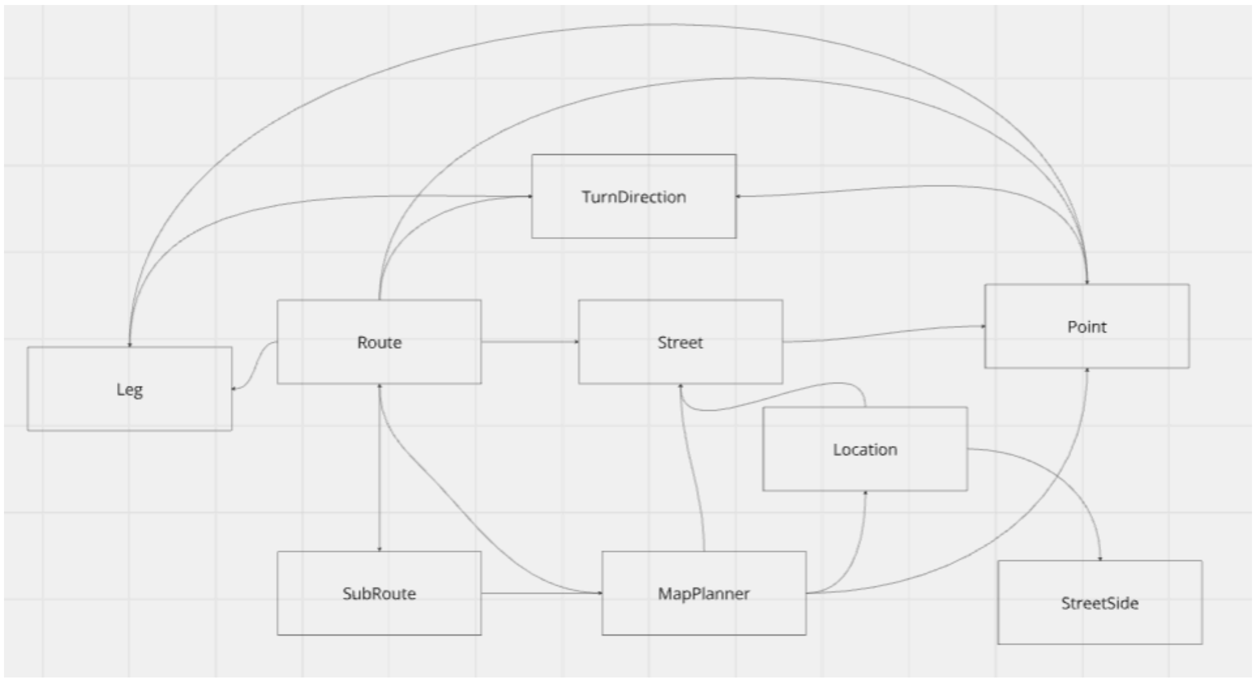
CSCI 3901 Assignment 1
Krishna Tej Nanda Kumar
B00975537
kr657835@dal.ca

Overview

The Data Routes, Streets, Points, Location, Tolerance Degree, SubRoute, Legs.
The Ask To implement a Graph given a set of Streets that will be able to find the furthest street or a Route to a destination with no Left Turns, given a Depot Location, and to detect and simplify loops in a Route.

Files and external data



Classes and Responsibilities:

1. Route Class

Responsibilities:
- Define and manage a route consisting of multiple legs (turns and streets).
- Append new legs to the route.
- Retrieve information about specific legs, such as the street turned onto and turn direction.
- Calculate the total length of the route.
- Identify loops within the route.
- Simplify the route by removing unnecessary straight segments.

Collaborators:

- MapPlanner: Provides street information and degree tolerance for turns.
- Leg: Represents individual segments or legs of the route.
- Street: Provides information about streets, such as start and end points and length.
- Point: Used to determine the start and end points of legs.
- SubRoute: Represents subsegments of the route, particularly loops.
- TurnDirection: Enum is used to specify the direction of turns.

2. SubRoute Class

Responsibilities:

- Represent a portion of a main route, defined by start and end leg numbers
- Provide access to the main route it's derived from
- Allow extraction of the subroute as a standalone route
- Adjust leg numbers for subroute representation

Collaborators:

- Route: The main route from which this subroute is derived
- MapPlanner: Provides context for extracting routes

3. MapPlanner Class

Responsibilities:

- - Create and manage a map of the city with streets and intersections
- - Set and manage the depot location
- - Add streets to the map
- - Find the furthest street from the depot
- - Compute routes without left turns
- - Manage the degree of tolerance for turn calculations
- - Maintain a graph representation of street connections
- - Provide information about streets and adjacent streets

Collaborators:

- - Location: Represents specific locations on streets
- - Street: Represents individual streets in the map
- - Point: Represents coordinates for street endpoints
- - Route: Used for route calculations and returned as a result
- - SubRoute: Used in route calculations
- - TurnDirection: Used to determine types of turns in routes
- - StreetSide: Used to specify sides of streets

4. Leg Class

Responsibilities:

- Represent a single leg or segment within a route.

- Store information about the turn direction, street ID, start point, end point, and accumulated length.

Collaborators:

- TurnDirection: Specifies the direction of the turn for this leg.
- Point: Represents the start and end points of this leg.

5. Street Class

Responsibilities:

- Represent a street in the city map with a unique ID and defined start and endpoints.
- Calculate the length of the street using its start and endpoints.
- Determine the opposite end of a street given one end.

Collaborators:

- Point: Used to define the start and end points of the street.

6. Point Class

Responsibilities:

- Represent a twodimensional point with integer coordinates.
- Calculate distances to other points.
- Determine turn types based on sequences of points.

Collaborators:

- TurnDirection: Used to determine the type of turn between three points.

7. Location Class

Responsibilities:

- Represent a specific location on a map
- Store and manage information about a location, including:
    - The street ID where the location is situated
    - The side of the street where the location is
-
- Provide access to location properties (street ID and street side)

Collaborators:

- String: Used to store the street ID
- StreetSide: Enum representing the side of the street (assumed to be an enum)
- MapPlanner: (Implied) The Location class is likely used by the MapPlanner class to represent locations on the map

8. TurnDirection Enum (Assumed)

Responsibilities:

- Define possible directions for turns (e.g., Left, Right, Straight, UTurn).

9. StreetSide Enum

Responsibilities:

- Define the possible sides of a street
- Provide a type-safe way to represent and work with street sides in the system

Collaborators:

- Location: (Implied) Likely uses StreetSide to specify which side of a street a location is on
- Route: (Potential) Might use StreetSide in more detailed routing instructions
- MapPlanner: (Potential) Could use StreetSide for more precise location specifications

Values:

- Left
- Right

Assumptions

- All streets given to addStreet are two-way streets. - - - - - -
- A street id names the part of the street between two intersections uniquely.
- A location defined on a street is at the centre of the street.
- All streets with a given (x, y) endpoint will meet at that point and form an intersection.
- A street that ends on its own would allow a drive to turn around at the end of the street (not the middle) as a UTurn.
- Streets will only cross one another at an intersection.
- If two streets meet at a T intersection then it will be presented as 3 streets meeting at one intersection and not as one long street with another one intersecting in the middle of the long street.
- The distance to the furthest street is unique. You won't be given a map with two furthest streets at exactly the same distance from the depot.

Constraints

- You cannot do a UTurn at an intersection that isn't a street dead end (end of a street with no connecting street).
- You may use any data structures from the Java Collection Framework. • If you use a graph then you may not use a library package to for your graph or for the algorithm on your graph.

- If in doubt for testing, I will be running your program on timberlea.cs.dal.ca. Correct operation of your program shouldn't rely on any packages that aren't available on that system.

Key algorithms and design elements

FurthestStreet Method

Purpose: Find the street that is furthest away from the depot by distance, allowing for left turns.

Key Algorithm: Depth-First Search (DFS) with path tracking

Design Elements:
1. Initialization:
   - Start with the depot location
   - Initialize a route from the depot
   - Use a stack for DFS traversal
   - Maintain maps for visited streets, shortest paths, and visited routes

2. DFS Traversal:
   - For each unvisited neighboring street:
     - Calculate the turn type to reach this street
     - If not a U-turn, append to the current route
     - Update visited streets and shortest paths
     - Push unvisited neighbors onto the stack

3. Backtracking:
   - If a street has been visited before, compare the new path length with the previous one
   - Backtrack if the new path is not shorter

4. Termination:
   - Continue until all possible paths are explored
   - Return the street ID with the maximum distance from the depot

RouteNoLeftTurn Method

Purpose: Compute a route from the depot to a given destination without making any left turns.

Key Algorithm: Modified Depth-First Search (DFS) with left turn restriction

Design Elements:
1. Initialization:

  - Similar to furthestStreet, but with an additional check for the destination
  - Start with the depot location
  - Initialize maps for visited streets, shortest paths, and routes

2. DFS Traversal with Left Turn Restriction:
  - For each unvisited neighboring street:
    - Calculate the turn type to reach this street
    - If the turn is not a U-turn and not a left turn, append to the current route
    - Update visited streets and shortest paths
    - Push unvisited neighbors onto the stack

3. Backtracking and Optimization:
  - If a street has been visited before, compare the new path length with the previous one
  - Backtrack if the new path is not shorter or if a left turn is encountered

4. Termination:
  - Continue until the destination is reached or all possible paths are explored
  - Return the route to the destination if found, or null if no valid route exists

Common Design Elements:
- Both methods use a graph representation of the street network
- They employ a combination of DFS and dynamic programming techniques
- Both methods maintain information about visited streets and path lengths
- The algorithms handle complex scenarios like loops and intersections

Both these algorithms use a modified dijkstra algorithm to implement the DFS which is similar for both methods.

Limitations

- The Graph is not Tested for a complicated and large number of streets which are more than 20, this does not mean that they wouldn't, more that I am unsure whether or not it will.

- Another limitation of the code is that the implementation of furthest Street and routeNoleft turn are very elaborate and complicated to understand. It took hours for me to debug one test case as it stores a lot of variables and it has to go through a lot of iterations.

- The routeNoLeft and FurthestStreet Turn will not work if we are going towards one directions which is deadend but for the street we want to find behind us, the code would fail.

Thank you.