Design and implement an OO next event scheduling simulation environment in C#. The environment must be able to simulate the model below, and it must be flexible enough to simulate other models based on entities, events, queues and servers. I will use the term model and simulation to refer to the system being simulated.

**Entities.** Entities move through the simulation. They are created by the loader server and deleted when they are shipped or recycled. There can be many entities in a model at any time. Entities are temporary objects that are generated, processed, and destroyed during the simulation. Each has a unique id and creation time. Entities have an updateDestroy method that updates any statistics associated with the entity (for example, min, average, and max total time, and queue waiting time in the model). Report separate statistics for shipped and recycled entities.

**Servers.** A simulation can have many servers. Servers are persistent active model components that provide a "service" to entities. Servers sample a value (from a random – uniformly distributed min to max) to represent the service time for each entity creation or service. Servers have a percent busy and percent blocked statistic. Each service can have a collection of input queues from which it takes entities. During service a server may have to decide which output queue to place the entity and the end of service. After providing its service the server either blocks if its output queue collection is full, sends the entity to its output queue, or has the entity updateDestroy itself (Shipper and Recycler – update statistics delete self). A server's collections of queues are ordered to represent a queuing policy – such that it will examine and use the first available queue in the collection. Servers become idle when there are no entities to service. Servers become blocked when there is no place to send the entity they have serviced. Blocked servers hold an entity until there is a place for them to place it.

**Queues.** Queues are a FIFO collection that holds entities waiting for service from a busy server. Queues have a capacity (number of entities they can hold) and have a minimum, average, and maximum length statistics. Queues are passive components in the simulation model.

**Events.** There can be many dynamic events in a simulation. Events are temporary passive objects that are created by the simulation's schedule method. Schedule creates an event that
represents an activity that should occur in some time value in the future (now + futureOffset). Events have a reference to an active simulation object (server or simulation), a scheduled time (eventTime) to occur, and one or more string arguments (e.g., a trace/display string). The arguments to an event should be general to any system that is being simulated. Scheduled events are placed in the simulation's EventList. For example, a service that creates entities (e.g., Loader) activity method could do something like the following:

```java
    e = new Entity()
    if (OutputQueue.hasSpace()) // OutputQueue is a property
        OutputQueue.arrive(e)
        schedule(this, now + random(10, 30), "s1")
    else
        block()
```

Statistics. There are two types of statistics: TimeWeightedStatistic and TimeIndependentStatistic. These are passive objects. TimeWeightedStatistics accumulate statistical data that is time sensitive. The average length of a queue is a good example. TimeIndependentStatistics accumulate statistical data that is not affected by time; the average time an entityType has been in the system is a good example. Statistics can be reinitialized (reset) to remove possible biases (start up conditions).

Simulation. The simulation creates the persistent components of the simulation model, runs the simulation, and reports the statistics. A simulation model has an attribute named "now" that is the current time in the simulation; an EventList that is a collection of events ordered on their scheduled event time; a current count of the entities created during simulation that is used to provide a unique id for each entity generated; and a method to uniformly sample a number between a minimum and maximum value. A Simulation has one or more servers and statistics that it creates. After creating the components of the simulation model and scheduling initial events, the simulation invokes its "simulate" method. The simulate method removes the first event from the model's EventList, sets now to the event's eventTime, and invokes the event's simulation object's activity method. Every active simulation component has an activity method. The Simulation's scheduled event is either to reset statistics or to stop the simulation. Its activity method determines whether to reset or stop based on the event's argument (the string "Stop" or "Reset"). If "Reset" the method invokes the resetStatistics method to re-initialize all statistics. If "Stop" it evokes the reportStatistics method which calculates final statistics, and reports simulation statistics.

For each server report percent busy and blocked statistics. For each queue report minimum, average, and maximum queue length statistics. Report the minimum, average, and maximum elapsed time to service entities.

The simulation should begin by creating the following events (where now == 0.0):

```java
    schedule (loader, now + random(aLoader.min, aLoader.max), "load")
    schedule (this, now + 100, "Reset")
    schedule (this, now + 400, "Stop")
```

Where loader is a Loader object and this is the current SimulationModel object.
**Activity() and schedule(...)**. A server’s activity method simulates the “end of service”. For example when loader’s activity is called at time $(\text{now} + \text{random}(a\text{Loader}.\text{min}, a\text{Loader}.\text{max}))$, it is the time when loading of an entity is done. The schedule method represents the beginning of a service (“transaction” or “activity”) and the activity() method call represents the end of that service.

Simulate the model drawn above with the following parameters:

<table>
<thead>
<tr>
<th>Component</th>
<th>Activity Description</th>
<th>minimum</th>
<th>maximum</th>
<th>percent</th>
<th>output Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loader</td>
<td>make entity</td>
<td>5</td>
<td>15</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Builder</td>
<td>service entity</td>
<td>10</td>
<td>30</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Tester</td>
<td>service entities from FixedQ before BuiltQ. Determine if entity is to be fixed, shipped, or recycled.</td>
<td>20</td>
<td>40</td>
<td>100</td>
<td>20 FixQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>70 ShipQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10 RecycleQ</td>
</tr>
<tr>
<td>Fixer</td>
<td>service entity</td>
<td>80</td>
<td>120</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Recycler, Shipper</td>
<td>service entity, updateDestroy</td>
<td>40</td>
<td>60</td>
<td></td>
<td>no outputQ</td>
</tr>
<tr>
<td></td>
<td>update entity statistics, destroy entity</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BuildQ, TestQ, ShipQ</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>FixQ, RepairedQ, RecycledQ</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

I strongly suggest that you design your simulation environment with trace capabilities so that you can debug the model. You may want to start with a simpler model, for example: server, queue, and server. You might consider how you could run the simulation and step through processing the EventList.

The figure above represents the model as graphical nodes. You can use the Simulation “starter kit” as the starting point for your project. Your program must run with a graphical representation of the simulation and generate a printed statistics output.

Given a SimulationModel class you should be able to simulate any model where servers can have 0 or more input queues and 0 or more output queues without having to edit/recompile the SimulationModel class.

I reserve the right to change the parameters of the simulation model.
Submission. Each group should submit their solution on removable storage media (floppy disc, CD, USB drive, etc.). The media should be labeled with the names of the group members. Please submit the entire project directory – source files and executable files. In addition there should be a printed UML lite class diagram of the solution. The names and email addresses of every group member must be in displayed in the about display as well as usage information, in comments in the source of each class, on the submitted media and printed on the UML lite class diagram. There should also be a printed statement about how to run the solution. You can included any additional documentation that you think will help me grade your submission (more documentation does not imply a higher grade). Do not print the source file of the classes for submission. Submit media and printed documentation in an 8.5 by 11” folder. I will provide folders.