Rapport TP 5

TAIBI Sofiane
FISSEL LARAKI Mohamed

22 octobre 2020

Table des matières

1 Remarques d'implémentaion **2**
1.1 En général 2
1.2 Classe Vector3D 2
1.3 Classe Earth 3

2 Réponses **3**
2.1 TP5 3
2.1.1 Exercice 2 3

3 Tests **4**
3.1 testSatellite1 4
3.2 testSimulation4 4
3.3 testSimulation3 4
3.4 testMobile2 4
3.5 testSimulation1 4
3.6 testMobile1 5
3.7 testVector3D 5
3.8 testEarth 5

Introduction

Le but de ce TP est de simuler des mobiles pesants ou pas dans un environnements à 3 dimensions. On sera amener à faire attention sur certains points dont :

- d'utiliser le passage des arguments par référence, selon ce qui a été indiqué en cours ;
- de faire attention aux fuites de mémoire (undeletepour unnew) ;
- utilsier le mot clefconstchaque fois que c'est utile, pour les arguments des méthodescomme pour les méthodes elles-mêmes ;
- de ne pas mettre d'attributs publics, à moins d'avoir une justification pour cela.

1 Remarques d'implémentaion

Dans cette section, on va aborder des points qui ne sont pas forcèment communs aux autres projets, et essayer de prouver pourquoi ce choix est judicieux à notre goût.

1.1 En général

On a essayé de faire en sorte, que chaque classe ait des attributs privés et aucun publique, ce qui, on pense a bien été réussi. Certaines méthodes, ont dûes être convertie en virtual afin d'être utilsées correctement. On pense surtout aux méthodes **mass** et **gravity** , qui sans le mot clé **virtual** poseraient problèmes vu qu'elles étaient principalement appelées depuis des pointeurs sur Mobile* et non pas MobileHeavy*.

1.2 Classe Vector3D

Pour la classe Vector3D, on a décidé d'implementer les coordonnées de notre vecteur sous la forme d'un tableau statique de 3 doubles au lieu d'utiliser 3 variales de type double séparemment. Ceci nous paraissait plus juste et simplifiait l'accès aux differentes composantes de notre vecteur grâce aux boucles **for**. De plus, cette implémentation facilite le passage de vecteurs de dimensions 3 à plus (ou moins). En ce qui concerne, les déclarations et définitions de méthodes, on a cherché à mettre les méthodes en **const** ou bien de mettre les paramètres constants dès que ceci était nécessaire. C'est à dire qu'au fur et à mesure de

l'avancement de notre travail, on modifiant le prototype des fonctions afin que ce dernier puisse répondre aux attentes du TP.

1.3 Classe Earth

```
class Earth : public MobileHeavy {

  private:
    Earth();
    static Earth* instance;

  public:
    Earth(Earth& other) = delete;
    ~Earth();
    void operator=(const Earth&) = delete;
    static Earth* GetInstance();
  };
```

On a rendu notre constructeur privé afin qu'il ne soit accessible depuis l'exterieur. Il fallait rendre l'operateur = inutile ainsi que constructeur par copie. Enfin, il a fallu créer une fonction *GetInstance* qui :

- Si, l'attribut *instance* est NULL, on créé alors une nouvelle instance de Earth, sinon on créé rien.
- On renvoie *instance* .

2 Réponses

2.1 TP5

2.1.1 Exercice 2

Question 1

> 'Attention,il est possible que cette liste contienne le mobile lui-même : que risque-t-il de se passer si on necontrôle pas ce cas dans cette méthode ?'

Si on essaye de calculer la force qu'exerce un objet sur lui même en utilisant la fonction gravity :

```
Vector3D MobileHeavy::gravity(const Vector3D& p) {
  double d = norm( p - this->position);
  return   (p - (this->position))*(-(this->mass()*6.6708e-11)/(d*d*d));
}
```

On remarque qu'on obtiendra une distance **d** nulle et par la suite dans le return, on divisera par 0.

3 Tests

3.1 testSatellite1

On fait avancer notre satellite jusqu'à faire le tour de la Terre. On sauvegarde sa position de départ et sa position d'arrivée afin de calculer l'écart entre les 2.On obtient une distance de 65km (en ayant pris les mêmes valurs de test que le professeur). Le test renvoie true si la distance obtenue est en dessous d'une certaine limite prédifinie manuellement.

3.2 testSimulation4

Pour testSimulation4, on créé une simulation, on remplit son **bodies** et ensuite on créé une deuxième simulation. Enfin, on lance la fonction **simulate** pour une même durée dans les deux simulations et on vérifie qu'à la fin, les 2 ont la même valeur de **time** .

3.3 testSimulation3

Dans cette section, On a deux objets et la Terre dans une simulation. On lache les 2 objets à une altitude de 100m de la surface de la terre. Le test réussit si on l'objet pesant atteint le sol au bout de 4,5 secondes et que la position de l'objet non pesant ne varie pas.

3.4 testMobile2

Ce test est exactement le même que le précedent sauf qu'on a qu'un seul objet pesant.

3.5 testSimulation1

On verifie les differentes méthodes de la classe Simulation telle que :
— getBodies()
— addBody()
— removeBody()
— etc
Afin de tester **addBody** , on parcourt la liste et on verifie si l'objet courant correspond à l'un des objets qu'on a créé auparavant. Pour **getBodies** , on utilise la taille de la liste, et pareil pour **removeBody()**.

3.6 testMobile1

On créé un objet Mobile avec une position P et une vitesse V. On verifie donc les fonction **getPosition et getVitesse** en comparant la valeur renvoyée à P ou V. On verifie enfin la fonction avance en verifiant si la nouvelle position est bien celle qu'on obtient en faisant le calcul à la main.

3.7 testVector3D

Dans ce premier test, on a verifié que tous les opérateurs qu'on a redéfini fonctionnent correctement. On a par ailleurs testé les fonctions **norme() et distance()**.

3.8 testEarth

Afin de tester la classe Earth, on a créé deux instance de Earth, et on a verifié que le fait de modifier la valeur d'une instance modifiait la valeur dans l'autre instance. Ensuite, on a testé notre fonction **gravity** en retrouvant la valeur de g = 9.81.

On a obtenu 9.82

.

Conclusion

On pense que l'objectif du TP est atteint, on a essayé de respecter autant que possibles les points cités dans l'introduction. Ce TP a été pour nous une occasion de concretiser les notions vues en cours et donc par la suite d'approndir nos connaissances en language C++ et plus généralement en orienté objet. On est satisfait de notre résultat malgré qu'il ne soit pas conforme au résultat de l'enseignant. On pense que la différence provient de petites erreurs de calculs qui se sont beaucoup accumulées. On a d'ailleurs pris soins de vérifier le résultat de valgrind afin de s'assurer qu'on libère bien toute la mémoire allouée.

```
[mohamed@archie source]$ valgrind --leak-check=yes ./exec
==209418== Memcheck, a memory error detector
==209418== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==209418== Using Valgrind-3.16.1 and LibVEX; rerun with -h for copyright info
==209418== Command: ./exec
==209418==
==209418==
==209418== HEAP SUMMARY:
==209418==     in use at exit: 0 bytes in 0 blocks
==209418==   total heap usage: 393,060 allocs, 393,060 frees, 9,507,504 bytes allocated
==209418==
==209418== All heap blocks were freed -- no leaks are possible
==209418==
==209418== For lists of detected and suppressed errors, rerun with: -s
==209418== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
[mohamed@archie source]$
```