r5.py

```
from utilities_new import ServiceRequest, ServiceType, assign_customers_to_best_van, Van
import random
import networkx as nx
import matplotlib.pyplot as plt

seed=1000           # seed the graph for reproducibility, you should be doing this
G= nx.gnp_random_graph (100, .4, seed=seed )       # here we create a random binomial graph with 10 nodes and an average (expected) connectivity of 10*.3= 3.
nx.is_connected(G)

for u, v in G.edges:      # needed for requirement R3.
  G.add_edge(u, v, weight=round(random.random(),1))

vans = []
for i in range(1,61):
  van = Van(i)
  van.route.append(0)
  vans.append(van)

customer_id = 0
clocktick = 0
while clocktick < 600: #check for time, 600 clock ticks = 600min = 10hrs = runtime for simulation
  clocktick += 1

  unassigned_service_requests = []
  #randomize 10 requests per clock tick, this makes 600 requests per hour
  #unassigned_service_requests.append(ServiceRequest(1, ServiceType.Pickup, 8))
  for i in range(0, 10):
    unassigned_service_requests.append(ServiceRequest(customer_id, ServiceType.Pickup, random.randint(0,99)))
    unassigned_service_requests.append(ServiceRequest(customer_id, ServiceType.Dropoff, random.randint(0,99)))
    customer_id += 1

  # 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
```

```python
  for van in vans:
    van.move_to_next_node(G)


while True:
  # 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)

  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}")

```