TP SMA
Outil de tests unitaires en C++
Bibliothèque Catch

Villedieu de Torcy Mathieu

17 mars 2022

Résumé

La bibliothèque de tests catch est une bibliothèque de tests unitaires pour des projets en C++. Dans sa version 1, elle supporte l'ensemble des versions C++ tandis que la version 2 ne prend en charge que du C++ 2011 ou ultérieur. Dans ce projet nous avons utilisé la version 1 de la bibliothèque catch.

L'un des principaux avantages de cette bibliothèques tient du fait que l'ensemble des fonctionnalités est contenu dans le fichier catch.hpp. Il suffit donc simplement d'inclure ce fichier et de définir une macro spécifique pour créer un main qui lancera les tests.

Les principaux défauts est la lenteur de compilation dû à la longueur du fichier catch.hpp et l'exécution n'est pas threadsafe.

1 Procédure d'installation

N'ayant utilisé que la version 1 de catch, nous n'aborderons pas la version 2 dans la suite du rapport.

La principale caractéristique de cette bibliothèque, comme évoqué plus haut, est qu'elle n'est contenue que dans un seul fichier. Ce fichier *catch.hpp* contient l'ensemble des fonctionnalités nécessaire pour réaliser des tests unitaires en CPP.

Pour installer la bibliothèque, il suffit de se rendre github [1] et de télécharger le fichier *catch.hpp*.
Pour l'inclure et mettre en place des tests unitaires dans le projet, une fois le fichier *catch.hpp* téléchargé, il faut créer un main spécifique.

```
1 #define CATCH_CONFIG_MAIN
2 #include "catch.hpp"
```

La macro *CATCH_CONFIG_MAIN* permet de créer la fonction main du programme qui lancera les tests définis dans un autre fichier.

2 Présentation des fonctionnalités

Les principales fonctions peuvent êtres des succès ou bien des échecs. Elles peuvent être bloquantes ou lever des exceptions.

On utilise les *macros* suivantes :

— CHECK (expression) en cas d'échec arrête la fonction
— REQUIRE (expression) continue même en cas d'échec, mais un affichage sera fait lors de l'exécution des tests

Ses macros attendent que l'expression passée en paramètre retourne la valeur true. Il existe leur équivalent pour tester une expression fausse : *CHECK_FALSE* et *REQUIRE_FALSE*.

Dans le cas où l'on souhaite tester des réel, la librairie permet des tests avec une certaine tolérance. En utilisant *Approx(réel)* dans le test l'expression est évaluée avec une tolérance sur l'égalité.

```
1 REQUIRE( valeur_a_verifier == Approx( 10.2 ) );
```

On peut également définir le seuil de tolérance de l'approximation.

— *Approx(100).epsilon(0.01)* permet un seuil de tolérance de 1% de la valeur, ie dans ce cas la valeur testée devra être comprise entre 99.0 et 100.0.
— *Approx(100).margin(5)* permet de définir un delta de tolérance, ie dans ce cas la valeur testée devra être comprise entre 95.0 et 105.0.

Lorsqu'une exception est levée, la condition est supposée fausse, mais il est possible de vérifier quelle exception est levée. Pour cela on utilise les macros suivantes :

— *CHECK_NOTHROW* et *REQUIRE_NOTHROW* où aucune exception ne doit être levée (comportement par défaut)
— *CHECK_THROWS* et *REQUIRE_THROWS* où une exception, qu'elle soit doit être levée pour que le test réussisse
— *CHECK_THROWS_AS* et *REQUIRE_THROWS_AS* où l'exception dont le type est spécifiée doit se produire

Exemple d'utilisation pour vérifier le lancement d'une exception :

```
1 REQUIRE_THROWS_AS( m.create(), std::bad_alloc );
```

3 Guide d'utilisation

Pour effectuer des tests sur notre code, on peut s'appuyer sur les macros :

— *TEST_CASE* qui définit un environnement de test
— *SECTION* qui réinitialise l'environnement de test

Cela permet de créer un ou plusieurs objets dans le l'environnement de test et grâce aux différentes *SECTIONs* de tester plusieurs situations, valeurs sur les méthodes de l'objet.

```
1  TEST_CASE("Deplacement agent")
2  {
3    Agent a{0, 0, EQUIPE::BLEU};
4
5    // Position (0,0)
6    REQUIRE(a.getX() == 0);
7    REQUIRE(a.getY() == 0);
8
9    SECTION("Deplacement Nord Ouest")
10   {
11     // Deplacement NORD OUEST
12     a.deplacer(DIRECTION::NORDOUEST);
13
14     REQUIRE(a.getX() == 0);
15     REQUIRE(a.getY() == -1);
16   }
17
18   SECTION("Deplacement Ouest")
19   {
20     // Deplacement OUEST
21     a.deplacer(DIRECTION::OUEST);
22
23     REQUIRE(a.getX() == -1);
24     REQUIRE(a.getY() == 0);
25   }
26   // Autres tests pour les différentes directions ...
27 }
```

Pour exécuter les tests, il faut simplement compiler de manière classique l'ensemble de fichiers *.cpp* utilisés avec *GCC* puis lancer l'exécutable produit.

Les tests sont alors effectués et une vue synthétique des résultats des tests est affichée dans la console.

```
mavilledie4@ada:~/shared/zz2/s12/ALQ/sma/src$ make test && ./test
g++ -MMD -g -O2 -fdiagnostics-color=auto -o build/Carte.o -c Carte.cpp
g++ -MMD -g -O2 -fdiagnostics-color=auto -o build/Agent.o -c Agent.cpp
g++ -MMD -g -O2 -fdiagnostics-color=auto -o build/Memoire.o -c Memoire.cpp
g++ -MMD -g -O2 -fdiagnostics-color=auto -o build/Manager.o -c Manager.cpp
g++ -MMD -g -O2 -fdiagnostics-color=auto -o build/Point.o -c Point.cpp
g++ -MMD -g -O2 -fdiagnostics-color=auto -o build/mt.o -c mt.cpp
g++ -MMD -g -O2 -fdiagnostics-color=auto -o build/main_test_agent_memoire.o -c m
oire.cxx
g++ -MMD -g -O2 -fdiagnostics-color=auto -o build/tests_catch.o -c tests_catch.c
g++ -o test build/Carte.o build/Agent.o build/Memoire.o build/Manager.o build/Po
uild/main_test_agent_memoire.o build/tests_catch.o
===============================================================================
All tests passed (46 assertions in 5 test cases)
```

FIGURE 1 – Tests unitaires réussis

Dans le cas où un test n'aurait pas réussi, la ligne ou le test a échoué est affichée avec la valeur testée et la valeur attendue.

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
test is a Catch v1.9.6 host application.
Run with -? for options

-------------------------------------------------------------------------------
Deplacement agent
  Deplacement Sud Ouest
-------------------------------------------------------------------------------
tests_catch.cxx:50
...............................................................................

tests_catch.cxx:55: FAILED:
  REQUIRE( a.getX() == 4 )
with expansion:
  -1 == 4

===============================================================================
test cases:  5 |  4 passed | 1 failed
assertions: 45 | 44 passed | 1 failed
```

FIGURE 2 – Tests unitaires : échec

4 Conclusion

La librairie *catch* permet de réaliser des tests unitaires simplement en *cpp*. Elle est rapide et simple à installer dans un projet. Elle propose des macros permettant de réaliser différents tests suivant les cas que l'on souhaite tester. Cela passe par des tests d'assertions classiques, avec ou sans nombres réels, aux tests de levés d'exceptions. Elle permet également de définir des environnements de tests pour réaliser successivement plusieurs tests suivant différentes valeurs à partir d'un même état initial.

Références

[1] *Github : Catch*. URL : https://github.com/catchorg/Catch2/tree/Catch1.x (visité le 2022).