# Test Description

**Test Name or ID**: TI01

**Test Type**: Integration test

**Description**: If isTruckFull is false, loadTruck could add shipment into it. Otherwise, loadTruck should reject adding shipment.

**Setup:**

**Test Function**: loadTruck + isTruckFull

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Nominal input test | truck.curWeight = 500.0;  truck.curSpace = 18.0;  box.weight = 100.0;  box.size = 10.0; | truck.curWeight = 600.0;  truck.curSpace = 28.0 | truck.curWeight = 600.0;  truck.curSpace = 28.0 | Pass |
| Maximum input test | truck.curWeight = 0;  truck.curSpace = 0;  box.weight = 1000.0;  box.size = 36.0; | truck.curWeight = 1000.0;  truck.curSpace = 36.0 | truck.curWeight = 1000.0;  truck.curSpace = 36.0 | Pass |
| Make weight equal to full load. | truck.curWeight = 800;  truck.curSpace = 10;  box.weight = 200.0;  box.size = 10.0; | truck.curWeight = 1000  truck.curSpace = 20 | truck.curWeight = 1000  truck.curSpace = 20 | Pass |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Bugs Found**:

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

**Test Name or ID**: TI02

**Test Type**: Integration test

**Description**: Test the integration of the functions extractBuildingEdges and getClosestEdgePoint working together to find the closest edge point.

**Setup:** Using map from populateMap function that have specific building layouts for the test.

**Test Function**: extractBuildingEdges + getClosestEdgePoint

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Basic Scenario | Point dest = { 12,5 }; Point start = { 0,0 }; | 12,5 | 12,5 | Pass |
| No Building Edges | Point dest = { 18,18 };  Point start = { 5,5 }; | 5,5 | 5,5 | Pass |
| Multiple Closest Edge Points | Point dest = { 7,22 };  Point start = { 8,22 }; | 7,21 or 6,22 | 7,21 | Pass |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Bugs Found**:

None

**Test Name or ID**: TI03

**Test Type**: Integration test

**Description**: Check if read works properly validating messages and returns desired shipment with proper values

**Setup:** Using file input to feed the program numbers for the validation

**Test Function**: read

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| testing for multiple validation messages and if it returns proper shipment | 100 0.5 3x  100 2 2B  1005 0.5 3X  1000 0.5 2B | Struct Shipment package {1000, 0.5, 2B} | Struct Shipment package = {1000, 0.5, 2B} | Pass |
| read() testing for exiting program | 1005 0.5 12L  0 0 x | Struct Shipment package =  {-1, -1, 1A} | Struct Shipment package =  {0, -1, 1A} | Fail |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Bugs Found**:

Had issue with initial exit program condition, accidentally displayed truck route with as skipping condition is weight = -1 not weight = 0

**Test Name or ID**: TI04

**Test Type**: Integration test

**Description**: Checking if the shortest path algorithm works correctly as expected

**Setup:**  Feeds the base map and start and end points for the shortest path

**Test Function**: shortestPath

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Basic testing with shortestPath to see if it properly returns path | Point start(2,0)  Point dest(9,2) | testShort.numPoints = 11 | testShort.numPoints = 11 | Pass |
| Testing edge of the map location case with shortest path | Point start(2,0)  Point dest(6,24) | testShort.numPoints = 28 | Program crashes and goes out of bounds with index for array, adds too many points | Fail |
| Testing shortest path when it uses a building point as the destination point | Point start(6,21)  Point dest(7,24) | testShort.numPoints = 3 | Program crashes and goes out of bounds with index for array, adds too many points | Fail |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Bugs Found**:

It gets stuck on the corners of the map or if a point is directly on top of a building being a 1 square and ends up adding too many points to the route by repeatedly going back and fourth on the same points. Crashing the array by going out of bounds as a result.

**Test Name or ID**: TI05

**Test Type**: Integration test

**Description**: Check after shortestPath, if divergence is correct or not.

**Setup:**

**Test Function**: shortestPath + determineDivergencePath

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| No divergence for Yellow Route | struct Point start = { 0, 0 };  struct Point dest = { 9, 4 }; | truck.s\_Path.numPoints = 0; | truck.s\_Path.numPoints = 0; | Pass |
| Divergence for Yellow Route | struct Point start = { 0, 0 };  struct Point dest = { 13, 3}; | truck.s\_Path.numPoints = 2; | truck.s\_Path.numPoints = 2; | Pass |
| No divergence for Blue Route | struct Point start = { 0, 0 };  struct Point dest = { 8, 10 }; | truck.s\_Path.numPoints = 0; | truck.s\_Path.numPoints = 0; | Pass |
| Divergence for Blue Route | struct Point start = { 0, 0 };  struct Point dest = { 13, 18}; | truck.s\_Path.numPoints = 7; | truck.s\_Path.numPoints = 7; | Pass |
| No divergence for Green Route | struct Point start = { 0, 0 };  struct Point dest = { 2, 20 }; | truck.s\_Path.numPoints = 0; | truck.s\_Path.numPoints = 0; | Pass |
| Divergence for Green Route | struct Point start = { 0, 0 };  struct Point dest = { 15, 22}; | truck.s\_Path.numPoints = 7; | truck.s\_Path.numPoints = 7; | Pass |
|  |  |  |  |  |

**Bugs Found**:

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

**Test Name or ID**: TI06

**Test Type**: Integration test

**Description**: This integration test verifies the functionality of the returnTouchingSquare function.

**Setup:**

- Prepare a mock Map with specific square configurations.

- Define mock start and destination Points.

**Test Function**: returnTouchingSquare

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Destination surrounded by obstacles. | Mock Map with obstacle squares, start Point: (0, 0), destination Point: (20, 0) | No points returned as adjacent squares are obstacles. | No points returned as adjacent squares are obstacles. | PASS |
| Destination with mixed squares and closest reachable point. | Mock Map with mixed squares, start Point: (1, 2), destination Point: (15, 12) | Closest reachable Point to start is returned. (Closest Point: (1, 3)) | Closest reachable Point to start is returned. | PASS |
| Destination surrounded by empty squares on all sides. | Mock Map with empty squares, start Point: (5, 5), destination Point: (10, 10) | Closest Point to start is returned from adjacent squares. | Closest Point to start is returned from adjacent squares. | PASS |
| Destination against map boundaries. | Mock Map with boundary squares, start Point: (3, 7), destination Point: (23, 10) | No points returned as adjacent squares are outside the map. | No points returned as adjacent squares are outside the map. | PASS |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Bugs Found**:

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

|  |  |
| --- | --- |
| **Bug Description** | **Reproduction Steps** |
| Issue with incorrect handling of boundary cases. | 1. Create a map with destination point against a boundary.  2. Call the 'returnTouchingSquare' function.  3. Observe no points returned, contrary to expectations. |
| Incorrect handling of obstacles. | 1. Create a map with obstacle squares around the destination.  2. Call the 'returnTouchingSquare' function.  3. Observe points returned, contrary to expectations. |

**Test Name or ID**: TI07

**Test Type**: Integration test

**Description**: This integration test verifies the functionality of the calculateShortestPathIndex function.

**Setup:**

- Initialize a map and a truck structure

- Create an array of points representing a path between two points of the map

- Set the truck’s path to the created path

- Define a destination points

**Test Function**: calculateShortestPathIndex

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Test to see if the shortest path index could be calculated when a simple path(between two points on the map) inputed | pathPoints[]={{0,0},{1,1},{2,2}};,destination = {3,3}; | 2 | 2 | PASS |
| Test to see if the shortest path index could be calculated when dealing with a predefined green route on the map | pathPoints[]={{0,0},{1,1},{2,2}},truck.path=getGreenRoute(),destination = {0,24}; | 27 | 27 | PASS |
| Test to see if the shortest path index could be calculated when no valid path exists between the truck's path and the destination. | truck.path.numPoints = 0;  destination = {12, 12}; | 0 | 0 | PASS |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Bugs Found**:

No buys found.

**Test Name or ID**: TI08

**Test Type**: Integration test

**Description**: Check if it finds the best truck shipment by returning an index number

**Setup:** Feeding it the map, list of trucks and the shipment package

**Test Function**: findBestTruckForShipment

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Description** | **Test Data** | **Expected Result** | **Actual Result** | **Pass/Fail** |
| **Basic testing to see if it picks truck closest to destination drop-off** | truckList[]  tList[0].path = getBlueRoute();  tList[1].path = getGreenRoute();  tList[2].path = getYellowRoute();  struct Shipment package = {10, 0.5, 8Y) | 1 | 1 | Pass |
| **Test if the best truck possible is already full so pick the next best one for the route** | truckList[]  tList[0].path = getBlueRoute();  tList[1].path = getGreenRoute();  tList[2].path = getYellowRoute();  truckList[1].curWeight = 1000  struct Shipment package = {10, 0.5, 8Y) | 0 | 0 | Pass |
| Tests if none of the trucks can return the package if all of them are already full | truckList[]  tList[0].path = getBlueRoute();  tList[1].path = getGreenRoute();  tList[2].path = getYellowRoute();  truckList[0].curWeight = 1000  truckList[1].curWeight = 1000  truckList[2].curWeight = 1000  struct Shipment package = {10, 0.5, 8Y | -1 | -1 | Pass |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Bugs Found**:

None