Simulation
An Application of Queue ADT

Revised based on textbook author's notes.

Computer Simulations

- Computers can be used to model and simulate real-world systems and phenomena.
  - Computer applications.
  - Designed to represent and react to significant events in the system.
- Examples:
  - Weather forecasting
  - Flight simulators
  - Business activities

Airline Ticket Counter

- How many ticket agents are needed at certain times of the day in order to provide timely service?
  - Too many agents will cost the airline money.
  - Too few will result in angry customers.

- A computer simulation can be developed to model this real system.

Queuing System

- A system where customers must stand in line awaiting service.
  - A queue structure is used to model the system.
  - Simple systems only require a single queue.
  - The goal is to study certain behaviors or outcomes.
    - average wait time
    - average queue length
    - average service time

Discrete Event Simulation

- Consists of a sequence of significant events that cause a change in the system.
  - Time driven and performed over a preset time period.
  - Passing of time is represented by a loop, one iteration per clock tick.
  - Events can only occur at discrete time intervals.
  - Time units must be small enough to accommodate the events.

Structure of a simulation program

for each time step in range of total time:
    processing event type one
    processing event type two
    …

Sample Events

- Some sample events include:
  - Customer arrival
  - Start or end of a transaction (service)
  - Customer departure

System Parameters

- A simulation is commonly designed to allow user supplied parameters to define conditions:
  - Length of the simulation (begins at time 0).
  - Number of servers.
  - Expected time to complete a transaction.
  - Distribution of arrival times.

- By adjusting these, the conditions can be changed under which the simulation is performed.

Event Rules

- A set of rules are defined for handling the events during each tick of the clock.
- The specific rules depend on what is being studied.

Sample Event Rules

- To determine the average wait time:
  - If a customer arrives, he is added to the queue.
    - at most one customer can arrive per time step.
  - If there are free servers and customers waiting, the next customer in line begins her transaction.
    - we begin a transaction for each free server.
  - If a transaction ends, the customer departs and the server becomes free.
    - multiple transactions can complete in one time step.

Random Events

- To correctly model a queuing system, some events must occur at random. (i.e. customer arrival)
- We need to model this action as close as possible.
  - Specify the odds of a customer arriving at each time step as the average time between arrivals.
  - Use a random number generator to produce a value.
  - Compare the value to the average arrival time.

Sample Simulation

- Analyze the average time passengers have to wait for service at an airport ticket counter.
  - Multiple ticket agents.
  - Multiple customers that must wait in a single line.

System Inputs

- The program will prompt the user for the queuing system parameters.

```
Number of minutes to simulate: 25
Number of ticket agents: 2
Average service time per passenger: 3
Average time between passenger arrival: 2
```

- For simplicity, we use minutes as the discrete time units.

System Outputs

- After performing the simulation, the program will produce the following output:

```
Number of passengers served =  12
Number of passengers remaining in line = 1
The average wait time was 1.17 minutes.
```

Debug Info

- We also display event information that can help verify the validity of the program.

```
Time   2: Passenger 1 arrived.
Time   2: Agent 1 started serving passenger 1.
Time   3: Passenger 2 arrived.
Time   3: Agent 2 started serving passenger 2.
Time   5: Passenger 3 arrived.
Time   5: Agent 1 stopped serving passenger 1.
Time   6: Agent 1 started serving passenger 3.
Time   6: Agent 2 stopped serving passenger 2.
Time   8: Passenger 4 arrived.
Time   8: Agent 2 started serving passenger 4.
Time   9: Agent 1 stopped serving passenger 3.
Time  10: Passenger 5 arrived.
Time  10: Agent 1 started serving passenger 5.
Time  11: Passenger 6 arrived.
Time  11: Agent 2 stopped serving passenger 4.
Time  12: Agent 2 started serving passenger 6.
Time  13: Passenger 7 arrived.
```

Class Organization

- Our design will be an object-oriented solution with multiple classes.
  - Passenger – store info related to a passenger.
  - TicketAgent – store info related to an agent.
  - TicketCounterSimulation – manages the actual simulation.

Passenger Class

simpeople.py

```
class Passenger :
   # Creates a passenger object.
  def __init__( self, id_num, arrival_time ):
    self._id_num = id_num
    self._arrival_time = arrival_time

   # Gets the passenger's id number.
  def id_num( self ) :
    return self._id_num

   # Gets the passenger's arrival time.
  def time_arrived( self ) :
    return self._arrival_time
```

TicketAgent Class

simpeople.py

```
class TicketAgent :
  def __init__( self, id_num ):
    self._id_num = id_num
    self._passenger = None
    self._stop_time = -1

  def id_num( self ):
    return self._id_num

  def is_free( self ):
    return self._passenger is None

  def is_finished( self, cur_time ):
    return self._passenger is not None and self._stop_time == cur_time

  def start_service( self, passenger, stop_time ):
    self._passenger = passenger
    self._stop_time = stop_time

  def stop_service( self ):
    the_passenger = self._passenger
    self._passenger = None
    return the_passenger
```

The Simulation Class

simulation.py

```
from array import Array
from llistqueue import Queue
from simpeople import TicketAgent, Passenger

class TicketCounterSimulation :
  def __init__( self, num_agents, num_minutes,
                          between_time, service_time ):
     # Parameters supplied by the user.
    self._arrive_prob = 1.0 / between_time
    self._service_time = service_time
    self._num_minutes = num_minutes

     # Simulation components.
    self._passenger_q = Queue()
    self._the_agents = Array( num_agents )
    for i in range( num_agents ) :
      self._the_agents[i] = TicketAgent(i+1)

     # Computed during the simulation.
    self._total_waitTime = 0
    self._num_passengers = 0

# ...
```

The Simulation Class

simulation.py

```
class TicketCounterSimulation :
# ...
   # Run the simulation using the parameters supplied earlier.
  def run( self ):
    for cur_time in range(self._num_minutes + 1) :
      self._handle_arrival( cur_time )
      self._handle_begin_service( cur_time )
      self._handle_end_service( cur_time )

   # Print the simulation results.
  def print_results( self ):
    num_served = self._num_passengers - len(self._passengerq)
    avg_wait = float( self._total_waitTime ) / num_served
    print( "" )
    print( "Number of passengers served = ", num_served )
    print( "Number of passengers remaining in line = %d" %
           len(self._passenger_q) )
    print( "The average wait time was %4.2f minutes." % avg_wait )

  # The remaining methods that have yet to be implemented.
  # def _handle_arrive( cur_time ):        # Handles simulation rule #1.
  # def _handle_begin_service( cur_time ):  # Handles simulation rule #2.
  # def _handle_end_service( cur_time ):    # Handles simulation rule #3.
```

The Simulation Class

simulation.py

```
class TicketCounterSimulation :
# ...
  # The remaining methods that have yet to be implemented.
  # def _handle_arrive( cur_time ):        # Handles simulation rule #1.

  def _handle_arrival( self, cur_time ):
    p = random.random()
    if p < self._arrive_prob:    # a passenger should arrive
      passenger = Passenger( self._num_passengers, cur_time )
      self._passenger_q.enqueue( passenger )
      print( 'Time ', cur_time, ': Passenger ', \
               self._num_passengers, ' arrived.' )
     self._num_passengers += 1

  # def _handle_begin_service( cur_time ):  # Handles simulation rule #2.
  # def _handle_end_service( cur_time ):    # Handles simulation rule #3.
```

The Simulation Class

simulation.py

```
class TicketCounterSimulation :
# ...
  # The remaining methods that have yet to be implemented.
  # def _handle_begin_service( cur_time ):  # Handles simulation rule #2.

  def _handle_begin_service( self, cur_time ):
    if self._passenger_q.is_empty() == False:    # handle a customer
      agent_ID = self._find_free_agent()
      if agent_ID >= 0:    # found a free one
        this_passenger = self._passenger_q.dequeue()
        stop_time = cur_time + self._service_time
        self._the_agents[agent_ID].start_service(this_passenger, stop_time)
        self._total_wait_time += cur_time - this_passenger._arrival_time
        print( 'Time ', cur_time, ': Agent ', agent_ID, \
          ' started serving passenger ', this_passenger.id_num(), '.' )
# def _handle_end_service( cur_time ):    # Handles simulation rule #3.
```

The Simulation Class

simulation.py

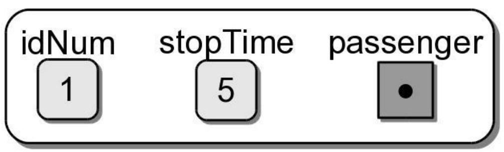
```
class TicketCounterSimulation :
# ...
  # The remaining methods that have yet to be implemented.
  # def _handle_end_service( cur_time ):    # Handles simulation rule #3.

  def _handle_end_service( self, cur_time ):
     agent_ID = self._find_finish_agent( cur_time )
     if agent_ID >= 0:    # found one who should complete the service
       this_passenger = self._the_agents[ agent_ID ].stop_service()
       print( 'Time ', cur_time, ': Agent ', agent_ID, \
         ' stopped serving passenger ', this_passenger.id_num(), '.' )
```

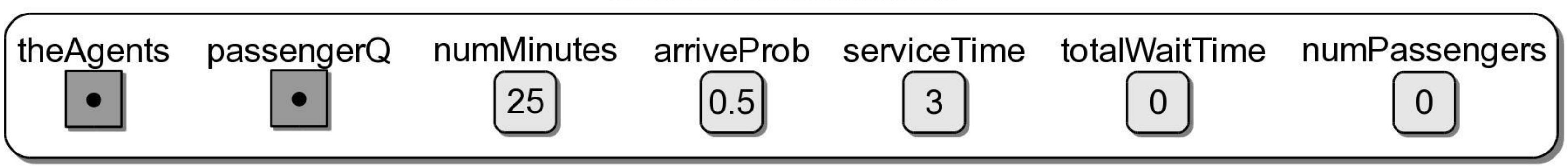
Simulation Objects

- Sample instances of each class.

Passenger



TicketAgent

TicketCounterSimulation


Sample Results

Num Minute s	Num Agents	Avg Service	Time Between	Avg Wait	Passengers Served	Passengers Remaining
100	2	3	2	2.49	49	2
500	2	3	2	3.91	240	0
1000	2	3	2	10.93	490	14
5000	2	3	2	15.75	2459	6
10000	2	3	2	21.17	4930	18
100	2	4	2	10.60	40	11
500	2	4	2	49.99	200	40
1000	2	4	2	95.72	400	104
5000	2	4	2	475.91	2000	465
10000	2	4	2	949.61	4000	948
100	3	4	2	0.51	51	0
500	3	4	2	0.50	240	0
1000	3	4	2	1.06	501	3
5000	3	4	2	1.14	2465	0
10000	3	4	2	1.21	4948	0