système de programmation permet des programmes travaillant sur différents espaces en même temps, de pouvoir créer une infinité d'objet, ou encore permet la modification de façon rapide est aisé grâce au système de parents (extends ou implements pour les interfaces)

Le projet proposé par le professeur est en apparence très simple : créer une grille, où il est possible de placer une source et différents tuyaux. L'objectif ? Créer un chemin de tuyaux passant par le plus de cases possibles. Ce projet étant inspiré des jeux de « Plombier » disponibles partout sur la toile, l'interface allait fortement y ressembler.

Certains autres objectifs optionnels ont été l'ajout du temps de calcul, l'ajout d'un interface affordante, ou encore l'ajout de boutons reset ou stop pour arrêter le programme.



Applicaton :
Jeu du Plombier

Objectifs

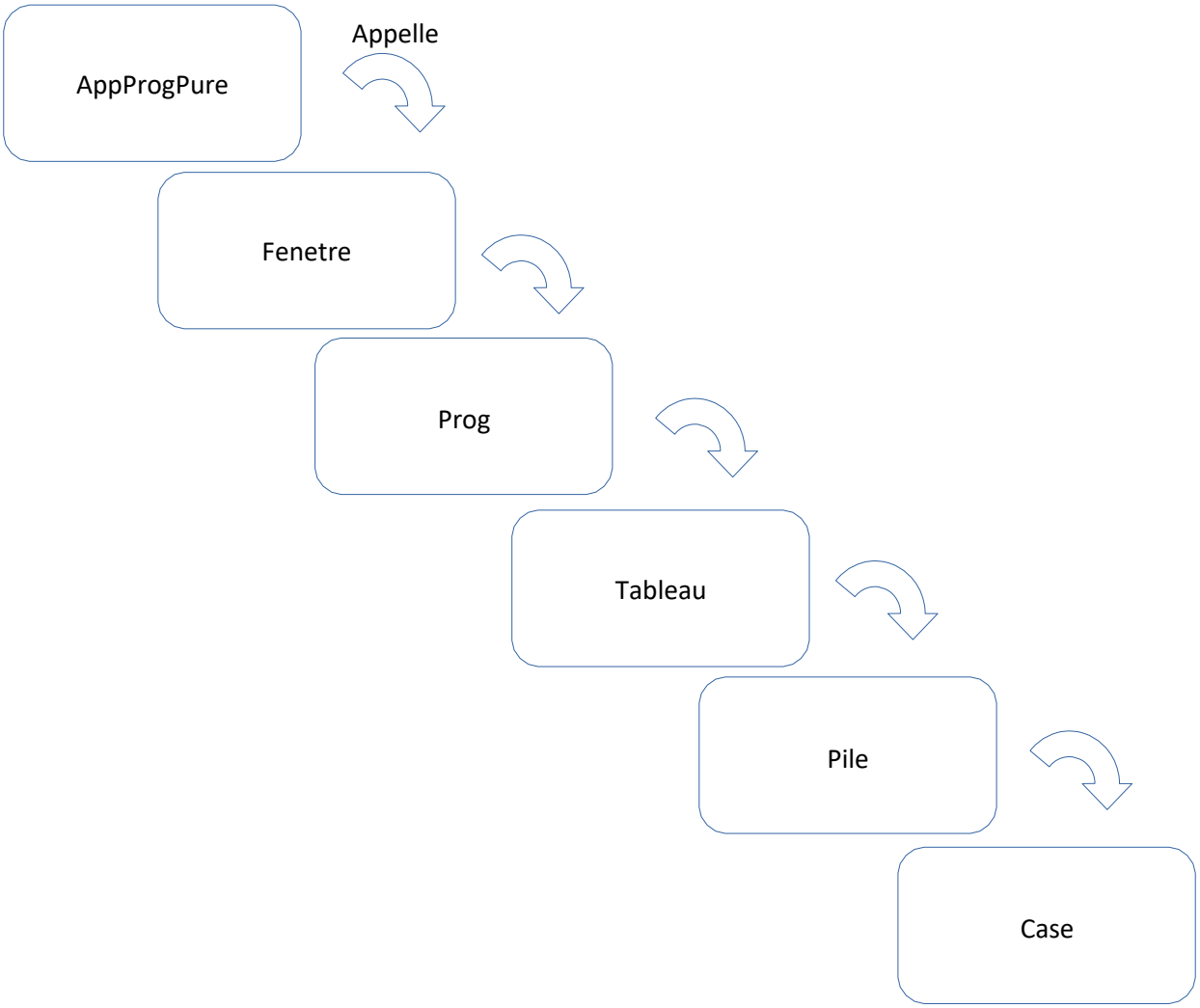
L'objectif du projet étant de donner à l'utilisateur le meilleur chemin possible, il fallait donc réaliser la totalité des chemins possibles. Je me suis donc posé des objectifs à respecter, puis les ai réalisé pas à pas pour obtenir un programme construit efficacement. Voici mes objectifs à atteindre :

- Réaliser un algorithme capable de remplir les cases d'un tableau de dimension 2
- Réaliser les fonctions pour tester les cases suivantes
- Réaliser un retour, ceci permettant de revenir en arrière pour tester tout les chemins possibles
- Mettre en places des tuyaux fixes, et modifier l'algorithme en conséquence
- Créer une interface visuellement correct et affordante, c'est-à-dire facile d'utilisation et intuitive pour l'utilisateur

Mes objectifs et mon plan de travail établis, je réalisais mon programme. Je vais dans ce dossier expliquer le fonctionnement de celui-ci.

Organisation des Class

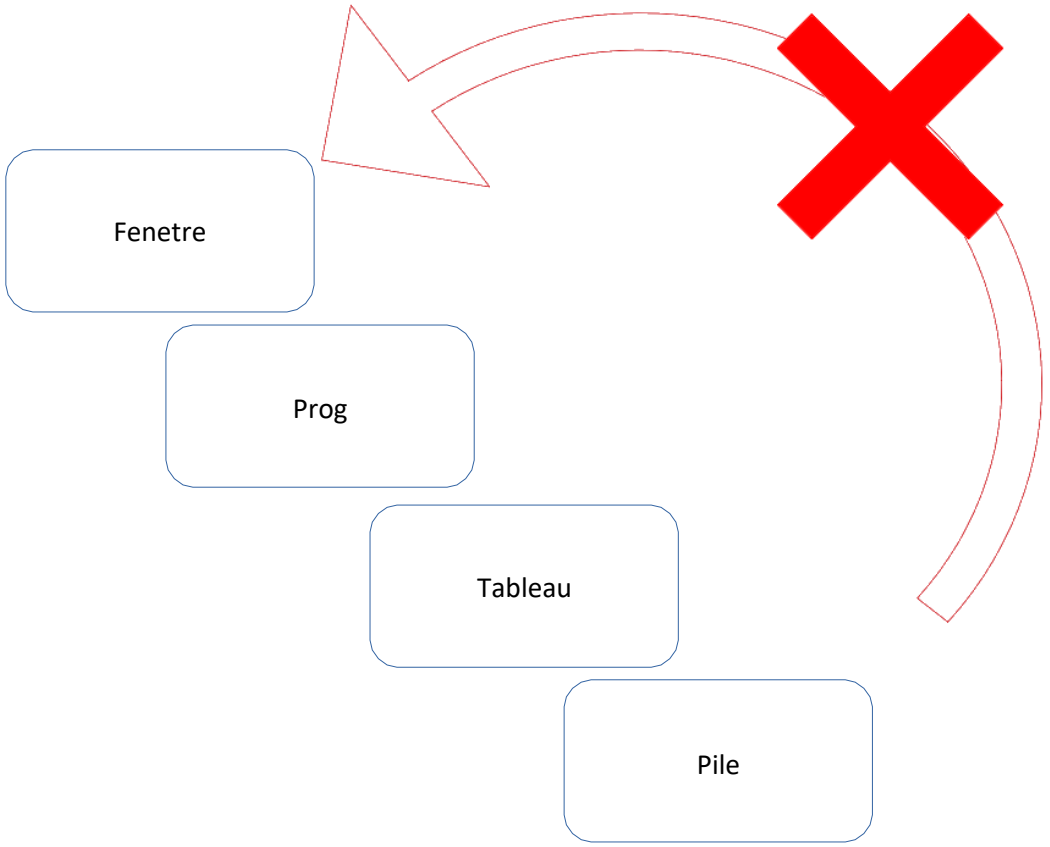
Les class de mon programme fonctionnent en une sorte de cascade, chaque fonction appelle d'autres fonctions d'une certaines class, mais uniquement de celle-ci. Voici un schéma de ces class :



Explications des class :

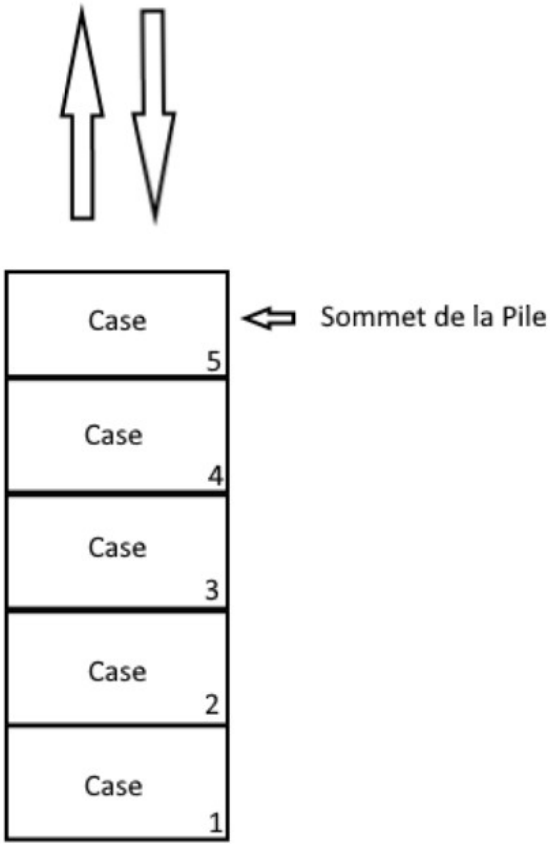
AppProgPure : le programme main, il créé un nouvel objet Fenetre
Fenetre : l'interface du programme, l'utilisateur choisis sa configuration
Prog : l'algorithme du programme, le constructeur appelle un objet Tableau
Tableau : l'endroit où se trouve le tableau de tuyaux et où se trouvent les fonctions de tests de cases suivantes
Pile : la pile de Cases
Case : l'objet Case contient des arguments utiles pour l'algorithme

Pour revenir au fonctionnement en cascade, la class Case ne communique pas avec Pile mais Pile communique avec Case, Pile ne peut pas communiquer avec Fenetre, mais Fenetre ne peut lui non plus pas communiquer avec Pile.



Fonctionnement des Piles

La Pile est un nouvel élément découvert cette année en Java. Son fonctionnement est assez simple : il utilise un tableau de vecteur, dont la taille peut varier, mais aussi le nombre d'attributs.



Ici, on empile des Cases, comprenant les coordonnées actuelles de la case, son orientation et l'orientation de la case précédente. Seule la dernière Case empilée est disponible. Ici c'est la Case n°5. La Case n°4 n'est pas disponible, et le seul moyen de la voir est de dépiler la Case n°5. De cette manière le Sommet de la Pile sera la Case n°4 et on y aura accès.

J'ai utilisé cette méthode d'empilement pour effectuer le retour de mon chemin et pouvoir effectuer tout les chemins possibles.

Conception de algorithme

Pour différencier la mobilité de tuyaux (fixes ou mobiles), leur type (horizontal, verticale…) leur orientation (vers la gauche, vers le coin en bas à droite…) j'ai utilisé des int. Mon quadrillage est donc un tableau de int, et les orientations des Cases seront également des int.

0 : vide	+10 : fixe	1 : →
1 : ‖	+20 = mobile	2 : ↓
2 : ═		3 : ←
3 : ╗	-1 : limites	4 : ↑
4 : ╝		
5 : ╔		
6 : ╚		
7 : source		
8 : bouche d'égout		

Cette façon d'écrire permet, avec uniquement 2 int, d'avoir un type de tuyau et son orientation. Par exemple, si la Case vaut 22 et que son orientation vaut 3, cela veut dire que c'est un tuyau horizontale, fixe et allant vers la gauche.

Remarque :

La bouche d'égout (la sortie), la source et la case vide n'ont pas de dizaine : 18, 27 ou 20 n'existent pas, car ces Cases ne sont pas des tuyaux.

L’algorithme quant à lui fonctionne de cette façon :

- L’algorithme se situant dans la class Prog (ici nommé programme) teste la case actuelle. S’il se positionne sur une source, un tuyau fixe ou une case vide, on appelle la fonction se trouvant dans la class Tableau correspondante.

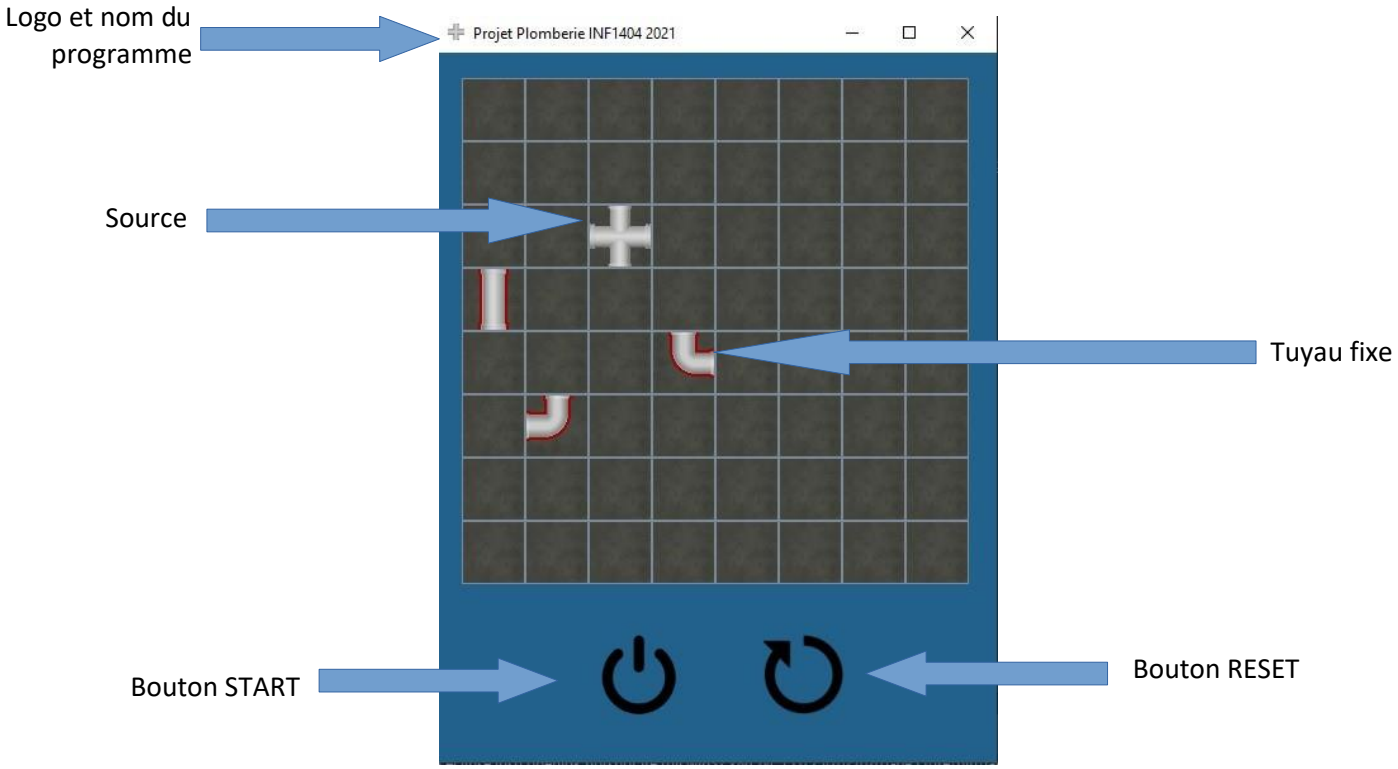
- L’algorithme renvoie un int, et en fonction de ce int, le programme va soit ne pas bouger mais augmenter l’orientation de 1, ce qui permet de tester une case différente car celle qui vient d’être testé n’est pas disponible, ajouter une case en l’empilant ou effectuer un retour en arrière et dépiler la case précédente.

- Le programme va ensuite réitérer ces étapes jusqu’à ce que toutes les cases soit remplies, ou lorsque que tout les chemins ont été effectués. Dans ce dernier cas, une sauvegarde du meilleur chemin aura été effectué et ce dernier apparaîtra a l’utilisateur.

- Si l’avant dernière case est rempli, le programme place une bouche d’égout sur la dernière case et le programme se termine. L’interface est alors actualisé.

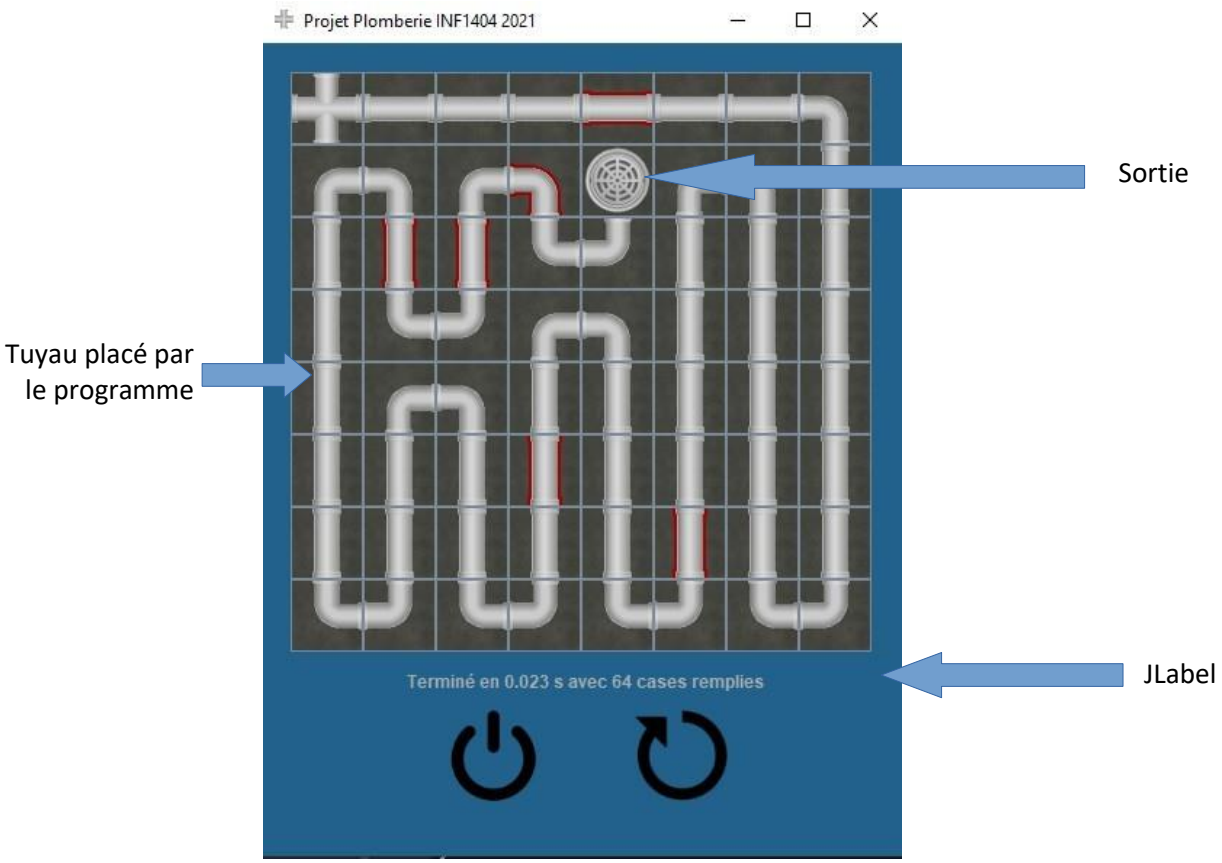
Interface

S’en suit la partie graphique de ce programme. L’affichage se constitue de cette manière :



Lorsqu’il lance le programme, l’utilisateur appuie sur des cases d’un tableau de boutons, ce qui place en premier lieu la source. Sur les cases suivantes se placeront les tuyaux fixes, entourés de rouge pour les distinguer. Un roulement de tuyaux permettra à l’utilisateur de choisir le tuyau qu’il souhaite. Les boutons Start et Reset servant respectivement à lancer le programme et vider le quadrillage.

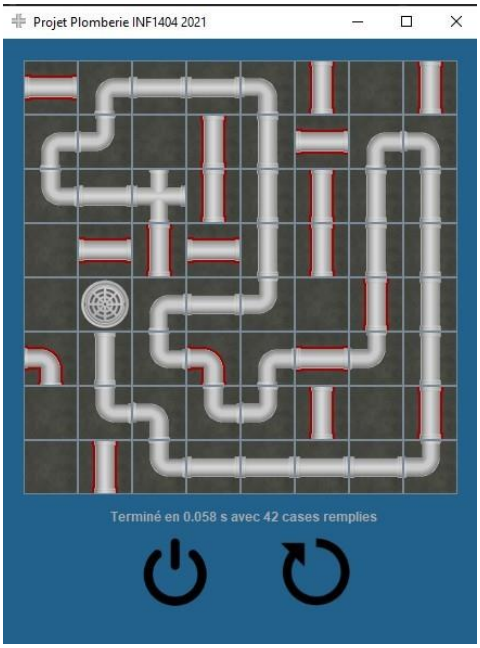
Une fois le calcul effectué, ce type d'interface apparaît :



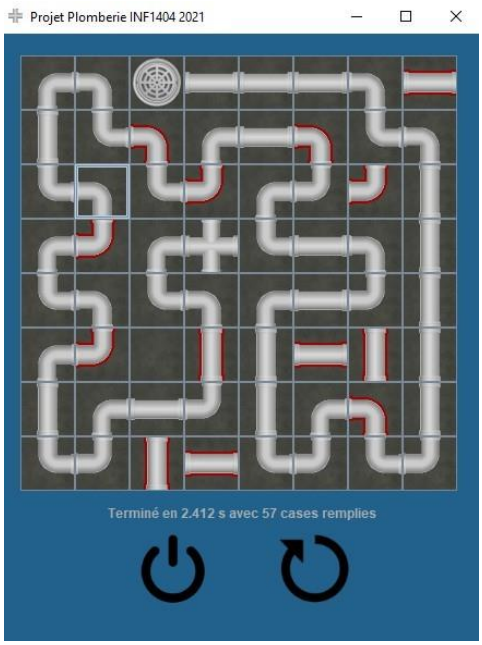
On y retrouve les même éléments, avec en plus un Jlabel pour afficher le temps de calcul et le nombre de cases remplies, les tuyaux mobile et la bouche d'égout placée.

Résultats

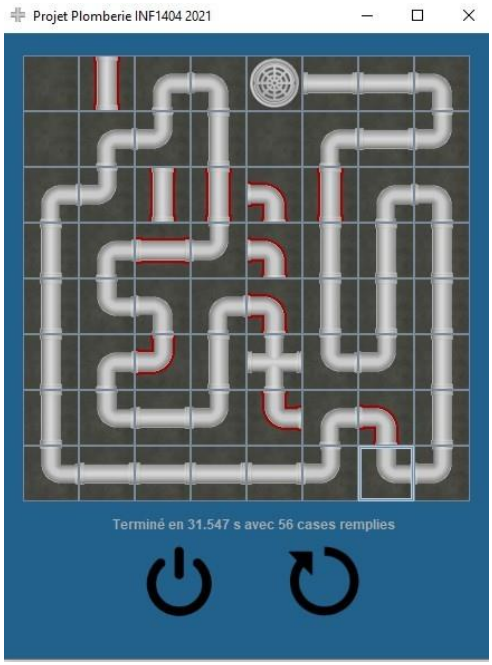
Le programme terminé, on peut maintenant tester quelques grilles. Cependant, du fait de la façon de programmation, les tuyaux qui seront testés en premiers (si vers le haut), ou le nombre de tuyaux fixes (le programme n'ayant pas à chercher les 3 orientations possibles pour ces tuyaux), le temps de calcul peut énormément varier. Sur un tableau de 8 cases sur 8, le temps de calcul peut s'avérer extrêmement long. Voici cependant quelques résultats que j'ai obtenu. On peut voir la différence flagrante de temps de calcul :



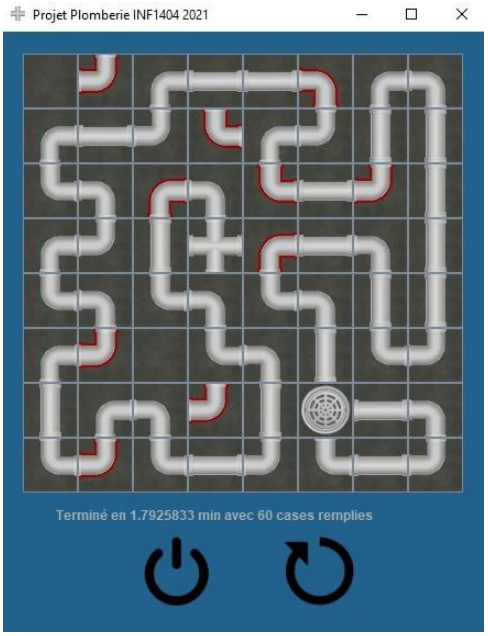
Ici le calcul est presque instantané, cela est dût au types de fixes, au fait qu'ils sont dirigés vers l'extérieur, au grand nombres de fixes...



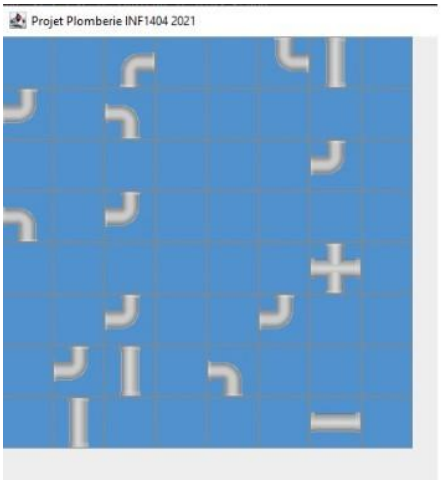
Ici un temps de calcul un tout petit peu plus long, dût au types de fixes : des courbés

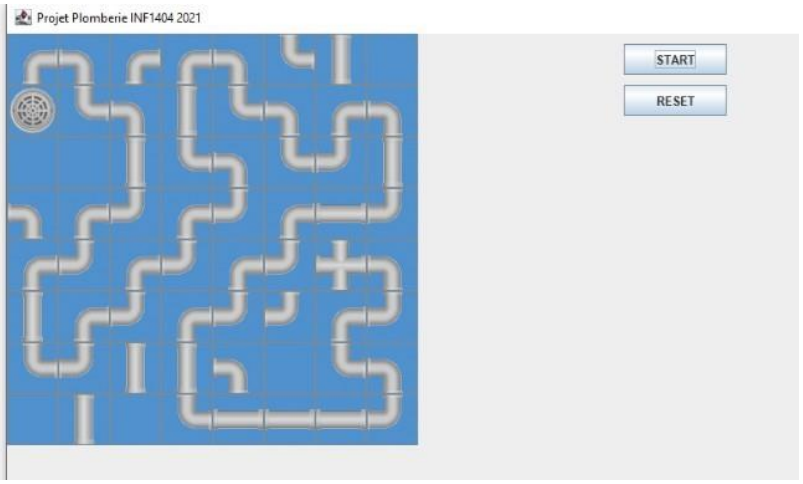


Ici les fixes sont moins présents, le temps de calcul est plus long

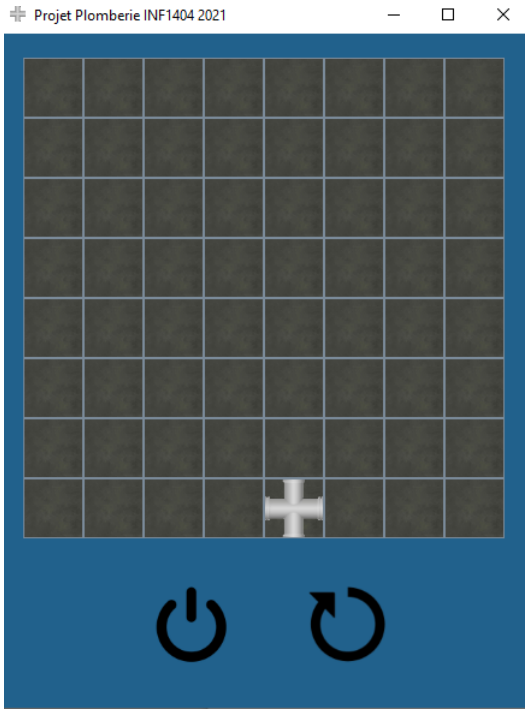


Même remarque

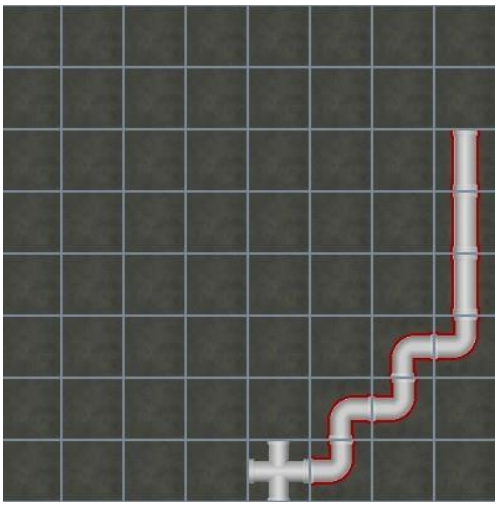




Ici le plus long calcul effectué. Il a duré 49 minutes. L'interface n'en était qu'a sa première version, le temps de calcul et le nombre de cases n'étaient pas encore affichés.
Ce long temps de calcul s'explique par la position de la source et du fait des tuyaux fixes : ils sont éparpillés. Le programme à dût tester l'ensemble des possibilités, et cela lui à prit un temps considérable.



Ici une grille non résolue, car le temps de calcul était bien trop long. J'ai stoppé le programme au bout de 5 heures de calculs. Cela s'explique par le fait que les fixes sont inexistants, et que la solution finale est une des dernières à être testés. En effet, le programme va d'abord tester toutes les possibilités du tableau alors que les cases du bas ne sont pas remplies (photo ci dessous). Cela rallonge de manière considérable le temps de calcul. Voici une des limites du programme : moins il y a de fixes, plus le calcul est long, car il ne passe pas de cases.



Tests

Durant la réalisation de ce programme, j'ai dût réaliser de nombreux tests pour corriger les erreurs de mon programme. Voici les techniques que j'ai utilisé :

- Le classique System.out.println(variable) ; pour tester une variable (aussi possible en System.out.println(''Boucle passée'') ;)

- L'affichage du tableau, avec des nombres correspondant aux types (-1,0,7,21,16…)

-L'affichage du tableau amélioré, avec des caractères spéciaux pour visualiser plus facilement le chemin en cours.

- Ce même affichage mais avec un Wait pour visualiser plus lentement le cheminement du programme, lorsqu'il se bloque…

Remarque :

La fonction Afficher() dans la class Tableau permet de visualiser avec des caractères UTF-8. Il est donc essentiel de passer l'encodage de Eclypse en UTF-8 pour profiter de cet affichage.

Ces tests m'ont permis de comprendre lorsque le programme ne rentre pas dans une boucle, de visualiser les coordonnées, de visualiser le chemin en temps réel, pour comprendre pourquoi il ne revient pas en arrière ou pourquoi il bloque à certains endroits…

Ils m'ont permis de me sortir de nombreuses erreurs incompréhensibles, et ces tests (allant de paire avec les erreurs) sont le meilleur debugueur possible, et un outil indispensable pour tout programmeur.

Limites et améliorations:

Mon programme marche. C'est un fait. Mais il existe de nombreuses limites et des nombreuses améliorations possibles. Comme limite principale nous retrouvons le temps de calcul qui peut parfois être long en fonction des paramètres choisis (comme vu dans la section Résultats). Une amélioration possible allant dans ce sens serait une meilleur optimisation de mon programme. Il est encore assez long est peut être amélioré. Malheureusement je n'ai pas encore les compétences nécessaires pour cette amélioration. J'aurai également pût créer une meilleur interface, plus belle, et plus intuitive (bien que celle actuelle soit déjà intuitive, elle l'aurait pût être encore plus). Une autre amélioration possible aurait été de rajouter un bouton stop permettant d'arrêter le programme s'il devient trop long et de donner à l'utilisateur le meilleur chemin trouvé avant qu'il stoppe le programme, ceci à l'aide d'un buffer. Malheureusement, par manque de connaissances techniques et par manque de temps, je n'ai pas pût réaliser ce bouton.

Il existent de nombreuses améliorations à mon programme, mais au-delà des limites du programme, il existe aussi des limites au programmeur. Je n'ai pas eu les connaissances nécessaires et surtout le temps disponible pour réaliser ces améliorations.

Conclusion :

Ce projet à été très intéressant. J'ai appris énormément de choses sur le maniement des langages de programmations orientés objets. Je ne connaissais pas du tout ce système et j'ai tout appris sur le tas. Ce type de programmation est très intéressant, j'ai pût y déceler certaines des utilisations possibles de Java, et des objets en général. Ce fût un projet très instructif, en terme de programmation, de gestion d'un lourd projet, et de gestion du temps.

Cependant il a également, à mon humble avis, de nombreux défauts. Tout d'abord le temps disponible pour sa réalisation : quelques semaines seulement n'étaient pas assez pour « redécouvrir » une façon de coder. Le Java est très différent du C et on ne code pas de la même manière. Je trouve aussi que le projet était assez complexe. J'ai heureusement réussit à le réaliser, mais j'y ai passé énormément de temps. Il n'y a également eu à mon avis pas assez de TD sur l'utilisation des Event, ou de l'affichage, ou encore des Piles (un mécanisme que j'ai eu du mal à comprendre au début). Je pense que des TD en parallèles et en lien avec le projet auraient été bénéfiques.

Pour conclure, malgré ces défauts j'ai aimé passé du temps dessus. Je suis très fier de ce que j'ai réalisé, et très heureux d'avoir découvert le Java, même si mon cœur reste du côté du C, malgré ses défauts de portage ou de débuguage.S