

IFT3913
Rapport - Tâche 2

David Qi 20250126
Yilin Zang 20209818

9 octobre 2024

Test #1 add(Point2D)

Il est important de tester cette méthode car elle valide que l'opération d'addition des coordonnées dans un objet Point2D fonctionne correctement, garantissant ainsi que les calculs géométriques impliquant des points sont précis. Cela aide à prévenir des erreurs dans les manipulations de points dans l'application, ce qui est essentiel pour des fonctionnalités basées sur des calculs spatiaux.

Emplacement

Code: convenience/Point2D ligne 68.

Test: convenience/Point2DTest ligne 8.

Test #2 normalize()

Ce test est important car il vérifie que la méthode de normalisation des vecteurs fonctionne correctement, en s'assurant que la longueur du vecteur est bien ramenée à 1 tout en préservant sa direction. Il teste également le comportement pour un vecteur nul, garantissant ainsi que le programme gère correctement des cas limites.

Emplacement

Code: convenience/Point2D ligne 55.

Test: convenience/Point2DTest ligne 27.

Test #3 scale(double)

Ce test est important car il s'assure que l'opération de mise à l'échelle (scale) des coordonnées d'un point fonctionne correctement, à la fois pour des facteurs positifs et négatifs. Cela garantit que les transformations géométriques appliquées aux objets Point2D produisent les résultats attendus, ce qui est crucial pour des calculs précis dans des applications graphiques ou géométriques.

Emplacement

Code: convenience/Point2D ligne 40.

Test: convenience/Point2DTest ligne 53.

Couverture **Test #1 Test #2 Test #3**

Avant

Point2D

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
• normalize()	<div><div></div></div>	0%	<div><div></div></div>	0%	2	2	6	6	1	1
• add(Point2D)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	3	3	1	1
• scale(double)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	3	3	1	1
• lengthSquared()	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	1	1	1	1
• Point2D(Point2D)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	2	2	1	1
• length()	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	1	1	1	1
• distanceSquared(Point2D)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	3	0	1
• equalsEpsilon(Point2D, double)	<div><div></div></div>	100%	<div><div></div></div>	100%	0	2	0	1	0	1
• Point2D(double, double)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	4	0	1
• set(Point2D)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	3	0	1
• set(double, double)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	3	0	1
• distance(Point2D)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	1	0	1
• Point2D()	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	1	0	1
Total	75 of 138	45%	2 of 4	50%	7	15	16	32	6	13

Après

Point2D

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
• Point2D(Point2D)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	2	2	1	1
• normalize()	<div><div></div></div>	100%	<div><div></div></div>	100%	0	2	0	6	0	1
• distanceSquared(Point2D)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	3	0	1
• add(Point2D)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	3	0	1
• scale(double)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	3	0	1
• lengthSquared()	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	1	0	1
• equalsEpsilon(Point2D, double)	<div><div></div></div>	100%	<div><div></div></div>	100%	0	2	0	1	0	1
• Point2D(double, double)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	4	0	1
• set(Point2D)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	3	0	1
• set(double, double)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	3	0	1
• distance(Point2D)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	1	0	1
• length()	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	1	0	1
• Point2D()	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	1	0	1
Total	7 of 138	94%	0 of 4	100%	1	15	2	32	1	13

Test #4 lengthSquared(double dx,double dy)

Ce test est important car il permet de s'assurer que la méthode de calcul de la longueur au carré d'un vecteur (dx, dy) est correcte. Elle est utilisée pour mesurer des distances ou effectuer des calculs spatiaux. Tester cette méthode de manière approfondie garantit qu'elle se comporte comme prévu dans différents scénarios.

Emplacement

Code: src/main/java/com/marginallyclever/convenience/helpers/MathHelper.java ligne 26.

Test: src/test/java/com/marginallyclever/convenience/helpers/MathHelperTest.java ligne 91.

Test #5 equals(Tuple2d a0, Tuple2d a1, Tuple2d b0, Tuple2d b1, double epsilon)

Tester cette méthode est essentiel car cela permet d'identifier correctement les segments de ligne comme étant égaux. En testant différents scénarios, on s'assure que la méthode equals() est précise et robuste, gérant les correspondances exactes, inversées et les petites erreurs de calcul, tout en rejetant les segments non correspondants ou partiellement correspondants. Cela garantit la fiabilité des comparaisons de segments dans diverses applications.

Emplacement

Code: src/main/java/com/marginallyclever/convenience/helpers/MathHelper.java ligne 72.

Test: src/test/java/com/marginallyclever/convenience/helpers/MathHelperTest.java ligne 110.

Test #6 lerp(double t, double a, double b)

Le test de la méthode lerp(double t, double a, double b) est crucial pour garantir une interpolation linéaire correcte entre deux valeurs lorsque t varie entre 0 et 1. Il assure que la méthode gère les cas limites et les différentes valeurs avec précision, garantissant ainsi des transitions fiables et sans erreurs dans diverses applications.

Emplacement

Code: src/main/java/com/marginallyclever/convenience/helpers/MathHelper.java ligne 88.

Test: src/test/java/com/marginallyclever/convenience/helpers/MathHelperTest.java ligne 158.

Couverture Test #4 Test #5 Test #6

Avant

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
equals(Tuple2d, Tuple2d, Tuple2d, double)	<div><div></div></div>	0%	<div><div></div></div>	0%	5 5	6 6	1 1
intersectionOfCircles(double, double, double)	<div><div></div></div>	78%	<div><div></div></div>	62%	3 5	0 7	0 1
lengthSquared(double, double)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1 1	1 1	1 1
lerp(double, double, double)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1 1	1 1	1 1
MathHelper()	<div><div></div></div>	0%	<div><div></div></div>	n/a	1 1	1 1	1 1
between(Tuple2d, Tuple2d, Tuple2d, double)	<div><div></div></div>	100%	<div><div></div></div>	100%	0 4	0 12	0 1
lerp(Tuple2d, Tuple2d, double)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0 1	0 4	0 1
roundOff3(double)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0 1	0 2	0 1
Total	116 of 287	59%	11 of 22	50%	11 19	9 34	4 8

Après

MathHelper

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
intersectionOfCircles(double, double, double)	<div><div></div></div>	78%	<div><div></div></div>	62%	3 5	0 7	0 1
MathHelper()	<div><div></div></div>	0%	<div><div></div></div>	n/a	1 1	1 1	1 1
between(Tuple2d, Tuple2d, Tuple2d, double)	<div><div></div></div>	100%	<div><div></div></div>	100%	0 4	0 12	0 1
equals(Tuple2d, Tuple2d, Tuple2d, double)	<div><div></div></div>	100%	<div><div></div></div>	87%	1 5	0 6	0 1
lerp(Tuple2d, Tuple2d, double)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0 1	0 4	0 1
roundOff3(double)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0 1	0 2	0 1
lengthSquared(double, double)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0 1	0 1	0 1
lerp(double, double, double)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0 1	0 1	0 1
Total	18 of 287	93%	4 of 22	81%	5 19	1 34	1 8

Test #7 getPoint(double t, Point2D p)

Le test de la méthode `getPoint(double t, Point2D p)` est essentiel pour vérifier que l'interpolation entre les points de départ et d'arrivée est correcte lorsque `t` varie entre 0 et 1. Il garantit que la méthode renvoie les bonnes coordonnées pour des valeurs allant de 0 (point de départ) à 1 (point d'arrivée), tout en assurant des transitions fluides et sans erreur pour les valeurs intermédiaires, ce qui est crucial dans les applications de géométrie ou de simulation.

Emplacement

Code: `src/main/java/com/marginallyclever/convenience/LineInterpolator.java` ligne 21.

Test: `src/test/java/com/marginallyclever/convenience/LineInterpolatorTest.java` ligne 9.

Test #8 getTangent(double t, Point2D v)

Le test de la méthode getTangent(double t, Point2D v) est primordial pour s'assurer que le vecteur tangent à une ligne est correctement calculé et normalisé. La tangente étant indispensable pour définir la direction du mouvement, ce test est essentiel pour garantir la précision des calculs.

Emplacement

Code: src/main/java/com/marginallyclever/convenience/LineInterpolator.java ligne 30.

Test: src/test/java/com/marginallyclever/convenience/LineInterpolatorTest.java ligne 34.

Test #9 getNormal(double t, Point2D n)

Le test de la méthode getNormal(double t, Point2D n) est nécessaire pour valider que le vecteur normal, perpendiculaire à la tangente, est correctement déterminé.

Emplacement











Code: src/main/java/com/marginallyclever/convenience/LineInterpolator.java ligne 55.

Test: src/test/java/com/marginallyclever/convenience/LineInterpolatorTest.java ligne 55.

Couverture **Test #7 Test #8 Test #9**











Avant

LineInterpolator

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
getTangent(double, Point2D)		0%		0%	4	4	14	14	1	1
getPoint(double, Point2D)		0%		n/a	1	1	3	3	1	1
LineInterpolator(Point2D, Point2D)		0%		n/a	1	1	6	6	1	1
getNormal(double, Point2D)		0%		n/a	1	1	5	5	1	1
LineInterpolator()		0%		n/a	1	1	3	3	1	1
setStart(Point2D)		0%		n/a	1	1	2	2	1	1
setEnd(Point2D)		0%		n/a	1	1	2	2	1	1
getStart()		0%		n/a	1	1	1	1	1	1
getEnd()		0%		n/a	1	1	1	1	1	1
Total	156 of 156	0%	6 of 6	0%	12	12	35	35	9	9

Après

LineInterpolator

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
LineInterpolator()		0%		n/a	1	1	3	3	1	1
getTangent(double, Point2D)		93%		50%	3	4	0	14	0	1
setStart(Point2D)		0%		n/a	1	1	2	2	1	1
setEnd(Point2D)		0%		n/a	1	1	2	2	1	1
getStart()		0%		n/a	1	1	1	1	1	1
getEnd()		0%		n/a	1	1	1	1	1	1
getPoint(double, Point2D)		100%		n/a	0	1	0	3	0	1
LineInterpolator(Point2D, Point2D)		100%		n/a	0	1	0	6	0	1
getNormal(double, Point2D)		100%		n/a	0	1	0	5	0	1
Total	31 of 156	80%	3 of 6	50%	8	12	7	35	5	9

Test #10 Insert()

Vérifie que le QuadGraph insère correctement les points dans les limites et assure un comptage précis des points. C'est crucial pour garantir une gestion fiable des données spatiales.

Emplacement

Code: src/main/java/com/marginallyclever/convenience/QuadGraph.java ligne 53.

Test: src/test/java/com/marginallyclever/convenience/QuadGraphTest.java ligne 21.

Test #11 testCountPoints()

Valide que le QuadGraph compte correctement le nombre total de points après insertion, assurant l'intégrité des données pour les opérations spatiales.

Emplacement

Code: src/main/java/com/marginallyclever/convenience/QuadGraph.java ligne 125.

Test: src/test/java/com/marginallyclever/convenience/QuadGraphTest.java ligne 52.

Couverture **Test #10 Test #11**

Avant

QuadGraph

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
• split()	<div><div></div></div>	0%	<div><div></div></div>	0%	2	2	14	14	1	1
• search(Point2D)	<div><div></div></div>	0%	<div><div></div></div>	0%	10	10	18	18	1	1
• render(GL2)	<div><div></div></div>	0%	<div><div></div></div>	0%	3	3	13	13	1	1
• insert(Point2D)	<div><div></div></div>	0%	<div><div></div></div>	0%	4	4	7	7	1	1
• QuadGraph(double, double, double, double)	<div><div></div></div>	0%	n/a	n/a	1	1	6	6	1	1
• countPoints()	<div><div></div></div>	0%	<div><div></div></div>	0%	3	3	5	5	1	1
• moveSitesIntoChildren()	<div><div></div></div>	0%	<div><div></div></div>	0%	2	2	5	5	1	1
• addCellToOneQuadrant(Point2D)	<div><div></div></div>	0%	<div><div></div></div>	0%	3	3	3	3	1	1
Total	341 of 341	0%	40 of 40	0%	28	28	71	71	8	8

Après

QuadGraph

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
• search(Point2D)	<div><div></div></div>	0%	<div><div></div></div>	0%	10	10	18	18	1	1
• render(GL2)	<div><div></div></div>	0%	<div><div></div></div>	0%	3	3	13	13	1	1
• addCellToOneQuadrant(Point2D)	<div><div></div></div>	88%	<div><div></div></div>	75%	1	3	1	3	0	1
• split()	<div><div></div></div>	100%	<div><div></div></div>	100%	0	2	0	14	0	1
• insert(Point2D)	<div><div></div></div>	100%	<div><div></div></div>	100%	0	4	0	7	0	1
• QuadGraph(double, double, double, double)	<div><div></div></div>	100%	n/a	n/a	0	1	0	6	0	1
• countPoints()	<div><div></div></div>	100%	<div><div></div></div>	100%	0	3	0	5	0	1
• moveSitesIntoChildren()	<div><div></div></div>	100%	<div><div></div></div>	100%	0	2	0	5	0	1
Total	140 of 341	58%	23 of 40	42%	14	28	32	71	2	8