



QUALITE LOGICIELLE : TINYMONKEY

GUILLAUME ESCARIEUX - FIL A2 - EMN

V1 : 01/02/2014 V2 : 08/02/2014

TABLE DES MATIERES

Cahier de tests

INTRODUCTION _____	1
OBJECTIFS ET PERIMETRE DES TESTS _____	2
DEROULEMENT ET RESULTATS DES TESTS _____	4
CONCLUSION _____	9
TABLE DES ILLUSTRATIONS _____	10



Figure 1 : Terrible singe erratique

INTRODUCTION

OBJET :

Ce document retranscrit le déroulement des tests du projet TinyMonkey : un jeu vidéo développé en JAVA dans lequel le joueur, pirate échoué sur une île, cherche à attraper le trésor caché tout en échappant à une horde de singes erratiques. Ce document retranscrit la description des tests réalisés, leur implémentation et leurs résultats.



Figure 2 : Logo du langage JAVA

PORTEE :

Les personnes concernées par ce document sont tout d'abord le demandeur du logiciel, ensuite l'équipe développeur-testeur et enfin toute personne cherchant à découvrir le secret de la (formidable) réalisation de ce logiciel.

OBJECTIFS ET PERIMETRE DES TESTS

ETENDU DES TESTS

Le cahier des charges exprime que la partie du logiciel la plus complexe est le déplacement automatique des singes erratiques. Nous allons concentrer nos tests sur cette fonctionnalité en utilisant l'outil de test unitaire JUnit. Les autres fonctionnalités seront donc vérifiées par interface graphique, en exécutant le programme.



Figure 3 : Logo de JUnit

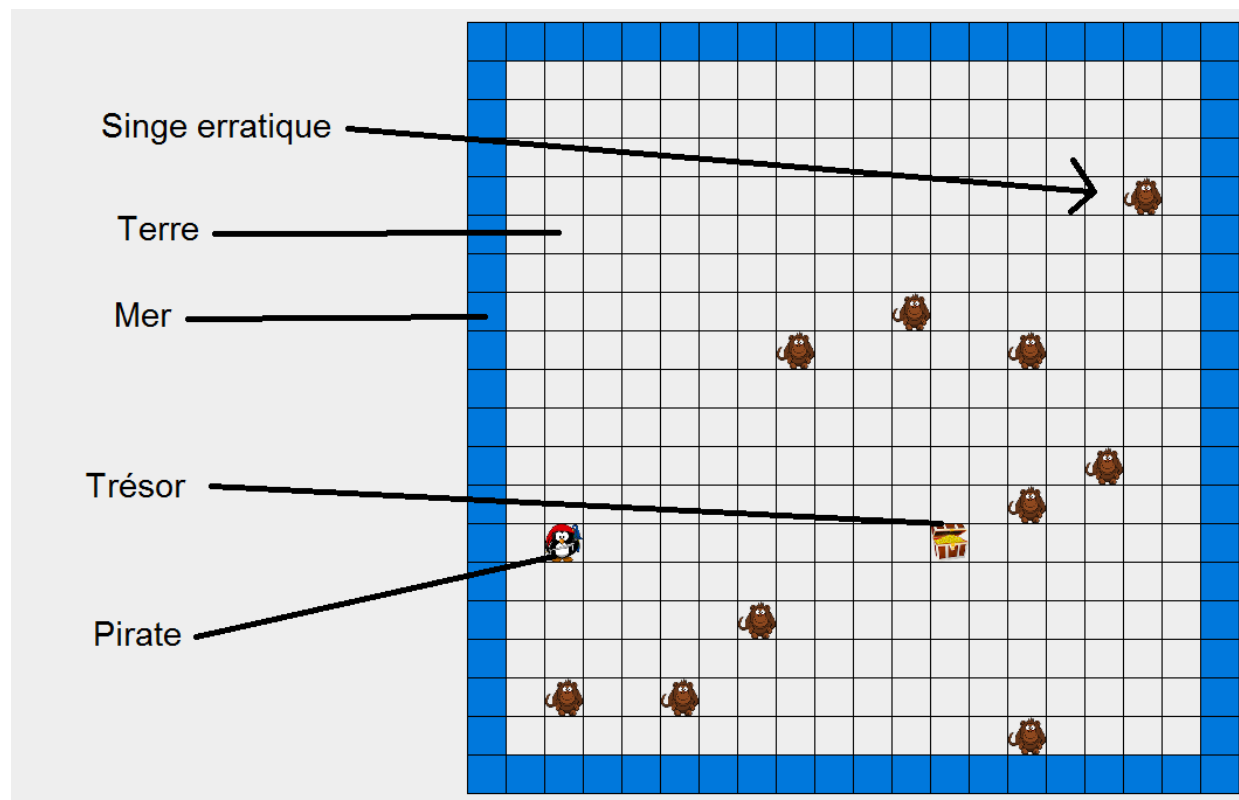


Figure 4 : Capture d'écran du jeu

DEPLACEMENT DES SINGES ERRATIQUES

La règle de déplacement des singes est simple : chaque singe peut se déplacer (une case par une case) dans une des quatre cases qui l'entourent (au-dessus, en dessous, à gauche et à droite), à condition qu'il n'y est ni d'élément MER ni un autre singe. La présence d'un trésor ou du pirate ou bien une case vide n'empêche donc pas le singe de se déplacer. Chaque singe doit se déplacer de manière aléatoire dans une des cases libres adjacentes.

C'est la méthode `deplacerSinge()` des singes erratiques que l'on cherche à tester. Nous simulerons donc les déplacements d'un singe lorsqu'il a quatre cases disponibles, trois cases disponibles (un obstacle), deux cases disponibles (deux obstacles...), une seule et enfin aucune case disponible (pas de déplacement). Nous testerons également l'équiprobabilité du déplacement des singes : nous vérifierons bien que chaque case libre disponible autour du singe a autant de chances d'être choisie que les autres.

Pour réaliser ces tests, nous utiliserons l'outil EasyMock qui permettra de simuler le comportement et les méthodes de l'élément Ile (bouchonnage). Par sécurité, nous allons tout de même tester les méthodes utiliser dans l'île et ensuite tester le déplacement des singes sans bouchonnage.



Figure 5 : Logo de EasyMock

DEROULEMENT ET RESULTATS DES TESTS

DEPLACEMENT DES SINGES ERRATIQUES EN BOUCHONNE

Nous avons tout d'abord utilisé des mocks pour boucher l'île car les singes font appels à des méthodes de celle-ci pour évaluer leur environnement.

La première chose à tester est un déplacement simple (sur une carte déserte). Nous demandons donc à un singe de se déplacer et nous vérifions bien qu'il s'est déplacé sur une case adjacente (non immobile, non déplacer en diagonale, non « téléporté » plusieurs cases plus loin).

Lorsque ceci a été testé, nous avons pu tester toutes les possibilités de déplacements possibles. Les mocks nous ont permis de simuler simplement 1 obstacle autour, puis deux, puis trois et enfin des obstacles tout autour du singe. Pour chaque cas, nous avons vérifié que les coordonnées du singe après son déplacement ne correspondent pas à celles d'un obstacle.

Une fois testé que le singe ne se déplace jamais sur un obstacle environnant, nous avons testé que le singe choisisse bien de manière équiprobable une des cases disponibles. Pour mettre à bien ce test, nous avons reproduit 10000 fois le déplacement du singe pour chaque configuration précédente (en faisant varier le nombre d'obstacles environnants). A l'aide d'un compteur, nous avons pu vérifier le nombre de fois que chaque case disponible a été choisie et comparer ce nombre à un pourcentage du nombre total. Par exemple s'il y a un seul obstacle autour du singe, il reste trois déplacements possibles, et donc 33% de chance que chaque case soit choisie. 33% de 10000 donne 3300, nous nous donnons une marge de 2% soit 200 car l'aléatoire ne permet pas une équiprobabilité parfaite. Pour deux obstacles, il a deux cases disponibles, soit 50% de chance par case, et quand il y trois ou quatre obstacles, là il y a exactement 100% de chance que le singe choisisse la case disponible dans le premier cas et qu'il ne bouge pas dans le deuxième cas.

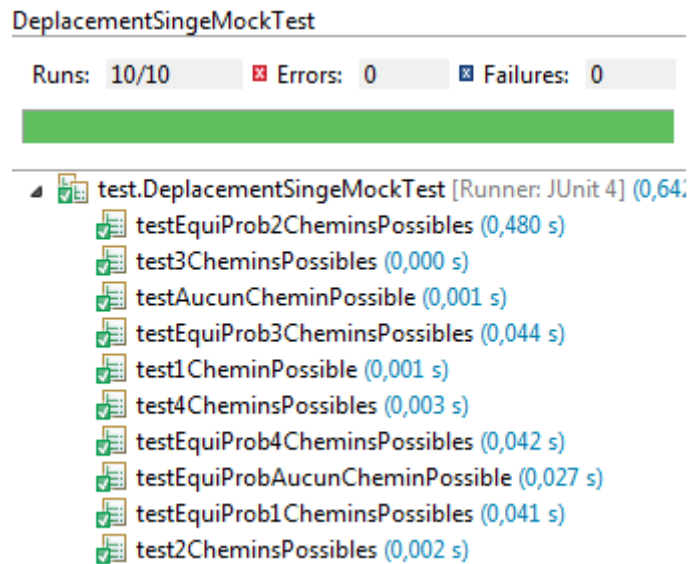


Figure 6 : Tests de déplacement de singe erratique en bouchonné

TEST DES METHODES DE L'ÎLE

Les tests vus dans la section précédente ont montrés que l'algorithme de déplacement des singes est correct et conforme au cahier des charges. Cependant il reste à tester ce déplacement dans des conditions réelles (sans bouchonnage de l'île). Avant d'utiliser les méthodes de l'île nous allons les tester pour s'assurer qu'elles fonctionnent correctement. Les trois méthodes de l'île utilisées par les singes erratiques sont :

-singeEstPresentMock (case) : cette méthode permet de savoir si un singe est présent sur une case donnée. Cette méthode fait appel à tous les écouteurs des singes pour savoir si leur position correspond à la case testée. Nous avons dans un premier temps mocké un singe pour simuler sa position. Nous avons fait deux tests : l'un en faisant croire qu'un singe était présent et l'autre qu'il était absent. La méthode a bien réagi.

-singeEstPresent (case) : Nous avons réalisé exactement le même test qu'auparavant sans bouchonnage. Nous avons testé la méthode sur une case vide et sur une case où nous avons rajouté un vrai singe erratique : une fois encore l'île a bien détecté la présence ou non d'un singe.

-`deplacementsPossibles(caseDuSinge)` : Cette méthode de l'île est très importante car elle définit les déplacements possibles autour d'une case. Nous avons testé cette méthode en ajoutant concrètement des obstacles de manière progressive tout autour de la case à tester.

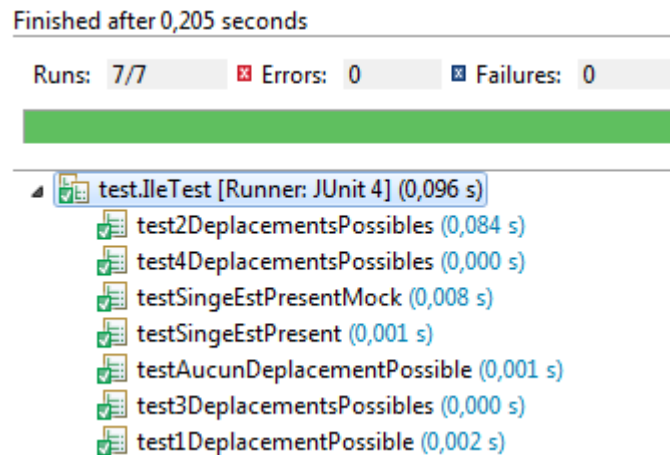
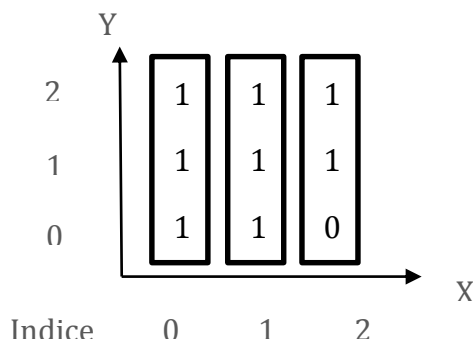


Figure 7 : Tests de l'île

DEPLACEMENT DES SINGES ERRATIQUES NON BOUCHONNE

Maintenant que nous avons vérifié que les méthodes de l'île appelées par les singes sont conformes au cahier des charges, nous pouvons tester le déplacement des singes en non bouchonné. Pour cela, nous avons créé un véritable objet île auquel nous avons inséré notre singe erratique à tester. La figure 8 représente la matrice de la carte taille 3x3. Elle est définie de cette manière : `carte[Abscisse][Ordonne]`. Le X correspond aux abscisses et le Y correspond aux ordonnées. Donc dans l'exemple ci-dessous, la case MER (0) correspond à l'indice : `carte[2][0]`.



Le singe à tester est placé au centre (en 1,1)

Figure 8 : Matrice carte de l'île de taille 3x3

En se basant sur cette matrice, nous avons pu rajouter des éléments de type MER ou bien d'autres singes erratiques, qui constituent les obstacles rendant les cases environnantes indisponibles. Grâce à cela, nous avons pu reproduire avec de véritables objets les conditions des tests réalisées avec les mocks dans `DeplacementSingeMockTest`. Nous avons effectués la totalité des tests : aucun obstacle autour, un obstacle, deux obstacles, trois obstacles et des obstacles tout autour. Nous avons pu également tester l'équiprobabilité des déplacements en répétant le déplacement tout en réinitialisant l'emplacement du singe à tester à chaque fois.

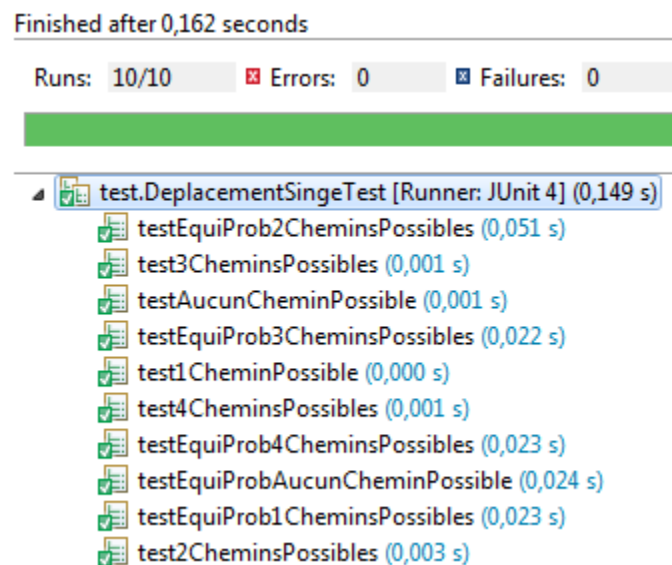


Figure 9 : Tests de déplacements de singes erratiques en non bouchonné

BILAN SUR LA COUVERTURE DES TESTS

Grâce à l'outil Emma, nous pouvons observer la couverture des tests. La figure 10 indique le pourcentage de couverture par source en lançant la suite des tests décrits plus haut. Nous pouvons remarquer que le pourcentage de couverture de la vue, du contrôleur et de la classe principale est de 0%, cela est normal puisque nous désirons tester uniquement l'algorithme de déplacements des singes. La couverture de la classe `SingeErratique` est de 76,4% car comme nous n'avons pas créé de `Pirate` lors des tests, toute la partie gestion de collision n'a pas été utilisée. La couverture de l'Ile est de 60,8% car les parties concernant

le pirate et le trésor n'ont pas été utilisées puisque nous ne nous sommes pas préoccupés de ces objets.

MonkeyIslandTestSuite (3 févr. 2014 17:04:46)































Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
▲ TinyMonkey	 69,4 %	4 625	2 035	6 660
▲ src	 69,4 %	4 625	2 035	6 660
▲ tinymonkeys.vue	 0,0 %	0	1 117	1 117
▷ VueCarte.java	 0,0 %	0	312	312
▷ Fenetre.java	 0,0 %	0	297	297
▷ GestionImages.java	 0,0 %	0	209	209
▷ VueElement.java	 0,0 %	0	132	132
▷ VuePirate.java	 0,0 %	0	70	70
▷ VuePersonnage.java	 0,0 %	0	39	39
▷ VueSingeErratique.java	 0,0 %	0	29	29
▷ VueTrésor.java	 0,0 %	0	29	29
▲ tinymonkeys.modele	 48,8 %	456	479	935
▷ Ile.java	 60,8 %	271	175	446
▷ BandeDeSingesErratiques.java	 11,5 %	19	146	165
▷ Pirate.java	 25,1 %	43	128	171
▷ SingeErratique.java	 76,4 %	81	25	106
▷ Trésor.java	 0,0 %	0	5	5
▷ AbstractElement.java	 100,0 %	34	0	34
▷ Singe.java	 100,0 %	8	0	8
▲ test	 93,4 %	4 169	296	4 465
▷ DeplacementSingeMockTest.java	 92,4 %	1 819	149	1 968
▷ DeplacementSingeTest.java	 91,9 %	1 644	144	1 788
▷ MonkeyIslandTestSuite.java	 88,0 %	22	3	25
▷ IleTest.java	 100,0 %	684	0	684
▲ tinymonkeys.controleur	 0,0 %	0	130	130
▷ Controleur.java	 0,0 %	0	130	130
▲ tinymonkeys	 0,0 %	0	10	10
▷ TinyMonkeys.java	 0,0 %	0	10	10
▲ utils	 0,0 %	0	3	3
▷ Constantes.java	 0,0 %	0	3	3

Figure 10 : Couverture des tests avec Emma

CONCLUSION

D'après les résultats observés dans ce cahier de test, nous pouvons conclure que l'objectif de ces tests a bien été atteint : le déplacement aléatoire des singes erratiques a été testé et fonctionne conformément au cahier des charges.

Les singes erratiques sont en capacité d'évaluer les éléments sur les quatre cases qui les entourent, et donc de ne pas s'aventurer sur une case contenant un élément singe ou un élément mer. De plus, il a été également montré que l'algorithme de déplacement est non seulement aléatoire mais également équiprobable.

Or, des tests ne sont jamais exhaustifs. Il reste des fonctionnalités qui n'ont pas été testées unitairement (disparition et réapparition de trésor, mort du pirate). En plus des tests unitaires et fonctionnels réalisés, il serait possible de réaliser d'autres tests comme par exemple des tests de performance. Il est possible d'utiliser des outils tels que JMeter afin de stresser le programme et observer son comportement lorsque l'on ajoute beaucoup de singes erratiques en même temps.

Pour terminer, le programme a été codé selon le cahier des charges imposé, cependant le code a été implémenté de manière à rester évolutif. Il sera notamment facile d'implémenter le fait que les singes mangent les trésors, ou leur faire privilégier de se déplacer sur le joueur lorsque celui-ci se trouve sur une case adjacente, etc...

TABLE DES ILLUSTRATIONS

Figure 1 : Terrible singe erratique	1
Figure 2 : Logo du langage JAVA.....	1
Figure 3 : Logo de JUnit.....	2
Figure 4 : Capture d'écran du jeu.....	2
Figure 5 : Logo de EasyMock.....	3
Figure 6 : Tests de déplacement de singe erratique en bouchonné	5
Figure 7 : Tests de l'île.....	6
Figure 8 : Matrice carte de l'île de taille 3x3	6
Figure 9 : Tests de déplacements de singes erratiques en non bouchonné	7
Figure 10 : Couverture des tests avec Emma	8