Dossier de tests

SAE 2.02	Version : 1.0
Dossier de tests	Date : 22/05/2023
Responsable de rédaction : Florian DE SOUSA, Timothée LONCHAMPT	

Méthodes testées :

Méthode testée : compareTo

1. **Introduction**

Ce dossier répertorie les tests afin de pouvoir créer la méthode **compareTo** dans la classe **Position**

2. **Description de la procédure de test**

On a d'abord effectué des tests en boîte noire dont on détermine le comportement en analysant les entrées et sorties. Pour cela on définit les partition d'équivalence pour tester tout les cas possibles :

- Pour chPosX1 = { x ∈ N, x <= 0 }
- Pour chPosY1 = { x ∈ N, x <= 0 }
- Pour chPosX2 = { x ∈ N, x <= 0 }
- Pour chPosY2 = { x ∈ N, x <= 0 }

3. **Description des informations à enregistrer pour les tests**
    1. **Campagne de test**

Définition du contexte des tests en s'appuyant sur le type de tableau suivant :

Produit testés : méthode compareTo	
Configuration logicielle : IntelliJ	
Configuration matérielle : /	
Date de début : 22/05/2023	Date de finalisation : 22/05/2023
Tests à appliquer : Vérifier que la méthode trouve la distance entre un point x1,y1 et x2,y2	
Responsable de la campagne de test : Florian DE SOUSA, Timothée LONCHAMPT	

2. Tests

Définition de chaque test selon le tableau suivant :

Identification du test : méthode compareTo										
Description du test : Test de toutes les positions pouvant être données, renvoyant la distance parcourue										
Ressources requises : Langage de programmation Java										
Responsable : Florian DE SOUSA, Timothée LONCHAMPT										
Classe	**Position actuelle**		**Position Comparée**		**Résultat Attendu**	**Valeurs numériques 1**		**Valeurs numérique 2**		**Résultat**
	x1	**y1**	**x2**	**y2**		**x1**	**y1**	**x2**	**y2**	
P1	x1 = 0	y1 = 0	x2 = 0	y2 = 0	(|x1-x2|) + (|y1-y2|) = 0	0	0	0	0	0
P2	x1 = 0	y1 = 0	x2 > 0	y2 = 0	(|x1-x2|) + (y1 - y0)	0	0	1	0	1
P3	x1 = 0	y1 = 0	x2 = 0	y2 > 0	(x1-x2) + (|y1 - y0|)	0	0	0	1	1
P4	x1 = 0	y1 = 0	x2 > 0	y2 > 0	(|x1-x2|) + (|y1 - y0|)	0	0	1	1	2
P5	x1 > 0	y1 > 0	x2 > 0, x2 < x1	y2 > 0, y2 < y1	(x1-x2) + (y1-y2)	2	2	1	1	2
P6	x1 > 0	y1 > 0	x2 > 0, x2 > x1	y2 > 0, y2 > y1	(|x1-x2|) + (|y1-y2|)	2	2	3	3	2
P7	x1 > 0	y1 > 0	x2 > 0, x2 < x1	y2 > 0, y2 > y1	(x1-x2) + (|y1-y2|)	2	2	1	3	2
P8	x1 > 0	y1 > 0	x2 > 0, x2 > x1	y2 > 0, y2 < y1	(|x1-x2|) + (y1 - y0)	2	2	3	1	2
P9	x1 > 0	y1 > 0	x2 = x1	y2 > 0, y2 > y1	(x1-x2) + (|y1 - y0|)	2	2	2	3	1
P10	x1 > 0	y1 > 0	x2 = x1	y2 > 0,	(x1-x2) + (y1-y2)	2	2	2	1	1

				y2 < y1						
P11	x1 > 0	y1 > 0	x2 > 0, x2 < x1	y2 = y1	(x1-x2) + (y1-y2)	2	2	1	2	1
P12	x1 > 0	y1 > 0	x2 > 0, x2 > x1	y2 = y1	(|x1-x2|) + (y1-y2)	2	2	3	2	1

3. Résultats des tests

Définition des résultats de chaque test selon le tableau suivant :

Référence du test appliqué : méthode compareTo
Responsable : Florian DE SOUSA, Timothée LONCHAMPT
Date de l'application du test : 04/06/2023
Résultat du test : OK
Occurrences des résultats : systématique

4. Conclusions

Maintenant qu'on a théorisé tous les tests possibles sur des positions différentes, nous pouvons à présent commencer à coder la méthode dans le langage Java.

Après avoir codé tous les tests nous les avons testés et ils fonctionnent tous parfaitement pour toutes les valeurs numériques énoncées.

Fonctionnement des tests :

Pour effectuer les tests nous avons définis 5 listes, des tableaux correspondant aux coordonnées : **x1, y1, x2, y2** et un tableau pour les **résultats qu'on est censé obtenir** .
On a ensuite créé une boucle qui va instancier à un objet **Position** "positionactuelle"les valeurs de **x1** et **y1** et un objet **Position** "positioncompare" les valeurs de **x2** et **y2**. Et va regarder si la méthode **compareTo** renvoie bien le résultat inscrit dans le tableau "**resultatsattendus**". La boucle tourne jusqu'à ce qu'elle parcourt la longueur du tableau **x1** (qui pourrait être n'importe quel tableau instanciés, car ils ont tous la même longueur).

Méthode testée : estPossible

1. **Introduction**

Ce dossier répertorie les tests afin de pouvoir créer la méthode **estPossible** dans la classe **Precond**

2. **Description de la procédure de test**

On a d'abord effectué des tests en boîte noire dont on détermine le comportement en analysant les entrées et sorties. Pour cela on définit les partition d'équivalence pour tester tout les cas possibles :

- Pour condition0 = { x ∈ N, x <= 0 }
- Pour condition1 = { x ∈ N, x <= 0 }
- Pour condition2 = { x ∈ N, x <= 0 }
- Pour condition3 = { x ∈ N, x <= 0 }

3. **Description des informations à enregistrer pour les tests**
    1. **Campagne de test**

Définition du contexte des tests en s'appuyant sur le type de tableau suivant :

<table>
<tr><td colspan="2"><u>Produit testés :</u> méthode estPossible</td></tr>
<tr><td colspan="2"><u>Configuration logicielle :</u> IntelliJ</td></tr>
<tr><td colspan="2"><u>Configuration matérielle :</u> /</td></tr>
<tr><td><u>Date de début :</u> 22/05/2023</td><td><u>Date de finalisation :</u> 22/05/2023</td></tr>
<tr><td colspan="2"><u>Tests à appliquer :</u> Vérifier que la méthode renvoie true si dans les deux duos de précondition, au moins une des préconditions est remplie. Sinon elle renvoie false.</td></tr>
<tr><td colspan="2"><u>Responsable de la campagne de test :</u> Florian DE SOUSA, Timothée LONCHAMPT</td></tr>
</table>

2. Tests

Définition de chaque test selon le tableau suivant :

Identification du test : méthode estPossible						
Description du test : Test de toutes les positions pouvant être données, renvoyant la distance parcourue						
Ressources requises : Langage de programmation Java						
Responsable : Florian DE SOUSA, Timothée LONCHAMPT						
Classe	**Préconditions remplies**	**condition 0**	**condition 1**	**condition 2**	**condition 3**	**Resultat**
P1	Aucune	0	0	0	0	true
P2	Aucune	0	0	y	0	false
	1 condition					true
P3	Aucune	w	0	0	0	false
	1 condition					true
P4	Aucune	0	0	y	z	false
	1 condition					true
	2 conditions					true
P5	Aucune	w	0	y	0	false
	1 condition					false
	2 conditions					true
P6	Aucune	w	x	0	0	false
	1 condition					true
	2 conditions					true
P7	Aucune	w	0	y	z	false
	1 condition					false
	2 conditions (2 dans le même duo)					false
	2 conditions(1 dans chaque duo)					true

	3 conditions					true
P8	Aucune	w	x	y	0	false
	1 condition					false
	2 conditions (2 dans le même duo)					false
	2 conditions(1 dans chaque duo)					true
	3 conditions					true
P9	Aucune	w	x	y	z	false
	1 condition					false
	2 conditions (2 dans le même duo)					false
	2 conditions(1 dans chaque duo)					true
	3 conditions					true
	4 conditions					true

Les applications numériques sont présentes dans le fichier Test “**TestPrecond**”, dues à leur nombre important elles ne peuvent pas rentrer dans ce rapport.

3. Résultats des tests

Définition des résultats de chaque test selon le tableau suivant :

Référence du test appliqué : méthode estPossible
Responsable : Florian DE SOUSA, Timothée LONCHAMPT
Date de l'application du test : 04/06/2023
Résultat du test : OK
Occurrences des résultats : systématique

4. Conclusions

Maintenant qu’on a théorisé tous les tests possibles sur des positions différentes, nous pouvons à présent commencer à coder la méthode dans le langage Java.

Après avoir codé tous les tests nous les avons testés et ils fonctionnent tous parfaitement pour toutes les valeurs numériques énoncées.

Fonctionnement des tests :

Pour effectuer les tests nous avons définis étudié tous les cas pour des valeurs **w,x,y et z**.
Les résultats obtenus correspondent bien aux booléens renseignés.

Méthode testée : Precond(String)

1. **Introduction**

Ce dossier répertorie les tests afin de pouvoir créer la méthode **Precond (String)** dans la classe **Precond**

2. **Description de la procédure de test**

On a d'abord effectué des tests en boîte noire dont on détermine le comportement en analysant les entrées et sorties. Pour cela on définit les partition d'équivalence pour tester tout les cas possibles :

- Pour Precond = {0 =< nbPrecond =< 4}

3. **Description des informations à enregistrer pour les tests**
    1. **Campagne de test**

Définition du contexte des tests en s'appuyant sur le type de tableau suivant :

Produit testés : méthode Precond (String)	
Configuration logicielle : IntelliJ	
Configuration matérielle : /	
Date de début : 22/05/2023	Date de finalisation : 22/05/2023
Tests à appliquer : Vérifier que la méthode instancie le bon nombre de champs (nombre de champs != 0) et si elle y met les bons int	
Responsable de la campagne de test : Florian DE SOUSA, Timothée LONCHAMPT	

2. Tests

Définition de chaque test selon le tableau suivant :

Identification du test : méthode Precond (String)		
Description du test : Test de toutes les types de précondition que les scénarios peuvent nous donner.		
Ressources requises : Langage de programmation Java		
Responsable : Florian DE SOUSA, Timothée LONCHAMPT		
Classe	**str donné (x est un chiffre qui import peux)**	**Résultat**
P1	“()”	Precond(0,0,0,0)
P2	“((x,x),(x,))”	Precond(x,x,x,0)
P3	“((x,),(x,))”	Precond(x,0,x,0)
P4	“((x,x),)”	Precond(x,x,0,0)
P5	“((x,),)”	Precond(x,0,0,0)
P6	“((x,),(x,x))”	Precond(x,0,x,x)
P7	“((x,x),(x,x)”	Precond(x,x,x,x)

Les applications numériques sont présentes dans le fichier Test “**TestPrecond**”. et pour tester si les precondition sont bonne on vas le comparer avec compareTo a des précondition qui sont nos résultat attendu ;

3. Résultats des tests

Définition des résultats de chaque test selon le tableau suivant :

Valeurs numériques 1	**Valeurs numérique 2**	**Résultat du compaTo entre x1 et x2**
x1	**x2**	
“()”	Precond(0,0,0,0)	true
“((2,3),(1,))”	Precond(2,3,1,0)	true
“((4,),(5,))”	Precond(4,0,5,0)	true
“((2,3),)”	Precond(2,3,0,0)	true

“((2,),)”	Precond(2,0,0,0)	true
“((8,),(7,4))”	Precond(8,0,7,4)	true
“((8,9),(7,4))”	Precond(8,9,7,4)	true

Référence du test appliqué : méthode Precond (String)
Responsable : Florian DE SOUSA, Timothée LONCHAMPT
Date de l'application du test : 04/06/2023
Résultat du test : OK
Occurrences des résultats : systématique

4. Conclusions

Maintenant qu’on a théorisé tous les tests possibles sur les Precond différentes, nous pouvons à présent commencer à coder la méthode dans le langage Java.

Après avoir codé tous les tests nous les avons testés et ils fonctionnent tous parfaitement pour toutes les valeurs énoncées.

Méthode testée : faireQuete

1. **Introduction**

Ce dossier répertorie les tests afin de pouvoir créer la méthode **faireQuete(int)** dans la classe **Joueur**

2. **Description de la procédure de test**

On a d'abord effectué des tests en boîte noire dont on détermine le comportement en analysant les entrées et sorties. Pour cela on a besoin de définir les partition d'équivalence

- cas1 : int = {0}
- cas2 : int ={ x > 0 }

3. **Description des informations à enregistrer pour les tests**
    2. **Campagne de test**

Définition du contexte des tests en s'appuyant sur le type de tableau suivant :

Produit testés : méthode faireQuete(int)	
Configuration logicielle : IntelliJ	
Configuration matérielle : /	
Date de début : 22/05/2023	Date de finalisation : 22/05/2023
Tests à appliquer : Vérifier que la méthode modifie les bon champs et avec les bonnes valeurs.	
Responsable de la campagne de test : Florian DE SOUSA, Timothée LONCHAMPT	

2. Tests

Définition de chaque test selon le tableau suivant :

Identification du test : méthode faireQuete(int)		
Description du test : Teste différente quête que le joueur pourrait faire		
Ressources requises : Langage de programmation Java		
Responsable : Florian DE SOUSA, Timothée LONCHAMPT		
Classe	**int : x**	**Résultat tout les champs qui sont modifier**
P1	x = 0	,temps, coordonees
P2	x > 0	temps, chExperience, coordonees

Les applications numériques sont présentes dans le fichier Test “**TestJoueur**”. et pour tester nous avons donner a la classe joueur le scénario scenario_test qui a été créé pour ça.

3. Résultats des tests

Définition des résultats de chaque test selon le tableau suivant :

int : x	**quête à laquelle fait référence x**	**champs qui ont été modifiés et leurs valeurs**
0	0|(1,1)|((2,),)|4|250| ce battre avec un lion	temps : 6 coordonees : (1-1)
1	1|(4,3)|()|2|100| aller au toillette	temps : 9 chExperience : 100 coordonees : (4-3)
2	2|(3,1)|((1,),)|1|150| ce mettre bien	temps : 5 chExperience : 150 coordonees : (3-1)
3	3|(0,4)|((2,),)|3|100| manger du chocolat	temps : 7 chExperience : 200 coordonees : (0-4)

Référence du test appliqué : méthode faireQuete(int)
Responsable : Florian DE SOUSA, Timothée LONCHAMPT
Date de l'application du test : 04/06/2023

Résultat du test : OK
Occurrences des résultats : systématique

4. Conclusions

Maintenant qu'on a théorisé tous les tests possibles sur le estPosible différentes, nous pouvons à présent commencer à coder la méthode dans le langage Java.

Après avoir codé tous les tests nous les avons testés et ils fonctionnent tous parfaitement pour toutes les valeurs énoncées.

Méthode testée : TestGraphe, GlouEfficace et GlouExhaustive

les méthodes TestGraphe, GlouEfficace et GlouExhaustive les teste on bien été coder mais étant donné la complexité de ces test nous les avons vérifiées visuellement, donc nous ne faisons pas de dossier test car secerait trop compliqué.