Apurba Pokharel
11627243

The overall structure

The code is divided into classes.
The classes in the code car.py are:

1. Graph

   - Generates the graph
   - Computes the Astar path length as well as determines the shortest Astar path
   - Can plot the graph if visualization is needed.
   - The graph code is referred from Dr. Russel's tutorial 1 with slight changes as necessary.

2. Car

- Has methods to handle customer pickup requests, pickup customer, dropoff customer.
- Stores the distance and trip for each car object
- The information about capacity is stored here in the class.

3. Customer

- Has just the pickup and dropoff node info stored in the class

4. Agent

- The brains of the entire operation.
- Has an array that stores all the cars and customer objects.
- An index (the actual array index) is used to refer to these cars and customers in the codebase.
  For Example: a car at index 0 in the car_array will be referred to by index 0 all over the code.
  Similarly, customer has the same rule.
- The Car object creation and customer object creation are done by the agent as specified by us in the main function.
- The request for picking up new customers, and selecting a car based on the shortest distance as well as the current capacity is handled here by the agent.
- The process of updating the wait queue based on the distance to the nearest customer is done here as well.
- Picking up and dropping off the customer is done by the agent.

Simple overall pseudocode

1. Generate the cars and graph as needed.

2. In the time simulation (the for loop), add another for loop to handle the number of requests in a minute. For example: If for 200 ticks, a request of 5 customers per tick is needed then.

```
for i in range(200):
    for j in range(5):
        processNewCustomerRequest()
        //Move ahead with the assignment and pickup dropoff as necessary after this
        moveAllCars()
        //This method will contain all the logic with car assignment, pickup, dropoff, and selecting the car's next node/destination.
```

3. At the end check if all customers who have requested before the cutoff time, have been served. If not run an additional tick simulation to complete them.

4. Output the average distances and average trip as necessary.

Customer Generation Pseudocode

1. A random pick_up point is generated using the random package.
2. Another random drop_off point is generated using the same package and is accepted as long as pick_up != drop_off

The scheduling Pseudocode

When a new customer is generated

The key factor to consider is, if cars have the same distance then allocate the first car that is empty, if all equidistant cars are non-empty then allocate the first car that has the smallest car index.
If there is no equidistant cars then allocate the car that has the smallest distance to the customer's pickup point.

1. Loop over all the cars.
2. Run a simple sorting algorithm to get the car that has the smallest distance between its current node and the customer's pick-up node.
3. Consider the capacity of the car here
4. I will use AStar to compute these distances.

```
smallest_distance = 10000000000
car_index = -1
eq_distant_array = []
for i in range(car_array_size):
    if(car_array[i].isFull())
        print("Car ", i, "is full")
        continue
    distance = computeAStarPathLength(customer_pick_up_node, car_current_node)
```

```
        if distance < smallest_distance:
            smallest_distance = distance
            eq_distant_array.clear()
            car_index = i


        if distance == smallest_distance:
            eq_distant_array.append(i)

    if len(eq_distant_array) != 0:
        first_non_empty_car_index = self.getFirstEmptyCar(eq_distant_array)
        if first_non_empty_car_index != -1:
            return first_non_empty_car_index
        else:
            return eq_distant_array[0]
    else:
        return car_index
```

5. Once the customer has been assigned to the car, update the car's customers_in_wait_queue array.
6. Since Dr. Russel wants us to update the customers based on their distance constantly as per his tutorial 2(explained in Clock Tick 3). Another sorting, bubble sorting will need to be run to always have customers arranged in the wait_queue based on the shortest distance concerning the car's node position.

```
    for i in range(customers_in_wait_queue_length):
        for j in range(i, len(customers_in_wait_queue)):

            distance_i = computeAStarPathLength(car_current_node,
    customer_index_i_pick_up_distance)

            distance_j = computeAStarPathLength(car_current_node,
    customer_index_j_pick_up_distance)

            if distance_j < distance_i:
                temp = customers_in_wait_queue[j]
                customers_in_wait_queue[j] = customers_in_wait_queue[i]
                customers_in_wait_queue[i] = temp

    update the wait queue with the new sorted queue
```

7. Based on this queue car will always serve the customer at the beginning of this array (behaves as a queue).
8. The car will either travel to pick up this customer at index 0 or will move towards dropping it off.

9. Once dropped of the new customer at the beginning of this queue will be served.

Important factors for the pickup and dropoff algorithm.

1. As per Dr. Russel's tutorial 2 (clock tick 3), if two or more customers share the same pickup points then they need to be picked up together as long as the space is available.

2. The same goes for dropoff, if the currently being served customer shares dropoff with two or more customers (that are already picked up) then they need to be dropped off together.