CSC110 Lecture 31: Discrete-Event Simulators

Hisbaan Noorani

December 02, 2020

Contents

1 Exercise 1: Completing `FoodDeliverySimulation`

Your task for this exercise is to review the code we covered in lecture for the `FoodDeliverySimulation` class, and then complete the two helper methods for the initializer. Again, you can complete your work here or in the starter file `simulation.py` we've posted on Quercus.

```
class FoodDeliverySimulation:
    """A simulation of the food delivery system.

    >>> simulation = FoodDeliverySimulation(datetime.datetime(2020, 11, 30), 7, 4, 100, 50)
    >>> simulation.run()
    """
    # Private Instance Attributes:
    #   - _system: The FoodDeliverySystem instance that this simulation uses.
    #   - _events: A collection of the events to process during the simulation.
    _system: FoodDeliverySystem
    _events: EventQueue

    def __init__(self, start_time: datetime.datetime, num_days: int,
                 num_couriers: int, num_customers: int,
                 num_restaurants: int) -> None:
        """Initialize a new simulation with the given simulation parameters.

        start_time: the starting time of the simulation
        num_days: the number of days that the simulation runs
        num_couriers: the number of couriers in the system
        num_customers: the number of customers in the system
        num_restaurants: the number of restaurants in the system
        """
        self._events = EventQueueList()
        self._system = FoodDeliverySystem()

        self._populate_initial_events(start_time, num_days)
        self._generate_system(num_couriers, num_customers, num_restaurants)

    def _populate_initial_events(self, start_time: datetime.datetime, num_days: int) -> None:
```

```
        """Populate this simulation's Event priority queue with GenerateOrdersEvents.

        One new GenerateOrdersEvent is generated per day for num_days,
        starting with start_time.
        Each GenerateOrdersEvent's duration is 24 hours.
        """
        # TODO: complete this method

    def _generate_system(self, num_couriers: int, num_customers: int, num_restaurants: int) -> None:
        """Populate this simulation's FoodDeliverySystem with the specified number of entities.

        You can initialize restaurants with empty menus.
        """
        for i in range(0, num_customers):
            location = _generate_location()
            customer = Customer(f'Customer {i}', location)
            self._system.add_customer(customer)

        # TODO: complete this method

    def run(self) -> None:
        """Run this simulation.
        """
        while not self._events.is_empty():
            event = self._events.dequeue()

            new_events = event.handle_event(self._system)
            for new_event in new_events:
                self._events.enqueue(new_event)
```

2 Exercise 2: Reporting statistics

As we discussed in lecture, now that we have a full `FoodDeliverySimulation` class, we can write methods to report statistics on a simulation that has already been run.

Your task is to implement the method `FoodDeliverySimulation.restaurant_order_stats`, which returns the maximum, minimum, and average number of completed orders for a single restaurant during the run of the simulation.

```
class FoodDeliverySimulation:
    ...

    def restaurant_order_stats(self) -> Dict[str, float]:
        """Return summary statistics for how many orders each restaurant received.

        The returned dictionary contains three keys:
            - 'max': the maximum number of orders made to a single restaurant
            - 'min': the minimum number of orders made to a single restaurant (can be 0)
            - 'average': the average number of orders made to a single restaurant

        Preconditions:
            - self.run() has already been called
        """
```

```
        # As we discussed yesterday, we can add a new method FoodDeliverySystem.get_restaurants()
        # instead of accessing a private attribute _restaurants.
        orders_per_restaurant = {name: 0 for name in self._system._restaurants}

        for order in self._system._orders:
            orders_per_restaurant[order.restaurant.name] += 1

        keys = list(orders_per_restaurant.keys())
        order_nums = [orders_per_restaurant[key] for key in keys]
        max_num = float(max(order_nums))
        min_num = float(min(order_nums))
        average = float(sum(order_nums) / len(order_nums))

        return {'max': max_num, 'min': min_num, 'average': average}
```