# Test Description

**Test Name or ID**: Integration\_JM1

**Test Type**: integration

**Description**: Testing functions from shipment.h

**Setup:** Implementation of testing code

**Test Function**: checkTruckWeightCap(), bestRoute

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Truck's weight capacity is 1500, and shipment weight is 1450 | struct Truck truck1 = { 1500 };  struct Shipment shipment1 = { 1450 }; | 1 | 1 | Pass |
| Truck's weight capacity is 300, and shipment weight is 300 | struct Truck truck2 = { 300 };  struct Shipment shipment2 = { 300 }; | 1 | 1 | Pass |
| Truck's weight capacity is not exceeded | struct Truck truck3 = { 700 };  struct Shipment shipment3 = { 200 }; | 1 | 1 | Pass |
| Truck's weight capacity is exactly 1000 | struct Truck truck4 = { 1000 };  struct Shipment shipment4 = { 0 }; | 1 | 1 | Pass |
| No routes available | struct Route\* routes1[MAX\_ROUTE] = { NULL };  struct Shipment shipment1 = { 0, 0, {3, 3} };  int size1 = 0; | -1 | -1 | Pass |
| Single route, shortest distance | struct Route\* routes2[MAX\_ROUTE] = { &route2 };  struct Shipment shipment2 = { 0, 0, {5, 5} };  int size2 = 1; | 0 | 0 | Pass |
| Single route, equal distances | struct Route\* routes3[MAX\_ROUTE] = { &route3 };  struct Shipment shipment3 = { 0, 0, {5, 5} };  int size3 = 1; | 0 | 0 | Pass |
| Multiple routes, different distances | struct Route\* routes4[MAX\_ROUTE] = { &route4\_1, &route4\_2 };  struct Shipment shipment4 = { 0, 0, {5, 5} };  int size4 = 2; | 1 | 1 | Pass |
| No routes available | struct Route\* routes5[MAX\_ROUTE] = { NULL };  struct Shipment shipment5 = { 0, 0, {3, 3} };  int size5 = 0; | -1 | -1 | Pass |

**Bugs Found**:

None

# Test Description

**Test Name or ID**: Integration\_HJ

**Test Type**: Integration

**Description**: Testing functions that can be united for use.

**Setup:** Implementing testing code

**Test Function**: checkTruckWeightCap, calCapacity

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| When there is capacity for both weight and volume + vaild shipment data | struct Truck truck1 = { 1, {1000,47}, {0,0}, 0, 0 };  struct Shipment testShipment1\_a = { {30,0.5},{'a',1} }; | 1 | 1 | Pass |
| When there is capacity for both weight and volume + invaild weight shipment data | struct Truck truck1 = { 1, {1000,47}, {0,0}, 0, 0 }; struct Shipment testShipment1\_b = { {1000,1},{'a',1} }; | 0 | 0 | Pass |
| When there is capacity for both weight and volume + invaild volume shipment data | struct Truck truck1 = { 1, {1000,47}, {0,0}, 0, 0 }; struct Shipment testShipment1\_c = { {100,2.0},{'a',1} }; | 0 | 0 | Pass |
| When truck is empty + valid input | struct Truck truck2 = { 2, {0,0}, {0,0}, 0,0 };  struct Shipment testShipment2\_a = { {30,0.5},{'a',1} }; | 1 | 1 | Pass |
| When truck is empty + invalid weight | struct Truck truck2 = { 2, {0,0}, {0,0}, 0,0 }; struct Shipment testShipment2\_b = { {3000,1},{'a',1} }; | 0 | 0 | Pass |
| When truck is empty + invalid volume | struct Truck truck2 = { 2, {0,0}, {0,0}, 0,0 }; struct Shipment testShipment2\_c = { {100,50},{'a',1} }; | 0 | 0 | Pass |
| when there is no capacity for weight + valid input | struct Truck truck3 = { 3, {1500, 2.0}, {0,0},0,0 }; struct Shipment testShipment3\_a = { {30,0.5},{'a',1} }; | 0 | 0 | Pass |
| when there is no capacity for weight + invalid input | struct Truck truck3 = { 3, {1500, 2.0}, {0,0},0,0 }; struct Shipment testShipment3\_b = { {100,50},{'a',1} }; | 0 | 0 | Pass |
| when there is no capacity for volume + valid input | struct Truck truck4 = { 4, {500, 48},{0,0},0,0 };  struct Shipment testShipment4\_a = { {30,0.5},{'a',1} }; | 0 | 0 | Pass |
| when there is no capacity for volume + invalid input | struct Truck truck4 = { 4, {500, 48},{0,0},0,0 };  struct Shipment testShipment4\_b = { {3000,1},{'a',1} }; | 0 | 0 | Pass |
|  |  |  |  |  |

**Bugs Found**:

None

# Test Description

**Test Name or ID**: Integration\_JM2

**Test Type**: Integration

**Description**: Testing functions from shipment.h and mapping.h.

**Setup:** Implementing testing code

**Test Function**: distance, addPointToRoute, getPossibleMoves

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Test the distance function | struct Point point1 = { 0, 0 };  struct Point point2 = { 3, 4 }; | 5 | 5 | Pass |
| Test the addPointToRoute function | struct Route testRoute = { {0, 0}, 0, DIVERSION };  addPointToRoute(&testRoute, 1, 2);  addPointToRoute(&testRoute, 3, 4); | 2 | 2 | Pass |
| Test the getPossibleMoves function | struct Map testMap = populateMap();  struct Point currentPoint = { 1, 1 };  struct Point backpath = { 0, 0 }; | 0 | 0 | Pass |
| Additional test cases for addPointToRoute function | addPointToRoute(&testRoute, 5, 6); | 3 | 3 | Pass |
| Additional test cases for getPossibleMoves function | struct Point currentPoint2 = { 0, 0 };  struct Point backpath2 = { 1, 1 }; | 2 | 2 | Pass |
|  |  |  |  |  |

**Bugs Found**:

None

# Test Description

**Test Name or ID**: Integration\_JM3

**Test Type**: Integration

**Description**: Testing functions from shipment.h.

**Setup:** Implementing testing code

**Test Function**: detectBuildings, validateDestination

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Route does not intersect with any buildings | struct Route route1 = { {point1, point2, point3}, 3 };  struct Map map1 = {  {  {0, 0, 0},  {0, 0, 0},  {0, 0, 0}  }  }; | 0 | 0 | Pass |
| Route intersects with a building | struct Route route2 = { {point4, point5, point6}, 3 };  struct Map map2 = {  {  {0, 0, 0},  {0, 2, 0},  {0, 0, 0}  }  }; | 1 | 1 | Pass |
| Route with Multiple Intersections | struct Route route3 = { {point7, point8, point9, point10}, 4 }; // Route with four points  struct Map map3 = {  {  {0, 0, 0},  {0, 2, 0},  {0, 0, 0}  }  }; | 1 | 1 | Pass |
| Route Completely Outside the Map | struct Route route4 = { {point11, point12}, 2 }; // Route with two points  struct Map map4 = {  {  {0, 0, 0},  {0, 0, 0},  {0, 0, 0}  }  }; | 0 | 0 | Pass |
| Route with No Intersections | struct Route route5 = { {point13, point14}, 2 }; // Route with two points  struct Map map5 = {  {  {0, 0, 0},  {0, 0, 0},  {0, 0, 0}  }  }; | 0 | 0 | Pass |
| Shipment destination is present in the route | struct Route route1 = { {point1, point2, point3}, 3 }; struct Shipment shipment1 = { 0, 0, destination1 }; | 1 | 1 | Pass |
| Shipment destination is not present in the route | struct Route route2 = { {point4, point5, point6}, 3 struct Shipment shipment2 = { 0, 0, destination2 }; | 0 | 0 | Pass |
| Empty route, destination should not be validated | struct Route route3 = { NULL, 0 };  struct Shipment shipment3 = { 1, 1, { 1, 1 } }; | 0 | 0 | Pass |
| Shipment destination matches the first point in the route | struct Route route4 = { {point7, point8, point9}, 3 };  struct Shipment shipment4 = { 1, 1, destination3 }; | 1 | 1 | Pass |
| Shipment destination matches the last point in the route | struct Route route5 = { {point10, point11, point12}, 3 }; struct Shipment shipment5 = { 1, 1, destination4 }; | 1 | 1 | Pass |
| Shipment destination matches a middle point in the route | truct Route route6 = { {point13, point14, point15}, 3 }; struct Shipment shipment6 = { 1, 1, destination5 }; | 1 | 1 | Pass |
| Shipment destination is outside the route bounds | struct Route route7 = { {point16, point17, point18}, 3 };  struct Shipment shipment7 = { 1, 1, destination6 }; | 0 | 0 | Pass |

**Bugs Found**:

None

# Test Description

**Test Name or ID**: Integration\_JM4

**Test Type**: Integration

**Description**: Testing functions from shipment.h and mapping.h.

**Setup:** Implementing testing code

**Test Function**: populateMap(), validateDestination

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Invalid Destination1 | invalidDestinationShipment1.destination.row = 30; | Integration Test 1 Passed: Invalid Destination | Integration Test 1 Passed: Invalid Destination | Pass |
| Invalid Destination2 | invalidDestinationShipment2.destination.col = 'K' | Integration Test 2 Passed: Invalid Destination | Integration Test 2 Passed: Invalid Destination | Pass |
| Invalid Destination3 | invalidDestinationShipment3.destination.row = 20;  invalidDestinationShipment3.destination.col = 'Z'; | Integration Test 3 Passed: Invalid Destination | Integration Test 3 Passed: Invalid Destination | Pass |
| Invalid Destination4 | invalidDestinationShipment4.destination.col = '1'; | Integration Test 4 Passed: Invalid Destination | Integration Test 4 Passed: Invalid Destination | Pass |

**Bugs Found**:

None