Rapport de projet Simulation de remplissage de baignoire

Encadré par : Monsieur Azim ROUSSANALY

Fait par : Ikram BADIR

Introduction

Objectif du Projet

Ce projet vise à simuler le remplissage d'une baignoire en prenant en compte divers paramètres tels que le débit d'eau entrant et les fuites éventuelles. L'application fournit une interface utilisateur graphique (IHM) intuitive pour permettre aux utilisateurs de configurer les paramètres de la simulation et de visualiser les résultats en temps réel.

Technologies Utilisées

- **Langage de programmation** : Java
- **Interface graphique** : JavaFX
- **Outils de build** : Maven

Interface Homme-Machine (IHM)

Présentation Générale

L'IHM de l'application est conçue pour être intuitive et facile à utiliser. Elle permet aux utilisateurs de configurer les paramètres de la simulation, de lancer, mettre en pause, arrêter et réinitialiser la simulation, et de visualiser les résultats sous forme de graphique.

Explications

Écran Principal

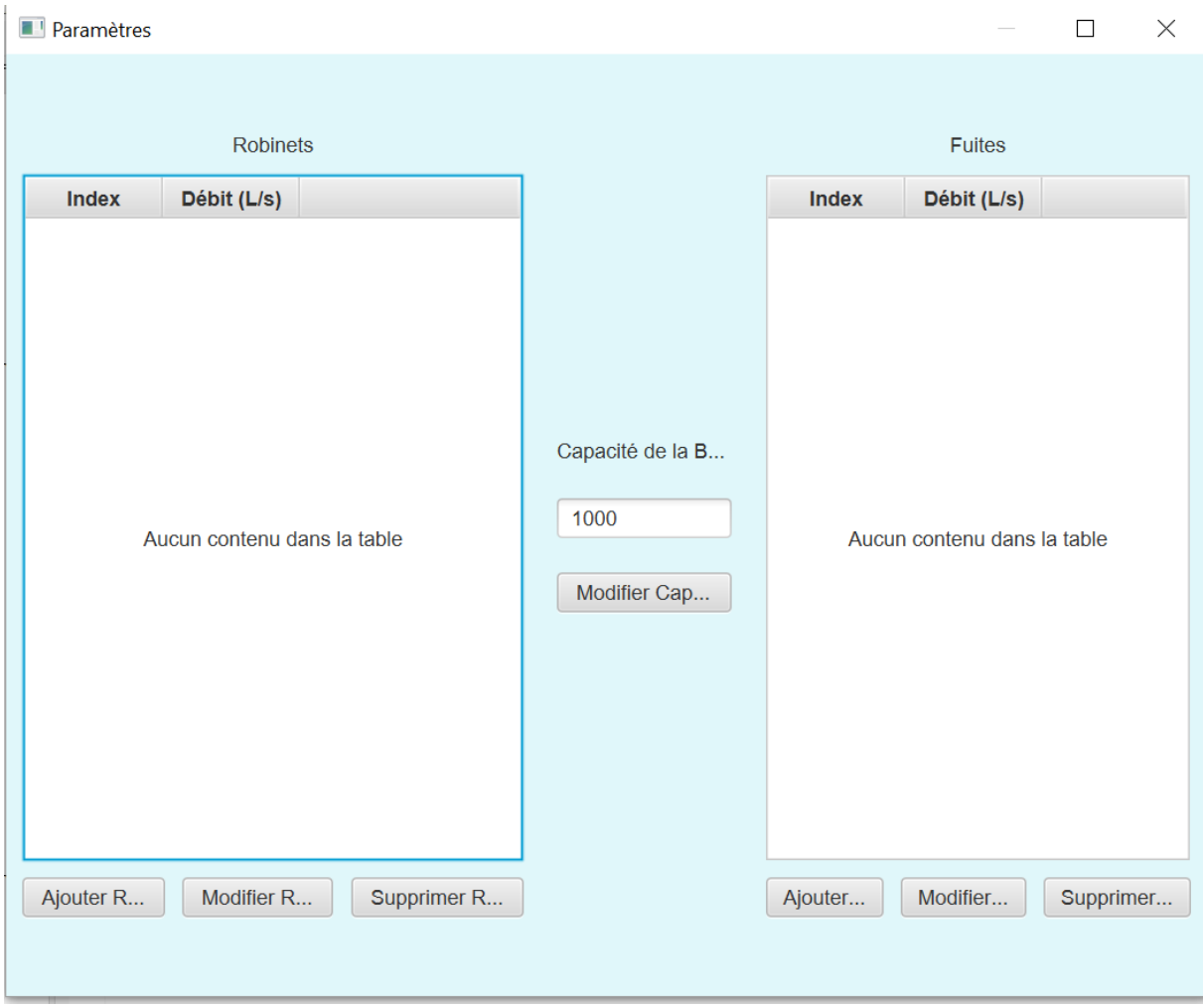
- Description : Cet écran permet de lancer et contrôler la simulation.
- Capture d'écran :

- Explications : Boutons pour démarrer, mettre en pause, arrêter et réinitialiser la simulation, Accéder aux paramètres ou on peut gérer les robinets, fuites et capacité de la baignoire, exporter le fichier csv, et le bouton pour visualiser la courbe d'avancement.

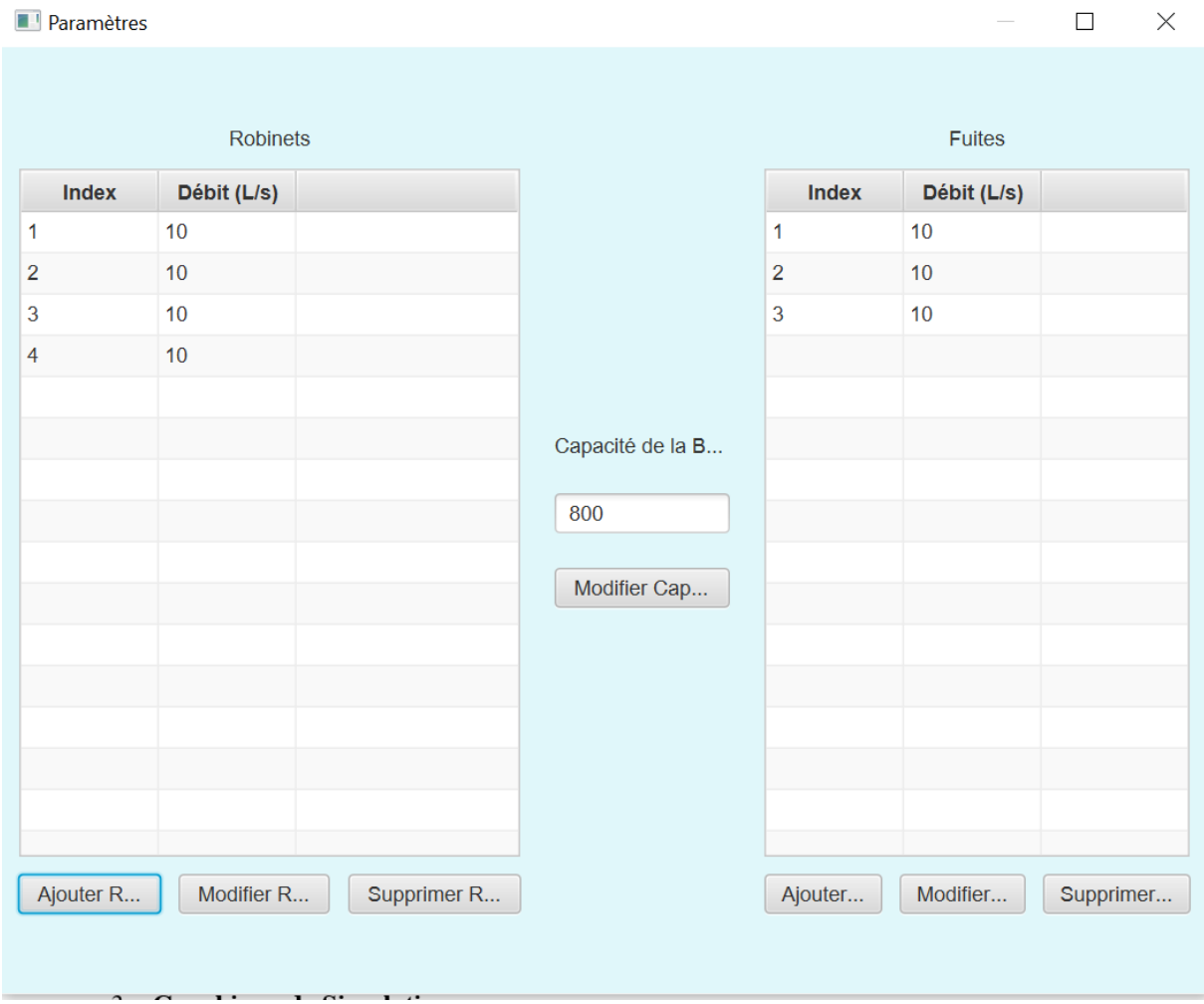
  On a aussi les informations sur la somme des débits des robinets, la somme des débits des fuites, la quantité actuelle de la baignoire et l'heure actuelle.

Configuration des Paramètres

- Description : Quand on appuie sur le bouton paramètres dans la fenêtre principale, une nouvelle fenêtre s'ouvre. Cet écran permet de configurer les paramètres tels que le débit du robinet et les fuites, ainsi que la capacité de la baignoire.
- Capture d'écran :



- Explications : Champs pour entrer les valeurs de débit et boutons pour ajouter ou supprimer des fuites.
- Exemple :



Ici en appuyant sur Ajouter Robinet on ajoute un robinet avec un débit par défaut 10, la même chose pour les fuites en appuyant sur Ajouter Fuite, et ça se sauvegarde directement, contrairement à la capacité de la baignoire qui était par défaut 1000 et si on veut modifier on entre la nouvelle valeur (ici 800) et on appuie sur modifier Capacité pour la sauvegardé.

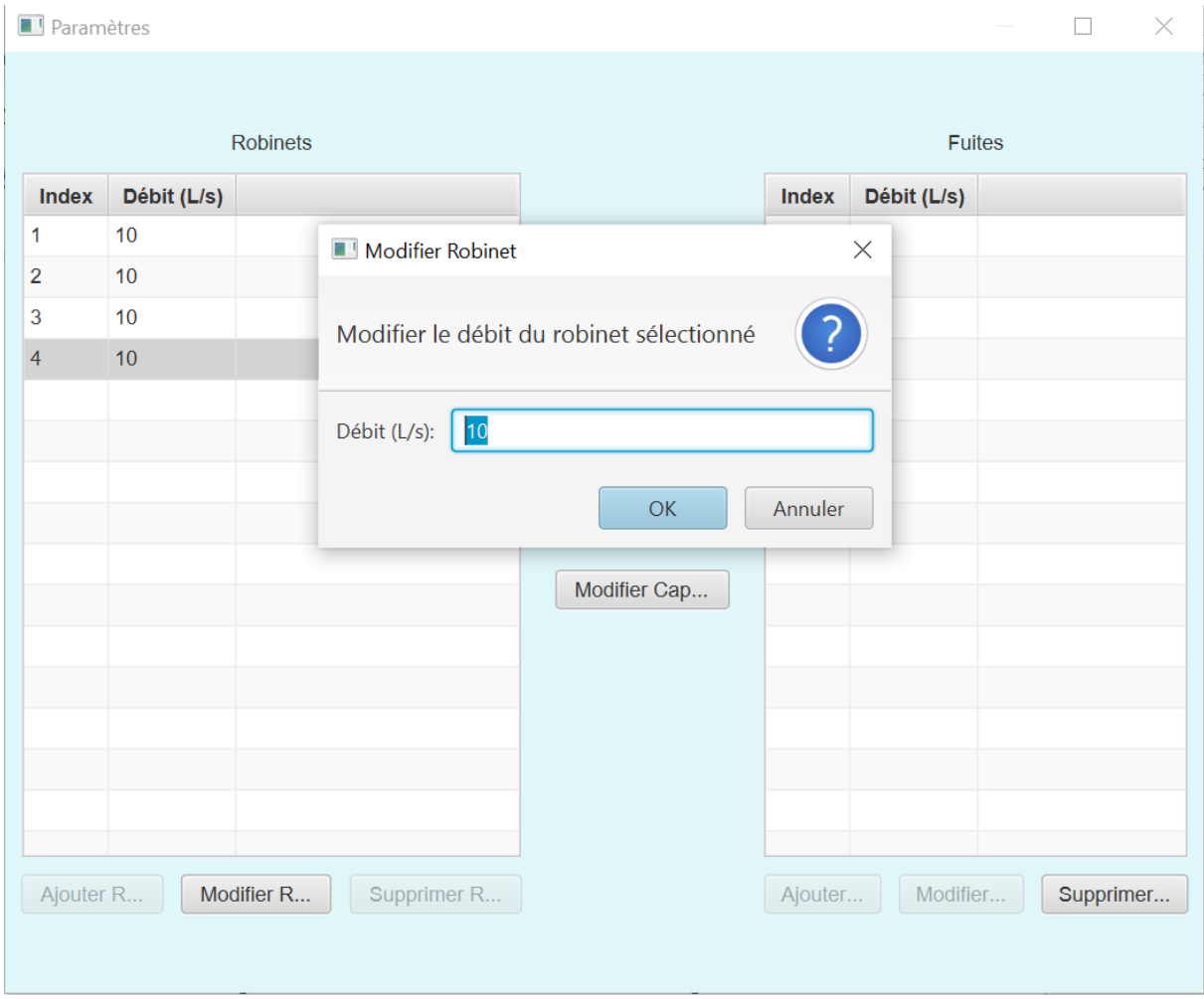
Ensuite on ferme la fenêtre de paramètres et on visualise la modification des informations sur la fenêtre principale.

Pour commencer on appuie sur le bouton commencer.

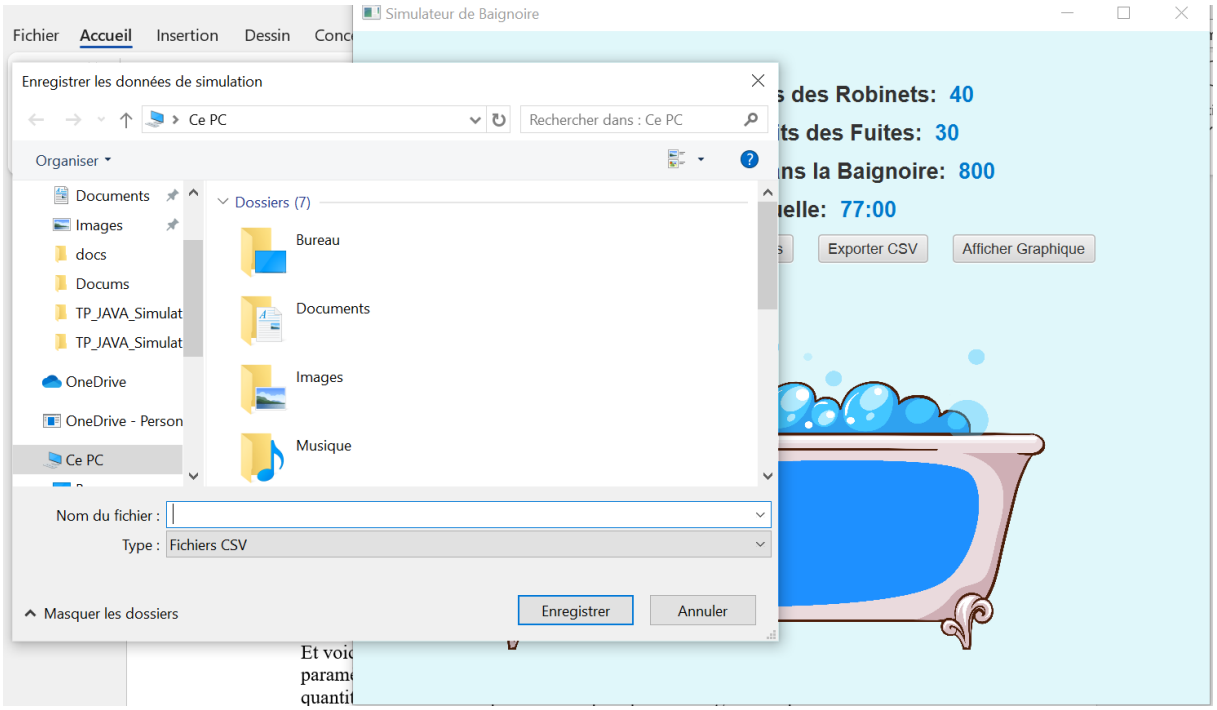
Et on commence à visualiser la simulation et modification de l'heure actuelle et quantité actuelle dans la baignoire à fur et à mesure.

Au cours de la simulation on peut modifier le débit d'un robinet, on le sélectionne et on clique sur modifier débit, une alerte s'ouvre pour entrer la nouvelle valeur souhaitée.

Et on peut aussi sélectionner une fuite et la supprimer.


Paramètres
Robinets
Fuites
Index
Débit (L/s)
1
10
2
10
3
10
4
10
Modifier Robinet
Modifier le débit du robinet sélectionné
Débit (L/s):
10
OK
Annuler
Modifier Cap...
Ajouter R...
Modifier R...
Supprimer R...
Ajouter...
Modifier...
Supprimer...

Et voici la vue finale, si on veut recommencer la simulation en gardant les mêmes paramètres on fais renitialiser, on peut télécharger le fichier .csv qui contient la quantité d'eau dans chaque instant séparé par une virgule.



On peut aussi voir la courbe d'évolution d'eau dans la baignoire en appuyant sur Afficher Graphique.

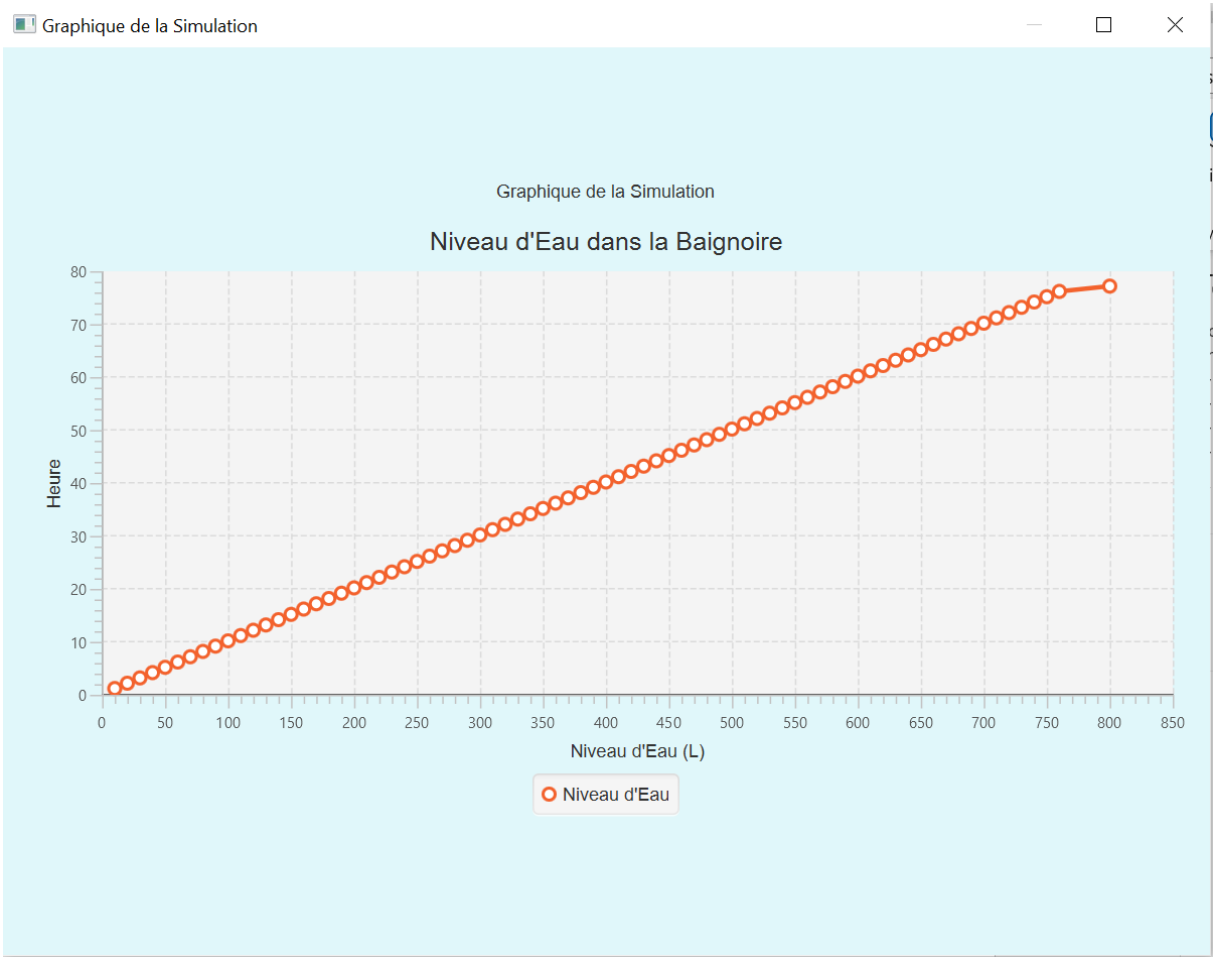
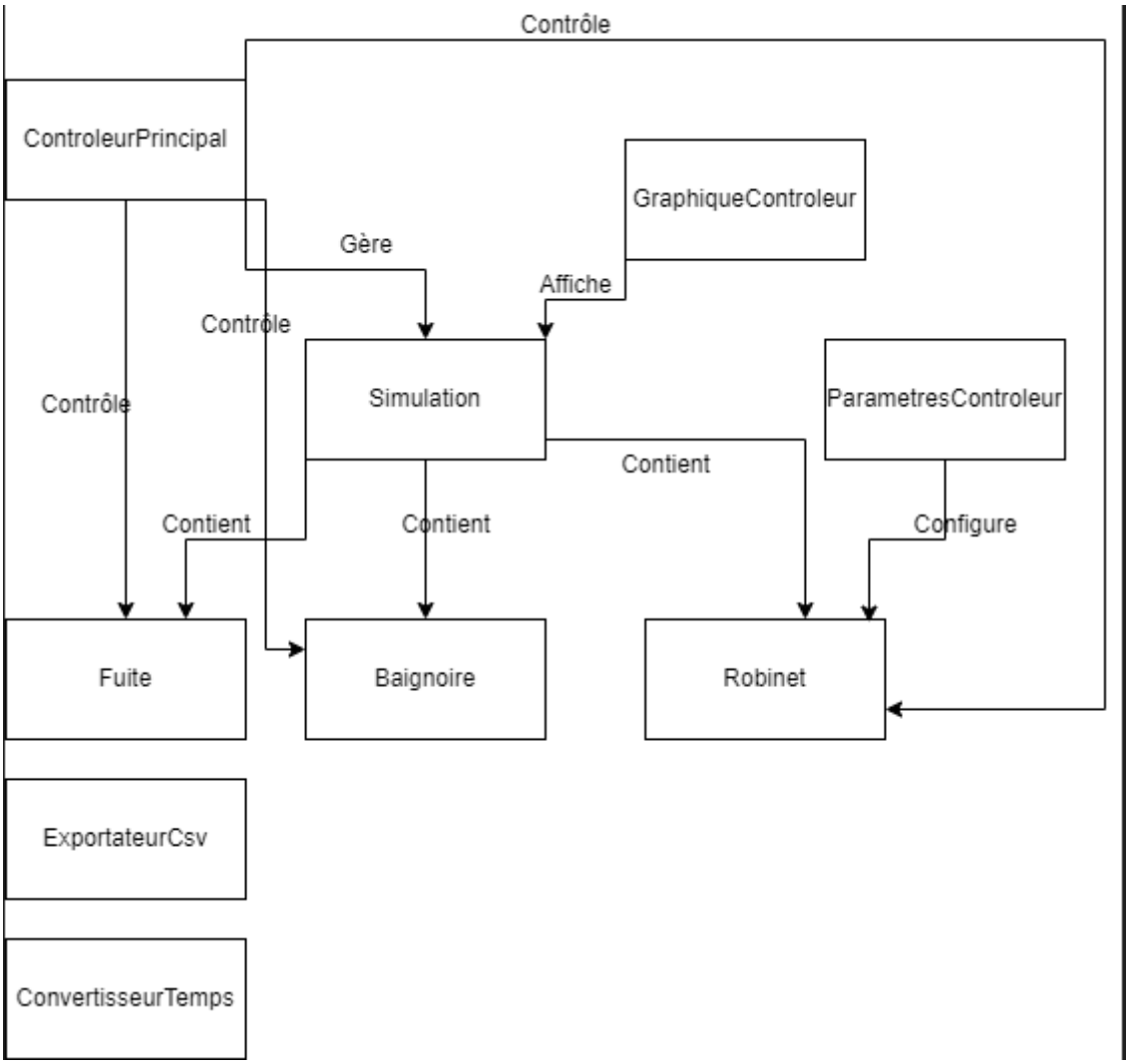


Diagramme de Classe

Présentation et Explication

Voici le diagramme de classe du projet :

Le diagramme est sans inclusion d'attributs et méthodes à cause du problème de lisibilité. Les attributs et méthodes de quelques classes sont définis en dessous du diagramme.



Explications

Classe Simulation

- **Attributs** :
  - baignoire : l'objet Baignoire représentant la baignoire.
  - robinet : l'objet Robinet représentant le robinet.
  - fuites : une liste d'objets Fuite représentant les fuites dans la baignoire.
  - niveauEau : le niveau d'eau actuel dans la baignoire.
  - volumeInitial : le volume initial d'eau dans la baignoire.

        - running : un booléen indiquant si la simulation est en cours d'exécution.
        - paused : un booléen indiquant si la simulation est en pause.
    - **Méthodes** :
        - lancerSimulation()
        - mettreEnPause()
        - arreterSimulation()
        - reinitialiserSimulation()

Classe Baignoire

- **Attributs** :
    - volume : le volume de la baignoire.
    - niveauEau : le niveau d'eau actuel dans la baignoire.
- **Méthodes** :
    - ajouterEau(volume)
    - retirerEau(volume)
    - getNiveauEau()

Classe Robinet

- **Attributs** :
    - debit : le débit du robinet.
- **Méthodes** :
    - ouvrir()
    - fermer()
    - getDebit()

Classe Fuite

- **Attributs** :
    - debit : le débit de la fuite.
- **Méthodes** :
    - getDebit()

Classe ControleurPrincipal

- **Attributs** :
    - startButton
    - pauseButton
    - stopButton
    - resetButton
    - niveauEauLabel
    - debitRobinetLabel
    - debitFuiteLabel
- **Méthodes** :
    - initialize()
    - startSimulation()

- pauseSimulation()
- stopSimulation()
- resetSimulation()
- setNiveauEau(double niveauEau)
- setDebitRobinet(double debitRobinet)
- setDebitFuite(double debitFuite)

Classe ParametresControleur

- **Attributs** :
  - debitRobinetField
  - ajouterFuiteButton
  - supprimerFuiteButton
  - fuitesList
  - appliquerButton
- **Méthodes** :
  - initialize()
  - ajouterFuite()
  - supprimerFuite()
  - appliquerParametres()
  - setDebitRobinet(double debitRobinet)

Section : Difficultés Rencontrées et Solutions Apportées

1. Difficulté : Gestion de la Concurrence

Problème : Lors de la simulation, plusieurs robinets doivent ajouter de l'eau à la baignoire simultanément. Cela crée un problème de concurrence où plusieurs threads peuvent essayer de modifier le niveau d'eau en même temps, ce qui peut conduire à des résultats incorrects ou à des conditions de course.

Solution : Pour résoudre ce problème, nous avons utilisé des sémaphores pour synchroniser les threads. Chaque thread représentant un robinet doit acquérir un sémaphore avant d'ajouter de l'eau à la baignoire et le relâcher une fois l'opération terminée. Cela garantit qu'à tout moment, un seul thread peut modifier le niveau d'eau.

```
semaphores.get(index).acquire();
try {
    baignoire.ajouterEau(debit);
} finally {
    semaphores.get((index + 1) % semaphores.size()).release();
}
```

2. Difficulté : Synchronisation des Threads

Problème : La synchronisation des threads pour que les opérations se déroulent dans un ordre spécifique et éviter les conflits était un défi. Chaque thread devait attendre son tour pour ajouter de l'eau, et la simulation devait se dérouler en temps réel.

Solution : L'utilisation de sémaphores avec un mécanisme de pause a permis de gérer efficacement la synchronisation. Chaque thread vérifie si la simulation est en pause avant de continuer, et les sémaphores garantissent que les threads ajoutent de l'eau dans l'ordre correct.

```
while (isPaused) {
    Thread.sleep(100); // Délai pour éviter une attente active trop agressive
}
```

3. Difficulté : Gestion de l'État de la Baignoire

Problème : Assurer que l'état de la baignoire (niveau d'eau et si elle est pleine) est correctement mis à jour et ne tombe pas dans un état incohérent était une autre difficulté. Par exemple, ajouter trop d'eau ou retirer plus d'eau que disponible pouvait conduire à des incohérences.

Solution : Des méthodes synchronisées ont été utilisées pour ajouter et retirer de l'eau, garantissant que les modifications au niveau d'eau sont thread-safe et que les conditions sont vérifiées après chaque opération.

```java
public synchronized void ajouterEau(int quantite) {
    if (!pleine) {
        this.niveauEau += quantite;
        if (this.niveauEau >= this.capacite) {
            this.niveauEau = this.capacite;
            this.pleine = true;
        }
    }
}

public synchronized void retirerEau(int quantite) {
    if (!pleine) {
        this.niveauEau -= quantite;
        if (this.niveauEau < 0) {
            this.niveauEau = 0;
        }
    }
}
```

4. Difficulté : Mise en Pause et Reprise de la Simulation

Problème : Permettre la mise en pause et la reprise de la simulation sans perdre l'état courant était essentiel pour la flexibilité et le contrôle de la simulation.

Solution : Un drapeau isPaused a été introduit, et les threads vérifient ce drapeau avant d'effectuer des opérations. Lors de la mise en pause, les threads entrent dans une boucle d'attente jusqu'à ce que le drapeau soit réinitialisé.

```java
while (isPaused) {
    Thread.sleep(100); // Délai pour éviter une attente active trop agressive
}
```

Conclusion

Ce projet de simulation de remplissage de baignoire a permis de développer une application interactive avec une interface utilisateur graphique intuitive. L'application permet de configurer les paramètres de la simulation, de visualiser les résultats en temps réel et de gérer les différentes conditions de la simulation comme le débit d'entrée et les fuites.