# Test Description

**Test Name or ID**: calCapacity

**Test Type**: White box

**Description**: Ensure that the function accurately determines the truck's remaining capacity under various scenarios.

**Setup:** Create a Unit-test project

**Test Function**: struct Size calCapacity(const struct Truck truck);

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| This test checks the accuracy of the function when applied to a empty truck. It aims to ensure that the function properly calculates the remaining capacity considering the partial cargo. | struct Truck emptyTruck = { 1, {0.0f, 0.0f}, {0.0f, 0.0f}, 0, nullptr }; | struct Size result(1500, 48) | struct Size return(1500, 48) | pass |
| This test ensures the function behaves correctly when applied to a partially loaded truck. It checks whether the function correctly indicates that the truck has reached its maximum capacity. | struct Truck partiallyLoadedTruck = { 2, {30.0f, 15.0f}, {0.0f, 0.0f}, 0, nullptr }; | struct Size result(1200, 33) | struct Size return(1200, 33) | pass |
| This test ensures the function behaves correctly when applied to a fully loaded truck. It checks whether the function correctly indicates that the truck has reached its maximum capacity. | struct Truck fullTruck = { 3, {1500.0f, 48.0f}, {0.0f, 0.0f}, 0, nullptr }; | struct Size retsult(0, 0) | struct Size return(0, 0) | pass |
| This test ensures the function behaves correctly when applied to a partially loaded truck. It checks whether the function correctly indicates that the truck has reached its maximum capacity. | struct Truck partiallyLoadedTruck = { 2, {1000.0f, 30.0f}, {0.0f, 0.0f}, 0, nullptr }; | struct Size retsult(500, 16) | struct Size retsult(500, 16) | pass |

**Bugs Found**:

(none found)

# Test Description

**Test Name or ID**: checkTruckWeightCap

**Test Type**: White box

**Description**: This test evaluates the internal logic of the checkTruckWeightCap function to ensure it accurately assesses whether the combined weight of a truck and shipment exceeds the truck's weight capacity or not.

**Setup:** Create a Unit-test project

**Test Function**: int checkTruckWeightCap(struct Truck truck, struct Shipment shipment);

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Test when the combined weight of the loaded truck and shipment is below the truck's weight capacity. | truck.loaded.weight = 200  shipment.box.weight = 300 | The function should return 1, indicating that the truck is capable of carrying this weight. | 1 | pass |
| Test when the combined weight of the loaded truck and shipment is below the truck's weight capacity. | truck.loaded.weight = 200  shipment.box.weight = 100 | The function should return 1, indicating that the truck is capable of carrying this weight. | 1 | pass |
| Test when the combined weight of the loaded truck and shipment exceeds the truck's weight capacity. | truck.loaded.weight = 1500  shipment.box.weight = 100 | The function should return 0, indicating that the truck is incapable of carrying this weight. | 0 | pass |
| Test when the combined weight of the loaded truck and shipment exceeds the truck's weight capacity. | truck.loaded.weight = 1000  shipment.box.weight = 600 | The function should return 0, indicating that the truck is incapable of carrying this weight. | 0 | pass |
| Test when the combined weight exactly equals the truck's weight capacity | truck.loaded.weight = 1000  shipment.box.weight = 500 | The function should return 1, indicating that the truck is capable of carrying this weight. | 1 | pass |

**Bugs Found**:

(none found)

# Test Description

**Test Name or ID**: validBoxSize

**Test Type**: White box

**Description**: This test assesses the internal logic of the validBoxSize feature to ensure that you have entered the correct box size.

**Setup:** Create a Unit-test project

**Test Function**: int validBoxSize(double boxSize)

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Valid small size box | boxSize = 0.5 | Function should return 1 because of correct input value | Function return 1. | Pass |
| Valid medium size box | boxSize = 1.0 | Function should return 1 because of correct input value | Function return 1. | Pass |
| Valid large size box | boxSize = 2.0 | Function should return 1 because of correct input value | Function return 1. | Pass |
| Invalid small size box | boxSize = 0.3 | Function should return 0 because of incorrect input value | Function return 0. | Pass |
| Invalid medium size box | boxSize = 1.5 | Function should return 0 because of incorrect input value | Function return 0. | Pass |
| Invalid large size box | boxSize = 5.0 | Function should return 0 because of incorrect input value | Function return 0. | Pass |
|  |  |  |  |  |

**Bugs Found**:

(none found)

# Test Description

**Test Name or ID**: getNumCols

**Test Type**: White Box

**Description**: This test assesses the internal logic of the getNumCols feature to ensure that you have entered a valid range of numbers.

**Setup:** Create a Unit Test

**Test Function**: int getNumCols(const struct Map\* map)

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| ValidNumCols\_LessThan\_Max\_Value | numCols < MAP\_COLS + 1 | Function test should be true | True | Pass |
| InvalidNumCols\_GreaterThan\_Max\_Value | numCols > MAP\_COLS | Function test should be false | False | Pass |
| NumCols\_Same\_As\_Max\_Cols | numCols == MAP\_COLS | Function test should be true | True | Pass |
| InvalidNumCols1 | numCols == 35 | Function test should be false | False | Pass |
| InvalidNumCols2 | numCols == 56 | Function test should be false | False | Pass |
| InvalidNumCols3 | numCols == 3529 | Function test should be false | False | Pass |
|  |  |  |  |  |

**Bugs Found**:

(none found)

# Test Description

**Test Name or ID**: distance

**Test Type**: White Box

**Description**: To verify the correctness of the distance function, which calculates the Euclidean distance between two points in a 2D space.

**Setup:** Create a unit test project.

**Test Function**: double distance(const struct Point\* p1, const struct Point\* p2);

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Tests the calculation of the Euclidean distance between two points. | Point p1 at (0, 0)  Point p2 at (3, 4) | Distance should be approximately 5.0. | 5.0, indicating that the distance calculation is correct. | Pass |
| Tests the calculation of the distance with negative coordinates. | Point p3 at (-1, -1)  Point p4 at (2, 3) | Distance should be approximately 5.0. | 5.0, indicating that the distance calculation handles negative coordinates correctly. | Pass |
| Tests the calculation of the distance between the same point. | Point p5 and p6 both at (0, 0) | Distance should be approximately 0.0. | 0.0, indicating that the distance between the same point is 0. | Pass |
| Tests the calculation of the distance with decimal coordinates. | Point p7 at (1.5, 2.5)  Point p8 at (4.5, 6.5) | Distance should be approximately 5.0. | 5.0, indicating that the distance calculation handles decimal coordinates correctly. | Pass |
|  |  |  |  |  |
|  |  |  |  |  |

**Bugs Found**:

(none found)

# Test Description

**Test Name or ID**: validatePackageWeight

**Test Type**: White Box

**Description**: Testing package weight to check if it is a valid package weight. This makes sure we have the right package weights.

**Setup:** Create a unit test project.

**Test Function**: int validatePackageWeight(double num).

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| T01\_lowestweight | 1 | 1(true) | 1(true) | Pass |
| T02\_highestweight | 1500 | 1(true) | 1(true) | Pass |
| T03\_outOfRangeWeight1 | 2500 | 1(true) | 0(false) | Fail |
| T04\_outOfRangeWeight2 | -5 | 0(false) | 0(false) | Pass |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Bugs Found**:

Description of each bug found above and how to reproduce it.

# Test Description

**Test Name or ID**:

**Test Type**: Black box, White Box, integration, Acceptance, etc.

**Description**: What we are testing and why.

**Setup:** How to set up the environment to carry out the test.

**Test Function**: name of the test function associated with this description.

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Bugs Found**:

Description of each bug found above and how to reproduce it.