|  |  |
| --- | --- |
| **ID** | 1 |
| **Verfahren** | Parametrisierter JUnit-Test |
| **Klasse** | Ant |
| **Methoden** | calculateLambda(double distance, double pheromone): BigDecimal  Berechnung des Lambda-Wertes einer Strecke zwischen zwei Städten |
| **Vorbedingung** | - |
| **Eingaben** | 3, 1 |
| **Erwartetess Ergebnis** |  |
| **Ergebnis** | **Test erfolgreich** |

|  |  |
| --- | --- |
| **ID** | 2 |
| **Verfahren** | Parametrisierter JUnit-Test |
| **Klasse** | Ant |
| **Methoden** | calculateLambdas(ArrayList<City> cities): Map<City, BigDecimal>  Berechnung der Menge an Lambda-Werten aller Nachbarn |
| **Vorbedingung** | City(1), City(2), City(3)  (Stadt 2 und 3 haben 3 als Distanz zu Stadt 1) |
| **Eingaben** | ArrayList<City>(City(2), City(3)) |
| **Erwartetes Ergebnis** | Map(Stadt(2), ; Stadt(3), ) |
| **Ergebnis** | Map(Stadt(2), ; Stadt(3), )  **Test erfolgreich** |

|  |  |
| --- | --- |
| **ID** | 3 |
| **Verfahren** | Parametrisierter JUnit-Test |
| **Klasse** | Ant |
| **Methoden** | calculateProbability(BigDecimal lambda, BigDecimal sum): BigDecimal  Berechnung der Wahrscheinlichkeit, ob eine bestimmte Stadt besucht wird |
| **Vorbedingung** | - |
| **Eingaben** | , 3 |
| **Erwartetes Ergebnis** |  |
| **Ergebnis** | **Test erfolgreich** |

|  |  |
| --- | --- |
| **ID** | 4 |
| **Verfahren** | Parametrisierter JUnit-Test |
| **Klasse** | Ant |
| **Methoden** | calculateProbabilities(ArrayList<City> cities): Map<City, BigDecimal>  Berechnung der Wahrscheinlichkeiten aller erreichbaren Nachbarn |
| **Vorbedingung** | City(1), City(2), City(3)  (Stadt 2 und 3 haben 3 als Distanz zu Stadt 1) |
| **Eingaben** | ArrayList<City>(City(2), City(3)) |
| **Erwartetes Ergebnis** | Map(Stadt(2), 0.5; Stadt(3), 0.5) |
| **Ergebnis** | Map(Stadt(2), 0.5; Stadt(3), 0.5)  **Test erfolgreich** |

|  |  |
| --- | --- |
| **ID** | 5 |
| **Verfahren** | Parametrisierter JUnit-Test |
| **Klasse** | Ant |
| **Methoden** | run()  Hauptalgorithmus einer Ameise: Berechnen der möglichen Wegstrecke |
| **Vorbedingung** | City(1), City(2), City(3)  (Städte haben untereinander die Distanz 2)  (Zwischen allen Städten liegt ein Pheromonwert von jeweils 1.5) |
| **Eingaben** | - |
| **Erwartetes Ergebnis** | Route der Ameise > 0  Aktuelle Stadt der Ameise == City(1) |
| **Ergebnis** | Route der Ameise == 4  Aktuelle Stadt der Ameise == City(1)  **Test erfolgreich** |

|  |  |
| --- | --- |
| **ID** | 6 |
| **Verfahren** | Parametrisierter JUnit-Test |
| **Klasse** | Ant |
| **Methoden** | updatePheromones()  Aktualisieren der Pheromonmatrix der Kolonie |
| **Vorbedingung** | City(1), City(2), City(3)  (Städte haben untereinander die Distanz 3)  Ameise hat eine Route von City(1), City(2), City(3), City(1) |
| **Eingaben** | - |
| **Erwartetes Ergebnis** | {{1, 1+, 1}, {1, 1, 1+}, {1+, 1, 1}} |
| **Ergebnis** | {{1, 1+, 1}, {1, 1, 1+}, {1+, 1, 1}}  **Test erfolgreich** |

|  |  |
| --- | --- |
| **ID** | 7 |
| **Verfahren** | Parametrisierter JUnit-Test |
| **Klasse** | Ant |
| **Methoden** | visitCity(City b)  Wechsel der aktuellen Stadt sowie Aktualisierung der Route |
| **Vorbedingung** | City(1), City(2)  (Städte haben untereinander die Distanz 2) |
| **Eingaben** | City(2) |
| **Erwartetes Ergebnis** | Aktuelle Stadt der Ameise == City(2) |
| **Ergebnis** | Aktuelle Stadt der Ameise == City(2)  **Test erfolgreich** |

|  |  |
| --- | --- |
| **ID** | 8 |
| **Verfahren** | Parametrisierter JUnit-Test |
| **Klasse** | Ant |
| **Methoden** | visitCity(City b)  Wechsel der aktuellen Stadt sowie Aktualisierung der Route |
| **Vorbedingung** | City(1), City(2), City(3)  (Städte haben untereinander die Distanz 2)  Ameise kann nur City(1) erreichen |
| **Eingaben** | City(2) |
| **Erwartetes Ergebnis** | Ameise bleibt in City(1) |
| **Ergebnis** | Ameise bleibt in City(1)  **Test erfolgreich** |