UNIVERSITÉ DE SHERBROOKE
Faculté de génie
Département de génie électrique et génie informatique

RAPPORT DE PROBLÉMATIQUE

CONCEPTS AVANCÉS EN PROGRAMMATION ORIENTÉE OBJET
S2-H24-GI-APP3, GIF242

Présenté à
Charles-Antoine Brunet, Domingo Palao Munoz, Amauray Daniel Palao Garcia

Présenté par
Équipe numéro Z-S2-APP3-GI-47
Shawn Couture – COUS1912
Francis Gratton - GRAF2102

Sherbrooke - 9 février 2024

TABLE DES MATIÈRES

LISTE DES FIGURES

1. DÉVELOPPEMENT

1.1 DIAGRAMME DE CAS D'UTILISATION DE GRAPHICUS-03

Le diagramme suivant présente un diagramme de cas d'utilisation de l'application Graphicus-03. Ceci approche ou dépasse la limite de lisibilité d'un diagramme et devra être divisé en plus petite partie pour de futur itérations de Graphicus.

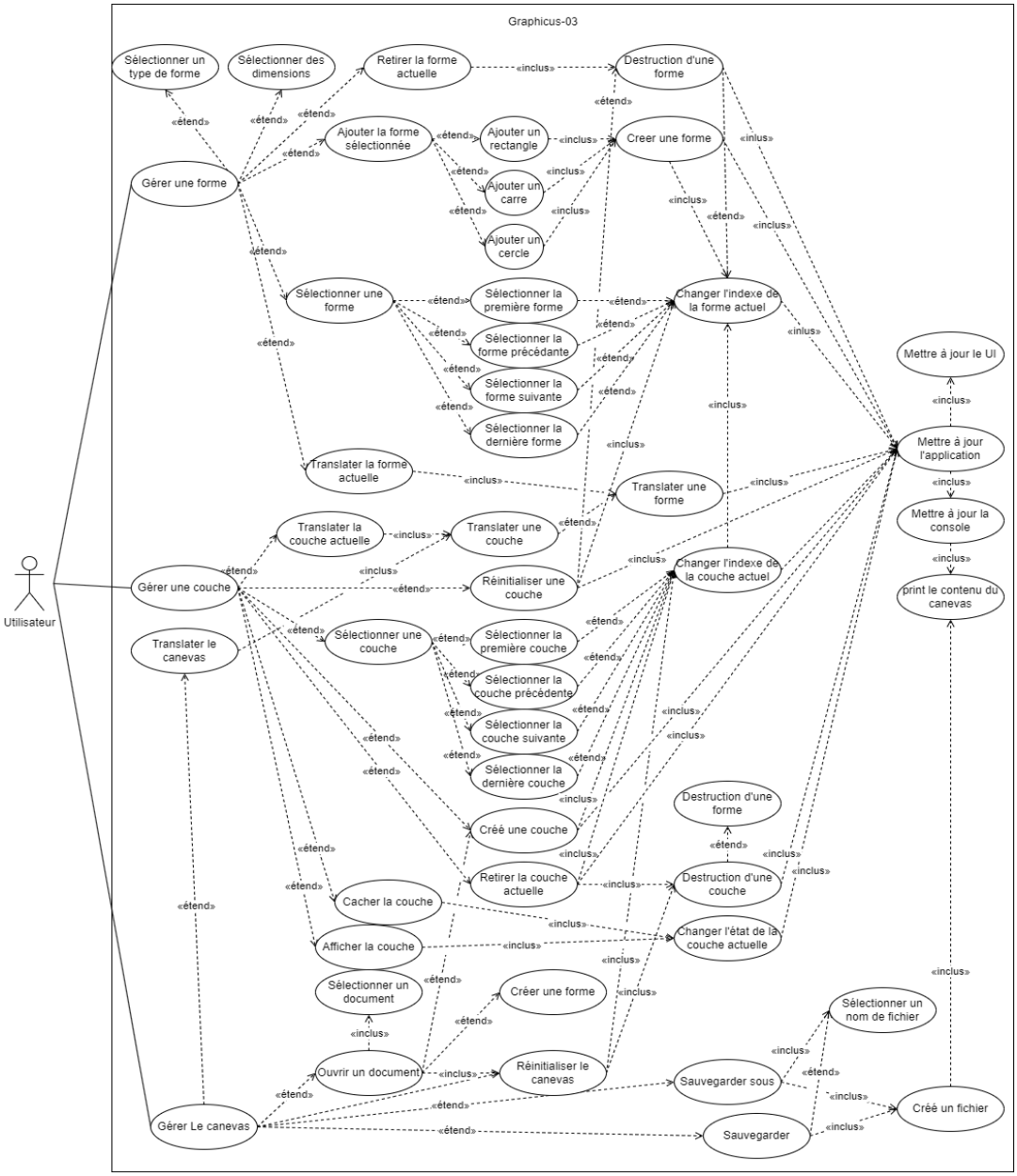

Figure 1: Diagramme de cas d'utilisation

1.2 DIAGRAMME DE CLASSES DE GRAPHICUS-03

Le diagramme suivant présente le diagramme de classes de Graphicus-03. Il s'agit du diagramme de classes de Graphicus-02 lequel a été modifié pour être conforme au contenue de Graphicus-03.

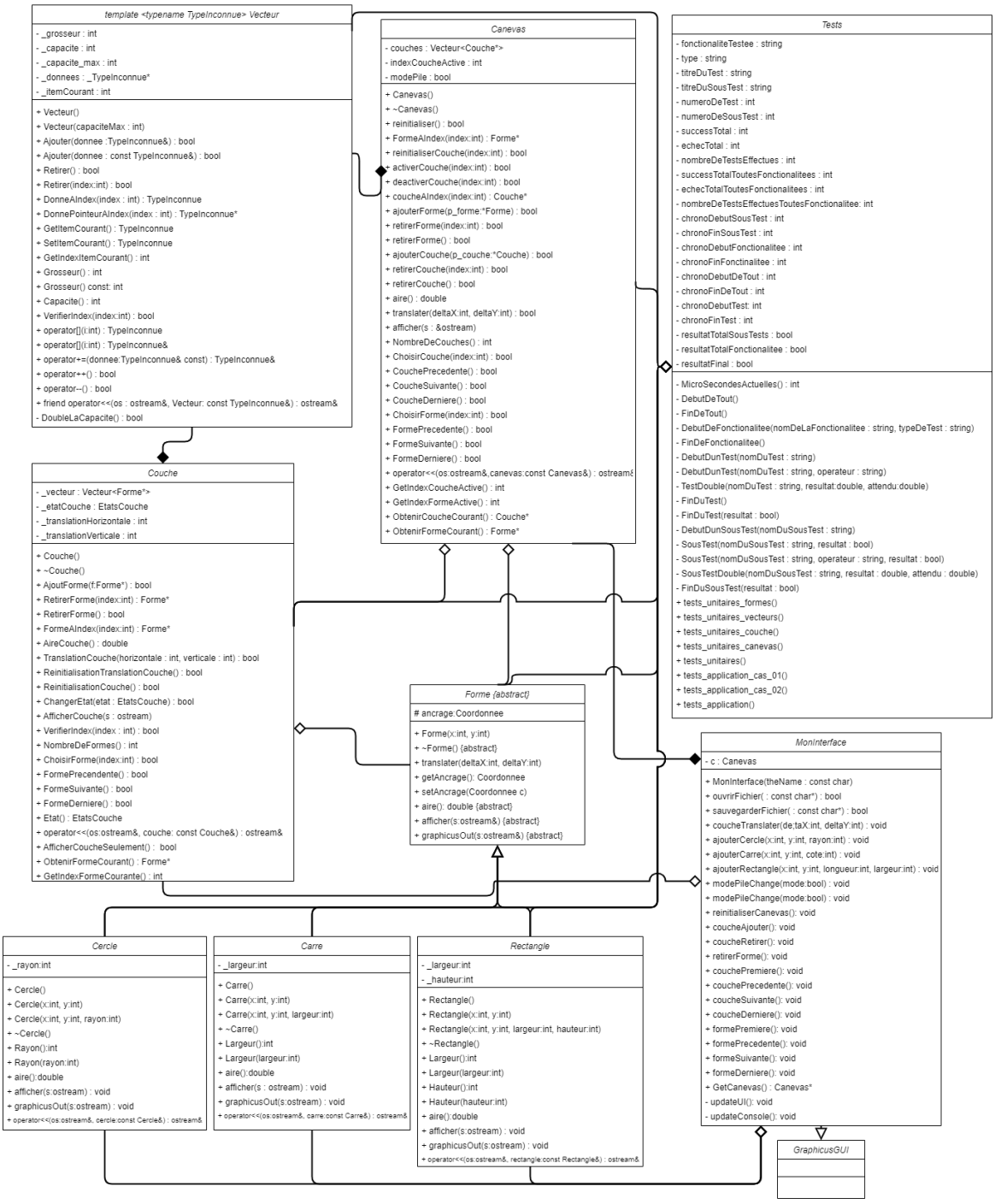

Figure 2: Diagramme de classe

1.3 DIAGRAMME DE SÉQUENCE DE GRAPHICUS-03

La figure suivante présente un exemple de diagramme de séquence pour l'application Graphicus-03 dans lequel l'interaction entre un utilisateur, l'interface graphique ainsi que le Canevas est présenté. Dans cette interaction, l'usager ouvre l'application, change un paramètre de forme, sélectionne un carrée, ajoute le carré, créé une nouvelle couche, sauvegarde le document, puis ouvre un autre document. Il est important de noter que l'interaction peut être beaucoup plus complexe que celle représentée.

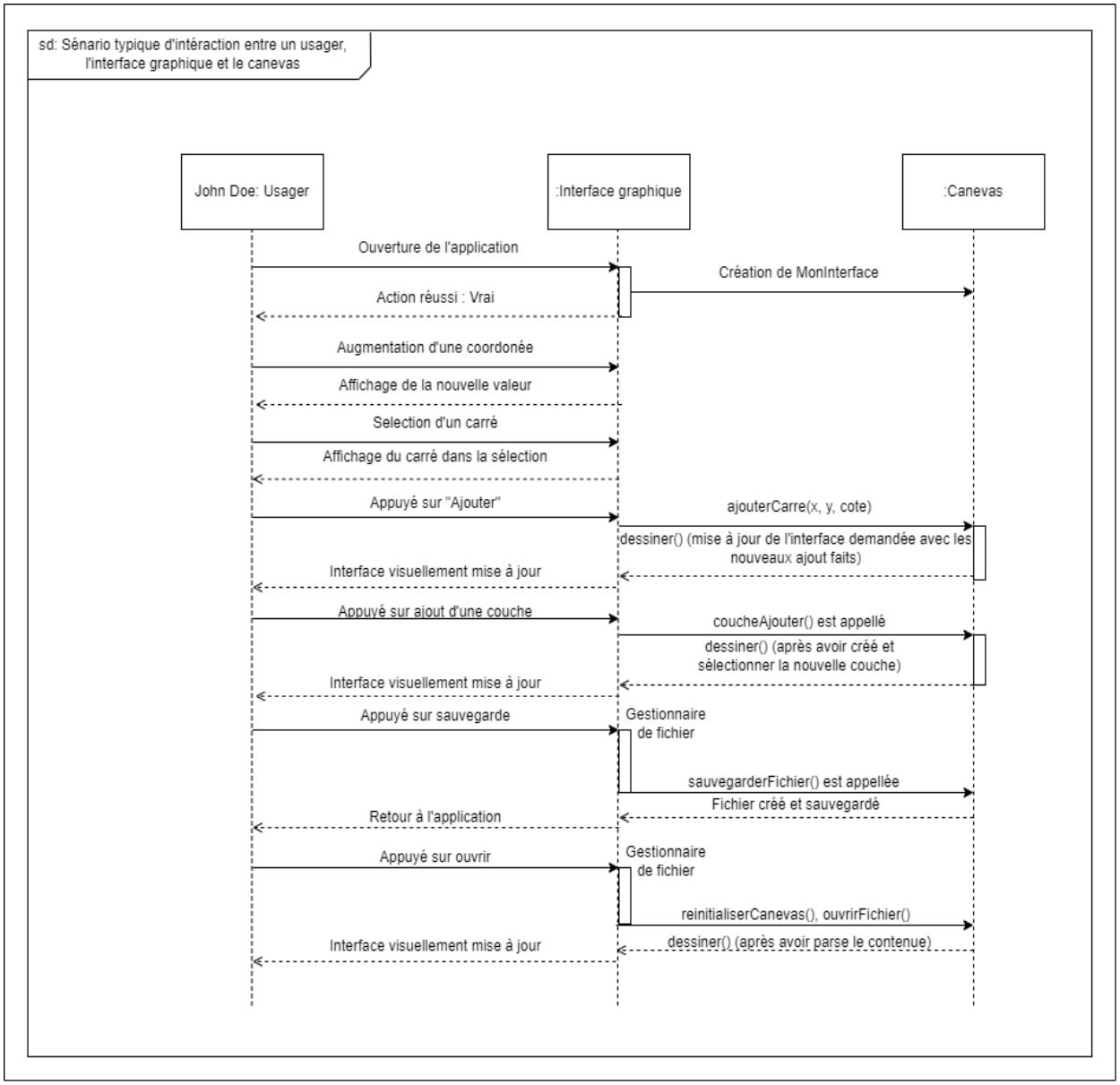

Figure 3: Diagramme de séquence

1.4 TESTS DE GRAPHICUS-03

Les tests unitaires vont très en détails sur les aspects de chaque classes testé et tout tests effectués est groupé en catégories de fonctionnalité, test, et sous-tests. Les résultats sont affichés dans une console ou dans un fichier. Les étapes de tests sont disponibles dans le fichier "tests.h" comme en-tête dans le format DoxyGen ou dans le format automatique de Visual Studio (XML). Étant donné l'ardeur et l'ampleur des tests, il est impossible d'entrée toutes les étapes de tous les tests dans ce document tout en restant dans la cible du 5 à 6 pages. (Les résultats des tests est fournis dans le même dossier que la remise de ce PDF). Les deux tests significatifs sont le test unitaire des vecteurs ainsi que le test d'application cas 2.

Le plan de test de Graphicus-03 consiste, sommairement, à la liste généralisée suivante laquelle a été créé ainsi suite au manque de gabarit et de format demandée:

1. Tests unitaires
    1.1. Tests unitaires des différentes classes de formes
    1.2. Tests unitaires des vecteurs (plusieurs types avec templates)
    1.3. Tests unitaires de la classe de couches
    1.4. Tests unitaires de la classe de canevas
2. Tests d'application
    2.1. Test d'application (premier cas) (APP1)
    2.2. Tests d'application (deuxième cas)

Voici le plan de tests d'application (cas deux) pour Graphicus-03:

1. Vérification des paramètres initiales de l'application
    1.1. Nombre de couches (1)
    1.2. Nombres de formes (0)
    1.3. État de la couche (Initialisée)
2. Tentative de navigation initiales (échec car 1 seule couche et aucune forme)
    2.1. Tentative de changer de couche (toute formes = échec)
    2.2. Vérification de l'index de la couche (0) après chaque tentative
    2.3. Tentative de changer de forme actuelle (toute forme = échec)
    2.4. Vérification de l'index de la forme sélectionnée (0)
3. Manipulation de couches

3.1. Création de 10 couches

3.2. Vérification des états initiales des couches

3.3. Navigation dans les couches + changement d'états de couches

3.4. Navigation pour vérifier l'état des couches après la navigation

3.5. Tentative de retirer la couche actuelle + vérification du nombre de couches

4. Manipulation de formes

4.1. Ajout de formes aléatoire à coordonnées aléatoires dans des couches aléatoire

4.2. Vérification des couches et des formes créées précédemment.

4.3. Tentative de navigation dans les formes et vérification d'indexe

4.4. Changement de couche active + vérification d'index de forme active

4.5. Tentative de retirer une forme sélectionnée

4.6. Tentative de réinitialiser la couche active + vérification

5. Tentative de translation du canevas + vérification
6. Tentative de sauvegarde du document
7. Tentative de réinitialisation du canevas

7.1. Vérification initiale refaites (étape 1)

8. Tentative d'ouverture du document sauvegardée

8.1. Vérification des couches (nombres, états, formes)

Cette liste décrit très sommairement le test qui a été effectués. Le vrai test est situé dans test.cpp dans la méthode « tests_application_cas_02 » et consiste en 94 tests totaux. (Il doit être fait en désactivant l'affichage à la console de Graphicus-03 et en le décommentant dans «testGraphicus03.cpp». Voici le dessin créé par ce test:

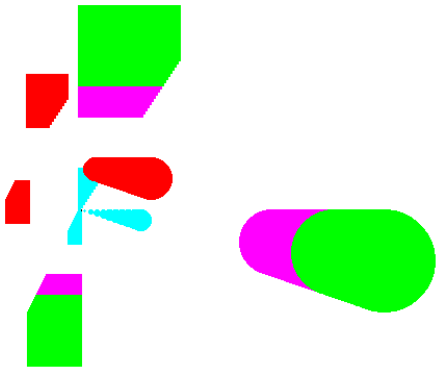

Figure 4: Affichage du tests_application_cas_02

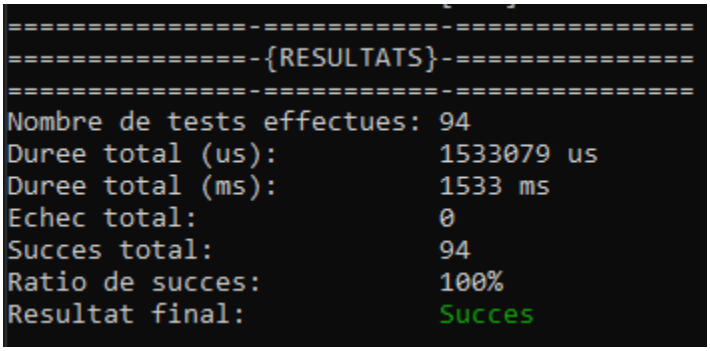

Figure 5: Résultats du tests_application_cas_02 dans la console

Les choses testés dans ce test sont; création de couche, sauvegarde du fichier, ouverture du fichier, navigation dans les couches, navigation dans les formes, ajout de toute les formes, destruction de couche et de forme, test de limites d'index, test de réinitialisation du canevas etc.

Étant donné que ce test, test automatiquement plusieurs fonctionnalités de Graphicus, nous ne jugeons pas la pertinence d'ajouter un deuxième test graphique. Cependant, un fichier de résultat de test est disponible présentant le résultat de 240 tests totaux unitaire qui ont été modifié pour répondre aux besoins de l'app 3. Le deuxième test montré est lui des vecteurs puisqu'il a maintenant des tests spécifiques pour les templates (118 tests).

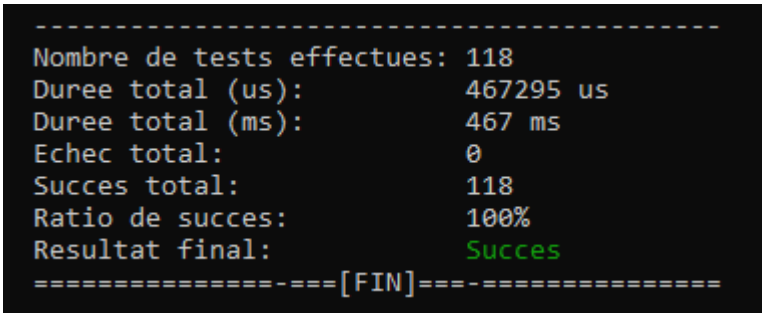

Figure 6: Résultats de tous les tests

Voir le fichier «RESULTAT_TOUS_TESTS.txt» pour les tests d'application plus les tests unitaires.

1.5 RÉPONSE À LA QUESTION DEMANDÉE

Quels concepts de programmation orientée-objet permettent d'associer les actions de l'usager posées dans GraphicusGUI et le code spécifique à votre application?

Le principe d'héritage permet à la classe de notre application d'hérité des fonctionnalités de bases de GraphicusGUI. Cela permet à la classe de notre application de redéfinir les méthodes nécessaires (polymorphisme) définie dans GraphicusGUI afin qu'elles effectuent des actions propres à notre application plutôt que celles de bases. La redéfinition des fonctions permet à la librairie fournis d'appeler ces fonctions sans qu'elle sache que le contenu de ces méthodes sont différentes. De notre côté, cela transforme notre application en application «event-driven». C'est à dire que ces méthodes ne sont pas appelées périodiquement, mais uniquement lorsqu'un événement (comme l'ajout d'une forme) arrive.