r2.py

```python
from utilities_new import ServiceRequest, ServiceType, assign_customers_to_best_van, Van
import networkx as nx
from enum import Enum

seed = 1000
G = nx.gnp_random_graph(10, .3, seed=seed)
# print(G.nodes())
# print(G.edges())

nx.is_connected(G)

G.add_edges_from([(0, 3, {'weight': 0.1}), (0, 8,{'weight': 0.8}), (3,8,{'weight': 0.8}), (8,1, {'weight': 1.0}), (8,6, {'weight': 0.7}), (1,6, {'weight': 1.0}),  (1,4, {'weight': 0.6}),  (6,4, {'weight': 0.5}),  (6,7,{'weight': 0.9}), (4,5, {'weight': 0.5}), (4,7, {'weight': 0.4}), (4,9, {'weight': 1.0}),(7,5, {'weight': 0.8}),(7,9, {'weight': 0.4})])

#make two vans
vans = []
for i in range(1,3):
  van = Van(i)
  van.route.append(0) # Van starts at node 0
  vans.append(van)

clocktick = 0
while clocktick < 6:
  clocktick += 1

  print("---------------------------------------- START CLOCKTICK ----------------------------------")
  print(f"The current clocktick is: {clocktick}")
  for van in vans:
    print(f"Van {van.id} route history is: {van.route}\n")

  unassigned_service_requests = []

  if clocktick == 1: # adding clock tick 1 requirements
    # add {id1, p8, d9}, {id2, p3, d6} to customer requests
    unassigned_service_requests.append(ServiceRequest(1, ServiceType.Pickup, 8))
    unassigned_service_requests.append(ServiceRequest(1, ServiceType.Dropoff, 9))
    unassigned_service_requests.append(ServiceRequest(2, ServiceType.Pickup, 3))
    unassigned_service_requests.append(ServiceRequest(2, ServiceType.Dropoff, 6))

  elif clocktick == 2: # adding clock tick 2 requirements
    # add {id3, p4, d7}, {id4, p2, d4} to customer requests
    unassigned_service_requests.append(ServiceRequest(3, ServiceType.Pickup, 4))
```

```
    unassigned_service_requests.append(ServiceRequest(3, ServiceType.Dropoff, 7))
    unassigned_service_requests.append(ServiceRequest(4, ServiceType.Pickup, 2))
    unassigned_service_requests.append(ServiceRequest(4, ServiceType.Dropoff, 4))
  elif clocktick == 3: # adding clock tick 3 requirements
    # add {id5, p1, d7}, {id6, p1, d9} to customer requests
    unassigned_service_requests.append(ServiceRequest(5, ServiceType.Pickup, 1))
    unassigned_service_requests.append(ServiceRequest(5, ServiceType.Dropoff, 7))
    unassigned_service_requests.append(ServiceRequest(6, ServiceType.Pickup, 1))
    unassigned_service_requests.append(ServiceRequest(6, ServiceType.Dropoff, 9))

  # Perform any pickups or dropoffs
  for van in vans:
    van.pickup_or_dropoff()

  assign_customers_to_best_van(vans, unassigned_service_requests, G)

  # Sort van service queues
  for van in vans:
    van.sort_service_queue2(G)

  # Move vans to next nodes
  for van in vans:
    van.move_to_next_node(G)

  for van in vans:
    print(f"Van {van.id} queue is: {van.queue_as_string()}\n")

  empty_count = 0
  for van in vans:
    if len(van.queue) == 0:
      empty_count += 1

  if empty_count == len(vans):
    break

total_distance = 0
total_trips = 0
for van in vans:
  total_distance += van.distance_travelled
  total_trips += van.trips_taken

average_distance = total_distance / len(vans)

print(f"Average Distance Travelled: {average_distance}")
print(f"Total Trips Taken: {total_trips}")

```