

INF2990

Projet de logiciel graphique interactif

Tests logiciels

Version 14.3

Auteur original : Julien Gascon-Samson

Éditeur : Olivier Gendreau

# Tâche à réaliser

Dans le cadre du cours LOG1000, vous avez notamment vu le fonctionnement des tests unitaires. Vous avez travaillé avec l’outil *cppunit* pour créer vos tests unitaires, les maintenir et les exécuter. L’équipe académique a retenu cet outil pour le projet intégrateur INF2990. La création et l’exécution de cas de tests constituent une très bonne pratique du génie logiciel. Ainsi, pour intégrer le concept des tests unitaires à votre projet, nous vous demanderons de créer des suites de tests que vous maintiendrez à jour et exécuterez tout au long de la session. En vous référant au document « Aide au développement », vous trouverez plus d’informations sur la façon dont nous avons intégré *cppunit* au cadriciel de base du projet et sur la manière de générer vos jeux de tests.

Bien évidemment, il ne serait pas réaliste de vous demander de tester exhaustivement chacune des classes de votre projet. Dans la pratique, il est rare que des tests aussi exhaustifs soient demandés puisque cela demanderait trop de ressources. Une de vos tâches en tant que futur(e) ingénieur(e) logiciel consiste à identifier quelles classes il serait davantage pertinent de tester au moyen de tests unitaires. Sans donner de règle universelle, généralement, les tests les plus intéressants sont ceux qui visent à tester :

* les fonctions effectuant un calcul quelconque et retournant une valeur;
* les algorithmes;
* les structures de données;
* les conditions booléennes.

Nous vous demandons de créer des jeux de tests pour tester un minimum de 6 classes. Idéalement, la plupart des méthodes de ces classes devraient être testées. À cette fin, nous demandons que vous ayez des cas de tests pour couvrir un minimum de 20 méthodes (réparties à votre choix parmi les 6 classes à tester). Notez que vous n’avez pas l’obligation de tester uniquement du code que vous avez écrit; vous pouvez également tester du code faisant partie à la base du cadriciel, tel que la classe Vecteur ou l’arborescence des nœuds, par exemple.

Vous devez remplir un tableau pour chaque classe testée. En complément à la vérification de vos tableaux, le correcteur jettera un coup d’œil à vos jeux de tests lors de l’évaluation du livrable.

Vous trouverez, aux pages 5 et 6, un tableau informatif indiquant comment remplir les tableaux de tests, ainsi qu’un tableau d’exemples (pages 7 et 8), rempli pour les tests unitaires de la classe *NoeudAbstrait*. Notez que les tests unitaires pour cette classe sont fournis en exemple avec le cadriciel (classe *NoeudAbstraitTest*).

# Grille de correction

L’évaluation des tests logiciels vaut 5% du cours. Le correcteur appliquera la grille de correction détaillée ci-dessous. Cette grille tient compte des deux éléments suivants :

* La **pertinence** et la **justification** des cas de tests choisis parmi les différents jeux de tests, ainsi que les tableaux remplis;
* la **qualité** de l’implémentation des cas de tests.

## Choix des cas de tests

|  |  |
| --- | --- |
| **2,5 pts** | Les cas de tests choisis sont **très pertinents**.  **ET**  La justification est **très claire et complète**. |
| **2 pts** | Les cas de tests choisis sont **pertinents** ou **très pertinents**.  **ET**  La justification est **généralement claire** et **complète**. |
| **1,5 pt** | Les cas de tests choisis sont **pertinents**.  **ET**  La justification est **moyennement claire** et/ou **incomplète**. |
| **1 pt** | Les cas de tests choisis sont **peu pertinents**.  **ET/OU**  La justification est **floue** ou **incomplète**. |
| **0 pt** | Les cas de tests choisis ne sont **pas pertinents** ou sont **manquants**.  **ET/OU**  La justification est **très floue**, **incomplète** ou **absente**. |

## Implémentation des cas de tests

|  |  |
| --- | --- |
| **2,5 pts** | La qualité de l’implémentation des cas de tests est **excellente**. |
| **2 pts** | La qualité de l’implémentation des cas de tests est **très bonne**. |
| **1,5 pt** | La qualité de l’implémentation des cas de tests est **bonne**. |
| **1 pt** | La qualité de l'implémentation des cas de tests est **médiocre**. |
| **0 pt** | La qualité de l'implémentation des cas de tests est **nettement insuffisante**. |

|  |  |  |
| --- | --- | --- |
| ***Jeu de tests i*** | | |
| **Classe :** | *Nom de la classe testée (et si pertinent, le nom de la classe du jeu de tests)* | |
| **Justification :**  *(pertinence de tester cette classe)*  *5 lignes max.* | *Décrivez sommairement pourquoi avez choisi de tester cette classe et la pertinence de la tester au moyen de tests unitaires. (maximum 5 lignes)* | |
| **Cas test #1 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *10 lignes max* | *Décrivez brièvement chacun de vos cas de tests. Justifiez brièvement le choix des valeurs en entrée pour les tests et indiquez toute autre information pertinente. (maximum 5 lignes)* | *Copiez-collez ici le code source de votre cas de test* |
| **Cas test #2 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max* | *Décrivez brièvement chacun de vos cas de tests. Justifiez brièvement le choix des valeurs en entrée pour les tests et indiquez toute autre information pertinente. (maximum 5 lignes)* | *Copiez-collez ici le code source de votre cas de test* |
| **Cas test #3 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max* | *Décrivez brièvement chacun de vos cas de tests. Justifiez brièvement le choix des valeurs en entrée pour les tests et indiquez toute autre information pertinente. (maximum 5 lignes)* | *Copiez-collez ici le code source de votre cas de test* |
| **Cas test #4 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* | *Décrivez brièvement chacun de vos cas de tests. Justifiez brièvement le choix des valeurs en entrée pour les tests et indiquez toute autre information pertinente. (maximum 5 lignes)* | *Copiez-collez ici le code source de votre cas de test* |
| **Cas test #5 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* | *Décrivez brièvement chacun de vos cas de tests. Justifiez brièvement le choix des valeurs en entrée pour les tests et indiquez toute autre information pertinente. (maximum 5 lignes)* | *Copiez-collez ici le code source de votre cas de test* |
| **Cas test #6 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* | *Décrivez brièvement chacun de vos cas de tests. Justifiez brièvement le choix des valeurs en entrée pour les tests et indiquez toute autre information pertinente. (maximum 5 lignes)* | *Copiez-collez ici le code source de votre cas de test* |
| **Cas test #7 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* | *Décrivez brièvement chacun de vos cas de tests. Justifiez brièvement le choix des valeurs en entrée pour les tests et indiquez toute autre information pertinente. (maximum 5 lignes)* | *Copiez-collez ici le code source de votre cas de test* |
| **Cas test #8 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* | *Décrivez brièvement chacun de vos cas de tests. Justifiez brièvement le choix des valeurs en entrée pour les tests et indiquez toute autre information pertinente. (maximum 5 lignes)* | *Copiez-collez ici le code source de votre cas de test* |

|  |  |  |
| --- | --- | --- |
| ***Jeu de tests #i - Exemple*** | | |
| **Classe :** | NoeudAbstrait (NoeudAbstraitTest) | |
| **Justification :**  *(pertinence de tester cette classe)*  *5 lignes max.* | Nous testons ici un nœud non-composite*.* Il est intéressant de tester cette classe puisque nous pouvons vérifier le bon fonctionnement des méthodes à la base de l’arbre de rendu. Nous testons notamment certaines conditions booléennes et nous assurerons qu’aucun nœud « enfant » ne peut être ajouté. | |
| **Cas test #1 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *10 lignes max* | testPositionRelative : nous assignons une position relative quelconque et vérifions si la position relative obtenue (set-get) est identique. | void NoeudAbstraitTest::testPositionRelative()  {  noeud->assignerPositionRelative(Vecteur3(2.2, 3.3, 4.4));  Vecteur3 vecteur = noeud->obtenirPositionRelative();  CPPUNIT\_ASSERT( utilitaire::EGAL\_ZERO(vecteur[0] - 2.2) );  CPPUNIT\_ASSERT( utilitaire::EGAL\_ZERO(vecteur[1] - 3.3) );  CPPUNIT\_ASSERT( utilitaire::EGAL\_ZERO(vecteur[2] - 4.4) );  } |
| **Cas test #2 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max* | testType : on s’assure que le type du nœud est celui qui est attendu selon l’instance concrète utilisé (nœud cône-cube) | void NoeudAbstraitTest::testType()  {  CPPUNIT\_ASSERT( noeud->obtenirType() == ArbreRenduINF2990::NOM\_CONECUBE );  } |
| **Cas test #3 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max* | testEtat : nous testons la définition (set) et l’obtention (get) de certains états booléens : « affiche » et « estEnregistrable », pour les valeurs *true* et *false*. Nous nous assurons que les états récupérés correspondent aux états attendus. | void NoeudAbstraitTest::testEtat()  {  // Régler tous les états (booléens) à faux  noeud->assignerAffiche(false);  noeud->assignerEstEnregistrable(false);  // Vérifier que la valeur attribuée est fausse  CPPUNIT\_ASSERT( noeud->estAffiche() == false );  CPPUNIT\_ASSERT( noeud->estEnregistrable() == false );  // Régler tous les états (booléens) à true  noeud->assignerAffiche(true);  noeud->assignerEstEnregistrable(true);  // Vérifier que la valeur attribuée est fausse  CPPUNIT\_ASSERT( noeud->estAffiche() == true );  CPPUNIT\_ASSERT( noeud->estEnregistrable() == true );  } |
| **Cas test #4 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* | testSelection : un nœud peut être sélectionné seulement s’il est sélectionnable. Il s’agit d’un état booléen qui dépend aussi d’un autre état. Nous testons chaque paire conditionnelle possible (vrai-vrai, vrai-faux, faux-vrai, faux-faux) pour nous assurer que les états sont cohérents avec ce qui est attendu. | void NoeudAbstraitTest::testSelection()  {  // Cas 1 : non sélectionnable, non sélectionné  noeud->assignerEstSelectionnable(false);  noeud->assignerSelection(false);  CPPUNIT\_ASSERT( noeud->estSelectionnable() == false );  CPPUNIT\_ASSERT( noeud->estSelectionne() == false );  // Cas 2 : non sélectionnable, sélectionné  noeud->assignerEstSelectionnable(false);  noeud->assignerSelection(true);  CPPUNIT\_ASSERT( noeud->estSelectionnable() == false );  CPPUNIT\_ASSERT( noeud->estSelectionne() == false );  // Cas 3 : sélectionnable, non sélectionné  noeud->assignerEstSelectionnable(true);  noeud->assignerSelection(false);  CPPUNIT\_ASSERT( noeud->estSelectionnable() == true );  CPPUNIT\_ASSERT( noeud->estSelectionne() == false );  // Cas 4 : non sélectionnable, sélectionné  noeud->assignerEstSelectionnable(true);  noeud->assignerSelection(true);  CPPUNIT\_ASSERT( noeud->estSelectionnable() == true );  CPPUNIT\_ASSERT( noeud->estSelectionne() == true );  } |
| **Cas test #5 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* | testEnfants : nous nous assurons ici que les fonctions *obtenirNombreEnfants* et *calculerProfondeur* retournent les valeurs attendues pour un nœud non-composite (0 enfants et profondeur de 1). | void NoeudAbstraitTest::testEnfants()  {  CPPUNIT\_ASSERT( noeud->obtenirNombreEnfants() == 0);  CPPUNIT\_ASSERT( noeud->calculerProfondeur() == 1);  } |
| **Cas test #6 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* | testAjout : nous testons l’ajout d’un nœud enfant. Bien entendu, puisqu’il s’agit d’un nœud non-composite, l’ajout ne devrait pas être permis. Nous nous assurons ici que l’ajout n’est pas réalisé. | void NoeudAbstraitTest::testAjout()  {  // Essaie d'ajouter un noeud  NoeudAbstrait\* nouveauNoeud = new NoeudConeCube(ArbreRenduINF2990::NOM\_CONECUBE);  bool ajout = noeud->ajouter(nouveauNoeud);  // L'ajout devrait avoir échoué puisqu'il s'agit d'un noeud abstrait...  CPPUNIT\_ASSERT( ajout==false );  // Assurons-nous que le noeud ne possède pas d'enfant...  CPPUNIT\_ASSERT( noeud->obtenirNombreEnfants() == 0 );  // Nettoyage  delete nouveauNoeud;  } |
| **Cas test #7 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #8 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |

|  |  |  |
| --- | --- | --- |
| ***Jeu de tests #1*** | | |
| **Classe :** | ProjectionOrtho (ProjectionOrthoTest) | |
| **Justification :**  *(pertinence de tester cette classe)*  *5 lignes max.* | Cette classe contient les manipulations de la vue exigés pour le livrable 1. Il est important que cette classe fonctionne adéquatement, car c’est à travers laquelle tout le jeux est visualisé quand nous sommes en mode projection orthogonal. Elle est responsable pour l’effet de loupe avec clavier/souris, l’effet de loupe avec rectangle elastique, le dimensionnement et le déplacement de la vue. | |
| **Cas test #1 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *10 lignes max* | testZoomerIn: nous effectuons un zoom in en appelant la méthode zoomerIn() et nous vérifions que le les coordonnées de la fenêtre relative on été modifier de la bonne manière. |  |
| **Cas test #2 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max* | testZoomerOut: nous effectuons un zoom out en appelant la méthode zoomerOut() et nous vérifions que le les coordonnées de la fenêtre relative on été modifier de la bonne manière. |  |
| **Cas test #3 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max* |  |  |
| **Cas test #4 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #5 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #6 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #7 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #8 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |

|  |  |  |
| --- | --- | --- |
| ***Jeu de tests #2*** | | |
| **Classe :** | VisiteurTranslation (VisiteurTranslationTest) | |
| **Justification :**  *(pertinence de tester cette classe)*  *5 lignes max.* | Le visiteur translation est responsable de réaliser la translation sur les nœuds. Le bon fonctionnement de cette classe est primordial pour que nous puissions repositionner des objets sur la zone de jeux dans le mode éditeur. | |
| **Cas test #1 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *10 lignes max* | testVisit : | noeud1 = new NoeudAbstrait{ ArbreRenduINF2990::NOM\_CIBLE }; std::vector<NoeudAbstrait\*> noeudSelectionnes; noeudSelectionnes.push\_back(noeud1); monVisiteur = new VisiteurTranslation(noeudSelectionnes); glm::dvec3 mouvement(5.0, 10.0, 15.0); monVisiteur->visiteAll(mouvement); CPPUNIT\_ASSERT(noeud1->obtenirPositionRelative() == mouvement); delete monVisiteur; |
| **Cas test #2 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max* | testRevert : | //Position Initial glm::dvec3 positionInit(0.0,0.0,0.0) ; //Deplacement glm::dvec3 mouvement(1.0, 0.0, 0.0); noeud1 = new NoeudAbstrait{ ArbreRenduINF2990::NOM\_CIBLE }; std::vector<NoeudAbstrait\*> noeudSelectionnes; noeudSelectionnes.push\_back(noeud1); monVisiteur = new VisiteurTranslation(noeudSelectionnes); monVisiteur->visiteAll(mouvement); monVisiteur->revert(); CPPUNIT\_ASSERT(noeud1->obtenirPositionRelative() == positionInit); delete monVisiteur; |
| **Cas test #3 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max* |  |  |
| **Cas test #4 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #5 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #6 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #7 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #8 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |

|  |  |  |
| --- | --- | --- |
| ***Jeu de tests #3*** | | |
| **Classe :** | NoeudComposite (NoeudCompositeTest) | |
| **Justification :**  *(pertinence de tester cette classe)*  *5 lignes max.* | Dans le cadre de ce projet, le nœud composite est le nœud qui garde les objets de la scène sous forme de nœuds enfants.  Grace à lui il existe la possibilité de traiter les objets comme un ensemble unis pour y effectuer les manipulations voulue. | |
| **Cas test #1 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *10 lignes max* | testChercher : | NoeudAbstrait\* noeudFils = new NoeudAbstrait{ ArbreRenduINF2990::NOM\_CIBLE }; noeudComposite->ajouter(noeudFils); CPPUNIT\_ASSERT(noeudComposite->chercher(ArbreRenduINF2990::NOM\_CIBLE) == noeudFils); } |
| **Cas test #2 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max* | testAjouter : | NoeudAbstrait\* noeudfils1 = new NoeudAbstrait{ ArbreRenduINF2990::NOM\_CIBLE }; NoeudAbstrait\* noeudfils2 = new NoeudAbstrait{ ArbreRenduINF2990::NOM\_TROU}; noeudComposite->ajouter(noeudfils1); noeudComposite->ajouter(noeudfils2); noeudComposite->chercher(ArbreRenduINF2990::NOM\_TROU); CPPUNIT\_ASSERT(noeudComposite->chercher(ArbreRenduINF2990::NOM\_TROU) == noeudfils2); |
| **Cas test #3 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max* | testEffacer : | NoeudAbstrait\* noeudFils2 = new NoeudAbstrait{ ArbreRenduINF2990::NOM\_TROU }; noeudComposite->effacer(noeudFils2); CPPUNIT\_ASSERT(noeudComposite->chercher(ArbreRenduINF2990::NOM\_TROU) == nullptr); |
| **Cas test #4 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* | testSelectionExiste : | NoeudAbstrait\* noeudFils1 = new NoeudAbstrait{ ArbreRenduINF2990::NOM\_CIBLE }; noeudComposite->ajouter(noeudFils1); noeudFils1->inverserSelection(); CPPUNIT\_ASSERT(noeudComposite->selectionExiste()); |
| **Cas test #5 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* | testSelectionnerTout : | NoeudAbstrait\* noeudFils1 = new NoeudAbstrait{ ArbreRenduINF2990::NOM\_CIBLE }; NoeudAbstrait\* noeudFils2 = new NoeudAbstrait{ ArbreRenduINF2990::NOM\_TROU }; noeudComposite->ajouter(noeudFils1); noeudComposite->ajouter(noeudFils2); noeudComposite->selectionnerTout(); CPPUNIT\_ASSERT(noeudComposite->chercher(ArbreRenduINF2990::NOM\_TROU)->estSelectionne() && noeudComposite->chercher(ArbreRenduINF2990::NOM\_CIBLE)->estSelectionne()); |
| **Cas test #6 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #7 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #8 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |

|  |  |  |
| --- | --- | --- |
| ***Jeu de tests #4*** | | |
| **Classe :** | VisiteurRotation (VisiteurRotationTest) | |
| **Justification :**  *(pertinence de tester cette classe)*  *5 lignes max.* | Le visiteur rotation est responsable de réaliser les rotations sur les nœuds. Pour que nous puissions rotionner des objets ou des ensembles d’objets sur la zone de jeux ce visiteur dois être fonctionnel. | |
| **Cas test #1 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *10 lignes max* | testVisit : | glm::dvec3 centre(0.0, 0.0, 0.0); noeud1 = new NoeudAbstrait{ ArbreRenduINF2990::NOM\_CIBLE }; std::vector<NoeudAbstrait\*> noeudSelectionnes; noeudSelectionnes.push\_back(noeud1); monVisiteur = new VisiteurRotation(centre, noeudSelectionnes); glm::dvec3 rotation(1.0,0.,0.); monVisiteur->visiteAll(rotation); CPPUNIT\_ASSERT(noeud1->obtenirAngleRotation() == rotation[0]); delete monVisiteur; |
| **Cas test #2 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max* | testRevert : | glm::dvec3 centre(0.0, 0.0, 0.0); noeud1 = new NoeudAbstrait{ ArbreRenduINF2990::NOM\_CIBLE }; std::vector<NoeudAbstrait\*> noeudSelectionnes; noeudSelectionnes.push\_back(noeud1); monVisiteur = new VisiteurRotation(centre, noeudSelectionnes); glm::dvec3 rotation(1.0, 0., 0.); monVisiteur->visiteAll(rotation); monVisiteur->revert(); // le revert consiste a 1% de diminution CPPUNIT\_ASSERT(noeud1->obtenirAngleRotation() == 0.99\*rotation[0]); delete monVisiteur; |
| **Cas test #3 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max* |  |  |
| **Cas test #4 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #5 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #6 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #7 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #8 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |

|  |  |  |
| --- | --- | --- |
| ***Jeu de tests #5*** | | |
| **Classe :** | ArbreRenduINF2990 (ArbreRenduINF2990Test) | |
| **Justification :**  *(pertinence de tester cette classe)*  *5 lignes max.* |  | |
| **Cas test #1 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *10 lignes max* | testInitialiser : | monArbre->initialiser(); CPPUNIT\_ASSERT(monArbre->obtenirNombreEnfants() == 1); |
| **Cas test #2 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max* |  |  |
| **Cas test #3 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max* |  |  |
| **Cas test #4 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #5 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #6 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #7 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #8 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |

|  |  |  |
| --- | --- | --- |
| ***Jeu de tests #6*** | | |
| **Classe :** |  | |
| **Justification :**  *(pertinence de tester cette classe)*  *5 lignes max.* |  | |
| **Cas test #1 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *10 lignes max* |  |  |
| **Cas test #2 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max* |  |  |
| **Cas test #3 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max* |  |  |
| **Cas test #4 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #5 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #6 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #7 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |
| **Cas test #8 :**  *(description, choix valeurs, autres infos et*  *code source du cas de test)*  *(si nécessaire)*  *10 lignes max.* |  |  |