# PROJET ITIC

ATRMOUH Boujamaa

MAZAN Kamil

Année 2023-2024 - 24 décembre 2023

## Table des matieres

Table des matières

[PROJET ITIC 0](#_Toc154050116)

[Table des matieres 1](#_Toc154050117)

[Test Fonctionnel 2](#_Toc154050118)

[Valeur (valeur : Object C → Object O) 2](#_Toc154050119)

[Test Structurel 3](#_Toc154050120)

[IsSolved (void → void) 3](#_Toc154050121)

[Rapport et tests 4](#_Toc154050122)

[Pourcentage de Couverture à travers clEmma (Eclipse) 4](#_Toc154050123)

[Pourcentage de Couverture à travers de tests Mutationnels 5](#_Toc154050124)

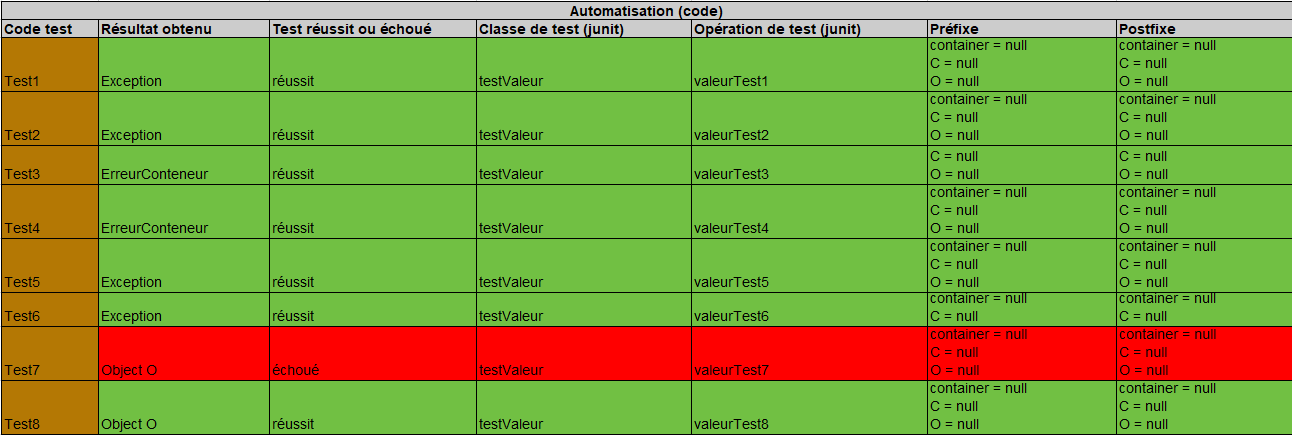
## Test Fonctionnel

### Valeur (valeur : Object C → Object O)

Après une analyse des valeurs d’entrée, et établissement des partitions suivantes :

|  |  |  |
| --- | --- | --- |
| Paramètre d’entrée | Code\_Partition | Condition de partition |
| Object C | Partition1\_1 | Object C absente |
| Partition1\_2 | Object C presente |
| Partition1\_3 | Object C is null |
| Partition1\_4 | Object C not null |
| This.Object | Partition2\_1 | This.object is empty |
| Partition2\_2 | This.object is not empty |

Voici les résultats des tests JUnit, effectuées :

****

Les résultats sont les mêmes pour testEtat4.jar et pour testEtat7.jar.

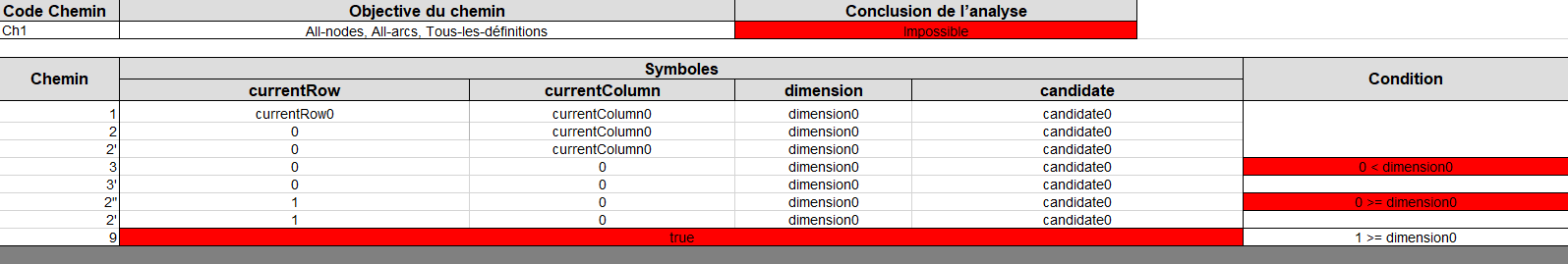
Un test a échoué, car il a été supposé que la clé existait mais était nulle, et que le conteneur n'était pas vide. Selon le cahier des charges, une clé ne peut pas être vide, donc nous attendions une exception. Cependant, le test a retourné une valeur, ce qui indique que la clé peut être vide. Par conséquent, il est nécessaire de corriger la fonction d'ajout ou le constructeur du conteneur afin de vérifier si la clé que l'on tente d'ajouter est vide. Dans le cas où la clé est vide, l'ajout dans le conteneur ne devrait pas être autorisé.

## Test Structurel

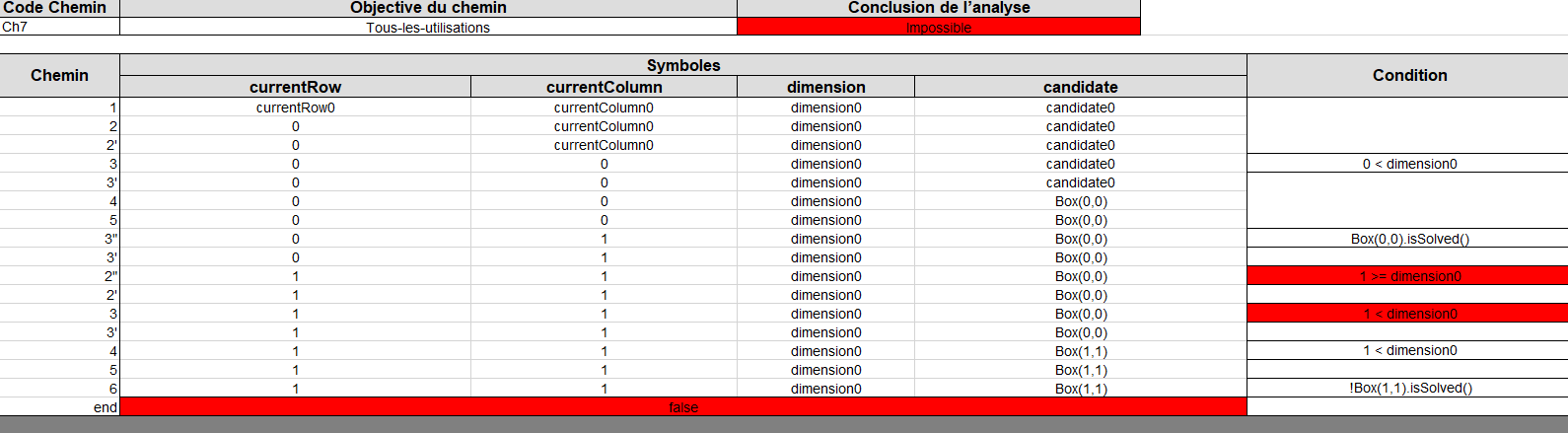
### IsSolved (void → void)

Après une analyse en profondeur sur différents crités, comportant All-nodes ou encore Tous-les-utilisations, nous avons pu identifier 9 chemins, cependant après analyse symbolique il se trouve que certains d’entre eux ne sont pas exécutables :

All-nodes, all-arcs, tous-les-définitions (Chemin 1) :



Tous-les-utilisations (Chemin 7) :

****

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Automatisation (code) | | | | | | |
| Code test | **Résultat obtenu** | **Test réussit ou échoué** | **Classe de test** | **Opération de test** | **Préfixe** | **Postfixe** |
| Test1 | false | réussit | testIsSolved | isSolvedTest1 | this.Puzzle = null | this.Puzzle = null |
| Test2 | true | réussit | testIsSolved | isSolvedTest2 | this.Puzzle = null | this.Puzzle = null |
| Test3 | false | réussit | testIsSolved | isSolvedTest3 | this.Puzzle = null | this.Puzzle = null |
| Test4 | true | réussit | testIsSolved | isSolvedTest4 | this.Puzzle = null | this.Puzzle = null |
| Test5 | false | réussit | testIsSolved | isSolvedTest5 | this.Puzzle = null | this.Puzzle = null |
| Test6 | false | réussit | testIsSolved | isSolvedTest6 | this.Puzzle = null | this.Puzzle = null |
| Test7 | false | réussit | testIsSolved | isSolvedTest7 | this.Puzzle = null | this.Puzzle = null |

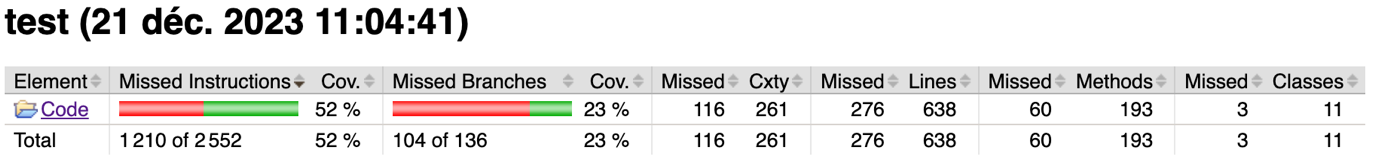
Ici, nous voyons bien qu’il y a contradiction dans les conditions, la dimension ne peux pas être à la foi inférieure à 0 et supérieur ou égale à 0. Ainsi, les chemins ne sont jamais parcourus.

L’analyse symbolique nous permet d’établir 7 tests distincts :

## Rapport et tests

### Pourcentage de Couverture à travers clEmma (Eclipse)

Ici, nous avons décidé d’utiliser testEtat7.jar. Car après l’analyse fonctionnel, nous avons constaté que cette version fonctionnait bien mieux.

****

**Une image contenant texte, capture d’écran, nombre, Police

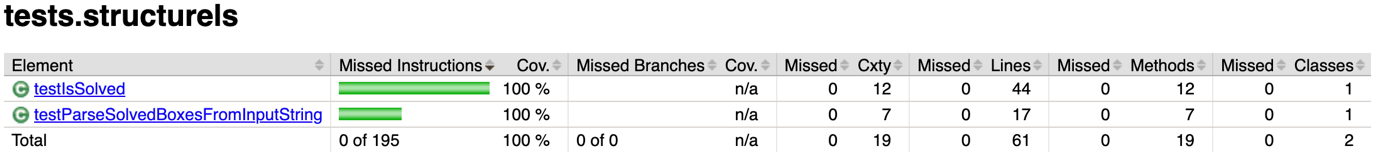
Description générée automatiquement**

**Une image contenant texte, capture d’écran

Description générée automatiquement**

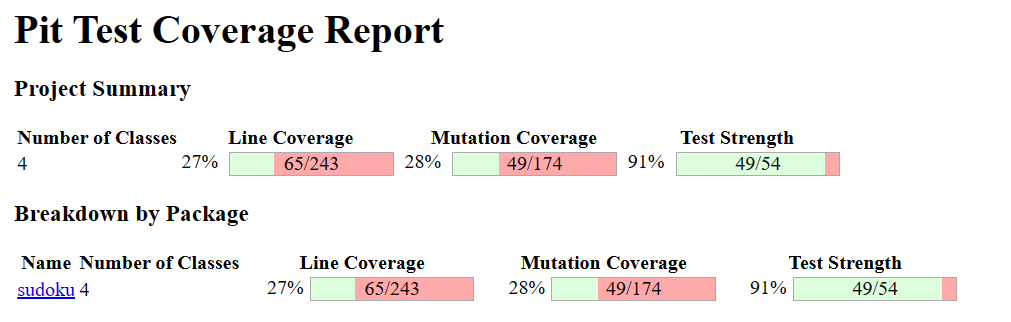
**Une image contenant texte, capture d’écran, Police, nombre

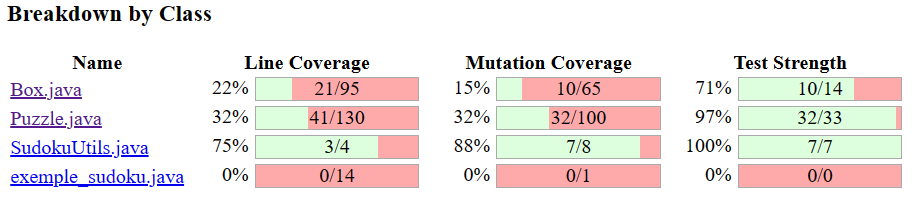
Description générée automatiquement**

****

Nous constatons que nos tests fonctionnels sont couvert à 78% par l’exécuteur. Et les tests structurels à 100%.

### Pourcentage de Couverture à travers de tests Mutationnels

****

****

Line coverage : Ici on peut voir que nos tests gèrent 65 lignes sur 243 lignes de codes ce qui représente 27% sachant qu’on teste uniquement 2 méthodes. Mais vu que ces dernières en appel d’autres, ce résultat est tout à fait plausible.

Mutation Coverage : Nos tests qui gèrent uniquement 65 lignes de codes gèrent 49 mutations sur 174 possibles trouver par le PIT sur l’ensemble du code.

Test Strength : Nos tests ont permis de gérer 49 mutations sur 54 mutations trouver par le PIT, ce qui correspond à 91% de mutations.