

[Adresse](https://www.google.fr/search?biw=1280&bih=563&q=polytech%27tours+adresse&stick=H4sIAAAAAAAAAOPgE-LWT9c3NDKszLDMNtKSzU620s_JT04syczPgzOsElNSilKLiwEgIq2zLgAAAA&sa=X&sqi=2&ved=0ahUKEwjh2eSaj5TRAhVEOhoKHWkAD8sQ6BMIhAEwEg)**:**64 Avenue Jean Portalis, 37200 Tours

[Téléphone](https://www.google.fr/search?biw=1280&bih=563&q=polytech%27tours+t%C3%A9l%C3%A9phone&stick=H4sIAAAAAAAAAOPgE-LWT9c3NDKszLDMNtLSz0620k_Oz8lJTS7JzM_Tz87LL89JTUlPjS9IzEvNKdbPSCyOL8jIz0u1ApMAkj5NlUAAAAA&sa=X&sqi=2&ved=0ahUKEwjh2eSaj5TRAhVEOhoKHWkAD8sQ6BMIhwEwEw)**:**02 47 36 14 14

**Cahier de test**

Etudiants :

Peng BI

[peng.bi@etu.univ-tours.fr](file:///C:\Users\Clotaire\Desktop\peng.bi@etu.univ-tours.fr)

Encadrant :

Nicolas MONMARCHE

<nicolas.monmarche@univ-tours.fr>

Année : 2017-2018

Semestre : 10

Sommaire

[1. Introduction 3](#_Toc509981553)

[1.1 Avant-propos 3](#_Toc509981554)

[1.2 Objectif 3](#_Toc509981555)

[1.3 Pré-requis 3](#_Toc509981556)

[1.3.1 Typologie de test 3](#_Toc509981557)

[1.3.2 Eléments à tester 3](#_Toc509981558)

[1.3.3 Environnement de test 4](#_Toc509981559)

[2. Tests unitaires 4](#_Toc509981560)

[2.1 Tests unitaires de la classe « FileController » 6](#_Toc509981561)

[2.2 Tests unitaires de la classe « ProbabilityController » 8](#_Toc509981562)

[2.3 Tests unitaires de la classe « InstanceController » 8](#_Toc509981563)

[2.4 Tests unitaires de la classe « AlgorithmController » 9](#_Toc509981564)

[3. Tests fonctionnels 12](#_Toc509981565)

[3.1 Détermination du nombre d’itération 13](#_Toc509981566)

[3.2 Evaluation de la solution 14](#_Toc509981567)

[4. Tests de performance 15](#_Toc509981568)

[4.1 Test du temps d’exécution 15](#_Toc509981569)

[4.2 Test d 16](#_Toc509981570)

# Introduction

## Avant-propos

Ce document est rédigé par Peng BI, étudiant en Informatique à Polytech Tours pour le projet R&D. Ce document sert à enregistrer et expliquer les tests mis en pratique.

## Objectif

Le test logiciel est pour assurer la justesse, l’intégrité, la sécurité et la performance de logiciel. Pour ce projet, les tests servent à vérifier si l’algorithme proposé est correct et pratique, et vérifier si les modules de programme réalisé peuvent bien fonctionner. Par exemple, on peut savoir si le module de sélection de care peut bien sélectionner un centre d’accueil pour un bâtiment par le test. En plus, on peut constater si la sortie réelle correspond bien à la sortie attendue.

## Pré-requis

### Typologie de test

Les tests pour ce projet R&D contiennent les 3 types décrits dans le tableau 1.1.

|  |  |  |
| --- | --- | --- |
| ID | Type de test | Description |
| 1 | Test unitaire | Le test unitaire est pour vérifier l’entrée et la sortie de chaque méthode dans les classes en utilisant des cas de test. |
| 2 | Test fonctionnel | Le test fonctionnel est pour comparer le résultat obtenu avec le résultat attendu. |
| 3 | Test de performance | Le test de performance est pour constater le temps d’exécution et l’occupation de mémoire de certaines méthodes afin d’améliorer la performance de programme et d’algorithme. |

Tableau 1.1 Les Types de test mis en pratique

### Eléments à tester

Le tableau 1.2 montre les éléments à tester. Les éléments sont séparés en deux types : les classe de contrôleur et les fichiers obtenus.

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Type d’élément | Nom d’élément | Description |
| 1 | <Classe> | FileController | Le contrôleur est le contrôleur de fichier, y compris la lecture et l'écriture |
| 2 | <Classe> | InstanceController | Le contrôleur d’instance de projet, y compris les bâtiments, les centres d’accueil, les distances entre les bâtiments et les centres d’accueil, les fourmis et les deux types de phéromone. |
| 3 | <Classe> | ProbabilityController | Le contrôler de la probabilité de bâtiments ou de cares lorsque les fourmis déplacent. |
| 4 | <Classe> | AlgorithmController | Le contrôleur d'algorithme, qui réalise à chercher la meilleure solution d'affection. |
| 5 | <Fichier> | solution.txt | Le fichier de sortie qui stocke la meilleure solution |
| 6 | <Fichier> | quality.txt | Le fichier de sortie qui stocke les qualités des meilleures solutions et les qualités moyenne des solutions, la distance totale, le nombre total de sans-abris hébergés et le nombre total de bâtiments affectés de chaque itération. |

Tableau 1.2 Les élément à tester

### Environnement de test

L’environnement pour faire le test est présenté dans le tableau 1.3. Les types d’environnement sont séparés en 3 parties : la machine, l’IDE et la librairie utilisé pour les tests.

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Type d’environnement | Nom d’environnement | Utilisation |
| 1 | <Machine> | PC window 10, i7, 8G | C’est l’ordinateur où on fait les test |
| 2 | <IDE> | PyCharm | C’est la plateforme où on fait la programmation et les tests |
| 3 | <Librairie> | unittest | Cette librairie est pour faire le test unitaire |
| 4 | <Librairie> | matplotlib.pyplot | Cette librairie est pour faire le test fonctionnel, on l’utilise pour dessiner les figures |
| 5 | <Librairie> | line\_profiler | Cette librairie est pour faire le test de performance, qui teste le temps pris par la méthode ligne par ligne |
| 6 | <Libraire> | memory\_profiler | Cette librairie est pour faire le test de performance, qui teste le mémoire utilisé par la méthode ligne par ligne |

Tableau 1.3 L’environnement de test

# Tests unitaires

Le test unitaire est pour tester chaque fonction ou chaque méthode de la classe. On donnant des données d’entrée à la méthode à tester, et on a un résultat attendu d’avance. On constate si le résultat de sortie de méthode après l’exécution avec les données d’entrée est bien correspondant à notre résultat attendu. Si les deux résultats se correspondent bien, ça signifie que la méthode est correcte. Sinon, on doit traiter les défauts de la méthode.

La figure 2.1 montre un petit exemple de test unitaire pour ce projet.

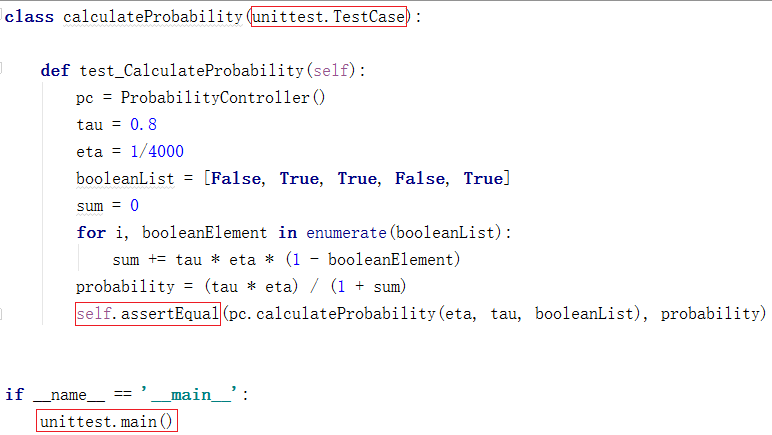


Figure 2.1 L’exemple de test unitaire avec la librairie « unittest »

La langage Python fournit une librairie qui s’appelle « unittest » pour facilement faire le test unitaire. Il suffit que la classe de test hérite la classe « unittest.TestCase » pour faire le test unitaire avec des cas de test. Cette libraire nous fournit beaucoup de méthodes d’assertion, par exemples, la méthode « assertTrue() » est pour vérifier si l’expression est vrai, la méthode « assertEqual() » est pour vérifier si les deux variables sont égales, la méthode « assertListEqual() » est pour vérifier si les deux listes sont pareilles, etc. Si la réponse d’assertion est positive, le test réussit. Sinon, il générera l’exception. Enfin, il suffit une phrase « unittest.main() » pour démarrer le test. Une fois cette phrase exécute, toutes les méthodes dans cette classe seront testées.

La Figure 2.2 montre le résultat de test unitaire avec succès.

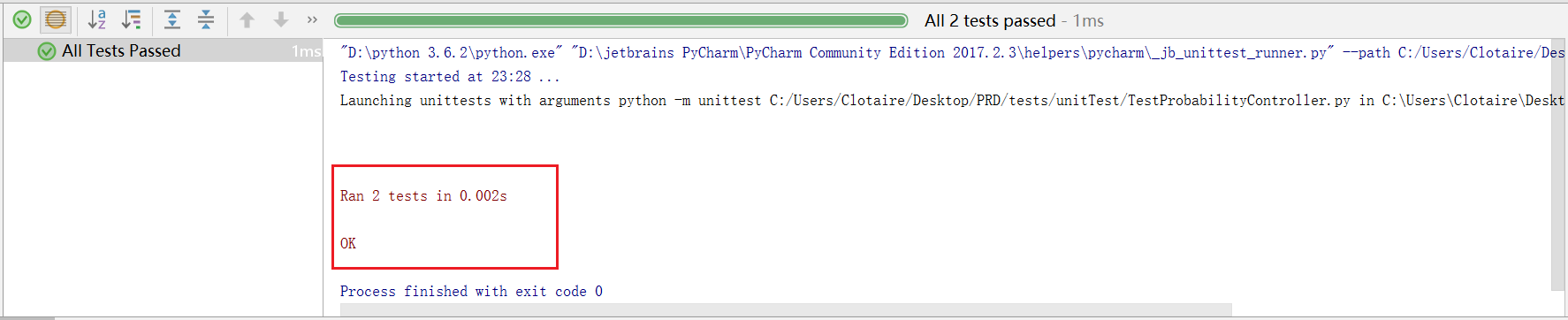


Figure 2.2 Le résultat de test unitaire avec succès

Le figure 2.2 signifie que l’on a testé deux méthodes dans cette classe, et tous les tests réussit. Si le test échoue, le résultat est comme figure 2.3.



Figure 2.3 Le résultat de test unitaire échoué

Dans la figure 2.3, il affiche que deux methodes sont testées, mais un test génère l’exception car le résultat de sortie n’est pas égale au résultat prévu.

## Tests unitaires de la classe « FileController »

Les tests unitaires des méthodes de la classe « FileController » sont listés dans le tableau 2.1.

|  |  |  |
| --- | --- | --- |
| **Test unitaire de la classe « FileController »** | | |
| **ID** | **Article** | **Contenu** |
| 1 | Nom de méthode à tester | readBuildingFile() |
| Description | Cette méthode est pour lire le fichier de bâtiment. |
| Données d’entrée | Le nom fichier texte de bâtiment avec 51 lignes (la première ligne est le titre) |
| Sortie attendue | La taille de la liste qui enregistre chaque ligne du fichier de bâtiment (sauf la première liste) est 50 |
| Résultat réel | 50 |
| 2 | Nom de méthode à tester | readCareFile() |
| Description | Cette méthode est pour lire le fichier de care (centre d’accueil). |
| Données d’entrée | Le nom fichier texte de care avec 11 lignes (la première ligne est le titre) |
| Sortie attendue | La taille de la liste qui enregistre chaque ligne du fichier de care (sauf la première liste) est 10 |
| Résultat réel | 10 |
| 3 | Nom de méthode à tester | readDistanceFile() |
| Description | Cette méthode est pour lire le fichier de distance. |
| Données d’entrée | Le nom fichier texte de distance avec 501 lignes (la première ligne est le titre) |
| Sortie attendue | La taille de la liste qui enregistre chaque ligne du fichier de distance (sauf la première liste) est 500 |
| Résultat réel | 500 |
| 4 | Nom de méthode à tester | writeSolutionFile() |
| Description | Cette méthode est pour écrire la meilleure solution dans le fichier de solution. |
| Données d’entrée | 1. Le nom du fichier de solution 2. La meilleure solution : [4,5,6,-1,3] 3. L’instance : les ids de bâtiments sont : [0,1,2,3,4], les ids de cares sont : [0,1,2,3,4,5,6] |
| Sortie attendue | Un fichier texte qui stocke la meilleure solution, ce fichier a 6 lignes (la première ligne est le titre) et 2 colonne, le nombre de ligne est égal à la taille de meilleure solution+1.  La première colonne est l’id de bâtiment, la deuxième colonne est l’id de care |
| Résultat réel | OK |
| 5 | Nom de méthode à tester | writeQualityFile() |
| Description | Cette méthode est pour écrire les qualités des meilleures solutions de chaque itération et les qualités moyennes des solutions de chaque itération, la distance totale et le nombre de sans-abris hébergés ainsi que le nombre de bâtiments affectés de chaque itération dans le fichier de qualité |
| Données d’entrée | 1. La liste de qualités des meilleures solutions de chaque itération : [0.1477832512315271,0.09865470852017937] 2. La liste de qualités moyennes de solutions de chaque itération : [0.07858062847465425, 0.05374583753472586] 3. La liste de distance totale de la meilleure solution de chaque itération : [100,200] 4. La liste de sans-abris totaux hébergés de la meilleure solution de chaque itération : [25,30] 5. La liste de nombre de bâtiments affectés de la meilleure solution de chaque itération : [4,5] |
| Sortie attendue | Un fichier texte qui stocke les données d’entrée, ce fichier a 3 lignes (la première ligne est le titre) et 6 colonne, le nombre de ligne est égal à la taille de la liste de qualités des meilleures solutions+1, chaque ligne est pour une itération.  La première colonne est l’id d’itération, a deuxième colonne est la qualité de meilleure solution, la troisième colonne est la qualité moyenne des solutions, la quatrième colonne est la distance totale de la meilleure solution, la cinquième colonne est le nombre de sans-abris hébergés de la meilleure solution, la dernière colonne est le nombre de bâtiments affectés de la meilleure solution. |
| Résultat réel | OK |

Tableau 2.1 Les tests unitaires des méthodes de la classe « FileController »

## Tests unitaires de la classe « ProbabilityController »

Les tests unitaires des méthodes de la classe « ProbabilityController » sont listés dans le tableau 2.2.

|  |  |  |
| --- | --- | --- |
| **Test unitaire de la classe « ProbabilityController »** | | |
| **ID** | **Article** | **Contenu** |
| 1 | Nom de méthode à tester | calculateProbability() |
| Description | Cette méthode est pour calculer la probabilité de déplacement. |
| Données d’entrée | 1. La désirabilité de déplacement : 1/4000 2. La quantité de phéromones existant : 0.8 3. Une liste booléenne : [False, True, True, False, True] |
| Sortie attendue | 0.00019992003198720514 |
| Résultat réel | 0.00019992003198720514 |
| 2 | Nom de méthode à tester | generateProbability() |
| Description | Cette méthode est pour sélectionner un événement qui correspond à une probabilité au hasard |
| Données d’entrée | 1. La séquence qui contient les identifiants : [0,1,2,3] 2. La liste de probabilité de déplacement : [0.15,0.26,0.58,0.73] |
| Sortie attendue | Après 100 itérations, les occurrences de 0, 1, 2 et 3 ne sont pas tous 100. (Parce qu’elle utilise le nombre aléatoire, on ne peut pas prévoir un résultat concret). |
| Résultat réel | Vrai |

Tableau 2.2 Les tests unitaires des méthodes de la classe « ProbabilityController »

## Tests unitaires de la classe « InstanceController »

Les tests unitaires des méthodes de la classe « InstanceController » sont listés dans le tableau 2.3.

|  |  |  |
| --- | --- | --- |
| **Test unitaire de la classe «InstanceController »** | | |
| **ID** | **Article** | **Contenu** |
| 1 | Nom de méthode à tester | constructInstance() |
| Description | Cette classe est pour construire l'instance de projet |
| Données d’entrée | 1. Le nombre de fourmis : 10 2. Le nom fichier texte de bâtiment avec 51 lignes (la première ligne est le titre) 3. Le nom fichier texte de care avec 11 lignes (la première ligne est le titre) 4. Le nom fichier texte de distance avec 501 lignes (la première ligne est le titre) |
| Sortie attendue | Une instance est construite, dont la taille de son attribut « buildingList » est 50 et la taille de son attribut « careList » est 10 et la taille de son attribut « distanceMatrix» est 50 et le nombre de colonne de « distanceMatrix» est 10 et la taille de son attribut « antList » est 10 |
| Résultat réel | Vrai |
| 2 | Nom de méthode à tester | solveProblem() |
| Description | Cette méthode fournit le service de résoudre le problème, elle n’a pas fonction réelle. |
| Données d’entrée | 1. La fois d'itération : 200 2. Le rayon d'attraction initial pour les cares : 3000m 3. Le nom du fichier texte sortie de solution 4. Le nom du fichier texte sortie de qualité |
| Sortie attendue | Elle peut appeler la méthode « run() » de la classe « AlgorithmController » et les méthodes «writeSolutionFile() » et « writeQualityFile() » avec succès, et afficher des messages. |
| Résultat réel | OK |

Tableau 2.3 Les tests unitaires des méthodes de la classe « InstanceController »

## Tests unitaires de la classe « AlgorithmController »

Les tests unitaires des méthodes de la classe « AlgorithmController » sont listés dans le tableau 2.4.

Pour faire les tests unitaires pour cette classe, on crée une instance globale pour toutes les méthodes de test d’avance : les valeurs de ses attributs sont :

* instance.distanceMatrix = [[10, 20, 30, 15, 5, 20, 30],

[30, 20, 15, 20, 15, 10, 5],

[20, 30, 10, 25, 5, 20, 10],

[40, 10, 5, 30, 25, 15, 10],

[30, 20, 30, 5, 10, 15, 20]]

* instance.buildingList.idBuilding = [0,1,2,3,4]

instance.buildingList.population = [9, 5, 16, 27, 30]

* instance.careList.idCare = [0,1,2,3,4,5,6]

instance.careList.capacity = [8, 15, 6, 34, 13, 10, 18]

* instance.pheromoneNode.eta = instance.buildingList.population

instance.pheromoneNode.rho = [0.000001]\*5

instance.pheromoneNode.tau = 0.8

* instance.pheromoneEdge.eta = 1/instance.distanceMatrix

instance.pheromoneEdge.rho = [[0.1] \* 7] \*5

instance.pheromoneEdge.tau = 0.9

* instance.antList = [AntModel(), AntModel(), AntModel(), AntModel(), AntModel()]

instance.antList.solution.solutionArray = [-1,-1,-1,-1,-1]

|  |  |  |
| --- | --- | --- |
| **Test unitaire de la classe « AlgorithmController »** | | |
| **ID** | **Article** | **Contenu** |
| 1 | Nom de méthode à tester | merge\_sort() |
| Description | Cette méthode est pour trier une liste d'indice de bâtiments en référant la matrice de distance avec la trie par fusion |
| Données d’entrée | 1. La liste de distance de care à référer (une colonne de la matrice de distance) : [2,56,3,22] 2. La liste d'indice de bâtiment à trier : [0,1,2,3] |
| Sortie attendue | [0,2,3,1] |
| Résultat réel | [0,2,3,1] |
| 2 | Nom de méthode à tester | sortBuildingIndexForEachCareInDistanceMatrix() |
| Description | Cette méthode est pour trier les indices de bâtiments pour chaque care en référant la matrice de distance, elle dépend de la méthode « merge\_sort() » |
| Données d’entrée | 1. La matrice de distance à référer: [[2,4,5,3,1],[56,29,3,43,11], [3,54,67,13,4], [22,51,78,32,45]] 2. La matrice d’indice à trier : [[0,1,2,3],[0,1,2,3],[0,1,2,3],[0,1,2,3],[0,1,2,3]] |
| Sortie attendue | [[0,2,3,1],[0,1,3,2],[1,0,2,3],[0,2,3,1],[0,2,1,3]] |
| Résultat réel | [[0,2,3,1],[0,1,3,2],[1,0,2,3],[0,2,3,1],[0,2,1,3]] |
| 3 | Nom de méthode à tester | objectiveFunctionF() |
| Description | Cette méthode est pour réaliser la fonction objective f(x) |
| Données d’entrée | 1. Une solution : [4,6,4,2,3] 2. L’instance dessus |
| Sortie attendue | 435 |
| Résultat réel | 435 |
| 4 | Nom de méthode à tester | objectiveFunctionG() |
| Description | Cette méthode est pour réaliser la fonction objective g(x) |
| Données d’entrée | 1. Une solution : [4,6,4,2,3] 2. L’instance dessus |
| Sortie attendue | 87 |
| Résultat réel | 87 |
| 5 | Nom de méthode à tester | objectiveFunctionH() |
| Description | Cette méthode est pour réaliser la fonction objective h(x) |
| Données d’entrée | 1. Une solution : [4,6,4,2,3] 2. L’instance dessus |
| Sortie attendue | 5 |
| Résultat réel | 5 |
| 6 | Nom de méthode à tester | calculateAverageSolutionQualityForEachIteration() |
| Description | Cette méthode est pour calculer la qualité moyenne des solutions générées par chaque fourmi dans une itération |
| Données d’entrée | 1. la liste de qualités de chaque solution d'une itération : [0.5, 0.4, 0.6,0.3, 0.2] 2. L’instance dessus |
| Sortie attendue | 0.4 |
| Résultat réel | 0.4 |
| 7 | Nom de méthode à tester | chooseCare() |
| Description | Cette méthode est pour sélectionner les cares à remplir |
| Données d’entrée | 1. La solution initiale : [-1, -1, -1, -1, -1] 2. La liste qui marque si le care est plein : [False, False, False, False, False, False, False] 3. L’instance dessus 4. La liste copiée de bâtiments à affecter : buildingToAllocateList = copy.deepcopy(instance.buildingList) 5. La liste copiée de care à remplir : careToFillList = copy.deepcopy(instance.careList) |
| Sortie attendue | Après 100 itération, pour le bâtiment dont l’indice est 4, il est affecté au care 3 à l’équilibre |
| Résultat réel | Vrai |
| 8 | Nom de méthode à tester | allocateBuilding() |
| Description | Cette méthode est pour sélectionner les bâtiments à affecter |
| Données d’entrée | 1. L’instance dessus 2. La matrice d’indices de bâtiments triées : générée par la méthode « sortBuildingIndexForEachCareInDistanceMatrix() » 3. La liste de solutions initiale pour une itération : [] 4. La liste de qualités de solution initiale pour une itération : [] |
| Sortie attendue | Après l’exécution d’une fois, la taille de liste de qualité est 1 et la valeur n’est pas 0, et la solution n’est plus [-1,-1,-1,-1,-1] |
| Résultat réel | Vrai |
| 9 | Nom de méthode à tester | run() |
| Description | Cette méthode est l'entrèe de l'algorithme, et synthétise les solutions génèrèe par chaque fourmi dans chaque itération, et obtenir la meilleure solution |
| Données d’entrée | 1. L’instance dessus 2. Le nombre d’itérations : 300 |
| Sortie attendue | Après 300 itération, afficher la meilleure solution, la population totale hébergé, la distance totale, la qualité de solutions (parce qu’il y a beaucoup de contraintes pour déterminer une solution, c’est difficile de prévoir la meilleure solution d’avance) |
| Résultat réel | [4, 6, -1, -1, 3], 44, 15, 2.75 |
| 10 | Nom de méthode à tester | calculateDistanceTotalOfOneSolution() |
| Description | Cette méthode est pour calculer la distance totale d'une solution |
| Données d’entrée | 1. L’instance dessus 2. Une solution : [4,6,-1,-1,3] |
| Sortie attendue | 15 |
| Résultat réel | 15 |
| 11 | Nom de méthode à tester | calculatePopulationAllocatedOfOneSolution() |
| Description | Cette méthode est pour calculer le nombre de sans-abris hébergés d'une solution |
| Données d’entrée | 1. L’instance dessus 2. Une solution : [4,6,-1,-1,3] |
| Sortie attendue | 44 |
| Résultat réel | 44 |
| 12 | Nom de méthode à tester | calculateBuildingAllocatedOfOneSolution() |
| Description | Cette méthode est pour calculer le nombre de bâtiments affectés d'une solution. |
| Données d’entrée | 1. L’instance dessus 2. Une solution : [4,6,-1,-1,3] |
| Sortie attendue | 3 |
| Résultat réel | 3 |

Tableau 2.4 Les tests unitaires des méthodes de la classe « AlgorithmController »

# Tests fonctionnels

Le test fonctionnel est pour vérifier si les fonctionnalités du logiciel sont réalisées et correspondent aux besoins de client. Pour ce projet, ses fonctionnalités principales sont la lecture des 3 fichiers (bâtiment, care, distance), la recherche la meilleure solution d’affectation, et l’écriture des solutions au fichier. Les fonctionnalités sur le fichier sont déjà testé pendant le test unitaire, donc cette chapitre décrit principalement le test fonctionnel pour la recherche la meilleure solution d’affectation．

On a deux parties à faire : déterminer le nombre d’itération, et évaluer la solution trouvée. On suppose que le nombre de fourmis est 50. On prend 3 échantillons pour faire le test :

* 20 bâtiments et 5 cares ;
* 50 bâtiments et 20 cares ;
* 500 bâtiments et 187 cares.

## Détermination du nombre d’itération

Le nombre d’itération dépend de la taille d’échantillon et le nombre de fourmis. Si la taille d’échantillon est plus petite et le nombre de fourmis est plus grande, la vitesse de trouver le meilleur est plus rapide, c’est-à-dire que le nombre d‘itérations plus petite.

La solution sera presque stable dans certain nombre d’itération. On peut prendre un nombre plus grand possible comme la fois d’itérations, et après que l’exécution de programme finit, on peut dessiner un figure de qualités de solution pour regarder si le courbe atteint la stabilité, et imprime l’id d’itération où la qualité de solution est maximum.

On dessiner un figure dont l’axe x est l’itération et l’axe y est la qualité de solution. Il y a deux courbes dans la figures, la courbe bleu représente la qualité de meilleure solution d’une itération, et la courbe rouge représente la qualité moyenne de solutions d’une itération. Si la différence entre les deux courbe est plus petite, cela signifie que la meilleure solution actuelle est plus fiable.

1. Pour 20 bâtiments et 5 cares

On fait 100 itération, la figure de qualités est comme figure 3.1.

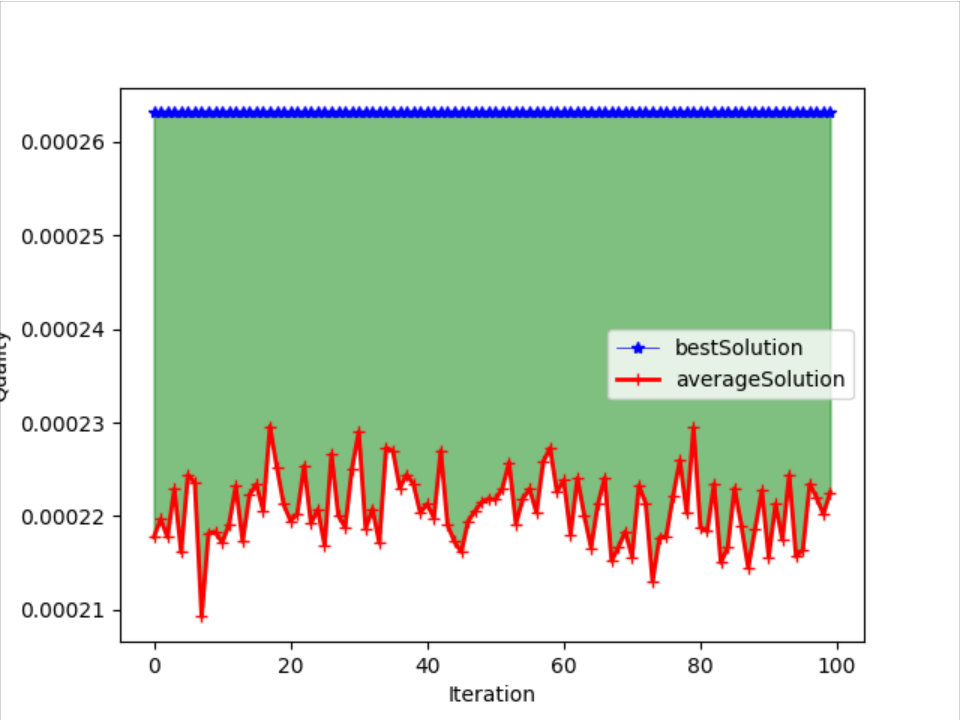


Figure 3.1 la figure de qualités pour 20 bâtiments et 5 cares avec 100 itérations

On peut voir que le courbe bleu est stable à partir de début. Et l’id d’itération imprimé de la meilleure solution est 0, donc pour cet échantillon, le nombre d’itération n’est pas important.

1. Pour 50 bâtiments et 20 care

On fait 500 itérations, la figure de qualité est comme figure 3.2.

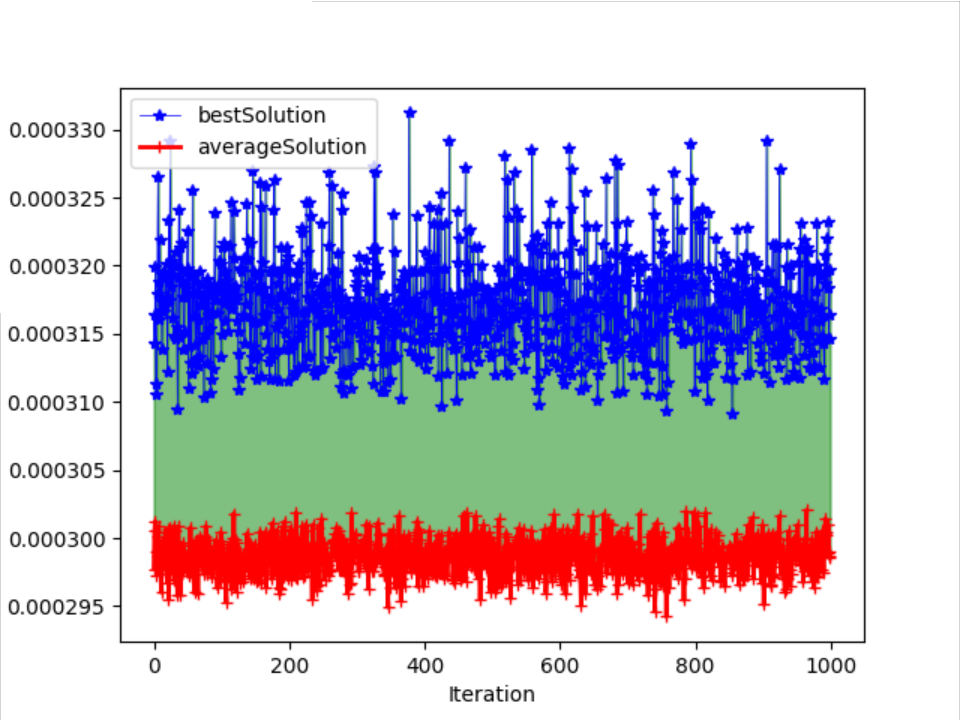


Figure 3.2 La figure de qualités pour 50 bâtiments et 20 cares avec 1000 itérations

Selon la figure 3.2, on peut voir que la valeur de courbe bleu est entre 0.000330 et 0.000310, et la valeur de courbe rouge est déjà stable. L’id d’itération de la meilleure solution imprimé est 377, donc pour cet échantillon, on peut mettre le nombre d’itération en 500. (On mettre un nombre un peu plus grand pour éviter le cas accidentel).

1. Pour 500 bâtiments et 187 cares.

On fait 70 itération, la figure de qualités est comme figure 3.3.

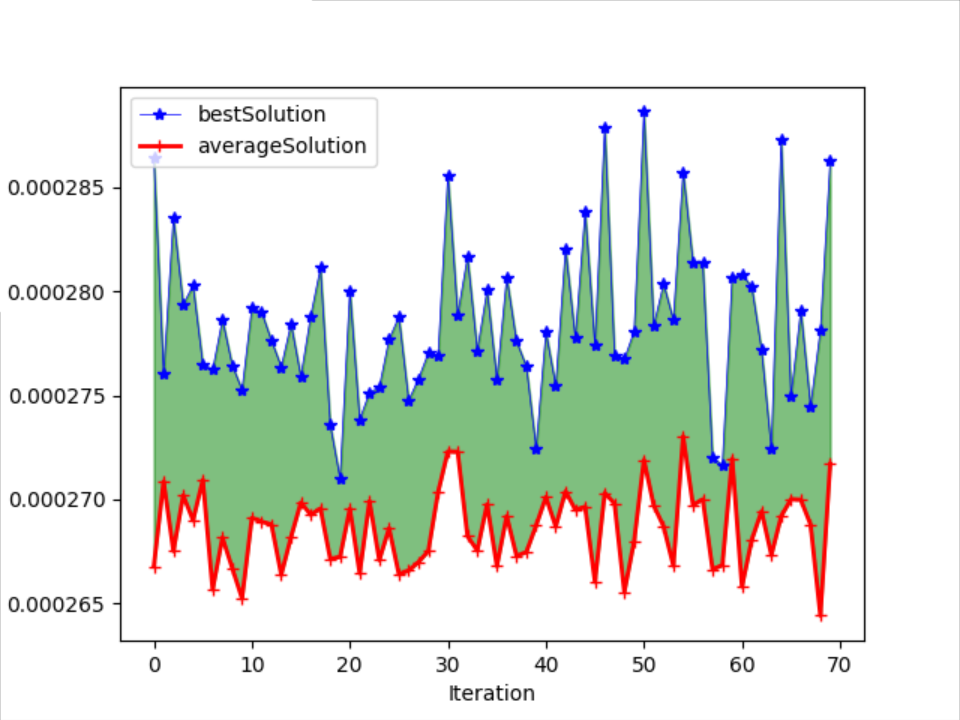


Figure 3.3 la figure de qualités pour 500 bâtiments et 187 cares avec 70 itérations

Selon la figure 3.3, on peut voir que quand le programme exécute jusqu’à 70 itérations, le courbe n’est pas encore stable, et l’id d’itération de la meilleure solution imprimé est 70. Donc pour dermatite le nombre d’itérations pour cet échantillon. Il faut faire plus d’expérimentation et prendre plus d’itération.

## Evaluation de la solution

D’abord, on peut dissiner les points de bâtiments et care. La figure 3.4 montre que l’échantillon de 50 bâtiments et 10 care.

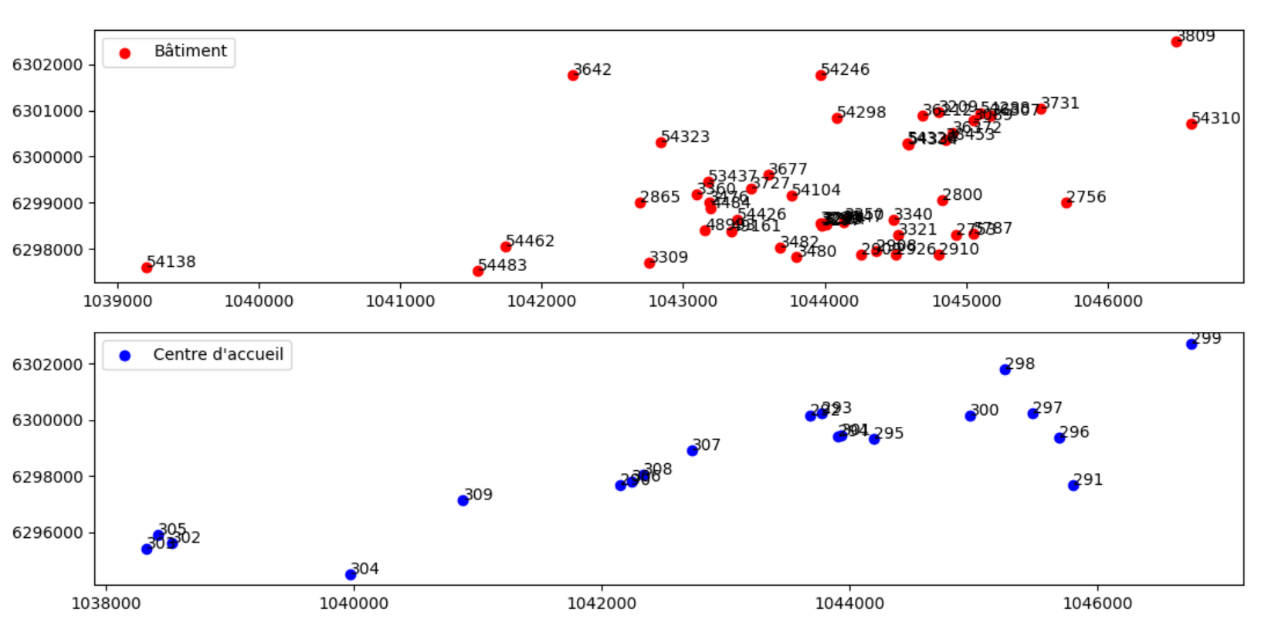


Figure 3.4 La figure d’échantillon de 50 bâtiments et 10 care

Ensuite, on peut référer le fichier de solution pour évaluer la meilleure solution trouvée.

En fait, pour l’échantillon dont la tailles est petite, on peut facilement et directement savoir si la solution est bonne ou pas selon les distances et les capacités de care. Parce que la solution dépend du plusieurs contraintes, on peut vérifier la justesse de solution, mais il faut peut-être plus de conaissance sur l’aménagement pour vérifier si la solution est raisonnable.

# Tests de performance

Le test de performance est pour améliorer la performance de logiciel par évaluer le temps d’exécution, l’occupation de mémoire et l’utilisation de CPU, etc.

La langage Python fournit un outil qui s’appelle « profile » pour détecter la performance de programme. Pour ce projet, le tests de performance que on fait consiste à deux parties : le test sur le temps d’exécution, et l’occupation de mémoire. Il suffit ajouter l’annotation « @profile » avant le nom de méthode à tester. On teste principalement la performance de choisir les cares et la performance d‘affecter les bâtiments. Donc on ajoute l’annotation avant la méthode « allocateBuilding() » et « chooseCare() » dans la classe « AlgorithmController ».

## 4.1 Test du temps d’exécution

Pour tester le temps d’exécution, il faut installer la librairie « line\_profiler » pour le Python. Après l’installation, on utilise la commande suivante dans le CMD pour tester le script de Python :

|  |
| --- |
| kernprof -l -v [nom du fichier Python à tester] |

Cette libraire peut mesurer le temps d’exécution de chaque ligne de méthode, et afficher le résultat ligne par ligne. La figure 4.1 et la figure 4.2 montrent respectivement le résultat de test du temps d’exécution.

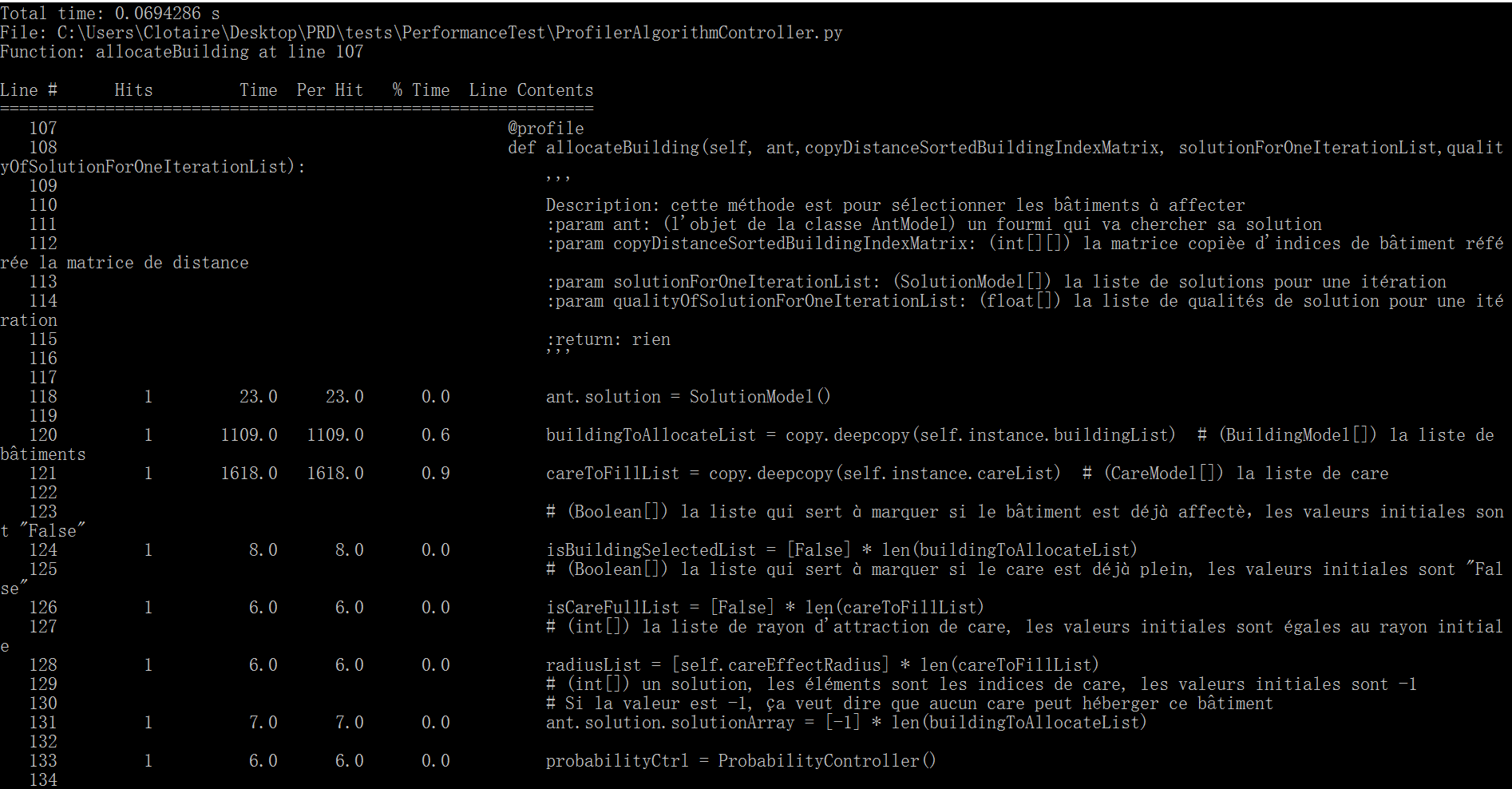


Figure 4.1 Le test du temps d’exécution lors d’affecter les bâtiments



Figure 4.2 Le test du temps d’exécution lors de choisir les cares

L’unité de temps est 3.95062e-07 sec. Dans la figure 4.1 et la figure 4.2, la troisième colonne est le temps d’exécution de la ligne, et la cinquième colonne est le pourcentage de temps.

Selon les résultats, on peut voir que les parties suivantes prennent la plupart du temps :

* Copier la liste de bâtiments et copier la liste de cares ; ( 0.6% et 0.9%)
* Mettre à jour le paramètre de phéromone « deltaTau » (0.5%)
* Calculer la qualité de solution (0.4%)
* La plupart du temps est pris par la fonction « print() » pour afficher le déroulement du programme (presque 96%)

## 4.2 Test d’occupation de mémoire

Pour tester l’occupation de mémoire, il faut installer la librairie « memory\_profiler » pour le Python. Après l’installation, on utilise la commande suivante dans le CMD pour tester le script de Python :

|  |
| --- |
| python -m memory\_profiler [nom du fichier Python à tester] |

Cette libraire peut mesurer l‘utilisation de mémoire de chaque ligne de méthode, et afficher le résultat ligne par ligne. La figure 4.3 et la figure 4.4 montrent respectivement le résultat de test de l’occupation de mémoire.

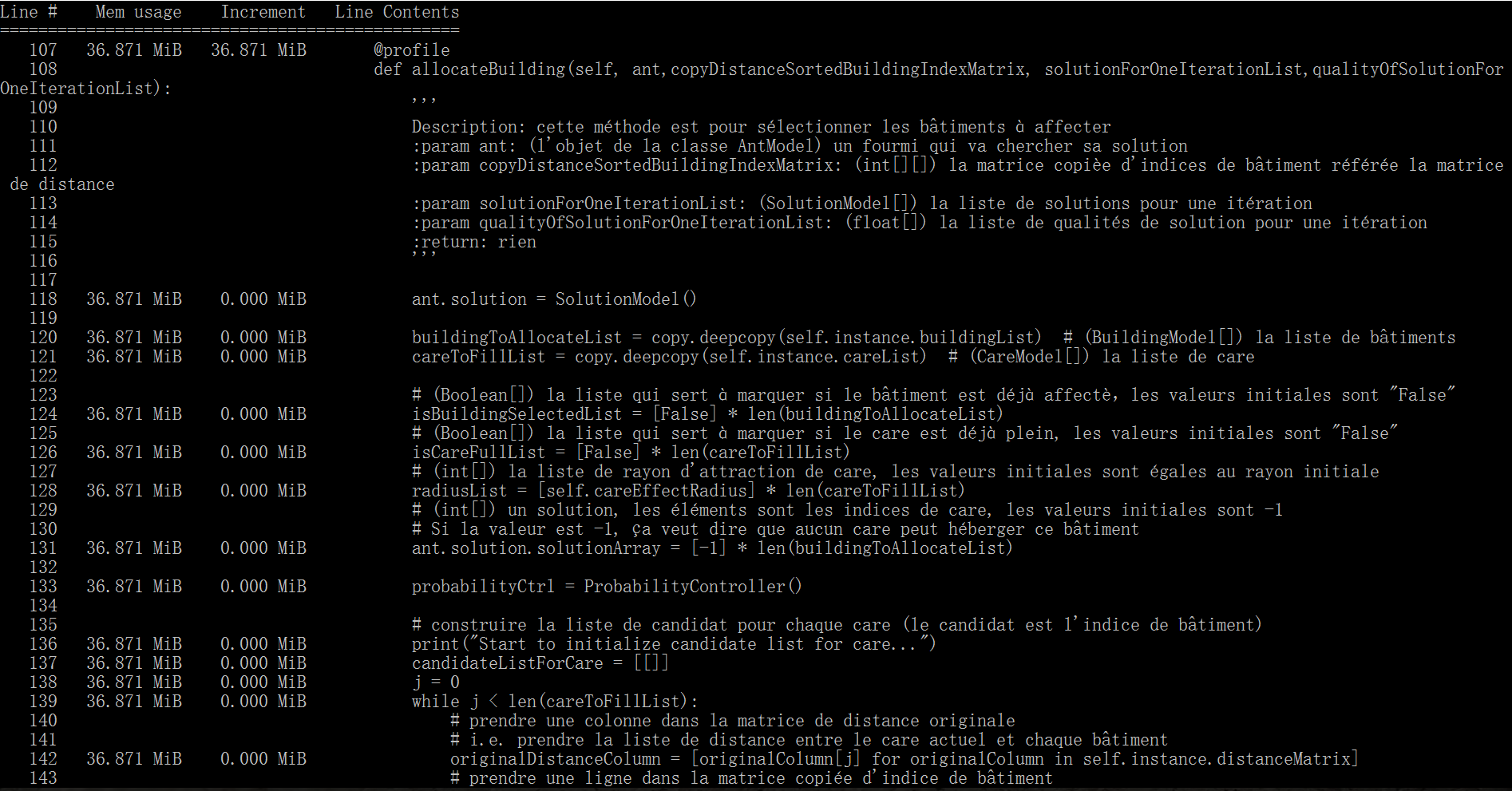


Figure 4.3 Le test de l’occupation de mémoire lors d’affecter les bâtiments

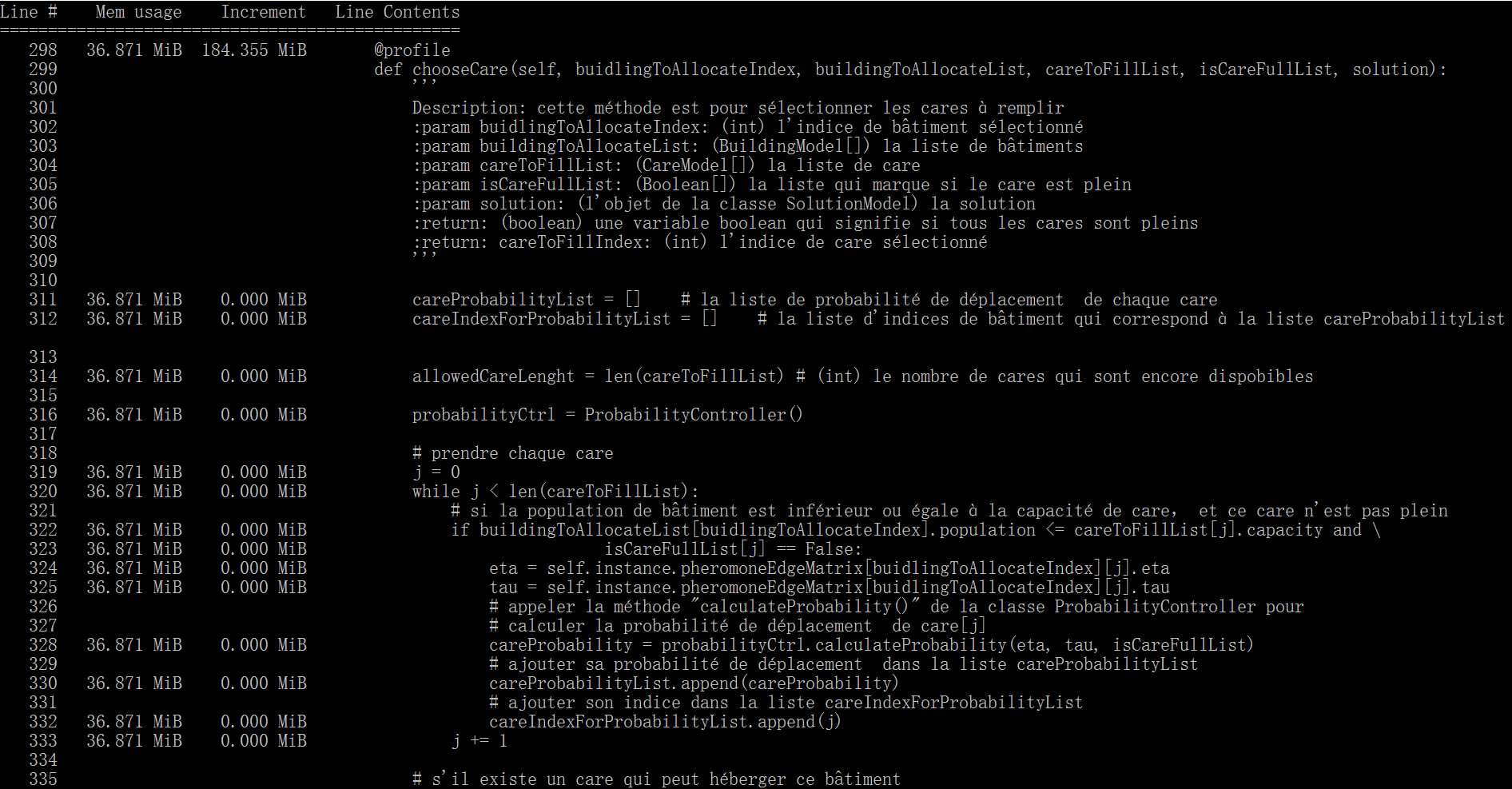


Figure 4.4 Le test de l’occupation de mémoire lors de choisir les cares

Dans la figure 4.3 et la figure 4.4, la deuxième colonne est le mémoire utilisé de la ligne, et la troisième colonne est l’incrément de mémoire utilisé.

Selon les résultats, on peut voir que quand la méthode « allocateBuilding() » est appelée, l’utilisation de mémoire incrémente de 36.871MB, et quand la méthode de « chooseCare() » est appelé, l’utilisation de mémoire incrémente de 184.355MB, les autres ligne ne fait pas l’incrémentation de mémoire utilisé.