GRENOBLE INP UGA Ensimag

Cours

C++ pour les mathématiques appliquées

Simulation des systèmes particulaires complexes

- Auteurs -
Zineb et-tarraf et Safia Echarif

17 mai 2022

Table des matières

Table des figures

1 Introduction

L'objectif de ce projet est de mettre en place une simulation à différentes échelles de systèmes particulaires complexes. Il s'agit aussi de montrer une maîtrise des notions de classe, pointeur, référence, opérateurs... du C++.

Dans ce projet nous sommes également amenés à apprendre comment améliorer les performances de notre code en faisant des optimisations dans les choix algorithmiques ainsi que dans les choix des structures utilisées.

Pour pouvoir montrer notre compréhension du cahier des charges, il est nécessaire de montrer que notre code répond aux besoins fonctionnels via des diagrammes expressifs : **diagramme de cas d'utilisation, diagrammes de séquence, diagrammes d'état-transition.**

2 L'arborescence du projet :

2.1 TP 2 : Introduction

Dans cette étape de projet on implémenté une première version de la classe Particule. On a verifié l'efficacité des différentes structures de données utilisées dans le TP (Cf Analyse de Performance pour plus de détails) . On a ensuite implémenté l'algorithme de Störmer-Verlet.

Les tests consernants cette étape sont :

Affichage de l'avancement du système gravitationnel *Soleil, Terre, Jupiter,Haley* .
Test des performances des structures : *deque , list , vector* .

2.2 TP 3 : Utilisation des opérateurs

On a implémenté la classe Vecteur qui avait pour but de faciliter les calculs .C'est la classe où on a bien maniplulé la notion des opérateurs . On a ensuite modifié la classe particule pour utilsé la classe Vecteur .Puis on a implémenté la classe univers .
Les tests consernants cette étape sont :

Test de l'initialisation de l'univers .
Test de l'initialisation et de l'avancement de l'univers .
Test de la classe *Vecteur* .

2.3 TP final 4&5 : Découpage de l'éspace et Test et visualisation :

Dans cette étape de projet on vous fournit une version finale de la classe Univers après avoir implémenté la classe Grille qui contient un ensemble d'éléments de type cellule (une classe qu'on a implémenté aussi).C'est à l'aide de cette grille qu'on a fait la mise à jour des positions .

On fournit aussi deux sous classes de la classe particule qui sont :
Particulegravitationnelle ,ParticuleLennard qui representent deux type de particules .

3 Analyse de Performance

3.1 Evalution de la performance de différentes structures pour stocker les particules :

Pour évaluer la performance de différentes structures de stockage de particules. On génére aleatoirement des particules et on les stocke dans des structures différentes puis on calcule le temps d'insersion ainsi que le temps de parcours pour chacune de ces structures.

⚠ Pour bien factoriser le code on a pensé à utiliser la fonction template de C++ qui permet d'adapter l'entrée à tout type Cf. TP2/scr/Particule.cxx la fonction *display-generate-particles.* Cette fonction prend en entrée une structure quelconque et génère aléatoirement un nombre (donné en paramètre) de particules qui les stocke ensuite dans cette structure et parcourt cette dernière pour afficher les caractéristiques de chaque particule.

On peut remarquer dans les deux figures qui suivent la structure deque est plus rapide que list et vector au niveau d'insersion . Bien que la différence observée soit faible pour un petit nombre de particules ,la différence se voit plus nettement avec un grand nombre de particules.

```
position [ 0.465546  0.380884  ]  speed [ 0.155084  0.69644  ]
mass : 7.8655e-05  id :5  type: No type

position [ 0.616862  0.492371  ]  speed [ 0.437366  0.281907  ]
mass : 0.00138551  id :0  type: No type

Pour 50 particles :
list performance : 0.00448112s  vector performance : 0.00335352s  deque performance : 0.00357431s
```

FIGURE 1 – Performance des structures pour 50 particules générées aléatoirement

```
position [ 0.387991  0.700923  ]  speed [ 0.351464  0.758291  ]
mass : 0.00727087  id :6  type: No type

position [ 0.800571  0.2624  ]  speed [ 0.190925  0.568112  ]
mass : 0.0012311  id :0  type: No type

position [ 0.552798  0.562472  ]  speed [ 0.381176  0.360319  ]
mass : 0.000346384  id :7  type: No type

Pour 5268 particles :
list performance : 0.464185s  vector performance : 0.362255s  deque performance : 0.396921s
```

FIGURE 2 – Performance des structures pour 5268 particules générées aléatoirement

3.2 Amélioration du découpage de l'espace :

Afin d'éviter que à chaque itération la grille crée un nouveau tableau ,ce qui augmente la complexité de l'algorithme , On a pensé à fournir la classe *Cellule* par la liste des cellules voisines ainsi que la position de la cellule dans une grille.
Ceci va permettre aussi d'éviter de prendre en considération les cellules vides puisque on ne va ajouter dans les voisins que les cellules comportant des particules.

4 Satisfaction des besoins fonctionnels du cahier des charges

4.1 Diagramme de cas d'utilisation :

Afin de décrire les principales fonctionnalités de notre produit : la simulation des systèmes complexes . On propose le diagramme de cas d'utilisation suivant :

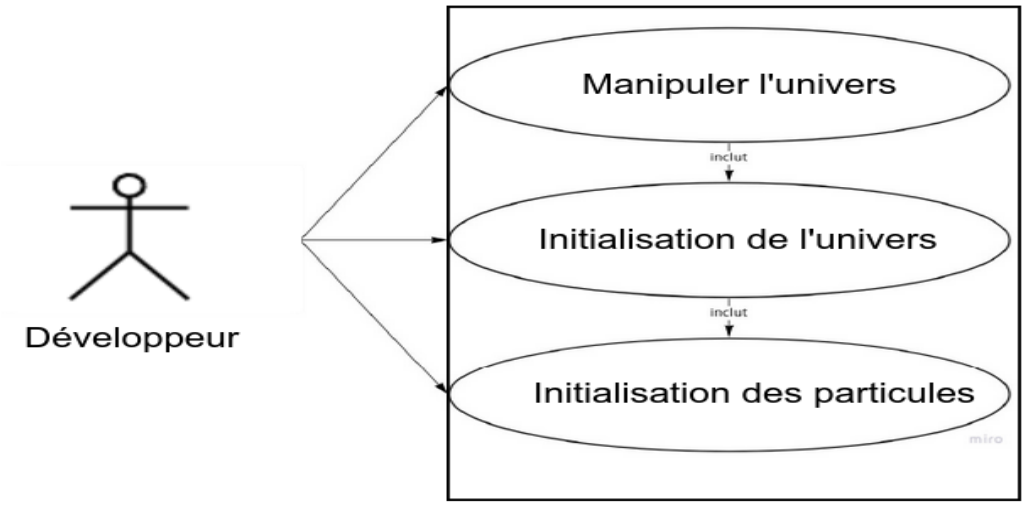

FIGURE 3 – Diagramme de cas d'utilisation du système développé

— **Identification des acteurs :** Dans notre cas, le seul acteur pouvant utiliser ou développer notre bibliothèque est un développeur.
— **Identification et structuration des cas d'utilisation :** Aprés avoir initialisé les particules ainsi que l'univers . Le developpeur ne peut manipuler l'univers qu'à travers la classe univers .

Finalement ce diagramme a pour but de montrer comment on compris et reformulé le cahier des charges ainsi comment on a structuré les besoins fonctionnels.

4.2 Diagrammes de séquence :

Le diagrammes de séquence système a pour but de mettre l'accent sur sur les interactions acteurs/système .Ceci en définissant les évènements auxquels le système peut réagir ainsi que les évènements que le système peut générer .

Dans notre cas pour que la simulation avance notre système à besoin qu'il soit initialisé ainsi que de lui fournir le pas de temps à utiliser .

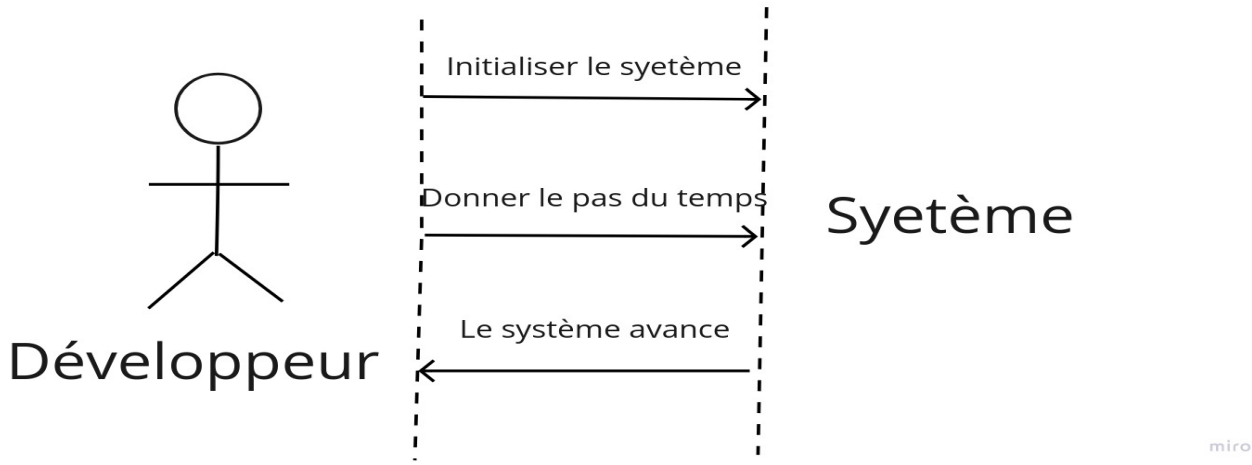

FIGURE 4 – Diagrammes de séquence du système développé

4.3 Diagrammes d'état-transition :

Voici l'algorithme de Störmer-Verlet et le le diagramme de transition du système correspondant :

Algorithme 1 : algorithme de Strömer-Verlet

Data : particles initiales
Data : vecteur de force $\mathbf{F}^{old}$
Initialisation : calcul des forces $\mathbf{F}$;
while $t < t_{end}$ **do**
  $t = t + \delta_t$;
  for *toutes les particles i* **do**
    $x_i = x_i + \delta_t * (v_i + 0.5/m_i * \mathbf{F}_i * \delta_t)$;
    $F_i^{old} = F_i$;
  end
  Calculer les forces $\mathbf{F}$;
  for *toutes les particles i* **do**
    $v_i = v_i + \delta_t * 0.5/m_i * (\mathbf{F}_i + \mathbf{F}_i^{old})$;
  end
  Calculer les quantités dérivées ;
  Afficher les quantités t, x, y ;
end

FIGURE 5 – L'algorithme de Störmer-Verlet

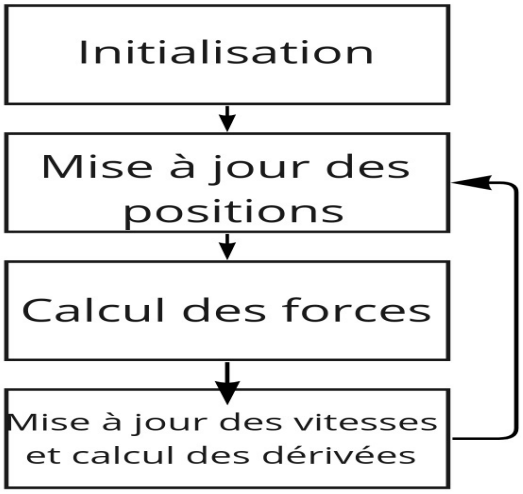

FIGURE 6 – Diagrammes d'état-transition du système développé

5 Conclusion :

Le projet en question nous a permis de nous familiariser avec les notions de base du C++ : classe, pointeur, référence, opérateurs...
En plus d'aller au-delà et de pouvoir discuter la performance et de pouvoir l'améliorer ainsi que de se mettre dans la peau d'un ingénieur et de pouvoir s'engager professionnellement sur les besoins du cahier des charges.Nous avons pu apprendre comment nous monter que notre produit répond aux spécifications requises.