



Projet 30 : Graphe3D
API Java pour la visualisation de graphes mathématiques en 3D

DOSSIER DE TESTS

Objet du document	Dossier de tests du projet
Destinataires du document	Franck Mosser
Chef de projet	Jérôme Catric
Groupe	DAUBERT Erwan, LINO Christophe, MAGNIN Nicolas, POPA Iuliana, L3 INFO
Référence projet	n°30
Version/ date	2.0 06/04/2007

Validation		
Chef de projet	Catric Jérôme	
Groupe	Daubert Erwan	
	LinoChristophe	
	Magnin Nicolas	
	Popa Iuliana	
Tuteur	Bonnel Nicolas	

Table des matières

Introduction.....	3
I. Schéma fonctionnel.....	4
I.1. schéma général.....	4
I.2. Schéma de la visualisation.....	5
I.3. Schéma de l'édition et de la conception.....	6
II. Schéma technique.....	7
II.1. Portabilité de Graphe3D.....	7
III. Tests.....	8
III.1. Tests fonctionnels.....	8
III.2. Tests de performance.....	9
III.3. Tests techniques.....	10
IV. Fiches de tests.....	11
IV.1. Fiche de test fonctionnel : chargement/sauvegarde > chargement.....	11
IV.2. Fiche de test fonctionnel : chargement/sauvegarde > sauvegarde.....	14
IV.3. Fiche de test fonctionnel : Vue3D > gestion des collisions.....	16
IV.4. Annexes.....	18
V. Plan qualité.....	21
V.1 Homogénéité du code.....	21
V.2 Qualité du code.....	22
V.3 Automatisation des tâches.....	22

Introduction

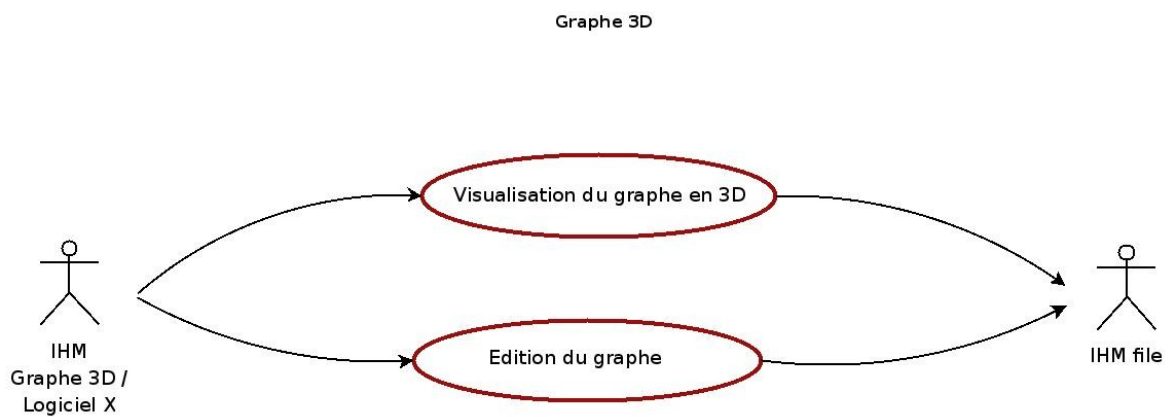
Ce document est un dossier de tests associé au projet de Licence 3 intitulé Graphe 3D, qui consiste en la conception d'une API java pour la visualisation de graphes mathématiques en 3 dimensions.

Ce dossier traite de l'organisation de la phase de tests (fonctionnels, techniques et de performance) et constitue la ligne directrice des phases de codage et de tests de l'API. Il complète le dossier de conception Version 2.0.

I. Schéma fonctionnel

I.1. schéma général

Shéma fonctionnel général

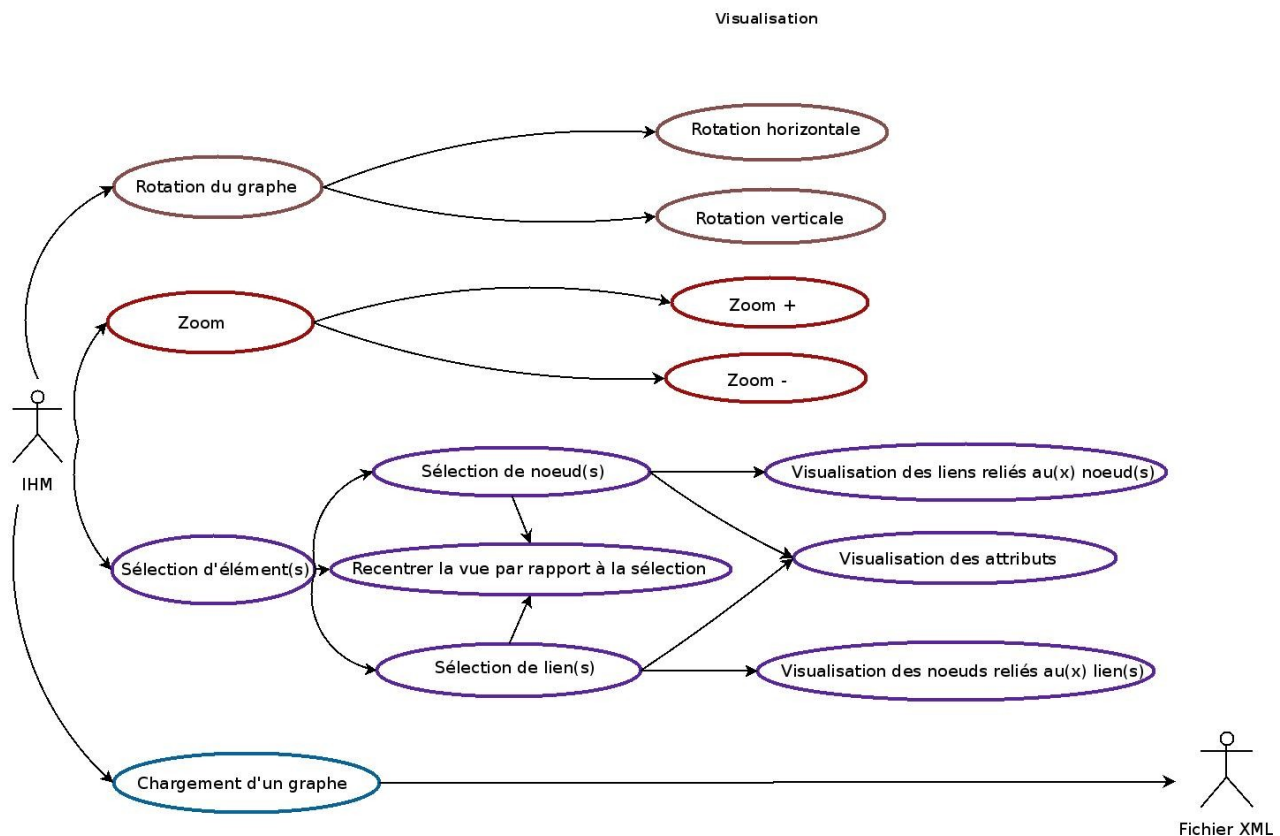


L'API Graphe3D permettra deux types d'utilisation.

La visualisation de graphe mathématique en 3 dimension ainsi que l'édition et la conception de graphe mathématique.

1.2. Schéma de la visualisation

Shéma fonctionnel de la visualisation



La visualisation permettra à l'utilisateur, au travers de l'interface de charger un graphe mathématique (à partir de son fichier XML) afin de voir sa représentation en 3 dimension.

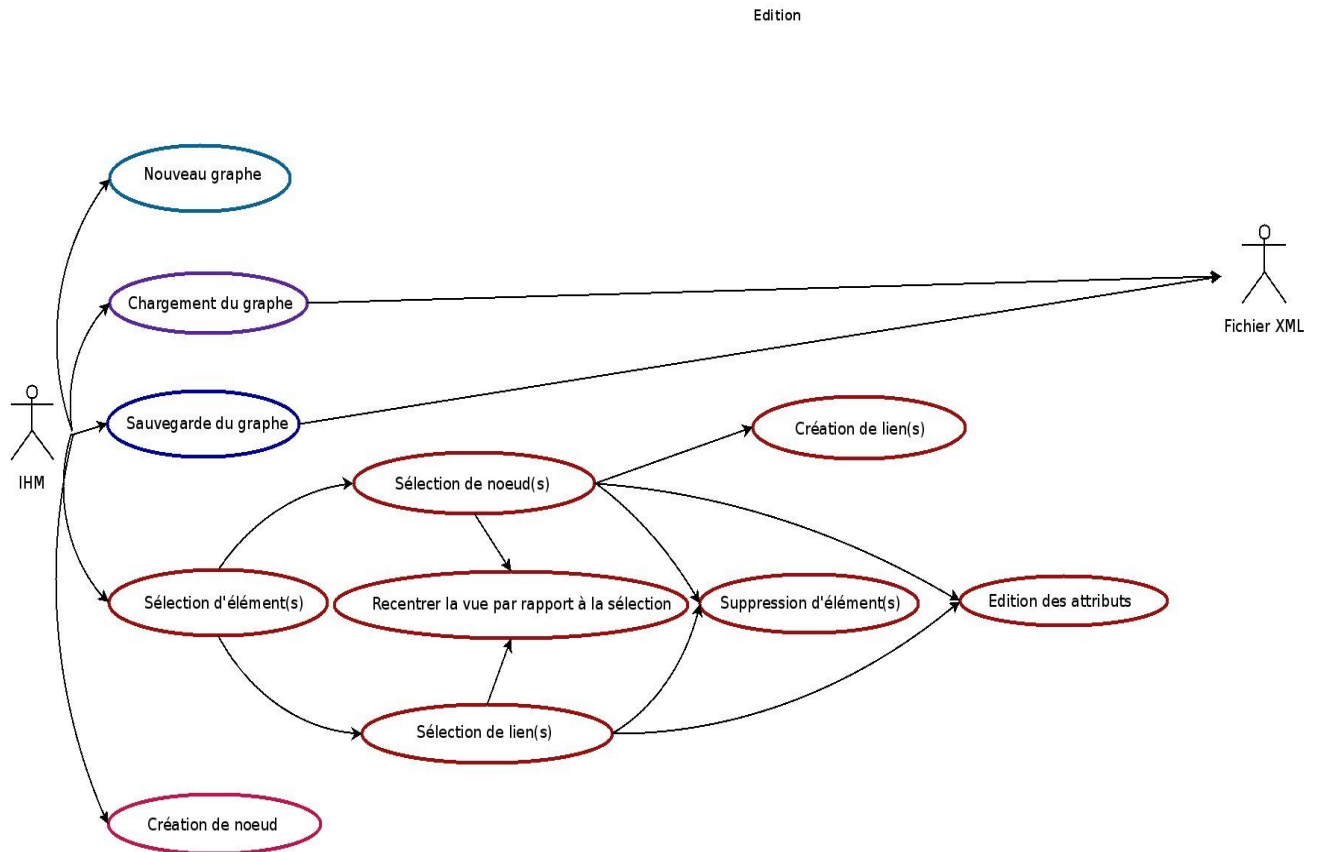
Une fois un graphe chargé, l'utilisateur pourra manipuler le rendu à sa guise grâce à la possibilité de zoomer sur ce rendu (zoom + ou zoom -), d'effectuer des rotations (horizontales ou verticales) et enfin de sélectionner des éléments composant le graphe.

La sélection permettra de visualiser les attributs de l'élément sélectionnés ainsi que les autres éléments qui lui sont raccordés (exclusivement des liens pour les noeuds sélectionnés et des noeuds pour les liens sélectionnés).

La sélection donne aussi la possibilité de recentrer le rendu sur elle.

1.3. Schéma de l'édition et de la conception

Shéma fonctionnel édition



L'édition et la conception de graphe, quant à elle permettra de charger un graphe (toujours à partir de son fichier XML) afin de le modifier ou de concevoir un nouveau graphe. Une fois les modification et/ou la conception effectués, il est possible de sauvegarder le graphe dans son fichier XML.

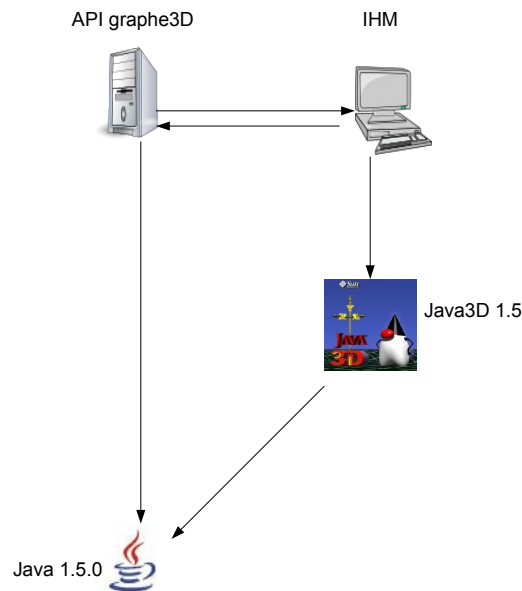
Pour pouvoir éditer on créer un graphe, il est possible de créer de nouveau noeuds.

Enfin, la modification d'éléments déjà existants se fait grâce à la sélection de ceux-ci. Une fois sélectionner, il est possible de supprimer l'élément ainsi que d'éditer ses attributs. Si le ou les éléments sélectionnés sont des noeuds, il est possible de les connecter entre eux en créant des liens.

Comme pour la visualisation, il est possible de recentrer la vue sur les éléments sélectionnés afin de faciliter leur manipulation.

II. Schéma technique

II.1. Portabilité de Graphe3D



L'API graphe3D s'appuie sur la technologie Java et l'interface homme-machine sur Java3D qui est une API basé sur Java permettant de faire de la visualisation en 3D.

Étant donné l'utilisation de Java, et ses normes qui assurent une certaine portabilité des applications, nous pouvons dire que notre API pourra être utilisé aussi bien sous Windows, Linux, Macintosh.

Il faudra cependant que, sur ces machines, soit disponible l'API Java3D (disponible sur <https://java3d.dev.java.net/> rubrique "download").

Carte graphique et drivers

Il faudra aussi prendre en compte la carte graphique disponible sur les machines étant donné leur utilisation par l'API Java3D. Notamment pour les drivers installés. En effet, nous avons pu nous apercevoir que selon le driver de la carte, les performances de Java3D sont très différentes. Principalement pour Windows Vista ainsi que sous Linux.

L'idéal concernant les drivers serait d'installer le driver officiel fournir par l'éditeur et disponible sur son site internet.

Configurations testées :

Nous pourrons tester sur aux moins 4 machines de configuration différentes ainsi que sur 4 systèmes d'exploitations(Windows XP, Vista, Linux Ubuntu, Mandriva)

Pour plus de précision sur les configurations qui seront testées, aller voir la partie test technique.

III. Tests

III.1. Tests fonctionnels

1) Chargement/Sauvegarde

1. Chargement

- Vérification avec le parser que le graphe chargé est valide : le fichier XML doit respecter le schéma XSD.
- On ne doit pas pouvoir charger un fichier qui n'est pas un fichier XML (dans la fenêtre de chargement seulement les fichiers en .xml doivent être visibles).
- Au chargement tous les éléments doivent être visibles, la vue doit contenir tout le graphe3D.

2. Sauvegarde

- La sauvegarde du graphe doit s'effectuer dans un fichier XML valide, le chargement de ce fichier avec le parser doit représenter le même graphe.

2) Vue 3D

1. Collisions

- Gestion des collisions, deux noeuds ne peuvent pas être superposés dans la vue, leurs coordonnées doivent être différentes :
 - lors d'un placement automatique après un chargement.
 - modification des coordonnées par l'utilisateur.

2. Zoom

- Pas de limites pour le zoom, l'effet du zoom doit être progressif avec la gestion de rester appuyer sur le bouton du zoom.
- Vérification du zoom plus (+) respectivement du zoom moins (-).
- Il doit être possible de zoomer sur les éléments qui ont été sélectionnés (zoom sur une sélection simple ou multiple).

3. Rotation (dans Java3D)

- Vérification des rotations :
 - verticales
 - horizontales
 - recentrer la vue sur les élément(s) sélectionné(s)

4. Sélection

- "Feedback" dans la vue 3D, mise en valeur du ou des éléments sélectionnés.
- Mise à jour correct de la liste des caractéristiques des éléments sélectionnés ainsi que de la liste des objets reliés.
- Possibilité d'une sélection simple ou multiple (CTRL + clic sur les éléments avec la souris).

III.2. Tests de performance

1) Édition

- Modification de l'attribut d'un élément : rafraîchissement de la vue 3D.
- Test des coordonnées pour la gestion des collisions lors :
 - de la création d'un noeud,
 - modification des coordonnées d'un noeud.
- La suppression d'un noeud plus la suppression des liens auxquels il est raccordé si nécessaire.
- La création ou suppression d'un lien.
- Suppression multiple : suppression de plusieurs éléments sélectionnés.
- Dé-sélection des éléments dans les listes lors de leur suppression.

1) Java3D

- Éviter une montée en charge importante pour des graphes de taille importante.
- Pas d'erreur d'exécution de Java3D .
- Le temps processeur et mémoire ne doivent pas être trop élevés.
- Mise à jour des drivers de carte graphique parfois nécessaire pour Java3D (Linux principalement).

III.3. Tests techniques

- On utilise Java 1.5 / 1.6 et Java 3D 1.5.
- Nous ne pourrions tester notre application que sous Windows et Linux cependant Java respecte des normes qui nous permette de dire que notre application Java3D pourra fonctionner sous Linux / Windows / Mac.
- Nous ne pouvons certifier le fonctionnement de notre API Graphe3D pour les versions précédentes de Java 1.5.
- Configuration des ordinateurs sur lesquels l'API Graphe3D sera testée :

Ordinateur Portable DELL INSPIRON 6400

2 Go de RAM

Intel Centrino Duo 2 GHz

DD 120 Go

ATI Mobility X1400 – 256 RAM

OS : Windows Vista, Linux : Mandriva 2007 noyau : 2.6.17

Ordinateur Portable DELL INSPIRON 9100

1 Go de RAM

Pentium 4 3,4 GHz

DD 80 Go

ATI Mobility 9700 – 128 RAM

OS : Windows XP, Linux : UBUNTU

Ordinateur Portable DELL INSPIRON 8600

512 Go de RAM

Intel centrino 1,7 GHz

DD 60 Go

ATI Mobility 9600

OS : Windows XP Sp2, Linux : UBUNTU 7.03 noyau : 2.6.20

Ordinateur Portable DELL INSPIRON 9400

1 Go de RAM

Intel Centrino Duo 2 GHz

ATI Mobility X1400 – 256 RAM

OS : Windows XP Média Center, Linux : Mandriva 2007

IV. Fiches de tests

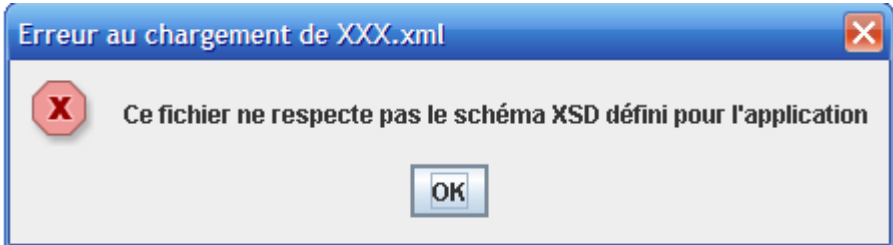
IV.1. Fiche de test fonctionnel : chargement/sauvegarde > chargement.

Nom : _____ prenom : _____ date : __ / __ / ____

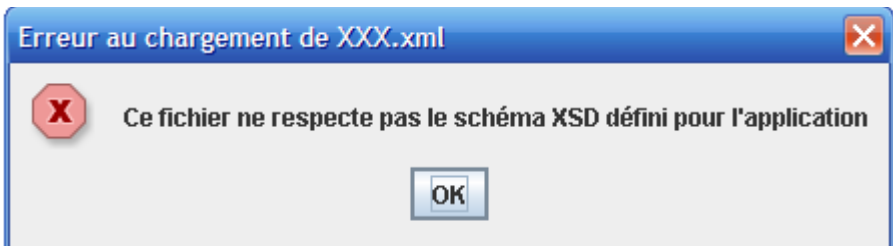
Prérequis : avoir lancé l'application.

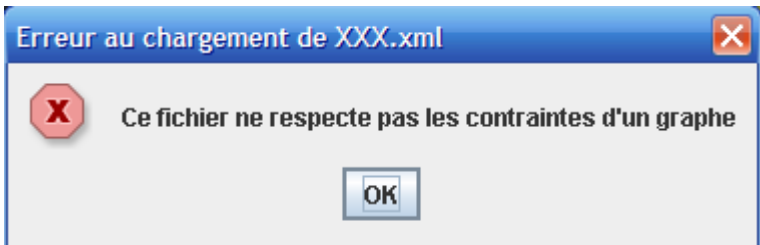
Données d'entrée : chargement de fichiers contenant des graphes

	NOM(S) DU OU DES FICHIERS	TEST RÉUSSI
Fichiers non suffixé .xml	nonValide.truc	<input type="checkbox"/> Oui <input type="checkbox"/> Non
Résultat attendu : <i>l'application doit afficher le message d'erreur suivant :</i> <div data-bbox="269 824 1075 1075" data-label="Image"></div> <i>et l'application ne charge pas de graphe.</i>		
Remarques :		

Fichiers ne respectant pas le schéma XSD	nonValideXsd1.xml (annexe 1)	
Résultat attendu : <i>l'application doit afficher le message d'erreur suivant :</i>		<input type="checkbox"/> Oui <input type="checkbox"/> Non
		
Remarques :		

Graphe3D : API Java pour la visualisation de graphes mathématiques en 3D

Fichiers ne respectant pas le schéma XSD	nonValideXsd2.xml (annexe 1)	
Résultat attendu : <i>l'application doit afficher le message d'erreur suivant :</i>		<input type="checkbox"/> Oui <input type="checkbox"/> Non
		
Remarques :		

Fichier non valide sémantiquement	nonValideSem.xml (annexe 2)	
Résultat attendu : <i>l'application doit afficher le message d'erreur suivant :</i>		<input type="checkbox"/> Oui <input type="checkbox"/> Non
		
Remarques :		

Fichier valide	valide.xml (annexe 3)	
Résultat attendu : <i>Aucun message d'erreur ne doit s'afficher.</i> <i>L'application doit afficher une vue qui corresponde au graphe.</i> <i>Tout le graphe doit être visible dans la vue.</i>		<input type="checkbox"/> Oui <input type="checkbox"/> Non
Remarques :		

Graphe3D : API Java pour la visualisation de graphes mathématiques en 3D

Remarque :


Tous ces tests de chargement pourront être automatisés. Pour cela nous pourront utiliser une classe de tests automatiques en appelant la fonction de chargement, en vérifiant les erreurs soulevées ou pas, et en répercutant les résultats des tests (« test de chargement de XXX.xxx : OK ou KO » par exemple) dans un fichier de synthèse dont le nom comportera la date des tests.


IV.2. Fiche de test fonctionnel : chargement/sauvegarde > sauvegarde.

Nom : _____ prenom : _____ date : __/__/____

Prérequis : avoir lancé l'application et chargé le fichier « valide.xml » (cf Annexe 1).

Données d'entrée : sauvegarde du graphe dans un fichier. Le nom du fichier est celui qui doit être passé en paramètre à l'application pour la sauvegarde.

	NOM DU FICHIER	TEST RÉUSSI
Nom de fichier non suffixé .xml	nonvalide.doc	<div><input type="checkbox"/> Oui</div> <div><input type="checkbox"/> Non</div>
Résultat attendu : <i>l'application doit afficher le message d'erreur suivant :</i> <div><div>Erreur de sauvegarde</div><div> Le nom de fichier n'est pas valide ! Vous devez absolument enregistrer votre graphe au format XML.</div><div>OK</div></div>		
Remarques :		

Nom de fichiers incorrect pour le système de fichiers	non:valide?.xml	<input type="checkbox"/> Oui <input type="checkbox"/> Non
Résultat attendu : <i>l'application doit afficher le message d'erreur suivant :</i>		
<div><div>Erreur de sauvegarde</div><div><div>non:valide?.xml Ce nom de fichier n'est pas valide.</div><div>OK</div></div></div>		
Remarques :		

Graphe3D : API Java pour la visualisation de graphes mathématiques en 3D

Nom de fichier correct	sauvegarde.xml	
Résultat attendu : <i>l'application ne doit pas afficher de message d'erreur lors de la sauvegarde.</i> <i>Vous devez alors vérifier qu'en re-chargeant le fichier « sauvegarde.xml » :</i> <ul style="list-style-type: none">– aucun message d'erreur n'apparaît.– votre graphe n'a pas changé.		<input type="checkbox"/> Oui <input type="checkbox"/> Non
Remarques : 		

Remarque :

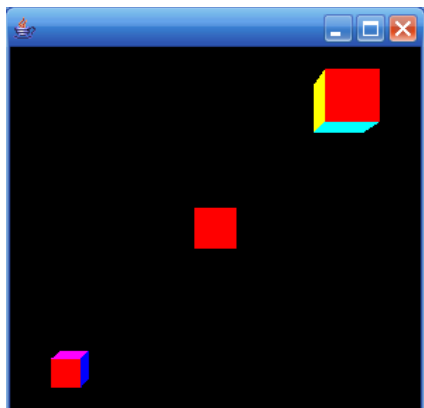
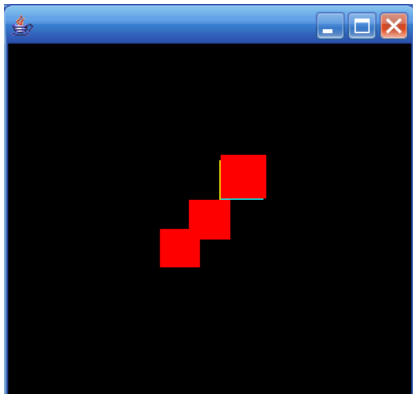
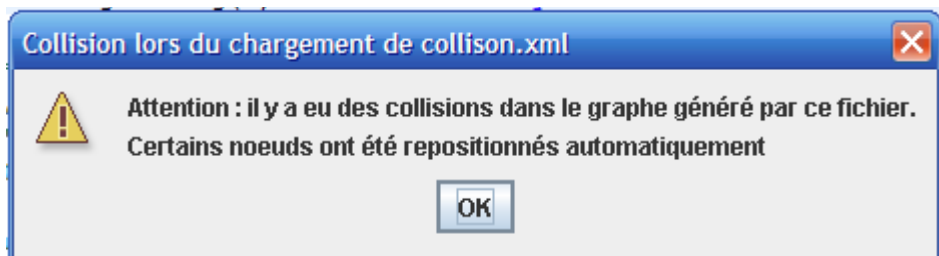
Les deux premiers tests de sauvegarde (nom de fichier mal suffixé ou nom invalide pour le système) pourront être automatisés. Pour cela nous pourront utiliser une classe de tests automatiques en appelant la fonction de sauvegarde, en vérifiant que des erreurs sont soulevées ou pas, et en répercutant les résultats des tests (« test de sauvegarde de XXX.xxx : OK ou KO » par exemple) dans un fichier de synthèse dont le nom comportera la date des tests.

IV.3. Fiche de test fonctionnel : Vue3D > gestion des collisions.

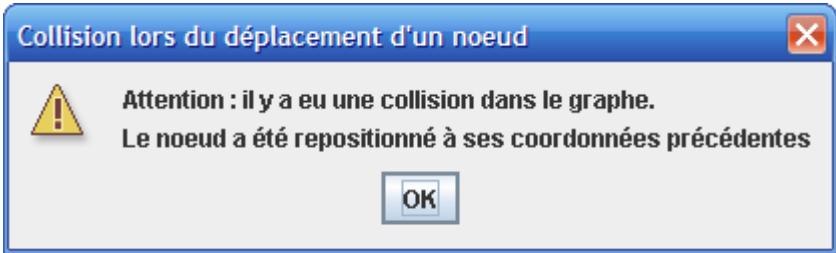
Nom : _____ prenom : _____ date : __/__/____

Prérequis : avoir lancé l'application.

Données d'entrée : chargement d'un fichier contenant un graphe dont le nom est spécifié, puis pour le second cas modification des coordonnées d'un noeud.

	NOM DU FICHIER	TEST RÉUSSI
Chargement d'un fichier avec collisions	collision.xml (annexe 4)	<div><input type="checkbox"/> Oui</div> <div><input type="checkbox"/> Non</div>
<p>Résultat attendu : <i>l'application doit afficher une vue avec le premier noeud aux coordonnées de départ et les autres noeuds doivent être repositionnés automatiquement.</i> <i>La vue doit donc être du type : et pas de ce type :</i></p> <div></div> <div></div> <p><i>De plus un message de signalisation doit apparaître :</i></p> <div></div>		
Remarques :		

Graphe3D : API Java pour la visualisation de graphes mathématiques en 3D

Modification des coordonnées d'un noeud	collision.xml (annexe 4)	
Résultat attendu : <i>après avoir sélectionné dans la vue puis changé les coordonnées du noeud appelé « node 2 » (coordonnées mis à $X=0$, $Y=0$, $Z=0$), l'application doit repositionner le noeud dans le graphe et afficher un message de signalisation comme suit :</i>		<input type="checkbox"/> Oui <input type="checkbox"/> Non
		
Remarques :		

Remarque :

Ces tests de gestion de collision pourront être automatisés. Pour cela nous pourront utiliser une classe de tests automatiques en appelant la fonction de chargement ou la fonction de mise à jour de la vue 3D, en vérifiant que des erreurs sont soulevées ou pas, que les coordonnées des noeuds ont bien été modifiées comme prévu, et en répercutant les résultats des tests (« test de collision au chargement de XXX.xxx : OK ou KO » par exemple) dans un fichier de synthèse dont le nom comportera la date des tests.

IV.4. Annexes

ANNEXE 1 :

nonValideXsd1.xml :

/*

Il manque des attributs pour l'élément 'node0', des attributs pour le sous-élément coordonnates, et l'élément coordonnates est manquant pour 'node1'. Il y a une balise non définie et un attribut non défini pour un élément Node.

Le type du lien n'existe pas (accepte seulement arrow ou bridge).

*/

```
<graphe>
  <node name=node0>
    <coordonates x=1 z=0/>
  </node>
  <baliseinconnue>
  <node name='node1'></node>
  <link type='typeinexistant' name='lien0' class='bridge' />
</graphe>
```

nonValideXsd2.xml :

/*

Le graphe est vide (un graphe doit au minimum comporter un élément Node.

*/

```
<graphe>
</graphe>
```

ANNEXE 2 :

nonValideSem.xml :

```
/*
Il y a des types de noeud ou de lien qui sont non définis ('node1'
et 'lien1'), et des types d'attributs optionnels (enum-attr) qui
ne sont pas définis non plus ('attr1').
Le second lien ('lien2') fait appel à des noeud non encore définis
dans le fichier.
Il existe deux noeuds de nom 'node2' et deux liens de nom 'link1'.
*/

<graphe>
  <link type='arrow' name='lien1' class='typeinconnu' link_start='node1'
link_end='node1' />
  <link type='bridge' name='lien1' link_start='node2' link_end='node2' />
  <node name='node1' class='node'>
    <coordonates x=0 y=0 z=0>
      <enum-attr attr-name='attr1' attr-type='typenonautorise'
attr-value=0 />
    </node>
  <node name='node2' >
    <enum-attr attr-name='attr1' attr-type='int' attr-value=0 />
  </node>
  <node name='node2' >
    <enum-attr attr-name='attr1' attr-type='int' attr-value=0 />
  </node>
  <link type='arrow' name='lien1' class='typeinconnu' link_start='node1'
link_end='node1' />
</graphe>
```

ANNEXE 3 :

valide.xml :

```
<graphe>
  <node name='node1' class='typetest'>
    <coordonates x=0 y=0 z=0 />
    <enum-attr attr-name='entier' type='int' val=0 />
  </node>
  <node name='node2'>
    <coordonates x=1 y=1 z=1 />
  </node>
  <link type='arrow' name='lien1' class='arrow' link_start='node2'
link_end='node1' />
</graphe>
```

ANNEXE 4 :

collision.xml :

```
<graphe>
  <node name='node1'>
    <coordonates x=0 y=0 z=0/>
  </node>
  <node name='node2'>
    <coordonates x=0 y=0 z=0/>
  </node>
  <node name='node3'>
    <coordonates x=0 y=0 z=0/>
  </node>
</graphe>
```

V. Plan qualité

La phase de conception est une étape importante dans un projet. C'est également l'une des plus délicates, car pour un problème donné, il peut y avoir plusieurs solutions certaines meilleurs que d'autres, et si jamais une erreur n'est pas détecté pendant cette phase, celle-ci aura des répercussions importantes dans la suite du projet.

Nous avons choisis d'utiliser la modélisation UML afin de représenter sous une forme standardisée les différents éléments de la conception (Diagramme de classe, Cas d'utilisations,...) afin de faciliter le dialogue et la validation avec le client. Ce choix de représentation permettra aussi une reprise plus aisée du travail déjà effectuer par d'éventuelles futures équipes.

Les réunions hebdomadaires avec le client (M. Bonnel) ont permises de valider régulièrement les éléments de la phase de conception.

V.1 Homogénéité du code

V.1.1 Convention de nommage

Une convention de nommage à été mise en place afin d'améliorer la lecture (et relecture) du code, de faciliter l'échange de code entre les programmeurs. Celle-ci s'appuie sur le document « Java Code Conventions » de SUN. Nous serons amenés à suivre les règles suivantes :

- Les noms des classes et des constantes commenceront obligatoirement par une majuscule.

Exemple : NomDeMaClasse

- Le reste (paquetages, variables, méthodes) commencera toujours par une minuscule.

Exemple : nomDeMaVariable, nomDeMonPaquetage, maMethode()

- Le nom que l'on donnera a une classe, une méthode, un attribut ou une constante devra être parlant pour le programmeur.

Exemple : monNoeud.getNom() ;

- Si plusieurs noms composent le nom principal, chaque nom sera séparé par une majuscule.

Exemple : GNode monNoeud ;

- Le nom d'une constante se composera uniquement de majuscules.

Exemple : static final int ESPACE_ENTRE_LES_NŒUDS = 5 ;

- Pour le nom des méthodes, on choisira un verbe.

Exemple : void modifieNom(String nom) ;

Nous avons décidé que le nom des classes, des méthodes, des variables et des paquetages, ainsi que les commentaires et la Javadoc seront en anglais.

V.1.2 Mise en forme du code

Comme tout le monde travaillera sous Eclipse, on utilisera la mise en forme automatique du code selon des paramètres définis (réglage de l'unité d'indentation, commentaire Javadoc automatique, entêtes de fichiers,...). La définition de cette mise en forme pourra être exportée dans un fichier XML afin de s'assurer que tous les développeurs utilisent la même.

V.2 Qualité du code

Un code de qualité assure la pérennité, la diffusion, la maintenabilité du projet. Il est important de se souvenir que la réussite d'un projet dépend en partie de la qualité du code.

Mais, il est en général difficile d'évaluer la qualité du code produit par soi-même, en particulier lorsque l'on en est l'auteur de ce code. Afin de remédier à ce problème, il existe des outils qui permettent d'effectuer des mesures sur le code produit. Suivant le résultat de l'analyse fournie par ses outils sur ce code, il est possible d'avoir un avis sur la qualité du code.

Les deux principaux outils permettant d'effectuer des mesures de qualité de code sur des fichiers sources Java sont les suivants :

- **JDepend** : Cet outil fournit des statistiques sur la qualité du code produit. Il est sensible à l'extensibilité, la réutilisabilité, et la maintenabilité des sources. Il permet d'évaluer le degré d'abstraction de chaque package, sa stabilité, sa volatilité,... En résumé un paquetage stable, abstrait et faiblement couplé est d'excellente qualité.
- **JavaNCSS** : Il permet lui aussi d'obtenir différentes métriques sur le code produit. Comme le nombre de paquetage, de classe, de méthode, d'instructions sans commentaire, le nombre de commentaires Javadoc par paquetage, classe et méthode.

Ces deux outils peuvent être intégrés dans un fichier « build.xml » qui sera utilisé avec Ant, ce qui permettra d'automatiser les tâches d'analyse et de produire facilement des rapports de qualité. Les résultats des analyses peuvent se faire sous différentes formes (Fichier XML, HTML,...).

V.3 Automatisation des tâches

L'automatisation des tâches permet de standardiser les processus d'un projet et de minimiser les risques d'erreurs en déléguant le maximum de tâches à un processus. Une fois qu'une tâche a été configurée et validée, elle pourra être reproduite une infinité de fois dans les mêmes conditions. Cela est nécessaire pour que tous les tests mais est aussi très utile pour d'autres aspects du développement.

Graphe3D : API Java pour la visualisation de graphes mathématiques en 3D

Nous utiliserons Ant pour l'automatisation des tâches qui est un outil particulièrement très bien adapté pour le développement en Java. Il permet d'automatiser un très grand nombre de tâches, allant de la génération des fichiers binaires, des archives Jar, de la Javadoc, à la création de rapports (JDepend, JavaNCSS), en passant par l'exécution des tests unitaires (JUnit)

Eclipse contient une fonctionnalité permettant de compiler automatiquement un projet durant le codage. Cela permet de détecter en temps réel la présence d'erreurs de codage, de variables non utilisées ou encore les inclusions de bibliothèques nécessaires.

Les tests unitaires seront réalisés à l'aide de JUnit. Il permet de vérifier que pour chaque méthode, les post-conditions sont bien celles qui ont été prévues. Une fois tous les tests codés, il suffit de lancer l'exécution sur une classe ou un projet pour que les tests soient effectués un à un. En fin d'analyse, un rapport est produit indiquant les tests réalisés avec succès et ceux qui ont échoué. Enfin, il est intégré à Eclipse et peut donc être mis en place très simplement et il est facile d'utilisation

Exemple d'un test unitaire avec Junit

```
package graphe3D.elements;

import junit.framework.TestCase;
import java.util.Hashtable;

/**
 * Classe de test pour la classe {@link GNode}.
 */
public class GNodeTest extends TestCase {

    /**
     * Test du premier constructeur de la classe.
     * GNode avec un nom et coordonnées par défaut.
     */
    public final void testGNode1() {

        // Création des éléments à tester :
        GNode node1 = new GNode("node one");

        // Vérification du nom du nœud
        assertNotNull(node1.getName());
        assertEquals(node1.getName(), "node one");

        // Vérification des coordonnées
        assertEquals(node1.getCoordonates()[0], new Float(0));
        assertEquals(node1.getCoordonates()[1], new Float(0));
        assertEquals(node1.getCoordonates()[2], new Float(0));

        // Verification des attributs
        assertNull(node1.getAttributes());
    }
}
```

Graphe3D : API Java pour la visualisation de graphes mathématiques en 3D

```
/**
 * Test du second constructeur de la classe.
 * GNode avec un nom et des coordonnées.
 */
public final void testGNode2() {

    // Création des éléments à tester :
    GNode node1 = new GNode("node two",2,4,5);

    // Vérification du nom du noeud
    assertNotNull(node1.getName());
    assertEquals(node1.getName(), "node two");

    // Vérification des coordonnées
    assertEquals(node1.getCoordonates()[0], new Float(2));
    assertEquals(node1.getCoordonates()[1], new Float(4));
    assertEquals(node1.getCoordonates()[2], new Float(5));

    // Verification des attributs
    assertNull(node1.getAttributes());
}

/**
 * Test de la méthode permettant de changer le nom d'un nœud.
 * On vérifie qu'un changement effectué par l'appel du modificateur
 * est correctement enregistré.
 */
public final void testSetName(){

    // Création des éléments à tester :
    GNode node3 = new GNode("node three");

    // Modification du nom du noeud
    node3.setName("Node 3");

    // Verification du nom du noeud
    assertNotNull(node3.getName());
    assertEquals(node3.getName(), "Node 3");
}

/**
 * Test de la méthode permettant de changer les coordonnées d'un
 * nœud.
 * On vérifie qu'un changement effectué par l'appel du modificateur
 * est correctement enregistré.
 */
public final void testSetCoordonates(){

    // Création des éléments à tester :
    GNode node3 = new GNode("my node");

    // Modification des coordonnées d'un noeud
    node3.setCoordonates(new float[]{3,2,5});

    // Verification des coordonnees
    assertEquals(node3.getCoordonates()[0], new Float(3));
}
```


Graphe3D : API Java pour la visualisation de graphes mathématiques en 3D

```
        assertEquals(node3.getCoordonates()[1], new Float(2));
        assertEquals(node3.getCoordonates()[2], new Float(5));
    }

    /**
     * Test de la méthode permettant d'ajouter des attributs à un nœud.
     * On vérifie qu'un changement effectué par l'appel du modificateur
     * est correctement enregistré.
     */
    public final void testSetAttributes(){

        // Création des éléments à tester :
        GNode node3 = new GNode("my node");
        Hashtable attribut1 = new Hashtable<String, String>();
        attribut1.put("IP", "192.168.1.1");
        attribut1.put("Type", "Ethernet");

        // Modification de la liste des attributs
        node3.setAttributes(attribut1);

        // Vérification des attributs
        assertNotNull(node3.getAttributes());
        assertEquals(node3.getAttributes(), attribut1);
    }
}
```