Test et visualisation

Les objectifs de cette séance sont :

- de mettre en place une infrastructure de test pour les classes et méthodes du projet.
- de mettre en place une infrastructure de sauvegarde de données dans un format standardiser.

1 Mise en place de tests

Pour cela, nous allons utiliser le framework de test unitaire proposé par Google, `gtest` gtest. Lorsque les tests sont écrits, ils doivent correspondre aux critères suivants :

1. Les tests doivent être indépendants et répétables. Un test qui échoue ne doit en aucun cas entraîné l'échec (implicite) des autres tests.
2. Les tests doivent être bien organisés et refléter la structure du code testé. La cohérence des test est particulièrement pratique lors du développement d'une nouvelle base de code ou lorsque l'on rejoint un projet.
3. Les tests doivent être portables et réutilisables. Ils se doivent d'être indépendants de la plate-forme, du matériel, du système d'exploitation, a l'exception de fonctionnalités du code testé qui peut être plateforme dépendant.
4. Lorsque les tests échouent, ils doivent fournir autant d'informations que possible sur le problème. Il est préférable qu'ils ne s'arrêtent pas sur l'erreur mais poursuivent l'exécution pour l'ensemble du programme test.
5. Les tests doivent s'exécuter rapidement.

1.1 Concept de base

L'écriture de tests repose sur les assertions, déclarations vérifiant si une condition est vraie. Le résultat peut être un succès, un échec ou un échec fatal. En cas d'échec fatal, l'exécution de la fonction en cours est interrompue.

Une suite de test se compose de plusieurs tests. Une suite de test regroupe les tests qui reflètent la structure du code.

Un programme de test peut contenir plusieurs suites de test.

1.2 Assertion

Les assertions sont des macros. Le test d'une classe correspond à un test de son comportement.

Une assertion peut s'écrire de deux manières. `ASSERT_*` génèrent des échecs fatals lorsqu'elles échouent, et interrompent la fonction en cours. Les versions `EXPECT_*` génèrent des échecs non fatals, qui n'interrompent pas la fonction en cours.

Pour fournir un message d'échec personnalisé, il suffit de le diffuser dans la macro à l'aide de l'opérateur `«` ou d'une séquence de tels opérateurs.

```
1 ASSERT_EQ(x.size(), y.size()) « "Vectors x and y are of unequal length";
2
3 for (int i = 0; i < x.size(); ++i) {
4   EXPECT_EQ(x[i], y[i]) « "Vectors x and y differ at index " « i;
5 }
```

1.3 Tests

Pour créer un test :

1. On utilise la macro `TEST()` pour définir et nommer une fonction de test. Il s'agit de fonctions `C++` ordinaires qui ne renvoient pas de valeur.
2. Dans cette fonction, avec toutes les instructions `C++` valides que vous souhaitez inclure, utilisez les diverses assertions de pour vérifier les valeurs.
3. Le résultat du test est déterminé par les assertions ; si une assertion du test échoue (fatalement ou non), ou si le test se plante, le test entier échoue. Sinon, il réussit.

```
TEST(TestSuiteName, TestName) {
  ... corps de test ...
}
```

Pour illustrer l'utilisation des tests, prenons l'exemple de la fonction factorielle : For example, lets take a simple integer function :

```
int Factorial(int n);  // Renvoie la factorielle
```

Une suite de test pour la fonction factorielle peut s'écrire :

```
// Test factoriel de 0.
TEST(FactorialTest, HandlesZeroInput) {
  EXPECT_EQ(Factorial(0), 1);
}

// Tests factoriels de nombre positifs.
TEST(FactorialTest, HandlesPositiveInput) {
  EXPECT_EQ(Factorial(1), 1);
  EXPECT_EQ(Factorial(2), 2);
  EXPECT_EQ(Factorial(3), 6);
  EXPECT_EQ(Factorial(8), 40320);
}
```

Il ne s'agit que des bases de la création de tests.
Pour utiliser le framework `googletest`, il faut utiliser l'entête `##include <gtest/gtest.h>`.

1.4 Utiliser CMAKE

Il est possible de demander à `cmake` de s'occuper de charger les outils de tests

```
include(FetchContent)
FetchContent_Declare(
  googletest
  URL https://github.com/google/googletest/archive/git-commit-hash.zip
)
# Pour Windows: empêche le projet parent de forcer sa configuration de
↪  compilateur
```

```
set(gtest_force_shared_crt ON CACHE BOOL "" FORCE)
FetchContent_MakeAvailable(googletest)
```

Il suffit ensuite d'activer la gestion de test dans le `CMakeLists.txt`. Dans cet exemple, la mise en place est illustrée avec le fichier `hello_test.cc` et l'exécutable `hello_test`.

```
enable_testing()

add_executable(
  hello_test
  hello_test.cc
)
target_link_libraries(
  hello_test
  gtest_main
)

include(GoogleTest)
gtest_discover_tests(hello_test)
```

Question 1.
Ajouter une infrastructure de test à votre projet.

Question 2.
Ajouter des tests dans l'infrastructure pour les différentes classes de votre projet.

2 Visualisation

En visualisation scientifique, les données peuvent être soient visualisées en temps réel, soient sauvegardées dans un fichier structuré. Nous allons dans un premier temps sauvegarder les données dans une suite de fichier au format `xml` de vtk.

Ces fichiers peuvent être visualisés avec `Paraview` ou `Visit`.

L'entête du fichier contient les informations suivantes

```
<VTKFile type="UnstructuredGrid" version="0.1" byte_order="LittleEndian">
  <UnstructuredGrid>
    <Piece NumberOfPoints="3" NumberOfCells="0">
      <Points>
        <DataArray name="Position" type="Float32" NumberOfComponents="3"
        ↪  format="ascii">
          0 0 0 1 1 1 0 0 1
        </DataArray>
      </Points>
      <PointData Vectors="vector">
        <DataArray type="Float32" Name="Velocity" NumberOfComponents="3"
        ↪  format="ascii">
          4 4 4 4 0 0 2 2 -2
```

```
        </DataArray>
        <DataArray type="Float32" Name="Masse" format="ascii">
          0.1 0.5 1
        </DataArray>
      </PointData>
      <Cells>
        <DataArray type="Int32" Name="connectivity" format="ascii">
        </DataArray>
        <DataArray type="Int32" Name="offsets" format="ascii">
        </DataArray>
        <DataArray type="UInt8" Name="types" format="ascii">
        </DataArray>
      </Cells>
    </Piece>
  </UnstructuredGrid>
</VTKFile>
```

Pour un grand nombre de données, une version binaire du fichier est possible.
Les champs de données qui nous intéresse ici sont

- `NumberOfPoints` : nombre de points contenus dans le système.
- `Position` : les coordonnées de chaque particule. Il est possible de limiter le nombre de coordonnées.
- `Velocity` : la vitesse de chacune des particules.
- `Masse` : la masse de chacune des particules.

Les données sont stockées de manière continue dans chacun des champs prévus à cet effet.
Le format générique propose des éléments qui ne sont pas utilisés ici, mais qui permettent de sauvegarder des données de triangulations.

Question 3.
Intégrer dans votre projet une méthode pour enregistrer les données des particules.

Les fichiers générés peuvent être visualisés à l'aide du logiciel `Paraview`.

3 ACVL

Dans cette partie, nous allons procéder à l'analyse *a posteriori* des développements réalisés.

Question 4.
Réaliser les diagrammes des cas dutilisations.

Question 5.
Réaliser le diagramme de séquence.

Question 6.
Réaliser le diagramme de transitions.

Question 7.
Réaliser le diagramme de classes danalyse.