Auteurs : Hugo WARIN, Baptiste CAMPRUBI,

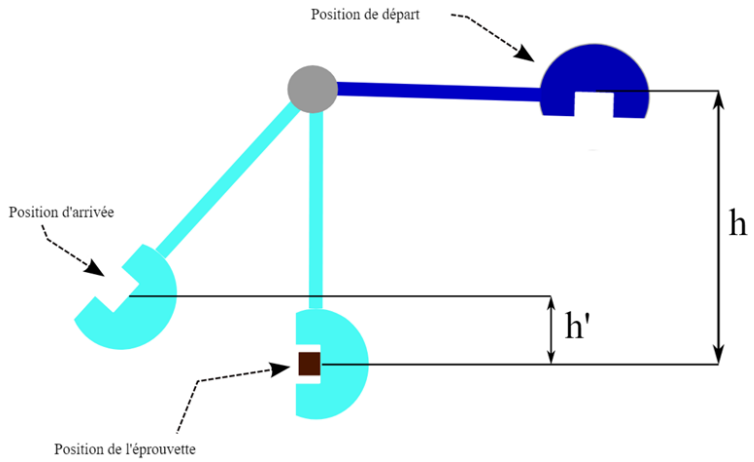
Théo BRUSQUE, Nathan CHAMPOUILLON

Simulateur de mouton de Charpy

I. Présentation générale

Durant ce projet d'informatique, nous souhaitons réaliser un algorithme simulant le fonctionnement d'un mouton de Charpy. Un mouton de Charpy, au fonctionnement analogue à celui d'un pendule amorti étudié en physique, est un dispositif permettant notamment de mesurer la résilience (la résistance à l'impact) d'un matériau en mesurant l'énergie nécessaire pour briser une éprouvette de dimensions connues. Tout **comme un véritable mouton de Charpy, vous pourrez simuler la réaction du pendule lorsqu'il casse une éprouvette.**

Voici un schéma du dispositif :



II. Cahier des charges

Le but de ce programme est de modéliser un mouton de Charpy afin de savoir si une éprouvette d'un matériau déterminé peut être cassée par le marteau. Le programme doit permettre de prendre en compte de nombreux paramètres comme la masse du marteau, la longueur de la tige ou la surface de l'éprouvette à casser. Le programme devra afficher une animation du pendule en mouvement, incluant la casse de l'éprouvette et prenant en compte la valeur du coefficient de frottement choisi. L'utilisateur doit également pouvoir choisir de rajouter à la base de données le matériau de son choix. A la fin de la simulation, un tableur doit être créé contenant les résultats de la simulation.

III. Description du problème

L'objectif de l'algorithme est donc de résoudre l'équation d'un pendule simple non linéaire. Notre pendule doit être le plus réaliste possible, c'est pourquoi les frottements ont été pris en compte. Nous souhaitons que tous les paramètres du système soient paramétrables par l'utilisateur. On modélise un pendule de masse m, de longueur l , d'angle initial θ_0 avec une vitesse angulaire initiale v avec un coefficient de frottement f. On note θ l'angle de notre pendule. En utilisant le principe fondamental de la dynamique on obtient l'équation différentielle suivante :

$$\ddot{\theta} - f\dot{\theta} + \frac{g}{l}\sin\theta = 0$$

A partir de l'angle θ, on peut ainsi déterminer les coordonnées du pendule à chaque itération du timer, que nous utiliserons pour l'affichage du déplacement.

IV. Principe de l'algorithme

IV.1. Fonctionnement général

Le lancement du programme crée une nouvelle fenêtre d'affichage avec la classe Fenetre. Les différentes caractéristiques du pendule que l'utilisateur choisit en écrivant dans les différents JTextField et en utilisant le JSlider sont collectées et sont paramètre dans le constructeur du Pendule. Lorsque l'utilisateur appuie sur Lancer, la simulation se lance. Au moment de l'impact entre le marteau et l'éprouvette, pour déterminer si l'éprouvette est brisée, l'énergie cinétique du pendule est comparée à la résilience du matériau.

IV.2. Résolution de l'équation différentielle et calcul des coordonnées

La fonctionnalité principale du programme est donc la résolution d'une équation différentielle non linéaire de second ordre.

Pour cela, nous avons utilisé un schéma Euler explicite. Ce schéma nécessite un pas de temps faible pour fonctionner, nous avons choisi ici un temps de 15 ms, soit une fréquence de 60 Hz. Le code est le suivant :

```
// Résolution d'équation différentielle avec ici la méthode Euler explicite
theta.add(theta.getLast() + 0.015 * omega.getLast());
omega.add(omega.getLast() + 0.015 * (-longueurReelle*Math.sin(theta.get(theta.size() - 2) / 9.81 + frottements * omega.getLast())));
```

Théta correspond à l'angle du pendule et oméga à la vitesse angulaire. La méthode d'Euler est une méthode par incrémentation qui fonctionne sur des approximations, c'est pour cela que le temps entre deux calculs doit être faible.

Théta et oméga sont des LinkedList, nous avons fait ce choix car il y a beaucoup d'ajout en fin de liste.

Les coordonnées des extrémités du marteau sont déterminées par le calcul suivant :

```
finX = (int) (centreX + longueur * Math.sin(theta.getLast()));
finY = (int) (centreY + longueur * Math.cos(theta.getLast()));
```

Le Marteau est ensuite dessinée de la façon suivante :

```
g.drawLine(centreX, centreY, finX, finY);
g.fillOval(finX - largeur, finY - largeur, 2 * largeur, 2 * largeur);
```

IV.3. Lecture du fichier de la base de données de matériaux

Les matériaux créés sont présents dans un fichier texte. Cependant, il faut que le programme puisse lire ces données, pour cela, nous avons utilisé un BufferedReader. Le code est le suivant :

```
// Permet de lire le fichier texte
BufferedReader input = new BufferedReader(new InputStreamReader(new FileInputStream( name: "BDMat.txt")));

// Ici on va lire ligne par ligne le fichier texte en séparant les données grâce aux virgules
String ligne;
final String SEPARATEUR = ",";

while((ligne = input.readLine()) != null) {

    String attributs[] = ligne.split(SEPARATEUR);
```

On crée un tunnel de connexion jusqu'au fichier. Le fichier est formaté de sorte que toutes les données soient séparées par des virgules. On récupère donc ligne par ligne le fichier et on utilise la méthode split() de la classe String pour séparer les données

IV.4. Ecriture des résultats de la simulation

```
// Initialisation du nom du fichier
SimpleDateFormat formatDatePourNomFichier = new SimpleDateFormat( pattern: "yyyyMMdd-HHmmss");
String datePourNomFichier = formatDatePourNomFichier.format(new Date());
String nomFichier = "resultat_" + datePourNomFichier + ".csv";

// L'association try/catch permet en cas d'erreur d'afficher un message universel qui sera inscrit dans le catch
// Il est obligatoire ici pour que le writer s'initialise
try {

    // Création du writer
    BufferedWriter writer = new BufferedWriter(new FileWriter("./output/" + nomFichier, append: false));

    // On écrit le résumé de la simulation
    writer.append(resumeSimulation);      You, il y a 5 jours • commentaires …
    if(ep.estVivant) {

        writer.append( csq: "L'eprouvette n'a pas ete detruite ! \n");
```

Pour écrire dans un fichier ou pour créer un fichier, il faut utiliser un BufferedWriter qui possède pour argument un FileWriter. Cela permet d'optimiser le programme et éviter certains bugs d'optimisation du FileWriter. Cependant, une des problématiques était de ne pas écraser les anciens fichiers. Le nom du fichier est donc généré automatiquement en fonction de la date et l'heure de la simulation. On définit donc un format de date pour pouvoir formater celle-ci de façon correcte.

Pour écrire une ligne, il suffit d'utiliser la méthode append() de BufferedWriter.

IV.5. Utilisation de try/catch

Dans beaucoup de méthodes, nous avons utilisé l'association try/catch qui permet de réagir si des erreurs se produisent. Le fonctionnement est le suivant : on ouvre le try, si une erreur est détectée à l'intérieur de celui-ci, il va jeter l'erreur et lancer le code contenu dans le catch. Cela était obligatoire pour le BufferedReader et le BufferedWriter dans les cas où les fichiers ne pouvaient pas être crées ou être lus.

```
// Permet de détecter si les données saisies possède le bon type ou non
try {

    if(Integer.parseInt(TxtTailleTige.getText()) > 6) {
```

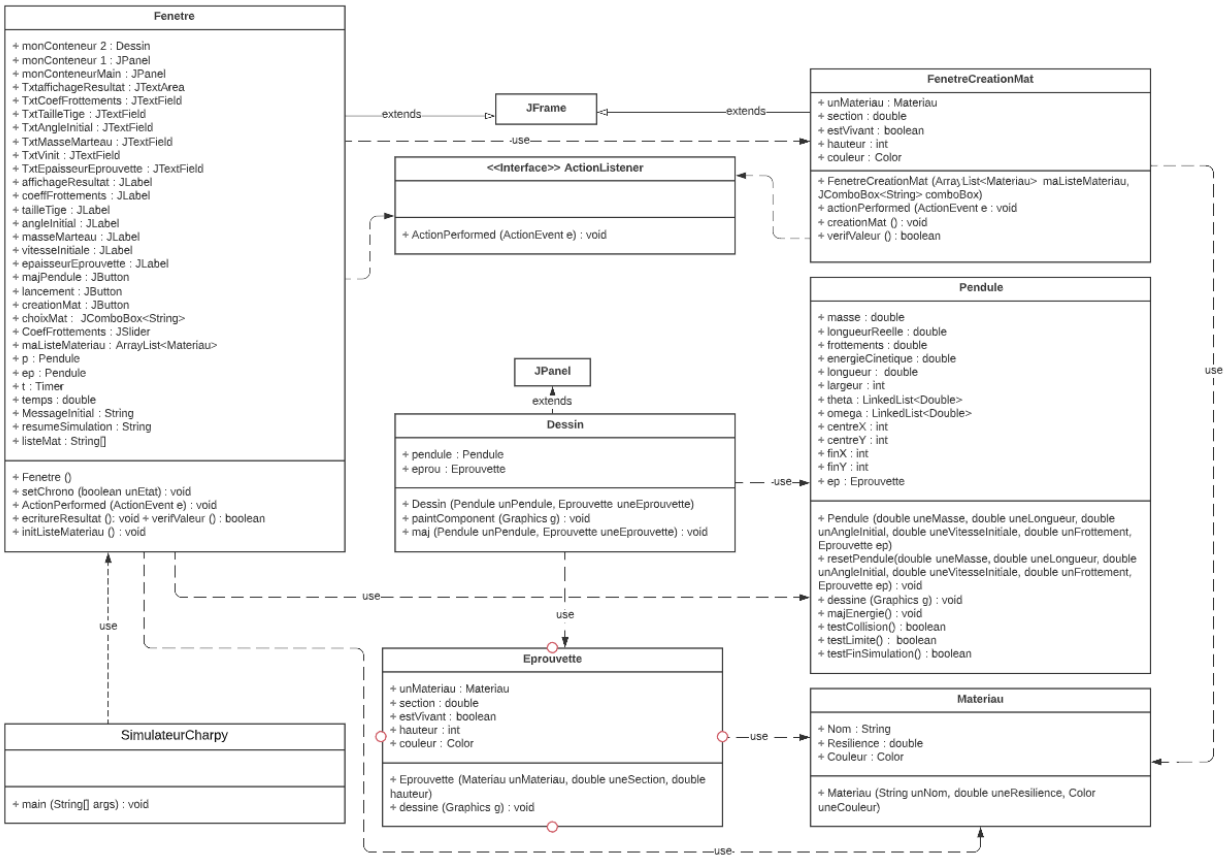
```
// Si le code entre dans cette méthode, c'est qu'une des données ne possède pas le bon type
} catch(Exception ex) {

    JOptionPane.showMessageDialog(this, message: "Valeurs saisies incorrectes", title: "Erreur saisie", messageType: 0);
    return(false);

}
```

Nous l'avons utilisé à d'autres endroits, comme par exemple lorsque l'on souhaite vérifier que les données saisies sont correctes. Dans le cas où l'on souhaite récupérer un double, on doit utiliser Double.parse-Double(« text ») mais si le texte n'est pas un double, le code dans la boucle catch va être exécuté.

V. Structuration des données



Le code est composé de 7 classes : Fenetre, FenetreCreationMat, SimulationCharpy, Dessin, Eprouvette, Pendule et Materiau.
Voici leur rôle respectif :

- SimulationCharpy : contenir la méthode main, qui créée la fenêtre principale lorsque l'utilisateur lance le code
- Fenetre : définir la fenêtre principale avec laquelle l'utilisateur interagit. Cette fenêtre implémente 3 conteneurs : monConteneurMain, monConteneur1 et monConteneur2. monConteneur1 possède plusieurs JButtons, JTextFields, JLabels, une JComboBox et un JTextField.
  monConteneur2 est la zone d'affichage de la simulation.
- FenetreCreationMat : définir la fenêtre que l'utilisateur utilise s'il souhaite créer un nouveau matériau pour la simulation ;
- Dessin : définir la zone dans laquelle l'éprouvette et le pendule sont dessinés ;
- Eprouvette : définir le type « Eprouvette » avec son constructeur et ses attributs (unMateriau, section, estVivant, hauteur, couleur);
- Pendule : définir le type « Pendule » avec son constructeur et ses attributs (masse, longueurReelle, frottements, energieCinétique, longueur, largeur, theta, omega, centreX, centrY, finX, finY, ep)
- Materiau : définir le type « Materiau » avec son constructeur et ses attributs (Nom, Resilience, Couleur);

SimulationCharpy utilise Fenetre pour lancer l'affichage.
Les classes Fenetre et FenetreCreationMat héritent de JFrame et implémentent la méthode ActionPerformed (ActionEvent e) de l'interface ActionListener tandis que Dessin hérite de JPanel.
Fenetre utilise FenetreCreationMat lorsque l'utilisateur souhaite créer un nouveau matériau.
FenetreCreationMat utilise le constructeur de la classe Materiau
Dessin utilise les méthodes dessine (Graphics g) de Eprouvette et de Pendule.
Fenetre utilise les méthodes testLimite(), testCollision() et testFinSimulation() de la classe Pendule et le constructeur Materiau() de la classe Materiau.

VI. Suggestions d'améliorations et bugs connus

On utilise la méthode Euler explicite pour résoudre l'équation différentielle. Cependant, cette méthode possède quelques limites. La première est que cette méthode est une approximation, elle dépend donc du pas de temps. Or si nous choisissons un coefficient de frottement nul, le pendule a tendance à gagner de l'énergie mécanique ce qui n'est pas très réaliste.

De plus, le simulateur commence à faire des erreurs de calculs lorsque le pendule passe à la verticale.

Une suggestion d'amélioration serait donc d'utiliser un schéma de résolution plus complexe permettant de faire cette simulation. Nous avons décidé de ne pas mettre un autre schéma car cela n'était pas pertinent dans l'utilisation première de notre programme.

Une suggestion d'amélioration serait également de pouvoir revenir en arrière dans la simulation. Utiliser la simulation comme un lecteur vidéo, ce qui peut être très pratique dans certains cas.

Une autre suggestion serait que le programme est créant le compte-rendu de la simulation dessine directement la courbe de l'angle du pendule en fonction du temps.

Un autre bug connu est que l'on peut créer un matériau ayant pour nom uniquement des espaces.

VII. Carnet de route

-1ère séance : Nous avons structuré nos idées pour le projet et défini les classes, méthodes et attributs qui nous seront nécessaires.

-2ème séance : Nous avons codé la fenêtre d'affichage, en particulier les JLabel, JPannel… permettant la sélection des attributs du pendule. Nous avons également codé la classe Materiau et la classe Eprouvette permettant de créer une éprouvette du matériau et de l'épaisseur souhaités. De plus, nous avons commencé la création du timer et de l'affichage graphique du pendule

-3ème séance : Nous avons amélioré l'affichage graphique et nous avons terminé de représenter le pendule graphique. A l'issue de cette séance, le pendule oscillait. Cependant, nous ne tenons pas encore compte de la collision avec l'éprouvette.

-4ème séance : Nous avons résolu l'équation différentielle du mouvement du pendule, complété la base de données des matériaux et créé un affichage pour indiquer que l'éprouvette est cassée.

-5ème séance : Durant cette dernière séance, nous avons corrigé les erreurs de fonctionnement du code. En particulier pour l'oscillation du pendule.

-Hors séance : Nous avons ajouté la gestion des erreurs, notamment par le biais de JOptionPane qui s'affichent lorsque les valeurs rentrées par l'utilisateur dépassent les limites de la simulation. Nous avons également créé une fenêtre permettant à l'utilisateur de créer un nouveau matériau qui s'ajoute directement dans la base de données. Enfin, nous avons ajouté la fonctionnalité suivante : lorsque la simulation est terminée, un tableur se crée contenant les résultats de la simulation.

VIII. Bibliographie

- Schéma du Mouton de Charpy :

https://upload.wikimedia.org/wikipedia/commons/thumb/4/41/Mouton_charpy.svg/1280px-Mouton_charpy.svg.png

- Equation différentielle du pendule simple :

https://fr.wikipedia.org/wiki/Pendule_simple

- Pour utiliser un slider en java :

https://docs.oracle.com/javase/tutorial/uiswing/components/slider.html

- Résilience des bois :

https://www.lignum.ch/files/_migrated/content_uploads/Propri%C3%A9t%C3%A9s_m%C3%A9caniques_du_bois_01.pdf

- Polycopié RDM_P2I3_2022 : pour les résiliences des matériaux
- Cours M3.BD P2I2

IX. Répartition du travail dans le groupe

Théo : 23%
Baptiste : 23%
Nathan : 23%
Hugo : 31%