Qualité logicielle - TinyMonkeys

Cahier de test

# Introduction :

Ce document présente tous les tests que j’ai réalisés sur le projet TinyMonkeys dans le cadre du cours de Qualité Logicielle, ainsi que les conclusions que j’ai pu en tirer.

# Tests :

Avant de tester la méthode de déplacement, il faut d’abord tester la méthode vérifiant l’autorisation ou non du déplacement car deplacerSinge utilise verifierDeplacementSinge et il ne faut pas fausser les résultats de tests de deplacerSinge qui pourraît venir de cette méthode verifierDeplacementSinge.

|  |  |
| --- | --- |
| Classe testée | SingeErratique.java |
| Méthode testée | verifierDeplacementSinge |
| Objectif du test | Vérifier que le déplacement du singe est autorisé ou interdit selon un certain contexte.  Le déplacement est autorisé si la case sur laquelle avancer est :   * De la terre * Un pirate (le pirate meurt)   Le déplacement est interdit si la case sur laquelle avancer est :   * Un autre singe * De l’eau |
| Bouchonnage ou pas | Oui |
| Données de tests / pré-conditions | Le singe est placé au centre (2,2).  Le pirate est situé à droite (2,3).  De l’eau se situe en dessous du singe (3,2).  Un singe se situe au-dessus (1,2).  Voici une capture résumant les données de tests : |
| Résultat | Le déplacement est bien autorisé pour la terre et le pirate, et bien interdit si un autre singe ou de l’eau se trouve la case à atteindre. |
| Conclusion | La méthode verifierDeplacementSinge est validée car elle réalise les fonctions attendues. |

|  |  |
| --- | --- |
| Classe testée | SingeErratique.java |
| Méthodes testée | deplacerSinge |
| Objectif du test | Vérifier que le singe se déplace à chaque itération.  Vérifier que le singe ne se déplace pas en diagonale. |
| Bouchonnage ou pas | Oui |
| Données de tests / pré-conditions | Le singe est placé à la case (2,2).  Il n’y a que de la terre autour de lui. |
| Résultat | Le singe se déplace à chaque nouvelle itération.  Le singe ne va pas en diagonale. |
| Conclusion | La méthode est valide car elle réalise les fonctions attendues. |

*Note : J’ai réalisé chaque test d’équiprobabilité avant et sans bouchonnage mais avec les mêmes données de tests car j’ai fait correspondre les deux (par exemple dans le mock j’ai mis de l’eau sur la case (1,0) ce qui est aussi le cas dans la carte d’exemple non bouchonnée). Il est important de d’abord tester avec bouchonnage pour voir si le programme nous retourne ce que l’on attend selon un certain contexte, on peut ensuite tester plus globalement.*

|  |  |
| --- | --- |
| Classes testées | SingeErratique.java |
| Méthodes testées | deplacerSinge |
| Objectif du test | Vérifier l’équiprobabilité du déplacement du singe lorsque 4 directions sont possibles (il n’y a que de la terre autour de lui). |
| Bouchonnage ou pas | Les deux |
| Données de tests / pré-conditions | Le singe est placé au centre (2,2).  Il n’y a que de la terre autour de lui. |
| Résultat | Avec un delta égal à 3 % et un nombre d’itération à 10 000 je constate que le nombre d’itération pour la gauche, le haut, la droite et le bas est équilibré. En effet pour chacune des directions il y a environ 1 / 4 itérations. |
| Conclusion | Lorsque le singe peut se déplacer dans quatre directions, les directions réalisées sont équiprobable sur 10 000 itérations. |

|  |  |
| --- | --- |
| Classes testées | SingeErratique.java |
| Méthodes testées | deplacerSinge |
| Objectif du test | Vérifier l’équiprobabilité du déplacement du singe lorsque 3 directions sont possibles (il y a de l’eau à sa gauche). |
| Bouchonnage ou pas | Les deux |
| Données de tests / pré-conditions | Le singe est placé au centre (2,2).  Il y a de l’eau à sa gauche (2,1). |
| Résultat | Avec un delta égal à 3 % et un nombre d’itération à 10 000 je constate que le nombre d’itération pour le haut, la droite et le bas est équilibré. En effet pour chacune des directions il y a environ 1 / 2 itérations. De même, le singe ne va jamais à gauche. |
| Conclusion | Lorsque le singe peut se déplacer dans trois directions, les directions réalisées sont équiprobable sur 10 000 itérations. |

|  |  |
| --- | --- |
| Classes testées | SingeErratique.java |
| Méthodes testées | deplacerSinge |
| Objectif du test | Vérifier l’équiprobabilité du déplacement du singe lorsque 3 directions sont possibles (il y a de l’eau à sa gauche). |
| Bouchonnage ou pas | Les deux |
| Données de tests / pré-conditions | Le singe est placé au centre (2,2).  Il y a de l’eau à sa gauche (2,1).  Il y a de l’eau au-dessus du singe. (1,2). |
| Résultat | Avec un delta égal à 3 % et un nombre d’itération à 10 000 je constate que le nombre d’itération pour la droite et le bas est équilibré. En effet pour chacune des directions il y a environ 1 / 2 itérations. De même, le singe ne va jamais à gauche et / ou en haut. |
| Conclusion | Lorsque le singe peut se déplacer dans deux directions, les directions réalisées sont équiprobable sur 10 000 itérations. |

|  |  |
| --- | --- |
| Classes testées | SingeErratique.java |
| Méthodes testées | deplacerSinge |
| Objectif du test | Vérifier l’équiprobabilité du déplacement du singe lorsque 3 directions sont possibles (il y a un autre singe à sa gauche). |
| Bouchonnage ou pas | Bouchonnage car il est difficile d’ajouter un singe à une case précise sans les mocks. |
| Données de tests / pré-conditions | Le singe est placé au centre (2,2).  Il y a un autre singe au-dessus de lui. (1,2). |
| Résultat | Avec un delta égal à 3 % et un nombre d’itération à 10 000 je constate que le nombre d’itération pour la gauche, la droite et le bas est équilibré. En effet pour chacune des directions il y a environ 1 / 3 itérations. De même, le singe ne va jamais n haut. |
| Conclusion | Lorsque le singe peut se déplacer dans trois directions, les directions réalisées sont équiprobable sur 10 000 itérations. |

# Couverture des tests avec EMMA

L’outil EMMA permet de visualiser la couverture du code. Je remarque que ma couverture du code n’est pas parfaite car la classe que j’ai testé : SingeErratique.java n’est ouvert qu’à 83,3 %. En effet c’est la méthode verifierDeplacementSinge qui est en cause. Cette analyse prouve l’utilité de cet outil qui permet alors de voir si du code a été écrit « pour rien » ou bien si les tests ont mal été écrit.

