## Functions Description

**Function Name: getDiversion**

**Parameter List:**

* **Map**
* **FromPoint**
* **ToBuilding**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Type | Description |
| Map | **Map** | Main map with buildings |
| ClosestPoint | **Point** | Closest truck route point found by getClosestPoint |
| Destination | **Point** | Valid building to deliver to |

**Returns:  
Route** of the diversion

**Description:**Creates a diversion to destination.

1. Create a Route called Diversion, set its numPoints to -1, and another called Adjacent, and another called TempRoute.
2. Set Adjacent to getPossibleMoves, with destination as argument for p1 and backpath
3. Loop through Adjacent’s points. Set TempRoute to shortestPath, with the point and closest as arguments.
4. If the Diversion’s numPoints is -1, set it to TempRoute (initializes it for comparison).
5. If the current TempRoute’s numPoints is less than the Diversion’s, set Shortest to that.
6. Return Diversion.

**Function Name: getInput(*should leave input outside & before function)***

**Parameter List:**

* **Map**
* **Weight**
* **Size**
* **Destination**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Type | Description |
| Map | **Map** | Main map with buildings |
| Weight | **&int** | Weight of shipment |
| Size | **&double** | Size of shipment |
| Destination | **&Point** | Building to deliver to |

**Returns:  
None**

**Description:**Gets, validates, and stores the user input into weight, size, and destination.

1. Create temporary variables to store the unvalidated weight (int), size (double), and unparsed destination (char[], not a point yet).
2. Loop until valid input.
3. Prompt user for input, weight size destination, in the format %d %lf %s.
4. Clear input buffer.
5. Store them into the corresponding temporary variables.
6. Validate them using validateInput.
7. If valid, store the temporary variables into the corresponding arguments. For x as the unparsed destination (exit), set the destination point’s row and col to be both X.
8. If not valid, repeat the loop.

**Function Name: validateInput**

**Parameter List:**

* **Map**
* **TempWeight**
* **TempSize**
* **TempDestination**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Type | Description |
| Map | **Map** | Main map with buildings |
| TempWeight | **int** | Weight of shipment to be validated |
| TempSize | **double** | Size of shipment to be validated |
| TempDestinationUnparsed | **char[]** | Building to deliver to to be validated |

**Returns:  
True/False**

**Description:**Validates the weight, size, and destination.

1. Create variables validWeight, validSize, validDestination, isExit, all Boolean.
2. validWeight must be 1-5000.
3. validSize must be 0.5, 2, or 5.
4. isExit set to false.
5. If destination’s first and second characters are not terminators, set validDestination to validateDestination.
6. Otherwise, if destination’s first character is x and second character is a terminator, set isExit to true if weight is 0 and size is 0.
7. Return validWeight and validSize and validDestination, or isExit.

**Function Name: validateDestination**

**Parameter List:**

* **Map**
* **TempDestinationUnparsed**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Type | Description |
| Map | **Map** | Main map with buildings |
| TempDestinationUnparsed | **char[]** | Building to deliver to to be validated |

**Returns:  
True/False**

**Description:**Validates the destination to see if it’s a building that can be delivered to.

1. Destination’s first character must be 1-26, second character must be A-Y.
2. If so, check this coordinate on the map to see if it’s a building. Otherwise return false.
3. If it’s a building, use getPossibleMoves, with the destination as argument for p1 and backpath, create a Route of the surrounding, call this AdjacentSpaces.
4. Return AdjacentSpaces’s numPoints.

**Function Name: getTruck (*break this up into smaller functions)***

**Parameter List:**

* **Map**
* **TruckArray[]**
* **Shipment**
* **Destination**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Type | Description |
| Map | **Map** | Main map with buildings |
| TruckArray[] | **Truck[]** | Array of the trucks |
| Shipment | **Shipment** | Valid shipment to be delivered |
| Destination | **Point** | Valid destination to be delivered to |

**Returns:  
Index or -1**

**Description:**Finds and returns the index of the best truck array that can take the shipment to the valid destination.

1. *First, check which truck can carry the shipment.*
2. Create a Boolean array of size 3, call this canCarry.
3. Loop through the trucks array. If truck at this index can carry the shipment, set the canCarry at this index to true, otherwise false.
4. If none of the truck can take the shipment, return 0.
5. *Second, get the shortest distance possible.*
6. Create an int variable called shortestPossible, set it to -1.
7. Loop through the trucks array. If canCarry at the current index is true:
8. Create an int called TempDistance, set this to calculateDistance.
9. If the shortestPossible is -1, set it to TempDistance (initializes it for comparison).
10. If the current TempDistance is less than shortestPossible, set shortestPossible to that.
11. *Third, find which truck(s) can reach there in the shortest distance. There can be more than 1 truck!*
12. Create a Boolean array of size 3, call this canShortest.
13. Loop through the trucks array. If canCarry at the current index is true, and if calculateDistance of the truck at this index is equal to shorestPossible, set the canShorest at this index to true, otherwise to false.
14. *Finally, for each of the truck that can deliver there in the shortest distance, select the one with the greatest space.*
15. Create an int variable called mostSpaciousIndex, set it to -1.
16. Create a double called LeastLimiting.
17. Loop through the trucks array. If canShortest at the current index is true:
18. Create an int called TempLimiting, set this to LimitingFactor.
19. If the mostSpaciousIndex is -1, set LeastLimiting to TempLimiting, and mostSpaciousIndex to the current index (initializes it for comparison).
20. If the current TempLimiting is less LeastLimiting, set LeastLimiting to TempLimiting, and mostSpaciousIndex to the current index.
21. Return mostSpaciousIndex.

***To do: calculateDistance and LimitingFactor functions***